S32G3 Bootloader + LLCE Logger Demo

by John Li (nxa08200)

本文说明在S32G3 RDB3板上如何定制开发Bootloader，主要实现功能是：

- Bootloader启动一个M核，MCAL驱动测试程序：本文测试了LLCE Logger驱动示例代码。
- Bootloader同时启动A53 Linux，实现LLCE CAN Logger。

本文为G2 Bootloader应用文档的姐妹篇，关注在G3+更新的软件+不同的测试Demo。

历史	说明	作者
V1	● 创建本文	John.Li

目录

1 需要的软件与工具

1.1 软件工具与文档

软件/工具/文档	名称	说明
开发板	S32G3 RDB3	S32G3 开发板
配置与烧录工具	S32 Design Studio 3.5 with the update 3	从www.nxp.com，个人帐号中下载
EB TresosStudio 27.1	EB TresosStudio 27.1	从www.nxp.com，个人帐号中下载评估版
Bootloader 工程：	S32G2 Platform Software Integration 2023.02	从www.nxp.com，个人帐号中下载 内含 bootloader 工程
AutoSar MCAL 基础软件	RTD-MCAL 4.0.2: SW32G_RTD_4.4_4.0.2_D2203.exe	从www.nxp.com，个人帐号中下载
LLCE MCAL	S32G_LLCE_GATEWAY_SRC_1.0.7_D2303.exe	收费软件： S32G_LLCE_1_0_7_FDK 中的安装包才含有 llce logger demo，不需要 logger demo 的可以采用其它 demo
Linux BSP	BSP 38	从www.nxp.com，个人帐号中下载
App Note	S32G_Bootloader_V*.pdf	https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ ta-p/1519838 G2 的 Bootloader 文档，本文为 G3+更新软件版本+不同 MCAL demo 版本。
App Note	S32G_Supplemental_documentation _on_resolving_clock_conflicts	https://community.nxp.com/t5/ NXP-Designs-Knowledge-Base/ S32G-Bootloader-Customzition/ ta-p/1519838 解决 G3 时钟冲突的文档，本文包含了其部分内容
App Note	S32G LLCE CAN Linux Driver Test	https://community.nxp.com/t5/

		NXP-Designs-Knowledge-Base/ S32G-LLCE-CAN-Linux-Driver-Test-Chinese-Version/ ta-p/1712830 LLCE CAN Linux 独立编译说明

1.2 开发说明

本文是文档《S32G_Bootloader_V*.pdf》的升级篇，相关说明参考前文，不同之外在于：

- 本文以 G3 RDB3 为参考平台。
- 本文使用 Bootloader 2023_0+LLCE_1.0.7_FDK+BSP38 为软件测试平台，软件版本更新。
- 本文使用 LLCE M Core to A Core Demo 做为示例。

注意点：

- 《S32G_Bootloader_V*.pdf》的 G2 平台 A Core 工作在 1.0G，而 G3 平台的 Linux 是默认工作在 1.3G 的，所以与 M Core 400Mhz 同 Core PLL 时，需要将频率修改为 1.2G，而 G2 不用修改，所以 G3 平台需要修改 ATF 代码。
- Bootloader 升级代码变化比较大。
- LLCE Logger 相关对 Linux 代码有所要求。
- 其它未尽事宜，如 Debug 方法，参考文档《S32G_Bootloader_V*.pdf》和源代码。

2 测试软件安装编译说明

2.1 安装 LLCE Logger 驱动

注意，本文是使用 LLCE Logger Demo 做为 M7 核的 MCAL 示例的，而 Logger 功能是需要 LLCE FDK 支持的，是付费软件，所以没有采购 FDK 的客户可以使用其它示例，比如说：《S32G_Bootloader_V*.pdf》中的 UART MCAL 示例，或者其它非 AF 的 LLCE Demo。

确认 RTD：C:\NXP\SW32G_RTD_4.4_4.0.2 已经安装，并且：C:\EB\tresos\links 目录下有：SW32G_RTD_4.4_4.0.2.link：

```
path=C:/NXP/SW32G_RTD_4.4_4.0.2
```

双击 S32G_LLCE_GATEWAY_SRC_1.0.7_D2303.exe 安装。安装向导中会要求提供 EB Tresos Studio 的路径(C:\EB\tresos\)。

安装后的目录在：C:\NXP\S32G_LLCE_1_0_7_FDK，最好确认一下 C:\EB\tresos\links 目录下只有一份 LLCE link 文件：S32G_LLCE_1_0_7_FDK.link。

然后将 LLCE 中本来有的组件 C:\NXP\S32G_LLCE_1_0_7_FDK\eclipse\plugins* 拷贝到 C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins 中

2.2 编译 LLCE 驱动测试程序(以 CAN Logger 为例)

打开CAN Logger工程：

EB tresos Studio 27.1->File->Import…->General->Existing Projects into Workspace:->Next

Select root directory->Browse…-> C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\sample_app_can_llce\tresos_s32g3\Tresos_Multihost_Project，(注意：Host1实际使用Multihost工程，所以如果需要修改Host1配置，则要使用此工程)。

使用EB生成代码后，代码位于 C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\sample_app_can_llce\tresos_s32g3\Tresos_Multihost_Project\Multihost_Cfg\output\generated。

而参考C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\Makefile

```
# Configuration source files and application for multihost app (host1 app)
SRC_CAN_LLCE_HOST1_LOGGING = ./sample_app_can_llce_af/can_mcal_logging/src/main.c     \
                             $(wildcard $(MULTIHOST_GENERATE_DIR)/src/*.c)
SRC_app_host1_logging := $(SRC_NVIC) $(SRC_EXCEPT) $(SRC_PLATFINIT) $(SRC_CAN_COMMON)
$(SRC_LLCEBIN) $(SRC_FW_LOADING) $(SRC_CAN_LLCE_HOST1_LOGGING)
$(SRC_CAN_LLCE_PLUGINS) $(SRC_HEARTBEAT)
COMPILE_app_host1_logging := $(COMPILE_can_multihost)
```

所以Logger的Host1 app会编译multihost下generate的代码。

修改：C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\config.mk

```
TOOLCHAIN ?= gcc
TARGET_DEVICE ?= S32G3
# GHS_location = C:/ghs/comp_202014
GCC_location = C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2/gcc-9.2-arm32-eabi/bin
LLCE_BIN_LOCATION = ../../firmware/llce_bin/$(TARGET_DEVICE)/bin
LLCE_INTERFACE_LOCATION = ../../firmware/llce_interface
PLUGINS_DIR_LLCE = ../../eclipse/plugins
PLUGINS_DIR_RTD = C:/NXP/SW32G_RTD_4.4_4.0.2/eclipse/plugins
HSE_FW_S32G2_DIR = C:/NXP/HSE_FW_S32G2XX_0_2_22_0
HSE_FW_S32G3_DIR = C:/NXP/HSE_FW_S32G3XX_0_2_22_0
HSE_FW_DIR = $(HSE_FW_$(TARGET_DEVICE)_DIR)
AR_PKG_RTD_NAME = TS_T40D11M40I2R0
AR_PKG_LLCE_NAME = TS_T40D11M10I7R0
```

然后在C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\Makefile中，如下修改导出bin文件：

```
app_host1_logging: $(OBJS) $(SRC_STARTUP)
        @$(COMPILE) $(SRC_STARTUP_ASM) $^ -o $(TMP_location)/$@.elf -T $(LF_ASR_HOST1)
        $(GCC_location)/arm-none-eabi-objcopy.exe -O binary ./$(TMP_location)/$@.elf ./$(TMP_location)/$@.bin
```

然后因为我们只需要app_host1_logging程序，所以把host2去掉，(要不然.map文件中是host2的内容)。

```
#app_host2_logging: $(OBJS) $(SRC_STARTUP)
#       @$(COMPILE) $(SRC_STARTUP_ASM) $^ -o $(TMP_location)/$@.elf -T #$(LF_ASR_HOST2)
can_logging:
```

```
	$(MAKE) $(TMP_location)
	$(MAKE) app_host1_logging
	# $(MAKE) app_host2_logging
```

Cygwin中编译LLCE Logger驱动示例程序的命令如下：

C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af

$ make -j8 can_logging.

生成镜像在

C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\tmp

app_host1_logging.bin

app_host1_logging.elf

.map

2.3 Logger Demo 功能说明

参考文档：C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\Readme.docx 中 MULITHOST Demo，Logger Demo 只用到了：

Using Wired Loopback inside the same Host(First Part)模式。

参考文档：

C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\sample_app_can_llce_af\README_AF.txt

LOGGING demo: this demo does a loopback on host1 and logging and storing frame on host2. Host1 is started first and does the loading of firmware and also the initialization of Llce through Can_Init routine. Host2 can start after that (!!!after Can_Init of first host !!!), configures and registers its own interrupt for being notified whenever the logging fifo is not empty (FNEMPTY).Host1 will do loopback and after that you can stop host2 and see the structure which stores the logged frames and the logged frames counter. The logger app (running on host2) does not use Can MCAL driver. It can be built using: make can_logging

我们只使用到了 Host1, Host2 修改为 Linux Logger 驱动。

参考 EB 配置：

Sample_Multihost_Cfg_S32G3->test_llce(…)->CAN_43_LLCE(…)->Can_43_LLCE->CanHardwareObject:

CanHO_Config1_RX0_log：

- Can Hw Filter Code (0 -> 4294967295) =41
- Can LLCE Advanced Feature Reference= /Llce_Af/Llce_Af/LlceAfGeneral/CanAdvancedFeature_0

而 CanAdvancedFeature_0 为：

- Enable Logging feature on LLCE=checked
- Enable Host Receive feature on LLCE=checked

所以这是一个 Logger 接收口。

CanHO_Config1_TX1：一般的一个发送口。

同理可以分析 CanHO_Config1_RX2_log 和 CanHO_Config1_TX3。

参考源代码：
C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\sample_app_can_llce_af\can_mcal_logging\src\logger.c

```
/* Sending frames from CONTROLLER1/3 to CONTROLLER0/2. They have wired connection on the EVB = LoopBack. */
  for (u16MbGlobalIndex = 0; u16MbGlobalIndex < ITER_NR; u16MbGlobalIndex++)
  {
    /* Ctrl 1 sends a message with ID=41 to ctrl 0, which has a filter (CanHO_Config1_RX0_log) which accepts it and has also an advanced
    feature reference to CanAdvancedFeature_0 in Llce_Af plugin which says "send this frame to the main host, but also to a second host (the logger app)" */
    CanMessage.id = 41 | CAN_LPDU_FD_U32;
    can_retval = Can_43_LLCE_Write(CanHO_Config1_TX1, &CanMessage);
    ASSERT(E_OK == can_retval);
    /* Check status on this host (main host) and shuffle the payload in order to send another frame to the logger*/
    can_retval = Check_Status(&CanMessage);
    ASSERT(E_OK == can_retval);

    /* Ctrl 3 sends a message with ID=43 to ctrl 2, which has a filter (CanHO_Config1_RX2_log) which accepts it and has also an advanced
    feature reference to CanAdvancedFeature_1 in Llce_Af plugin which says "send this frame to the main host, but also to a second host (the logger app)" */
    CanMessage.id = 43 | CAN_LPDU_FD_U32;
    can_retval = Can_43_LLCE_Write(CanHO_Config1_TX3, &CanMessage);
    ASSERT(E_OK == can_retval);
    can_retval = Check_Status(&CanMessage);
    ASSERT(E_OK == can_retval);
```

所以是外部 loopback 连接 CAN1/0 和 CAN3/2，然后从 CAN1/3 发送的帧外部回环回 CAN0/2，然后 CAN0/2 由 Host1 接收的同时，logging 到 Host2。

修改 Logger 源代码，将 for 改为 while(1)来一直发送 CAN 包：

```
/* Sending frames from CONTROLLER1/3 to CONTROLLER0/2. They have wired connection on the EVB = LoopBack. */
  for (u16MbGlobalIndex = 0; u16MbGlobalIndex < ITER_NR; u16MbGlobalIndex++)
//while(1)
  {
```

2.4 M7 BootLoader ATF 镜像冲突检查

目前 LLCE Logger 镜像链接文件是：
C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\Makefile:

```
app_host1_logging: $(OBJS) $(SRC_STARTUP)
        @$(COMPILE) $(SRC_STARTUP_ASM) $^ -o $(TMP_location)/$@.elf -T $(LF_ASR_HOST1)

LF_ASR_HOST1    = $(PLUGINS_DIR_RTD)/Platform_$(AR_PKG_RTD_NAME)/build_files/$(TOOLCHAIN)/linker_ram_c0.ld # Linker file for Host1
```

所以其 Mapping 是：

C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Platform_TS_T40D11M40I2R0\build_files\gcc\linker_ram_c0.ld

```
MEMORY
{
  int_itcm                : ORIGIN = 0x00000000, LENGTH = 0x00000000 /* 0KB - Not Supported */
  int_dtcm                : ORIGIN = 0x20000000, LENGTH = 0x0000E000 /* 64K */
  int_dtcm_stack          : ORIGIN = 0x2000E000, LENGTH = 0x00002000 /* 8K */
  int_sram_shareable      : ORIGIN = 0x22C00000, LENGTH = 0x00004000 /* 16KB */
  int_sram_c0             : ORIGIN = 0x34000000, LENGTH = 0x00180000 /* 1.5MB */
  int_sram_no_cacheable_c0  : ORIGIN = 0x34180000, LENGTH = 0x00080000 /* 512KB, needs to include int_results */
  ram_end_c0              : ORIGIN = 0x34200000, LENGTH = 0x00000000 /* End of core 0 ram */
  int_sram_c1             : ORIGIN = 0x34200000, LENGTH = 0x00180000 /* 1.5MB */
  int_sram_no_cacheable_c1  : ORIGIN = 0x34380000, LENGTH = 0x00080000 /* 512KB, needs to include int_results */
  ram_end_c1              : ORIGIN = 0x34400000, LENGTH = 0x00000000 /* End of core 1 ram */
  int_sram_c2             : ORIGIN = 0x34400000, LENGTH = 0x00180000 /* 1.5MB */
  int_sram_no_cacheable_c2  : ORIGIN = 0x34580000, LENGTH = 0x00080000 /* 512KB, needs to include int_results */
  ram_end_c2              : ORIGIN = 0x34600000, LENGTH = 0x00000000 /* End of core 2 ram */
  ram_rsvd2               : ORIGIN = 0x34600000, LENGTH = 0        /* End of SRAM */

  LLCE_CAN_SHAREDMEMORY   : ORIGIN = 0x43800000 LENGTH = 0x3C800
  LLCE_LIN_SHAREDMEMORY   : ORIGIN = 0x4383C800 LENGTH = 0xa0
  LLCE_BOOT_END           : ORIGIN = 0x4383C8A0 LENGTH = 0x50
  LLCE_MEAS_SHAREDMEMORY  : ORIGIN = 0x4384FFDF LENGTH = 0x20
}
```

而Bootloader的SRAM镜像mapping是：
C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\linkfiles\linker_ram_gcc.ld：

```
MEMORY
{
  /* 16KiB */
  int_sram_hse          : ORIGIN = 0x22C00000, LENGTH = 0x00004000
  /* 128KiB */
  int_sram_no_cacheable   : ORIGIN = 0x35300000, LENGTH = 0x00020000
  /* 512KiB */
```

```
    int_sram              : ORIGIN = 0x35320000, LENGTH = 0x00080000
    /* 4KiB  */
    int_sram_stack        : ORIGIN = 0x353A0000, LENGTH = 0x00001000
    /* End of SRAM */
    ram_rsvd2             : ORIGIN = .,         LENGTH = 0
}
```

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 C0 Core0镜像和fip的SRAM镜像加载运行地址是没有冲突的，所以不需要移动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_4.0.2\eclipse\plugins\Platform_TS_T40D11M40I2R0\startup\include\core_specific.h。

```
#if defined(CORE0)
/* Multiple core approach */
/* Number of entries in the memory tables */
#define CPU_MPU_MEMORY_COUNT (11U)

/*
  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 0x2000FFFF        64  Normal            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 1MB)       0x34000000 0x340FFFFF      1024  Normal            Write-Back/Allocate  Write-Back/Allocate
No           Yes            Read/Write           Read/Write
      6  RAM(second 512K)   0x34100000 0x3417FFFF       512  Normal            Write-Back/Allocate
Write-Back/Allocate  No           Yes            Read/Write           Read/Write
      7  Non-Cacheable RAM  0x34180000 0x341FFFFF       512  Normal            None                 None                 Yes
Yes          Read/Write           Read/Write
      8  Peripherals        0x40000000 0x5FFFFFFF     524288  Strongly Ordered  None                 None                 Yes        No
```

```
Read/Write        Read/Write
     9 LLCE         0x43000000 0x43FFFFFF    16384 Strongly Ordered 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, 0x34100000UL, 0x34180000UL, 0x40000000UL, 0x43000000UL,
0xE0000000UL};
static const uint32 rasr[CPU_MPU_MEMORY_COUNT] = {0x1004003FUL, 0x03080039UL, 0x0308001FUL,
0x130C001BUL, 0x030B001DUL, 0x030B0027UL, 0x030B0025UL, 0x130C0025UL, 0x13040039UL,
0x1304002FUL, 0x13040027UL};
```

所以总结如下：

镜像名	范围	地址	M7 MCAL MPU 配置
M7 MCAL 镜像	Start address	0x34000000	0x34000000~ 0x3417FFFF Cacheable. 0x34180000 0x341FFFFF Non-Cacheable
	End address	0x34200000	
Bootloader 镜像	Start address	0x35300000	Non-Cacheable
	End address	0x353A1000	
ATF FIP 镜像	Start address	0x343008c0	Non-Cacheable
	End address	0x3432c300	

镜像无重叠，MPU配置无冲突。

2.5 LLCE Logger Demo 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。

C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\platform_common\platform_init\src\Platform_Init.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
```

2.6 LLCE Logger Demo 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\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\platform_common\platform_i

nit\src\Platform_Init.c中，将模式初始化删除掉：

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

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
  Mcu_Init(MCU_VARIANT);
#endif
```

2.7 LLCE Logger Demo 程序去掉 PORT INIT

C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\platform_common\platform_init\src\Platform_Init.c中

```
#if 0
#ifdef USE_PORT_HLD
  Port_Init(PORT_VARIANT);
#else
  Siul2_Port_Ip_Init(NUM_OF_CONFIGURED_PINS0, g_pin_mux_InitConfigArr0);
#endif
#endif
```

2.8 中断冲突说明

在代码C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\generic\src\bootloader.c中：

```
int main(void)
{…
 SysDal_DisableAllInterruptSources(); //所以Bootloader会把之前初始化的中断全部关闭
}
```

所以在代码C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\platform_common\platform_init\src\Platform_Init.c中，重新初始化LLCE相关中断：

```
  /* Configurations for IRQ routing, priority and enable through Platform plugin. */
  Platform_Init(NULL);
```

所以以上代码不会导致 M 核的中断配置冲突。

2.9 去掉其它无用初始化

在代码C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\platform_common\platform_init\src\Platform_Init.c中，去掉RM模块初始化。

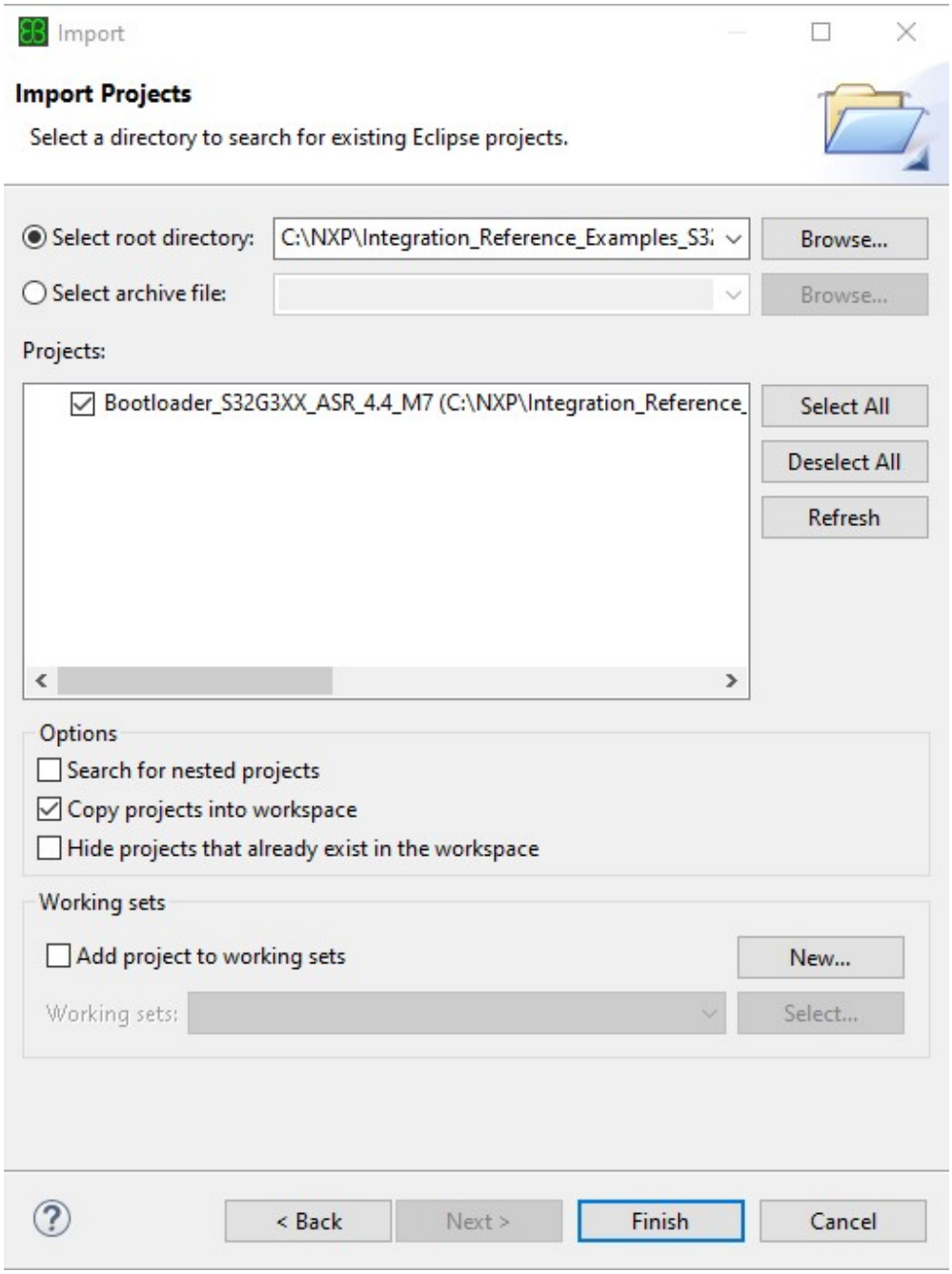
```
#if 0
```

```
  #if (RM_PRECOMPILE_SUPPORT == STD_OFF)
      Rm_Init(&Rm_Config_VS_0);
  #else
      Rm_Init(NULL);
  #endif
#endif
```

3 Bootloader 工程说明

运行Platform_Software_Integration_S32G3_2023_02.exe安装bootloader工程，然后将C:\NXP\Integration_Reference_Examples_S32G3_2023_02\applications\realtime\Tresos\eclipse\plugins下的所有plugin拷贝到C:\NXP\SW32G_RTD_4.4_4.0.2\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_S32G3_2023_02\applications\realtime\Tresos\workspaces\Bootloader_S32G3XX_ASR_4.4_M7
从而打开工程Bootloader_S32G3XX_ASR_4.4_M7。(勾选 Copy projects into workspace)

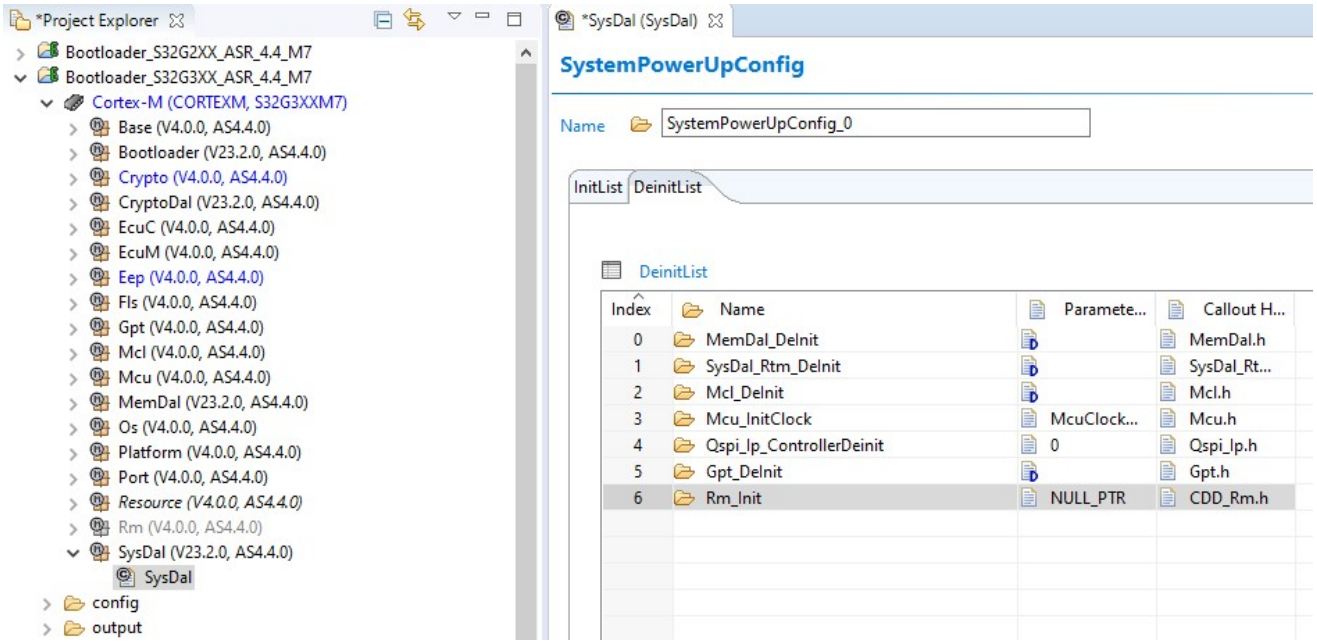


然后双击 Bootloader_S32G3XX_ASR_4.4_M7->Cortex-M(…)，则可以打开所有模块(如果有模块加载失败，检查之前是否将 platform 的 plugin 拷贝到了 mcal 中，并且确保 C:\EB\tresos\links 仅有 SW32G_RTD_4.4_4.0.2.link 的连接)。

3.1 关掉 XRDC 支持

为了简单化工程，先去掉 XRDC 支持：

1. 在 Rm(V4.0.0,AS4.4.0)模块上右击，选择 Disable 关掉此模块。
2. 选择 SysDal(V23.2.0,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：

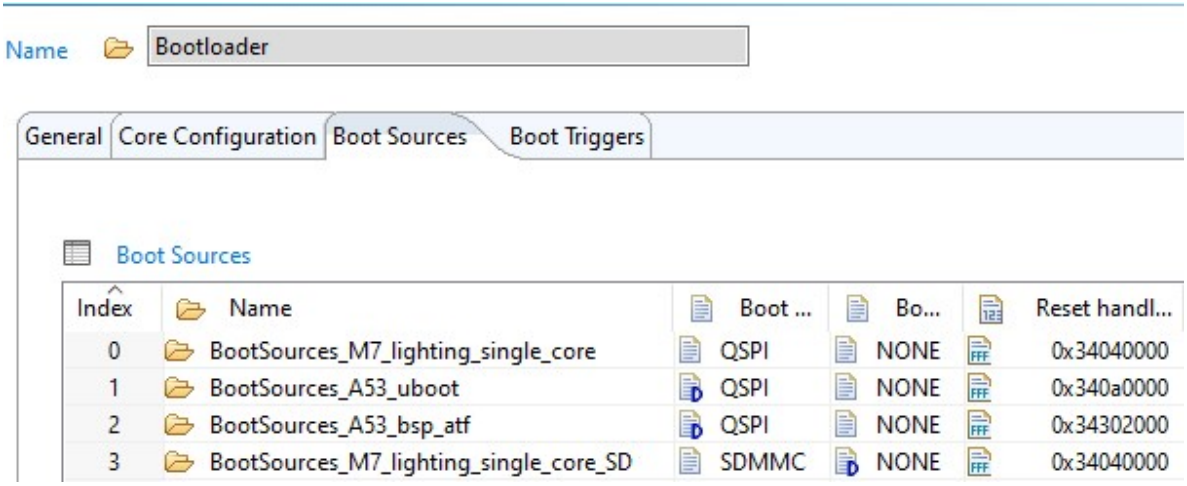
1. Bootloader(…)->Cortex-M(…)：disabled 掉 Eep 和 MemDal 模块。
2. Bootloader(…)-> Cortex-M (…)->SysDal (…)->SysDal->Powerup-> SystemPowerUpConfig_0:

InitList 中将 MemDal 的 init 删除掉。

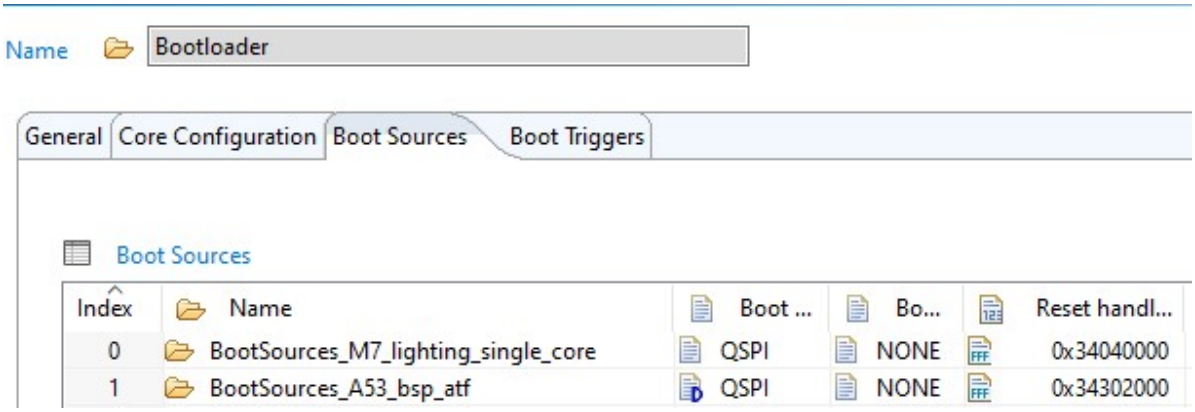
DeinitList 中将 MemDal 的 deinit 删除掉。

3. 去掉 eMMC/SD 后，还需要把 Boot Sources 去掉：

Bootloader(…)-> Cortex-M (…)->Bootloader (…)-> Bootloader->Boot Sources:



将位于 SDMMC 上的 BootSources_M7_lighting_single_core_SD 删除掉，然后我们只保留一个 M7 镜像，一个 A53 ATF 镜像，所以把 uboot 也删除掉：



然后在 Bootloader(…)右击，选把 Generate Project 生成 secure boot 的源代码，可以成功。

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(…)-> Cortex-M (…)->Bootloader (…)-> Bootloader->Boot Sources-> BootSources_M7_lighting_single_core->Boot image fragments-> ImageFragments_0:
   - SMR Index= 0 //1 修改为 0，这样才可能保证 disable secure boot 成功
2. 同理修改掉BootSources_A53_bsp_atf
3. Bootloader(…)-> Cortex-M (…)：disabled 掉CryptoDal 和Crypto 模块。

2. Bootloader(…)-> Cortex-M (…)->Bootloader(…)->Bootloader->General: Enable Secure Boot= unchecked.

3. Bootloader(…)-> Cortex-M (…)->SysDal (…)->SysDal->Powerup-> SystemPowerUpConfig_0: InitList 中将CryptoDal_Init 的init 删除掉。

然后在 Bootloader(…)右击，选把 Generate Project 生成 non secure boot 的源代码，可以成功。

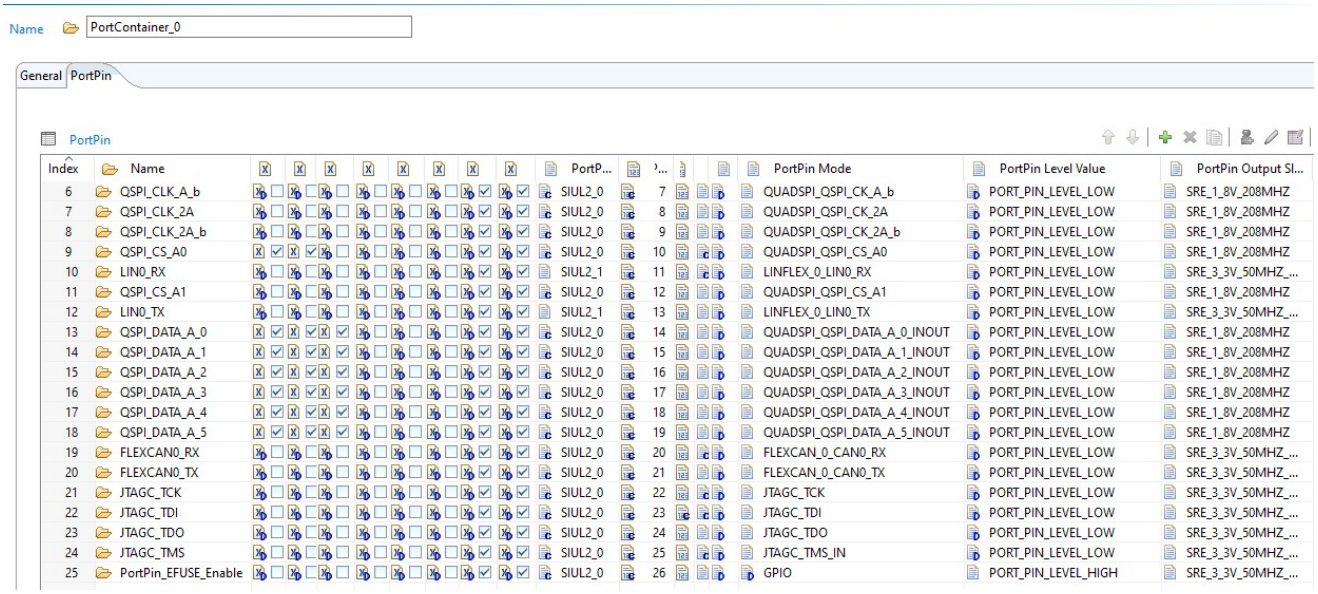
3.4 增加 LLCE 驱动所需要的 PORT 的初始化

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…-> Cortex-M (…)->Port(…) -> Port->PortContainer-> PortContainer_0->PortPin：

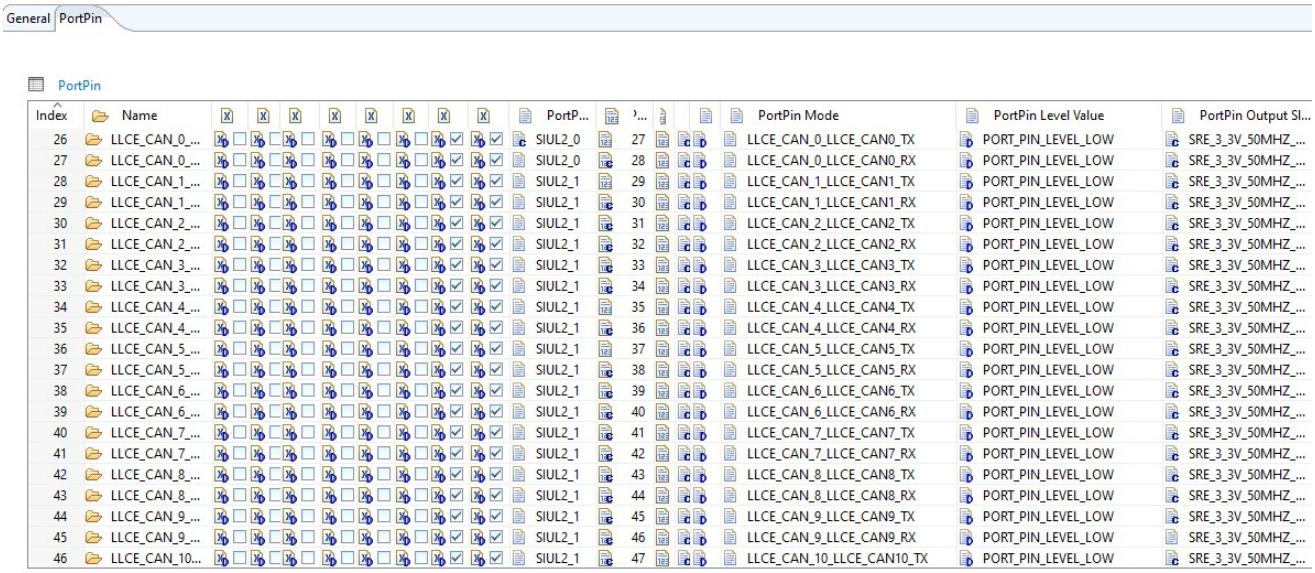
将USDHC 相关管脚全部删除。之后其它管脚的Port ID 全部要自动计算重排，方法如下：

以LIN0_TX为例，点击LIN0_TX进入，在PortPinID外，点击“铅笔”图标，会切换为“计算器”图标，再点击一下，就会将PortPinID自动计算为13，以此方法重排所有报错管脚。

然后在General 里的：PortNumberOfPortPins*自动计算为26。



增加LLCE CAN的2X16=32个管脚：本文是采用将：C:\EB\tresos\workspace\Sample_App_LLCE_CAN2CAN_HSE_S32G3XX\config\Port.xdm中LLCE CAN管脚直接拷贝到C:\EB\tresos\workspace\Bootloader_S32G3XX_ASR_4.4_M7\config\Port.xdm然后自动修改的方法：



总数自动计算为26+2X16=58。

3.5 解决 Bootloader,MCAL 与 Linux 的 clock 冲突

以LLCE CAN Logger 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：(注意 under MCU control 代表 M7 是否控制此时钟，如果没有控制，说明 M 核不需要，可以由 Linux 去初始化，

A 核时钟是考虑了修改 A core 为 1.2G 的情况下的导出值)。

考虑 PFE 支持，最终 Bootloader 配置也参考了 PFE Master 工程。

	Bootloader: clocksettingconfig	LLCE LOGGER:MCU clocksettingconfig	Linux	Bootloader: clocksettingconfig
	Config_0(初始化)	Config_0 Config_		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		保持不变
McuCorePLL	under MCU control=checked 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: 1.6E9		Source: 修改为Linux 要求的源和值 FXOSC_CLK Name: RDIV:1 MFD:60 PLL_PHI0: 1.2E9 PLL_PHI1: 0 PLL_VCO: 2.4E9
McuCoreDFS	under MCU control=checked Name: DFS1: 8.0E8 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 DFS1 MFI:1 DFS1 MFN:18 DFS2: 0 DFS3:0 DFS4: 0 DFS5: 0 DFS6: 0
McuPeriphPLL	under MCU control=checked Source: FXOSC_CLK Name: PLL_PHI0: 1.0E8 PLL_PHI1: 8.0E7 PLL_PHI2: 4.0E7 PLL_PHI3:0 PLL_PHI4:0 PLL_PHI5:1.25E8 PLL_PHI6:0 PLL_PHI7:0 PLL_PHI8:0 PLL_VCO: 2.0E9	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_PHI1: 6.25E7 PLL_PHI2: 0 PLL_PHI3:0 PLL_PHI4:0 PLL_PHI5: 1.25E8 PLL_PHI6:0 PLL_PHI7:5.0E7 PLL_VCO: 2.0E9		Source: FXOSC_CLK Name: PLL_PHI0: 1.0E8 PLL_PHI1: 8.0E7 PLL_PHI2: 4.0E7 PLL_PHI3: 1.25E8//UART 的根时钟，所以配置为与 Linux 相同 PLL_PHI4: 4.0E7 PLL_PHI5:1.25E8 PLL_PHI6:0 PLL_PHI7:0 PLL_PHI8:0 PLL_VCO: 2.0E9
McuPeriphDFS	under MCU control=checked 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		保持不变 Name: DFS1: 8.0E8 DFS2: 0 DFS3: 8.0E8 DFS4: 0 DFS5: 0 DFS6: 0
McuAccelPLL	under MCU control=unchecked Source: FXOSC_CLK Name:	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_PHI1: 6.0E8		LLCE时钟源 Source: FXOSC_CLK Name: PLL_PHI0: 0

	PLL_PHI0: 0 PLL_PHI1: 0 PLL_PHI2: 0 PLL_VCO:0	PLL_VCO:2.4E9		PLL_PHI1: 6.0E8 PHI1 Division:3 PLL_VCO:2.4E9 RDIV:1 MFD:60
McuDDRPLL	under MCU control=unchecked Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_VCO:0	Source: FXOSC_CLK Name: PLL_PHI0: 0 PLL_VCO:0		配置为不受 MCU 控制
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:(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	保持不变 不受 MCU 控制	Source: FXOSC_CLK CLKOUT0:0	保持不变 不受 MCU 控制
Cgm0Mux2	Source: FXOSC_CLK CLKOUT0:0	保持不变 不受 MCU 控制	Source: FXOSC_CLK CLKOUT0:0	保持不变 不受 MCU 控制
Cgm0Mux3	Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7	Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7		保持不变 Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7
Cgm0Mux4	under MCU control=unchecked Source: PERIPH_PLL_PHI1_CLK	Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK:	Source: PERIPH_P	under MCU control=unchecked Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 0

	FTM_0_REF_CLK: 0	5E6	LL_PHI1_CLK FTM_0_REF_CLK: 8E7	
Cgm0Mux5	under MCU control=unchecked 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: 0	Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 8E7	保持不变 不受 MCU 控制
Cgm0Mux6	under MCU control=unchecked Source: PERIPH_PLL_PHI1_CLK FLEXRAY_PE_CLK: 0	Source: FIRC_CLK FLEXRAY_PE_CLK: 0	FLEXRAY_PE_CLK: 133M	保持不变 不受 MCU 控制
Cgm0Mux7	under MCU control=checked Source: PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 4.0E7	under MCU control=checked Source: PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 4.0E7	Source: PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 1.3E8	保持不变 Source: PERIPH_PLL_PHI2_CLK CAN_PE_CLK: 4.0E7
Cgm0Mux8	under MCU control=unchecked 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 LIN	Source: (修改为Linxu 使用的时钟源和值，) PERIPH_PLL_PHI3_CLK CAN_PE_CLK: 1.25E8 (先配置完 periph_pll_phi3 后再

			BAUD_CLK: 1.25E8	可以配置)
Cgm0Mux12	under MCU control=checked 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:2.6E8	保持不变 Source: PERIPH_PLL_DFS1_CLK QSPI_2X_CLK:2.6E8
Cgm0Mux14	under MCU control=checked Source: FIRC_CLK SDHC_CLK: 4.0E7	Source: FIRC_CLK SDHC_CLK: 0	Source: PERIPH_PLL_DFS3_CLK SDHC_CLK: 4.0E7	修改为不受 MCU 控制 Source: PERIPH_PLL_DFS3_CLK SDHC_CLK: 0
Cgm0Mux16	under MCU control=unchecked Source: FIRC_CLK SPI_CLK: 4.8E7	Source: PERIPH_PLL_PHI7_CLK SPI_CLK: 5E7	SPI_CLK:0	保持不变 不受 MCU 控制
Cgm1Mux0	under MCU control=checked 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:1.2E9	Source:此为A53 clock，修改为与Linux 相同的源和值 CORE_PLL_PHI0_CLK A53_CORE_CLK:1.2E9 (先配置完core_pll_phi0后再可以配置)
Cgm1Pcs	under MCU control=unchecked PCFS_4: 8.0E8	PCFS_4: 0		保持不变 不受 MCU 控制

Cgm2Pcs	under MCU control=unchecked PCFS_33: 0	PCFS_33: 0		保持不变 不受 MCU 控制
Cgm2Mux0	under MCU control=unchecked Source: FIRC_CLK PFE_PE_CLK, ACCEL_3_CLK:0	Source: ACCEL_PLL_PHI1_CLK PFE_PE_CLK, ACCEL_3_CLK: 6.0E8		Source: ACCEL_PLL_PHI1_CLK PFE_PE_CLK, ACCEL_3_CLK: 6.0E8
Cgm2Mux1	under MCU control=unchecked Source: FIRC_CLK PFE_MAC_0_TX_DIV_CLK, ACCEL_4_CLK:0	Source: PERIPH_PLL_PHI5_CLK PFE_MAC_0_TX_DI V_CLK, ACCEL_4_CLK:1.25E8	1.25E8	注意，本来应该配置为：under MCU control=checked Source: SERDES_1_XPCS_0_TX PFE_MAC_0_TX_DIV_CLK, ACCEL_4_CLK:1.25E8。 但在实际配置中发现会报错，所以这个保持为： under MCU control=unchecked //一个可能的bug，只能以后用到时手动修改源代码修复。
Mcu GENCTRL1 _EMAC0	under MCU control=unchecked Source: PFE_MAC0_TX_CLK= SERDES_1_XPCS_0_TX PFE_MAC_0_TX_CLK : 1.25E8	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
Mcu GENCTRL1 _EMAC1	under MCU control=unchecked Source: PFE_MAC1_TX_CLK= SERDES_1_XPCS_1_TX PFE_MAC_1_TX_CLK : 1.25E8	Source: PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0		under MCU control=checked Source: SERDES_1_XPCS_1_TX PFE_MAC_1_TX_CLK : 1.25E8
Mcu GENCTRL1 _EMAC2	under MCU control=unchecked Source: PFE_MAC2_TX_CLK= SERDES_0_XPCS_1_TX PFE_MAC_2_TX_CLK : 1.25E8	Source: PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0		under MCU control=checked Source: PFEMAC2_TX_DIV_CLK PFE_MAC_2_TX_CLK : 1.25E8
Cgm2Mux2	under MCU control=unchecked Source: FIRC_CLK	Source: PERIPH_PLL_PHI5_CLK PFE_MAC_1_TX_CLK,	1.25E8	支持LLCE to PFE Master Source: PERIPH_PLL_PHI5_CLK

	PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK:0	GMAC_1_TX_CLK, REC_CLK, PFE_MAC_1_TX_DIV_CLK : 1.25E8		PFE_MAC_1_TX_CLK, GMAC_1_TX_CLK, REC_CLK, PFE_MAC_1_TX_DIV_CLK : 1.25E8
Cgm2Mux3	under MCU control=unchecked Source: FIRC_CLK PFE_MAC_2_TX_CLK, GMAC_1_REF_DIV_CL:0	Source: PERIPH_PLL_PHI5_CLK PFE_MAC_2_TX_CLK, GMAC_1_REF_DIV_CLK, PFE_MAC_2_TX_DIV_CLK: 1.25E8	1.25E8	支持LLCE to PFE Master Source: PERIPH_PLL_PHI5_CLK PFE_MAC_2_TX_CLK, GMAC_1_REF_DIV_CLK, PFE_MAC_2_TX_DIV_CLK: 1.25E8
Cgm2Mux4	under MCU control=unchecked Source: FIRC_CLK PFE_MAC_0_RX_CLK, GMAC_1_RX_CLK：4.8E7	Source: PFE_MAC_0_EXT_RX_CLK 1.25E8	1.25E8	支持LLCE to PFE Master under MCU control=checked Source: SERDES_1_XPCS_0_CDR (PFE_MAC_0_RX_CLK, GMAC_1_RX_CLK)：1.25E8
Cgm2Mux5	under MCU control=unchecked Source: FIRC_CLK PFE_MAC_1_RX_CLK, SEQ_CLK：4.8E7	under MCU control=unchecked Source: PFE_MAC_1_EXT_RX_CLK 1.25E8	1.25E8	under MCU control=checked Source: PFE_MAC_1_EXT_RX_CLK PFE_MAC_1_RX_CLK, SEQ_CLK： 1.25E8
Cgm2Mux6	under MCU control=unchecked Source: FIRC_CLK PFE_MAC_2_RX_CLK, APEXD_0_CLK：4.8E7	Source: PFE_MAC_2_EXT_RX_CLK 1.25E8	1.25E8	Source: PFE_MAC_2_EXT_RX_CLK PFE_MAC_2_RX_CLK, APEXD_0_CLK ：1.25E8
Cgm2Mux7	under MCU control=unchecked Source: FIRC_CLK PFEMAC0_REF_DIV_CLK：0	Source: PFE_MAC_0_EXT_REF_CLK ：5.0E7		under MCU control=checked Source: PFE_MAC_0_EXT_REF_CLK PFEMAC0_REF_DIV_CLK ：5.0E7
Cgm2Mux8	under MCU control=unchecked Source: FIRC_CLK PFEMAC1_REF_DIV_CLK：0	Source: FIRC_CLK PFEMAC1_REF_DIV_CLK：0		under MCU control=checked Source: PFE_MAC_1_EXT_REF_CLK PFEMAC1_REF_DIV_CLK ：5.0E7
Cgm2Mux9	under MCU control=unchecked	Source: FIRC_CLK		under MCU control=checked

	Source: FIRC_CLK PFEMAC2_REF_DIV_CLK：0	PFEMAC2_REF_DIV_CLK：0		Source: PFE_MAC_2_EXT_REF_CLK PFEMAC2_REF_DIV_CLK ：5.0E7
Cgm5Mux0	under MCU control=unchecked 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
Cgm6Mux0	under MCU control=checked Source: GMAC_EXT_TS_CLK GMAC_TS_CLK: $1.0^{E}8$	Source: FIRC_CLK GMAC_TS_CLK: 0	Source: DDRPLL_PHI0 DDR_CLK: 2.0E8	under MCU control=checked Source: PERIPH_PLL_PHI4_CLK GMAC_TS_CLK: $2.0^{E}8$
Cgm6Mux1	under MCU control=checked Source: PERIPH_PLL_PHI5_CLK GMAC_0_TX_CLK: 1.25E8		0	under MCU control=checked Source: PERIPH_PLL_PHI5_CLK GMAC_0_TX_CLK: 1.25E8
Cgm6Mux2	under MCU control=checked Source: GMAC_0_RX_CLK: GMAC_0_EXT_RX_CLK 1.25E8	under MCU control=checked Source: GMAC_0_RX_CLK: FIRC_CLK 4.8E7	2.5E7	保持不变 under MCU control=checked Source: GMAC_0_RX_CLK: GMAC_0_EXT_RX_CLK 1.25E8
Cgm6Mux3	under MCU control=unchecked Source: FIRC_CLK GMAC_0_REF_CLK: 4.8E7	under MCU control=checked Source: FIRC_CLK GMAC_0_REF_CLK: 4.8E7		Source: 保持不变，不受MCU 控制
McuRtc ClockSelect	under MCU control=checked Source: FIRC_CLK 4.8E7	Source: FIRC_CLK 4.8E7		Source: 保持不变 FIRC_CLK 4.8E7
McuClk Monitor	27 项	27 项		保持
McuClock Reference	Name: CM7_CLK= XBAR_CLK:4.0E8 A53_CLK=	Name: M7_CLK_REF= XBAR_CLK:4.0E8 CAN_PE_CLK= CAN_PE_CLK:4.0E7		Name: Name: CM7_CLK= XBAR_CLK:4.0E8 A53_CLK=

	A53_CORE_CLK:8.0E8 CAN_CLK= CAN_PE_CLK:4.0E7 PIT_CLK= XBAR_DIV3_CLK:1.3E8 uSDHC_CLK =SDHC_CLK：4.8E8	LLCE_SUBSYS= XBAR_DIV2_CLK2.0E8 LIN_PE_CLK= LINFLEXD_CLK=2.4E7		A53_CORE_CLK:8.0E8 CAN_CLK= CAN_PE_CLK:4.0E7 PIT_CLK= XBAR_DIV3_CLK:1.3E8 LLCE_SUBSYS= XBAR_DIV2_CLK:2.0E8 LIN_PE_CLK= LINFLEXD_CLK: 6.25E7

说明：

1. 从BSP38用以下命令导出clk：

Uboot:

```
=> clk dump
ID  Rate            Used       Name
----------------------------------------------
0   1200000000        0        a53
1   400000000         1        serdes_axi
2   51000000          1        serdes_aux
3   133333333         1        serdes_apb
4   100000000         1        serdes_ref
5   80000000          0        ftm0_sys
6   0             0        ftm0_ext
7   80000000          0        ftm1_sys
8   0             0        ftm1_ext
9   133333333         0        flexcan_reg
10  133333333         0        flexcan_sys
11  40000000          0        flexcan_can
12  200000000         0        flexcan_ts
13  62500000          0        linflex_xbar
14  125000000         1        linflex_lin
15  200000000         1        gmac0_ts
16  25000000          0        gmac0_rx_sgmii
17  125000000         0        gmac0_tx_sgmii
18  25000000          0        gmac0_rx_rgmii
19  125000000         0        gmac0_tx_rgmii
20  0             0        gmac0_rx_rmii
21  0             0        gmac0_tx_rmii
22  0             0        gmac0_rx_mii
23  0             0        gmac0_tx_mii
24  400000000         0        gmac0_axi
25  0             0        spi_reg
26  0             0        spi_module
27  133333333         0        qspi_reg
```

```
28  133333333       0       qspi_ahb
29  266666666       0       qspi_flash2x
30  133333333       0       qspi_flash1x
31  400000000       0       usdhc_ahb
32  133333333       0       usdhc_module
33  400000000       1       usdhc_core
34  32000           0       usdhc_mod32k
35  133333333       0       ddr_reg
36  800000000       0       ddr_pll_ref
37  800000000       0       ddr_axi
38  400000000       0       sram_axi
39  133333333       0       sram_reg
40  133333333       0       i2c_reg
41  133333333       0       i2c_module
42  66666666        0       siul2_reg
43  51000000        0       siul2_filter
44  133333333       0       crc_reg
45  133333333       0       crc_module
46  120000000       0       eim0_reg
47  120000000       0       eim0_module
48  66666666        0       eim123_reg
49  66666666        0       eim123_module
50  66666666        0       eim_reg
51  66666666        0       eim_module
52  66666666        0       fccu_module
53  51000000        0       fccu_safe
54  66666666        0       rtc_reg
55  32000           0       rtc_sirc
56  51000000        0       rtc_firc
57  133333333       0       swt_module
58  51000000        0       swt_counter
59  133333333       0       stm_module
60  133333333       0       stm_reg
61  133333333       0       pit_module
62  133333333       0       pit_reg
63  400000000       0       edma_module
64  400000000       0       edma_ahb
65  80000000        1       sar_adc_bus
66  66666666        0       cmu_module
67  66666666        0       cmu_reg
68  133333333       0       tmu_module
69  133333333       0       tmu_reg
70  133333333       0       flexray_reg
71  0               0       flexray_pe
72  66666666        0       wkpu_module
73  66666666        0       wkpu_reg
74  66666666        0       src_module
75  66666666        0       src_reg
76  66666666        0       src_top_module
77  66666666        0       src_top_reg
78  133333333       0       ctu_module
79  80000000        0       ctu_ctu
```

```
 80  200000000          0        dbg_sys4
 81  400000000          0        dbg_sys2
 82  400000000          0        m7
 83  133333333          0        dmamux_module
 84  133333333          0        dmamux_reg
 85  600000000          0        gic_module
 86  133333333          0        mscm_module
 87  133333333          0        mscm_reg
 88  133333333          0        sema42_module
 89  133333333          0        sema42_reg
 90  66666666           0       xrdc_module
 91  66666666           0       xrdc_reg
 92  0              0        clkout0
 93  0              0        clkout1
 94  100000000          0        usb_mem
 95  32000            0       usb_low
 96  125000000          1        pfe0_rx_sgmii
 97  125000000          1        pfe0_tx_sgmii
 98  125000000          0        pfe0_rx_rgmii
 99  125000000          0        pfe0_tx_rgmii
100 125000000          0        pfe0_rx_rmii
101 125000000          0        pfe0_tx_rmii
102 125000000          0        pfe0_rx_mii
103 125000000          0        pfe0_tx_mii
104 125000000          1        pfe1_rx_sgmii
105 125000000          1        pfe1_tx_sgmii
106 125000000          0        pfe1_rx_rgmii
107 125000000          0        pfe1_tx_rgmii
108 125000000          0        pfe1_rx_rmii
109 125000000          0        pfe1_tx_rmii
110 125000000          0        pfe1_rx_mii
111 125000000          0        pfe1_tx_mii
112 125000000          0        pfe2_rx_sgmii
113 125000000          0        pfe2_tx_sgmii
114 125000000          1        pfe2_rx_rgmii
115 125000000          1        pfe2_tx_rgmii
116 125000000          0        pfe2_rx_rmii
117 125000000          0        pfe2_tx_rmii
118 125000000          0        pfe2_rx_mii
119 125000000          0        pfe2_tx_mii
120 300000000          0        pfe_axi
121 300000000          0        pfe_apb
122 600000000          1        pfe_pe
123 200000000          0        pfe_ts
124 40000000           0       llce_can_pe
125 200000000          0        llce_sys
=>
```

Kernel:

```
root@s32g399ardb3:~# cat /sys/kernel/debug/clk/clk_summary
                                 enable  prepare  protect                                duty  hardware
   clock                         count    count    count        rate   accuracy phase  cycle    enable
```

```
-------------------------------------------------------------------------------------------------
pcf85063-clkout          0      0      0      32768          0      0  50000         Y
llce_sys                 0      0      0  200000000          0      0  50000         Y
llce_can_pe              0      0      0          0          0      0  50000         Y
pfe_ts                   0      0      0  200000000          0      0  50000         Y
pfe_pe                   0      0      0          0          0      0  50000         Y
pfe_apb                  0      0      0          0          0      0  50000         Y
pfe_axi                  0      0      0          0          0      0  50000         Y
pfe2_tx_mii              0      0      0          0          0      0  50000         Y
pfe2_rx_mii              0      0      0  125000000          0      0  50000         Y
pfe2_tx_rmii             0      0      0          0          0      0  50000         Y
pfe2_rx_rmii             0      0      0  125000000          0      0  50000         Y
pfe2_tx_rgmii            0      0      0          0          0      0  50000         Y
pfe2_rx_rgmii            0      0      0  125000000          0      0  50000         Y
pfe2_tx_sgmii            0      0      0          0          0      0  50000         Y
pfe2_rx_sgmii            0      0      0  125000000          0      0  50000         Y
pfe1_tx_mii              0      0      0          0          0      0  50000         Y
pfe1_rx_mii              0      0      0          0          0      0  50000         Y
pfe1_tx_rmii             0      0      0          0          0      0  50000         Y
pfe1_rx_rmii             0      0      0          0          0      0  50000         Y
pfe1_tx_rgmii            0      0      0          0          0      0  50000         Y
pfe1_rx_rgmii            0      0      0          0          0      0  50000         Y
pfe1_tx_sgmii            0      0      0          0          0      0  50000         Y
pfe1_rx_sgmii            0      0      0          0          0      0  50000         Y
pfe0_tx_mii              0      0      0          0          0      0  50000         Y
pfe0_rx_mii              0      0      0          0          0      0  50000         Y
pfe0_tx_rmii             0      0      0          0          0      0  50000         Y
pfe0_rx_rmii             0      0      0          0          0      0  50000         Y
pfe0_tx_rgmii            0      0      0          0          0      0  50000         Y
pfe0_rx_rgmii            0      0      0          0          0      0  50000         Y
pfe0_tx_sgmii            0      0      0          0          0      0  50000         Y
pfe0_rx_sgmii            0      0      0          0          0      0  50000         Y
usb_low                  0      0      0      32000          0      0  50000         Y
usb_mem                  1      1      0  100000000          0      0  50000         Y
```

clkout1	0	0	0	0	0	0	50000	Y
clkout0	0	0	0	0	0	0	50000	Y
xrdc_reg	0	0	0	66666666	0	0	50000	Y
xrdc_module	0	0	0	66666666	0	0	50000	Y
sema42_reg	0	0	0	133333333	0	0	50000	Y
sema42_module	0	0	0	133333333	0	0	50000	Y
mscm_reg	0	0	0	133333333	0	0	50000	Y
mscm_module	0	0	0	133333333	0	0	50000	Y
gic_module	0	0	0	600000000	0	0	50000	Y
dmamux_reg	0	0	0	133333333	0	0	50000	Y
dmamux_module	0	0	0	133333333	0	0	50000	Y
m7	0	0	0	400000000	0	0	50000	Y
dbg_sys2	0	0	0	400000000	0	0	50000	Y
dbg_sys4	0	0	0	200000000	0	0	50000	Y
ctu_ctu	0	0	0	80000000	0	0	50000	Y
ctu_module	0	0	0	133333333	0	0	50000	Y
src_top_reg	0	0	0	66666666	0	0	50000	Y
src_top_module	0	0	0	66666666	0	0	50000	Y
src_reg	0	0	0	66666666	0	0	50000	Y
src_module	0	0	0	66666666	0	0	50000	Y
wkpu_reg	0	0	0	66666666	0	0	50000	Y
wkpu_module	0	0	0	66666666	0	0	50000	Y
flexray_pe	0	0	0	0	0	0	50000	Y
flexray_reg	0	0	0	133333333	0	0	50000	Y
tmu_reg	0	0	0	133333333	0	0	50000	Y
tmu_module	1	1	0	133333333	0	0	50000	Y
cmu_reg	0	0	0	66666666	0	0	50000	Y
cmu_module	0	0	0	66666666	0	0	50000	Y
sar_adc_bus	2	2	0	80000000	0	0	50000	Y
edma_ahb	2	2	0	400000000	0	0	50000	Y
edma_module	2	2	0	400000000	0	0	50000	Y
pit_reg	0	0	0	133333333	0	0	50000	Y
pit_module	0	0	0	133333333	0	0	50000	Y
stm_reg	0	0	0	133333333	0	0	50000	Y
stm_module	1	1	0	133333333	0	0	50000	Y

swt_counter	0	0	0	51000000	0	0	50000	Y
swt_module	0	0	0	133333333	0	0	50000	Y
rtc_firc	1	1	0	51000000	0	0	50000	Y
rtc_sirc	1	1	0	32000	0	0	50000	Y
rtc_reg	1	1	0	66666666	0	0	50000	Y
fccu_safe	0	0	0	51000000	0	0	50000	Y
fccu_module	0	0	0	66666666	0	0	50000	Y
eim_module	0	0	0	66666666	0	0	50000	Y
eim_reg	0	0	0	66666666	0	0	50000	Y
eim123_module	0	0	0	66666666	0	0	50000	Y
eim123_reg	0	0	0	66666666	0	0	50000	Y
eim0_module	0	0	0	120000000	0	0	50000	Y
eim0_reg	0	0	0	120000000	0	0	50000	Y
crc_module	0	0	0	133333333	0	0	50000	Y
crc_reg	0	0	0	133333333	0	0	50000	Y
siul2_filter	0	0	0	51000000	0	0	50000	Y
siul2_reg	0	0	0	66666666	0	0	50000	Y
i2c_module	0	0	0	133333333	0	0	50000	Y
i2c_reg	0	3	0	133333333	0	0	50000	Y
sram_reg	0	0	0	133333333	0	0	50000	Y
sram_axi	0	0	0	400000000	0	0	50000	Y
ddr_axi	0	0	0	800000000	0	0	50000	Y
ddr_pll_ref	0	0	0	800000000	0	0	50000	Y
ddr_reg	0	0	0	133333333	0	0	50000	Y
usdhc_mod32k	0	0	0	32000	0	0	50000	Y
usdhc_core	0	0	0	0	0	0	50000	Y
usdhc_module	0	0	0	133333333	0	0	50000	Y
usdhc_ahb	0	0	0	400000000	0	0	50000	Y
qspi_flash1x	2	2	0	200000000	0	0	50000	Y
qspi_flash2x	0	0	0	400000000	0	0	50000	Y
qspi_ahb	0	0	0	133333333	0	0	50000	Y
qspi_reg	0	0	0	133333333	0	0	50000	Y
spi_module	2	2	0	100000000	0	0	50000	Y
spi_reg	0	0	0	100000000	0	0	50000	Y

gmac0_axi	2	2	0	400000000	0	0	50000	Y
gmac0_tx_mii	0	0	0	0	0	0	50000	Y
gmac0_rx_mii	0	0	0	0	0	0	50000	Y
gmac0_tx_rmii	0	0	0	0	0	0	50000	Y
gmac0_rx_rmii	0	0	0	0	0	0	50000	Y
gmac0_tx_rgmii	1	1	0	125000000	0	0	50000	Y
gmac0_rx_rgmii	1	1	0	25000000	0	0	50000	Y
gmac0_tx_sgmii	0	0	0	125000000	0	0	50000	Y
gmac0_rx_sgmii	0	0	0	25000000	0	0	50000	Y
gmac0_ts	1	1	0	200000000	0	0	50000	Y
linflex_lin	3	3	0	125000000	0	0	50000	Y
linflex_xbar	3	3	0	62500000	0	0	50000	Y
flexcan_ts	0	0	0	200000000	0	0	50000	Y
flexcan_can	0	0	0	0	0	0	50000	Y
flexcan_sys	0	0	0	133333333	0	0	50000	Y
flexcan_reg	0	0	0	133333333	0	0	50000	Y
ftm1_ext	0	0	0	0	0	0	50000	Y
ftm1_sys	0	1	0	80000000	0	0	50000	Y
ftm0_ext	0	0	0	0	0	0	50000	Y
ftm0_sys	0	1	0	80000000	0	0	50000	Y
serdes_ref	2	2	0	100000000	0	0	50000	Y
serdes_apb	2	2	0	133333333	0	0	50000	Y
serdes_aux	2	2	0	51000000	0	0	50000	Y
serdes_axi	2	2	0	400000000	0	0	50000	Y
a53	0	0	0	1200000000	0	0	50000	Y
serdes_125_ext	1	1	0	125000000	0	0	50000	Y
serdes_100_ext	1	1	0	100000000	0	0	50000	Y

root@s32g399ardb3:~#

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，而由A核去初始化时钟，所以原则是M/A核共用的时钟由Bootloader去初始化，比如Core时钟，调试串口时钟等，而M核/A核专用的时钟，M核的，由Bootloader初始化，比如说LLCE_CAN，QSPI NOR，而A核的则由A核去初始化，比如DDR, SDHC等。

3. Core CLK tree 如下：

24.1.2.1 Core-related clock overview

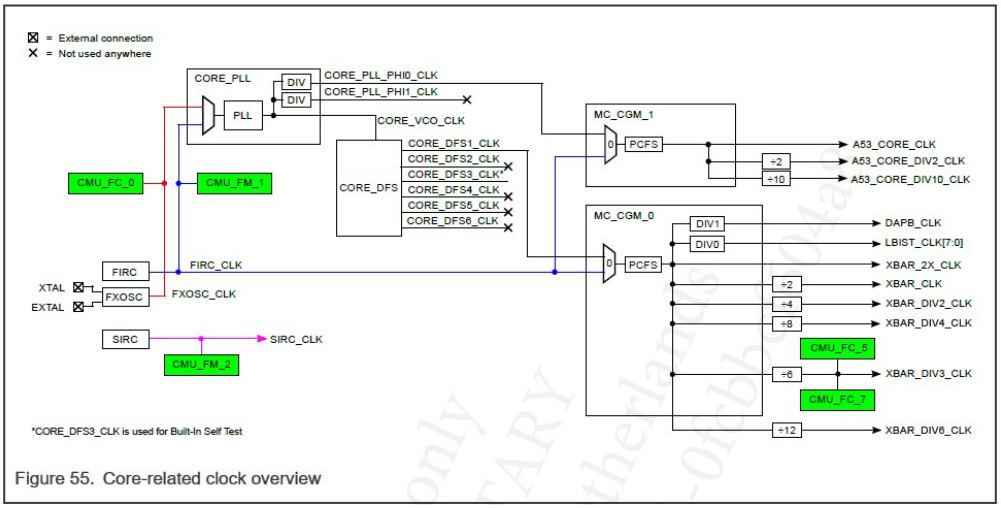
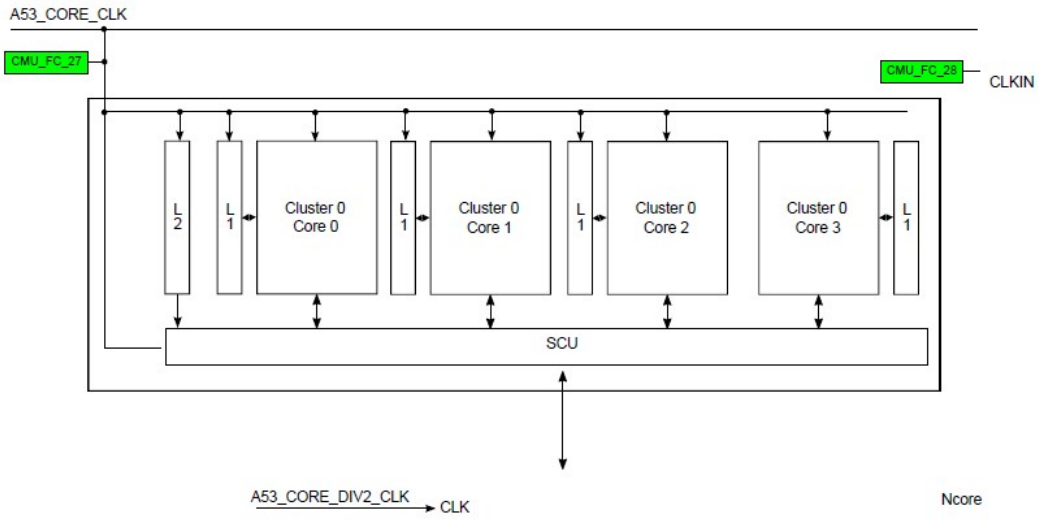


Figure 55. Core-related clock overview

24.7.11.1 Cortex-A53 cluster clocking



24.7.11.2 Cortex-M7 clocking

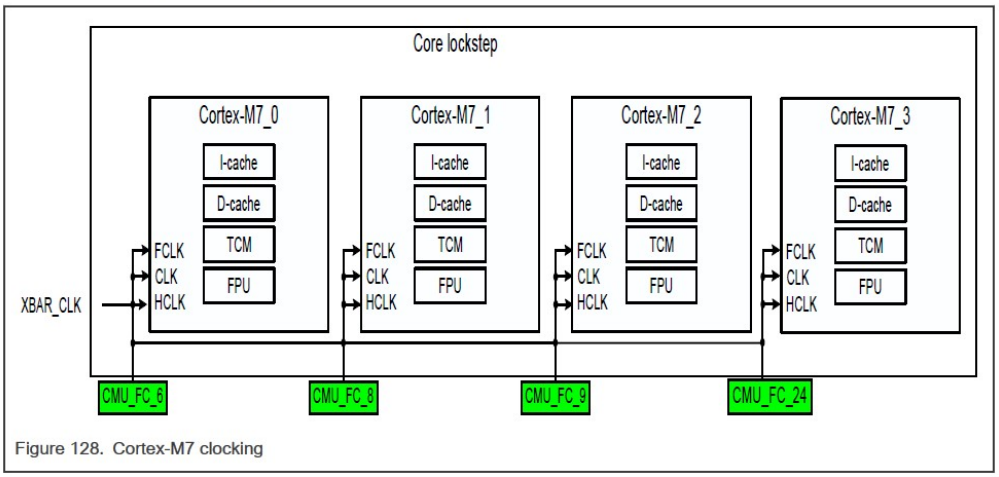


Figure 128. Cortex-M7 clocking

所以：

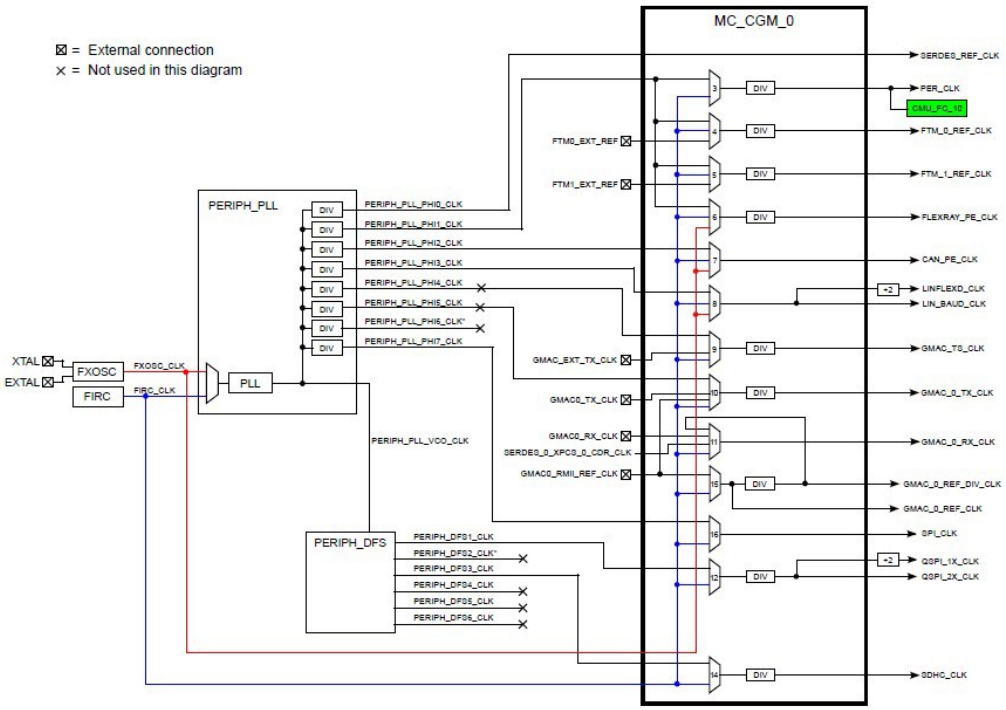
- …McuClockSettingConfig_0->McuCorePll:

RDIV=1; MFD (1 -> 255)=60; PHI0 Division value (0 -> 255) ,则：
PLL_PHI0 Frequency (Calculated) (dynamic range)= 1.2E9 //A53 的时钟为1.2G，修改为与Linux相同，Linux需要从1.3G修改为1.2G。
PLL_VCO Frequency (Calculated) (dynamic range)= 2.4E9

- …McuClockSettingConfig_0->McuCoreDFS: McuDfs_1

DFS1 MFI (1 -> 255)=1; DFS1 MFN (0 -> 35) =18;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

LLCE子系统为：

24.7.2.4 LLCE subsystem clocking overview

Figure 97 shows the LLCE clocking, and Table 104 shows the SIUL2 clock signal configuration for LLCE LPSPI*n*.

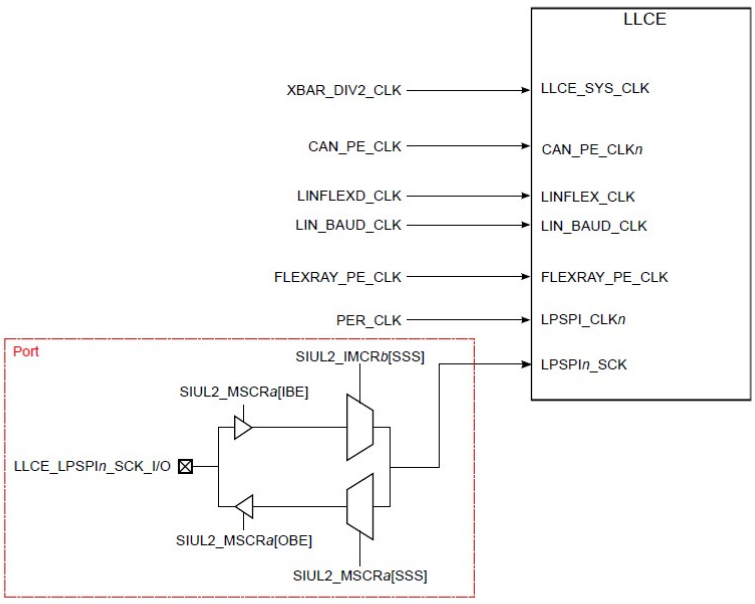


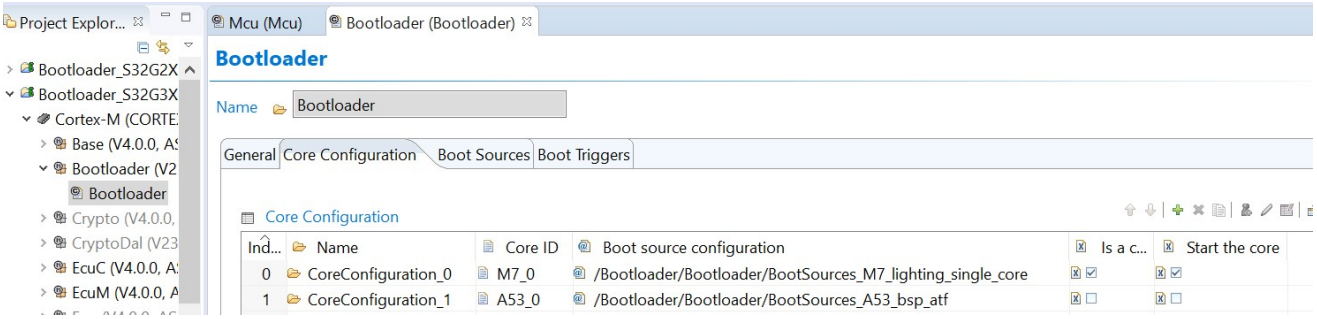
Figure 97. LLCE clocking

当只关注LLCE CAN时：需要关注XBAR_DIV2_CLK=LLCE_SYS_CLK:200Mhz，CAN_PE_CLK:=4.0E7

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 Bootloader(…)->Bootloader->Core Configuration:默认已经配置为:



所以是修改为启动 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: 0x2e200

Boot Core:    A53_0
IVT Location:   SD/eMMC
Load address:   0x342f9900
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：

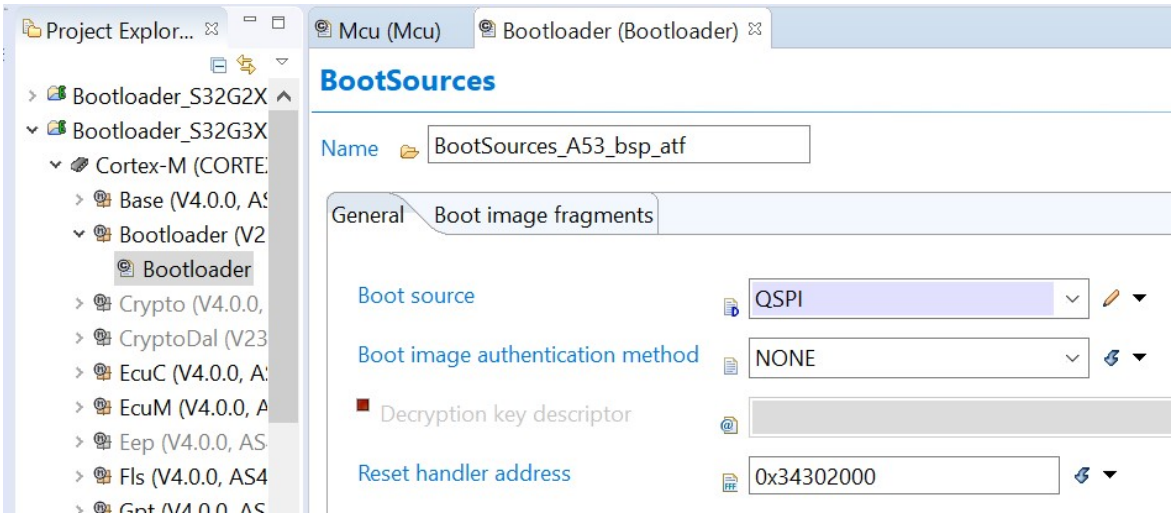
Load image at address (RAM)= 0x342f9900//load address。

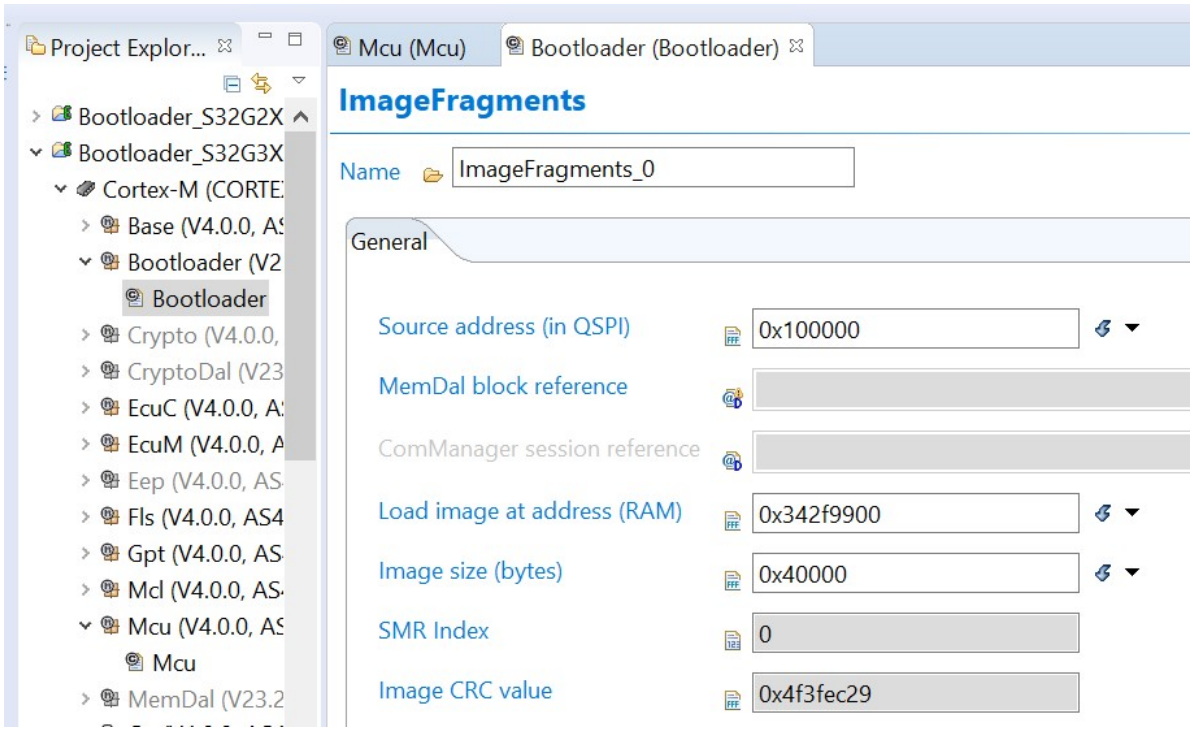
注意注释说明：**“The address to load the image into RAM.**
NOTE !: The start address must be multiple of 8 if you choose CRC32 authentication method, otherwise must be multiple of 64.”

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

Image size (bytes)= 0x4000=256KB= //>0x2e200+0x1240=0x2F440。

Source address(in QSPI)=0x100000//圆整为 4KB 的整数倍。





然后将 A53 也做为需要 Bootloader 启动的镜像，而不是需要 HSE 启动的镜像：

Bootloader(…)->Bootloader->Core Configuration->CoreConfiguration_1:

- Is a critical application=checked
- Start the core=checked

3.7 配置 M7 Boot sources:

打开 Bootloader(…)->Bootloader->Boot Sources->BootSources_M7_LightingApp_single_core:

修改名字：Name= BootSources_MCAL_Test。

参考链接文件：

C:\NXP\SW32G_RTD_4.4_4.0.2\eclipse\plugins\Platform_TS_T40D11M40I2R0\build_files\gcc\linker_ram_c0.ld

```
int_sram_c0             : ORIGIN = 0x34000000, LENGTH = 0x00180000 /* 1.5MB */
```

参考编译生成Mapping文件：

C:\NXP\S32G_LLCE_1_0_7_FDK\sample_app_llce\llce_sample_app_af\tmp\.map

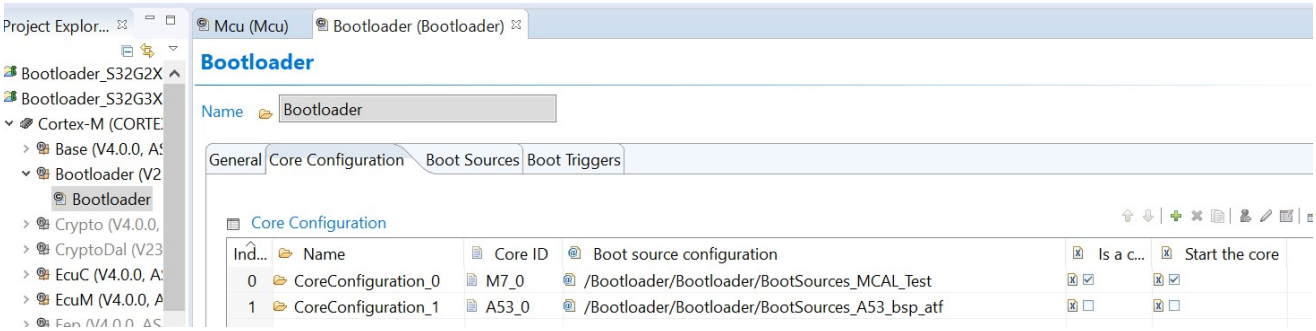
```
startup      0x34000010     0x1b4 C:\cygwin64\tmp\ccOjzPRn.o
             0x34000010             Reset_Handler
             0x34000010             _start
```

bin 文件大小为：1,839,104 Bytes=0x1C1000

得到 bin 文件大小为：1,843,200Bytes=0x1C2000 //已经对齐到 4K，增加一个 4K 大小。

综上所述：

1. BootSources_MCAL_Test->General-> Boot souce=QSPI //保持不变。
2. BootSources_MCAL_Tes->General ->Reset handler address =0x34000010
3. BootSources_MCAL_Tesp->Boot image fragments->ImageFragments_0->Load image at address (RAM)= 0x34000000 //保持不变
4. BootSources_MCAL_Tes->Boot image fragments->ImageFragments_0-> Image size(bytes)= 0x1C2000
5. BootSources_MCAL_Tes->Boot image fragments->ImageFragments_0-> Image CRC value=0x0
6. BootSources_MCAL_Tes->Boot image fragments->ImageFragments_0->SMR Index :0 //如前去掉secure boot部分
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.

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_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\configuration.bat

```
SET TRESOS_DIR=C:/EB/tresos
SET MAKE_DIR=C:/cygwin64
::SET GHS_DIR=
SET GCC_DIR=C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2
SET TOOLCHAIN=gcc
SET CORE=m7
SET SRC_PATH_DRIVERS=C:/NXP/SW32_RTD_4.4_4.0.0/eclipse/plugins ::注意此版本 Bootloader 默认对应原 RTD 版本是 4.4_4.0.0
:: SET SDHC_STACK_PATH=
:: SET SRC_PATH_SAF=
SET TRESOS_WORKSPACE_DIR=C:/EB/tresos/workspace/Bootloader_S32G3XX_ASR_4.4_M7/output
SET HSE_FIRMWARE_DIR=C:/NXP/HSE_FW_S32G3_0_2_16_1
…
```

2. 在 cygwin 中运行:

```
./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：

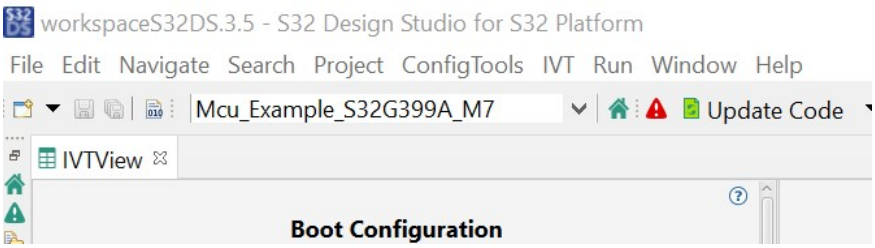
C:\NXP\Integration_Reference_Examples_S32G2_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\bin_bootloader

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

3.10 制造 Bootloader 的带 IVT 的镜像

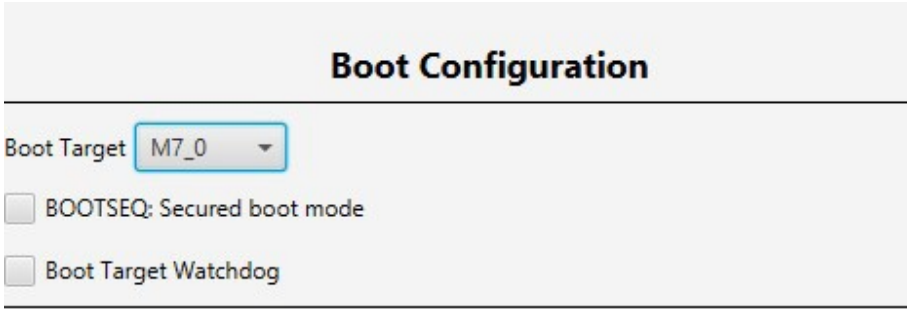
打开32 Design Studio for S32 Platform 3.5(3.5.3)，点击ConfigTools 下拉菜单，选择IVT。（注

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，必须选用 G3 的工程名，才能配置 G3 的 20MB SRAM）：



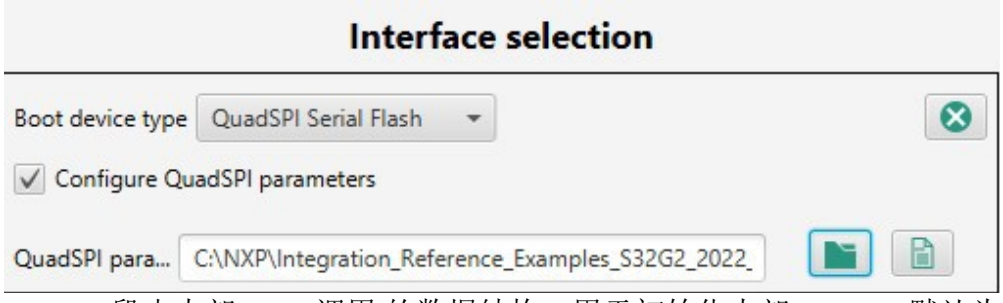
打开IVTView

- 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\S32DS.3.5\eclipse\mcu_data\processors\S32G399A\PlatformSDK_S32XX_4_0_0\quadspi\default_boot_images\mx25_sim200ddr.bin



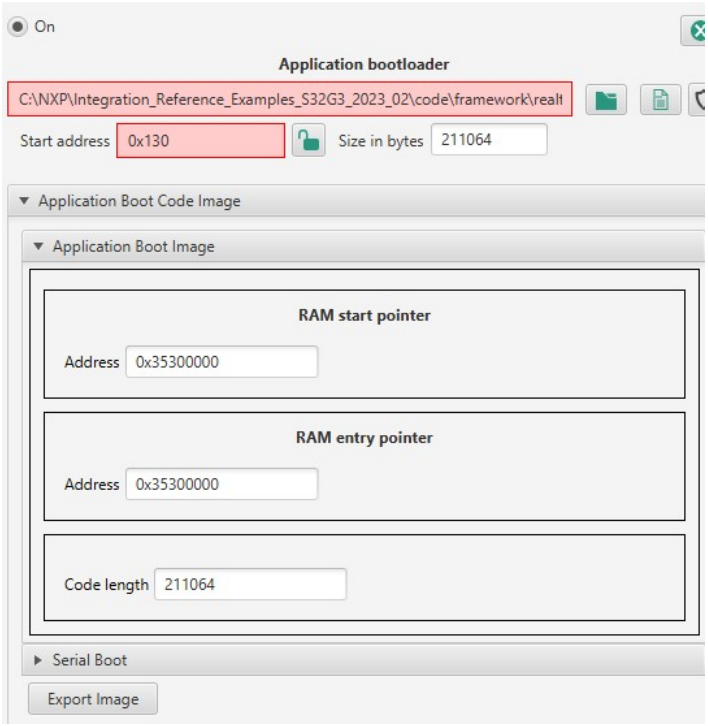
- DCD：DCD 段由内部ROM 调用 的数据结构，用于初始化内部SRAM。默认为On 打开，选择文件为：
C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\res\flash\ S32G3XX_DCD_InitSRAM.bin

- Application bootloader：选择我们编译出来的Bootloader：

C:\NXP\Integration_Reference_Examples_S32G3_2023_02\code\framework\realtime\swc\bootloader\platforms\S32G3XX\build\bin\Bootloader.bin
RAM start pointer 和RAM entry pointer 参考mapping 文件Bootloader.map：

```
.int_sram_no_cacheable
                0x35300000      0x40c
                0x35300000                . = ALIGN (0x4)
                0x35300000                __RAM_NO_CACHEABLE_START = .
                0x35300000                __RAM_INTERRUPT_START = .
 *(.intc_vector)
.intc_vector    0x35300000      0x408 ./bin/Startup_Vector_Table.o
                0x35300000                VTABLE
```

所以都是 0x35300000：

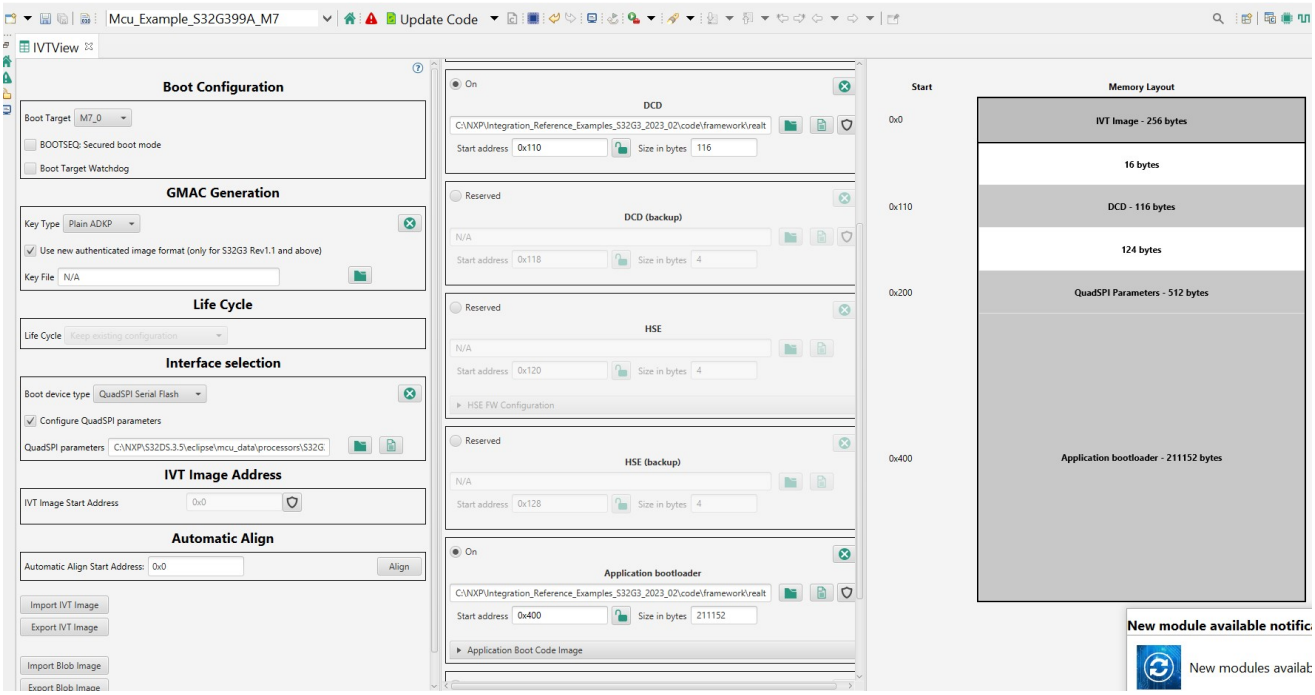


然后在Automatic Align 中，输入0x100，然后点击Align，如果自动Align 成功，会弹出成功提示框，如果失败，手动调节Align 值，再试一下。Align 的要求一般是针对eMMC 这种，QSPI NOR 要求不严格，不过，如果涉及到需要写操作QSPI NOR情况，则应该Align到4KB大小，也就是0x1000。

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RDB3 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.3.5\S32DS\tools\S32FlashTool\GUI\s32ft.exe，Target 选择S32G3xxx，Algorithm 选择MX25UW51245G。

COM 口的port names:中设置为设备管理器中看到的串口：COM11

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-s32g399ardb3.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
```

Image，和 LLCE 相关驱动模块 ko 文件也要拷贝到 SDCARD 中。

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 Linux LLCE logger 功能修改

Linux 镜像采用 BSP38，使用 LLCE CAN Logger 功能

4.1 ATF 的修改

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

注意点：

- 由于 DMA 搬远要求，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
```

- 解决时钟冲突

根据文档《S32G_Supplemental_documentation_on_resolving_clock_conflicts_V*.pdf》，对于G3，需要修改 ATF 代码，注意：

1. 如下函数调用： drivers\nxp\s32\clk\s32gen1_clk.c:

```
s32gen1_get_early_clks_freqs
|-> get_siul2_midr2_freq
return ((mmio_read_32(SIUL2_MIDR2) & SIUL2_MIDR2_FREQ_MASK)
                        >> SIUL2_MIDR2_FREQ_SHIFT);
```

19-16 FREQUENCY	Frequency Identifies maximum core frequency. Qualified by Product Line Letter to provide wider range of frequencies. 0001b - 64 MHz 1011b - 1000 MHz 1100b - 1100 MHz 1110b - 1300 MHz

所以芯片只 qualified 了 1G，1.1G 和 1.3G 的版本，所以对于我们要设定的 1.2G，需要借用 1.3G 的 FREQUENCY ID，如下：

Driver/nxp/s32/clk/s32g3_clk.c

```
//所以 BSP38 没有考虑与 400Mhz M7 时钟同源的问题时，是使用 1300Mhz 频率，所以需要修改回 1200Mhz
#define S32GEN1_A53_1_2_GHZ_FREQ                    (1200 * MHZ) //johnli bootloader
#define S32GEN1_ARM_PLL_VCO_1_2_GHZ_FREQ   (2400 * MHZ)//johnli bootloader
#define S32GEN1_ARM_PLL_PHI0_1_2_GHZ_FREQ   (1200 * MHZ) //johnli bootloader
#define S32GEN1_XBAR_2X_1_2_GHZ_FREQ    (800 * MHZ)


const struct siul2_freq_mapping siul2_clk_freq_map[] = {..
#if 1
SIUL2_FREQ_MAP(SIUL2_MIDR2_FREQ_VAL3, S32GEN1_A53_1_2_GHZ_FREQ,
            S32GEN1_ARM_PLL_VCO_1_2_GHZ_FREQ,
            S32GEN1_ARM_PLL_PHI0_1_2_GHZ_FREQ,
            S32GEN1_XBAR_2X_1_2_GHZ_FREQ),

#else
   SIUL2_FREQ_MAP(SIUL2_MIDR2_FREQ_VAL3, S32GEN1_A53_1_3_GHZ_FREQ,
                        S32GEN1_ARM_PLL_VCO_1_3_GHZ_FREQ,
```

```
                    S32GEN1_ARM_PLL_PHI0_1_3_GHZ_FREQ,
                    S32GEN1_XBAR_2X_1_3_GHZ_FREQ),
#endif
```

其它相关修改见《S32G_Supplemental_documentation_on_resolving_clock_conflicts_V*.pdf》。

使用 Yocto SDK 编译命令如下：

make PLAT=s32g399ardb3 BL33=../u-boot/u-boot-nodtb.bin LDFLAGS=""

Patch 见源代码。

4.2 Linux 中关于 LLCE 配置

根据文档《S32G3_LinuxBSP_38.0_User_Manual.pdf》章节 12.4 CAN Logger 说明：

12.4 CAN Logger

The LLCE CAN Logger driver receives CAN frames and timestamps from LLCE hardware and logs them.
The driver is able to operate in two modes: along with Linux LLCE CAN driver or in scenarios where the LLCE CAN host management is part of an another software stack.
For the cases when the LLCE subsystem is managed by Linux, the following lines should be added in <builddirectory>/conf/local.conf file.

```
DISTRO_FEATURES:append = " llce-fw-load llce-can llce-can-logger"
NXP_FIRMWARE_LOCAL_DIR = "/path/to/firmware/binaries/folder"
```

For the cases when the LLCE subsystem is managed by other software stacks, the added line should be:

```
DISTRO_FEATURES:append = " llce-can-logger"
```

所以如果是第一次使用Yocto编译镜像，只需要在<builddirectory>/conf/local.conf中增加logger的DISTRO_FEATURES即可，不需要增加llce-can, llce-can-multihost and llce-linflex。然后编译出来的LLCE驱动模块的自动加载方法为：

/etc/modprobe.d/

llce_core.conf：

```
options llce-core load_fw=0
```

llce-mailbox.conf：

```
options llce-mailbox config_platform=0
```

如果需要调试驱动修改为手动加载模式，则将llce_core/mailbox.conf文件去掉，重启后用以下命令手动加载模块：

```
insmod llce-mailbox.ko config_platform=0 //不再做 LLCE 平台初始化
insmod llce-core.ko load_fw=0  //不再加载 FW。
insmod llce_logger.ko
```

关于 LLCE 的手动编译与镜像安装方法，参考文档《S32G_LLCECAN_Linux_*.pdf》。

/linux/.config 配置如下：

```
CONFIG_CAN_LLCE=y
CONFIG_CAN_LLCE_CORE=y
```

```
# CONFIG_CAN_LLCE_CONTROLLER is not set
CONFIG_CAN_LLCE_LOGGER=m
CONFIG_LLCE_CORE=m
CONFIG_NXP_LLCE_MBOX=m
```

编译生成：

```
LD [M]  drivers/mailbox/llce-mailbox.ko
LD [M]  drivers/mfd/llce-core.ko
LD [M]  drivers/net/can/llce/llce_logger.ko
```

将 ATF，Image 更新，将 LLCE*.ko 拷贝到板上。

4.3 LLCE 相关初始化冲突说明

注意 LLCE 中断初始化在 MCAL 一端，所以理论上如果 Linux 端只初始化了 logger 功能，则不应当注册其它 LLCE 中断，不然 M core/A core 同时响应同一中断，可能会导致资源冲突。

当我们使用：

```
insmod llce-mailbox.ko config_platform=0
```

加载驱动时，是希望此参数可以禁止 Linux 端的 LLCE 相关初始化，而实际上在启动后：

```
root@s32g399ardb3:/home# cat /proc/interrupts
95:      1429       0     11518       0       0       0       0       0   GICv3 206 Level     rxin_fifo_0_7
96:         0       0         0       0       0       0       0       0   GICv3 207 Level     rxin_fifo_8_15
101:  3818088       0     11876       0       0       0       0       0   GICv3 216 Level     logger_rx
```

可以看出来 rxin_fifo 初始化没有被去掉，从而会导致与 M core 中断竞争资源。所以做如下修改：

Drivers\mailbox\llce-mailbox.c

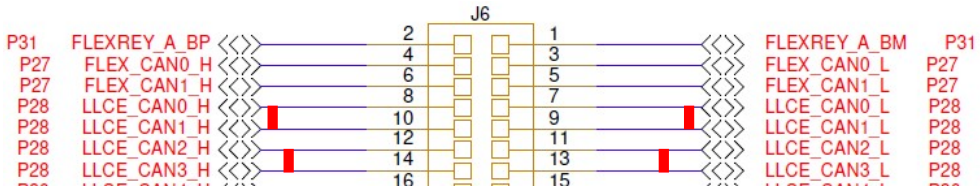
```
static int llce_hif_startup(struct mbox_chan *chan)
{
    struct llce_chan_priv *priv = chan->con_priv;
    struct llce_mb *mb = priv->mb;
    if (!config_platform) //add this
                                return 0;

    request_llce_pair_irq(mb, &mb->rxin_irqs);

    return 0;
```

}

5 测试

5.1 硬件连接

- 设置RDB3 为正常启动模式，从QSPI NOR 上启动：SW10-1=ON, SW10-2=OFF,SW4 all=OFF。
- 将J6接口上的LLCE CAN0/1连接，CAN2/3连接，注意是HtoH, LtoL。



- 连接电源，Linux调试串口到Uart0 J2。开关上电。

5.2 LLCE logger 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 终端正常启动的打印在UART0 终端中，则说明Bootloader Boot A 核成功：

然后

```
root@s32g399ardb3:/home# pwd
/home
root@s32g399ardb3:/home# ls
llce-core.ko  llce-mailbox.ko  llce_logger.ko  root
insmod llce-mailbox.ko config_platform=0
insmod llce-core.ko load_fw=0
insmod llce_logger.ko
ip link set up llcelogger0 type vcan
candump -t a -H llcelogger0,0:0,#FFFFFFFF
(1122556976.000000)  llcelogger0  029  [64]  16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15
…
cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
94:          0          0          0          0          0          0          0          0   GICv3 204 Level     llce_mb
101:       1657          0      11883          0          0          0          0          0   GICv3 216 Level     logger_rx
```