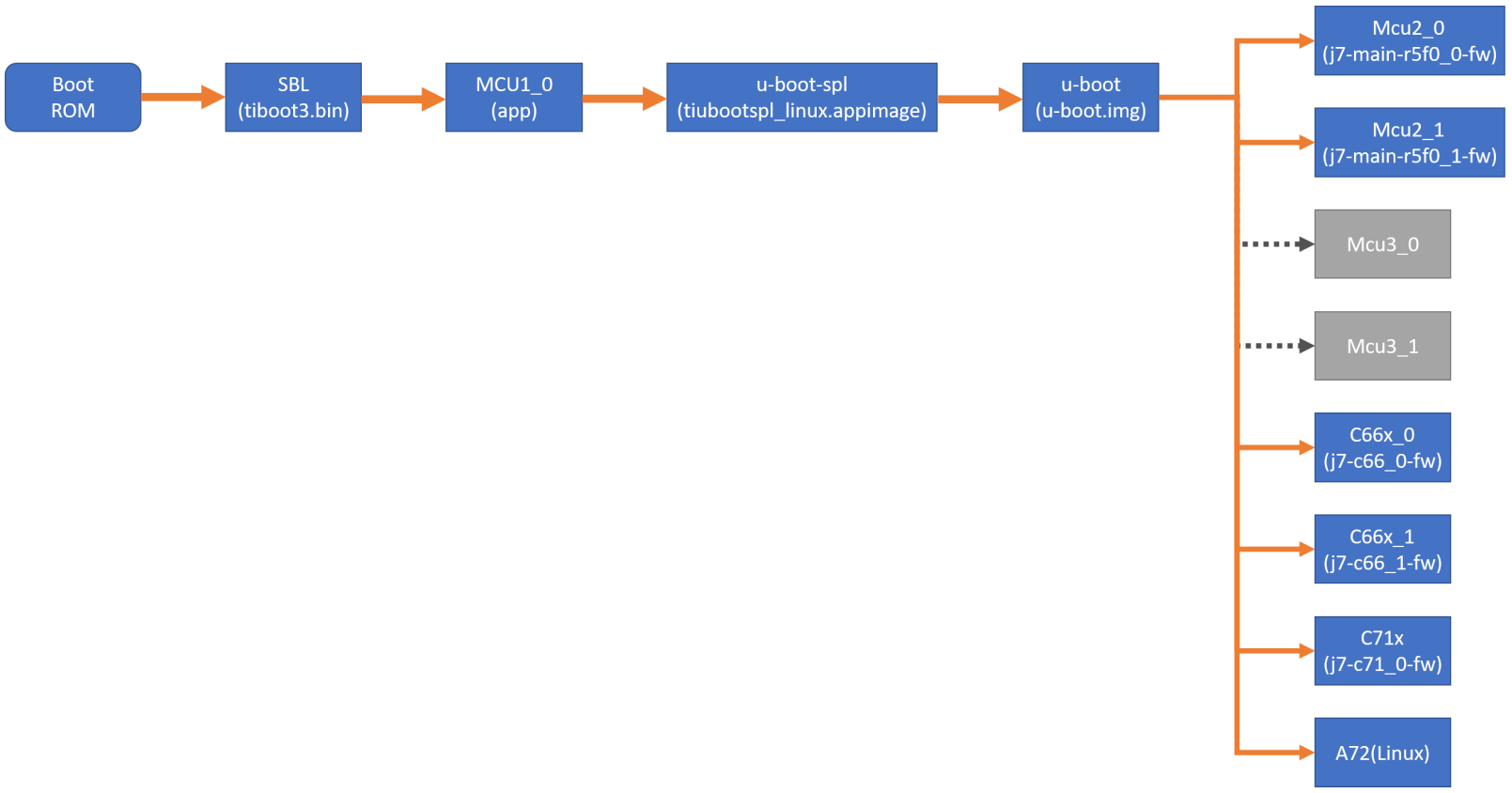
前言



tda4启动流程

- ROM Code
- SBL
  - tiboot3.bin
  - tifs.bin
- APP (MCU1_0)
  - app
  - atf_optee.appimage
  - tiubootspl_linux.appimage
- u-boot-spl
  - u-boot.img
- u-boot
  - j7-<core name>-fw
  - k3-j721e-common-proc-board.dtb
  - k3-j721e-vision-apps.dtbo
  - Image
- linux
  - Fs

TDA4-BOOT

tda4上面一共有三个功能域，每个域都包含特定的Arm核和外设：

1. WKUP（Wake-up，唤醒）域
    - 单核 ARM Cortex-M3 DMSC（Device Management and Security Controller，设备管理和安全控制器）
2. MCU（Microcontroller，微控制器）域
    - 双核 ARM Cortex-R5 处理器带有浮点运算单元
3. MIAN 域
    - 双核 64 位 ARM Cortex-A72
    - 2 * 双核 ARM Cortex-R5 子系统
    - 2 * C66x DSP（Digital signal processor，数字信号处理器）子系统
    - 带有 MMA（Matrix Multiplication Accelerator，矩阵乘法加速器） 的 C71x DSP 子系统

Figure 4-1. Initialization Process

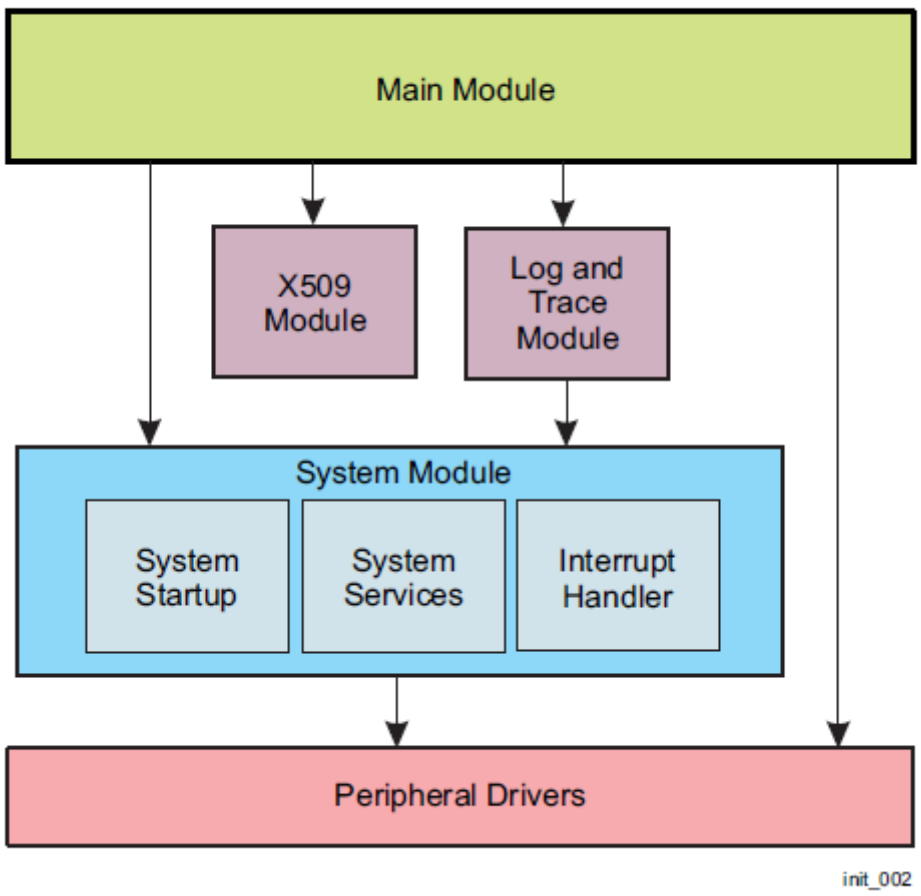
零、ROM Code

1. 介绍

ROM Code（又称 ROM Bootloader ）是芯片出厂时烧录到片上ROM的软件，是芯片上电自动运行的第一个代码块。

ROM Code 分为 MCU ROM Code 和 DMSC ROM Code，DMSC 是 MCU 内核的启动主控。其中 MCU ROM Code 软件架构如下图：

Figure 4-2. MCU ROM Code Architecture

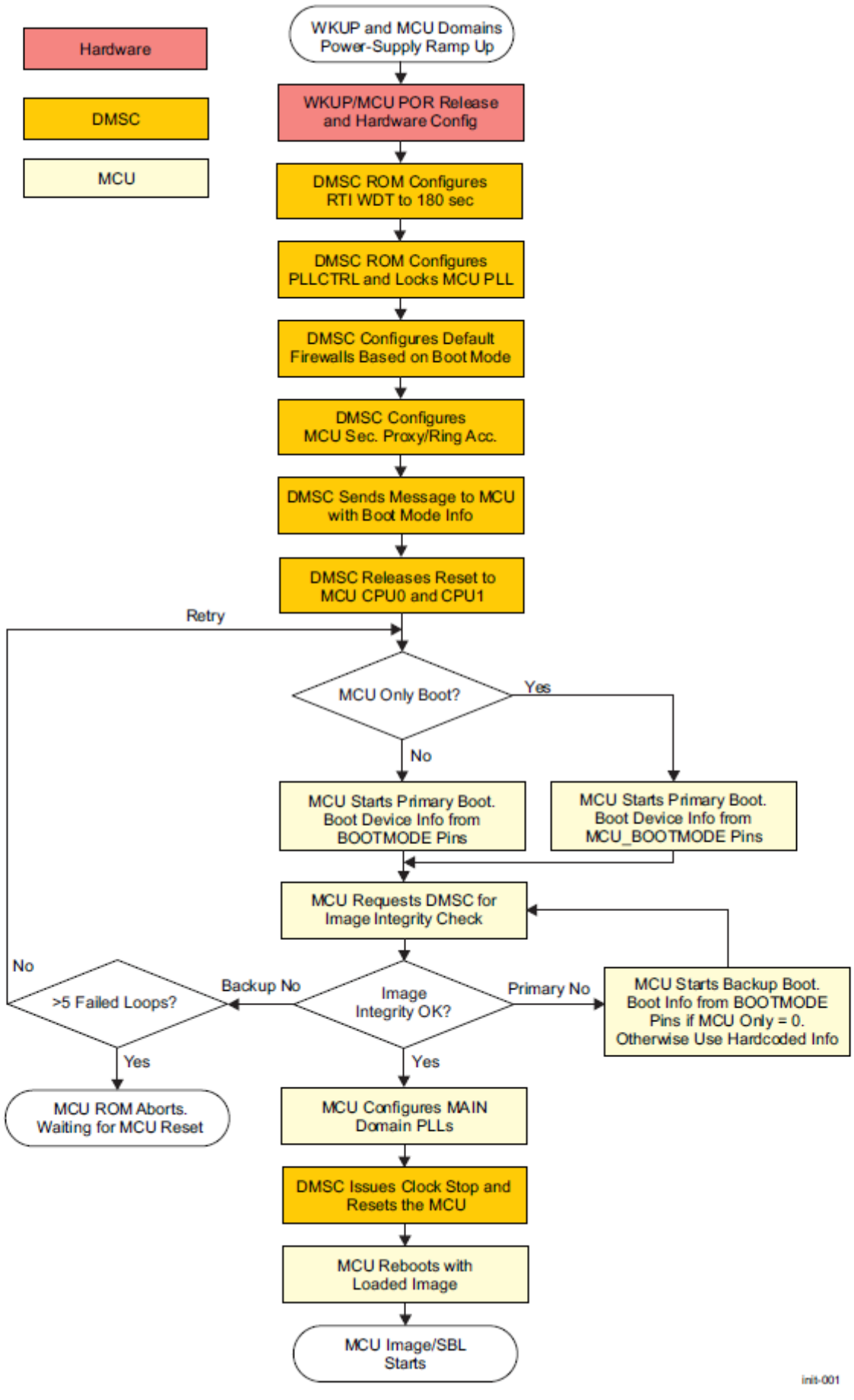


ROM Code的主要任务：

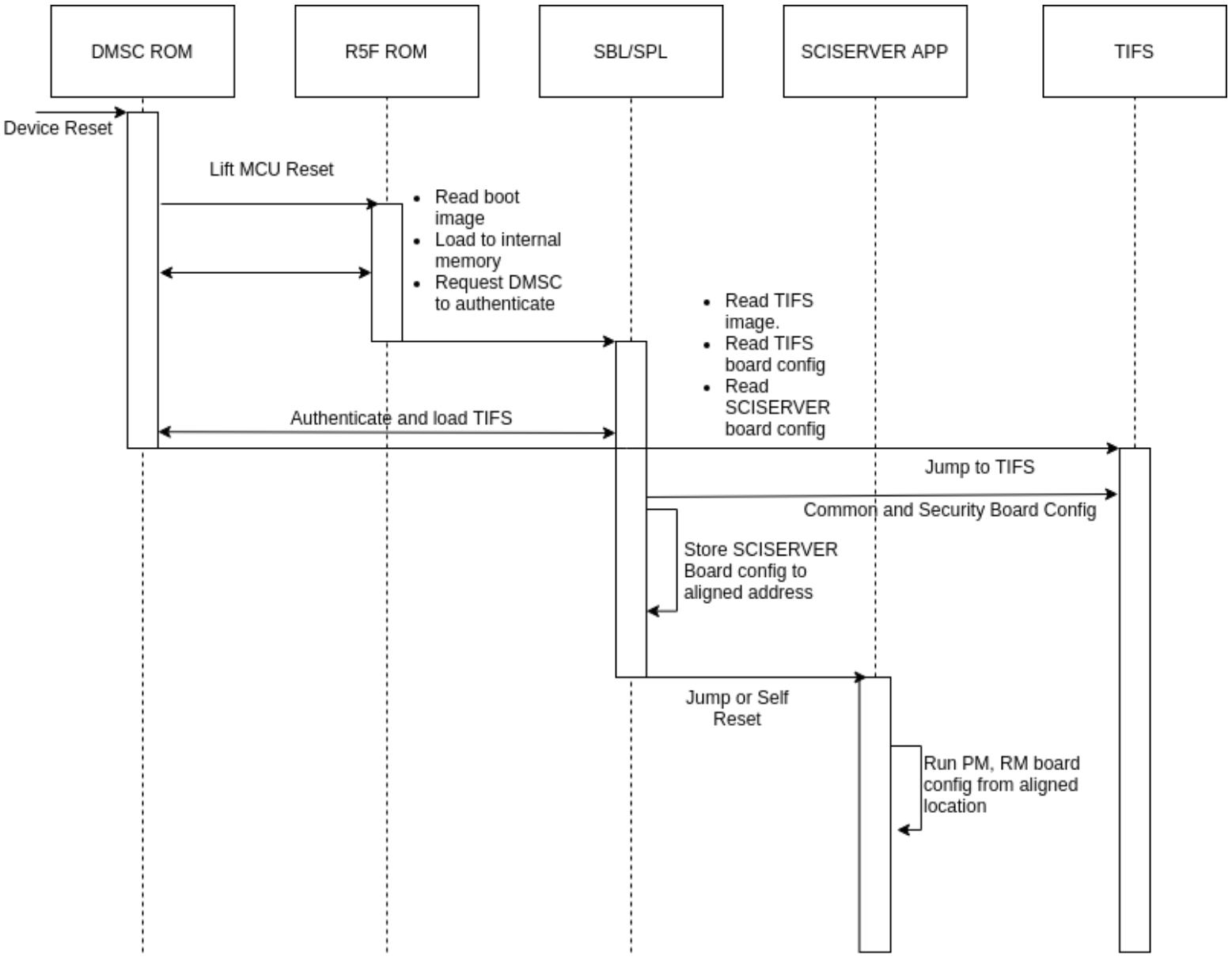
- 器件配置和主要外设初始化，初始化设备资源（PLL时钟，外设和引脚）
- 初始化启动设备并将程序镜像加载到片上SRAM并引导运行

2. 运行流程

Figure 4-3. Boot Process



详细运行流程



一、SBL

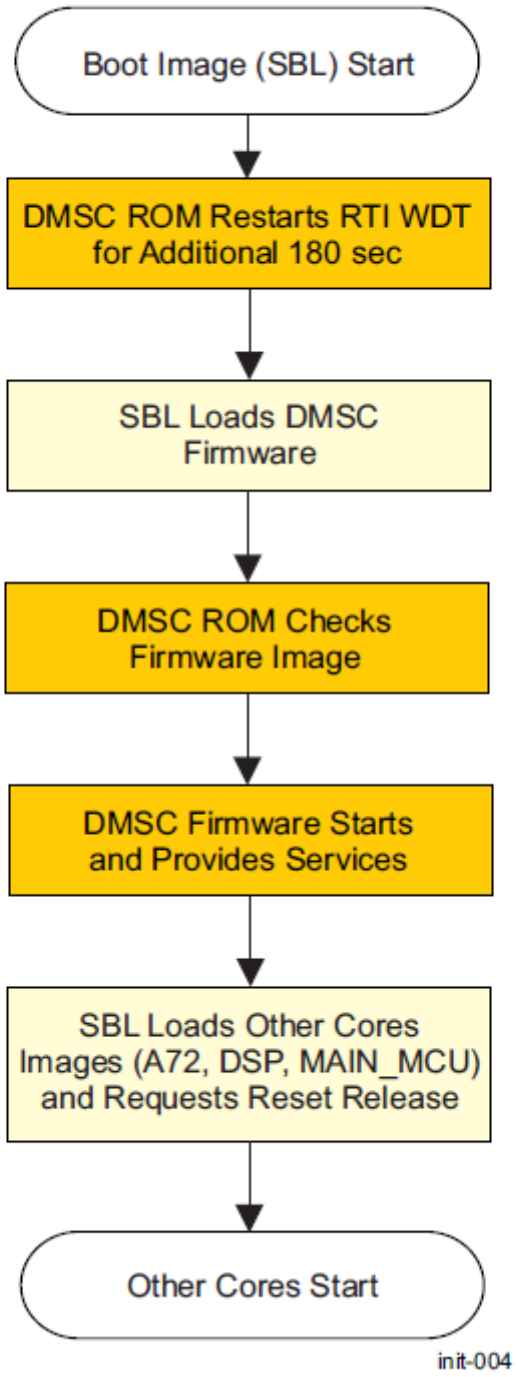
1. 介绍

SBL（Secondary Boot Loader，次级启动加载程序），用于加载DMSC固件和其他核的镜像文件。

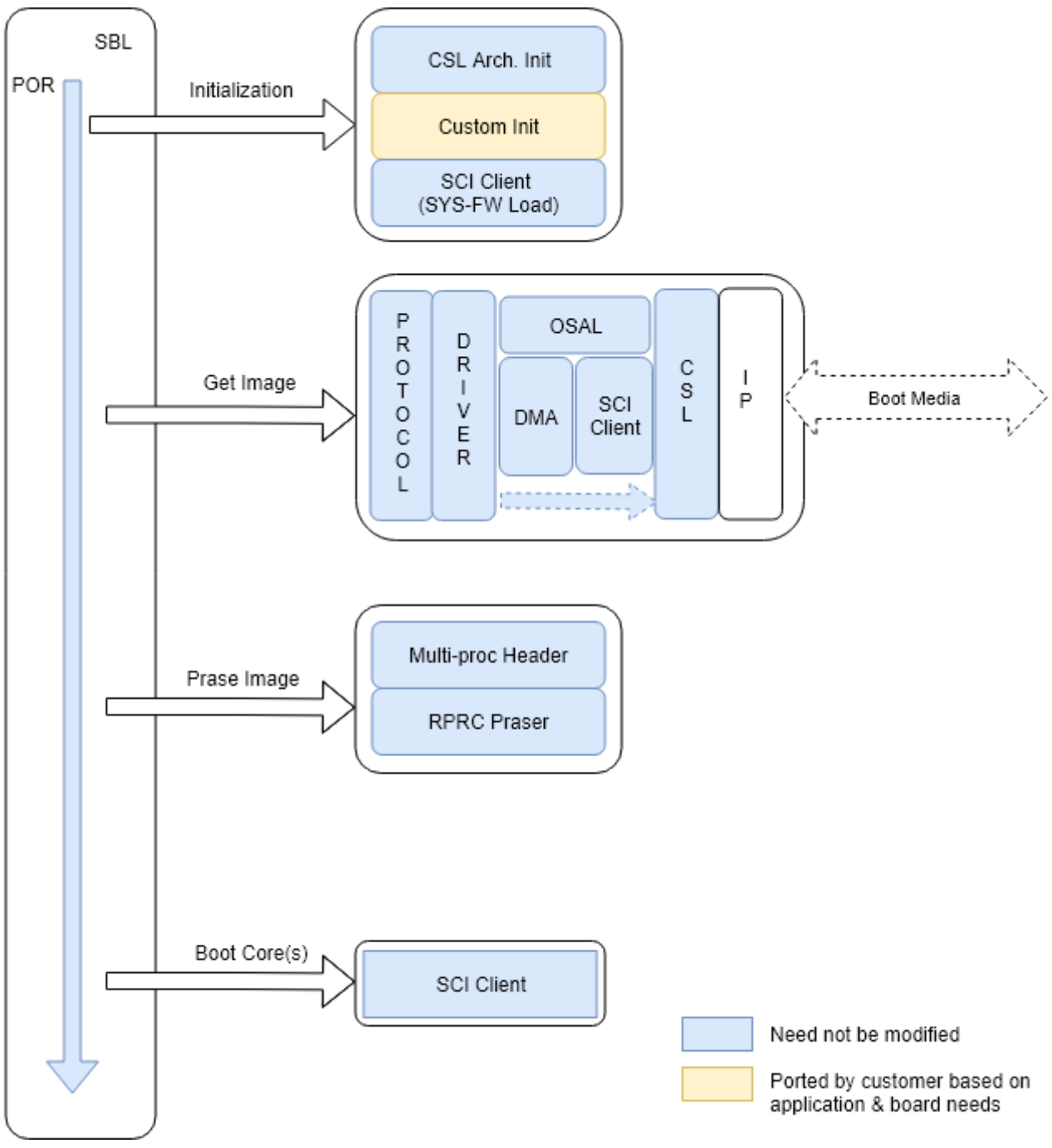
在当前启动流程中，SBL 只启动了 mcu1_0，即MCU域的0核。

2. 运行流程

Figure 4-4. External Bootloader Tasks



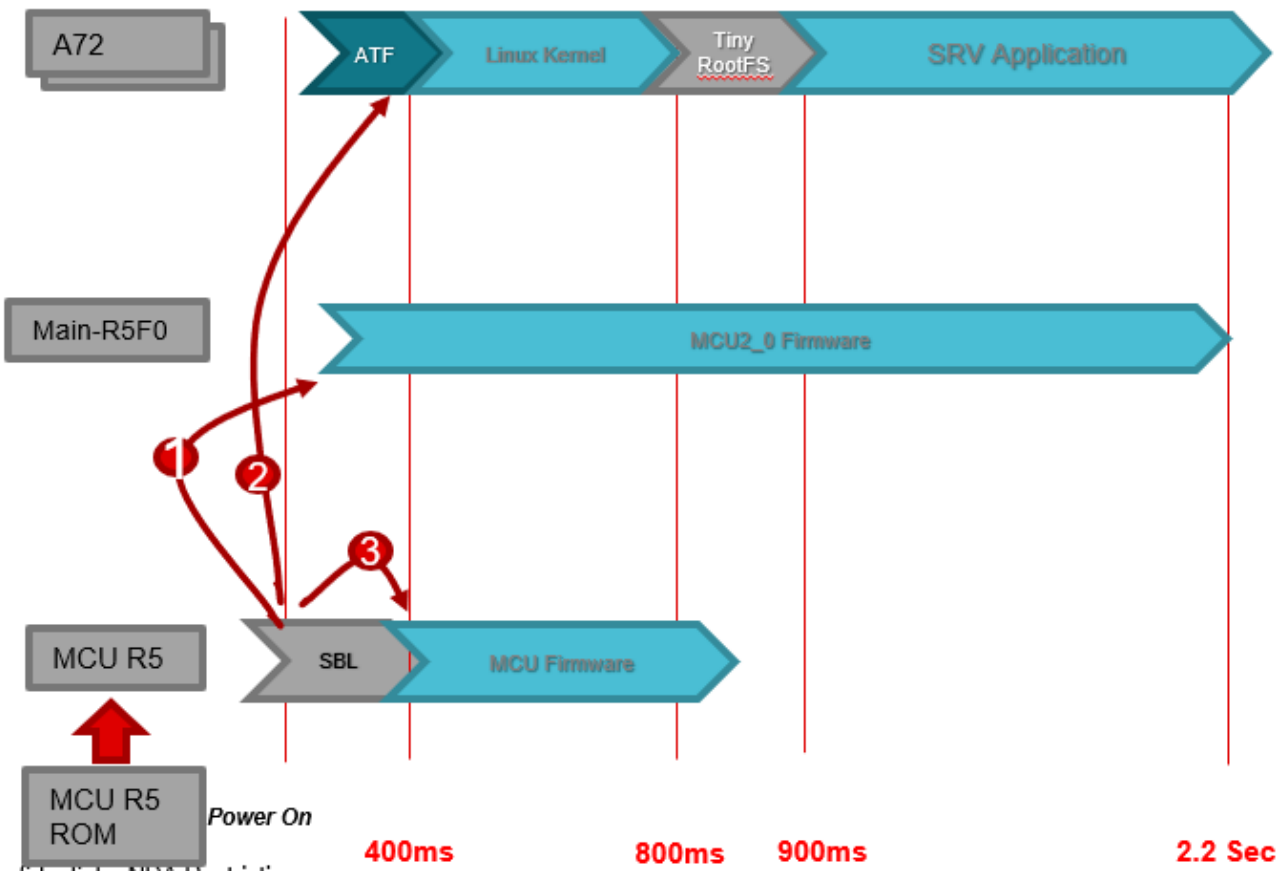
详细运行流程



3. SBL和SPL的区别

SBL	SPL
基于 RTOS 的自定义 Bootloader	基于标准 U-Boot 的 Bootloader
不同启动设备使用不同的镜像文件	不同启动设备可使用同一镜像文件
文件较小	文件较大
启动时间快	启动时间慢
适用于启动时间敏感型应用	功能更强大，使用更灵活

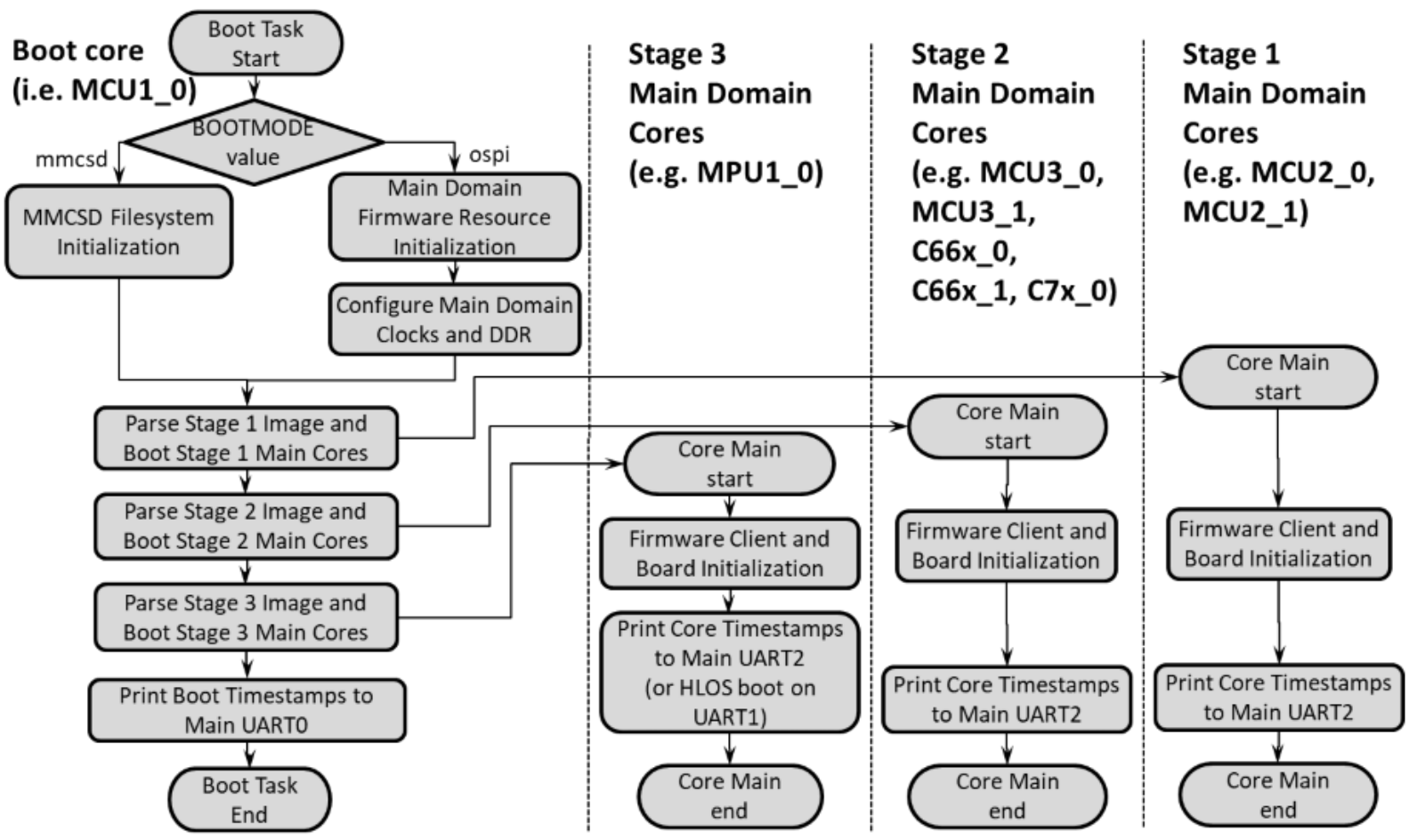
> 使用 OSPI 的 XIP（Execute In Place，就地执行）功能最快可以 2.2s 左右出图（待验证）。
>
> XIP 启动流程：RBL(non-XIP) -> SBL(non-XIP) -> Trampoline Function(non-XIP) -> Application(XIP)



二、APP

运行流程

Stage 1 和 Stage 2 直接跳过。使用 uboot-spl 作为 Mpu1_0 的 运行程序。



三、U-Boot

1. 介绍

U-Boot（Universal Boot Loader，通用引导加载程序）是一种开源的 Bootloader，主要用于嵌入式设备启动操作系统。

U-Boot 有两个阶段：

- 第一阶段为 SPL，SPL 是 Secondary Program Loader 的简称，之所以称作 secondary，是相对于 ROM code 来说的。SPL 是为了在正常的 u-boot.img 之外，提供一个独立的、小 size 的 SPL image，通常用于那些 SRAM 比较小（或者其它限制）、无法直接装载并运行整个 u-boot 的平台。
- 第二阶段为功能齐全的 U-Boot 版本，提供一个菜单供用户交互和控制启动过程，通过菜单可以配置启动设备并从中加载操作系统和相关镜像文件，最后启动操作系统。

2. 配置

U-Boot 使用的配置文件为 j721e_evm_r5_defconfig（非 SBL 使用） 和 j721e_evm_a72_defconfig，所有可用的配置选项在 scripts/config_whitelist.txt 中。

Kconfig 会解析 defconfig 文件并判断依赖选项**自动添加配置**， 最后生成的配置文件为 u-boot_build/a72/include/generated/autoconf.h。

```
# 取消配置
# CONFIG_FOO is not set

# 设置配置
CONFIG_FOO=y
```

SPL 是 u-boot 中独立的一个代码分支，由 `CONFIG_SPL_BUILD` 配置项控制。

代码中：

```
#if CONFIG_IS_ENABLED(FOO)
/*
 *  1 if CONFIG_SPL_BUILD is undefined and CONFIG_FOO is set to 'y',
 *  1 if CONFIG_SPL_BUILD is defined and CONFIG_SPL_FOO is set to 'y',
 *  1 if CONFIG_TPL_BUILD is defined and CONFIG_TPL_FOO is set to 'y',
 *  0 otherwise.
 */
#endif

#if defined(CONFIG_FOO)
#endif
```

Makefile 文件中：

```
ifeq ($(CONFIG_FOO),y)
endif

obj-$(CONFIG_FOO) += foo.o
# uboot 会编译所有的 obj-y 的代码
```

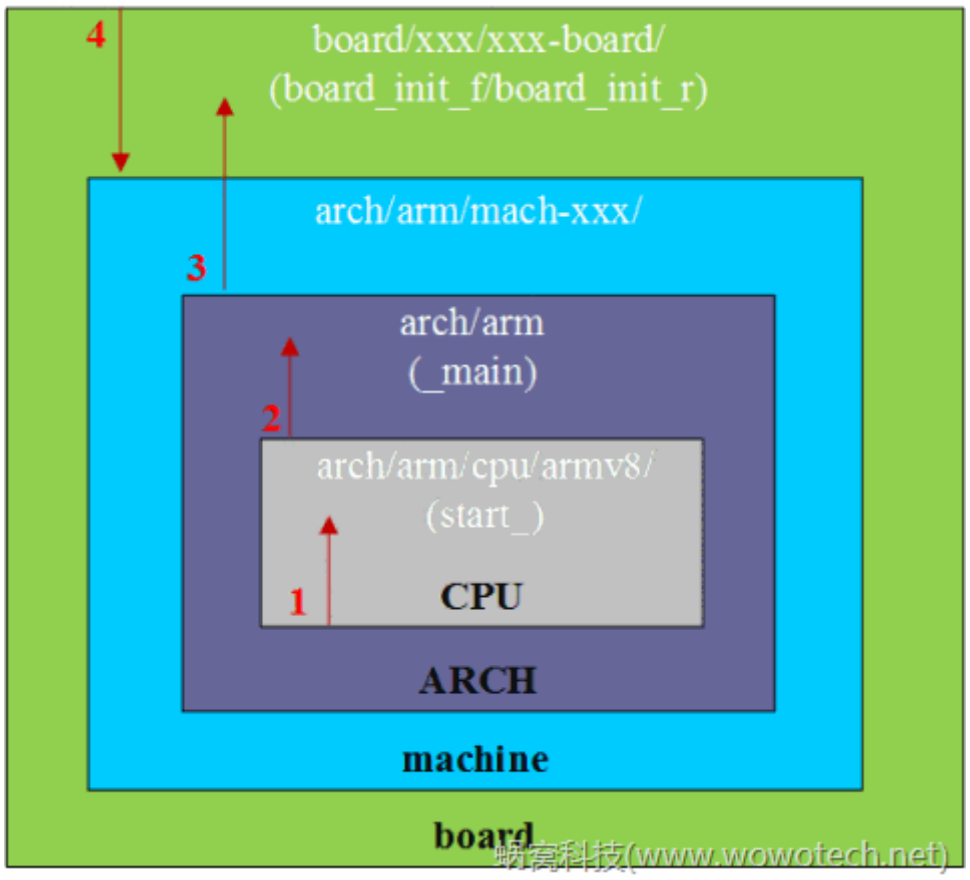
3. 编译

编译命令如下：

```
# 配置 U-Boot 代码，生成 Kconfig 文件和 board-support/u-boot_build/a72/include/generated/autoconf.h
make -j 7 -C board-support/u-boot-* \
    CROSS_COMPILE=x86_64-arago-linux/usr/bin/aarch64-none-linux-gnu- \
    j721e_evm_a72_defconfig \
    O=board-support/u-boot_build/a72
# 编译代码，同时生成 tispl.bin 和 u-boot.img
make -j 7 -C board-support/u-boot-* \
    CROSS_COMPILE=x86_64-arago-linux/usr/bin/aarch64-none-linux-gnu- \
    CONFIG_MKIMAGE_DTC_PATH=board-support/u-boot_build/a72/scripts/dtc/dtc \
    ATF=board-support/prebuilt-images/bl31.bin \
    TEE=board-support/prebuilt-images/bl32.bin \
    DM=board-support/prebuilt-images/ipc_echo_testb_mcu1_0_release_strip.xer5f \
    O=board-support/u-boot_build/a72
```

4. 软件抽象

u-boot 是一个跨平台、跨设备的 bootloader。u-boot 通过 “board -> machine -> arch -> cpu” 框架对软件进行抽象和封装，如下图所示：



该结构基本上和硬件的拓扑结构保持一致：

- board 代表一个可满足产品功能的“硬件实体”，该“实体”包括一些必要的外部设备，如显示屏、按键、麦克风、扬声器等等。

  > board-support/u-boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8/board/ti/j721e

- machine 代表 SOC（System on Chip，片上系统或系统级芯片），SOC 是“实体”的运算和控制中心，里面集成了很多和外部设备有关的功能，例如各种各样的设备控制器。

  > board-support/u-boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8/arch/arm/mach-k3

- arch 代表 SOC 的 IP 核，例如arm（包括arm32和arm64）、mips 和 x86。

  > board-support/u-boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8/arch/arm

- cpu 代表 IP 核的架构，例如 arm 的架构 armv7 和 armv8。

  > board-support/u-boot-2021.01+gitAUTOINC+44a87e3ab8-g44a87e3ab8/arch/arm/cpu/armv8

基于上图所示的架构，u-boot和平台有关的初始化流程如下：

1. u-boot启动后，会先执行CPU（arch/arm/cpu/armv8）的初始化代码。
2. CPU相关的代码，会调用ARCH的公共代码（arch/arm）。
3. ARCH的公共代码，在适当的时候，调用board（board/ti/j721e）有关的接口。u-boot的功能逻辑，大多是由common代码实现，部分和平台有关的部分，则由公共代码声明，由board代码实现。
4. board代码在需要的时候，会调用machine（arch/arm/mach-k3）提供的接口，实现特定的功能。因此machine的定位是提供一些基础的代码支持，不会直接参与到u-boot的功能逻辑中。

5. 启动流程

arch/arm/cpu/armv8/start.S:

> 从 `_start` 开始运行，配置 SCTRL 寄存器、中断向量、MMU、Endian 和 i/d Cache 等，中途调用 `lowlevel_init`，最后运行 `bl _main` 跳转到 arm 公共的 `_main` 中执行

arch/arm/lib/crt0_64.S:

> 从 `_main` 开始运行，设置C代码的运行环境，中途运行 `bl board_init_f` 调用 `board_init_f` 函数，完成前期初始化。
>
> **如果是 u-boot 会多一个 relocation 操作。**然后清除 BBS 段，最后调用`board_init_r`函数，执行后续的初始化操作。

5.1 SPL

arch/arm/mach-k3/j721e_init.c

```
1  void board_init_f(ulong dummy)
2  {
3  #if defined(CONFIG_CPU_V7R) && defined(CONFIG_K3_AVS0)
4      int offset;
5      u32 val, dflt = 0;
6  #endif
7  #if defined(CONFIG_K3_J721E_DDRSS) || defined(CONFIG_K3_LOAD_SYSFW) || \
8      defined(CONFIG_ESM_K3) || defined(CONFIG_ESM_PMIC)
9      struct udevice *dev;
10     int ret;
11 #endif
12     /*
13      * Cannot delay this further as there is a chance that
```

```
     * K3_BOOT_PARAM_TABLE_INDEX can be over written by SPL MALLOC section.
     */
    store_boot_info_from_rom();

    /* Make all control module registers accessible */
    ctrl_mmr_unlock();

#ifdef CONFIG_CPU_V7R
    disable_linefill_optimization();
    setup_k3_mpu_regions();
#endif

    /* Qos 总线消息设置 */
    if (soc_is_j721e()) {
#ifndef CONFIG_TI_SECURE_DEVICE
        setup_navss_nb();
        setup_c66_qos();
        setup_main_r5f_qos();
        setup_vpac_qos();
        setup_dmpac_qos();
        setup_dss_qos();
        setup_gpu_qos();
        setup_encoder_qos();
#endif
    }

    /* SPL dts 加载，设置gd->bootstage为BOOTSTAGE_ID_ACCUM_DM_SPL 此时串口未初始化  */
    /* Init DM early */
    spl_early_init();

/* R5-SPL */
#ifdef CONFIG_K3_LOAD_SYSFW
    /*
     * Process pinctrl for the serial0 a.k.a. MCU_UART0 module and continue
     * regardless of the result of pinctrl. Do this without probing the
     * device, but instead by searching the device that would request the
     * given sequence number if probed. The UART will be used by the system
     * firmware (SYSFW) image for various purposes and SYSFW depends on us
     * to initialize its pin settings.
     */
    ret = uclass_find_device_by_seq(UCLASS_SERIAL, 0, true, &dev);
    if (!ret)
        pinctrl_select_state(dev, "default");

    /*
     * Load, start up, and configure system controller firmware. Provide
     * the U-Boot console init function to the SYSFW post-PM configuration
     * callback hook, effectively switching on (or over) the console
     * output.
     */
    k3_sysfw_loader(is_rom_loaded_sysfw(&bootdata),
            k3_mmc_stop_clock, k3_mmc_restart_clock);

    /* R5-SPL */
    #ifdef CONFIG_SPL_OF_LIST
        do_dt_magic();
    #endif

    /* R5-SPL */
    #ifdef CONFIG_SPL_CLK_K3
        /*
         * Force probe of clk_k3 driver here to ensure basic default clock
         * configuration is always done.
         */
        ret = uclass_get_device_by_driver(UCLASS_CLK,
                          DM_GET_DRIVER(ti_clk),
                          &dev);
        if (ret)
            panic("Failed to initialize clk-k3!\n");
    #endif

        /* Prepare console output */
        preloader_console_init();

        /* Disable ROM configured firewalls right after loading sysfw */
    #ifdef CONFIG_TI_SECURE_DEVICE
        remove_fwl_configs(cbass_hc_cfg0_fwls, ARRAY_SIZE(cbass_hc_cfg0_fwls));
        remove_fwl_configs(cbass_hc0_fwls, ARRAY_SIZE(cbass_hc0_fwls));
        remove_fwl_configs(cbass_rc_cfg0_fwls, ARRAY_SIZE(cbass_rc_cfg0_fwls));
        remove_fwl_configs(cbass_rc0_fwls, ARRAY_SIZE(cbass_rc0_fwls));
```

```
        remove_fwl_configs(infra_cbass0_fwls, ARRAY_SIZE(infra_cbass0_fwls));
        remove_fwl_configs(mcu_cbass0_fwls, ARRAY_SIZE(mcu_cbass0_fwls));
        remove_fwl_configs(wkup_cbass0_fwls, ARRAY_SIZE(wkup_cbass0_fwls));
    #endif
#else
    /* A72-SPL */
    /* Prepare console output */
    preloader_console_init();
#endif /* CONFIG_K3_LOAD_SYSFW */

#if defined(CONFIG_DISPLAY_BOARDINFO)
    show_board_info();
#endif

    /* Output System Firmware version info */
    k3_sysfw_print_ver();

    /* Perform EEPROM-based board detection */
    if (IS_ENABLED(CONFIG_TI_I2C_BOARD_DETECT))
        do_board_detect();

/* R5-SPL */
#if defined(CONFIG_CPU_V7R) && defined(CONFIG_K3_AVS0)
    if (board_ti_k3_is("J721EX-PM1-SOM")) {
        offset = fdt_node_offset_by_compatible(gd->fdt_blob, -1,
                                "ti,am654-vtm");
        val = fdt_getprop_u32_default_node(gd->fdt_blob, offset, 0,
                                "som1-supply-2", dflt);
        do_fixup_by_compat_u32((void *)gd->fdt_blob, "ti,am654-vtm",
                       "vdd-supply-2", val, 0);
    }

    ret = uclass_get_device_by_driver(UCLASS_MISC, DM_GET_DRIVER(k3_avs),
                      &dev);
    if (ret)
        printf("AVS init failed: %d\n", ret);
#endif

/* R5-SPL */
#ifdef CONFIG_ESM_K3
    if (board_ti_k3_is("J721EX-PM2-SOM") ||
        board_ti_k3_is("J7200X-PM2-SOM")) {
        ret = uclass_get_device_by_driver(UCLASS_MISC,
                          DM_GET_DRIVER(k3_esm), &dev);
        if (ret)
            printf("MISC init failed: %d\n", ret);
    }
#endif

/* R5-SPL */
#ifdef CONFIG_ESM_PMIC
    if (board_ti_k3_is("J721EX-PM2-SOM") ||
        board_ti_k3_is("J7200X-PM2-SOM")) {
        ret = uclass_get_device_by_driver(UCLASS_MISC,
                          DM_GET_DRIVER(pmic_esm),
                          &dev);
        if (ret)
            printf("ESM PMIC init failed: %d\n", ret);
    }
#endif

/* R5-SPL */
#if defined(CONFIG_K3_J721E_DDRSS)
    ret = uclass_get_device(UCLASS_RAM, 0, &dev);
    if (ret)
        panic("DRAM init failed: %d\n", ret);
#endif
    spl_enable_dcache();
}
```

common/spl/spl.c

```
void board_init_r(gd_t *dummy1, ulong dummy2)
{
    u32 spl_boot_list[] = {
        BOOT_DEVICE_NONE,
        BOOT_DEVICE_NONE,
        BOOT_DEVICE_NONE,
        BOOT_DEVICE_NONE,
        BOOT_DEVICE_NONE,
```

```
    };
    struct spl_image_info spl_image;
    int ret;

    debug(">>" SPL_TPL_PROMPT "board_init_r()\n");

    /* 设置gd->bd (board information) */
    spl_set_bd();

/* Not Used */
#if defined(CONFIG_SYS_SPL_MALLOC_START)
    mem_malloc_init(CONFIG_SYS_SPL_MALLOC_START,
            CONFIG_SYS_SPL_MALLOC_SIZE);
    gd->flags |= GD_FLG_FULL_MALLOC_INIT;
#endif
    if (!(gd->flags & GD_FLG_SPL_INIT)) {
        /* 在spl_early_init中已初始化dm, 跳过 */
        if (spl_init())
            hang();
    }
#if !defined(CONFIG_PPC) && !defined(CONFIG_ARCH_MX6)
    /*
     * timer_init() does not exist on PPC systems. The timer is initialized
     * and enabled (decrementer) in interrupt_init() here.
     */
    timer_init();   // 空函数
#endif

    /* Not Used */
    if (CONFIG_IS_ENABLED(BLOBLIST)) {
        ret = bloblist_init();
        if (ret) {
            debug("%s: Failed to set up bloblist: ret=%d\n",
                  __func__, ret);
            puts(SPL_TPL_PROMPT "Cannot set up bloblist\n");
            hang();
        }
    }

    /* Not Used */
    if (CONFIG_IS_ENABLED(HANDOFF)) {
        int ret;

        ret = setup_spl_handoff();
        if (ret) {
            puts(SPL_TPL_PROMPT "Cannot set up SPL handoff\n");
            hang();
        }
    }

#if CONFIG_IS_ENABLED(BOARD_INIT)
    spl_board_init();
#endif

/* Not Used */
#if defined(CONFIG_SPL_WATCHDOG_SUPPORT) && CONFIG_IS_ENABLED(WDT)
    initr_watchdog();
#endif

    /* Not Used */
    if (IS_ENABLED(CONFIG_SPL_OS_BOOT) || CONFIG_IS_ENABLED(HANDOFF) ||
        IS_ENABLED(CONFIG_SPL_ATF))
        dram_init_banksize();

    bootcount_inc(); // 空函数

    memset(&spl_image, '\0', sizeof(spl_image));

/* Not Used */
#ifdef CONFIG_SYS_SPL_ARGS_ADDR
    spl_image.arg = (void *)CONFIG_SYS_SPL_ARGS_ADDR;
#endif

    spl_image.boot_device = BOOT_DEVICE_NONE;
    /**
     *  通过CTRLMMR_WKUP_DEVSTAT和CTRLMMR_MAIN_DEVSTAT寄存器获取启动配置
     *  *(u32 *)(0x41cffbfc) 0: PRIMARY 1: BACKUP 由ROM Code控制
     */
    board_boot_order(spl_boot_list);

```

```
    /**
     * @brief 初始化启动设备，加载镜像文件，解析镜像文件，保存镜像hea信息在spl_image中
     */
    if (boot_from_devices(&spl_image, spl_boot_list,
                  ARRAY_SIZE(spl_boot_list))) {
        puts(SPL_TPL_PROMPT "failed to boot from all boot devices\n");
        hang();
    }

    spl_perform_fixups(&spl_image); // 空函数
    /* Not Used */
    if (CONFIG_IS_ENABLED(HANDOFF)) {
        ret = write_spl_handoff();
        if (ret)
            printf(SPL_TPL_PROMPT
                   "SPL hand-off write failed (err=%d)\n", ret);
    }
    /* Not Used */
    if (CONFIG_IS_ENABLED(BLOBLIST)) {
        ret = bloblist_finish();
        if (ret)
            printf("Warning: Failed to finish bloblist (ret=%d)\n",
                   ret);
    }

#ifdef CONFIG_CPU_V7M
    spl_image.entry_point |= 0x1;
#endif
    switch (spl_image.os) {
    case IH_OS_U_BOOT:
        debug("Jumping to U-Boot\n");
        break;
#if CONFIG_IS_ENABLED(ATF)
    case IH_OS_ARM_TRUSTED_FIRMWARE:
        debug("Jumping to U-Boot via ARM Trusted Firmware\n");
        spl_fixup_fdt(spl_image.fdt_addr);
        spl_invoke_atf(&spl_image);
        break;
#endif
#if CONFIG_IS_ENABLED(OPTEE)
    case IH_OS_TEE:
        debug("Jumping to U-Boot via OP-TEE\n");
        spl_optee_entry(NULL, NULL, spl_image.fdt_addr,
                (void *)spl_image.entry_point);
        break;
#endif
#if CONFIG_IS_ENABLED(OPENSBI)
    case IH_OS_OPENSBI:
        debug("Jumping to U-Boot via RISC-V OpenSBI\n");
        spl_invoke_opensbi(&spl_image);
        break;
#endif
#ifdef CONFIG_SPL_OS_BOOT
    case IH_OS_LINUX:
        debug("Jumping to Linux\n");
#if defined(CONFIG_SYS_SPL_ARGS_ADDR)
        spl_fixup_fdt((void *)CONFIG_SYS_SPL_ARGS_ADDR);
#endif
        spl_board_prepare_for_linux();
        jump_to_image_linux(&spl_image);
#endif
    default:
        debug("Unsupported OS image.. Jumping nevertheless..\n");
    }
#if CONFIG_VAL(SYS_MALLOC_F_LEN) && !defined(CONFIG_SYS_SPL_MALLOC_SIZE)
    debug("SPL malloc() used 0x%lx bytes (%ld KB)\n", gd->malloc_ptr,
          gd->malloc_ptr / 1024);
#endif
    bootstage_mark_name(spl_phase() == PHASE_TPL ? BOOTSTAGE_ID_END_TPL :
                BOOTSTAGE_ID_END_SPL, "end " SPL_TPL_NAME);
#ifdef CONFIG_BOOTSTAGE_STASH
    ret = bootstage_stash((void *)CONFIG_BOOTSTAGE_STASH_ADDR,
                  CONFIG_BOOTSTAGE_STASH_SIZE);
    if (ret)
        debug("Failed to stash bootstage: err=%d\n", ret);
#endif

    debug("loaded - jumping to U-Boot...\n");
    /* turn off D-cache */
    spl_board_prepare_for_boot();
```

```
    /* 跳到 spl_image->entry_point中运行 */
    jump_to_image_no_args(&spl_image);
}
```

5.2 U-Boot proper

common/board_f.c:

```
void board_init_f(ulong boot_flags)
{
    gd->flags = boot_flags;
    gd->have_console = 0;
    /* init_sequence_f 包含一系列的初始化函数，例如 DSRAM 和 uart，为 board_init_r 函数运行做准备 */
    if (initcall_run_list(init_sequence_f))
        hang();

#if !defined(CONFIG_ARM) && !defined(CONFIG_SANDBOX) && \
        !defined(CONFIG_EFI_APP) && !CONFIG_IS_ENABLED(X86_64) && \
        !defined(CONFIG_ARC)
    /* NOTREACHED - jump_to_copy() does not return */
    hang();
#endif
}
```

common/board_r.c:

```
void board_init_r(gd_t *new_gd, ulong dest_addr)
{
    /*
     * Set up the new global data pointer. So far only x86 does this
     * here.
     * TODO(sjg@chromium.org): Consider doing this for all archs, or
     * dropping the new_gd parameter.
     */
#if CONFIG_IS_ENABLED(X86_64)
    arch_setup_gd(new_gd);
#endif

#ifdef CONFIG_NEEDS_MANUAL_RELOC
    int i;
#endif

#if !defined(CONFIG_X86) && !defined(CONFIG_ARM) && !defined(CONFIG_ARM64)
    gd = new_gd;
#endif
    gd->flags &= ~GD_FLG_LOG_READY;

#ifdef CONFIG_NEEDS_MANUAL_RELOC
    for (i = 0; i < ARRAY_SIZE(init_sequence_r); i++)
        init_sequence_r[i] += gd->reloc_off;
#endif
    /* init_sequence_r 包含一系列的初始化函数 */
    if (initcall_run_list(init_sequence_r))
        hang();

    /* NOTREACHED - run_main_loop() does not return */
    hang();
}
```

5.3 Linux 跳转

AArch64 异常模型由多个异常级（EL0 - EL3）组成，对于 EL0 和 EL1 异常级有对应的安全和非安全模式。EL2 是系统管理级，且仅存在于非安全模式下。EL3 是最高特权级，且仅存在于安全模式下。

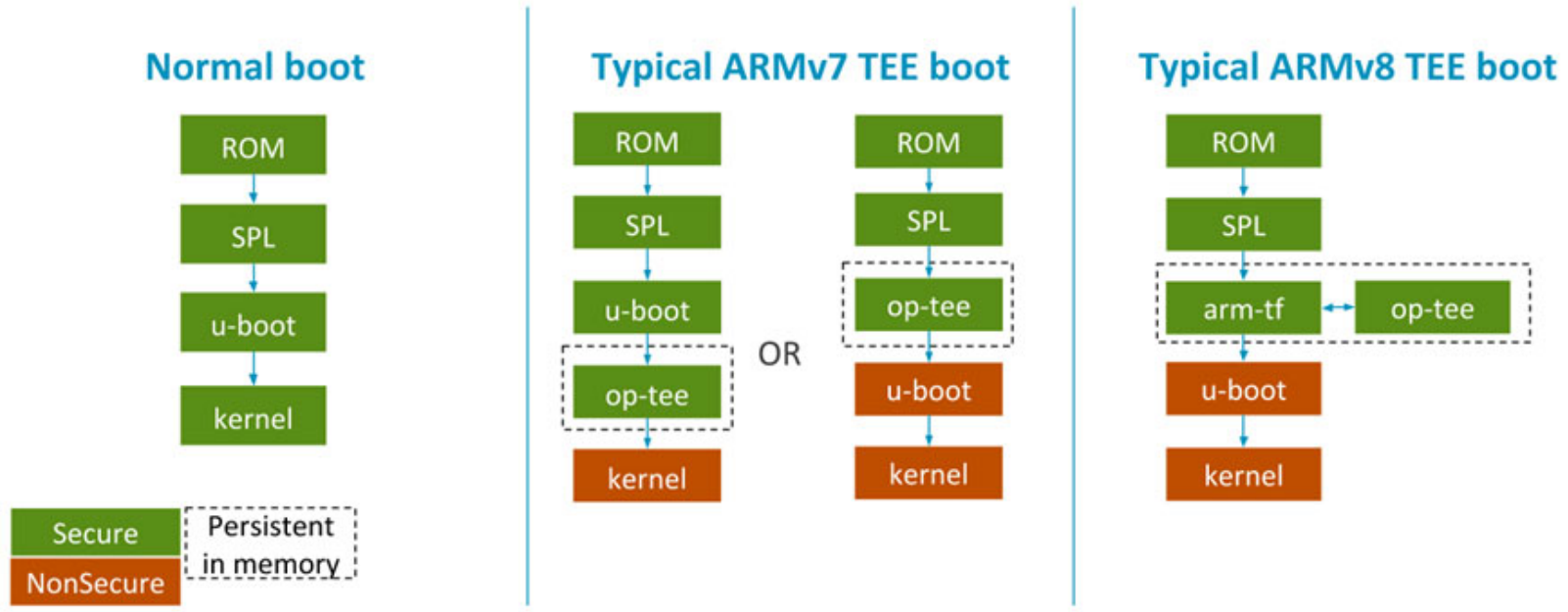
基本上，bootload（至少）应实现以下操作：

1. 设置和初始化 RAM
2. 设置设备树数据
    - 设备树数据块（dtb）必须 8 字节对齐，且大小不能超过 2MB
3. 调用内核映像
    - 内核映像必须被放置在任意一个可用系统内存 2MB 对齐基址的 text_offset 字节处，并从该处被调用。

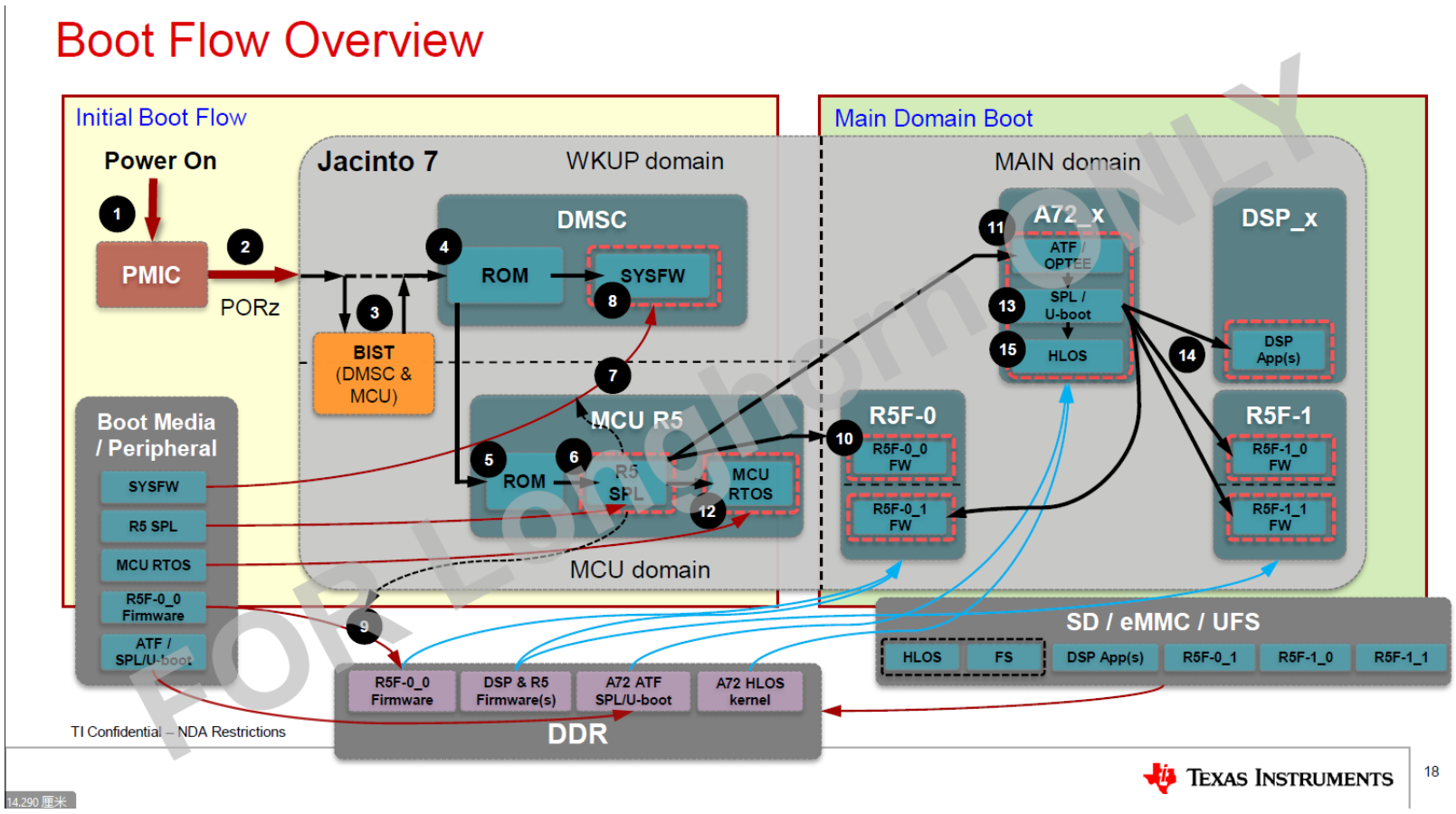
五、Linux

待添加

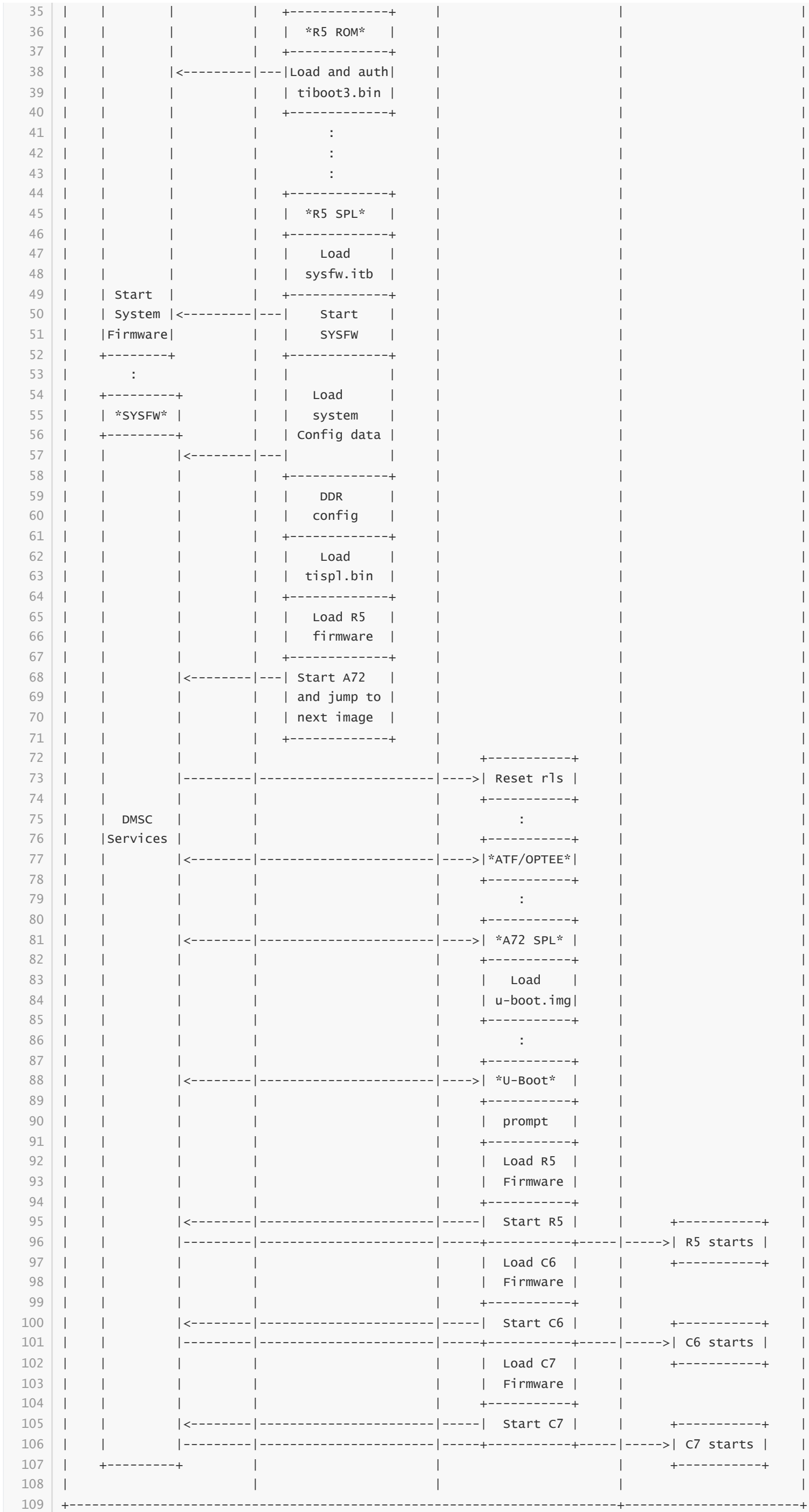
SPL启动



在基于 ARMv8 的处理器上，TEE（Trusted Execution Environment，可信执行环境） 引导流程：SPL 加载 ATF（ARM Trusted firmware，ARM 可信固件）、OP-TEE 和 U-Boot，然后SPL 跳转到 ATF，该固件随后将控制权交给 OP-TEE，OP-TEE 又在非安全上下文中跳转到 U-Boot。




```
Introduction:
-------------

The device is partitioned into three functional domains, each containing
specific processing cores and peripherals:
1. Wake-up (WKUP) domain:
    - Device Management and Security Controller (DMSC)
2. Microcontroller (MCU) domain:
    - Dual Core ARM Cortex-R5F processor
3. MAIN domain:
    - Dual core 64-bit ARM Cortex-A72
    - 2 x Dual cortex ARM Cortex-R5 subsystem
    - 2 x C66x Digital signal processor sub system
    - C71x Digital signal processor sub-system with MMA.


Boot Flow:
----------
Boot flow is similar to that of AM65x SoC and extending it with remoteproc
support. Below is the pictorial representation of boot flow:

+-------------------------------------------------------------------------+-----------------------+
|        DMSC             |      MCU R5           |        A72            |  MAIN R5/C66x/C7x     |
+-------------------------------------------------------------------------+-----------------------+
|    +--------+           |                       |                       |                       |
|    |  Reset |           |                       |                       |                       |
|    +--------+           |                       |                       |                       |
|         :               |                       |                       |                       |
|    +--------+           |   +-----------+       |                       |                       |
|    | *ROM*  |-----------|-->| Reset rls |       |                       |                       |
|    +--------+           |   +-----------+       |                       |                       |
|    |        |           |         :             |                       |                       |
|    |  ROM   |           |         :             |                       |                       |
|    |services|           |         :             |                       |                       |
```



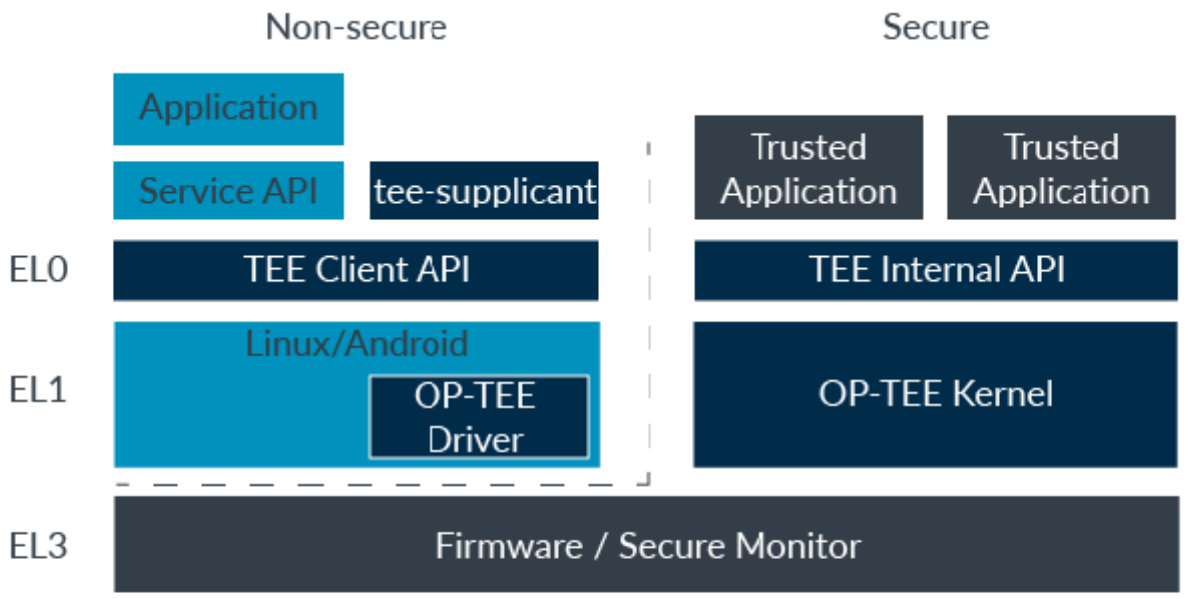
- Here DMSC acts as master and provides all the critical services. R5/A72 requests DMSC to get these services done as shown in the above diagram.

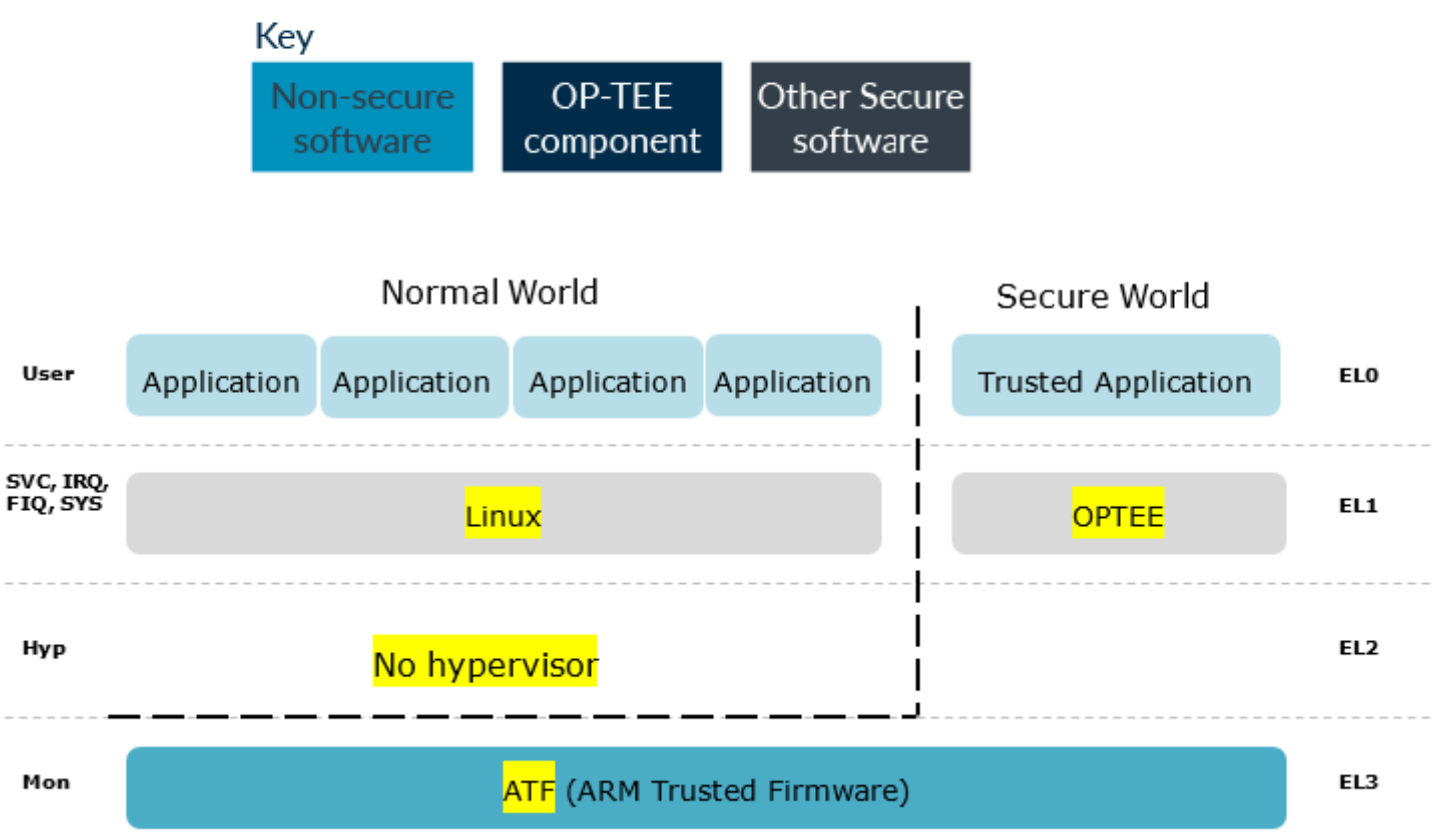
Sources:

```
--------
1. SYSFW:
    Tree: git://git.ti.com/k3-image-gen/k3-image-gen.git
    Branch: master

2. ATF:
    Tree: https://github.com/ARM-software/arm-trusted-firmware.git
    Branch: master

3. OPTEE:
    Tree: https://github.com/OP-TEE/optee_os.git
    Branch: master

4. U-Boot:
    Tree: https://gitlab.denx.de/u-boot/u-boot
    Branch: master

Build procedure:
----------------
1. SYSFW:
$ make CROSS_COMPILE=arm-linux-gnueabihf-

2. ATF:
$ make CROSS_COMPILE=aarch64-linux-gnu- ARCH=aarch64 PLAT=k3 TARGET_BOARD=generic SPD=opteed

3. OPTEE:
$ make PLATFORM=k3-j721e CFG_ARM64_core=y

4. U-Boot:

4.1. R5:
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- j721e_evm_r5_defconfig O=/tmp/r5
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- O=/tmp/r5

4.2. A72:
$ make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- j721e_evm_a72_defconfig O=/tmp/a72
$ make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- ATF=<path to ATF dir>/build/k3/generic/release/bl31.bin
TEE=<path to OPTEE OS dir>/out/arm-plat-k3/core/tee-pager.bin O=/tmp/a72

Target Images
--------------
Copy the below images to an SD card and boot:
- sysfw.itb from step 1
- tiboot3.bin from step 4.1
- tispl.bin, u-boot.img from 4.2

Image formats:
--------------

- tiboot3.bin:
                +-----------------------+
                |        X.509          |
                |      Certificate      |
                | +-------------------+ |
                | |                   | |
                | |        R5         | |
                | |   u-boot-spl.bin  | |
                | |                   | |
                | +-------------------+ |
                | |                   | |
                | |     FIT header    | |
                | | +---------------+ | |
                | | |               | | |
                | | |   DTB 1...N   | | |
                | | +---------------+ | |
                | +-------------------+ |
                +-----------------------+

- tispl.bin
                +-----------------------+
                |                       |
                |       FIT HEADER      |
                | +-------------------+ |
                | |                   | |
                | |      A72 ATF      | |
                | +-------------------+ |
                | |                   | |
                | |     A72 OPTEE     | |
                | +-------------------+ |
                | |                   | |
```

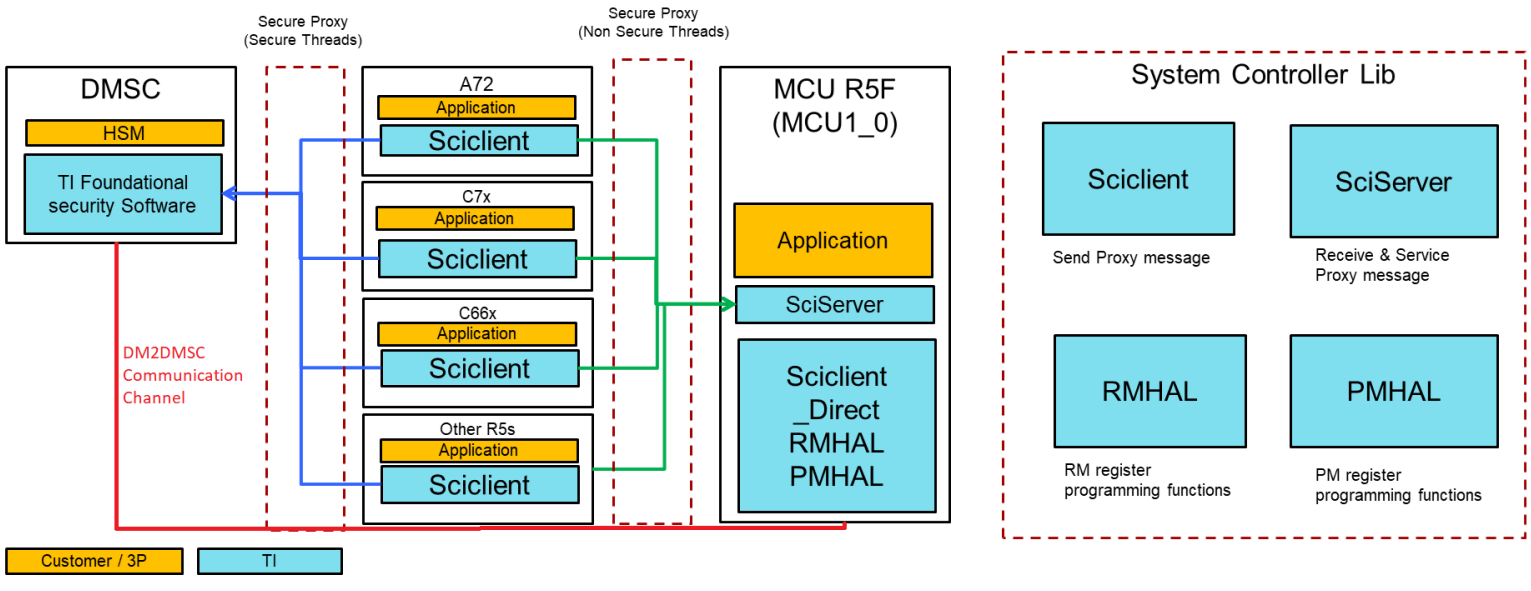
```
                | |      A72 SPL      | |
                | +-------------------+ |
                | |                   | |
                | |   SPL DTB 1...N   | |
                | +-------------------+ |
                +-----------------------+

- sysfw.itb
                +-----------------------+
                |                       |
                |       FIT HEADER      |
                | +-------------------+ |
                | |                   | |
                | |     sysfw.bin     | |
                | +-------------------+ |
                | |                   | |
                | |    board config   | |
                | +-------------------+ |
                | |                   | |
                | |     PM config     | |
                | +-------------------+ |
                | |                   | |
                | |     RM config     | |
                | +-------------------+ |
                | |                   | |
                | |    Secure config  | |
                | +-------------------+ |
                +-----------------------+


```

ARM 异常级别





DMSC 功能



Q&A

1. spl 的 DM 和 U-boot 的有什么不同?

> 使用的是同一个文件，有`u-boot,dm-spl`标识的节点将会被 spl 编译，u-boot 是全部编译。

2. DDR 的初始化在什么时候?

> 都在 tiboot3.bin 中。每次重新加载程序都需要重新初始化 DDR 一遍。

参考链接

- OP-TEE
- U-Boot wiki
- U-Boot SPL编译流程
- u-boot启动流程分析
- arm64/booting.txt
- Linux ARM64的启动过程
- Processor SDK RTOS J721E
- Processor SDK Linux J721E