Application Note

Surround View System ECU

Nikhil Dasan, J Keerthy, Brijesh Jadav

ABSTRACT

Surround view application involves multiple cores running in parallel. MCU R5F starts with booting the device, A72 triggering the vision apps application, R5F starting the camera and display. Using the default Linux SDK with vision_apps, the time taken to achieve surround view is of the order of 25 sec as depicted in Figure 1-1. This application note focuses on optimizing the time to get out the first frame of surround view to display in approximately 2.6 seconds.

Note

This document is based on the SDK 08.06.00.12 and is tested on the J721E EVM.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

Copyright © 2025 Texas Instruments Incorporated

1 Introduction

The Surround View application shows a 4-image sensor 3D Surround View image. The four cameras capture a raw image that is processed by the on-chip ISP. The four images are given to the GPU to do the 3D Surround View rendering. This image is given to the DSS to display on a screen.

The Surround View app contains two separate graphs. The first graph is simply used for generating the GPU LUT that contains the SRV bowl mapping. This graph is only executed once with the given application; however, if the GPU LUT needs to change depending on the scene, this can be run multiple times. The second graph contains the full GPU SRV, capturing using the capture node, giving the raw output to the VISS and AEWB nodes to perform the image processing. The output of the VISS is then given to the GPU to perform the rendering of the SRV image. Finally, this output image is given to the display node to be shown on the screen. Figure 1-1 shows the detailed block diagram for the SRV application.

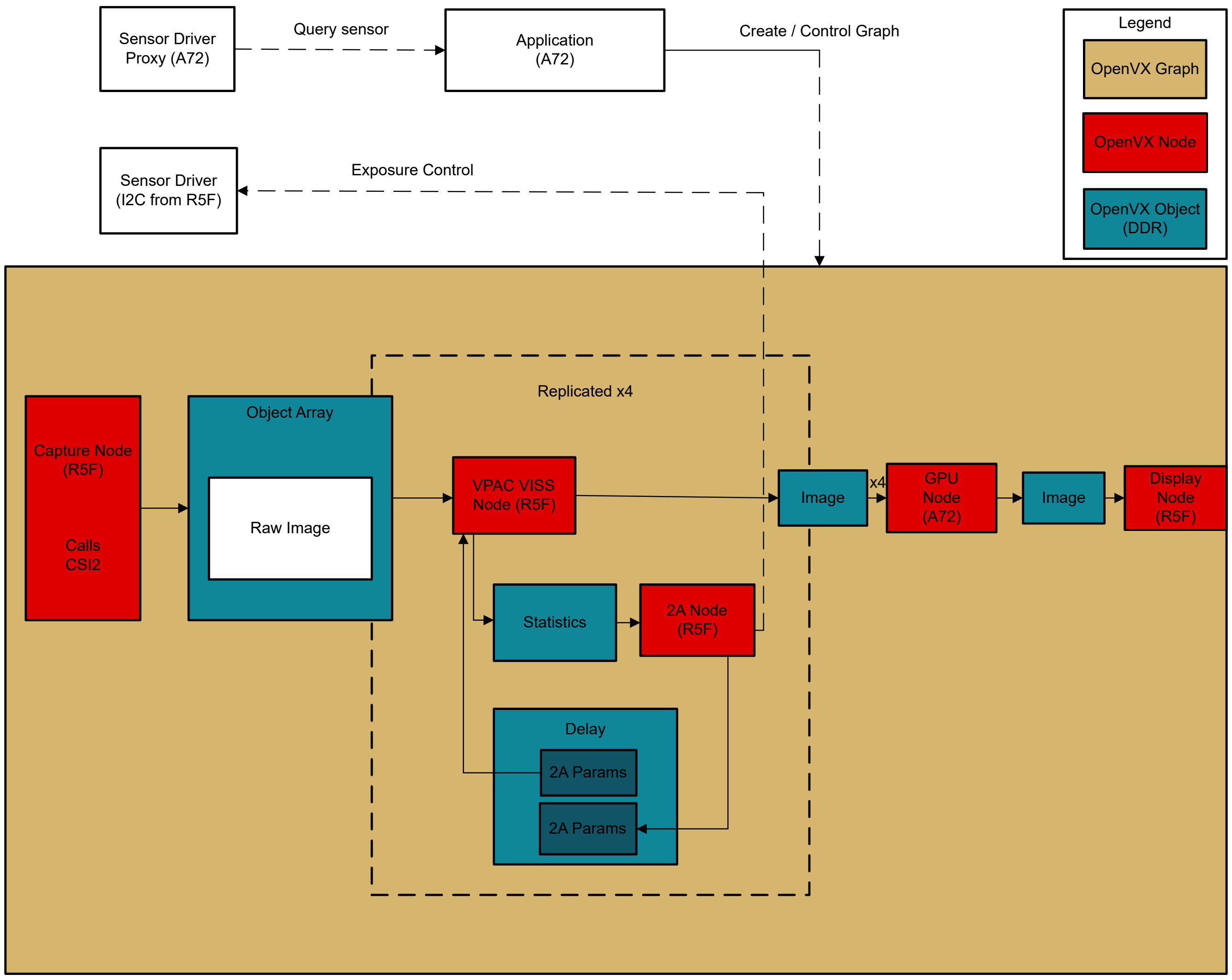

Figure 1-1. Surround View Block Diagram

Copyright © 2025 Texas Instruments Incorporated

In the current flow, the time taken to run this SRV demo using MMCSD as boot media is around 20 sec. The detailed timing diagram is as shown in Figure 1-2

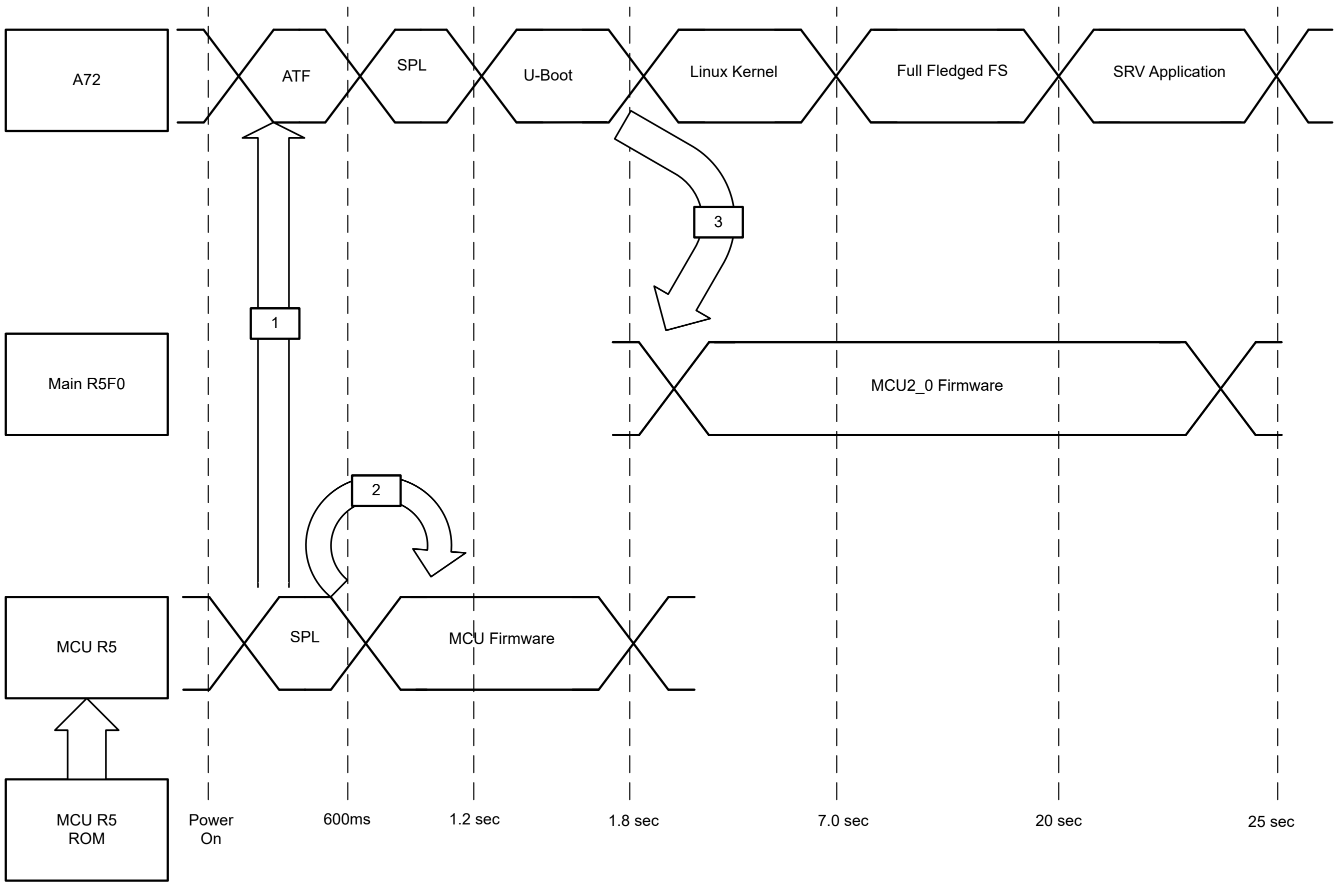

Figure 1-2. Default Timing Diagram of SRV Application

This application note addresses the optimizations methodically at every stage of boot:

- Bootloader switch to SBL from OSPI boot media.
- Linux device tree optimizations
- File system switch to tiny rootfs
- Vision_apps Framework optimizations
- Imaging (Sensor driver) Framework optimizations
- Vision_apps Application re-design

Each of these stages are elaborated in the coming sections.

Copyright © 2025 Texas Instruments Incorporated

Thus, eventually leading to running a surround view demo in approximately 3 seconds from power ON. The detailed timing diagram is as shown in Figure 1-3.

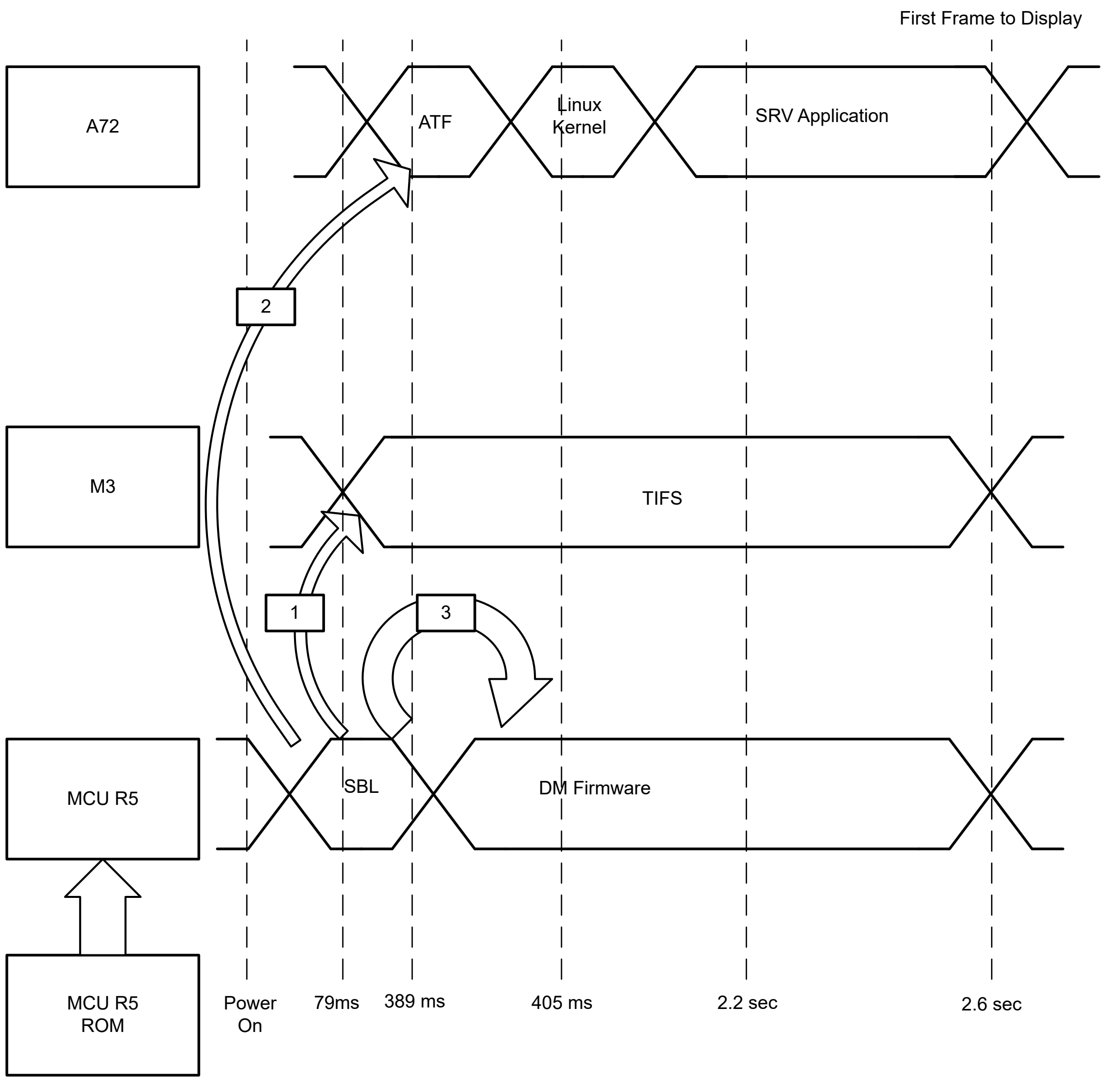

Figure 1-3. Optimized Timing Diagram of SRV Application

Copyright © 2025 Texas Instruments Incorporated

2 Optimization Methology

2.1 Bootloader Switch to SBL From OSPI Boot Media

In the case of SPL-based boot flow, the MCU2_0 firmware loading happens only at the U-Boot stage. This implies that it can take around 1.8 sec to load the firmware.

On the other hand, with the SBL-based boot flow, the firmware loading is supported from SBL. This implies that firmware loading starts as early as 100 ms.

Choosing the OSPI as the boot media further speeds up the booting against MMCSD boot media.

With the above path and chosen optimizations, it saved about 1.5 Seconds.

2.2 Linux Device Tree Optimizations

The major changes involved in Linux Optimizations are as mentioned below:

- Add loglevel=0 parameter to bootargs to suppress all the kernel prints
- Disable all the device tree nodes that are not needed for SRV
- C6x, C7x are not used for SRV so disable probing them
- USB, PCIe, UARTs, SPI, Serdes to name a few that are not needed for SRV.
- Switch the file system from MMC-SD to eMMC
- MMC-SD probing adds additional 1sec, so disable that as the last step once the file system is switched to eMMC

The above optimizations saved 6-7 Seconds to get to file system.

2.3 File System Switch to Tiny rootfs

Switch to tiny rootfs from full fledged arago file system. Tiny rootfs is a bare minimal file system and does not contain multiple libraries that are essential for the vision_apps framework and graphics modules to be functional. This optimizes file system loading time by avoiding all the non-essential file system services initialization.

Post this, copy /lib and /usr/lib contents from full fledged FS to the corresponding folders in tiny rootfs and then copy the above prepared SD card rootfs to eMMC.

Modify the Init script that does the below:

- Instead of booting all the way to prompt hijack the init script to do bare minimal tasks for SRV
- Mount all the necessary file system folders.
- Export the relevant paths.
- Insert the virtio_rpmsg_bus, ti_k3_r5_remoteproc (R5F driver), pvrsrvkm (Graphics driver) modules.
- Run the SRV application

The above optimizations saved another 9-10 Seconds to get to linux prompt.

2.4 Vision_apps Framework Optimizations

As the SRV application is a vision_apps-based demo, there are few optimizations that are done to the vision_apps framework that provides faster execution of the demo.

The optimizations are as follows:

- Removing C6x and C7x cores from build and from IPC.
- Hardcoding parameters for choosing the camera.
- Re-structure the MCU2_0 application to perform all the SRV and dependent tasks to be done before the others.
- Remove dependency of SRV on C6x and C7x.
- Split ISS sensor initialization so that camera sensor initialization is started independent of vison_apps IPC.

Copyright © 2025 Texas Instruments Incorporated

2.5 Imaging Framework Optimizations

The imaging framework is involved in configuring the four sensors and the UB953 - UB960 serdes. Hence, there are few optimizations that are done to the imaging framework, that maintains the configuration of the sensors and SER-DES.

The optimizations are as follows:

- Set I2C broadcast flag so that all four cameras are initialized in parallel over I2C.
- Optimize the delays for camera sensor register programming.
- Optimize delays in the UB960 and UB953 initialization delays.
- Parametrize the ISSSensor_init function so as to support camera initialization independent of vision_apps IP.

The above optimizations reduced camera initialization time from 3sec to 600-700 milli seconds.

2.6 Vision_apps SRV Application Redesign

The application itself is redesigned for a faster execution as mentioned below:

- Move out the camera initialization code to MCU2_0 firmware.
- The above design change makes sure that the camera initialization happens right after the MCU2_0 starts running and not when the SRV application starts.

This makes sure that the 600 milli seconds spent on camera initialization happens in parallel when A72 is booting Linux and loading file system.

Eventually leading to Surround view demo in approximately 3 seconds from power-on.

3 Detailed Design Procedure

Before moving on to the integration of patches, follow the instructions mentioned in Vision-apps User Guide to setup and build the Linux and RTOS SDK for J721E EVM.

3.1 Linux Integration (PSDKLA)

Download the attached tar ball to: $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 Imaging Integration(PSDKRA)

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

Download the attached tar ball to: $PSDKRA_PATH/imaging

8_6_srv_Imaging.patch

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

Copyright © 2025 Texas Instruments Incorporated

3.3 Vision_apps Integration (PSDKRA)

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

Download the attached patch to: $PSDKRA_PATH/vision_apps:

8_6_SRV_vision_apps.patch

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

3.4 PDK Implementation (PSDKRA)

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

3.4.1 Build R5 SBL for OSPI Boot Mode

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

Image is generated here:

`$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 Build combined_appImage

1. Replace the below file "config.mk" in `$PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage/config.mk`

   config.mk
2. Open the above file and edit the paths:
   a. *HLOS_BIN_PATH*
   b. *DTB_IMG*
   c. *VISION_APPS_FW_PATH*
   d. *KERNEL_IMG*
3. Build the combined appImage using the below commands:

   ```
   $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. The binary is generated under:`$PSDK_RTOS_PATH/pdk_jacinto_08_06_00_31/packages/ti/boot/sbl/tools/combined_appimage/bin/j721e_evm`

3.4.3 Copy TIFS and phy Tuning Parameters

1. Pick the tifs.bin prebuilt from $PSDKRA_PATH/pdk_jacinto_08_06_00_31/packages/ti/drv/sciclient/soc/V1/tifs.bin.
2. Pick the attached nor_spi_patterns.bin:
   a. nor_spi_patterns.bin

Copyright © 2025 Texas Instruments Incorporated

3.5 Flashing the Binaries to OSPI

There are multiple methods to flash the binaries to OSPI flash as mentioned in *TDA4 Flashing Techniques*. This application note uses the u-boot to copy the images onto the OSPI Flash.

1. Copy four images to SD card boot partition:
   a. combined_opt.appimage
   b. tifs.bin
   c. sbl_ospi_img_hlos_mcu1_0_release.tiimage
   d. Nor_spi_pattern.bin
2. Flash to OSPI using the below commands:

```
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. Switch to OSPI Boot mode. All the boot binaries are now in OSPI.

3.6 Steps to Install vision_apps With tiny-rootfs on SD Card

1. Insert the SD card and check whether /dev/sdb exists (already formatted card). If yes, do the following steps:

```
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. Download the below script to $PSDKRA_PATH/psdk_rtos/scripts/ folder and give executable permissions:
   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 Copy Test Data to SD card (one time only)

```
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
```

Copyright © 2025 Texas Instruments Incorporated

3.8 Init Script

1. To avoid losing time on filesystem mounting, bypass using an init script:
   a. Copy the init script file, init.sh, to the SD Card in /media/${USER}/rootfs/home/root directory
   b. After flashing the minimal filesystem to the card, boot up the first time. Login as root.

      Follow the below commands rrom the Linux command prompt on the evm:

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

3.9 Moving File System From SD to eMMC

Boot to Linux command prompt on the TDA4-EVM:

1. Format the emmc and execute the following commands to copy filesystem from MMC-SD to 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. **After the above step, you can optimize the MMC-SD aka sdhci1 node from DT by applying the patch below**.

   0003-Linux-eMMC-filesystem.patch
3. Insert the SD card in ubuntu host machine. Build the dtb and update the combined_appImage with the latest DTB.

   ```
   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. Now, insert the SD card to TDA4-EVM and burn the new combined.appimage to OSPI:

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

5. Power off the board.
6. Switch to OSPI.
7. Dip switch settings:
   a. **SW8: 00000000**
   b. **SW9: 01000000**
8. Boot the board.
9. **SRV demo can be up in approximately 2.6 seconds.**

Copyright © 2025 Texas Instruments Incorporated

4 Logs

```
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
```

Copyright © 2025 Texas Instruments Incorporated

```
[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 Summary

With the changes suggested and the procedure followed in this application note, the time to first frame to display for a SRV application is around 2 seconds.

6 References

- Vision Apps User Guide
- Texas Instruments: *TDA4 Flashing Techniques*

Copyright © 2025 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2025, Texas Instruments Incorporated