NXP Semiconductors
Application Notes

Document Number: AN13750

Rev. 0, 11/2022

Enabling Multicore Application on S32G2 using S32G2 Platform Software Integration

by: NXP Semiconductors

Contents

1. Introduction

This application note is a step-by-step guide you to build a multicore IPC (Inter-Processor Communication) application on NXP S32G2 processor using the NXP software Bundle-2022.07. You can follow this guide as an example to enable all application cores of S32G274A processor.

The guide does not target optimization of the booting time. The SD-Card is used for booting and root file system media for Linux OS running on ARM® Cortex-A53® cores.

The secure boot in example only supports verification of the bootloader image using HSE SMR (Secure Memory Region) service. The bootloader is not configured to perform verifications on subsequent images.

The following sections are covered in the document.

- Multicore IPC application description
- Hardware and software prerequisites
- Prepare images for Cortex-A53 cores
- Prepare images for Cortex-M7® cores
- Configure and build the bootloader
- Deployment on S32G2-VNP-RDB2

- Run the applications on S32G2-VNP-RDB2

2. Multicore IPC application description

After booting up, the bootloader loads the IPC application for each Cortex-M7 core. It also gets application to run a sample application on Linux to send “Hello world!” messages to Cortex-M7 cores via IPC channel. On receiving the message, the IPC application on Cortex-M7 core responds with an echo message.

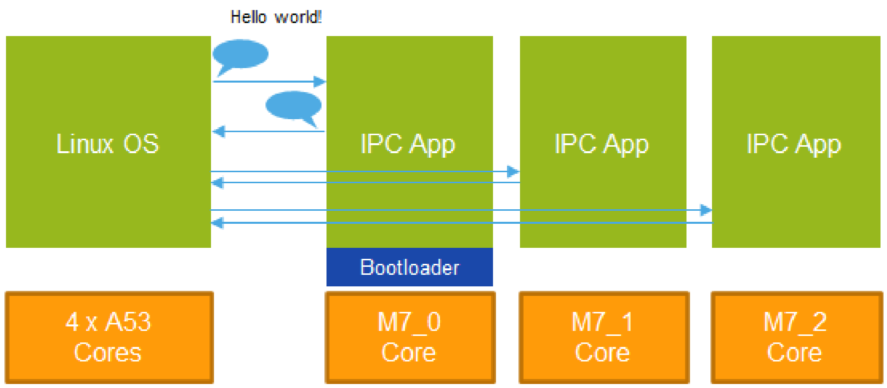


Figure 1. **Sample application**

2.1. S32G2 Boot flow

The secure boot is enabled in the default configuration of the Bootloader. For the First time boot after image deployment on RDB2, the secure boot is not enabled, it means, the BOOT_SEQ in the IVT is set to zero. When the bootloader runs for the first time, it detect this condition and configures the HSE for secure boot and then set BOOT_SEQ=1. After setting BOOT_SEQ=1, the bootloader issues a functional reset. For every following boot, secure boot is enabled.

You can disable the secure boot in the EB Tresos configuration. For detail, please refer to the section ‘7.2.2 Secure boot configuration’ of the Bootloader User Manual.

The following figures show S32G2 boot flow examples for both non-secure and secure boot.

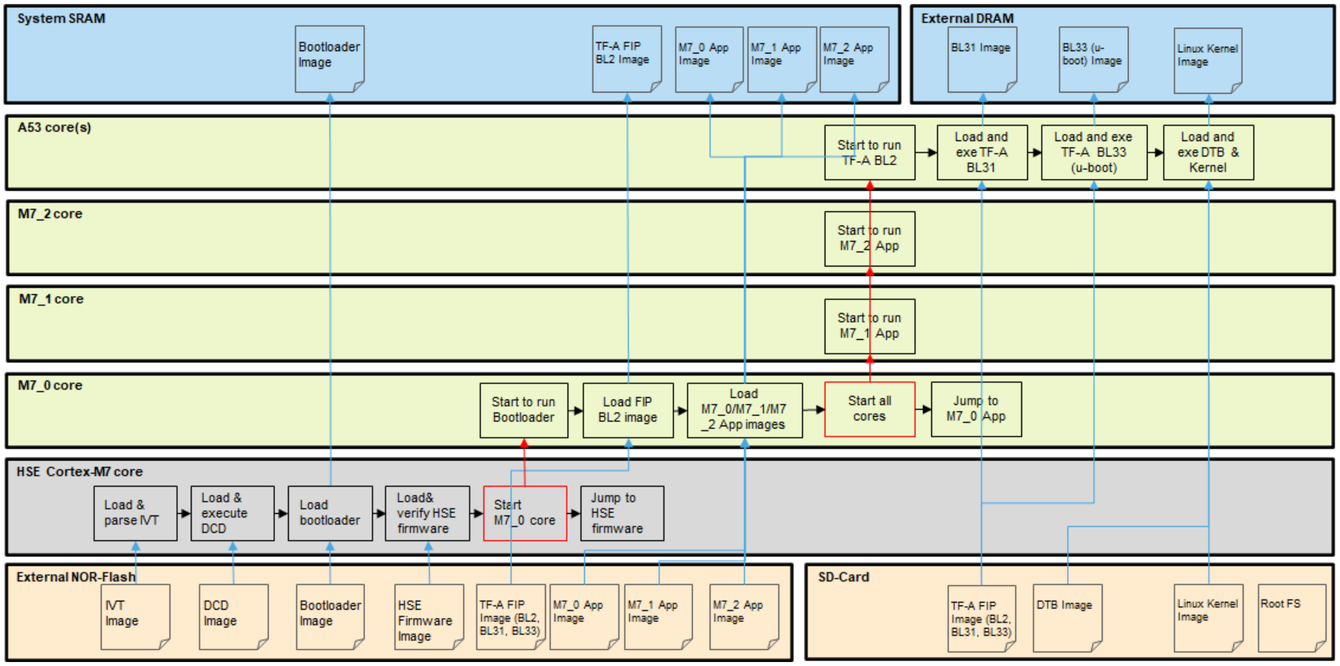


Figure 2. **Non-secure boot flow**

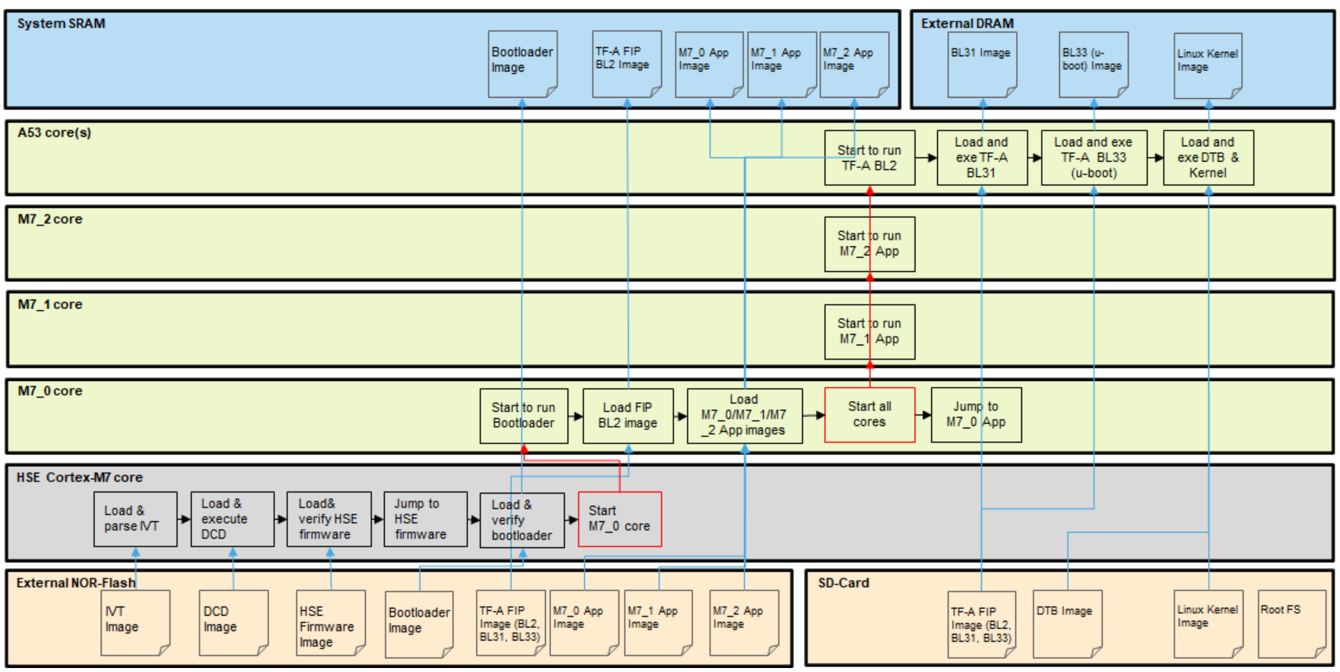


Figure 3. **Secure boot flow**

NOTE

For definition of BL2/BL31/BL33, refer to the section “25 ARM Trusted Firmware” in the “Linux BSP 33.0 User Manual for S32G2 platforms”.

3. Hardware and software prerequisites

The following table shows the hardware and software prerequisites.

Table 1. Harware and software prequisites

Item	Description	Note
S32G-VNP-RDB2	Hardware board with the processor S32G274A	
S32 Design Studio 3.4 with the update 3 (3.4.3_D2112)	IDE with S32 Configuration Tool and S32 Flash Tool.	
EB Tresos Studio 27.1	AUTOSAR configuration tool.	It is required to modify AUTOSAR configuration of the bootloader.
S32G2 Standard & Reference Software	NXP released software. Below items are required: S32G2 Platform Software Integration 2022.06 HSE Standard Firmware 0.1.0.5 Inter-Platform Communication Framework 4.6.0 Linux BSP 33.0.0 Real-Time Drivers 3.0.2 HF01 FreeRTOS 3.0.2 SDHC Stack 1.0.1 HF1	Please download each software in your NXP software account or use the Automotive Software Package Manager.
Cygwin	A large collection of GNU and Open Source tools which provide functionality similar to a Linux distribution on Windows.	It is used to run the make tool and to deploy images on SD-Card.
Putty	Serial terminal	

4. Prepare images for Cortex-A53 cores

Follow the user guide of Linux BSP 33 to build the u-boot and ATF

1. Download the GCC 10.2.0 toolchain for ARM 64-bit (download link).

2. Once you have downloaded the toolchain package, in order to install it, you just need to unzip it in a directory of your choice.

   $tar -xf gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu.tar.xz

3. Clone the GIT repository

   $export HOME=/path/to/your/workspace
   $cd $HOME
   $git clone https://source.codeaurora.org/external/autobsps32/u-boot
   $cd u-boot
   $git checkout release/bsp33.0-2020.04

4. Build the U-Boot bootloader

   $export CROSS_COMPILE=$HOME/gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
   $make s32g274ardb2_defconfig
   $make

To build the Arm-trusted-firmware the steps needs to be followed:

1. Install libssl-dev and openssl for headers required by fiptool. This is a one-time operation.
   $sudo apt-get install libssl-dev openssl
2. Check your host dtc version.
   $dtc --version
3. If your dtc is older or you do not have it installed, install/upgrade to dtc version 1.4.6 or above:
   $sudo apt-get install device-tree-compiler
4. Clone the GIT repository
   $cd $HOME
   $git clone https://source.codeaurora.org/external/autobsps32/arm-trusted-firmware
   $cd arm-trusted-firmware
   $git checkout release/bsp33.0-2.5

5. Apply the patch to modify the alignment of ATF. Please find the patch in accompanying software package of this application note.

   The bootloader expects a 64-bytes-aligned image. Thus, the ATF makefile arm-trusted-firmware/plat/nxp/s32/s32_common.mk needs the parameter `FIP_ALIGN := 16` change it to `FIP_ALIGN := 64` before you start the build.

   $git am < /path/to/0001-fip-align-and-mmc-init.patch

6. Build the ATF

   $make ARCH=aarch64 PLAT=s32g274ardb2 BL33=$HOME/u-boot/u-boot-nodtb.bin

7. After build is complete, the generated images are in the directory: arm-trusted-firmware/build/s32g274ardb2/release. The log shows load address and entry point of the generated FIP image, like below:

   IVT Location: SD/eMMC
   Load address: 0x342fc580
   Entry point: 0x34302000

The following steps shows how to build an IPC multiple instance example on Linux:

1. Build the kernel using the following command

   $cd $HOME

```
$git clone https://source.codeaurora.org/external/autobsps32/linux
$cd linux
$git checkout release/bsp33.0-5.10.109-rt
$make ARCH=arm64 s32gen1_defconfig
$make ARCH=arm64
```

The command generates the kernel binary (Image) in *arch/arm64/boot* and the board device tree blobs in *arch/arm64/boot/dts/freescale*.

2. Build the IPCF modules.

```
$cd $HOME
$git clone https://source.codeaurora.org/external/autobsps32/ipcf/ipc-shm
$cd ipc-shm
$git checkout release/SW32G_IPCF_4.6.0_D2205
```

Apply the patch to enable three IPC instances. Please find the patch in accompanying software package of this application note.

```
$git am < /path/to/0001-ipc-multi-instances.patch
```

Build IPCF driver and sample modules providing kernel source location

```
$cd $HOME
$make -C ./ipc-shm/sample_multi_instance KERNELDIR=$PWD/linux modules
```

To deploy the built image to a SD card the steps needs to be followed:

1. Download the pre-built images of Linux BSP 32 from nxp.com: *binaries_auto_linux_bsp33.0_s32g2_pfe.tgz*
2. Extract the package to your local folder.
3. Deploy images on SD-Card (On Windows)
    - Insert the SD-Card to your PC
    - Launch the *Cygwin* (run as administrator) and run the following commands to write bsp image to SD-Card

NOTE

/dev/sdb is the device node for the SD-Card.

```
$cd binaries_auto_linux_bsp33.0_s32g2_pfe/s32g274ardb2/
$dd if=/dev/zero of=/dev/sdb bs=512 count=1 && sync
$dd if=fsl-image-auto-s32g274ardb2.sdcard of=/dev/sdb bs=1M skip=4 seek=4
$dd if=fsl-image-auto-s32g274ardb2.sdcard of=/dev/sdb bs=1M count=4 && sync
```

- Copy the *fip.bin* and *fip.s32* from the directory `*arm-trusted-firmware/build/s32g274ardb2/release*` to the current folder. Use the newly built FIP image to replace the one from pre-built images:

```
$dd if=fip.s32 of=/dev/sdb skip=512 seek=512 iflag=skip_bytes oflag=seek_bytes
conv=fsync,notrunc
```

NOTE

fip.bin will be deployed on Nor-Flash.

- Copy the below images to FAT32 partition (boot_s32g27) of SD-Card

- Copy the kernel image (linux/arch/arm64/boot/Image) and replace the one on SD-Card.
- Copy the dtb image (linux/arch/arm64/boot/dts/freescale/s32g274a-rdb2.dtb) and replace the one on SD-Card.
- Copy the ipc image (ipc-shm/ipc-shm-dev.ko, ipc-shm/sample_multi_instance/ipc-shm-sample_multi-instance.ko).

5. Prepare images for Cortex-M7 cores

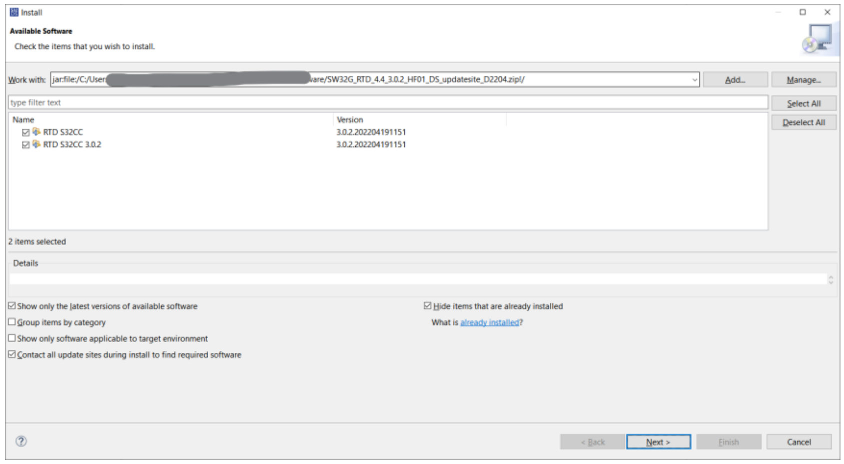
5.1. Building ipc application using S32DS

To install the S32 Design studio, follow these steps:

1. Download the S32 Design Studio 3.4 and complete installation
   - Download the installer file: S32DS.3.4_b201217_win32.x86_64.exe, finish installation following the S32DS Installation Guide.
   - Download the offline package includes Update 3 for S32 Design Studio v.3.4 and S32G development package: SW32G_S32DS_3.4.3_D2112.zip. Follow the release note to install it.

Follow the steps to install RTD and IPCF package for S32DS:

1. Download the RTD release: SW32G_RTD_4.4_3.0.2_HF01_DS_updatesite_D2204.zip.
2. Launch the S32DS. Go to Help > `Install New Software…`. In the Install window, click `Add…` > `Archive…` to open the SW32G_RTD_4.4_3.0.2_HF01_DS_updatesite_D2204.zip. Click Add to close the window.
3. In the Install window, select all items, click `Next >` to finish installation.
4. Download the below IPCF, FreeRTOS and SDHC release. Follow similar steps to finish installation
   - SW32G_IPCF_4.6.0_D2205_updatesite.zip
   - SW32_FreeRTOS_10_4_6_UOS_3_0_2_DS_updatesite_D2204.zip
   - S32G_SDHC_RTM_1_0_1_HF1_D2207_updatesite.zip



Follow the steps to build ipc application using S32DS:

1. Find the example used as an accompanying software package of this application note. Extract the package to your local folder.
2. Launch the S32 Design Studio.
3. Import the three projects into S32DS:
   - *IPCF_Example_multi_instance_S32G274_M7_0*,
   - *IPCF_Example_multi_instance_S32G274_M7_1*,
   - *IPCF_Example_multi_instance_S32G274_M7_2*.
4. For each project, double click the *mex* file, open the S32 Configuration Tool. Then click the `` `*Update Code*` `` button to generate the source code.
5. Build projects to get elf and binary image. The outputs are in the *Debug_RAM* folder.

5.2. Configure the bootloader

Follow the steps to install RTD and Platform Software Integration package for EB tresos:

1. Download and install the RTD software
   - Download the installer `` `*SW32G_RTD_4.4_3.0.2_HF01_D2204.exe*` ``, double-click to install it.
   - After installation, you will get the folder `` `*SW32G_RTD_4.4_3.0.2_HF01*` `` at your installation directory.
   - Copy all plugins of RTD to the EB directory: copy all items at the directory `` `*SW32G_RTD_4.4_3.0.2_HF01\eclipse\plugins*` `` to `` `*C:\EB\tresos\plugins*` ``
2. Download and install the platform software integration software `` `*Platform_Software_Integration_S32G2_2022_06.exe*` ``

- Download the installer `*Platform_Software_Integration_S32G2_2022_06.exe*`, double-click to install it. In this example, we install it to the default directory: C:\NXP\Integration_Reference_Examples_S32G2_2022_06.

Follow the steps to import the bootloader project:

1. Launch the EB tresos 27.1.0
2. Follow the menu *File* > *Import* to open the *Import* window. Select *General* > `*Existing Projects into Workspace*`, then click *Next*.
3. Click `*Browse…*` and in the `*Brower For Folder*` window navigate to the installation directory of `*Integration_Reference_Examples_S32G2_2022_06*`. Then click *Ok*.
4. It will show all projects in the selected directory. Uncheck the lighting projects. In the *Options* section, select `*Copy projects into workspace*`, then click *Finish*.

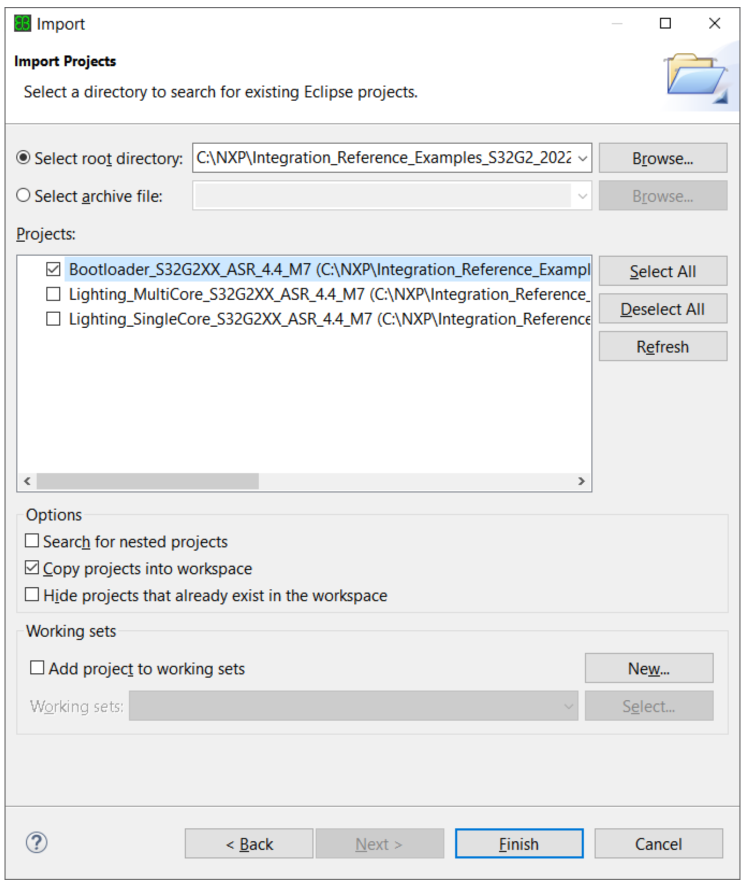
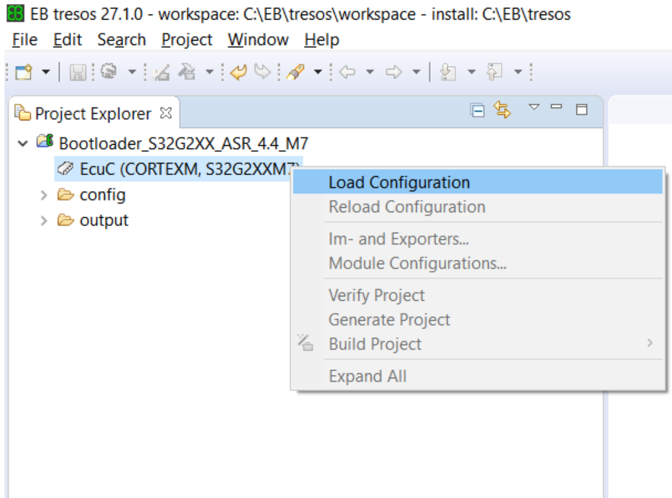


Figure 4. **Project import**

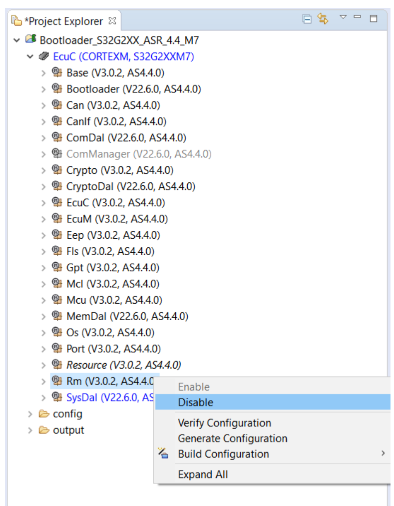
Follow the steps to Configure the bootloader:

1. In the `*Project Explorer*` view, right click the *Ecuc* item, and select the menu `*Load Configuration*`.

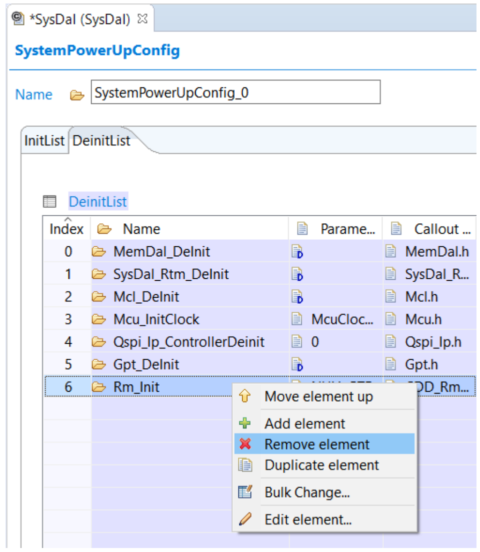


2. In order to make it simple, we will disable the XRDC in this example. Follow below steps to disable XRDC

   - Right click the *Rm* plugin and select the menu *Disable* to disable the RM plugin

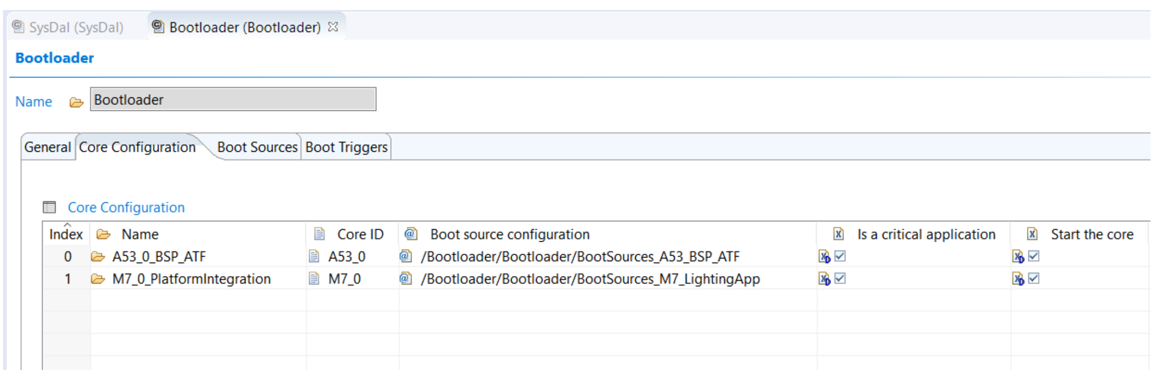


   - Navigate to the *SysDal* plugin configuration, select *SysDalBswConfig* > *PowerUp* > *DeinitList*. In the list, remove the item *Rm_Init*.

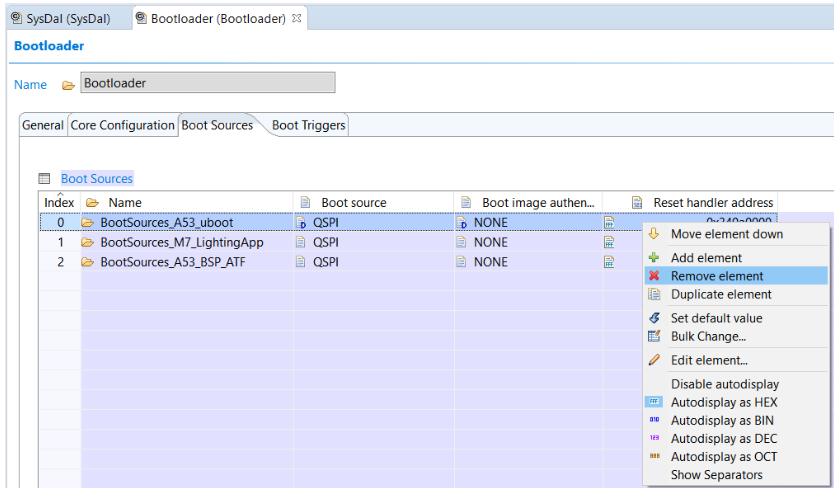


3. Navigate to the configuration of *Bootloader* plugin. Select the `*Core Configuration*` and it lists all boot targets and boot sources that will be loaded by the bootloader. This example uses the bootloader to bring up Linux BSP on A53 cores and IPC example on each M7 core. So, you need to configure the four elements in this list:
   - 1x for Cortex-A53 cores: Uses the images from Linux BSP 33
   - 3x for the three Cortex-M7 cores: Use the images of IPC application built by the S32 Design Studio

   These list will be edited later.



4. Navigate to the *Bootloader* plugin configuration. Select the ‘Boot Source’ configuration. Remove all elements in the boot source list.



5. Add Boot source for the ATF FIP image

- Click the button `*add new element with default values*`, and edit it as below:

Boot Sources

Index	Name	Boot source	Boot image authen...	Reset handler address
0	linux_bsp_atf	QSPI	NONE	0x34302000

- Navigate to `*Boot Sources*` > `*linux_bsp_atf*` > `*Boot image fragments*` and configure the `*Boot image fragments*` of the `*linux_bsp_atf*` as below:

Boot image fragments

Index	Name	Source address (in...	M.	C..	Load image at addr...	Image size (b...	Image CRC value
0	ImageFragments_0	0x100000			0x342fc580	262144	0x0

NOTE

- The `Reset handler address` and `Load image address` are from the log of building the ATF. The example:

  IVT Location: SD/eMMC

  Load address: 0x342fc580

  Entry point: 0x34302000

- The image size was set to 256KB. It should be larger than the size of BL2. Find the value from the log of building ATF. The example:

  Image Layout

  DCD: Offset: 0x200 Size: 0x1c

  IVT: Offset: 0x1000 Size: 0x100

  AppBootCode Header: Offset: 0x1200 Size: 0x40

  Application: Offset: 0x1240 Size: 0x2ec00

6. Add a Boot source for IPC application on Cortex-M7_0 core

- Click the button `*add new element with default values*`, and edit as shown below:

Boot Sources

Index	Name	Boot source	Boot image authen...	Reset handler address
0	linux_bsp_atf	QSPI	NONE	0x34302000
1	ipc_app_m7_0	QSPI	NONE	0x34380010

- Navigate to `*Boot Sources*` > `*ipc_app_m7_0*` > `*Boot image fragments*` and configure the `*Boot image fragments*` of the `*ipc_app_m7_0*` as shown below:

Boot image fragments*

Index	Name	Source address (in...	M.	C..	Load image at addr...	Image size (b...	Image CRC value
0	ImageFragments_0	0x200000			0x34380000	1048576	0x0

NOTE

The M7_0 core is the core running the bootloader. It will jump to the application by setting the PC to the reset exception handler address.

The `Reset handler address` and `Load image address` are indicated in the map file:

- Search the key word *Reset_Handler* in the map file to get the value of `*Reset handler address*`,
- Search the key word *int_sram_c0* in the map file to get the value of `*Load image address*`.

7. Add a Boot source for IPC application on Cortex-M7_1 core.

- Click the button `add new element with default values`, and edit as shown in the figure below.

Boot Sources*

Index	Name	Boot source	Boot image authen...	Reset handler address
0	linux_bsp_atf	QSPI	NONE	0x34302000
1	ipc_app_m7_0	QSPI	NONE	0x34380010
2	ipc_app_m7_1	QSPI	NONE	0x34501000

- Navigate to `Boot_Sources` > `ipc_app_m7_1` > `Boot image fragments` and configure the `Boot image fragments` of the `ipc_app_m7_1` as below:

Boot image fragments*

Index	Name	Source address (in...	M.	C..	Load image at addr...	Image size (b...	Image CRC value
0	ImageFragments_0	0x300000			0x34480000	1048576	0x0

NOTE

The `Reset handler address` and `Load image address` are indicated in the map file:

- Search the key word intc_vector in the map file to get the value of `Reset handler address`.
- Search the key word int_sram_c1 in the map file to get the value of `Load image address'.

8. Add a Boot source for IPC application on Cortex-M7_2 core.

   - Click the button `add new element with default values`, and edit as shown in the figure below:

Boot Sources

Index	Name	Boot source	Boot image authen...	Reset handler address
0	linux_bsp_atf	QSPI	NONE	0x34302000
1	ipc_app_m7_0	QSPI	NONE	0x34380010
2	ipc_app_m7_1	QSPI	NONE	0x34501000
3	ipc_app_m7_2	QSPI	NONE	0x34601000

Navigate to `Boot Sources` > `ipc_app_m7_2` > `Boot image fragments` and configurethe `Boot image fragments` of the `ipc_app_m7_2` as below:

Boot image fragments

Index	Name	Source address (in...	M.	C..	Load image at addr...	Image size (b...	Image CRC value
0	ImageFragments_0	0x400000			0x34580000	1048576	0x0

NOTE

The `Reset handler address` and `Load image address` are indicated in the map file:

- Search the key word intc_vector in the map file to get the value of `Reset handler address`,

- Search the key word int_sram_c2 in the map file to get the value of `Load image address`.

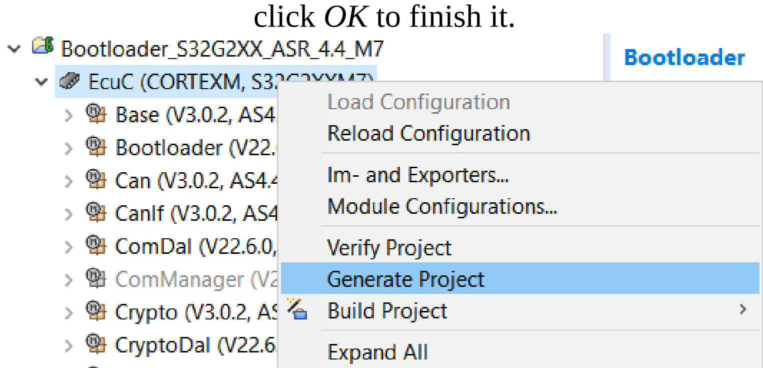
9. Navigate to *Bootloader* > `*Core Configuration*`, clear the core configuration list.

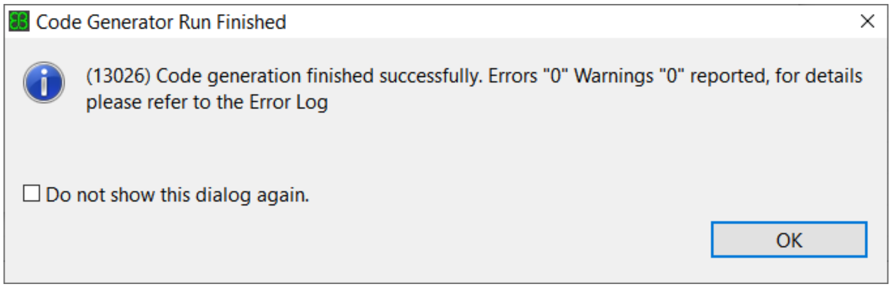
10. Add elements for each core with below configuration to the list as shown in the figure below.

Core Configuration

Index	Name	Core ID	Boot source configuration	Is a critical application	Start the core
0	A53_ATF	A53_0	/Bootloader/Bootloader/linux_bsp_atf	☑	☑
1	M7_0_IPC	M7_0	/Bootloader/Bootloader/ipc_app_m7_0	☑	☑
2	M7_1_IPC	M7_1	/Bootloader/Bootloader/ipc_app_m7_1	☑	☑
3	M7_2_IPC	M7_2	/Bootloader/Bootloader/ipc_app_m7_2	☑	☑

11. Right click the *EcuC* and select `*Generate Project*`. After successfully generated, click *OK* to finish it.





5.3. Build The Bootloader

Before compile the code, implement the below change to the file C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\ Bootloader.c:

```
#if (STD_ON == BL_CRYPTO_USED)
#include "CryptoDal.h"
+  #include "Hse_Ip.h"
-  static volatile uint32 ENABLE_BREAKPOINT_AT_MAIN = 0U;
+  static volatile uint32 ENABLE_BREAKPOINT_AT_MAIN = 1U;
int main(void)
{
#if (BL_USE_BREAKPOINT == STD_ON)
while (0U == ENABLE_BREAKPOINT_AT_MAIN) continue;
#endif /* BL_USE_BREAKPOINT == STD_ON */
+      while (1)  {
+          if ((Hse_Ip_GetHseStatus(0) & (HSE_STATUS_INIT_OK | HSE_STATUS_RNG_INIT_OK)) ==
+                (HSE_STATUS_INIT_OK | HSE_STATUS_RNG_INIT_OK))
+                break;
+      }
```

NOTE

The above code is in the patch format:

- `+` means inserting the line while `-` means deleting the line.
- The above changes disable the breakpoint by ENABLE_BREAKPOINT_AT_MAIN. Add the code to wait for completion of HSE firmware initialization, which avoids the conflict with HSE when initializing QuadSPI.

1. Navigate to installation directory `*Integration_Reference_Examples_S32G2_2022_06*`: C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build
2. Edit the file *launch.bat* to set build parameters, below is an example:

```
SET TRESOS_DIR=C:/EB/tresos
SET MAKE_DIR=C:/cygwin64
SET GCC_DIR=C:/NXP/S32DS.3.4/S32DS/build_tools/gcc_v9.2
```

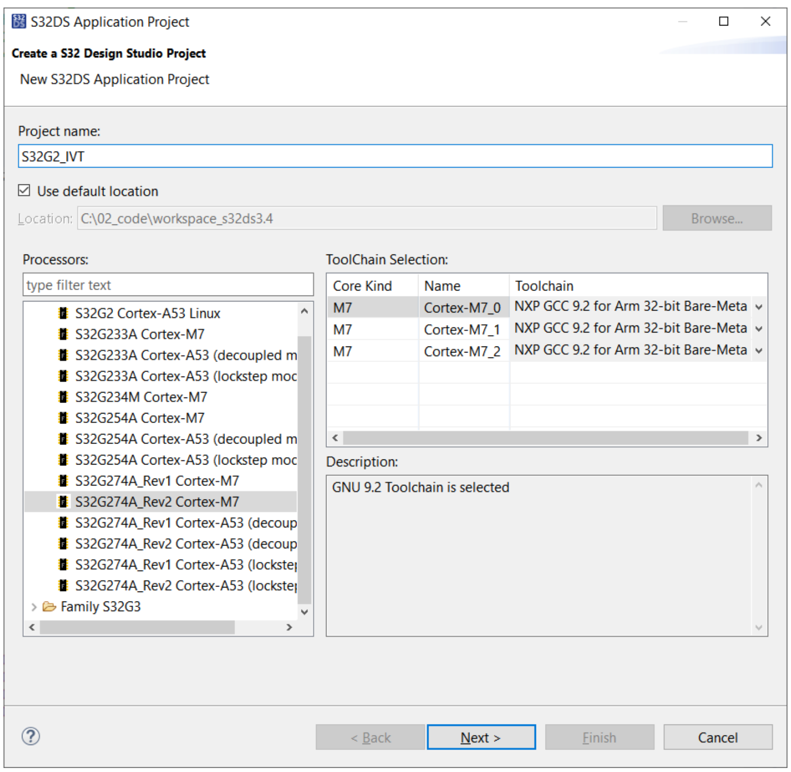
SET TOOLCHAIN=gcc
SET CORE=m7
SET SRC_PATH_DRIVERS=
C:/NXP/S32DS.3.4/S32DS/software/PlatformSDK_S32XX_2022_03/SW32_RTD_4_4_3_0_2_D2203
SET SDHC_STACK_PATH= C:/NXP/S32DS.3.4/S32DS/software/PlatformSDK_S32XX_2022_03/stacks/sdhc
SET TRESOS_WORKSPACE_DIR=C:/EB/tresos/workspace/Bootloader_S32G2XX_ASR_4.4_M7/output
SET HSE_FIRMWARE_DIR=C:/NXP/HSE_FW_S32G2_0_1_0_5

3. Launch *cmd.exe*, execute the *launch.bat*. After it finishes, find the output in the folder: *build/bin_bootloader*
   $cd
   C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build

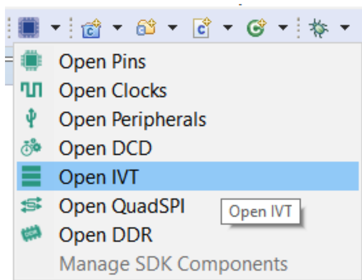
   $launch.bat

5.4. Generate S32G Boot Image Using S32DS IVT_TOOL

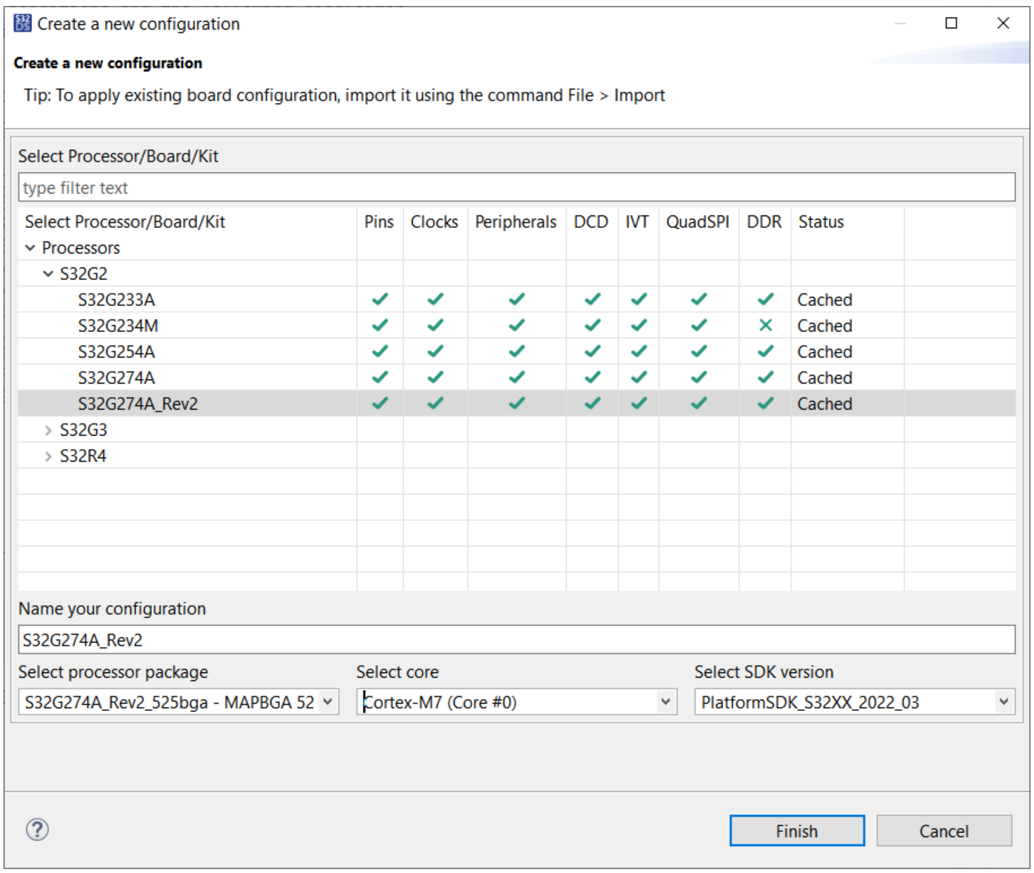
1. Open S32 Design Studio and click New -> S32DS Application Project.
2. In the dialog, select `*S32G274A_Rev 2 Cortex-M7*` (on the left-hand side), and on the right-hand side, select the *Cortex-M7_0* boot target.
3. Click *Next*, uncheck *Cortex-M7_1* and *Cortex-M7_2*, then click *Finish*.



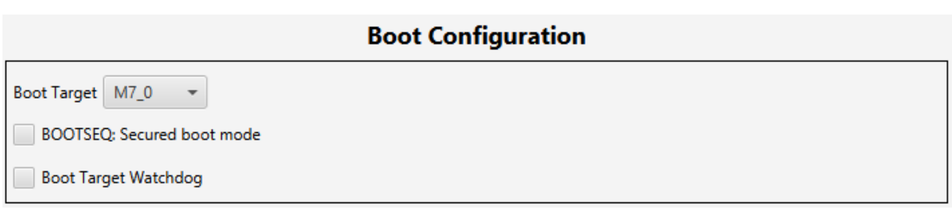
4. Click the button `*S32 Configuration Tools*` and select `*Open IVT*`.



5. In the Window `*Create a new configuration*`, select the processor *S32G274A_Rev2*.
6. Select SDK version and Core as below, then click *Finish*.

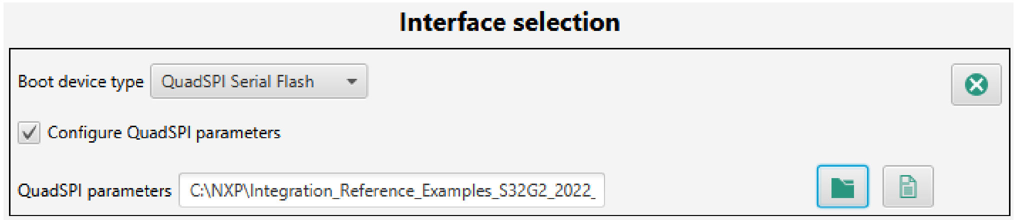


7. In the *IVT* view, set `*Boot Target*` to M7_0.



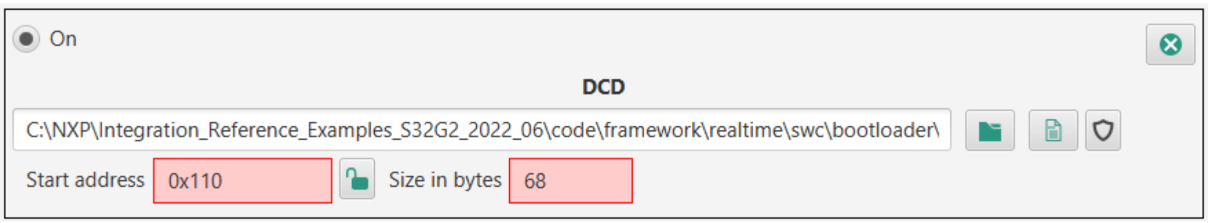
8. Boot device type select `*QSPI Serial Flash*`, and the QuadSPI parameters set to the file:

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



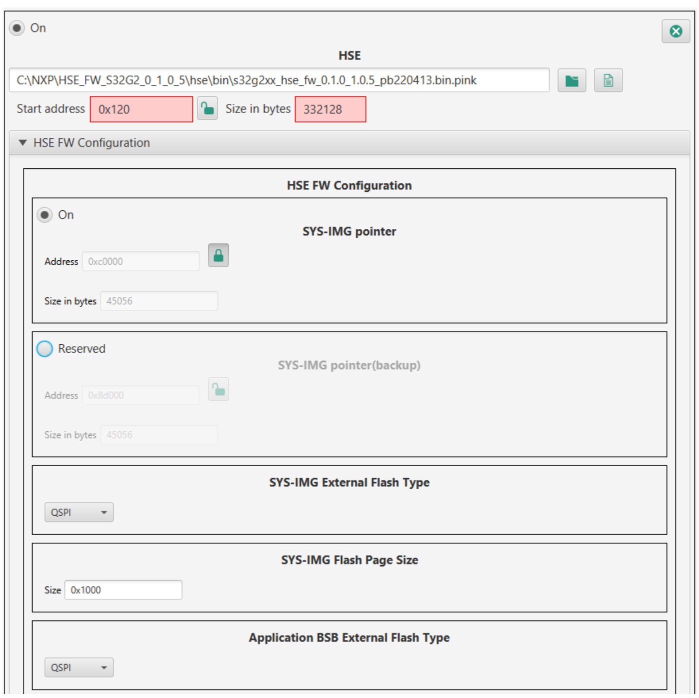
9. Configure the DCD image. This is used to initialize SRAM on boot stage. The image file is at:

Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\res\flash\S32G274_DCD_InitSRAM.bin



10. Configure the HSE images. Select the HSE firmware image, see the figure for the example.

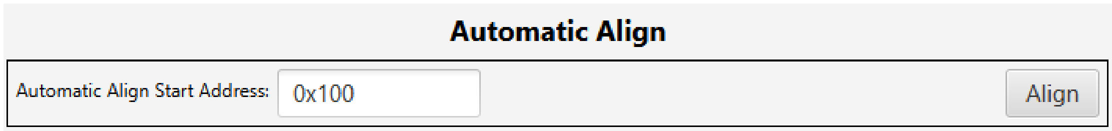
    For `*HSE firmware configuration*`, set the `*SYS-IMG pointer*` to the address *0xC0000*, then lock the address by clicking the lock.



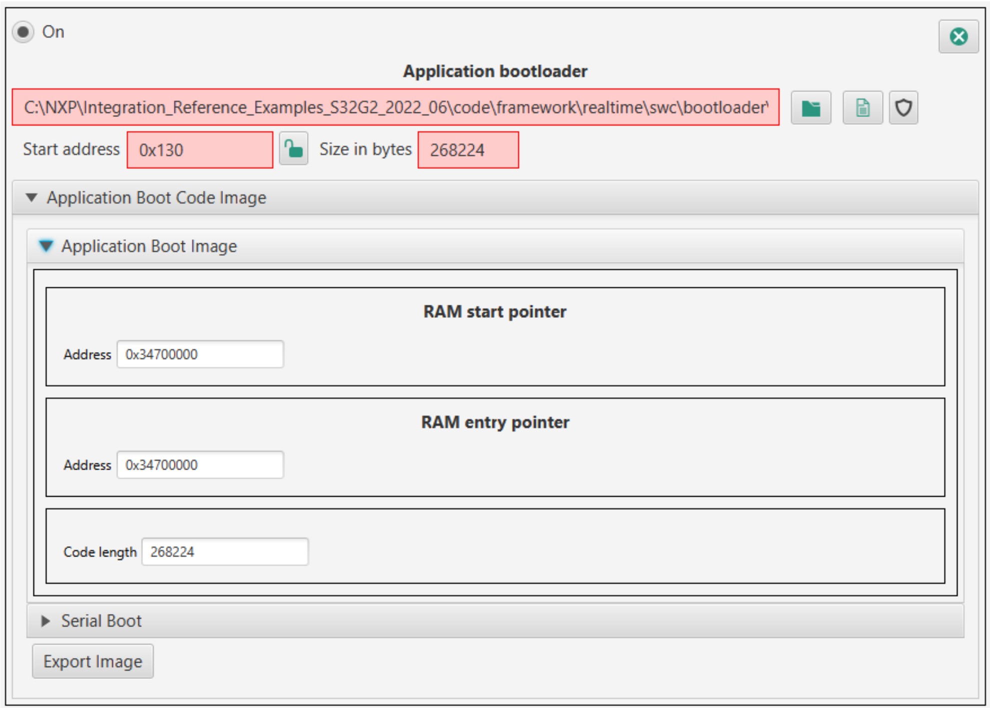
11. Configure the application bootloader image. Select the binary file *Bootloader.bin* in the folder: *build\bin_bootloader*.

The RAM start pointer and RAM entry pointer set to *0x34700000*.

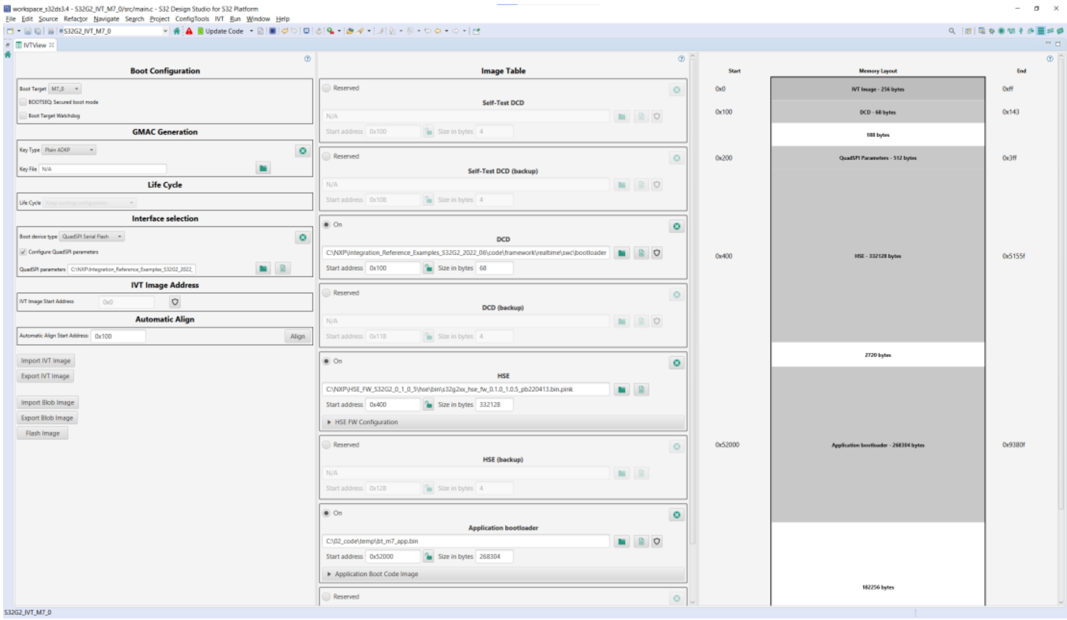
Set `*Automatic Align Start Address*` to *0x100* and click *Align*. If the start address of item is not aligned as expected, please manually adjust for that.



In the `*application bootloader*`, click `*Export Image*` and save it to local folder. In the example, we use *bt_m7_app.bin* as the file name.



12. Uncheck other images that are not used.



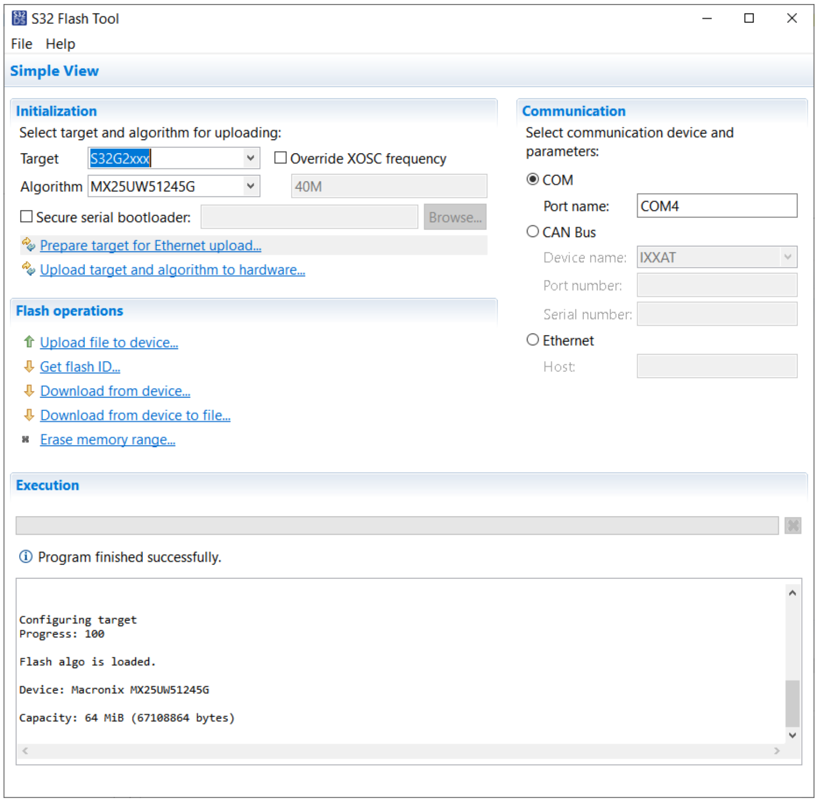
13. Click `*Export Blob Image*` and save the boot image to local folder. In the example, we use *bt_m7_blob.bin* as the file name.

6. Deployment on s32g-vnp-rdb2

The following are the steps to program the Nor-flash using the S32 flash tool.

1. Launch S32 Flash tool at the directory: C:\NXP\S32DS.3.4\S32DS\tools\S32FlashTool\GUI\s32ft.exe
2. Set the Target and Algorithm as the figure.
3. Connect RDB2 UART0 to your PC USB port
4. Set the COM port name, for example, *COM4*
5. Set the RDB2 to Serial boot mode: SW10-1=OFF,SW10-2=OFF
6. Power on the RDB2.
7. Click `*Upload target and algorithm to hardware…*`, then the tool will start to load the algorithm image and configure the target.
8. Click `*Erase memory range…*` and erase the memory range *0x0* - *0x500000*.
9. Click `*Upload file to device…*` and select the *bt_m7_blob.bin* to write it to the address *0x0*.
10. Click `*Upload file to device…*` and select the *fip.bin* to write it to the address *0x100000*.
11. Click `*Upload file to device…*` and select the *IPCF_Example_multi_instance_S32G274_M7_0.bin* to write it to the address *0x200000*.
12. Click `*Upload file to device…*` and select the *IPCF_Example_multi_instance_S32G274_M7_1.bin* to write it to the address *0x300000*.

13. Click `*Upload file to device…*` and select the *IPCF_Example_multi_instance_S32G274_M7_2.bin* to write it to the address *0x400000*.



7. Run the application on S32G-VNP-RDB2

The following are the steps to run the application on the S32G2 RDB2 board.

1. Insert the SD-Card to the slot on RDB2
2. Set RDB2 boot from external NOR-Flash

   SW10-1=ON,SW10-2=OFF

   SW4 all OFF
3. Connect RDB2 UART0 to PC USB port
4. Launch a Serial terminal on PC, setup the serial port with baudrate *115200*, format *8-n-1*
5. Power on the RDB2. If everything is ok, you will get the A53 booting log on the terminal.
6. After Kernel is up, enter *root* to login.
7. Copy the IPC module files. The following commands needs to be entered in the terminal.

   $mkdir ipc

   $mount /dev/mmcblk0p1 ./ipc

   $cp ./ipc/ipc-shm- ~*

   $umount ./ipc
8. Insert IPC modules
   $insmod ipc-shm-dev.ko
   $insmod ipc-shm-sample_multi-instance.ko
9. Run the IPC example. You will see logs showing IPC message between Linux Kernel and Cortex-M7 cores.

NOTE

CORE4, CORE5, CORE6 are the core ID for Cortex-M7_0, Cortex-M7_1 and Cortex-M7_2.

$echo 1 > /sys/kernel/ipc-shm-sample-instance0/ping
$echo 1 > /sys/kernel/ipc-shm-sample-instance1/ping
$echo 1 > /sys/kernel/ipc-shm-sample-instance2/ping
$dmesg | grep ipc-shm-sample_multi-instance
[462.703716] ipc-shm-sample_multi-instance: starting demo on instance 0...
[462.703739] ipc-shm-sample_multi-instance: INST0 ch 0 >> 19 bytes: SENDING MESSAGES: 1
[462.703753] ipc-shm-sample_multi-instance: INST0 ch 1 >> 32 bytes: #0 HELLO WORLD! from KERNEL
[462.703797] ipc-shm-sample_multi-instance: ch 0 << 19 bytes: REPLIED MESSAGES: 1
[462.703818] ipc-shm-sample_multi-instance: ch 1 << 32 bytes: #0 HELLO WORLD! from CORE 4
[462.703850] ipc-shm-sample_multi-instance: exit demo for instance 0

[470.935633] ipc-shm-sample_multi-instance: starting demo on instance 1...
[470.935653] ipc-shm-sample_multi-instance: INST1 ch 0 >> 19 bytes: SENDING MESSAGES: 1
[470.935667] ipc-shm-sample_multi-instance: INST1 ch 1 >> 32 bytes: #0 HELLO WORLD! from KERNEL
[470.935712] ipc-shm-sample_multi-instance: ch 0 << 19 bytes: REPLIED MESSAGES: 1
[470.935732] ipc-shm-sample_multi-instance: ch 1 << 32 bytes: #0 HELLO WORLD! from CORE 5
[470.935763] ipc-shm-sample_multi-instance: exit demo for instance 1

[476.639610] ipc-shm-sample_multi-instance: starting demo on instance 2...
[476.639630] ipc-shm-sample_multi-instance: INST2 ch 0 >> 19 bytes: SENDING MESSAGES: 1
[476.639644] ipc-shm-sample_multi-instance: INST2 ch 1 >> 32 bytes: #0 HELLO WORLD! from KERNEL
[476.639691] ipc-shm-sample_multi-instance: ch 0 << 19 bytes: REPLIED MESSAGES: 1
[476.639709] ipc-shm-sample_multi-instance: ch 1 << 32 bytes: #0 HELLO WORLD! from CORE 6
[476.639740] ipc-shm-sample_multi-instance: exit demo for instance 2

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Suitability for use in automotive applications — This NXP product has been qualified for use in automotive applications. If this product is used by customer in the development of, or for incorporation into, products or services (a) used in safety critical applications or (b) in which failure could lead to death, personal injury, or severe physical or environmental damage (such products and services hereinafter referred to as "Critical Applications"), then customer makes the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, safety, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. As such, customer assumes all risk related to use of any products in Critical Applications and NXP and its suppliers shall not be liable for any such use by customer. Accordingly, customer will indemnify and hold NXP harmless from any claims, liabilities, damages and associated costs and expenses (including attorneys' fees) that NXP may incur related to customer's incorporation of any product in a Critical Application.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Airfast — is a trademark of NXP B.V.

Altivec — is a trademark of NXP B.V.

CodeWarrior — is a trademark of NXP B.V.

ColdFire — is a trademark of NXP B.V.

ColdFire+ — is a trademark of NXP B.V.

CoolFlux — is a trademark of NXP B.V.

CoolFlux DSP — is a trademark of NXP B.V.

DESFire — is a trademark of NXP B.V.

EdgeLock — is a trademark of NXP B.V.

EdgeScale — is a trademark of NXP B.V.

EdgeVerse — is a trademark of NXP B.V.

eIQ — is a trademark of NXP B.V.

Embrace — is a trademark of NXP B.V.

Freescale — is a trademark of NXP B.V.

GreenChip — is a trademark of NXP B.V.

HITAG — is a trademark of NXP B.V.

ICODE and I-CODE — are trademarks of NXP B.V.

Immersiv3D — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

JCOP — is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Layerscape — is a trademark of NXP B.V.

MagniV — is a trademark of NXP B.V.

Mantis — is a trademark of NXP B.V.

MCCI — is a trademark of NXP B.V.

MIFARE — is a trademark of NXP B.V.

MIFARE Classic — is a trademark of NXP B.V.

MIFARE FleX — is a trademark of NXP B.V.

MIFARE4Mobile — is a trademark of NXP B.V.

MIFARE Plus — is a trademark of NXP B.V.

MIFARE Ultralight — is a trademark of NXP B.V.

MiGLO — is a trademark of NXP B.V.

MOBILEGT — is a trademark of NXP B.V.

NTAG — is a trademark of NXP B.V.

NXP SECURE CONNECTIONS FOR A SMARTER WORLD — is a trademark of NXP B.V.

PEG — is a trademark of NXP B.V.

Plus X — is a trademark of NXP B.V.

POR — is a trademark of NXP B.V.

PowerQUICC — is a trademark of NXP B.V.

Processor Expert — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

QorIQ Qonverge — is a trademark of NXP B.V.

RoadLink — wordmark and logo are trademarks of NXP B.V.

SafeAssure — is a trademark of NXP B.V.

SafeAssure — logo is a trademark of NXP B.V.

SmartLX — is a trademark of NXP B.V.

SmartMX — is a trademark of NXP B.V.

StarCore — is a trademark of NXP B.V.

Symphony — is a trademark of NXP B.V.

Synopsys & Designware — are registered trademarks of Synopsys, Inc.

Synopsys — Portions Copyright © 2021 Synopsys, Inc. Used with permission. All rights reserved.

Tower — is a trademark of NXP B.V.

TriMedia — is a trademark of NXP B.V.

UCODE — is a trademark of NXP B.V.

VortiQa — is a trademark of NXP B.V.

Vybrid — is a trademark of NXP B.V.

Date of release: 11/2022

Document identifier: AN13750