S32G Bootloader Customization

by John Li (nxa08200)

This article explains how to customize the Bootloader on the S32G2 RDB2 board. The main functions are:

- Bootloader starts a M core, MCAL driver test program: This article tests the MCAL driver sample code of MCU, DIO and UART respectively.
- Bootloader starts A53 Linux at the same time.

History	Comments	Author
V1	● Create the doc	John.Li
V2	● Tanslate to English	John.Li

Contents

1 Required software and tools

1.1 Software and Tools

Software and Tools	Name	Comments
Development board	S32G-VNP_RDB2	S32G2 RDB2 Board
Configuration and Programming tools	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 project：	S32G2 Platform Software Integration 2022.06	Include bootloader project
AutoSar MCAL	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 Reference Docs

- <<Hands-on S32G2 Multicore application enablement example.pdf>>:Wang Xuewei. This article mainly Refer to this article.
- <<S32G_RTD_MCAL_Vxxxx.pdf>>:JohnLi.

https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-MCAL-customization-application-doc/ta-p/1399899, MCAL customization instructions. Note that the original text is for version 2.0.0_hf04 and needs to be updated to 3.0.2.

- <<S32G_Kernel_BSP32_V4-20220513.pdf>>: JohnLi.

https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-Linux-BSP-customizationdoc/ta-p/1399902, Linux customization.

1.3 Development Instructions

The main difficulty of Bootloader project is to deal with M-core Bootloader and M-core autosar system (this doc uses MCAL for drive example code) resource conflicts, and also between the M core and the A core, these so-called resources include:

- SRAM space allocation and cooperation with MPU.
- CLOCK initialization conflict.

- Conflicts related to MCU configuration.
- Conflict of PORT configuration.
- Interrupt configuration conflicts.

  and many more.

  The general principles of conflict resolution are:
- If there is a conflict between Bootloader and MCAL, try to use Bootloader to initialize and remove the non-local driver of MCAL. initialization and configuration. So the bootloader is responsible for the initialization of clock, mcu configuration and port configuration, MCAL driver In the dynamic test example, it needs to be removed to avoid conflicts.
- M core conflicts with A core, use M core to initialize the clock of the core and its own peripherals, PORT and interrupt initialization, When it comes to the clock of the A core, try to be consistent with the settings in Linux, so that the A core will not be set again. A kernal is responsible for A Kernal institute Peripheral clocks, interrupts and PORT initialization.

Disclaimer: Bootloader itself is not required by the Autosar specification, so there is no quality assurance statement, and this article only provides some revision guidance, no quality assurance, please note.

2 Test software installation and compilation instructions

- Cygwin install guide please refer <<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 install guide please refer <<S32 Design Studio v3.4.1 及RTD 2.0.0 HF4 安装指南_v20210810.pdf>>: Chinese version. https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-Design-studio-amp-RTD-SDK-install-guide/ta-p/1399909.

Note Please install the software version required by this article.

2.1 Install RTD MCAL Driver

Double-click SW32G_RTD_4.4_3.0.2_D2203.exe to install. The installation wizard will ask for the path to EB Tresos Studio (C:\EB\tresos\).

After the RTD is installed correctly, you will see the source code and related documents in the installation path (default: C:\NXP\ SW32G_RTD_4.4_3.0.2). And the .link file will be seen in the links folder under the corresponding EB Tresos Studio installation path, which indicates that EB Tresos Studio has been associated with RTD. After opening EB Tresos Studio to create a new or import RTD configuration project, it will be associated with the RTD path. If the EB path is not provided during installation, you can also manually create a new file with the suffix .link in the corresponding path of EB, and write it into the RTD installation path.

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

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

Note that the other *.link files should be modified to backup names, such as SW32G2_RTD_4.4_2.0.0_HF04.bak, to prevent conflicts. For details of MCAL, please refer to the document <<S32G_RTD_MCAL_V*.pdf>>, Chinese verison.

2.2 Compile MCAL driver test program (take MCU as an example)

Firstly, take the simplest MCU MCAL driver as an example. Open the MCU MCAL driver test program example according to the instructions in the document <<S32G_RTD_MCAL_V*.pdf>>, Chinese verison. Right-click on the project Mcu_Example_S32G274A_M7, select generate project, and use EB to generate the code. And then copy
C:\EB\tresos\generate* to
C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\generate*

Modify the related compiling file:
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 # compiler location
#The path to the EB Tresos installation dir
TRESOS_DIR = C:/EB/tresos #EB install location
#The path to the T32 installation dir
T32_DIR = C:/T32 #Trace32 install location
…
```

Modify as follows:
C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\Makefile：output*.bin file。

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

The command to compile the MCU MCAL driver example program in Cygwin is as follows:
C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7

```
$ make build
$ make
```

Generate the binary at
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 Optimizition and rearrangement of M7 demo image and its coordination with MPU settings

The current MCU MCAL image link file Mapping is:
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 */

The bootloader's SRAM binary mapping is: 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 */b
int_sram_stack (rwx) : ORIGIN = 0x347EF000, LENGTH = 0x1000 /* 8K - if changed, sync with sys_init */
}

Linux Bootloader：fip.s32’s binary magging is：

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

So it can be seen that the SRAM image loading and running addresses of M7 and fip are conflicting, so we need to move the M7 image. Also note that the MPU configuration is not turned on by default in the Bootloader project, while the MCAL driver sample code is turned on by default. So check its MPU configuration as follows: 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};
```

So move the M7 image to an address above 4M, but it cannot be larger than 6M (above 7M is the Bootloader address space), and also note that the link address such as the interrupt handler needs to be in the no_cacheable address, so we also need this address of 0x34500000, (this will cause the exported *.bin file to be relatively large, with a continuous empty space in the middle, but for end customers, the entire autosar system is used, so this article no longer considers rearrangement to shrink the image):

```
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 */
```

After recompiling, it is converted into a *.bin file, and the file size has changed from the original 5.M to 1.25MB. (Actual size is only 256KB). The size of the bin file is obtained as: 1,310,720 Bytes=0x140000. Then the map file main.map looks like this:

```
.sram 0x34400000 0x1e21c //load address
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 //execute address

0x34400010 _start
```

Notes:

When BL2 is running, BL2 will continue to load INFO: Loading image id=0 at address 0x34400000 // #define FIP_IMAGE_ID U(0 FIP_IMAGE_ID will be placed at 0x34400000 - 0x34400200, which will conflict with M7 by 0x200 bytes. It should be because the M core runs faster, so there may be problems when bl2 is loaded, but the M core has already run through the first part. It is best to move it, but after

changing it, the settings of the MPU should be changed in the same way. Note that the MPU has alignment requirements. In this doc, we do not handle this conflict because no test fail, but suggest to fix it.

2.4 Remove CLOCK INIT

The clock initial needs to be removed for the following reasons:

1. The Bootloader has already configured the clock, so the MCAL driver configuration may conflict again.

2. The MCAL driver sample itself initializes the clock to run the lauterbach script alone. When integrated in autosar In the system, it is recommended to use the bootloader to initialize the clock.

- Can_llc-pfe-th->ECU(…)->Mcu(…)->Mcu->General->Mcu Init Clock API=unchecked. After that, the deinitialization of the clock needs to be removed in the Bootloader. Then in the source code: C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\src\main.c, remove the clock initial codes:

```
#if 0
Mcu_InitClock(McuClockSettingConfig_0);
while ( MCU_PLL_LOCKED != Mcu_GetPllStatus() )
{
/* Busy wait until the System PLL is locked */
}
Mcu_DistributePllClock();
#endif
```

Or in Can_llc-pfe-th->ECU(…)->Mcu(…)->Mcu->McuClockSettingConfig, remove McuClockSettingConfig_0.

2.5 Remove MCU related INIT

In order to avoid the conflict between Bootloader and MCAL driver MCU mode settings, initialize the MCU mode settings: (this MCU mode initialization will be set here, it will restart the partition, so it needs to be removed):

Then in the source code: C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\src\main.c, remove the initial codes:

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

Or in Can_llc-pfe-th->ECU(…)->Mcu(…)->Mcu->McuModeSettingConfig, removed McuModeSettingConfig_0.

In addition, RamSectorConf is also called in MCU_init to initialize the RAM, and we have already initialized it with DCD before in bootloader. So you can also comment out MCU_init, in this case, the main function of MCU is equivalent to no code call.

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

2.6 DIO MCAL program remove PORT INIT.

Use EB to open project: C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Dio_ToggleLed_S32G274A_M7\TresosProject。

Related compilation, memory movement and MPU settings, CLOCK removes INIT, MCU removes INIT, and MCU MCAL driver same.

The source code is modified as follows:

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
…
```

The MCU MCAL driver mainly includes the initialization of the clock and MCU mode. After all of them are removed, there is basically no code. In this section Take the DIO lighting MCAL example as an example to explain the resolution of the PORT conflict. There are basically two ways:

1: Bootloader is responsible for the whole Initialization of the external PORT.

2: Bootloader only initializes related drivers, other PINs should be set to their untouchedPortPin. Each MCAL driver is then responsible for its own initialization. In practice, the first method is mostly used, so this article also uses the first method:

Dio_ToggleLed…->someid->Port, right click and choose disabled. Generate the codes again:

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" remove port.h file
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 #remove Port, do not compile it
```

Then add the Port configuration of this GPIO to the Bootloader project.

2.7 UART MCAL program remove PORT INIT

Use EB to open project C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Dio_TS_T40D11M30I2R0\examples\EBT\Uart_TS_T40D11M30I2R0\TresosProject。

Related compilation, memory movement and MPU settings, CLOCK removes INIT, MCU mode removes INIT, and MCU MCAL drive move the same. And the way to disable Port in EB is the same as the previous section.

Since DIO does not require clock initialization, the UART MCAL project is used to illustrate the removal of CLOCK/MCU MODE/PORT case. The source code is modified as follows:
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 program modification CLOCK TREE

Refer to Section 3.5, in order to unify the UART CLOCK TREE of Bootloader/UART MCAL/Linux, it needs to be modified CLOCK TREE of UART:
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)* automatically calculated as 1.0E8。
- PLL_PHI1 Frequency (Calculated) (dynamic range) automatically calculated as 8.0E7。
- PLL_PHI3 Frequency (Calculated) (dynamic range)* automatically calculated as 1.25E8 。
- PLL_VCO Frequency (Calculated) (dynamic range)* automatically calculated as 2.0E9。

…McuClockSettingConfig_0-> McuCgm0ClockMux8：

- CGM0 Clock Mux8 Source= PERIPH_PLL_PHI3_CLK。
- Clock Mux8 Frequency (LIN_BAUD_CLK) (dynamic range) automatically calculated as 1.25E8 。

…McuClockSettingConfig_0->McuClockReferencePoint->UART_CLK

- Mcu Clock Reference Point Frequency (0 -> 5000000000)* automatically calculated as 1.25E8 。

Then check：Uart_example…->someid…->Uart…->Uart->UartChannel-> UartChannel_1

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

Note that the default baud rate is: 9600. In order to match the raised root clock, it is modified to:

- DesireBaudrate = LINFLEXD_UART_BAUDRATE_115200
- Uart Parity Enable= unchecked。So in fact, the default configuration is configured as no parity check, pay attention to the corresponding configuration on the PC side.

Also note that the default M7 root clock of the UART mirror is 48Mhz FIRC. This article is in order to match the configuration of the Bootloader and Linux. Set it to 800Mhz, so the loop rate of the CPU is theoretically 9 times faster, so:

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

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

Mcu Loops TimeOut (1 -> 4294967295)*= 100000 //it was 10000， There is no need to modify it here, because the init of the MCU has been commented out. (The above modification is only required when Uart Timeout Method=OSIF_COUNTER_DUMMY is used when using the bare code of the MCAL driver example.)

at last, modify:

C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Uart_TS_T40D11M30I2R0\examples\EBT\Uart_Example_S32G274A_M7\src\main.c, Remove the redundant test code and keep only one asynchronous send codes:

```
…
/* 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 Resolving Interrupt Conflicts

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

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

The above code will initialize the interrupt, in which Platform_Init will register the peripheral

interrupt of the A core and the M core, and the registered interrupt handler The number is empty, and Platform_InstallIrqHandler sets the interrupt handling function and optimization level of UART1, which is found in practical applications. Will affect the interrupt of the A core. So the following code simplifies the registration process of the interrupt to UART1:

```
#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
```

Avoid effects on A kernal drive.

2.10 Prepare A53 Linux image

Prepared the A53 SDcard image according to the document <<S32G_Kernel_BSP32_V4-20220513.pdf>>, Chinese version.

Check points：

- Due to the requirement of DMA moving away, the alignment is 64 Byte when without CRC, it is necessary to modify the ATF configuration

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

Compiling output as follows:

```
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
```

Since FIP has been modified, if you use the default demo image, you need to update FIP.bin/.S32.

- Since the DIO and UART1 MCAL programs of MCAL are planned to be tested, the serial port 1 and DIO of the Linux The IOMUX of GPIO_LED is removed accordingly: Currently, BSP32 does not configure the IOMUX and initialization of UART1. So no modification is needed, and the GPIO pins of DIO conflict with SPI1. SPI1 needs to be set in DTS of Uboot and kernel The driver

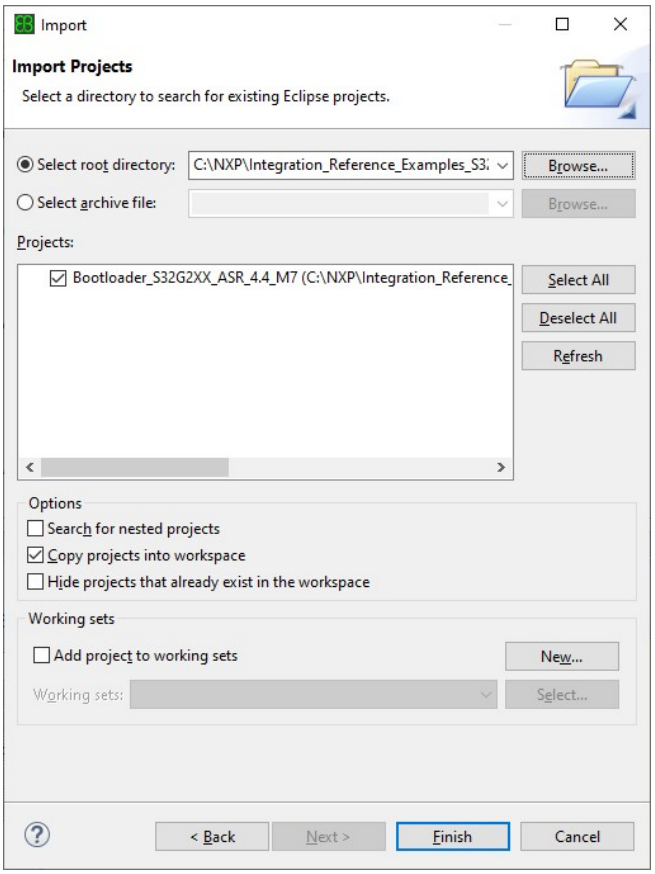
is disabled. This article is no longer described, please refer to the Linux customization documentation.

3 Bootloader Project instructions

Run Platform_Software_Integration_S32G2_2022_06.exe to install bootloader project, then copy C:\NXP\Integration_Reference_Examples_S32G2_2022_06\applications\realtime\Tresos\eclipse\plugins all plugins to C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins.

Then openEB 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

So to open project Bootloader_S32G2XX_ASR_4.4_M7. (checked Copy projects into workspace)



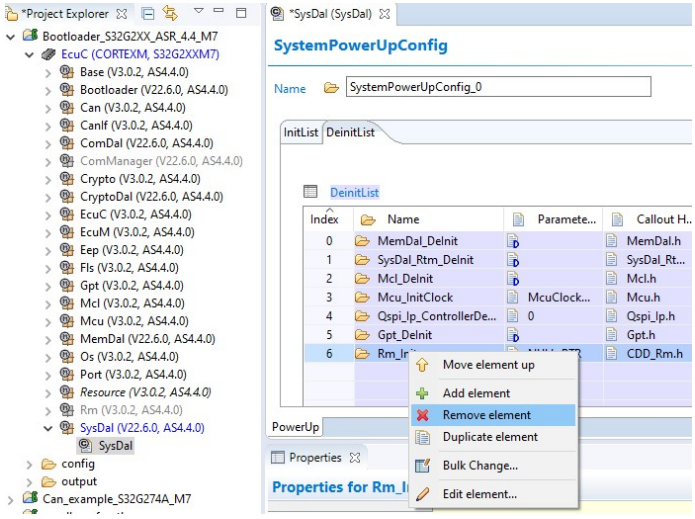
Then double-click Bootloader_S32G2XX_ASR_4.4_M7->EcuC…, you can open all modules (if any module fails to load, check whether the platform plugin was copied to mcal before, and make sure that C:\EB\tresos\links only has SW32G_RTD_4.4_3. 0.2.link connection).

3.1 Shut down XRDC support

To simplify the project, first remove the XRDC support:

1. Right-click on the Rm(V3.0.2, AS4.4.0) module and select Disable to turn off this module.

2. Select SysDal(V22.6.0,AS4.4.0)->SysDal->PowerUP->SystemPowerUpConfig_0->DeinitList: Remove Rm_Init:



3.2 Shut down eMMC/SD support(optional)

Since the images in this sample project are placed in QSPI NOR, eMMC support is not required, and can be turned off as follows:

1. Bootloader(…)->EcuC(…): Disable the Eep and MemDal modules.

2. Bootloader(…)->EcuC(…)->SysDal (…)->SysDal->Powerup->SystemPowerUpConfig_0: Delete the init of MemDal from the InitList. Delete the deinit of MemDal from the DeinitList.

3. Remove SDHC in the build configuration as described later.

3.3 Shut down secure boot(optional)

This project does not consider secure boot, so it can be removed as follows:

1. Bootloader(…)->EcuC(…): Disabled CryptoDal and Crypto modules.

2. Bootloader(…)->EcuC(…)->Bootloader(…)◊ Bootloader->General: Enable Secure Boot= unchecked.

3. Bootloader(…)->EcuC(…)->SysDal (…)->SysDal->Powerup->SystemPowerUpConfig_0: Delete the init of SysDal_Init from the InitList.

4. Remove secure boot from the build configuration as described later.

3.4 Increase the initialization of the PORT required by the MCAL driver

First of all, since SDHC support is removed from this Bootloader project, the corresponding pins should be removed: Bootloader…->EcuC(…)->Port(…)-> Port->PortContainer-> PortContainer_0->PortPin:

Delete all USDHC related pins.

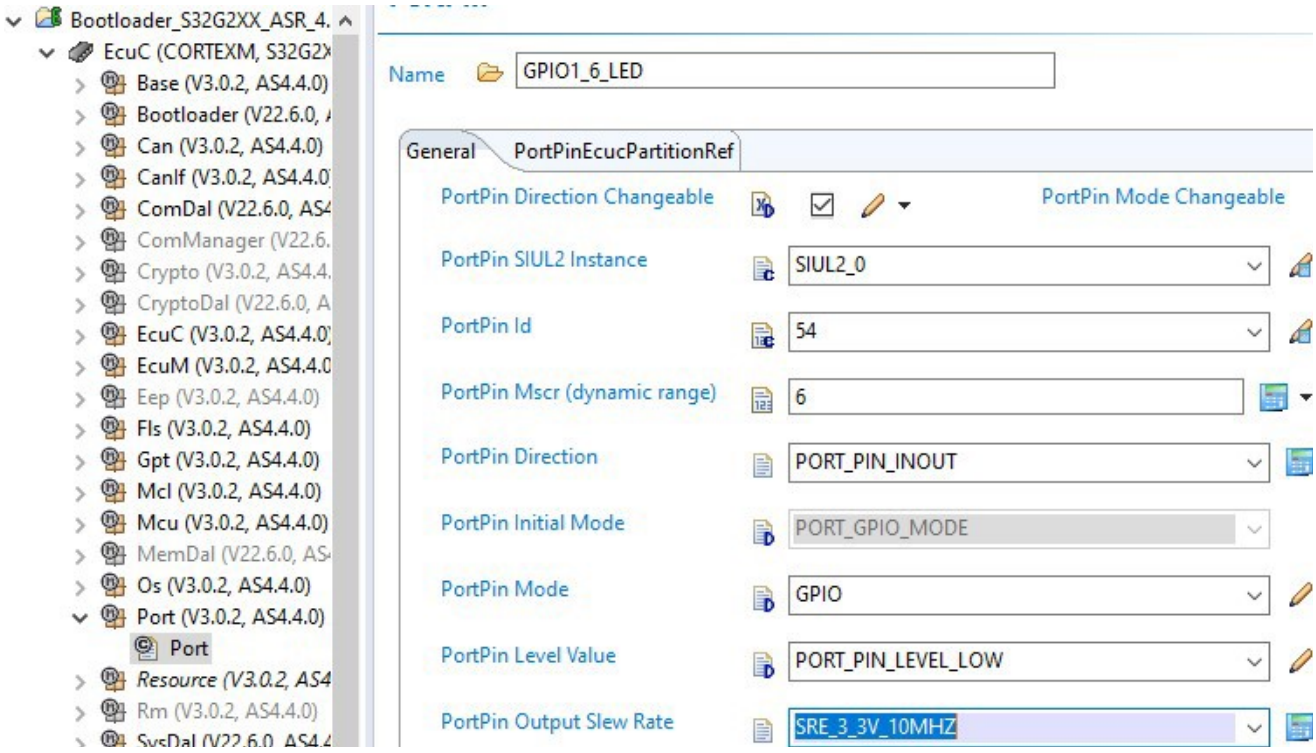
Then in General: PortNumberOfPortPins* is automatically calculated as 40. After that, the Port IDs of other pins must be Rearrangements are calculated automatically.

- DIO MCAL driver used GPIO：

Bootloader…->EcuC(…)->Port(…) -> Port->PortContainer-> PortContainer_0->General : PortNumberOfPortPins+1=41

PortPin : Click on the upper right corner + to add a Port: Double-click to enter:

1. Change name to GPIO1_6_LED
2. PortPin Id=40// automatic sorting
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

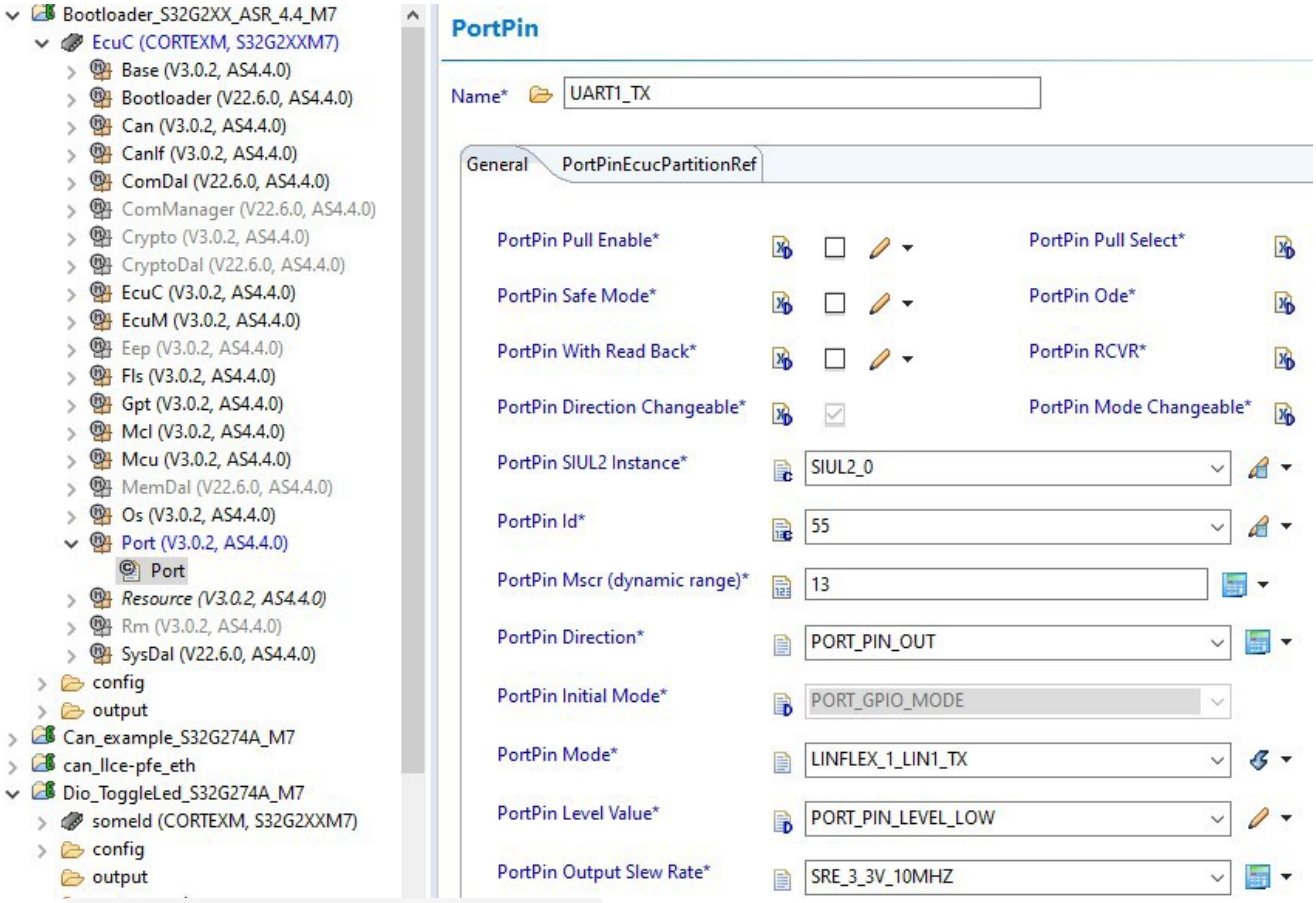


- UART MCAL driver used UART1_RX/TX

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

PortPin :Click on the upper right corner + to add a Port: Double-click to enter:

1. Change name to UART1_TX
2. PortPin Id=41 // automatic sorting
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



The settings for RX are the same as above:

1. Change name to UART1_RX
2. PortPin Id=42 // automatic sorting
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 Solve the clock conflict between Bootloader, MCAL and Linux

Take the UART MCAL driver example as an example: the core principles are:

1. The clock configuration only retains the initialization of the Bootloader, the deinitialization of the Bootloader and the initialization code in the Mcal code code removed.

2. The initialization of the Bootloader should take into account the final clock configuration required by the M core clock and the Mcal driver, and consider the A core clock The correct source and value of the clock (can be configured not to be controlled by the MCU code).

So: first in Sysdal->powerup->systempowerupconfig_0->DinitList:

- Set this item to Mcu_InitClock; McuClockSettingsDisablePLL; Mcu.h. delete. thereby removing the Bootloader Deinitialization in .

Then: modify the initialized value, taking into account the M core, peripherals and A core clock:

	Bootloader: clocksettingconfig	UART:MCU clocksettingconfig	Linux	Bootloader: clocksettingconfig
	Config_0(initialization)	Config_0 Config_		Config_0(initialization) How to modify:
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		keep it unmodified
McuFXOSC	4.0E7	4.0E7	4.0E7	keep it unmodified
McuCgm0Pcs	PCFS_4:8.0E8	PCFS_4:0		keep it unmodified
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:(This is M kernel clock, keep it unmodified 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	keep it unmodified =closed
Cgm0Mux2	Source: FXOSC_CLK CLKOUT0:0	Source: FXOSC_CLK CLKOUT0:2.0E7	Source: FXOSC_CLK CLKOUT0:0	keep it unmodified = closed
Cgm0Mux3	Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7	Source: PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7		Source: keep it unmodified PERIPH_PLL_PHI1_CLK PER_CLK: 8.0E7
Cgm0Mux4	Source: PERIPH_PLL_PHI1_CLK FTM_0_REF_CLK: 0	Source: PERIPH_PLL_PHI1_C LK FTM_0_REF_CLK:	Source: PERIPH_PLL_PHI1_C LK FTM_0_REF_CLK:	Source: keep it unmodified， UART MCAL driver do not use it

		5E6	8E7	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: keep it unmodified，UART MCAL driver do not use it 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: keep it unmodified 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: keep it unmodified 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: (modified to the same with Linux clock source and value, it is key settings of UART MCAL driver) 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: modified to not be MCU control 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: modified to not be MCU control PERIPH_PLL_PHI5_CLK GMAC_0_TX_CLK: 0
Cgm0Mux11	Source: GMAC_0_EXT_RX_CLK GMAC_0_RX_CLK: 1.25E8	Source: FIRC_CLK GMAC_0_RX_CLK: 4.8E7	Source: FIRC_CLK GMAC_0_RX_CLK: 0	Source: modified to not be MCU control 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: modified to not be MCU control 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: modified to not be MCU control 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: keep it unmodified FIRC_CLK GMAC_0_REF_CLK: 4.8E7
Cgm0Mux16	Source: FIRC_CLK SPI_CLK: 4.8E7	Source: FIRC_CLK SPI_CLK: 4.8E7		Source: keep it unmodified FIRC_CLK SPI_CLK: 4.8E7
Cgm1Mux0	Source: CORE_PLL_PHI0_CLK A53_CORE_CLK:8.0E8	Source: FIRC_CLK A53_CORE_CLK:4.8E 7	Source: CORE_PLL_PHI0_CLK A53_CORE_CLK:1.0E 9	Source: This is A53 clock, Modified to be the same as Linux source and value CORE_PLL_PHI0_CLK A53_CORE_CLK:1.0E 9
Cgm1Pcs	PCFS_4: 8.0E8	PCFS_4: 0		keep it unmodified =closed
Cgm2Pcs	PCFS_33: 0	PCFS_33: 0		keep it unmodified = closed
Cgm2Mux0	Source: FIRC_CLK PFE_PE_CLK, ACCEL_3_CLK:0	Source: FIRC_CLK PFE_PE_CLK, ACCEL_3_CLK:0		Source: keep it unmodified，modified to not be MCU control 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_DI V_CLK, ACCEL_4_CLK:0	Source: FIRC_CLK PFE_MAC_0_TX_DI V_CLK, ACCEL_4_CLK:0	Source: keep it unmodified，modified to not be MCU control FIRC_CLK PFE_MAC_0_TX_DIV_CLK, ACCEL_4_CLK:0
Mcu	Source:	Source:	Source:	Source: keep it

GENCTRL1	PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0	PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0	PFEMAC0_TX_DIV_CLK PFE_MAC_0_TX_CLK :0	unmodified，modified to not be MCU control PFEMAC0_TX_DIV_CLK PFE
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: keep it unmodified，modified to not be MCU control 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: keep it unmodified，modified to not be MCU control 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: keep it unmodified，modified to not be MCU control 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: keep it unmodified，modified to not be MCU control 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: keep it unmodified，modified to not be MCU control FIRC_CLK PFE_MAC_2_RX_CLK, APEXD_0_CLK： 4.8E7
Cgm2Mux7	Source: FIRC_CLK	Source: FIRC_CLK		Source: keep it

	PFEMAC0_REF_DIV_CLK：0	PFEMAC0_REF_DIV_CLK：0		unmodified, modified to not be MCU control 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: keep it unmodified, modified to not be MCU control 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: keep it unmodified, modified to not be MCU control 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: keep it unmodified, modified to not be MCU control ,linux control itself DDR clock initial FIRC_CLK DDR_CLK: 4.8E7
McuRtc ClockSelect	Source: FIRC_CLK 4.8E7	Source: FIRC_CLK 4.8E7		Source: keep it unmodified 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: Modified to Linux Required source and value 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 kernal root clock, so keep it 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_PHI5:1.25E8 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: 1.25E8 PLL_PHI4: 2.0E8 PLL_PHI5:1.25E8 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: 1.25E8// root of UART clock, so configured with Same as Linux PLL_PHI4:0 PLL_PHI5:1.25E8 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: 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	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	modified to not be MCU control
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	modified to not be MCU control
McuClk Monitor	27 项	27 项		keep it unmodified
McuClock Reference	Name: A53_CORE_CLK:8.0E8 CAN_CLK:4.0E7 CM7_CLK= XBAR_CLK:4.0E8 PIT_CLK= XBAR_DIV3_CLK:1.3E8 uSDHC=SDHC_CLK：4.8E8	Name: UART_CLK= LIN_BAUD_CLK=4. 8E7		Name: A53_CORE_CLK:1.0E9 CAN_CLK:4.0E7 CM7_CLK= XBAR_CLK:4.0E8 PIT_CLK= XBAR_DIV3_CLK:1.3E8 UART_CLK=LIN_BAUD_CLK=1.25E8//add this item

Illustrate:

1. Since BSP32, the version of clk that uses ATF by default is temporarily incomplete, so the Linux clock tree, to use the following command to export on BSP30:

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.The so-called " keep it unmodified " means no modification, and "not controlled by MCU" means to change the item:” MCU under this item control=unchecked. That is, the M core code does not initialize this clock. So apart from the UART clock, most of the Its peripherals can be modified to "not controlled by MCU".

3. Core CLK tree as follows:

23.1.2.1 Core-related clock overview

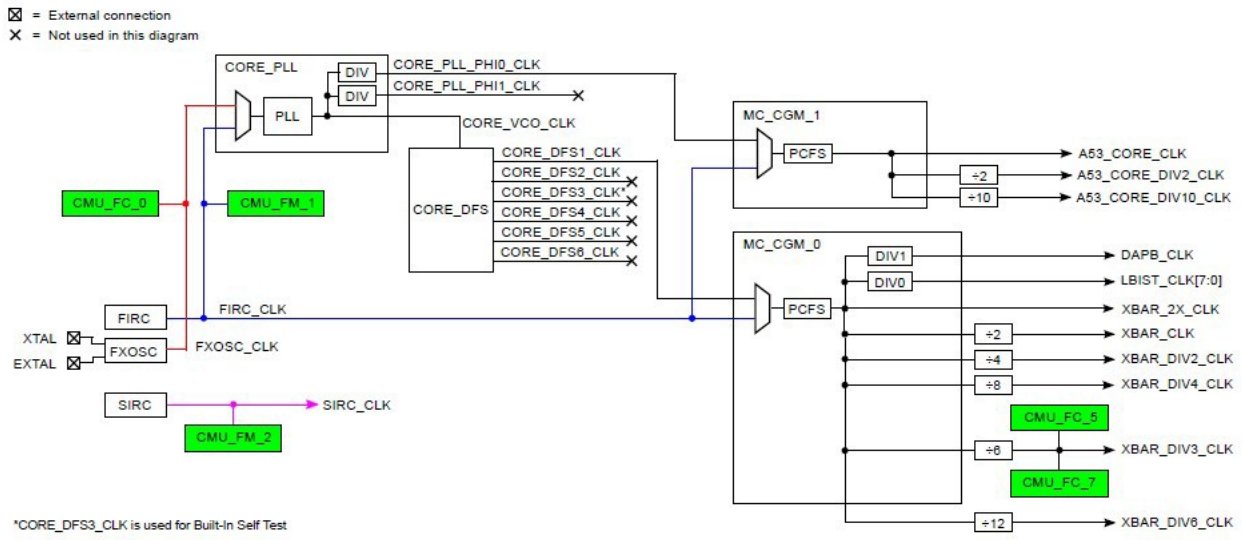


Figure 49. Core-related clock overview

23.7.11.1 Cortex-A53 cluster clocking

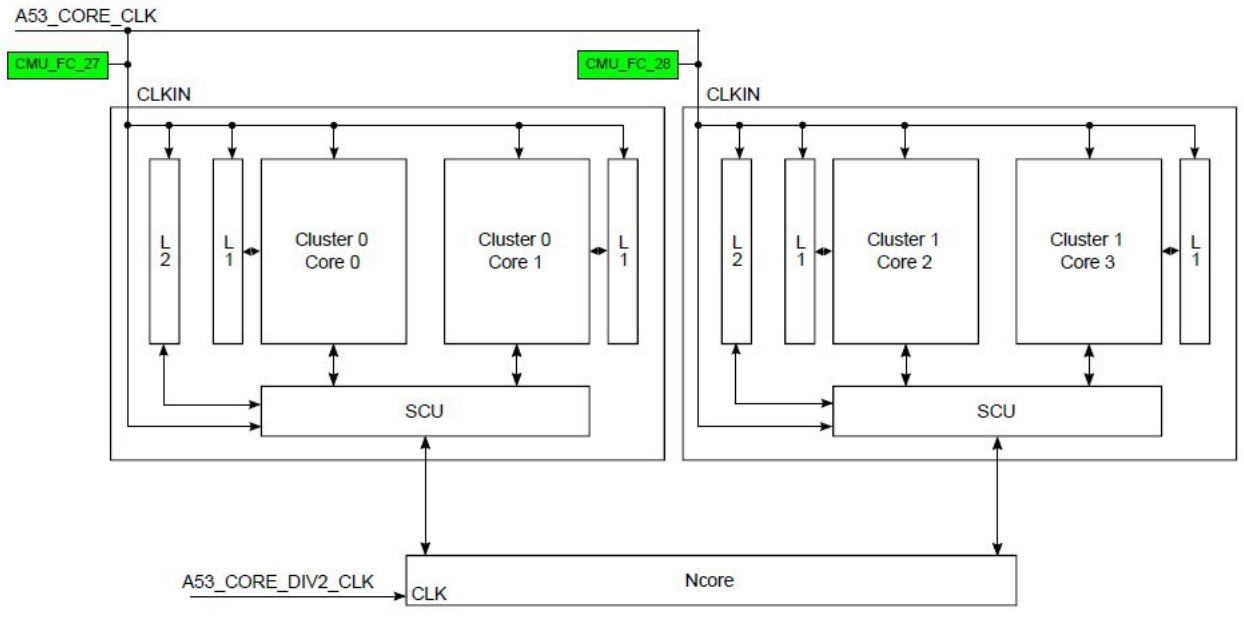


Figure 114. Cortex-A53 cluster clocking

23.7.11.2 Cortex-M7 clocking

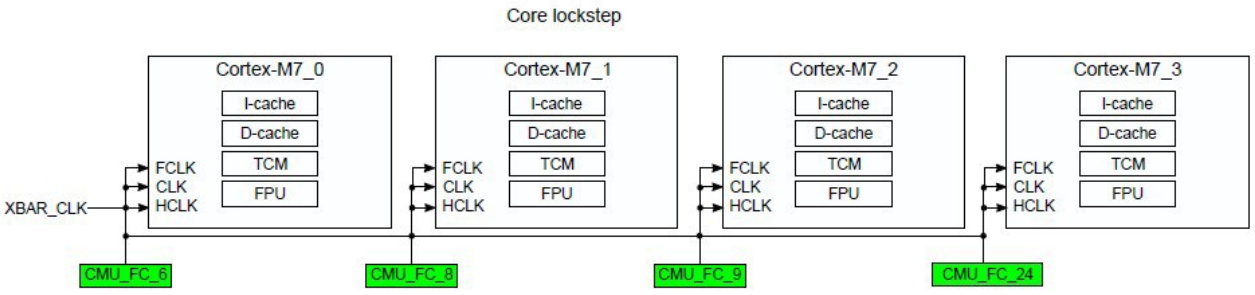


Figure 115. Cortex-M7 clocking

So:

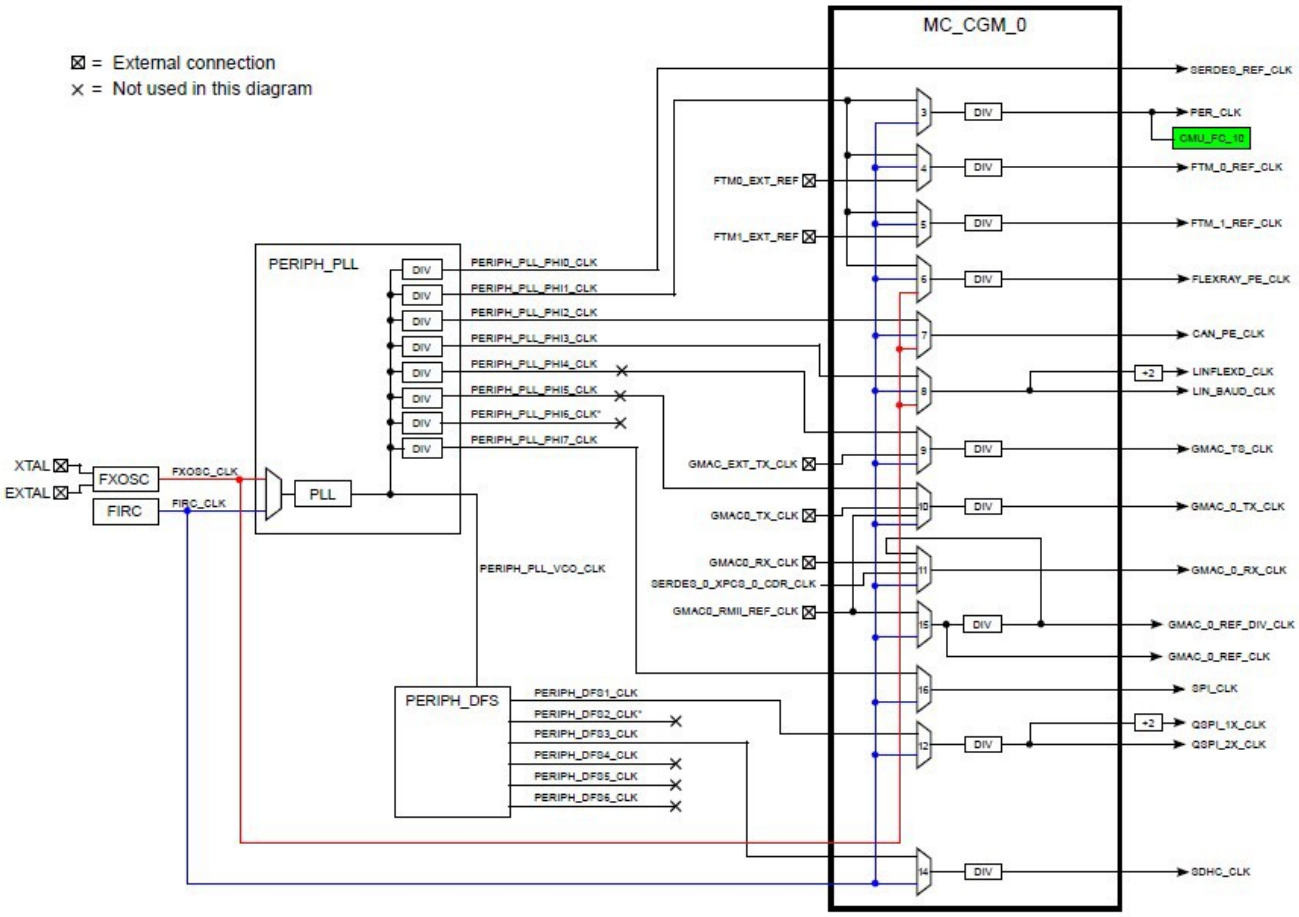
- …McuClockSettingConfig_0->McuCorePll:

RDIV=2; MFD (1 -> 255)=50; PHI0 Division value (0 -> 255) , then:
PLL_PHI0 Frequency (Calculated) (dynamic range)= 1.0E9 //
The clock of A53 is 1G, modified to be compatible with Linux same.
PLL_VCO Frequency (Calculated) (dynamic range)= 2.0E9

- …McuClockSettingConfig_0->McuCoreDFS: McuDfs_1

DFS1 MFI (1 -> 255)=1; DFS1 MFN (0 -> 35) =9,then:
DFS1_CLK Frequency (Calculated) (dynamic range)= 8.0E8 // Then the M core clock is 400Mh.

4. The peripheral CLK tree is as follows:



So the source of LIN_BAUD CLOCK is:
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. then:
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) automatically calculated as: 1.25E8.

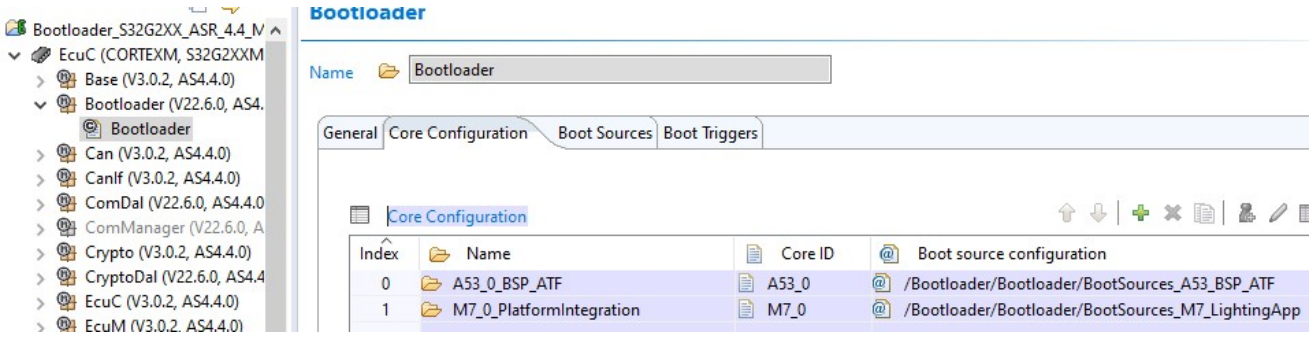
- ClockReferencePoint add UART_CLK。

…McuClockSettingConfig_0->McuClockReferencePoint: Click the + sign in the upper right corner to add an item, click to enter:
1. Modify Name= UART_CLK
2. Mcu Clock Frequency Select：select: LIN_BAUD_CLK
3. Mcu Clock Reference Point Frequency (0 -> 5000000000)：automatically calculated as:：1.25E8

The clock modification of other peripherals is mainly as mentioned above, which is not controlled by MCU.

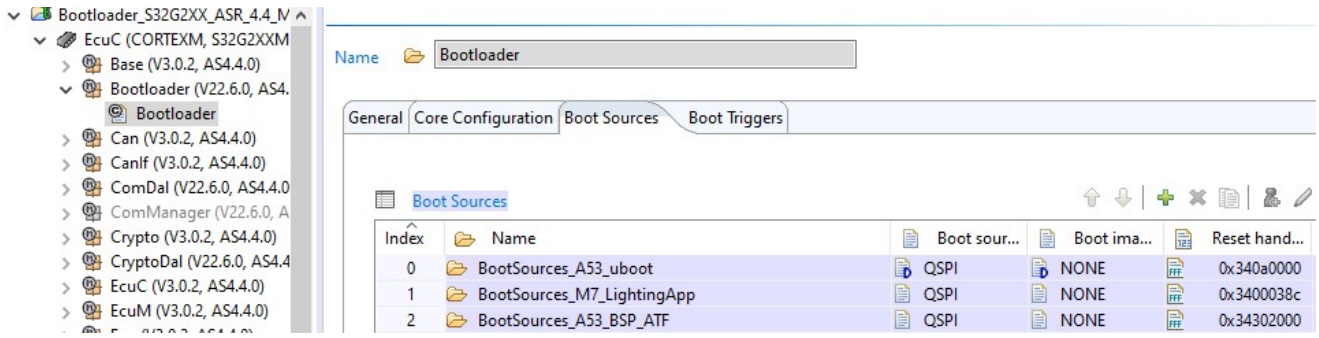
3.6 Configure A53 Boot sources

1. Open Bootloader(…)->Bootloader->Core Configuration: By default it has been configured as:



So it is to start the A53's Bootloader including ATF, and an APP of M7_0. Note that you can also add and start M7_1/2, which is not explained in this article.

2. Open Bootloader(…)->Bootloader->Boot Sources: The following is the reset address:



Enter ->BootSources_A53_BSP_ATF->Boot image fragment->ImageFragments_0: In addition, you need to set the load address and size of BL2 of ATF, you can view the compilation log of ATF as follows:

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

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

So in BootSources_A53_BSP_ATF->General->Reset handler address, keep the value 0x34302000 unchanged.

In BootSources_A53_BSP_ATF->Boot image fragments->ImageFragments_0:

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

Note the note: "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."

So the load image address requires 64 Byte alignment when compiling ATF. Image size (bytes)= 262144=256KB= //>0x2a800+0x1240=0x2BA40.

3.7 Configure M7 Boot sources

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

Modify the name: Name= BootSources_MCAL_Test。

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

```
int_sram : ORIGIN = 0x34400000, LENGTH = 0x00180000 /* 1.5MB size, 4MB offsit */
```

Refer the compiled mapping file:
C:\NXP\SW32G_RTD_4.4_3.0.2\eclipse\plugins\Mcu_TS_T40D11M30I2R0\examples\EBT\Mcu_Example_S32G274A_M7\out\main.map

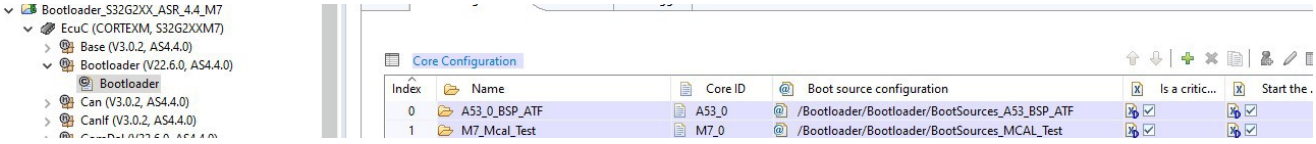
```
.startup 0x34400010 0x1d0 tmp/startup_cm7.o
0x34400010 Reset_Handler
0x34400010 _start
```

bin file size is: 1.28MBytes=0x140000

The size of the bin file is obtained as:1.28MBytes=0x140000。

In summary:
1. BootSources_M7_LLCEtoPFE_App->General-> Boot souce=QSPI //Keep unchanged
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 // Keep unchanged
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. Due to the modified name, in Bootloader(...)->Bootloader->Core Configuration: in A53_0_BSP_ATF Modify or add an item below, the name is changed to: M7_Mcal_Test, the Core ID is changed to: M7_0, Boot source

configuraiton point to /Bootloader/Bootloader/BootSources_MCAL_Test。



3.8 Turn off debug soft breakpoints

In Bootloader(…)->Bootloader->General Software breakpoint enable uncheck it. turn off the software Infinite loop, as described below:

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.

Source code description:

```
/* Compile switch to enable debug breakpoint */
#define BL_USE_BREAKPOINT STD_ON // Changed to OFF after modification
//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 */
```

Then generate the code:

Right-click on Bootloader(…)->EcuC(…) and select Generate Project to generate code. The generated code is located at: C:\EB\tresos\workspace\Bootloader_S32G2XX_ASR_4.4_M7\output, note that it is best to delete all files in this directory before regenerating after each modification.

Note the configuration of Source address (in QSPI) in Imagefragments_0: A53 is 0x100000, M7_0 is 0x200000, this address is the address of the A core fip.bin and M core *.bin to use flash tools next.

3.9 Compile the Bootloader project

1. Modify the build configuration

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

::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

…

Run in Cygwin:

./launch.bat

To compile it.

If met the error:

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'

Then in Bootloader(…)->EcuC(…)->Gpt(…)->Gpt->GptChannelConfiguration->GptChannelConfiguration_0:

Shut down: GptNotification.

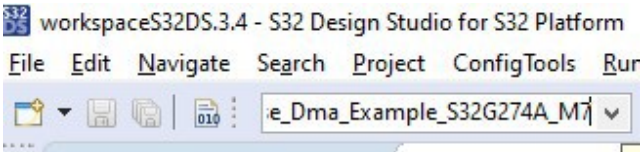
After the compilation is successful, the image is generated:

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\bin_bootloader

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

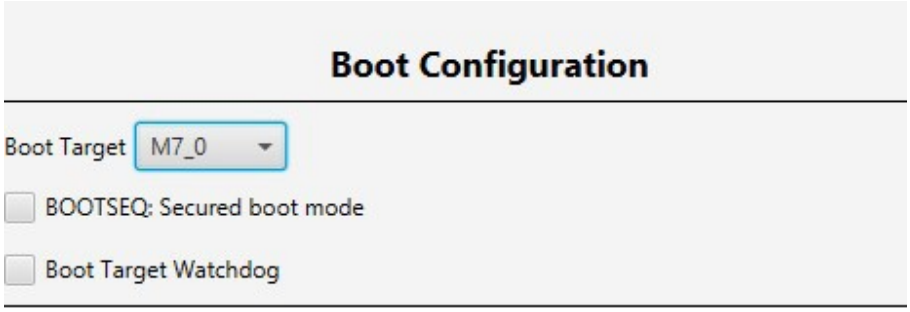
3.10 Making a Bootloader Image with IVT

Open 32 Design Studio for S32 Platform 3.4 (3.4.3), click the ConfigTools drop-down menu, and select IVT. (Note: You need to open a project to select this menu, marked as there is a project name in the project drop-down list):



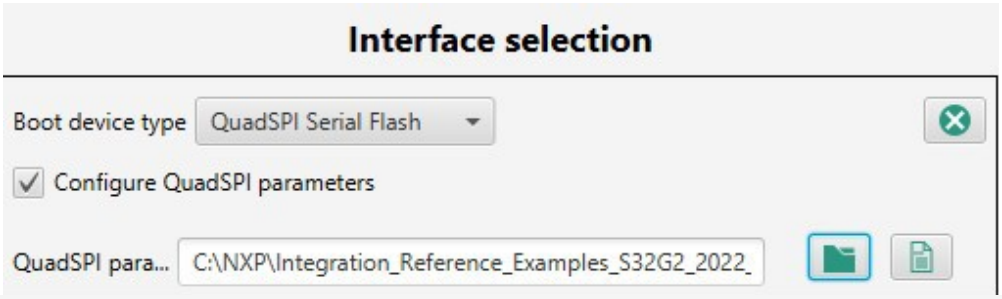
Open: IVTView

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



- Interface selection：In Boot device type choose：QuadSPI Serial Flash, then checked Configure QuadSPI parameters, choose QSPI NOR timming header:
C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_QuadSPI_133MHz_DDR_configuration.bin

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

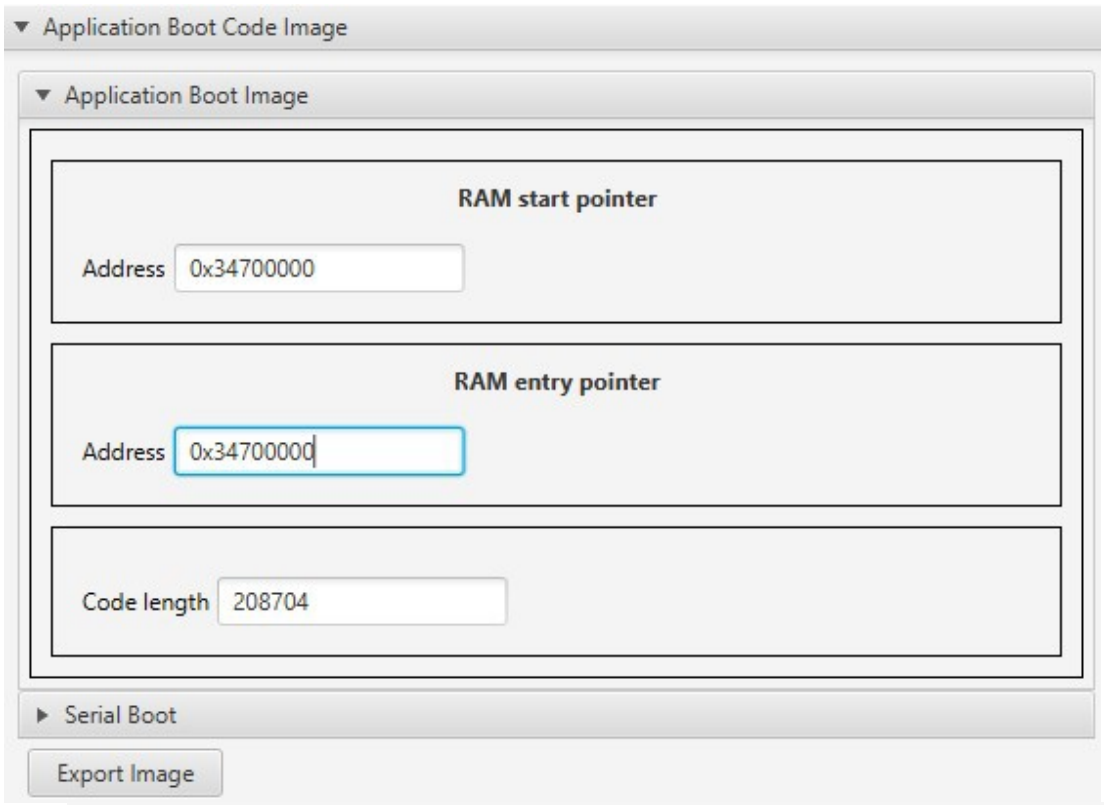


- DCD: The data structure called by the internal ROM in the DCD segment is used to initialize the internal SRAM. The default is On to open, Select file as:
C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_DCD_InitSRAM.bin.

- Application bootloader: choosed our compiled output Bootloader:
C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\bin_bootloader\Bootloader.bin.
RAM start pointer and RAM entry pointer refer mapping file: 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
```

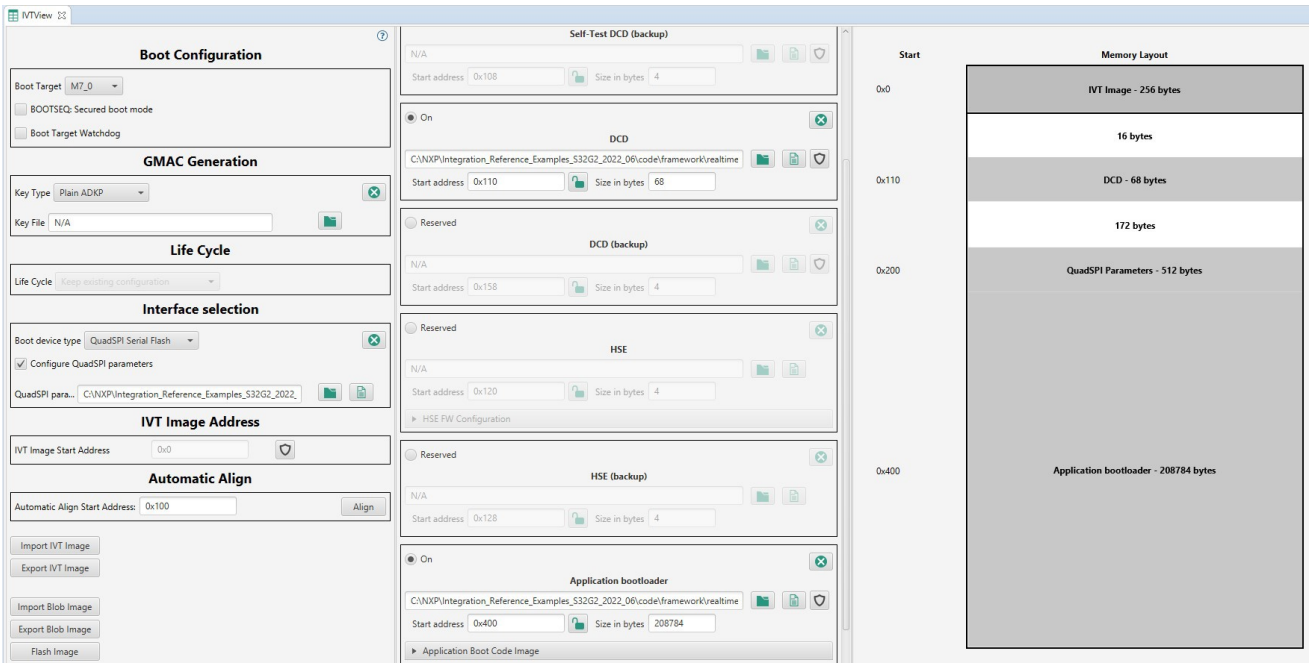
So all of them are:0x34700000:



Then in Automatic Align, enter 0x100, then click Align, if the automatic Align is successful, it will pop up successfully prompt box, if it fails, manually adjust the Align value and try again. The requirements of Align are generally for eMMC, QSPI NOR The requirements are not strict.

Finally, click the Export Image button in the Application Boot Image, name it (bootloader_a_m.bin) and save it in this then the file will be automatically re-opened in the Application bootloader.

- Click the On button to close all other unused image segments and turn them into Reserved. Click the Export Blob Image button, Export the final Bootloader image:



Stored the file name: ootloader_a_m_blob.bin.

3.11 Burning the Image

Hardware connection:

- Use USB cable to connect PC and UART0, J2 on RDB2 board
- RDB2 is set to download mode: SW10-1=OFF, SW10-2=OFF. Power-on.

Run C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\GUI\s32ft.exe, Target select S32G2xxx, Algorithm chose the MX25UW51245G. The port names: of the COM port is set to the serial port seen in the device manager: COM22 Then click Upload target and algorthim to hardware…, the programming tool will load the algorithm image and configure the programming equipment

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

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

- Use flash tools to program the bootloader image to QSPI:

Click Upload file to device… to program “bootloader_a_m_blob.bin” to address 0x0.

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

Click Upload file to device…, program "fip.bin" to address 0x100000, before programming address reference The QSPI source address configured by Bootloader MCAL, the programming is to note that the fip.bin file is programmed, this is not with A53 Bootloader fip.s32 for IVT header.

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

Click Upload file to device..., program "int_app.bin" to address 0x200000, this is LLCE to PFE MCAL test image.

- Program the A53 Linux image to SDcard: According to the document <<S32G_Kernel_BSP32_V4-20220513.pdf>>, it is stated that SDcard reader will be used in Ubuntu program into TFcard:

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

And update fip.s32 after modifying Align:

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

Then insert the TFcard into the J3 TFcard slot on the RDB2 board, and set SW3=On to switch to TFcard startup.

4 Test

4.1 Hardware Link

- Set RDB2 to normal boot mode, boot from QSPI NOR: SW10-1=ON, SW10-2=OFF, SW4 all=OFF.
- Use a USB cable to connect the PC and UART0, J2 on the RDB2 board, open the serial port terminal on the PC side, and set it to 115200, 8-n-1. λ DIO MCAL test requires SW11=on, so the pin is connected to LED GPIO instead of SPI.
- The UART MCAL test needs to be connected to UART1 on the PC and RDB2 board, open the serial port terminal on the PC side, set it to 115200, 8-n-1.

4.2 MCU MCAL+Linux test

Power on directly and start in normal boot mode. If the Linux terminal of the A core starts normally, it means that the Bootloader boots the A core successfully, and the M core MCAL image does not affect the A core startup.

4.3 DIO MCAL+Linux test

Power on directly and start in normal boot mode. If the A-core Linux terminal starts normally, it means that Bootloader Boot A Kernal success. U128 GPIO LED light flashes for a period of time (the A core driver stops after loading SPI), indicating that DIO MCAL M7_0 The example code starts normally.

4.4 UART MCAL+Linux test

Power on directly, start in normal boot mode, and use two USB cables to connect UART0/UART1 respectively. A core Linux If the normal startup of the terminal is printed in the UART0 terminal, it means

that the Bootloader Boot A core is successful. Serial port connected to UART1 (115200,8-n-1) terminal prints:

"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 source codes call roution

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

| | | | | |->// Peripheral driver initialization

```
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 // This is the main program to start each core:

The input parameters are:

```
coreStartAddress = (applicationConfig->u32ResetHandler) & 0xFFFFFFFC; //reset address
    coreId = Bl_GetCoreInternalId(applicationConfig->core); //such as coreID=0, which is the first A53 kernel
    partition = BL_GetPartition(applicationConfig->core);//such as A53  in partition 1
```

For details, please refer to the partition mapping in the Mode Entry Module MC_ME chapter of the chip manual. The following is the process of starting a core:

```
/* 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 //de initial

SysDal_Rtm_DeInit();

Mcl_DeInit();

Mcu_InitClock(McuClockSettingsDisablePLL); //

Qspi_Ip_ControllerDeinit(0);

Gpt_DeInit();

6 Bootloader customization

6.1 QSPI NOR driver

The Bootloader project integrates the QSPI NOR driver. Note that this driver is only for the QSPI NOR flash designed on the RDB2 board. If the QSPI NOR flash is modified, please refer to the document:

1. AN13563.pdf: <<S32G QuadSPI Deep Dive>>: This article mainly describes the QSPI NOR MCAL driver customization.
https://www.nxp.com/products/processors-and-microcontrollers/s32-automotive-platform/s32g-vehicle-network-processors/s32g2-processors-for-vehicle-networking:S32G2

2. <<S32G_QSPINOR_Custom_xxxxxxxx_Vx.pdf>>: This article mainly describes the customization of the QSPI NOR IVT header timming configuration header, as well as the customization of Linux and flash tools.
https://community.nxp.com/t5/NXP-Designs-Knowledge-Base/S32G-QSPI-Nor-customization-doc/ta-p/1399906

6.2 eMMC/SDcard Boot Support

Bootloader project is driven by M7 core supporting SDHC by default, please refer to the document <<Hands-on S32G2 Multicore application enablement example.pdf>> Wang Xuewei description.

But because eMMC/SDcard starts slowly, it is more common to put BL2 ATF in QSPI-NOR.

6.3 DDR initial

The current Linux BSP, starting from BSP32, is the version that supports ATF by default, so the Bootloader will load it first ATF into the internal SRAM, then ATF initializes the external DDR, and then

ATF is responsible for loading Uboot from eMMC/SDcard code into external DDR.

The previous BSP version does not support ATF by default, so the entire Uboot image needs to be loaded into the internal SRAM, This takes up too much space, so the general practice is to initialize the external DDR in the Bootloader first, and then use the Bootloader reads Uboot into external DDR from eMMC. So you need to initialize the LPDDR4 code in Uboot The code is ported to Bootloader. With the upgrade of BSP, it is no longer needed at present.

6.4 Secure Boot Support

Please refer doc<<Hands-on S32G2 Multicore application enablement example_review.pdf>> Wang Xuewei.

7 Debug

7.1 Bootloader Debug

Lauterbach script bit with: C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm

Therefore, as in Section 3.8, if the software interrupt is not turned off, the Bootloader code will stay at the beginning. At this time, run the script to connect the debugger to debug.

7.2 MCAL Driver Debug

Modify the Lauterbach script: 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. modification as below:

```
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
```

Then add in the main function of the source code of UART MCAL: 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);
```

Then the code will stop at while, at this time use the above script to connect to lauterbach, manually change debug to 0, you can to continue running debugging.