Rev. 1,9/2023

S32G On-demand SMR Verification Usage

by John Li (nxa08200)

本文说明S32G HSE On-demand SMR验证的应用方法，本文演示的示例应用为：

- Secure Bootloader对Linux Bootloader fip.bin的验证。

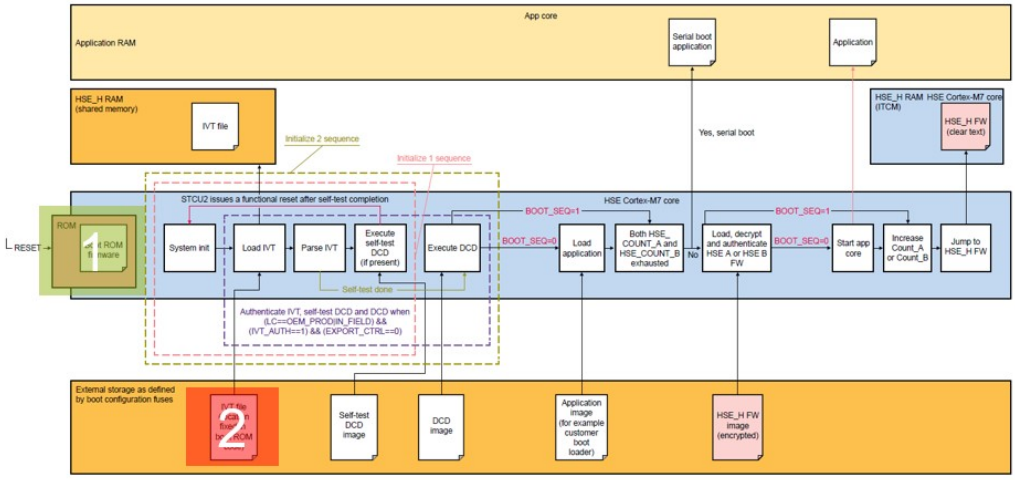
历史	说明	作者
V1	● 创建本文	● John.Li

目录

1 背景说明与参考资料

1.1 背景说明

HSE Secure Boot 的流程图如下：



整个 Secure Boot 过程如下：

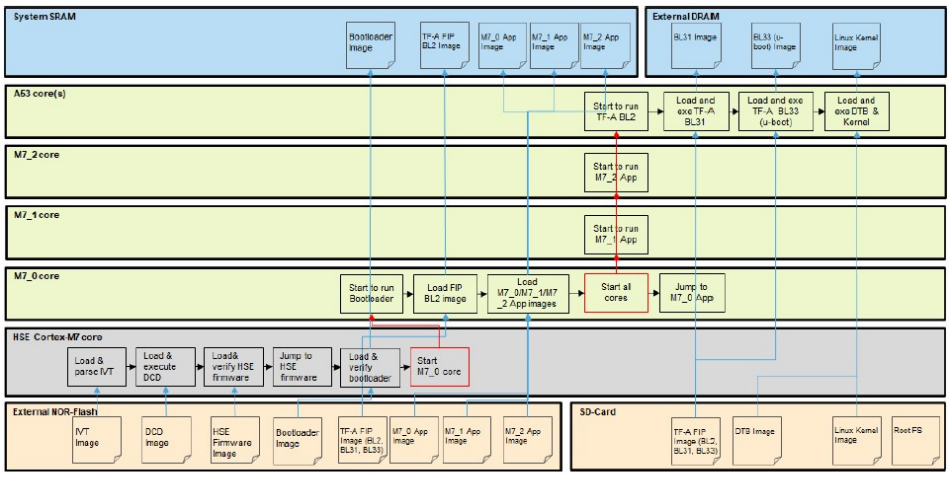


Figure 3. **Secure boot flow**

所以目前的防篡改验证链是：

- HSE ROM code verify IVT。
- HSE FW verify bootloader。

尚缺少：

- Bootloader 调用 HSE 去验证 fip.bin(ATF+Uboot)：Bootloader 是运行在 M7 上的，为了提高 效率，建议使用 HSE 验证。
- Uboot 验证 kernel，(Uboot 调用 HSE，参考 Linux User Manual)。
- Kernel 对文件系统或 disk 的验证，(使用相关内核公开技术，如 DM-Verity…)。

本文的重点在于说明与解决 Bootloader 调用 HSE 去验证 fip.bin(ATF+Uboot)，及启动 ATF+Uboot。

注意：

- **OnDemand SMR verification 可以由 Bootloader 调用，也可以由 Autosar secure app 调用。**
- **并不是只有使用 OnDemand SMR verification 来做 fip.bin 验证的唯一办法，由于 Bootloader 工程可以调用 crypto MCAL 驱动，所以调用驱动来编程验证也是可以的。**
- **一般正常的 Boot 需要先启动一个 Bootloader 来配置资源，所以与 preSMR 有并行关系的 postSMR 实用性没有 OnDemand SMR 灵活可用。**
- **为了避免 HSE/M7 竞争 QSPI NOR 的访问，不使用 HSE 从 QSPI NOR 直接加载 SMR 的方式，而是由 M7 加载到内存后，HSE 从内存中来验证，Bootloader 启动后的 QSPI NOR 访问由 Bootloader 全权负责。**
- **所有的防篡改信任链只考虑防篡改，不考虑加密，防否认之类的，以避免延长过多的启动速度，但本文并未测试 OnDemand SMR 对启动速度的影响。**
- **本文使用 fip.bin 为例说明，对于 M 核镜像的 OnDemand SMR 验证道理与之相同。**

1.2 参考资料

以 S32G2 RDB2 为例：

序号	资料	说明	如何获取
1	Platform_Software_Integration _S32G2_2022_06.exe	包含 bootloader 工程，安装后可参考其文档《Bootloader_UserManual.pdf》Secure Boot 一章	www.nxp.com/s32g
2	AN13750: Enabling Multicore Application on S32G2 using S32G2 Platform Software Integration – Application note	Bootloader 工程说明，包括 Secure Boot 配置说明(XueweiWang)。	www.nxp.com/s32g
3	S32G_Bootloader_V*.pdf	Bootloader 应用手册，包换非	https://community.nxp.com

		Secure Boot 部分(Johnli)	/t5/NXP-Designs-Knowledge -Base/ S32G-Bootloader-Customzition/ta-p/ 1519838
4	S32G_Secure_boot_*.pdf	Secure boot 应用手册及工程，本工程基于此工程之上开发 (Johnli)	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-Secure-boot-chinese-version/ta-p/ 1718083
5	HSE_H/M Firmware Reference Manual	HSE FW 用户手册	www.nxp.com/s32g (需要 secure 访问权限)
6	Secure Boot with HSE V0 - Draft B	Secure Boot 说明文档	www.nxp.com/s32g (需要 secure 访问权限)
7	HSE_DEMOAPP_S32G2_0_1_0_5.exe	HSE Demo App(S32DS 版本)	www.nxp.com/s32g
8	HSE_FW_S32G2_0_1_0_5.exe	HSE FW 安装包，安装后可以查看文档《HSE_FW_H_S32G2XX_0.1.0.5 _HSE_Service_API_ Reference_Manual.pdf》	www.nxp.com/s32g
9	S32G2_LinuxBSP_36.0_User_Manual.pdf	Linux BSP user manual	www.nxp.com/s32g
10	S32G_ATF_BSP32_V*.pdf	ATF 定制说明(Johnli)	https://community.nxp.com/ t5/NXP-Designs-Knowledge-Base/ S32G-ATF-customization-application-doc /ta-p/1450561

2 S32G On-demand SMR Verification 说明

2.1 SMR Verify 的说明

参考文档《HSE_H/M Firmware Reference Manual》了解 SMR 验证细节。

2.2 On-demand SMR Verify

关于 On-demand SMR verify 的部分如下：

当 SMR 条目不配置为链接到 CR 表上后，不会自动运行验证，需要显示调用服务 hseSmrVerifySrv_t 去主动验证，(或是 checkPeriod 不等于 0 的情况下自动触发)。

该服务有两个参数：

- entryIndex：指定要验证的 SMR 的索引。
- Options：SMR 验证选项：

o HSE_SMR_VERIFICATION_OPTION_NONE ：默认的验证方法：如果 SMR 已经从外部闪存加载了，仅在 SRAM 中验证； 如果 SMR 还没有从外部闪存加载过，它加载到 SRAM 存储器中并验证。

o HSE_SMR_VERIFICATION_OPTION_NO_LOAD ： SMR 从外部闪存验证（使用 pSmrSrc 地址），即使指定了 pSmrDest 并且已经加载到 SRAM 存储器中。

o HSE_SMR_VERIFICATION_OPTION_RELOAD - SMR 从外部闪存加载并且验证， 即使它已经加载过了。

此服务还可用于在运行时验证任何 SMR，而不管 SMR 与 CR 表的关系如何。 但是，当要验证的 SMR 在外部 Flash 中时，必须保证主要没有对外部 Flash 的并发访问，所以建议是 Bootloader 将镜像拷贝到 SRAM 后，触发验证服务。另外 OnDemand SMR 安装时不需要配置 CR。

3 环境搭建

先根据文档《S32G_Secure_boot_*.pdf》要求，搭建 secure boot 环境，关掉以下调试开关：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\Bootloader.c

```
// while (1); //johnli stop bootloader //停止继续 boot，方便调试。
```

保留最开始的调试开关：

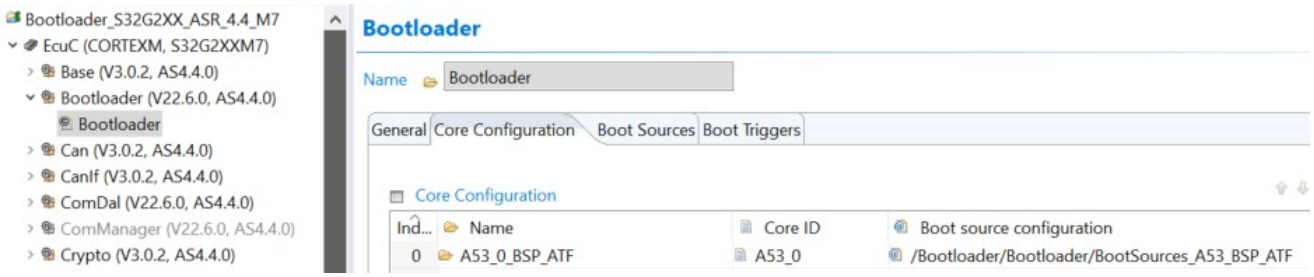
```
volatile int debug;

int main(void)

{…
debug =1;
while(debug); //增加调试开关接口，方便使用lauterbach来跟踪调试。
```

然后 EB 配置修改如下：

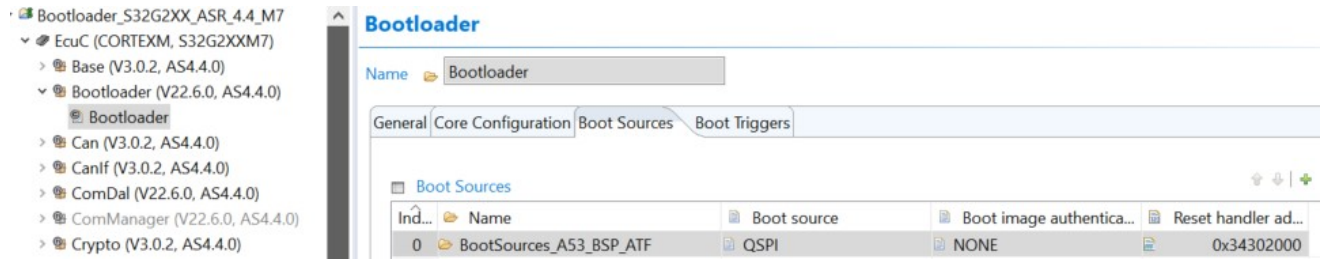
3.1 EB 配置说明

由于只考虑对 A 核 Linux fip.bin 的验证，所以 Bootloader 工程中只配置 A 核的 Boot Source：

1. 打开 Bootloader(…)->Bootloader->Core Configuration:删除其它只保留 A53_0_BSP_ATF:



2. 打开 Bootloader(…)->Bootloader->Boot Sources： 删除其它只保留：BootSources_A53_BSP_ATF，以下有 reset 地址：



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

Added BL2 and DTB to /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/fip.bin successfully

Trusted Boot Firmware BL2: offset=0x100, size=0x4BDC0, cmdline="--tb-fw"

EL3 Runtime Firmware BL31: offset=0x4BEC0, size=0x18239, cmdline="--soc-fw"

Non-Trusted Firmware BL33: offset=0x64100, size=0xAF530, cmdline="--nt-fw"

SOC_FW_CONFIG: offset=0x113640, size=0x6D84, cmdline="--soc-fw-config"

CREATE /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/bl2_w_dtb_size

CREATE /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/fip.cfgout

MKIMAGE /opt/samba/nxa08200/S32G/BSP36/standalone/arm-trusted-firmware/build/s32g274ardb2/release/fip.s32

Image Layout

DCD: Offset: 0x200 Size: 0x1c

IVT: Offset: 0x1000 Size: 0x100

AppBootCode Header: Offset: 0x1200 Size: 0x40

Application: Offset: 0x1240 Size: 0x4c000

Boot Core: A53_0

IVT Location: SD/eMMC

Load address: 0x342fb140

Entry point: 0x34302000

ll build/s32g274ardb2/release/fip.*

-rw-r--r-- 1 nxa08200 nxp 1156096 Sep 14 15:46 build/s32g274ardb2/release/fip.bin

-rw-r--r-- 1 nxa08200 nxp 1160768 Sep 14 15:46 build/s32g274ardb2/release/fip.s32

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

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

Load image at address (RAM)= 0x342fb140 //load address。

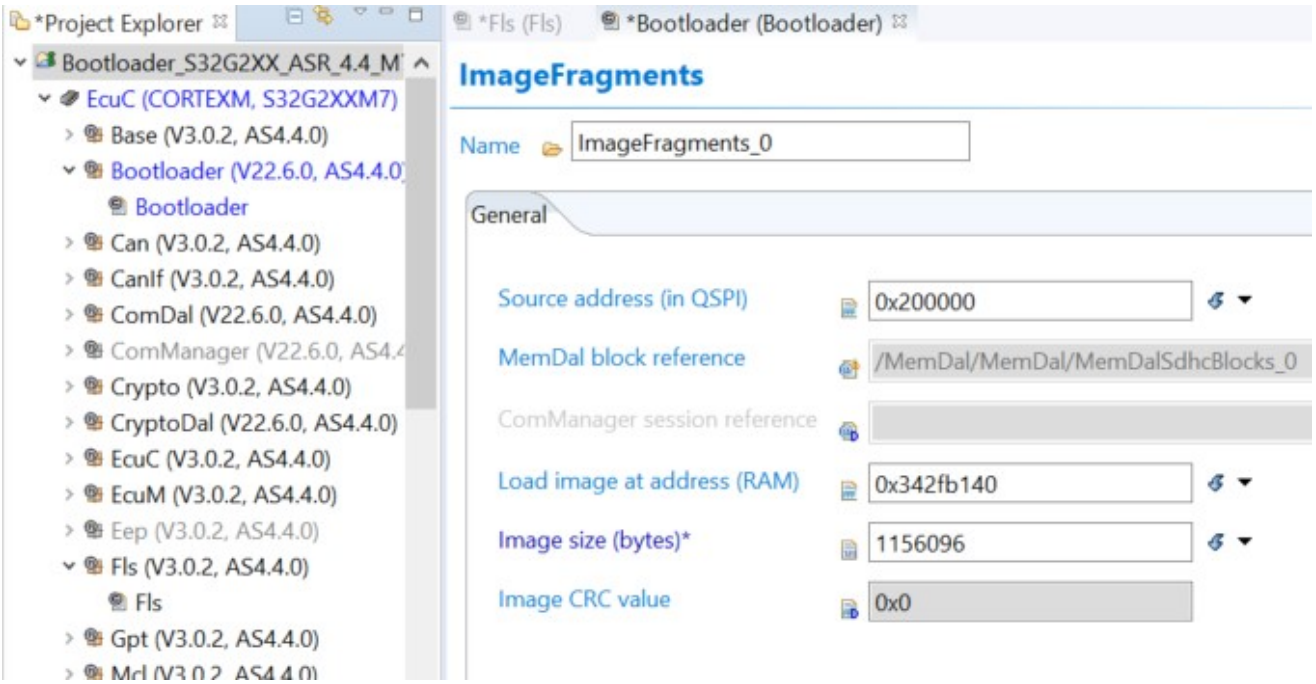
注意注释说明：**“The address to load the image into RAM.**

NOTE !: The start address must be multiple of 8 if you choose CRC32 authentication method, otherwise must be multiple of 64.”

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

Image size (bytes)= 1156096 >(0x113640+0x6D84)= 1156036，而且 1156096 可以被 64 整除。

另外 Source address (in QSPI)*为了避免和 secure boot 的 SYS-IMG published 重叠，修改为 0x200000：



3.2 ATF 编译说明

由于将整个 fip.bin(包括 ATF+Uboot)作为一个整体来验证，所以考虑将整个 fip.bin 放在 qspi nor 中，而不是如 Bootloader 工程所编译的 SDcard/eMMC 版本，根据文档《S32G2_LinuxBSP_36.0_User_Manual.pdf》要求，编译为非自加载模式，文档说明如下：

- Configure TF-A to read the FIP image from a defined memory address instead of reading it from the boot source storage (QSPI or MMC).

  The memory location of the FIP image is set using the **FIP_MEMORY_OFFSET** compilation parameter and can be used in two cases:

  1. Loading the FIP image from a predefined memory location instead of a storage device
  2. BL2 in-place execution from FIP image

  In the first case, TF-A uses the parameter to locate the image header and load the following stages from the given location, and in the second, the parameter contributes to in-place execution of the BL2 stage. In both cases, it is assumed the FIP image is copied to the specified memory address from outside TF-A. For example, the FIP image can be copied to specified SRAM address from the M7 bootloader configured as boot target, before starting BL2. This is how **FIP_MEMORY_OFFSET=<memory address>** is used for the first use case:

```
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu- \
    ARCH=aarch64 PLAT=<plat> BL33=<path-to-u-boot-nodtb.bin> \
    FIP_MEMORY_OFFSET=0x34520000
```

  In this case, BL2 is not executed in-place as part of the FIP image and must be stored at a location that does not overlap with the FIP image, which is expected to be loaded at **FIP_MEMORY_OFFSET**.

  This feature can be paired with `BL2_BASE` to prepare the resulting image and BL2 stage for in-place execution. The in-place execution of the BL2 requires a three-step compilation:

  1. Compile the TF-A using a custom base address for the BL2 stage if the default value is not suitable. For this purpose, the `BL2_BASE` must be appended to the compilation command.

```
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu- \
    ARCH=aarch64 PLAT=<plat> BL33=<path-to-u-boot-nodtb.bin> \
    BL2_BASE=0x34100000
...
Boot Core:      A53_0
IVT Location:   SD/eMMC
Load address:   0x340f8bc0
Entry point:    0x34100000
...
```

  2. Obtain the load address of the FIP image. In the above example, it is 0x340f8bc0 (see `Load address`)
  3. Recompile the TF-A using `FIP_MEMORY_OFFSET` set to identified load address.

```
make CROSS_COMPILE=/path/to/your/toolchain/dir/bin/aarch64-none-linux-gnu- \
    ARCH=aarch64 PLAT=<plat> BL33=<path-to-u-boot-nodtb.bin> \
    BL2_BASE=0x34100000 \
    FIP_MEMORY_OFFSET=0x340f8bc0
```

实际编译命令如下，编译log参考上节。

make PLAT=s32g274ardb2 BL33=../u-boot/u-boot-nodtb.bin BL2_BASE=0x34302000 FIP_MEMORY_OFFSET=0x342fb140 LDFLAGS=""

3.3 镜像烧写

根据文档《S32G_Secure_boot_*.pdf》要求，制造 IVT 镜像，然后根据此文档烧写镜像，注意以下几点：

- 先再点击Erase memory range…，选择0x0-0x500000。(强调需要先擦除，这样fip.bin圆整到64 BYTE尾部多余的部分全是0xFF，这样可以防止错误)。

  然后再如下烧写镜像：
- 使用flash tools 烧写bootloader 镜像到QSPI 中：

  点击 Upload file to device…，将“secureboot_odsmr.bin”烧写到地址 0x0 处。
- 使用flash tools 烧写A53 fip.bin 到QSPI 中：

  点击Upload file to device…，将“fip.bin”烧写到地址0x200000 处，烧写地址参考之前 Bootloader MCAL 配置的QSPI source address，烧写时注意是烧写fip.bin 文件，这个是不带 IVT 头的A53 Bootloader fip.s32。

  A53 Linux部分使用Linux PC烧写Linux镜像到SDcard。

4 Bootloader 代码开发

4.1 OnDemand SMR install

1. 获得 fip.bin 相关信息

调用：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\src\BootloaderSpecific.c：

```
StatusType Bl_ConfigureSecureBoot(void)
{…
    //johnli add for on-demand smr
   uint32_t u32FipSize;
   uint32_t u32FipRamAddr;
   uint32_t u32FipFlashAddr;
   uint8_t Bl_FipTag[32];
    uint32_t u32FipTagSize = 32;
    //end
  if (E_NOT_OK == Bl_IsSecureBootActive())
```

```
  {…
  Bl_GetHSEFwParams(&u32HseFwImageFlashAddr, &u32HseBackupFwImageFlashAddr);
```

实现：

```
static void Bl_GetFipParams(uint32_t *pFlashAddr, uint32_t *pRamAddr,
                            uint32_t *pSize)
{ // bootApplications 是由 EB 配置的 boot source,由于我们只配置了一个 A53 的 fip.bin bootapp 和一个 image fragments，所以序号为 0
  *pFlashAddr =bootApplications[0].pImageFragments[0].u32FlashAddress;
  *pRamAddr = bootApplications[0].pImageFragments[0].u32DestinationAddress;
  *pSize = bootApplications[0].pImageFragments[0].u32Size;
}
```

2. 安装 OnDemand SMR(由于 OnDemand SMR 不需要开机验证，所以不用配置 CR)。

调用：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\src\BootloaderSpecific.c：

```
StatusType Bl_ConfigureSecureBoot(void)
{…
   if (E_OK == status)
    { //同样为镜像生成 CMAC，注意此处用了和 bootloader 同样的密钥，也可以使用不同的密钥，需要自行配置
        status = CryptoDal_GenerateCmac(
        (uint8_t *) u32FipFlashAddr, &Bl_FipTag[0],
        BL_SEC_BOOT_KEY_INDEX, u32FipSize, &u32FipTagSize);
         //配置 OnDemand SMR
        Bl_ConfigureFipSMR(&SMR_CR_Config, u32FipFlashAddr,
                           u32FipRamAddr, u32FipSize,
                           &Bl_FipTag[0], &u32FipTagSize);
         //安装 OnDemand SMR      ，不安装 CR
        status = CryptoDal_SMR_Install(&SMR_CR_Config);
    }
```

实现：

```
static void Bl_ConfigureFipSMR(
  CryptoDal_SMR_CR_EntryType *pSMR_CR_Config, uint32_t u32AppFlashAddress,
```

```
    int32_t u32AppRamAddr, uint32_t u32AppSize, uint8_t *pTag,
    uint32_t *pTagSize)
{
    pSMR_CR_Config->u32FlashStorageAddr = u32AppFlashAddress;
    pSMR_CR_Config->u32RamDestAddr = u32AppRamAddr;
    pSMR_CR_Config->u32AppSize = u32AppSize;
    pSMR_CR_Config->pTag = pTag;
    pSMR_CR_Config->pTagSize = pTagSize;
    pSMR_CR_Config->u8KeyIndex = BL_SEC_BOOT_KEY_INDEX;
    pSMR_CR_Config->u8EntryIndex = 2; //注意需要修改 SMR 的 Index
    pSMR_CR_Config->u8CoreID = HSE_APP_CORE3; // |  3   | A53_0  但实际上这个参数和配置 CR 有关，用不上
}
```

实现 CryptoDal_SMR_Install 函数可以参考 CryptoDal_SMR_CR_Install 函数，要去掉 CR 安装部分。

另外，为了防止 HSE 和 M7 同时访问 QSPI NOR，所以我们在验证时采用先由 Bootloader 拷贝 fip.bin 到 SRAM 中，再触发验证服务的方式，而不是 HSE 主动拷贝的方式。所以相应修改 SMR 的安装项(见红字部分)：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\src\CryptoDal.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE)
CryptoDal_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST)
pSMR_CR_Config)
{
    VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
    if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
    {
#if CRYPTODAL_USE_CRYPTO
        eRetVal = CryptoDal_Crypto_SMR_Install(pSMR_CR_Config);
#endif /* CRYPTODAL_USE_CRYPTO */
    }
    return eRetVal;
}
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\include\CryptoDal.h

FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config);

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\src\CryptoDal_Crypto.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config)
{
  VAR(hseSmrEntry_t, AUTOMATIC) SmrEntry;
  VAR(hseCrEntry_t, AUTOMATIC) CoreResetEntry = {0};
  VAR(hseSrvDescriptor_t, AUTOMATIC) HseSrvDescriptor;
  VAR(hseSmrEntryInstallSrv_t, AUTOMATIC) SmrInstallRequest;
  VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_OK;
  VAR(uint8, AUTOMATIC) u8CryptoKeyIndex;
  /* Fetch internal decryption engine key reference */
  u8CryptoKeyIndex = CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8EncryptionKeyAlternateRef;
#if 0 //由于之前的调用已经注入了 KEY，此处删除掉
  /* Register key in using the alternate slot. This descriptor shall reside in NVM for verification usage */
  eRetVal = Crypto_KeyElementSet(u8CryptoKeyIndex, CRYPTO_KEY_MATERIAL_U32, CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8CryptoKey,
                  CryptoDal_Crypto_pGlobalConfigPtr[pSMR_CR_Config->u8KeyIndex].u8EncryptionKeyLen);
#endif
  MemLib_MemSet(&SmrEntry, 0, sizeof(SmrEntry));
  /* Configure SMR entry */
  SmrEntry.pSmrSrc                    = (HOST_ADDR)pSMR_CR_Config->u32RamDestAddr;// SMR 安装表项的源地址设置为 SRAM 地址
  SmrEntry.pSmrDest                   = (HOST_ADDR)NULL; //dest sram address=null, means no flash to memory copy 目的地址没有，无 copy 动作。
  SmrEntry.smrSize                    = pSMR_CR_Config->u32AppSize;
  SmrEntry.checkPeriod                = 0;
  SmrEntry.configFlags                = HSE_SMR_CFG_FLAG_QSPI_FLASH;
  SmrEntry.authKeyHandle              = CRYPTODAL_SMR_CMAC_KEY_HANDLE(u8CryptoKeyIndex);
  SmrEntry.authScheme.macScheme.macAlgo          = HSE_MAC_ALGO_CMAC;
  SmrEntry.authScheme.macScheme.sch.cmac.cipherAlgo  = HSE_CIPHER_ALGO_AES;
  SmrEntry.pInstAuthTag[0]            = 0;
```

```
    SmrEntry.pInstAuthTag[1]                    = 0;

    /* Create a SMR install request */
    SmrInstallRequest.accessMode                = HSE_ACCESS_MODE_ONE_PASS;
    SmrInstallRequest.entryIndex                = pSMR_CR_Config->u8EntryIndex;
    SmrInstallRequest.pSmrEntry                 = (HOST_ADDR)&SmrEntry;
    SmrInstallRequest.pSmrData                  = (HOST_ADDR)pSMR_CR_Config->u32FlashStorageAddr; //安装时都是从 flash 中读出。验证时从 SRAM 中获得
    SmrInstallRequest.smrDataLength             = pSMR_CR_Config->u32AppSize;
    SmrInstallRequest.pAuthTag[0]               = (HOST_ADDR)pSMR_CR_Config->pTag;
    SmrInstallRequest.authTagLength[0]          = *((uint32_t*)pSMR_CR_Config->pTagSize);

    MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));
    HseSrvDescriptor.srvId                  = HSE_SRV_ID_SMR_ENTRY_INSTALL;
    HseSrvDescriptor.hseSrv.smrEntryInstallReq = SmrInstallRequest;

    if (E_OK == eRetVal)
    {
        if(HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
        {
            eRetVal = E_NOT_OK;
        }
    }
    return eRetVal;
}
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\include\CryptoDal_Crypto.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_SMR_Install(P2VAR(CryptoDal_SMR_CR_EntryType, AUTOMATIC, CRYPTODAL_CONST) pSMR_CR_Config);
```

4.2 OnDemand SMR verify

按前文所说：先将 fip.bin 拷贝到 SRAM 中，再触发 OnDemand SMR 验证。

调用：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\generic\src\Bootloader.c：

```
StatusType Bl_BootApplications(void)
{…
  Bl_LoadApplication(u8Index); //先加载 fip.bin 到 SRAM 中
|-> Bl_FetchApplication
|   |-> Bl_LoadAndAuthFromQspi
|   |   |-> Bl_TransferImageFromQspi(u8ApplicationId, u8FragmentIdx);
//如下，此函数正是调用 QSPI NOR DMA 驱动将 fip.bin 从 flash 地址拷贝到 sram 地址：
static StatusType Bl_TransferImageFromQspi(uint8 u8ApplicationId,
                                           uint8 u8FragmentIdx)
{
  StatusType Status = E_NOT_OK;
  uint32 u32StorageAddress = bootApplications[u8ApplicationId]
                             .pImageFragments[u8FragmentIdx]
                             .u32FlashAddress;
  uint32 u32DestinationAddress = bootApplications[u8ApplicationId]
                                 .pImageFragments[u8FragmentIdx]
                                 .u32DestinationAddress;
  uint32 u32ChunkSize = BL_ALIGN_64B(bootApplications[u8ApplicationId]
                                     .pImageFragments[u8FragmentIdx]
                                     .u32Size);
  /* Start the DMA transfer with the 64-byte transfer size configuration. */
  Bl_StartDmaTransfer(u32StorageAddress, u32DestinationAddress,
                      BL_DMA_SIZE_64B, u32ChunkSize);
  /* Wait for DMA engine to transfer the application image */
  if (E_OK == Bl_WaitApplicationFetch())
  {
    Status = E_OK;
  }
  return Status;
}
```

```
…
    if (E_OK == Status)
    {
#if (BL_SYNCHRONIZED_BOOT == STD_ON)
#if 1
//如果 OnDemandSMR 验证失败，则停止，否则正常启动
if(E_NOT_OK == CryptoDal_OnDemandSMR_Verify(2))
{
    while(1);
}
else
{
    Bl_StartAllApplications();
}

#else
     Bl_StartAllApplications();
#endif
```

实现：

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\src\CryptoDal.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_OnDemandSMR_Verify(VAR(uint8, AUTOMATIC) entryIndex)
{
    VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_NOT_OK;
    if (NULL_PTR != CryptoDal_pGlobalConfigPtr)
    {
#if CRYPTODAL_USE_CRYPTO
        eRetVal = CryptoDal_Crypto_OnDemandSMR_Verify(entryIndex);
#endif /* CRYPTODAL_USE_CRYPTO */
    }
    return eRetVal;
}
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\generic\include\CryptoDal.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_OnDemandSMR_Verify(VAR(uint8, AUTOMATIC) entryIndex);
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\src\CryptoDal_Crypto.c

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_OnDemandSMR_Verify(VAR(uint32, AUTOMATIC)entryIndex)
{
        VAR(hseSrvDescriptor_t, CRYPTODAL_VAR) HseSrvDescriptor;
        VAR(Std_ReturnType, AUTOMATIC) eRetVal = E_OK;
        /* Clear previous request */
        MemLib_MemSet(&HseSrvDescriptor, 0, sizeof(HseSrvDescriptor));
        /* Fill the service descriptor */
         HseSrvDescriptor.srvId                                    = HSE_SRV_ID_SMR_VERIFY;
         HseSrvDescriptor.hseSrv.smrVerifyReq.entryIndex = entryIndex;
        if (E_OK == eRetVal)
        {
                if (HSE_SRV_RSP_OK != CryptoDal_Crypto_HseSrv_Request(&HseSrvDescriptor))
                {
                        eRetVal = E_NOT_OK;
                }
        }
        return eRetVal;
    }
```

C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\bsw\dal\cryptodal\platforms\S32G2XX\include\CryptoDal_Crypto.h

```
FUNC(Std_ReturnType, CRYPTODAL_APP_CODE) CryptoDal_Crypto_OnDemandSMR_Verify(VAR(uint32, AUTOMATIC)entryIndex);
```

5 测试

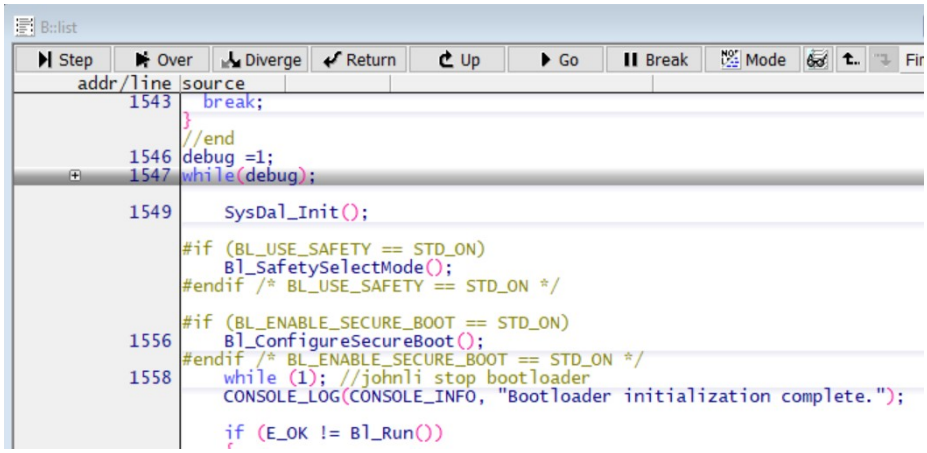
将 RDB2 板设置为 QSPI NOR 正常启动，SDcard 放入 Linux 镜像 fsl-image-auto-s32g274ardb2.sdcard，插入 SDcard 槽中，连接上串口和 Jtag 接口，上电启动。

5.1 Lauterbach 跟踪

启动后使用 Lauterbach 运行脚本：C:\NXP\Integration_Reference_Examples_S32G2_2022_06\code\framework\realtime\swc\bootloader\platforms\S32G2XX\build\cmm\connect_s32gxx_m7.cmm 连接上板子，板子会停在：

```
while(debug);
```

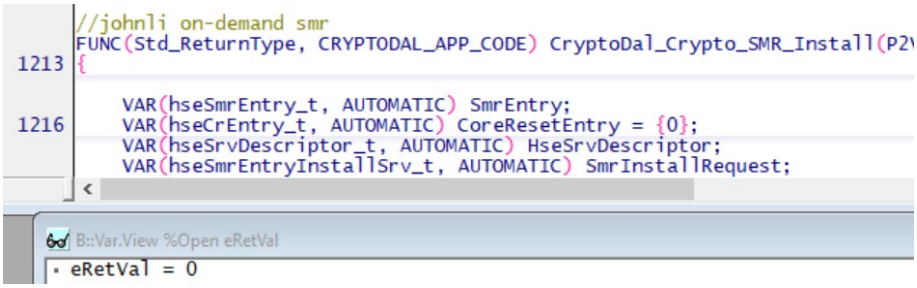
处，然后切换代码模式为源代码模式，就可以使用 Lauterbach 跟踪了：



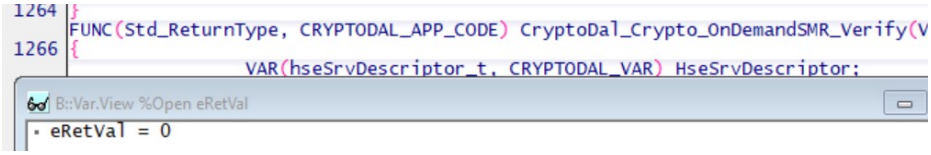
双击 debug，将 debug=1 修改为 debug=0，就可以继续往下运行了。

首先，确认一下 Bl_GetFipParams 获得的 bootApplications 相关参数是否正确：

其次，确认一下 OnDemand SMR 的安装是否成功：



Bl_ConfigureSecureBoot 配置结束后，点击 go 运行，会重启，关闭并重新打开 lauterbach 连接 RDB2 板，修改调试开关继续运行，确认一下 OnDemand SMR 的验证是否成功：



最后，确认一下 Linux Boot 是否成功：

Lauterbach 的 Trace32 界面中直接点 go 运行，检查串口消息看是否 ATF/Uboot/Linux 正常启动了：

NOTICE: Reset status: Power-On Reset

NOTICE: BL2: v2.5(release):bsp36.0-2.5-dirty

NOTICE: BL2: Built : 07:47:27, Sep 14 2023

NOTICE: BL2: Booting BL31

U-Boot 2020.04 (Jul 27 2023 - 10:10:14 +0800)

CPU: NXP S32G274A rev. 2.0

Model: NXP S32G274A-RDB2

DRAM: 3.5 GiB

MMC: FSL_SDHC: 0

Loading Environment from MMC... OK

Configuring PCIe0 as RootComplex

PCIe0: Failed to get link up

PCI: Failed autoconfig bar 20

PCI: Failed autoconfig bar 24

In: serial@401c8000

Out: serial@401c8000

Err: serial@401c8000

Board revision: RDB2/GLDBOX Revision C

…

fixup: pfe2 set to 00:01:be:be:ef:33

Starting kernel ...

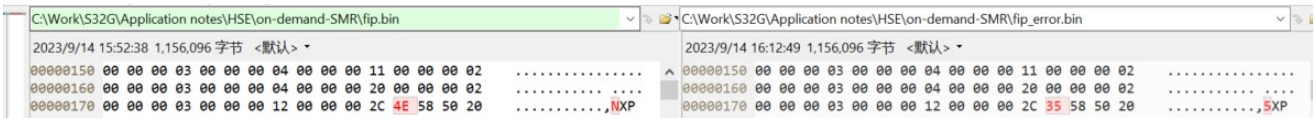
[0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd034]

[0.000000] Linux version 5.15.96-rt61-dirty (nxa08200@lsv11049.swis.cn-sha01.nxp.com) (aarch64-fsl-linux-gcc (GCC) 10.2.0, GNU ld (GNU Binutils) 2.35.1) #2 SMP PREEMPT Tue Aug 1 10:08:31 CST 2023

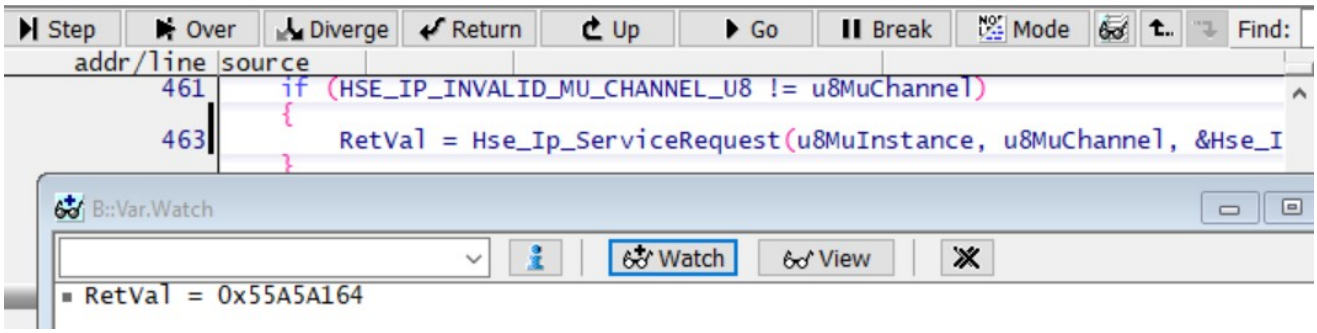
…

5.2 Fip.bin 破坏实验

将 fip.bin 中间某个 BYTE 修改掉：



然后再重新烧写到 QSPI NOR 中，启动后，同样确认一下 OnDemand SMR 的验证是否成功：

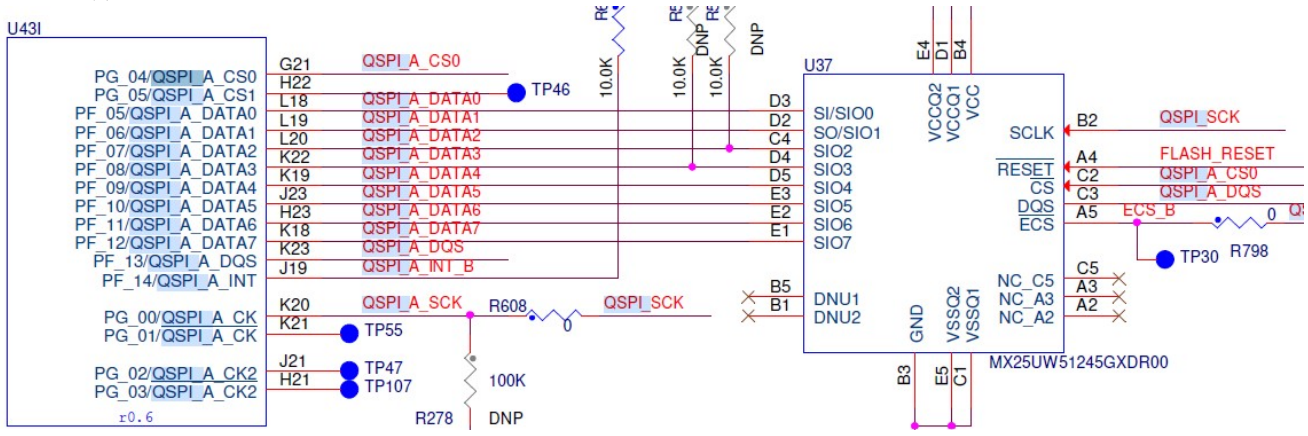


#define HSE_SRV_RSP_VERIFY_FAILED ((hseSrvResponse_t)0x55A5A164UL) /**< @brief HSE signals that a verification request fails (e.g. MAC and Signature verification). */

5.3 硬件确认

为了避免HSE和M7竞争QSPI NOR的访问权，所以我们的配置要保证在正常secure boot时，验证时HSE不访问QSPI NOR，而在安装时，因为只是运行一次，而且 理论上应该是CPU也会在等待安装，所以安装时HSE可以访问QSPI NOR。

如下硬件原理图：



量测电阻R608(在RDB2板背面，靠近S32G2，在S32G2和QSPI NOR之间)，可以发现在执行函数：

CryptoDal_SMR_Install(&SMR_CR_Config);

时，有信号 。
在执行函数：
CryptoDal_OnDemandSMR_Verify(2)
时，无信号，满足设计要求。