Application Note

环视系统 ECU

Nikhil Dasan, J Keerthy, Brijesh Jadav

摘要

环视应用涉及多个并行运行的内核。MCU R5F 从引导器件开始，A72 触发 vision apps 应用，R5F 启动摄像机和显示器。使用带有 vision_apps 的默认 Linux SDK 时，实现环视所需的时间约为 25 秒，如 图 1-1 所示。本应用手册重点介绍如何优化时间，以在大约 2.6 秒内完成第一帧环视显示。

备注

本文档以 SDK 08.06.00.12 为基础，并在 J721E EVM 上进行了测试。

内容

商标

1 简介

s 环视应用会显示四图像传感器 3D 环视图像。这四个摄像机采集由片上 ISP 处理的原始图像。将四张图像提供给 GPU 以执行 3D 环视渲染。此图像将提供给 DSS 以在屏幕上显示。

环视应用包含两个单独的图表。第一个图表仅用于生成包含 SRV 碗映射的 GPU LUT。此图表仅在给定应用中执行一次；但是，如果 GPU LUT 需要根据场景进行更改，则可以运行多次。第二个图表包含完整的 GPU SRV，使用采集节点采集，将原始输出提供给 VISS 和 AEWB 节点以执行图像处理。然后将 VISS 的输出提供给 GPU 以执行 SRV 图像的渲染。最后，将此输出图像提供给在屏幕上显示的显示节点。图 1-1 展示了 SRV 应用的详细方框图。

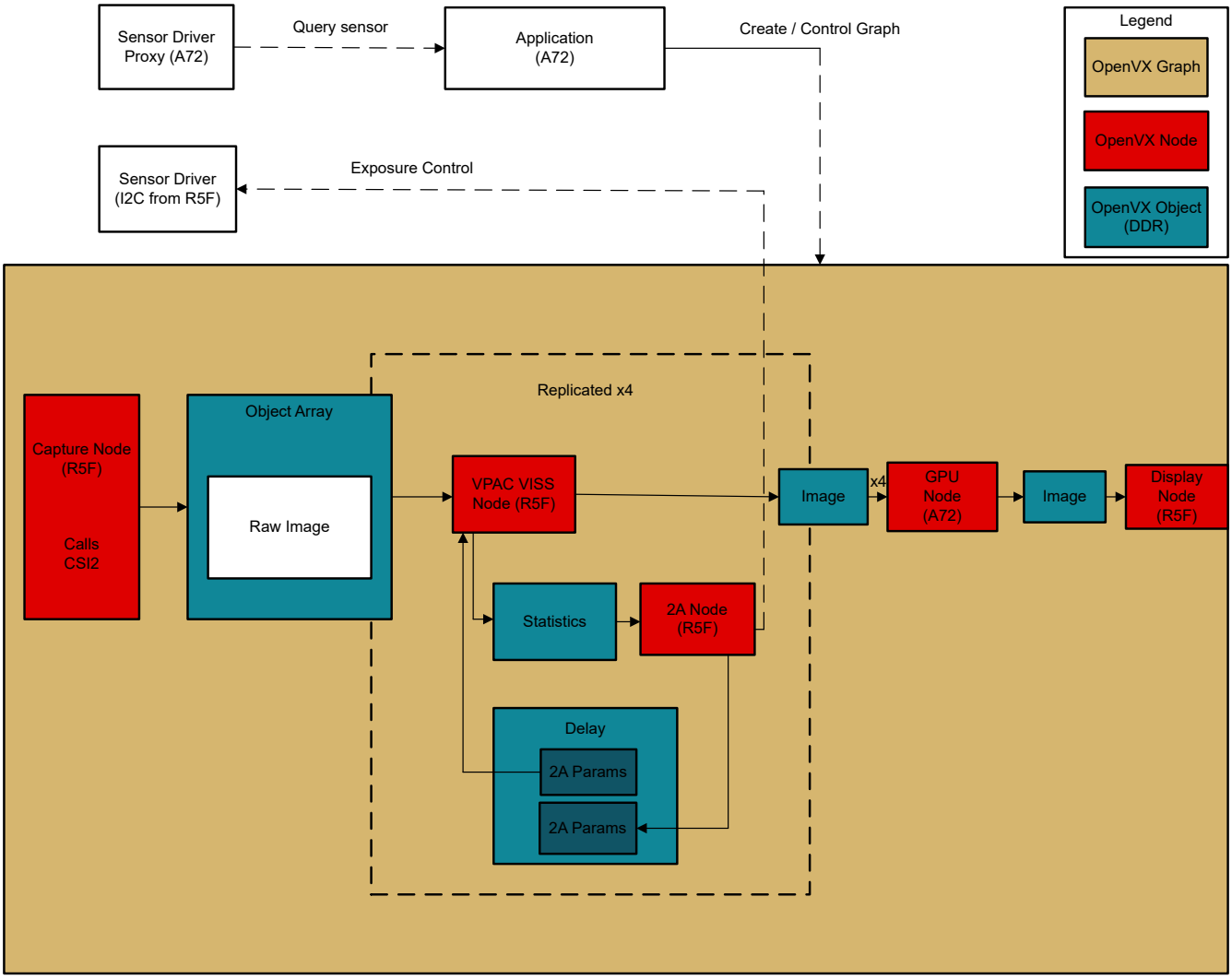

图 1-1. 环视方框图

在当前流程中，使用 MMCSD 作为引导介质运行此 SRV 演示所需的时间约为 20 秒。详细计时示意图如 图 1-2 所示

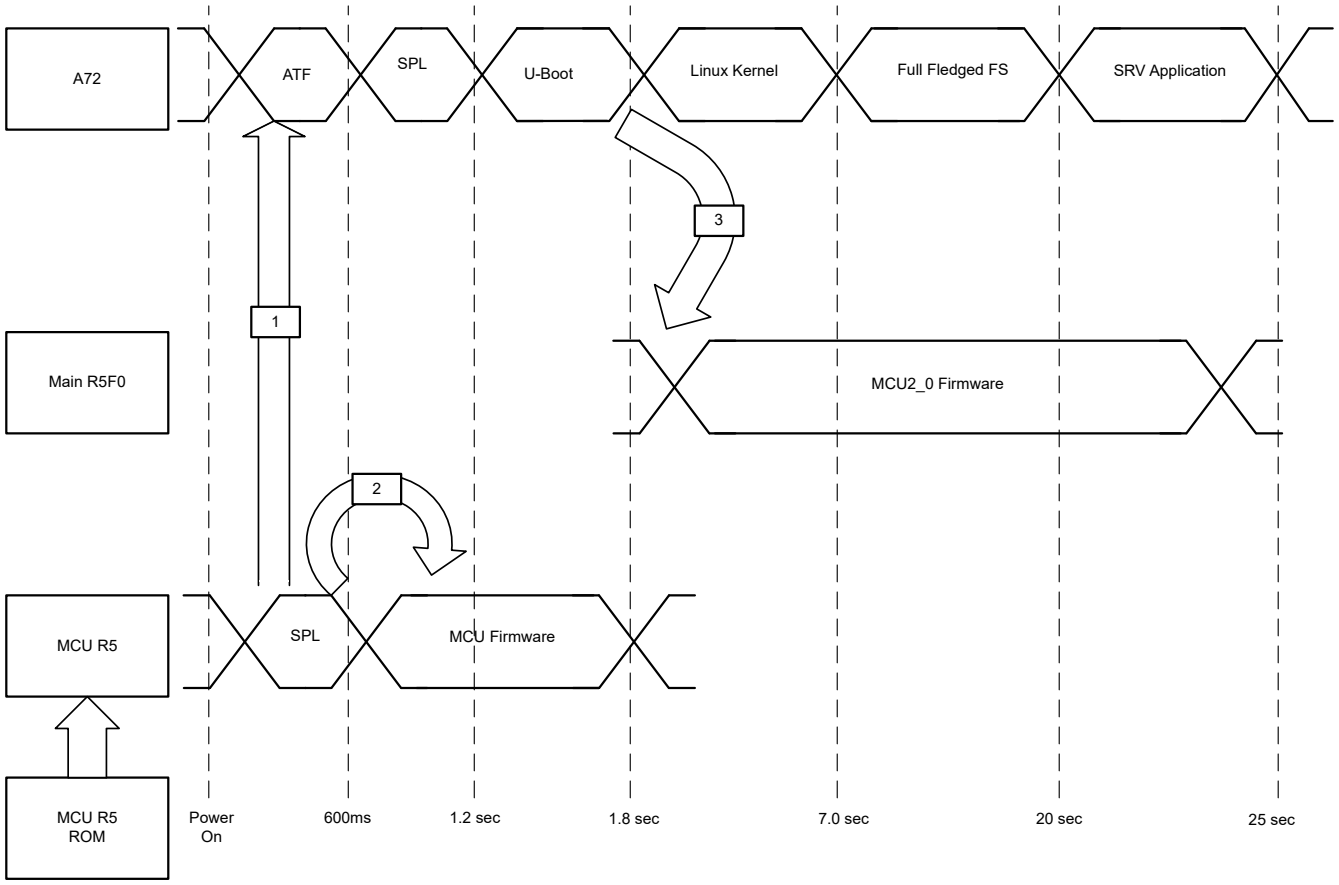

图 1-2. SRV 应用的默认计时示意图

此应用手册介绍了引导每个阶段的有条不紊的优化：

- 引导加载程序从 OSPI 引导介质切换到 SBL。
- Linux 器件树优化
- 文件系统切换到超小型 rootfs
- Vision_apps 框架优化
- 成像（传感器驱动程序）框架优化
- Vision_apps 应用重新设计

后续几节将详细说明其中的每个阶段。

如此一来，最终会在开机后的大约三秒内实现环视演示。图 1-3 展示了详细的计时示意图。

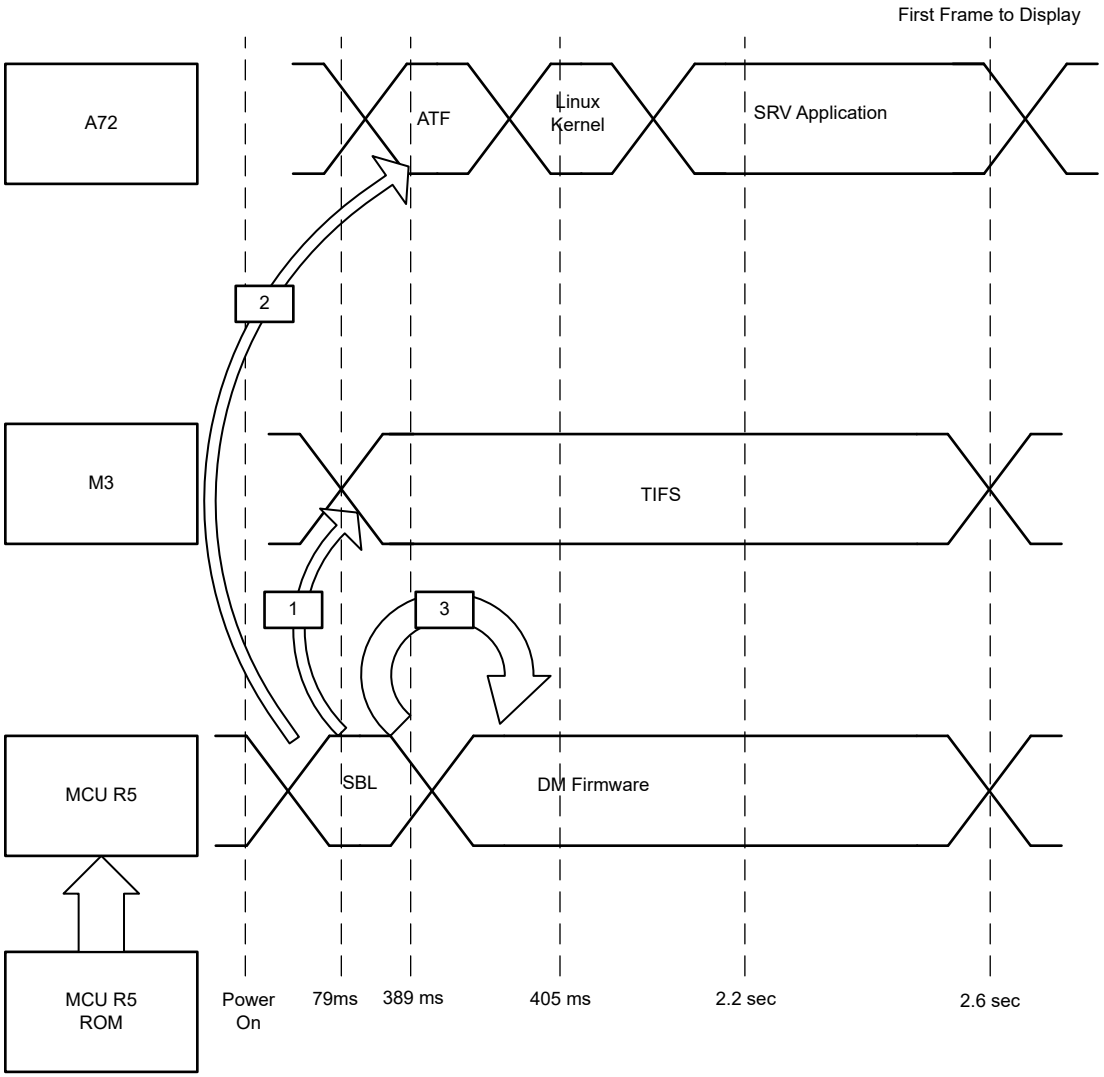

图 1-3. SRV 应用优化计时示意图

2 优化方法

2.1 引导加载程序从 OSPI 引导介质切换到 SBL

在基于 SPL 的引导流中，MCU2_0 固件加载仅在 U-Boot 阶段发生。这意味着加载固件可能需要大约 1.8 秒。

另一方面，使用基于 SBL 的引导流，支持从 SBL 加载固件。这意味着固件加载最早开始于 100ms。

选择 OSPI 作为引导介质可进一步加快针对 MMCSD 引导介质的引导速度。

使用上述路径和选择的优化后，节省了约 1.5 秒。

2.2 Linux 器件树优化

Linux 优化中涉及的主要更改如下所述：

- 将 loglevel=0 参数添加到 bootargs，以阻止所有内核打印
- 禁用 SRV 不需要的所有设备树节点
- C6x 和 C7x 不用于 SRV，因此禁用对它们的探测
- SRV 不需要的 USB、PCIe、UART、SPI、串行器等。
- 将文件系统从 MMC-SD 切换到 eMMC
- MMC-SD 探测额外增加了一秒，因此在文件系统切换到 eMMC 后，可以在最后一步禁用此功能

上述优化将到达文件系统的时间缩短了六至七秒。

2.3 文件系统切换到超小型 rootfs

从完备的 arago 文件系统切换到超小型 rootfs。超小型 rootfs 是一个裸最小文件系统，不包含多个使 vision_apps 框架和图形模块正常工作所必需的库。这可以避免所有不必要的文件系统服务初始化，从而优化文件系统加载时间。

发布此内容，将/lib 和/usr/lib 内容从完备的 FS 复制到超小型 rootfs 中的相应文件夹，然后将上述准备好的 SD 卡 rootfs 复制到 eMMC。

修改执行以下操作的 Init 脚本：

- 劫持 init 脚本而不是一直引导，以便为 SRV 执行最少的任务
- 安装所有必要的文件系统文件夹。
- 导出相关路径。
- 插入 virtio_rpmsg_bus、ti_k3_r5_remoteproc（R5F 驱动程序）、pvrsrvkm（图形驱动程序）模块。
- 运行 SRV 应用

通过上述优化，到达 linux 提示符的时间又缩短了九至十秒。

2.4 Vision_apps 框架优化

SRV 应用是基于 vision_apps 的演示，因此对 vision_apps 框架进行的优化很少，可以加快演示的执行速度。

优化如下：

- 从构建和 IPC 中移除 C6x 和 C7x 内核。
- 用于选择摄像机的硬编码参数。
- 重新构建 MCU2_0 应用，以便在其他任务之前执行所有 SRV 和附属任务。
- 移除 SRV 对 C6x 和 C7x 的附属关系。
- 拆分 ISS 传感器初始化，以便摄像机传感器初始化独立于 vison_apps IPC 启动。

2.5 成像框架优化

成像框架涉及配置四个传感器和 UB953 - UB960 串行器。因此，对成像框架的优化很少，可保持传感器和 SER-DES 的配置。

优化如下：

- 设置 I2C 广播标志，以便通过 I2C 并行初始化所有四个摄像机。
- 优化摄像机传感器寄存器编程的延迟。
- 优化 UB960 和 UB953 初始化延迟中的延迟。
- 对 ISSSensor_init 函数进行参数化，以支持独立于 vision_apps IP 的摄像机初始化。

上述优化将摄像机初始化时间从三秒缩短到 600-700 毫秒。

2.6 Vision_apps SRV 应用重新设计

应用本身经过重新设计以加快执行速度，如下所述：

- 将摄像机初始化代码移至 MCU2_0 固件。
- 上述设计更改确保摄像机初始化发生在 MCU2_0 开始运行后，而不是 SRV 应用启动时。

这样可确保当 A72 引导 Linux 并加载文件系统时，摄像机初始化所需的 600 毫秒能并行进行。

最终在开机后的大约三秒内实现环视演示。

3 详细设计过程

在继续进行补丁集成之前，请按照 Vision-apps 用户指南中的指令为 J721E EVM 设置和构建 Linux 和 RTOS SDK。

3.1 Linux 集成 (PSDKLA)

下载附件 tar 包到：$PSDK_Linux/board-support/linux-5.10.162+gitAUTOINC+76b3e88d56-g76b3e88d56.

0001-vision_apps-Integrated.patch

0002-linux-optimization.patch

```
cd $PSDKLA_PATH/board-support/linux-5.10.162+gitAUTOINC+76b3e88d56-g76b3e88d56
git apply 0001-vision_apps-Integrated.patch
git apply 0002-linux-optimization.patch
cd ..
make linux
```

3.2 成像集成 (PSDKRA)

```
cd $PSDKRA_PATH/imaging
git init
git add .
git commit -m "Initial commit"
```

下载附件 tar 包到：$PSDKRA_PATH/imaging

8_6_srv_Imaging.patch

```
git apply 8_6_srv_Imaging.patch
cd ../vision_apps
make imaging
```

3.3 Vision_apps 集成 (PSDKRA)

```
cd $PSDKRA_PATH/vision_apps
git init
git add .
git commit -m "Initial commit"
```

下载附件补丁到：$PSDKRA_PATH/vision_apps:

8_6_SRV_vision_apps.patch

```
git apply 8_6_SRV_vision_apps.patch
make sdk
```

3.4 PDK 实现 (PSDKRA)

```
cd $PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31
git init
git add .
git commit -m "Initial commit"
```

3.4.1 针对 OSPI 引导模式构建 R5 SBL

```
cd $PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/build
make ipc_echo_testb_freertos
make sbl_ospi_img_hlos
```

此处生成图像：

$PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/binary/j721e_evm/ospi/bin/sbl_ospi_img_hlos_mcu1_0_release.tiimage

3.4.2 构建 combined_appImage

1. 替换 $PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage/config.mk 中的以下文件“config.mk”

   config.mk
2. 打开上述文件并编辑路径：
   a. *HLOS_BIN_PATH*
   b. *DTB_IMG*
   c. *VISION_APPS_FW_PATH*
   d. *KERNEL_IMG*
3. 使用以下命令构建组合的 appImage：

   ```
   $PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage
   make clean
   make BOARD=j721e_evm HLOS_BOOT=optimized
   ```
4. 二进制文件生成路径为：$PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage/bin/j721e_evm

3.4.3 复制 TIFS 和 phy 调优参数

1. 从 $PSDKRA_PATH/pdk_jacinto_08_06_00_31/packages/ti/drv/sciclient/soc/V1/tifs.bin 选择预构建的 tifs.bin。
2. 选择附件 nor_spi_patterns.bin：
   a. nor_spi_patterns.bin

3.5 将二进制文件刷写到 OSPI

如 *TDA4 刷写技术* 中所述，有多种方法可以将二进制文件刷写到 OSPI 闪存。本应用手册使用 u-boot 将图像复制到 OSPI 闪存上。

1. 将四个图像复制到 SD 卡引导分区：
   a. combined_opt.appimage
   b. tifs.bin
   c. sbl_ospi_img_hlos_mcu1_0_release.tiimage
   d. Nor_spi_pattern.bin
2. 使用以下命令刷写到 OSPI：

```
sf probe
sf erase 0x0 0x4000000
fatload mmc 1 ${loadaddr} sbl_ospi_img_hlos_mcu1_0_release.tiimage;
sf update $loadaddr 0x0 $filesize;
fatload mmc 1 ${loadaddr} combined_opt.appimage;
sf update $loadaddr 0x100000 $filesize;
fatload mmc 1 ${loadaddr} tifs.bin;
sf update $loadaddr 0x80000 $filesize;
fatload mmc 1 ${loadaddr} nor_spi_patterns.bin;
sf update $loadaddr 0x3fe0000 $filesize;
```

3. 切换到 OSPI 引导模式。所有引导二进制文件现在都位于 OSPI 中。

3.6 在 SD 卡上使用超小型 rootfs 安装 vision_apps 的步骤

1. 插入 SD 卡并检查/dev/sdb 是否存在（已格式化的卡）。如果存在，请执行以下步骤：

```
cp $PSDK_LINUX_PATH/filesystem/tisdk-tiny-image-j7-evm.tar.xz $PSDK_RTOS_PATH
umount /dev/sdb1
umount /dev/sdb2
cd ${PSDK_RTOS_PATH}
sudo psdk_rtos/scripts/mk-linux-card.sh /dev/sdb
```

2. 将以下脚本下载到 $PSDKRA_PATH/psdk_rtos/scripts/ 文件夹并授予可执行权限：
   a. install_to_sd_card_tiny.sh

```
cd $PSDKRA_PATH
chmod +x psdk_rtos/scripts/install_to_sd_card_tiny.sh
cp $PSDKLA/filesystem/tisdk-tiny-image-j7-evm.tar.xz .
./psdk_rtos/scripts/install_to_sd_card_tiny.sh
```

3.7 将测试数据复制到 SD 卡（仅限一次）

```
cd /media/$USER/rootfs/
cp -r $PSDK_LINUX_PATH/targetNFS/lib/* /media/$USER/rootfs/lib
cp -r $PSDK_LINUX_PATH/targetNFS/usr/lib/* /media/$USER/rootfs/usr/lib/
cp -r $PSDK_LINUX_PATH/targetNFS/etc/*  /media/$USER/rootfs/etc/

mkdir -p opt/vision_apps
cd opt/vision_apps
tar --strip-components=1 -xf ${path/to/file}/psdk_rtos_ti_data_set_xx_xx_xx.tar.gz
sync

cd ${PSDK_RTOS_PATH}/vision_apps
make linux_fs_install_sd
```

3.8 init 脚本

1. 为避免在文件系统挂载上浪费时间，请使用 init 脚本绕过：
   a. 将 init 脚本文件 init.sh 复制到 SD 卡的 /media/${USER}/rootfs/home/root 目录中
   b. 将最小文件系统刷写到卡后，首次启动。以 root 身份登录。

      按照以下命令在 evm 上 rrom Linux 命令提示符：

```
chmod +x init.sh
cd /sbin/
rm init
ln -s /home/root/init.sh init
```

3.9 将文件系统从 SD 移动到 eMMC

在 TDA4-EVM 上引导至 Linux 命令提示符：

1. 格式化 emmc 并执行以下命令，将文件系统从 MMC-SD 复制到 eMMC。

```
mkdir /mnt/emmc
mkdir /mnt/sd
mount /dev/mmcblk0p2 /mnt/emmc
mount /dev/mmcblk1p2 /mnt/sd
cp -r /mnt/sd/* /mnt/emmc
sync
```

2. **完成上述步骤后，您可以通过应用下面的补丁，从 DT 优化 MMC-SD aka sdhci1 节点。**

   0003-Linux-eMMC-filesystem.patch
3. 将 SD 卡插入 ubuntu 宿主机。构建 dtb 并使用最新的 DTB 更新 combined_appImage。

```
cd $PSDK_LINUX_PATH/board-support/linux-5.10.162+gitAUTOINC+76b3e88d56-g76b3e88d56
git apply 0003-Linux-eMMC-filesystem.patch
make linux-dtbs
cd $PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage
make clean
make BOARD=j721e_evm HLOS_BOOT=optimized
cp $PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage/bin/
j721e_evm/combined_opt.appimage /media/$USER/BOOT
sync
```

4. 现在，将 SD 卡插入 TDA4-EVM 并将新的 combined.appimage 刻录到 OSPI：

```
sf probe
fatload mmc 1 ${loadaddr} combined_opt.appimage;
sf update $loadaddr 0x100000 $filesize;
```

5. 给电路板断电。
6. 切换到 OSPI。
7. Dip 开关设置：
   a. **SW8：00000000**
   b. **SW9：01000000**
8. 启动电路板。
9. **SRV 演示可以在大约 2.6 秒内启动。**

4 日志

```
MCU UART Logs
===============

[2025-02-17 18:04:19.893] SBL Revision: 01.00.10.01 (Feb 14 2025 - 17:25:21)
[2025-02-17 18:04:19.973] TIFS  ver: 8.6.3--v08.06.03 (Chill Capybar

MAIN UART Logs
================

[2025-02-17 18:04:20.282] NOTICE:  BL31: Built : 11:48:17, Oct 29 2024
[2025-02-17 18:04:20.282] ERROR:   GTC_CNTFID0 is 0! Assuming 200000000 Hz. Fix Bootloader
[2025-02-17 18:04:20.298] [    0.000000] Booting Linux on physical CPU 0x0000000000 [0x411fd080]
[2025-02-17 18:04:20.299] [    0.000000] Linux version 5.10.162-g76b3e88d56 (oe-user@oe-host)
(aarch3
[2025-02-17 18:04:20.331] [    0.000000] Machine model: Texas Instruments K3 J721E SoC
[2025-02-17 18:04:20.346] [    0.000000] earlycon: ns16550a0 at MMIO32 0x0000000002800000 (options
')
[2025-02-17 18:04:20.346] [    0.000000] printk: bootconsole [ns16550a0] enabled
[2025-02-17 18:04:20.474] ERROR:   GTC_CNTFID0 is 0! Assuming 200000000 Hz. Fix Bootloader
[2025-02-17 18:04:22.308] APP: Init ... !!!
[2025-02-17 18:04:22.308] MEM: Init ... !!!
[2025-02-17 18:04:22.308] MEM: Initialized DMA HEAP (fd=4) !!!
[2025-02-17 18:04:22.308] MEM: Init ... Done !!!
[2025-02-17 18:04:22.308] IPC: Init ... !!!
[2025-02-17 18:04:22.324] IPC: Init ... Done !!!
[2025-02-17 18:04:22.324] REMOTE_SERVICE: Init ... !!!
[2025-02-17 18:04:22.340] REMOTE_SERVICE: Init ... Done !!!
[2025-02-17 18:04:22.340]      0.000000 s: GTC Frequency = 0 MHz
[2025-02-17 18:04:22.340] APP: Init ... Done !!!
[2025-02-17 18:04:22.340]      0.000000 s:  VX_ZONE_INIT:Enabled
[2025-02-17 18:04:22.340]      0.000000 s:  VX_ZONE_ERROR:Enabled
[2025-02-17 18:04:22.340]      0.000000 s:  VX_ZONE_WARNING:Enabled
[2025-02-17 18:04:22.356]      0.000000 s:  VX_ZONE_INIT:[tivxInitLocal:130] Initialization Done !!!
[2025-02-17 18:04:22.356]      0.000000 s:  VX_ZONE_INIT:[tivxHostInitLocal:93] Initialization
Done !
[2025-02-17 18:04:22.372] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
[2025-02-17 18:04:22.372] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
[2025-02-17 18:04:22.388]      0.000000 s: ISS: Enumerating sensors ... !!!
[2025-02-17 18:04:22.388]      0.000000 s: ISS: Enumerating sensors ... found 0 : IMX390-UB953_D3
[2025-02-17 18:04:22.404]      0.000000 s: ISS: Enumerating sensors ... found 1 : AR0233-UB953_MARS
[2025-02-17 18:04:22.404]      0.000000 s: ISS: Enumerating sensors ... found 2 : AR0820-UB953_LI
[2025-02-17 18:04:22.404]      0.000000 s: ISS: Enumerating sensors ... found 3 :
UB9xxx_RAW12_TESTPN
[2025-02-17 18:04:22.420]      0.000000 s: ISS: Enumerating sensors ... found 4 :
UB96x_UYVY_TESTPATN
[2025-02-17 18:04:22.420]      0.000000 s: ISS: Enumerating sensors ... found 5 : GW_AR0233_UYVY
[2025-02-17 18:04:22.436] Sensor selected : IMX390-UB953_D3
[2025-02-17 18:04:22.436]      0.000000 s: ISS: Querying sensor [IMX390-UB953_D3] ... !!!
[2025-02-17 18:04:22.436]      0.000000 s: ISS: Querying sensor [IMX390-UB953_D3] ... Done !!!
[2025-02-17 18:04:22.436] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
[2025-02-17 18:04:22.452] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
[2025-02-17 18:04:22.452] Reading calmat file
[2025-02-17 18:04:22.452] file read completed
[2025-02-17 18:04:22.516] EGL: version 1.5
[2025-02-17 18:04:22.532] EGL: GL Version = (null)
[2025-02-17 18:04:22.548] EGL: GL Vendor = (null)
[2025-02-17 18:04:22.548] EGL: GL Renderer = (null)
[2025-02-17 18:04:22.548] EGL: GL Extensions = (null)
[2025-02-17 18:04:22.660] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
[2025-02-17 18:04:22.660] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
[2025-02-17 18:04:22.660]
[2025-02-17 18:04:22.661]
[2025-02-17 18:04:22.676]  ==========================
[2025-02-17 18:04:22.676]  Demo : Integrated SRV
[2025-02-17 18:04:22.676]  ==========================
[2025-02-17 18:04:22.676]
[2025-02-17 18:04:22.676]  p: Print performance statistics
[2025-02-17 18:04:22.676]
[2025-02-17 18:04:22.676]  e: Export performance statistics
[2025-02-17 18:04:22.676]
[2025-02-17 18:04:22.676]  x: Exit
[2025-02-17 18:04:22.676]
[2025-02-17 18:04:22.676]  Enter Choice: REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU
D
[2025-02-17 18:04:22.692] REMOTE_SERVICE: ERROR: CPU 4 is not enabled or invalid CPU ID
```

```
[2025-02-17 18:04:22.692]      0.000000 s: ISS: Starting sensor [IMX390-UB953_D3] ... !!!
[2025-02-17 18:04:22.708]      0.000000 s: ISS: Starting sensor [IMX390-UB953_D3] ... !!!
```

5 总结

根据建议的更改和本应用手册中遵循的过程，SRV 应用显示的第一帧所需时间约为两秒。

6 参考资料

- Vision Apps 用户指南
- 德州仪器 (TI)：*TDA4 刷写技术*

重要通知和免责声明

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 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。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