Jacitno7 Boot Loader 及 PSDKRA 启动方案

Fredy Zhang; Fan Zhang *EP FAE*

摘要

Jacinto™ 7 TDA4x 是目前 TI 最新一代的汽车处理器，面向 ADAS 和自动驾驶车辆 (AV) 应用， 并且建立在 TI 在 ADAS 处理器市场十余年领先地位中积累的广泛市场知识基础之上。TDA4x 以业界领先的功耗/性能比为传统和深度学习算法提供高性能计算，并具有很高的系统集成度，从而使支持集中式 ECU 或多种传感器的高级汽车平台实现可扩展性和更低的成本。

TI Jacinto™ 7 家族的处理器，基于异构、可扩展的架构开发，拿 TDA4VM 的处理器来说，该处理器包含了 TI DSP 处理器（C66/C7x）、Cortex A72、Main 域 Cortex R5F、MCU 域 Cortex R5F、图形处理器 GPU 等核，属于多核异构的架构。多核异构的优点是采用适合的核做擅长的事，再加上专用硬件加速器也可处理特定任务，从而在性能、功耗和成本 上达到最佳平衡。但是，由于采用了多核异构的架构，因系统需求的差异，启动流程也有一定的差异性，多核异构的启动流程比单核和多核同构的处理器会复杂些。

针对系统需求的不同，对于应用来讲，我们可以看到系统可能有如下的需求：Early CAN message response、Early Splash Screen、Early Camera(RVC)、Early Linux user space、Early Video(Boot animation)等。不同的应用可以部署在不同的核，不同核的启动时序影响着系统需求是否能实现。针对系统的应用需求，我们应设计合理的 Soc 启动流程来满足系统的需求。

Bootloader 是在操作系统运行之前执行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射表，从而建立适当的系统软硬件环境，为最终调用操作系统内核做好准备。在 Jacinto7 有两种 bootloader 可以选择 SPL 或 SBL。PSDKRA+PSDKLA 通常作为 ADAS 应用的基础组件，运行在 Jacinto7 Soc 之上。

本手册旨在对 Jacinto7 Bootloader 和 PSDKRA+PSDKLA 的 SPL 启动流程进行介绍，重点介绍了 PSDKRA+PSDKLA 的 SBL 启动流程的实现，为使用 TI Jacinto™ 7 的人员提供 Bootloader 选择及实现参考。

修改记录

Version	Date	Author	Notes
1.0	March 2022	Fredy Zhang	First release

目录

图

表

1. Soc 启动流程简介

Jacinto7 Soc 里面不仅集成了中央处理单元，还集成了 DMSC 设备管理和安全控制器，MCU 等和众多的外设模块。如图 1 所示，从一个 Soc 的初始化流程来看，前两步是面向硬件的，但具体流程也与设备的系统配置（Boot Mode）管脚的设置有关。 具体的初始化流程如下：

图 **1**. Device Initialization Process

1. **Pre-initialization:** 电源、时钟、一些控制连接和启动配置管脚必须保持在所需的逻辑电平；
2. **Power, clock, reset ramp sequence**：根据电源管理芯片（PMIC）指定的时序（Soc 启动时序）配置 Power、Clock、Reset ramp Sequence；
3. **ROM Code**： 两个 ROM Code 同时执行 (DMSC ROM & MCU R5 ROM)， 从指定存储空间寻找、下载、执行外部第一个程序 (SPL / SBL)
4. **Initial software (SPL or SBL)** ：加载、准备并将控制权传递给应用程序软件或高级操作系统 (HLOS) 的软件
5. **High-Level Operating System (HLOS) or App:** 运行操作系统或软件应用在主处理器。

如图 2：TDA4VM 系统框图所示，Jacinto7 Soc MCU Island 有双核的 R5F 和 DMSC，其中，MCU R5F 负责了 Soc 的启动管理，DMSC 负责了 Soc 资源管理和安全控制器。后文中用到的一些术语解释如下：

1. **DMSC:** Device Management & Security Controller， Soc 复位后执行 DMSC ROM。
2. **MCU R5F**: 启动控制器，DMSC ROM CODE 复位 MCU 域的 Cortex-R5F 后，该核控制着系统的启动流程。然后，该核开始执行 MCU R5 ROM。
3. **R5 SPL** or **SBL**: 第二级的 bootloader，通过选择的 BOOT PIN 的配置，MCU 从指定的存储器（OSPI Flash/EMMC 等）加载 R5 SPL 或 SBL。然后，该 Image 被 DMSC 认证后开始执行。
4. **SYSFW** ： System Firmware for DSMC, 由 MCU R5 加载并由 DMSC 验证该镜像后，DMSC 开始执行它。
5. **ATF:** ARM Trusted Firmware, ARMv8 Cortex A 核的安全固件，执行安全监控并处理 A72 初始化，同时加载随后的安全和非安全的 A72 镜像。
6. **OPTEE**：Open Portable Trusted Execution Environment，由 Linaro 维护的开源 TEE。 用作在 A72 内核上运行的 Secure-world 操作系统。 作为 Cortex-A 内核的 ATF 初始化序列的一部分加载。
7. **A72 SPL**： Secondary program loader，SPL 由 ATF 加载， 作为在 A72 上启动的第一个非安全代码。
8. **U-boot**：用于 A72 HLOS 和主域 R5 和 DSP 内核上的软件的引导加载程序。

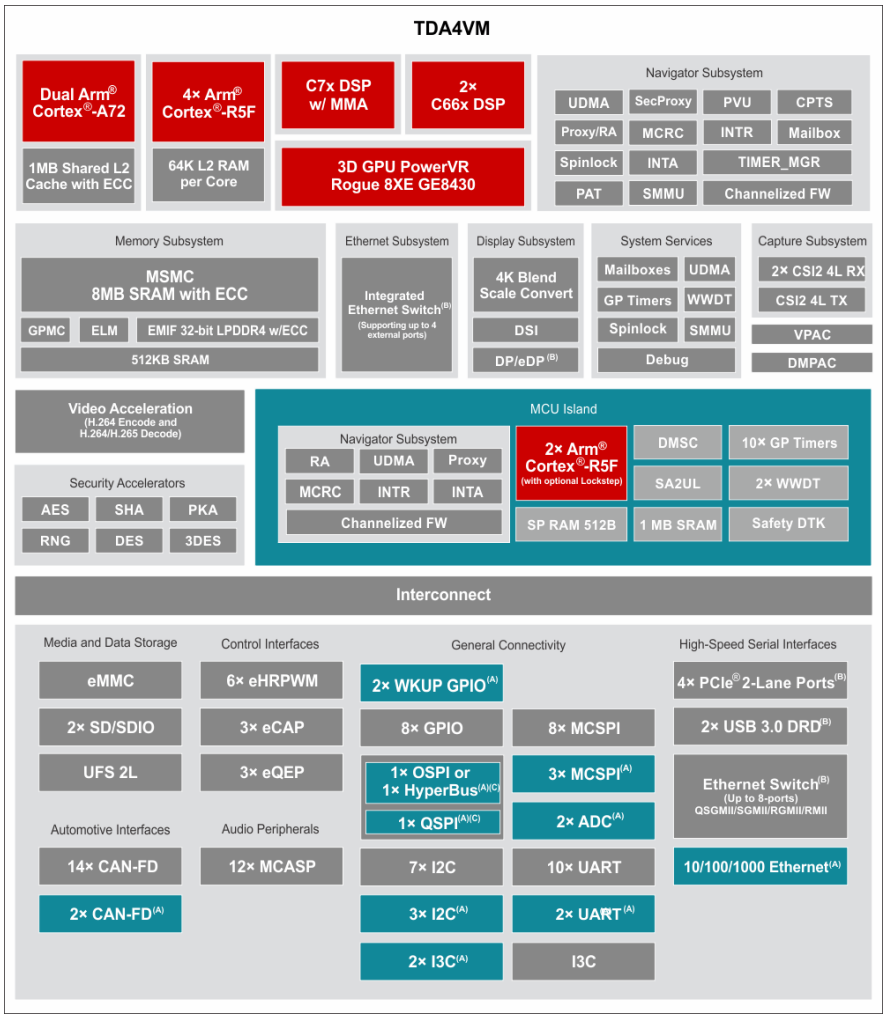

图 **2**. TDA4VM System Diagram

Jacinto7 Soc 里面集成了 A72，这里也简单介绍一下 ARM-v8 架构 Cortex A72 启动流程。 如图 3 ARM Boot Flow 所示：

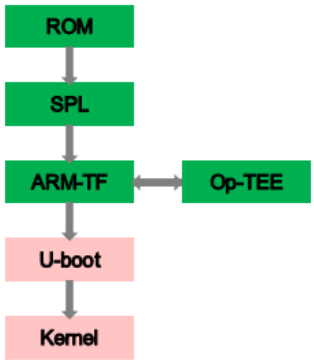

图 **3**. ARM Boot Flow

如图 4 所示 ARM BOOT FLOW， 在 Jacinto7 Soc 中，BL1 和 BL2 没有被集成（Jacinto7 集成了 DMSC&MCU R5 取代了 BL1 和 BL2 的功能）, BL31 执行 ATF，BL32 执行 OP-TEE，BL33 是 SPL/U-boot。

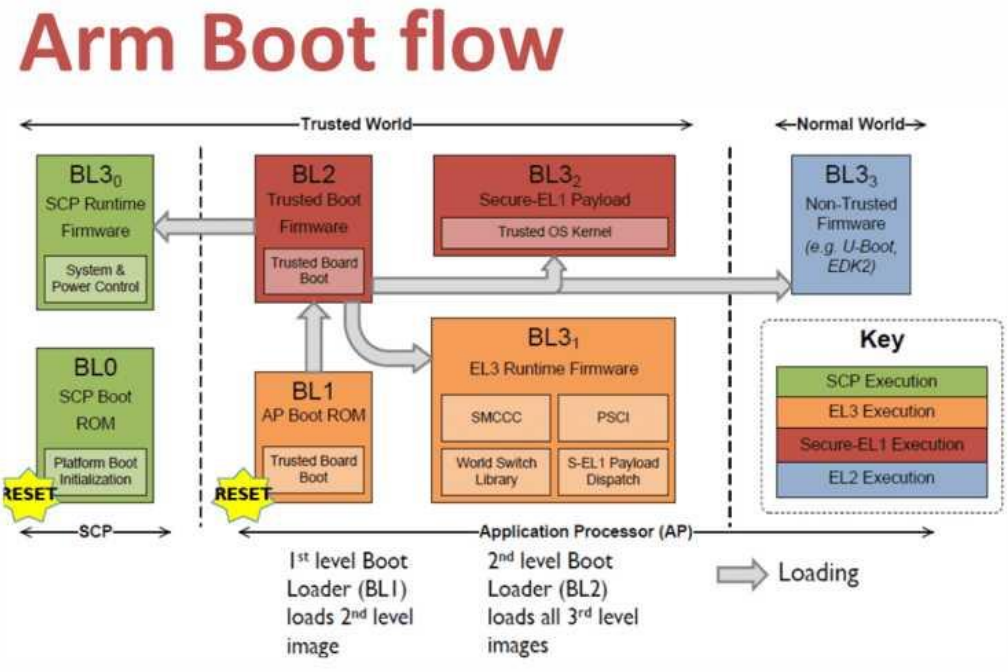

图 **4**. ARM Boot Flow [4]

本文后面的部分将以如表 2 Demo 环境进行介绍，对于其它 SDK 版本的环境，其中有疑问的地方，请查阅对应版本 SDK 中的说明。

表 1. Demo 环境

名称	内容
PSDKRA SDK	PROCESSOR-SDK-RTOS-J721E_08.01.00.11
PSDKLA	PROCESSOR-SDK-LINUX-J721E 08.01.00.07
PC 开发环境	Ubuntu18.04
EVM	TDA4VM EVM

Jacinto7 属于多核处理器，拥有 Cortex A72、C66x、C7x、Main Cortex R5F、MCU R5F 等。通常 A72 上运行 HLOS（Linux/QNX），其余核运行 RTOS，MCU 可以支持 AUTOSAR。PSDKRA+PSDKLA 通常作为 ADAS 应用的基础组件，运行在 Jacinto7 Soc 之上。除了 MCU R5F 是启动控制器，会首先启动并控制启动的流程。其余各个核的 Power 和 Reset 都可以独立控制。依据应用的不同需求可以灵活调整各个核的启动顺序，且当前 Jacinto7 支持 SPL 和 SBL 两种不同的 bootloader，因此，后文将通过不同的 Bootloader 介绍 Jacinto7 的启动流程。

2. PSDKRA+PSDKLA 的 SPL 启动流程

2.1. SPL 启动流程

这种方式的启动流程也是 SDK 默认的启动流程。Jacinto7 的启动流程相对而言比较复杂。SPL（Secondary Program Loader），这里所指的是运行 MCU R5 上的 SPL，意为第二级的 Bootloader，它的功能主要是硬件初始化，引导下一级的 bootloader 核或加载应用程序并运行。有第二级就有第一级的 Bootloader，第一级的 Bootloader，指 ROM 里面的程序，根据启动方式的选择，引导并加载 SPL，然后跳转到第二级的 bootloader，Jacinto7 Soc 里面的执行流程跟我们通常的理解稍有不同，其流程如图 4 所示，接下来我们将详细介绍：

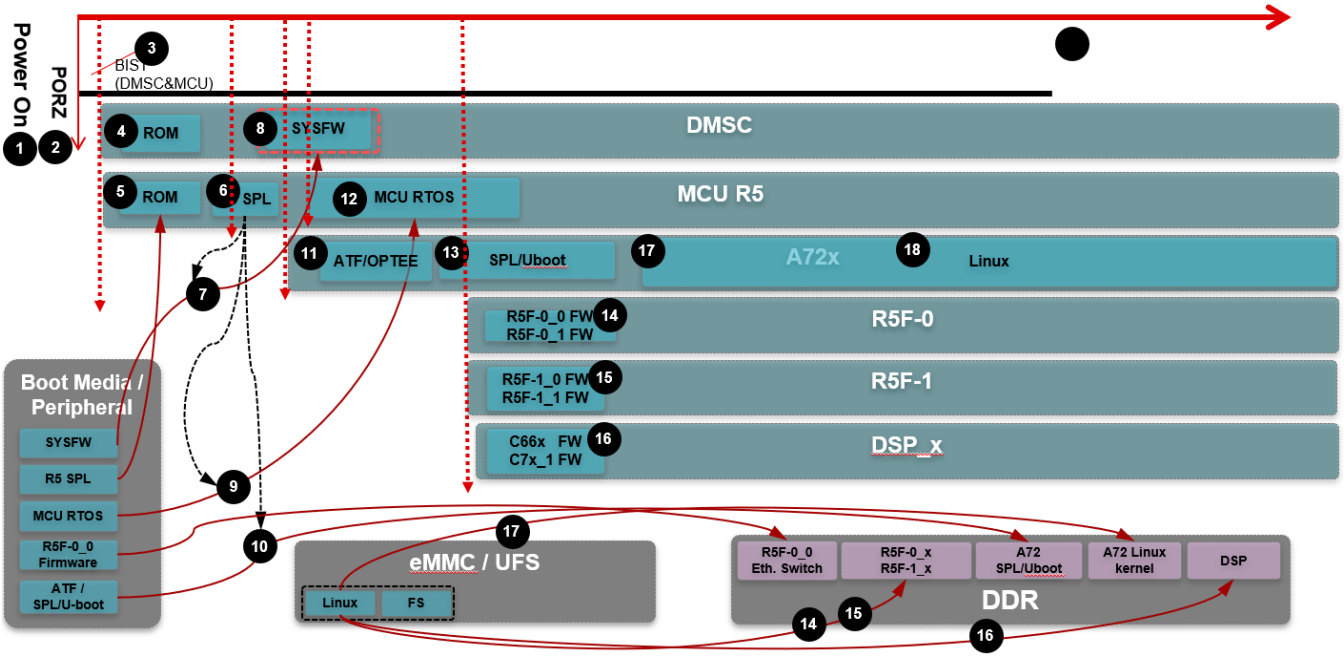

图 5. Jacinto7 TDA4VM SPL 启动流程

1. 系统上电
2. Soc PORZ
3. DMSC&MCU BIST （可选）
4. DMSC ROM 开始执行
5. MCU R5 ROM 开始执行
6. MCU R5 ROM 加载并运行 SPL
7. MCU R5 SPL（tiboot3.bin）加载 SYSFW
8. DMSC 认证并启动 SYSFW
9. MCU R5 SPL 加载 MCU App （包含在 A72 SPL 该 Image 里面包含了 ATF、OPTEE、MCU RTOS、A72 SPL）
10. MCU R5 SPL 加载 A72 SPL
11. ATF 启动并执行
12. MCU RTOS 启动 Sciclient Server 并运行 App
13. A72 ATF 加载 A72 SPL 并执行，然后加载 Uboot 执行
14. A72 uboot 加载 R5F 0 Image 并执行
15. A72 uboot 加载 R5F 1 Image 并执行

16. A72 uboot 加载 C66x 和 C7x Image 并执行
17. A72 uboot 加载 Kernel Image 并执行，然后挂载文件系统运行到 Linux 控制台。
18. 运行应用程序。

以上就是基于 SPL 的 Soc 启动过程。下一小节我们将介绍如何运行 TI Demo 基于上述启动流程运行 PSDKRA 的 demo。

2.2. 运行 PSDKRA Demo

上一章节描述了 TI PSDKLA+PSDKRA 默认 SDK 的启动流程，因此，非常容易运行 PSDKRA 的 demo。 参考链接：Run vision apps on EVM in Linux+RTOS mode (via SD card boot) 制作 SD 卡并配置环境。

然后，进行 Demo 环境初始化：

```
cd /opt/vision_apps
source ./vision_apps_init.sh
```

使用下面的命令运行具体的 Demo

```
./run_app_tidl.sh - Image classification demo (needs display)
./run_app_tidl_avp2.sh - Auto valet parking demo (needs display)
./run_app_dof.sh - Dense optical flow demo (needs display)
./run_app_stereo.sh - Stereo disparity demo (needs display)
./run_app_c7x.sh - C7x sample kernel demo
./run_app_srv.sh - 3D SRV 4 camera demo (needs display, Fusion1 board, 4x IMX390 camera)
./run_app_single_cam.sh - Single camera + VISS + Display demo (needs display, Fusion1 board, 1x
 IMX390 or compatible camera's)
./run_app_multi_cam.sh - Multi (4x) camera + VISS + Display demo (needs display, Fusion1 board,
 4x IMX390 or compatible camera's)
./vx_app_arm_opengl_mosaic.out - OpenGL + OpenVX test
./vx_app_linux_arm_ipc.out - inter processor communication test
./vx_app_linux_arm_mem.out - memory allocation test
./vx_app_tutorial.out - TI OpenVX tutorial
./vx_app_conformance.out - TI OpenVX conformance tests
```

3. PSDKRA+PSDKLA 的 SBL 启动流程

3.1. SBL 启动流程

SDK 里也提供了使用 SBL 启动 PSDKRA + PSDKLA 的方法。Jacinto7 的启动流程相对而言比较复杂。SBL（Secondary bootloader）是 TI 写的非常精简的 Bootloader，可以实现对外围设备进行配置，比如 DDR，可以加载并启动其它核。为了满足快速启动 MCU 执行相关的应用，MCU 可以先启动，然后使用 BOOT APP 进而引导其它应用程序。 在 SBL 启动流程中，SBL 可以直接加载 Linux 内核和 DTB。其流程如图 3 所示：

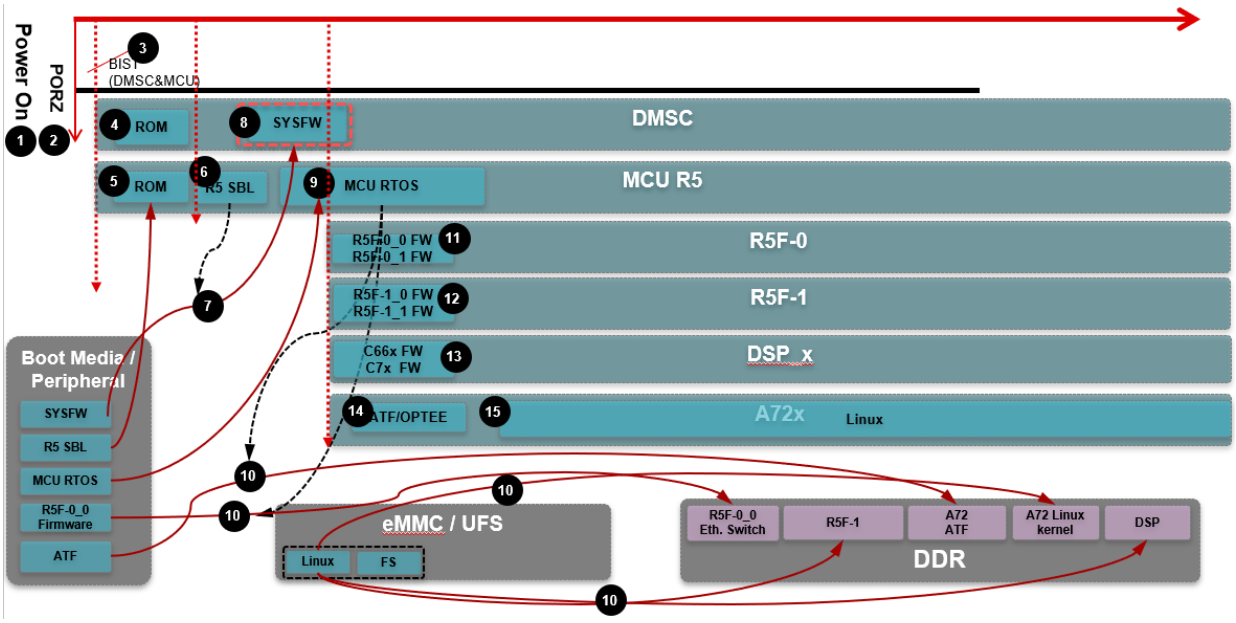

图 **6**. Jacinto7 TDA4VM SBL 启动流程

1. 系统上电
2. Soc PORZ
3. DMSC&MCU BIST （可选）
4. DMSC ROM 开始执行
5. MCU R5 ROM 开始执行
6. MCU R5 ROM 加载并运行 SBL
7. MCU R5 SBL（tiboot3.bin）加载 SYSFW
8. DMSC 认证并启动 SYSFW
9. MCU R5 SBL 加载 MCU Boot App, 运行 Sciclient Server。
10. MCU R5 Boot App 分别加载 Main R5F、DSP、ATF、DTB、Kernel Image
11. Main R5F 0 开始运行
12. Main R5F 1 开始运行
13. C66x、C7x 开始运行
14. ATF 开始运行
15. Kernel Image 开始执行，然后挂载文件系统运行到 Linux 控制台。
16. 运行应用程序。

以上就是 SBL 的启动过程。下一小节我们将介绍如何运行 TI Demo 基于上述启动流程运行 PSDKRA 的 demo。

3.2. 运行 PSDKRA Demo

上一章节描述了 TI PSDKLA+PSDKRA 的 SBL 的启动流程，MCU R5 BOOT APP 负责了其余各个核的加载和启动，其 Boot App 流程如图 7 所示。值得注意的是在上面的启动流程中，对于 A72 上的 Linux 启动流程，我们直接是从 BOOT APP 加载 Linux 的 DTB、ATF、Kernel Image。这里跟 SPL 的启动流程不同（没有加载和运行 A72 的 SPL 和 Uboot.img 镜像）从启动流程上讲，该流程对 A72 Linux 的启动流程进行了优化，启动时间也短。另外一个需要注意的 DTB，这里面的 dtb 需要更新 bootargs，如果需要运行 PSDKRA 的 Demo，还需要对 DTB 做相应的修改。

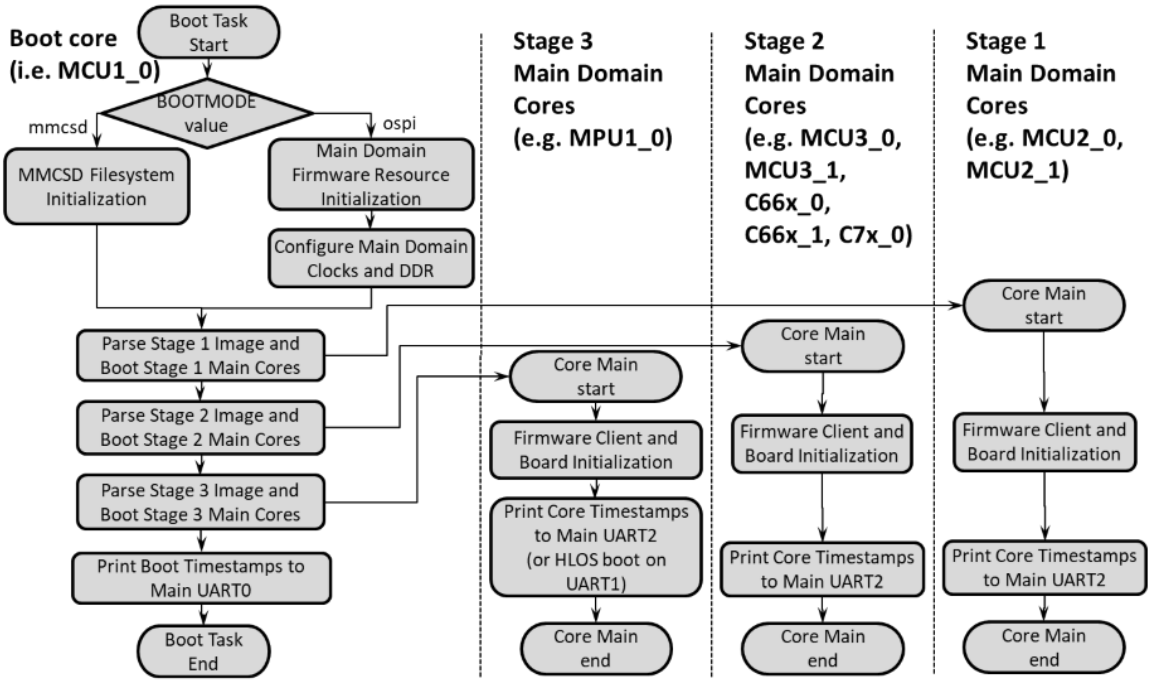

图 7. MCU1_0 Boot Task and Main Domain Applications

参考如下步骤来实现 SBL 启动并运行 PSDKRA 的 Demo（以 SD 卡启动为例）：

1. 编译 SBL: 编译 PSDKRA PDK sbl_mmcsd_img 生成 sbl_mmcsd_img_mcu1_0_release.bin，然后拷贝到 SD 卡 boot 分区 tiboot3.bin，编译方法如下：

```
Ubuntu18$ cd $PDK_PATH/packages/ti/build
Ubuntu18$ make BOARD=j721e_evm CORE=mcu1_0 BUILD_PROFILE= release pdk_libs      -sj
Ubuntu18$ make BOARD=j721e_evm CORE=mcu1_0 BUILD_PROFILE= release sbl_mmcsd_img -sj
```

2. 编译 can boot app: 在 PSDKRA MCUSW 里面编译出 Image（SDK 8.1 开始，也可以从 vision_app 下编译该 image，其命令是 sbl_mcusw_bootimage_sd，编译出的 image 在 $PATH_VISION_SDK_BUILD/out/sbl_bootfiles/app）：
   can_boot_app_mcu_rtos_mcu1_0_release.appimage, 然后拷贝到 SD 卡 boot 分区 app，编译方法如下：

```
Ubuntu18$ cd $MCUSW_PATH/build
Ubuntu18$ make -C can_boot_app_mcu_rtos HLOSBOOT=linux BOOTMODE=mmcsd SOC=j721e BOARD=j721e_
evm CORE=mcu1_0 BUILD_OS_TYPE=freertos CANFUNC=disabled BUILD_PROFILE=release  -sj
```

3. 编译 DTB：用 SBL + BOOT APP 方式直接启动 Linux，需要更新 DTB 的 bootargs 和在 k3-j721e-common-proc-board.dts 里面添加 k3-j721e-vision-apps.dts 的内容，最终将其编译为一个 dtb，以便支持 PSDKRA 的 demo 的运行环境。编译 k3-j721e-common-proc-board.dtb 并拷贝到 PSDKRA/board-support/prebuilt-images/base-board.dtb。其 bootargs 配置 k3-j721e-common-proc-board.dts 如下：

```
bootargs = "console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02800000";
```

更新到如下内容

```
bootargs = "console=ttyS2,115200n8 earlycon=ns16550a,mmio32,0x02800000 root=/dev/mmcblk1p2 r
w rootfstype=ext4 rootwait";
```

4. 转换 Linux 的 ATF、dtb、kernel Image

```
Ubuntu18$ cd PSDKRA/mcusw/mcuss_demos/boot_app_mcu_rtos/main_domain_apps/scripts/hlos/ && ./
constructappimageshlos.sh j721e_evm
```

转换完成后你将得到 atf_optee.appimage、tidtb_linux.appimage、tikernelimage_linux.appimage,这些 Image 也需要拷贝到 SD 卡 boot 分区。

5. Combine C66/C7x/R5F Core Image: 将 MCU2_0 和 MCU2_1 的 Image 合成为 lateapp1。将 C6x_1、C6x_2 和 C7x 合成 lateapp2.

```
Ubuntu18$ cd PSDKRA/vision_apps
Ubuntu18$ make sbl_vision_apps_bootimage
```

经过上述的步骤，我们就准备了 SBL 启动所需要的文件。接下来就需要将这些 Image 拷贝到 SD 中进行验证：

```
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_bootfiles/tiboot3.bin $SD_BOOT
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_bootfiles/tifs.bin    $SD_BOOT
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_bootfiles/app         $SD_BOOT
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_combined_bootfiles/atf_optee.appimage
             $SD_BOOT
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_combined_bootfiles/tidtb_linux.appimage $SD_BOOT
Ubuntu18$ cp $(SJ_PATH_VISION_SDK_BUILD)/out/sbl_combined_bootfiles/tikernelimage_linux.appi
mage $SD_BOOT
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_bootfiles/lateapp1    $SD_BOOT
Ubuntu18$ cp $PATH_VISION_SDK_BUILD/out/sbl_bootfiles/lateapp2    $SD_BOOT
```

MCU 的启动 log 如下：

```
SBL Revision: 01.00.10.01 (Mar 30 2022 - 08:00:22)
TIFS  ver: 21.9.1--v2021.09a (Terrific Lla
Starting Sciserver..... PASSED

MCU R5F App started at 0 usecs
Calling Sciclient_procBootRequestProcessor, ProcId 0x6...
Calling Sciclient_procBootRequestProcessor, ProcId 0x7...
Loading BootImage

 MMCBootImageLate: fp 0x 0x41cbf300, fileName is 0:/lateapp1

 Called SBL_MulticoreImageParse, status = 0
BootImage completed, status = 0
Sciclient_procBootReleaseProcessor, ProcId 0x6...
Sciclient_procBootReleaseProcessor, ProcId 0x7...
SBL_SlaveCoreBoot completed for Core ID#6, Entry point is 0x0
SBL_SlaveCoreBoot completed for Core ID#7, Entry point is 0x0
Calling Sciclient_procBootRequestProcessor, ProcId 0x8...
Calling Sciclient_procBootRequestProcessor, ProcId 0x9...
Calling Sciclient_procBootRequestProcessor, ProcId 0x3...
Calling Sciclient_procBootRequestProcessor, ProcId 0x4...
Calling Sciclient_procBootRequestProcessor, ProcId 0x30...
Loading BootImage

 MMCBootImageLate: fp 0x 0x41cbf300, fileName is 0:/lateapp2

 Called SBL_MulticoreImageParse, status = 0
BootImage completed, status = 0
Sciclient_procBootReleaseProcessor, ProcId 0x8...
Sciclient_procBootReleaseProcessor, ProcId 0x9...
Sciclient_procBootReleaseProcessor, ProcId 0x3...
Sciclient_procBootReleaseProcessor, ProcId 0x4...
Sciclient_procBootReleaseProcessor, ProcId 0x30...
SBL_SlaveCoreBoot completed for Core ID#10, Entry point is 0xa8d00c00
SBL_SlaveCoreBoot completed for Core ID#11, Entry point is 0xa9d00c00
SBL_SlaveCoreBoot completed for Core ID#12, Entry point is 0xaa200000
Calling Sciclient_procBootRequestProcessor, ProcId 0x20...
Loading BootImage

 MMCBootImageLate: fp 0x 0x41cbf300, fileName is 0:/atf_optee.appimage

 Called SBL_MulticoreImageParse, status = 0

 MMCBootImageLate: fp 0x 0x41cbf300, fileName is 0:/tikernelimage_linux.appimage
```

```
 Called SBL_MulticoreImageParse, status = 0

 MMCBootImageLate: fp 0x 0x41cbf300, fileName is 0:/tidtb_linux.appimage

 Called SBL_MulticoreImageParse, status = 0
BootImage completed, status = 0
Sciclient_procBootReleaseProcessor, ProcId 0x20...
SBL_SlaveCoreBoot completed for Core ID#0, Entry point is 0x70000000
Boot App: Started at 3164 usec
Boot App: Total Num booted cores = 6
Boot App: Booted Core ID #6 at 1583155 usecs
Boot App: Booted Core ID #7 at 1646155 usecs
Boot App: Booted Core ID #10 at 6264155 usecs
Boot App: Booted Core ID #11 at 6335155 usecs
Boot App: Booted Core ID #12 at 6406155 usecs
Boot App: Booted Core ID #0 at 17573156 usecs

MCU Boot Task started at 3164 usecs and finished at 21929155 usecs
```

等到 Linux 运行到 Console,运行如下初始化 PSDKRA Demo 运行环境：

```
cd /opt/vision_apps
source ./vision_apps_init.sh
```

使用下面的命令运行具体的 Demo

```
./run_app_tidl.sh - Image classification demo (needs display)
./run_app_tidl_avp2.sh - Auto valet parking demo (needs display)
./run_app_dof.sh - Dense optical flow demo (needs display)
./run_app_stereo.sh - Stereo disparity demo (needs display)
./run_app_c7x.sh - C7x sample kernel demo
./run_app_srv.sh - 3D SRV 4 camera demo (needs display, Fusion1 board, 4x IMX390 camera)
./run_app_single_cam.sh - Single camera + VISS + Display demo (needs display, Fusion1 board,
 1x IMX390 or compatible camera's)
./run_app_multi_cam.sh - Multi (4x) camera + VISS + Display demo (needs display, Fusion1 boa
rd, 4x IMX390 or compatible camera's)
./vx_app_arm_opengl_mosaic.out - OpenGL + OpenVX test
./vx_app_linux_arm_ipc.out - inter processor communication test
./vx_app_linux_arm_mem.out - memory allocation test
./vx_app_tutorial.out - TI OpenVX tutorial
./vx_app_conformance.out - TI OpenVX conformance tests
```

4. 总结

本文通过两种不同的 bootloader（SBL 和 SPL）来介绍 Jaicnto7 的启动流程。Jacitno7 PSDKLA+PSDKRA 默认采用了 SPL 的 bootloader。在 ADAS 应用中，尤其使用内部 MCU 的场景下，推荐使用 PSDKRA+PSDKLA 的 SBL 启动流程。

Jacinto7 启动流程多种多样，用户可以依据不同的应用需求（Early CAN message response、Early Splash Screen、Early Camera (RVC)、Early Linux user space、Early Video (Boot animation) 等），灵活设计启动流程，从而满足系统的需求。

5. 参考

1. https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/08_01_00_13/exports/docs/psdk_rtos/docs/user_guide/index.html
2. https://software-dl.ti.com/jacinto7/esd/processor-sdk-linux-jacinto7/08_01_00_07/exports/docs/devices/J7/linux/index.html
3. https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/08_01_00_13/exports/docs/vision_apps/docs/user_guide/RUN_INSTRUCTIONS.html
4. https://wiki.loliot.net/docs/linux/linux-uboot/embedded-linux-boot-process/

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265