MA35D1 TF-A FIP 映像备份功能

NuMicro® 64 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码通过 TF-A 中的 BL2 实现 FIP 映像的备份功能，确保在主要映像坏损的情形下仍能成功启动。
BSP 版本	Linux-5.10.x
开发平台	NuMaker-HMI-MA35D1 V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本范例代码实作了 Trusted Firmware-A (TF-A) 的备份功能，当 TF-A 中的 BL2 检测到 FIP 映像加载失败或者安全启动的签章验证失败时，会尝试加载备份区域的映像来完成系统启动。

TF-A 是 Arm® 官方提供的引导程序框架，用来实作芯片的安全开机流程，依据 Platform Security Architecture (PSA) 定义的架构，包含下列运行时间：

- BL1 (ROM code) ：芯片开机后第一个执行的 bootloader，MA35D1 实现于内部的 Mask ROM 当中，不可修改。
- BL2 (Trusted Bootloader) : 由 BL1 加载与验证，负责后续映像（如 FIP、SCP）的载入与验证。
- BL31 (EL3 Runtime Firmware)：负责提供安全监控与 context switch 的功能。
- BL32：执行于 Secure World 的可信任执行环境，MA35D1 采用开源的 OP-TEE 。
- BL33 : Non-secure World 的 bootloader，MA35D1 采用 U-Boot。

Firmware Image Package (FIP) 是一种容器格式，用以打包多个 TF-A 映像如 BL31、BL32 以及 BL33，供 BL2 一次验证与载入。其结构定义在 fiptool 工具中。

BL2 是 ARM TF-A 架构中非常关键的开机阶段。它由 BL1 调用执行，主要负责以下任务：

- 加载 FIP(Image List Search) 中的映像档并通过 ECDSA 算法验证。
- 确保 Secure Boot 可信任链（Chain of Trust）持续成立。
- 初始化必要硬件与内存控制器（如 DDR）。
- 启动下一阶段的映像（BL31/BL32/BL33）。

1.1 原理

MA35D1 完整的启动流程如下图，每次上电都由 Mask ROM 启动，当侦测到”secure boot”功能开启时，第一阶段由 Mask ROM 对 BL2 做检验，通过验证 BL2 映像附带的 ECDSA P-256 签章，来确认该映像的完整性以及真实性，验证成功后将 BL2 加载到系统指定的内存地址上，便开始由 BL2 接管后续 FIP 映像的加载过程。

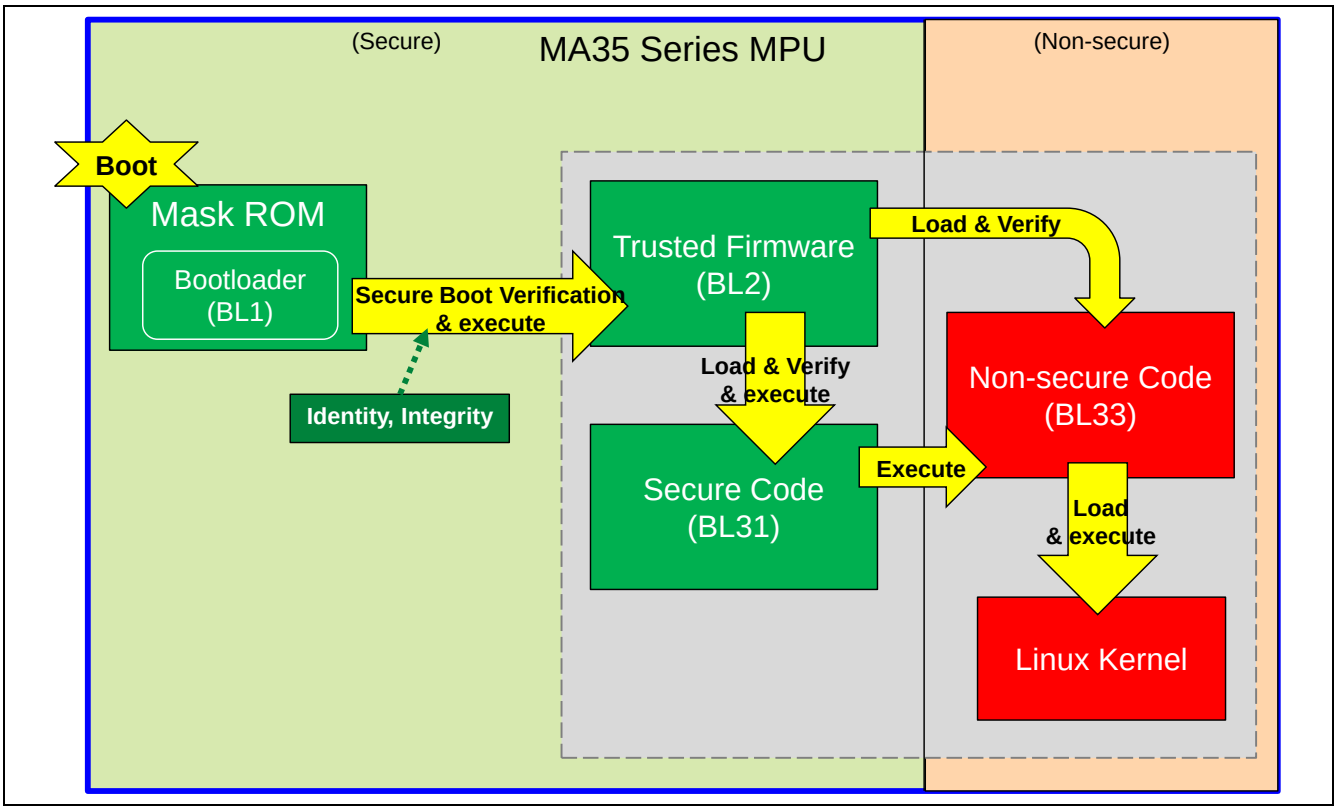


图 1-1 MA35D1 启动流程

BL2 加载 FIP 映像的过程可分成两个阶段来看，第一阶段将存放在外部存储设备上的映像文件，分别加载到内存中的指定地址，若外部存储设备缺少映像文件相关信息，则判断启动失败，改由预先设定好的备份地址区域，提取备份映像进行加载。

当成功将 FIP 加载到 DDR 以及 SRAM 上后，BL2 下一阶段开始对 FIP 映像做安全启动检验，根据 Image ID 决定要验证的 FIP 映像，再通过 ECDSA P-256 算法来验证该映像(BL31、BL32、BL33)的真实性以及完整性。若验证失败则会回传错误旗标，BL2 将重新进行第一阶段工作，将备份区域的 FIP 映像加载到 DDR 以及 SRAM 中，再次进行安全启动检验。示意图可参考如图 1-2:

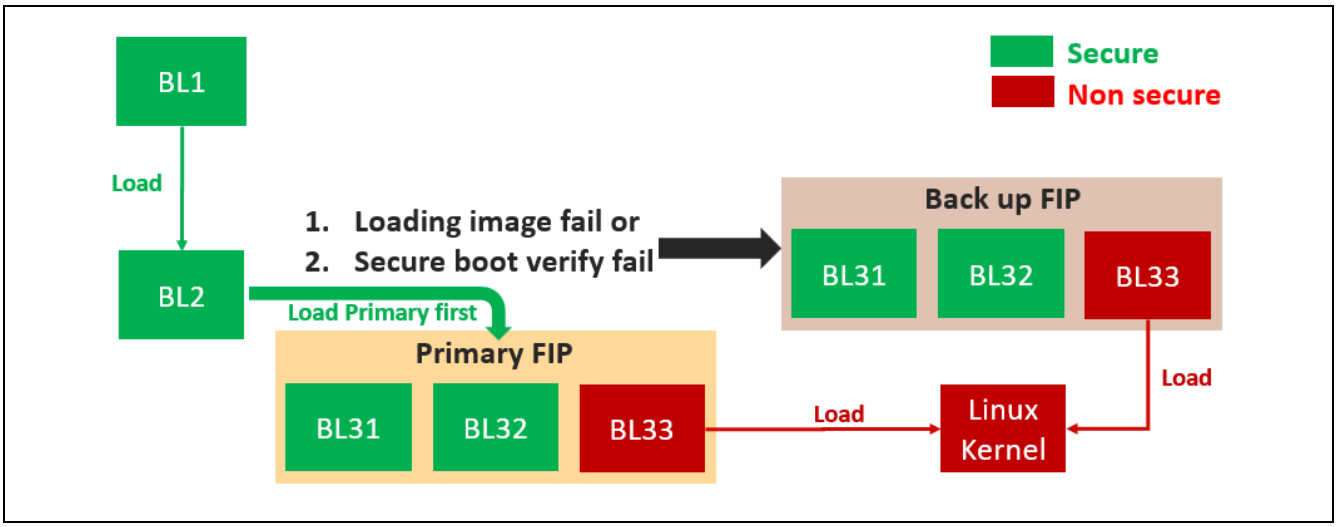


图 1-2 FIP 映像加载流程

MA35D1 的个别映像档，在编译过程会根据启动设备的 pack.json，被 NuWriter 打包成单一映像，其中包含个别映像的烧录信息，通过 NuWriter 烧录到外部存储时会根据相关信息，将 BL2、FIP、Kernel 以及文件系统写入到对应的位置。范例程序中修改了 pack.json 档，将 FIP 备份映像一起打包进 pack.image 中，当主要 FIP 映像加载失败时，便会将备份 FIP 加载到 DDR 上重新验证。

以下以 SD card 作为外部存储示例，pack-sdcard 映像架构如图 1-3 所示。

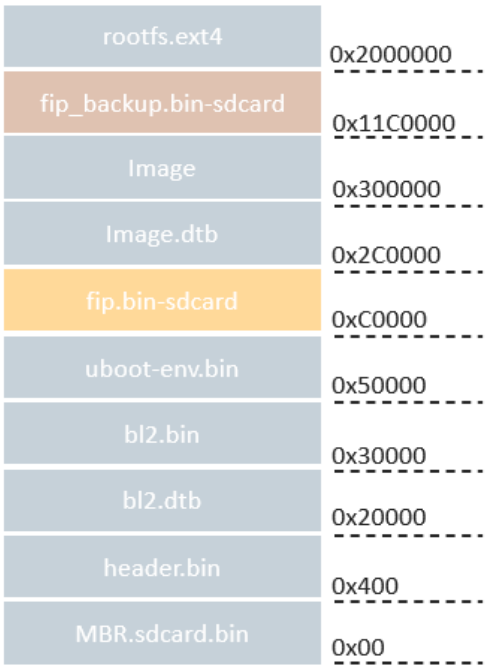


图 1-3 pack-sdcard 架构示意图

1.2 执行结果

本范例代码执行结果如图 1-4 所示。第一次执行前将主要 FIP 以外的映像档，烧录于外部储存设备上，MA35D1 启动时，当 BL2 被成功加载后便会根据 FIP 信息寻找映像位置，由于主要 FIP 并没有烧录到储存设备上，在显示加载失败的信息后，便会尝试从备份 FIP 映像位置读取。

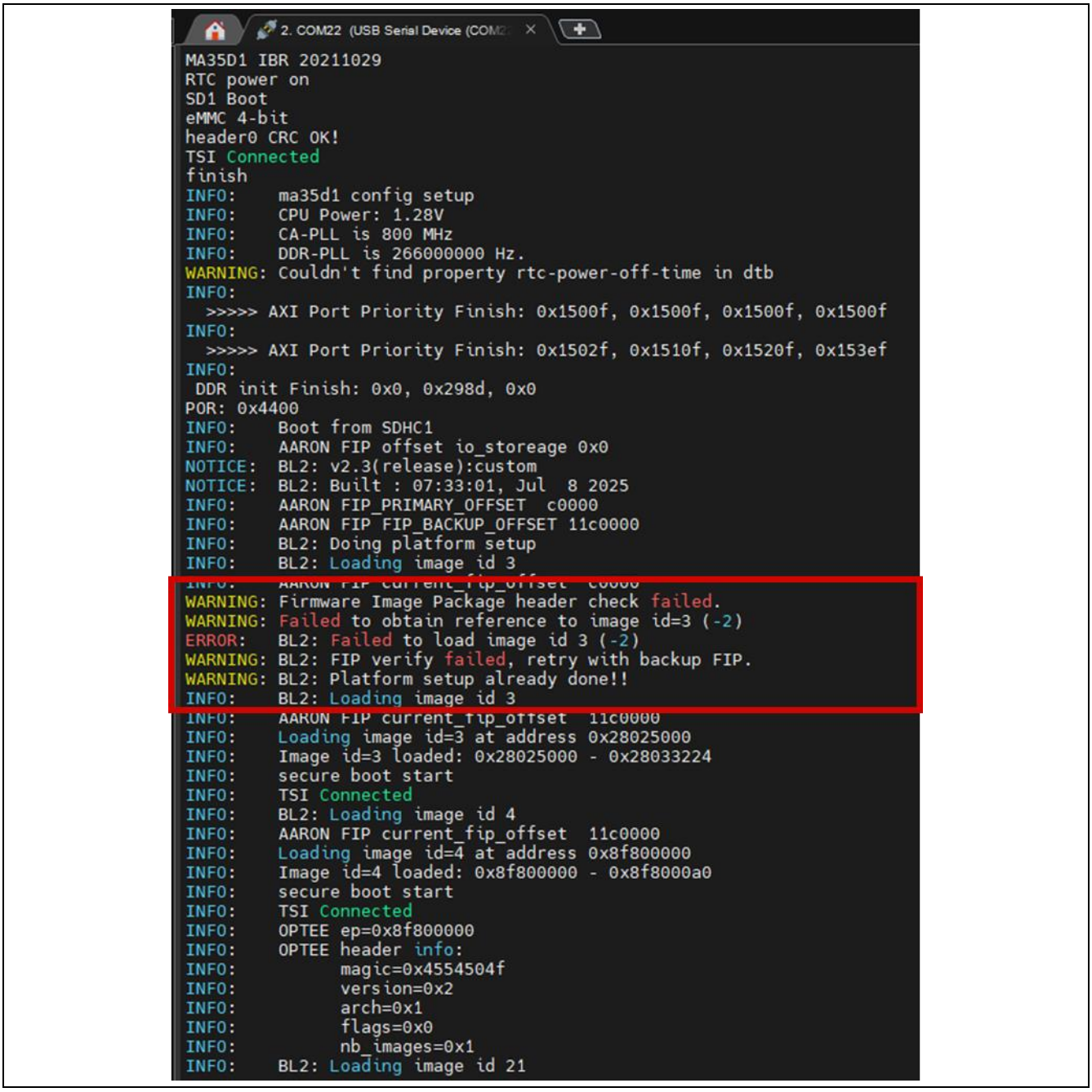


图 1-4 范例执行结果

第二次执行测试，进行验证签章验证失败的加载动作，执行前将带有错误签章的 FIP 映像烧录到主要位置。当 BL2 成功加载主要区域的 FIP 映像后，开始进行第二阶段的签章验证，由于主要 FIP 的签章是错误的，在验证失败后会尝试从备份 FIP 映像位置读取，并再次进行验证。

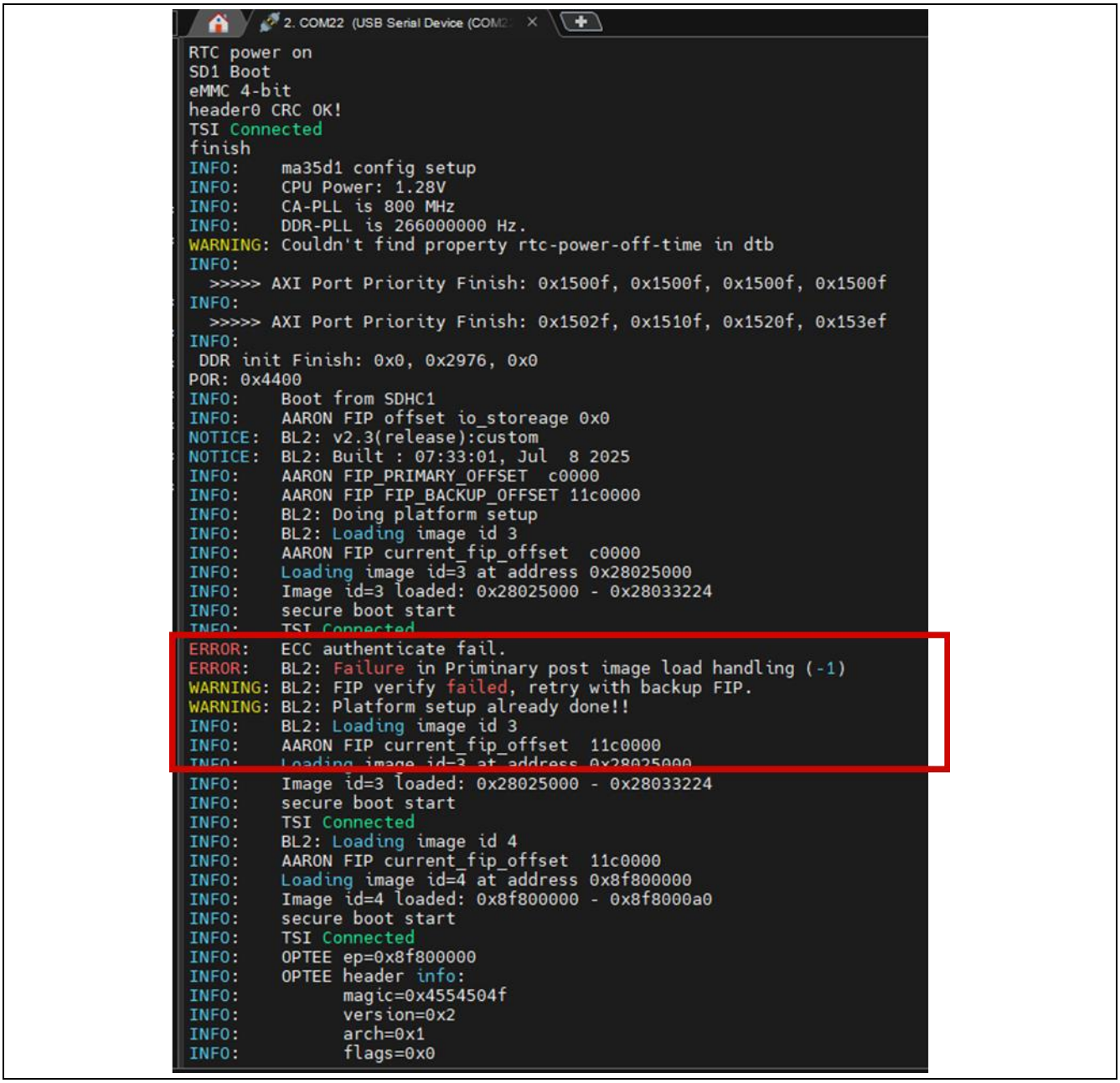


图 1-5 范例执行结果

2. 代码介绍

本范例主要通过修改 TF-A 中的 BL2 来实现备份功能，BL2 运行过程中会通过 bl2_load_images() 进行 FIP 映像的加载以及验证。使用该范例前，请先确认系统可以成功运行 MA35D1 安全启动功能。

范例拆成两部分来理解，第一段:

```
int plat_get_image_source(unsigned int image_id, uintptr_t *dev_handle, uintptr_t
*image_spec)
```

通过调用该函数来取得下列两部分信息:

- **dev_handle**：代表储存设备的存取接口，例如 SD card、NAND flash、SPI NAND flash 等。
- **image_spec** ：指定映像在该设备上的位置信息，通常包含 offset、大小、格式等。

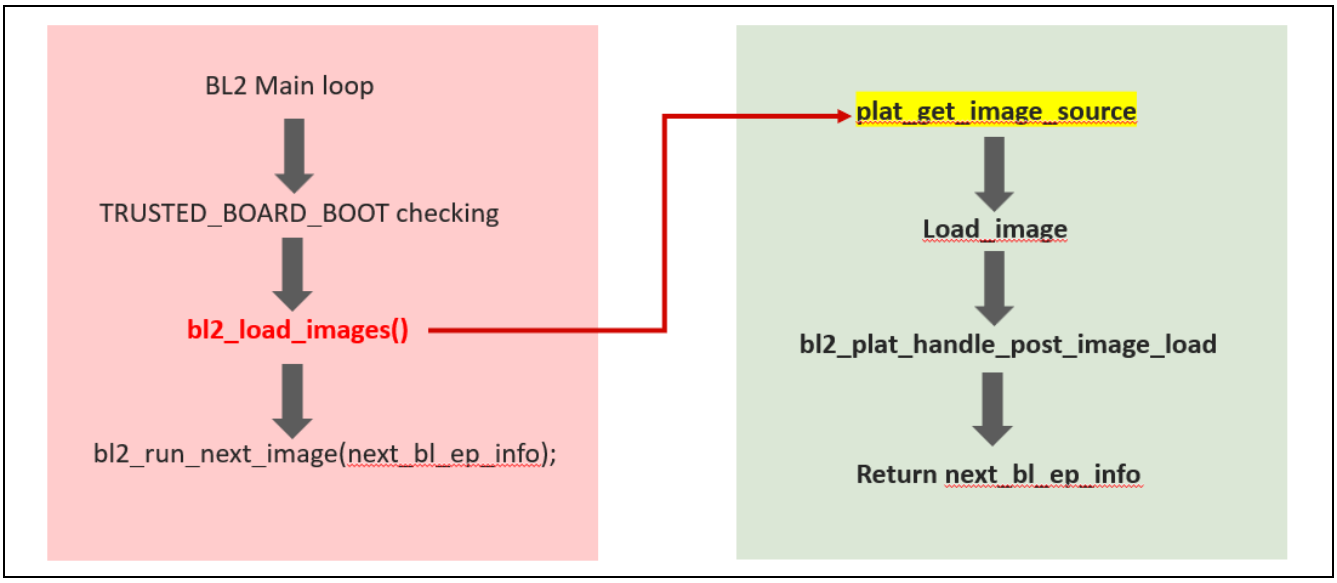


图 2-1 BL2 运作流程-映像载入

第二阶段，取得 dev_handle 与 image_spec 之后，Load_image 会调用：

- io_open(dev_handle, image_spec, &image_handle) : 开启设备
- io_read(image_handle, image_base, image_size, &bytes_read) : 从对应 offset 读取内容进入 DDR

接下来进入 bl2_plat_handle_post_image_load，MA35D1 在该函数内部实现第二阶段的安全启动，通过 ECDSA P-256 来验证 BL31、BL32 以及 BL33 映像的真实性以及完整性。

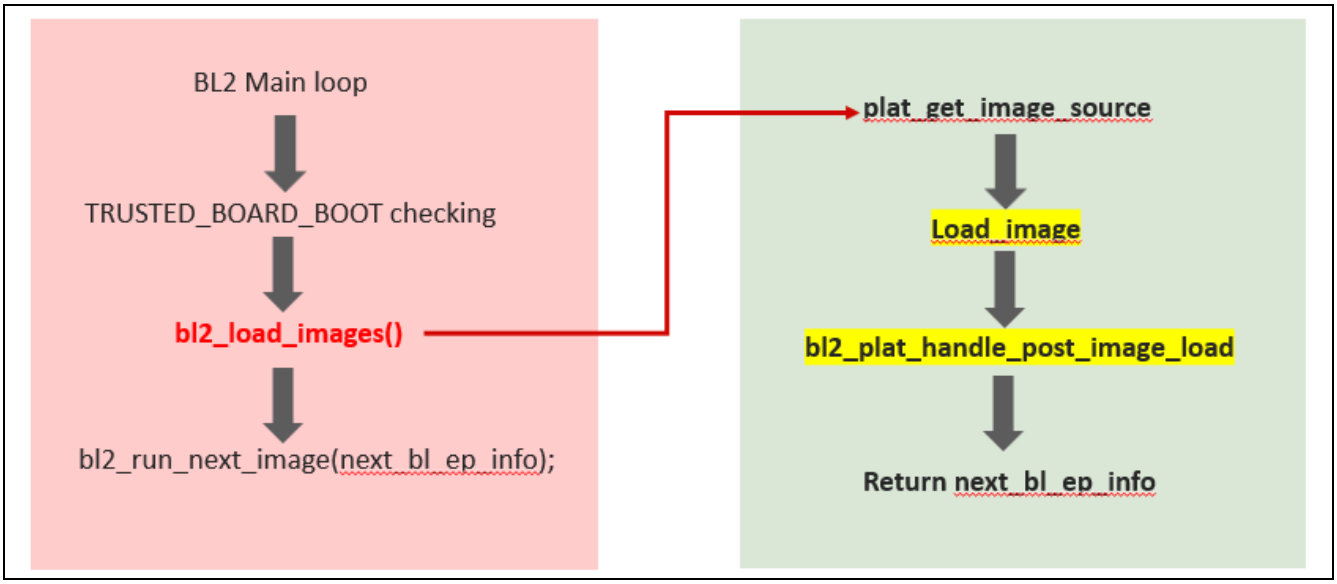


图 2-2 BL2 运作流程-映像检验

BL2 中的主要架构 bl2_load_images，当映像加载(load_auth_image) 或者映像安全检验(bl2_plat_handle_post_image_load) 任一项失败，FIP 映像加载地址被重新指向到备份区域，重新进行操作。当备份映像也启动失败时，打印出相关信息。

```
/*
 * Copyright (c) 2016-2020, ARM Limited and Contributors. All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

#include <assert.h>
#include <stdint.h>

#include <platform_def.h>

#include <arch.h>
#include <arch_helpers.h>
#include <common/bl_common.h>
#include <common/debug.h>
#include <common/desc_image_load.h>
#include <drivers/auth/auth_mod.h>
#include <plat/common/platform.h>

#include "bl2_private.h"

/*******************************************************************************
 * This function loads SCP_BL2/BL3x images and returns the ep_info for
 * the next executable image.
```

```
 ******************************************************************************/

 extern uint32_t FIP_PRIMARY_OFFSET;
 extern uint32_t FIP_BACKUP_OFFSET;
 extern uint32_t FIP_PRIMARY_OFFSET_SPI;
 extern uint32_t FIP_BACKUP_OFFSET_SPI;

 uint32_t current_fip_offset = 0x00;


struct entry_point_info *bl2_load_images(void)
{
    bl_params_t *bl2_to_next_bl_params;
    bl_load_info_t *bl2_load_info;
    const bl_load_info_node_t *bl2_node_info;
    int plat_setup_done = 0;
    int err;
    bool retried = false;

    INFO(" FIP_PRIMARY_OFFSET  %x\n", FIP_PRIMARY_OFFSET);
    INFO(" FIP FIP_BACKUP_OFFSET %x\n", FIP_BACKUP_OFFSET);
    /*
     * Current FIP offset setting.
     */
    current_fip_offset = FIP_PRIMARY_OFFSET;
    /*
     * Get information about the images to load.
     */
    bl2_load_info = plat_get_bl_image_load_info();
    assert(bl2_load_info != NULL);
    assert(bl2_load_info->head != NULL);
    assert(bl2_load_info->h.type == PARAM_BL_LOAD_INFO);
    assert(bl2_load_info->h.version >= VERSION_2);

    retry_fip:
    bl2_node_info = bl2_load_info->head;

    while (bl2_node_info != NULL) {
        /*
         * Perform platform setup before loading the image,
         * if indicated in the image attributes AND if NOT
         * already done before.
         */
        if ((bl2_node_info->image_info->h.attr &
             IMAGE_ATTRIB_PLAT_SETUP) != 0U) {
            if (plat_setup_done != 0) {
                WARN("BL2: Platform setup already done!!\n");
            } else {
                INFO("BL2: Doing platform setup\n");
                bl2_platform_setup();
                plat_setup_done = 1;
            }
        }

        err = bl2_plat_handle_pre_image_load(bl2_node_info->image_id);
        if (err != 0) {
```

```
            ERROR("BL2: Failure in pre image load handling (%i)\n", err);
            plat_error_handler(err);
        }

        if ((bl2_node_info->image_info->h.attr &
            IMAGE_ATTRIB_SKIP_LOADING) == 0U) {
            INFO("BL2: Loading image id %d\n", bl2_node_info->image_id);
            err = load_auth_image(bl2_node_info->image_id,
                bl2_node_info->image_info);
            if (err != 0) {
                ERROR("BL2: Failed to load image id %d (%i)\n",
                      bl2_node_info->image_id, err);
            goto try_backup;
            }
        } else {
            INFO("BL2: Skip loading image id %d\n", bl2_node_info->image_id);
        }

        /* Allow platform to handle image information.
         * Perform ECDSA verification and AES Image Decryption
         */
        err = bl2_plat_handle_post_image_load(bl2_node_info->image_id);
        if (err != 0) {
            ERROR("BL2: Failure in Primary post image load handling (%i)\n", err);
            goto try_backup;
        }

        /* Go to next image */
        bl2_node_info = bl2_node_info->next_load_info;
    }

    goto success;

    /*
     * Backup Image offset configuration.
     */
    try_backup:
    if (!retried){
        WARN ("BL2: Loading Primary failed, retry with backup FIP.\n");
        current_fip_offset = FIP_BACKUP_OFFSET ;
        retried = true;
        goto retry_fip;
    }else{
        ERROR("BL2: Primary and backup FIP both failed\n");
        plat_error_handler(err);
    }



    success:
    /*
     * Get information to pass to the next image.
     */
    bl2_to_next_bl_params = plat_get_next_bl_params();
    assert(bl2_to_next_bl_params != NULL);
    assert(bl2_to_next_bl_params->head != NULL);
```

```
    assert(bl2_to_next_bl_params->h.type == PARAM_BL_PARAMS);
    assert(bl2_to_next_bl_params->h.version >= VERSION_2);
    assert(bl2_to_next_bl_params->head->ep_info != NULL);

    /* Populate arg0 for the next BL image if not already provided */
    if (bl2_to_next_bl_params->head->ep_info->args.arg0 == (u_register_t)0)
        bl2_to_next_bl_params->head->ep_info->args.arg0 =
                    (u_register_t)bl2_to_next_bl_params;

    /* Flush the parameters to be passed to next image */
    plat_flush_next_bl_params();

    return bl2_to_next_bl_params->head->ep_info;
}
```

第一阶段，在 plat_get_image_source 内部会判断该映像是否为 FIP 映像，若为 FIP 映像则会根据目前 current_fip_offset 来决定要读取的位置。默认为主要 FIP 的储存位置，若加载或者验证失败，会重新将该位置指到备份 FIP 地址。

```
/*
 * Return an IO device handle and specification which can be used to access
 * an image. Use this to enforce platform load policy.
 */
extern  uint32_t current_fip_offset;
int plat_get_image_source(unsigned int image_id, uintptr_t *dev_handle,
              uintptr_t *image_spec)
{
    uintptr_t init_params;

    assert(image_id < ARRAY_SIZE(policies));

    *dev_handle = *policies[image_id].dev_handle;

    /*
     * If image id is FIP, using “current_fip_offset”.
     * Others, using platform load policy.
     */
    if (image_id == FIP_IMAGE_ID){
        INFO(" FIP current_fip_offset  %x\n", current_fip_offset);
        static io_block_spec_t local_fip_spec;
        local_fip_spec.offset = current_fip_offset;
        local_fip_spec.length = MA35D1_FIP_SIZE;
        *image_spec = (uintptr_t)&local_fip_spec;
        init_params = policies[image_id].init_params;
    }else{
         *image_spec = policies[image_id].image_spec;
         init_params = policies[image_id].init_params;
    }
    return io_dev_init(*dev_handle, init_params);
}
```

根据设定的外部存储接口，在初始化阶段读取设备树文件(ma35d1.dtsi)所填入的存放位置 。

```
static void ma35d1_sdhc_setup(struct mmc *mmc)
{
    int node;

    /* get device tree information */
    if (fdt_check_header(fdt) < 0) {
        WARN("device tree header check error.\n");
    }

    if (mmc->base == SDH0_BASE) {
        node = fdt_node_offset_by_compatible(fdt, -1,
                "snps,dwcmshc-sdhci0");
        if (node < 0) {
            WARN("The compatible property `snps,dwcmshc-sdhci0` not found\n");
        }
    } else {
        node = fdt_node_offset_by_compatible(fdt, -1,
                "snps,dwcmshc-sdhci1");
        if (node < 0) {
            WARN("The compatible property `snps,dwcmshc-sdhci1` not found\n");
        }
    }

    FIP_PRIMARY_OFFSET= fdt_read_uint32_default(fdt, node, "mmc-image-offset", 0);
    FIP_BACKUP_OFFSET= fdt_read_uint32_default(fdt, node, "mmc-image-backup-offset", 0);

    mmc->bus_width = fdt_read_uint32_default(fdt, node, "bus-width", 1);

    ma35d1_sdhc_hw_init(mmc);
}
```

ma35d1.dtsi:

```
    sdhci0: sdhci@40180000 {
     compatible = "snps,dwcmshc-sdhci0";
     bus-width = <4>;
     /* sdhc information */
     mmc-image-offset = <0xc0000>;
     mmc-image-backup-offset = <0x11c0000>;
    };

    sdhci1: sdhci@40190000 {
     compatible = "snps,dwcmshc-sdhci1";
     bus-width = <4>;
     /* sdhc information */
     mmc-image-offset = <0xc0000>;
     mmc-image-backup-offset = <0x11c0000>;
    };
```

第二部分，又称第二阶段安全启动检验，ma35d1_fip_verify 调用 TSI 进行 ECDSA 验证，通过之后再调用 ma35d1_fip_deaes，将加密的映像做 AES 解密还原。

```
/*******************************************************************************
```

```
 * This function can be used by the platforms to update/use image
 * information for given `image_id`.
 *****************************************************************************/
int bl2_plat_handle_post_image_load(unsigned int image_id)
{
    int err = 0;
    bl_mem_params_node_t *bl_mem_params = get_bl_mem_params_node(image_id);

    bl_mem_params_node_t *pager_mem_params;
    bl_mem_params_node_t *paged_mem_params = NULL;

    assert(bl_mem_params != NULL);
#if FIP_DE_AES
    INFO("secure boot start\n");
    if(ma35d1_fip_verify(bl_mem_params->image_info.image_base,bl_mem_params-
>image_info.image_size)!=0) {
        ERROR("ECC authenticate fail.\n");
        return -1;
        //while(1);
    }
    ma35d1_fip_deaes(bl_mem_params->image_info.image_base,bl_mem_params-
>image_info.image_size);
#endif
    switch (image_id) {
    case BL32_IMAGE_ID:
        pager_mem_params = get_bl_mem_params_node(BL32_EXTRA1_IMAGE_ID);
        assert(pager_mem_params);

        paged_mem_params = get_bl_mem_params_node(BL32_EXTRA2_IMAGE_ID);
        assert(paged_mem_params);

        err = parse_optee_header(&bl_mem_params->ep_info,
                &pager_mem_params->image_info,
                &paged_mem_params->image_info);
        if (err != 0) {
            WARN("OPTEE header parse error.\n");
        }

        /*
         * When ATF loads the DTB the address of the DTB is passed in
         * arg2, if an hw config image is present use the base address
         * as DTB address an pass it as arg2
         */
        bl_mem_params->ep_info.args.arg0 = bl_mem_params->ep_info.args.arg1;
        bl_mem_params->ep_info.args.arg1 = 0;
        bl_mem_params->ep_info.args.arg2 = 0;
        bl_mem_params->ep_info.args.arg3 = 0;

        bl_mem_params->ep_info.spsr = ma35d1_get_spsr_for_bl32_entry();

        break;

    case BL33_IMAGE_ID:
        /* BL33 expects to receive the primary CPU MPID (through r0) */
        bl_mem_params->ep_info.args.arg0 = 0xffff & read_mpidr();
        bl_mem_params->ep_info.spsr = ma35d1_get_spsr_for_bl33_entry();
```

```
        break;

    case SCP_BL2_IMAGE_ID:
        /* The subsequent handling of SCP_BL2 is platform specific */
        err = plat_ma35d1_bl2_handle_scp_bl2(&bl_mem_params->image_info);
        if (err) {
            WARN("Failure in platform-specific handling of SCP_BL2 image.\n");
        }
        break;

    default:
        /* Do nothing in default case */
        break;
    }

    return err;
}
```

3. 软件与硬件需求

3.1 软件需求

- BSP 版本
  - Linux-5.10.x

3.2 硬件需求

- 电路组件
  - NuMaker-HMI-MA35D1 V1.0

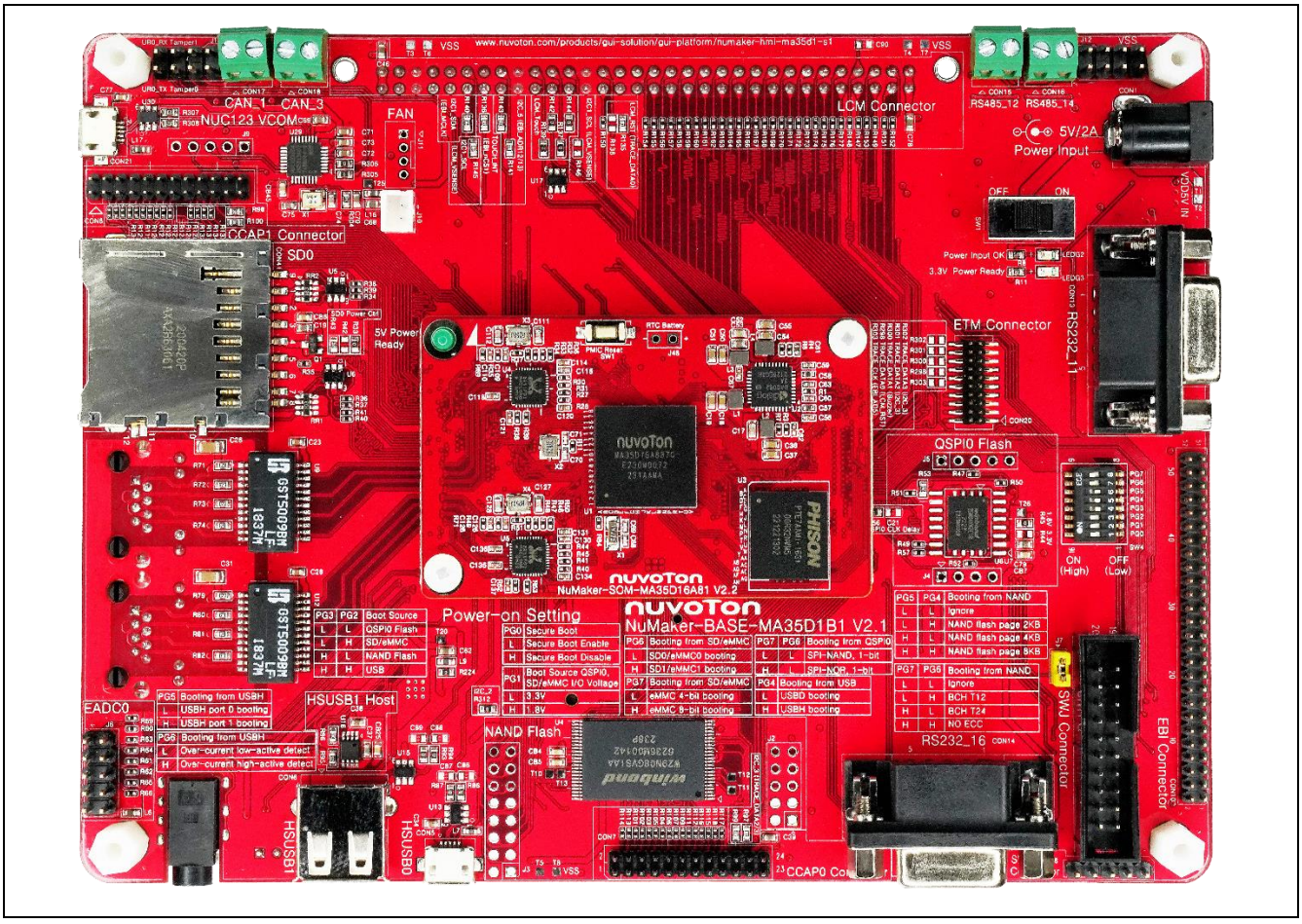


图 3-1 NuMaker-HMI-MA35D1 V1.0

4. 目录信息

目录结构如下图所示。

EC_MA35D1_TF-A_FIP_Backup_V1.00	
arm_trusted_firmware	Reference implementation of secure world
bl2	Trusted Bootloader
common	
driver/nuvoton	
fdts	
lib/optee	
plat/nuvoton	

表 4-1 目录信息

5. 范例程序执行

MA35D1 预设会编译 TF-A 映像，范例要能成功执行必须修改 arm-trusted-firmware 文件夹底下的相关 source code，以下提供 buildroot 相关步骤，外部存储设备为 SD card:

1. 进入 MA35D1 buildroot 相关路径(根据个人设定会有所不同):

```
user@e9e6e7eb0c21:~/shared/buildroot_2024/output/build/arm-trusted-firmware-custom$。
```

2. 将范例程序中各别文件夹的.c /.h /.dtsi 档复制，并取代原 arm-trusted-firmware-custom 底下的对应档案。注意，该动作是取代，并不是直接将范例程序文件夹直接整个复制到 arm-trusted-firmware-custom 文件夹。

3. 在 menuconfig 中开启 Secure Boot 以及设定相关密钥:

```
Bootloaders --- >
  [*] MA35D1 Secure Boot
  (0A0BC81E5AFBF836C5E0CFBEA12C1E269A2EBC3B0B6EC39EEE1C7142F760EBC4) AES Key
  (8F1571E9975006A545DF854264F7B18934B5CB2EE619E66DBC0FBEA52C71C919) ECDSA Key
```

4. 移除 arm-trusted-firmware-custom 底下的 build 文件夹。

5. 重新編譯 arm-trusted-firmware-custom。

```
make arm-trusted-firmware-rebuild
```

6. 进入以下路径:

```
user@e9e6e7eb0c21:~/shared/buildroot_2024/board/nuvoton/ma35d1/nuwriter $
```

7. 修改"uboot-env.bin-sdcard" offset 地址，以及增加备份 FIP 映像相关设定于 pack-sdcard.json。

```
{
  "image": [
    {
      "offset": "0x0",
      "file": "MBR.sdcard.bin",
      "type": 0
    },
    {
      "offset": "0x400",
      "file": "header.bin",
      "type": 0
    },
    {
      "offset": "0x20000",
```

```
      "file": "nuwriter/bl2.dtb",
      "type": 0
    },
    {
      "offset": "0x30000",
      "file": "nuwriter/bl2.bin",
      "type": 0
    },
    {
      "offset": "0x50000",
      "file": "uboot-env.bin-sdcard",
      "type": 0
    },
    {
      "offset": "0xC0000",
      "file": "fip.bin-sdcard",
      "type": 0
    },
    {
      "offset": "0x2c0000",
      "file": "Image.dtb",
      "type": 0
    },
    {
      "offset": "0x300000",
      "file": "Image",
      "type": 0
    },
    {
      "offset": "37748736",
      "file": "rootfs.ext4",
      "type": 0
    },
    {
      "offset": "0x11C0000",
      "file": "fip.bin-sdcard",
      "type": 0
    }
  ]
}
```

8. 重新编译映像档。

```
make
```

9. 通过 NuWriter 烧录 pack 映像。

10. 通过 NuWriter 清除主要 fip.bin-sdcard 所写入的位置(0xc0000-0x1c0000, 参考 pack_sdcard.json)。

11. 编译成功后，NuWriter 会生成一”otp_key.json”档案于 output/images/nuwriters 底下，包含 ECDSA 公钥以及 AES 密钥，通过 NuWriter 烧录于 MA35D1 IBR Key 区域。

12. 将开发板上的 Power-on setting 指拨开关设定为 boot from SD0，启动 MA35D1 开发板。

13. 进入 BL2 ，开始进行各 BL3x 映像的加载过程，由于主要 FIP 映像在步骤 10 时已被清除，BL2 无法正确获取相关信息显示启动失败，改由备份地址获取 FIP 映像，并完成启动。

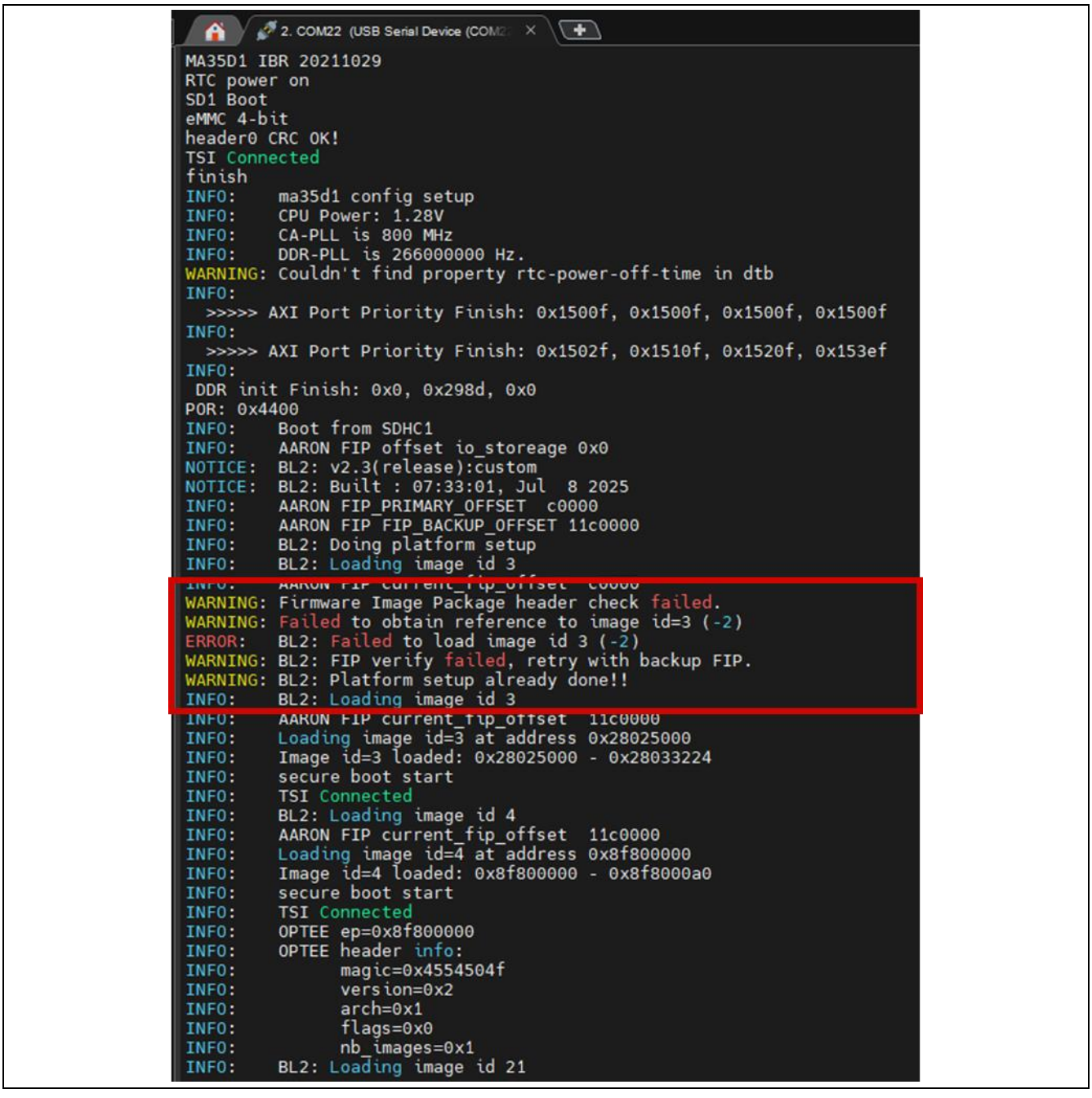


图 5-1 范例执行结果-主映像加载失败

14. 第二项测试，验证错误私钥加签的 FIP 映像，首先清除 arm-trusted-firmware。

```
make arm-trusted-firmwared-dirclean
```

15. 重新设定 ECDSA 私钥。

```
Bootloaders --- >
  [*] MA35D1 Secure Boot
  (0A0BC81E5AFBF836C5E0CFBEA12C1E269A2EBC3B0B6EC39EEE1C7142F760EBC4) AES Key
  (8F1571E9975006A545DF854264F7B18934B5CB2EE619E66DBC0FBEA52C71C918) ECDSA Key
```

16. 重新透過 make 編譯新的映像。

17. 再次通过 NuWriter 将错误私钥签署过的 fip.bin-sdcard ，烧写到主要 FIP 映像存放地址 (0xc0000, 参考 pack_sdcard.json)。

18. 再次启动 MA35D1 开发板，当 BL2 成功加载主要区域的 FIP 映像后，开始进行第二阶段的签章验证，由于主要 FIP 的签章是错误的，在验证失败后会尝试从备份 FIP 映像位置读取，并再次进行验证，验证成功后继续往下执行完成系统启动。

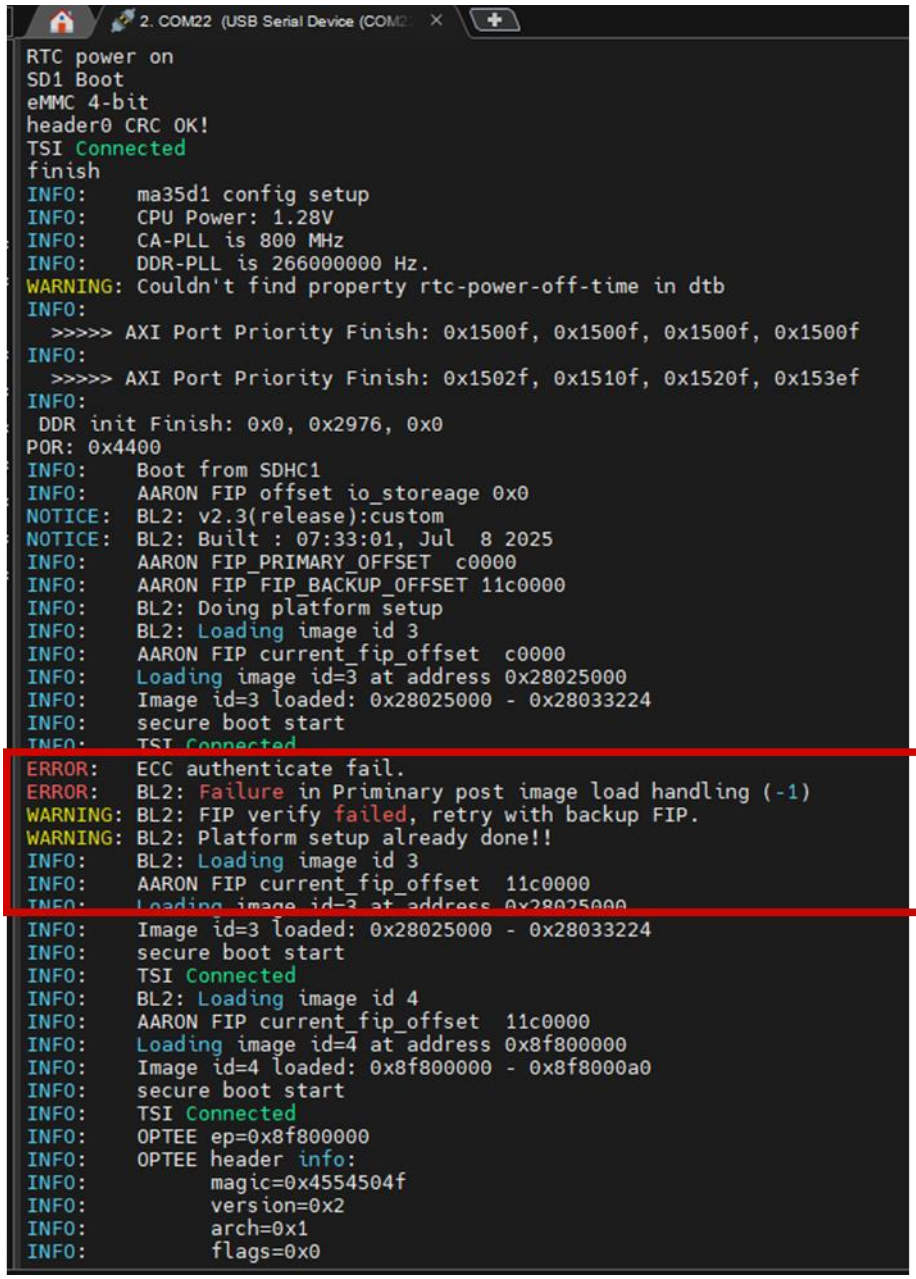


图 5-2 范例执行结果- 主映像签章验证失败

6. 修订纪录

Date	Revision	Description
2025.10.07	1.00	初始发布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.