OPTIMIZING INFOADAS BOOT TIME

Texas Instruments

August 25, 2015

Contents

List of Tables

List of Figures

1 Introduction

This document describes the steps perfomed by TI to optimize the boot time of the InfoADAS release supporting 2D SRV from 4 OV1063x cameras. With the steps described in this document, we were able to obtain a boot time of

1. 7.3 s from power on of the EVM using a LCD display
2. 7.6 s from power on using a HDMI display.

for 2D SRV using eMMC boot.

The boot times quoted above include 1.9 s of sensor initialization time.

The techniques described in this document to optimize the boot time such as

1. Using single stage boot loader

2. Application optimization
3. Kernel customization
4. Using initramfs

can be built upon by customers to optimize the boot time of their production system.

1.1 Document organization

First we set the baseline for the measurements including software and hardware and then describe the measurement mechansim. This will help anyone with the necessary hardware and software to reproduce the results.

Then we describe how to run the prebuilt binaries from SD card and then eMMC. This will allow us to establish the boot time claims made in the introduction.

Next we provide a brief overview of a Linux usecase optimization process and start optimizing the 2D SRV usecase. We will start with a single stage boot loader setup and basic application optimizations and then move to kernel optimizations and using initramfs.

At the end, we will describe further optimization steps that can be performed.

2 Measurement mechanism

2.1 Hardware/Software baseline

GLSDK release 7.01.00.03 and InfoADAS release 1.01.00.05 were used as the software baseline for the measurements in this document. The hardware baseline is the same as described in the **INFOADAS Getting Started Guide**.

We specifically used a Rev E1 EVM. All the cores were running at their nominal frequencies.

Core	Frequency (MHz)
A15	1000
IPU1	425
DSP2	600
DSP1	600

2.2 Interpreting the console logs

In this document, we use logs from the serial port output of the EVM to perform boot time measurements. The logs are collected and timestamped using the below command. We are assuming that the serial port from the EVM is identified on the host as `/dev/ttyUSB0`. Please modify this appropriately for your setup. Instructions on setting up grabserial are available at http://elinux.org/Grabserial.

```
host $ grabserial  -d /dev/ttyUSB0 -t -m "U-Boot SPL 2014.07" -e 20
```

A sample log is shown below.

```
[1.902233 1.902233] U-Boot SPL 2014.07-00003-ge488099 (Aug 21 2015 - 19:23:46)
[0.004875 0.004875] DRA752-GP ES1.0
[0.048773 0.043898] spl_mmc_load_image
[0.145771 0.096998] reading dra7-evm.dtb
```

```
[0.168737 0.022966] reading uImage
[0.173662 0.004925] reading uImage
```

The first column in each line of the log shows the time at which the first character of this line has been received. The time shown is relative to the line selected as time base. We are using the "U-Boot SPL 2014.07…" console print as the starting point for the timing the rest of the prints. This print occurs ~ 20 ms from start of MLO execution.

```
[1.902233 1.902233] U-Boot SPL 2014.07-00003-ge488099 (Aug 21 2015 - 19:23:46)
[0.004875 0.004875] DRA752-GP ES1.0
```

The second column of the line shows the number of seconds from receiving the first character of the previous line to receiving the first character of the current line. This shows the execution time of the code from the first console print to the second console print. From the log above, SPL took 0.004875 s to execute the first statement.

3 Running the prebuilt binaries

Follow the instructions in the **INFOADAS Getting Started Guide** to setup the hardware. Ensure that you are able to build and run the 2D SRV usecase from the base infoadas package.

The prebuilt binaries are included in the folder `prebuilt/boot_optimized` in this package. A description of the binaries is below.

Table 2: Description of prebuilt binaries

File.name	Purpose
MLO	First Stage Boot loader
uboot.img	Second stage boot loader. This is not necessary during optimized boot as we will be using single stage boot.
uImage	Kernel image for use by single stage boot
dra7-evm.dtb	DTB file used by single stage boot. This file is generated from dra7-evm-infoadas.dtb by adding bootargs and initramfs locations. The file is renamed to dra7-evm.dtb as this is the name expected by single stage boot
initramfs.cpio.gz	inital ram file system for use by single stage boot
initramfs.uImage	inital ram file system for use by second stage boot loader

We suggest running the binaries from an SD card first due to the ease of copying binaries over.

3.1 From SD card

Copy `MLO`, `uImage`, `dra7-evm.dtb` and `initramfs.cpio.gz` to the SD card and reboot the EVM. We recommend **backing up** the existing binaries in the boot parition of the SD card as these binaries are optimized for boot timing. The MLO has been modified to start in single stage boot. It will load the kernel, device tree and initramfs from the SD card to the appropriate locations and jump to the kernel. The file system is setup to

1. Automatically start weston to using connector 4
2. Load the IPU1, DSP2 and DSP1 with the appropriate binaries
3. Configure the sensors
4. Start infoadas application.

The infoadas application has been modified to automatically start 2D SRV. The time to boot the 2D SRV usecase is measured by timing the console logs.

3.1.1 Console log from SD boot

The major components in the execution can be seen from below. The full console log is shown after the table.

Table 3: Boot time breakdown from SD card

Stage	Log.lines	Execution.time	Remarks
Boot loader	1-8	2.003 s	0.247 s is spent in loading uImage. 1.582 s is spent in loading initramfs. The load times are higher as we are loading from SD card.
Kernel loading	8-9	1.89 s	Kernel initialization, unpacking the initramfs
Weston	10-11	0.044 s	Insert omapdrm_pvr.ko, start pvrsrvinit and weston
Remotecore loading	11-15	0.135 s	Load the firmware to the remotecores
Sensor configuration	15-17	0.080 s	I2C writes to the four OV1063 cameras
2D SRV application	17 - 30	2.71 s	Configure the processing and display stitched image on screen

```
[1.902233 1.902233] U-Boot SPL 2014.07-00003-ge488099 (Aug 21 2015 - 19:23:46)
[0.004875 0.004875] DRA752-GP ES1.0
[0.048773 0.043898] spl_mmc_load_image
[0.145771 0.096998] reading dra7-evm.dtb
[0.168737 0.022966] reading uImage
[0.173662 0.004925] reading uImage
[0.420775 0.247113] reading initramfs.cpio.gz
[2.003734 1.582959] Jumping to Linux
[3.894720 1.890986] INIT
[3.922738 0.028018] Weston
[3.966746 0.044008] Remoteproc
[3.967929 0.001183] Loading dra7-dsp1-fw.xe66
[4.002743 0.034814] Loading dra7-dsp2-fw.xe66
[4.050764 0.048021] Loading dra7-ipu1-fw.xem4
[4.101743 0.050979] Sensor Step 1
[4.163740 0.061997] Sensor Step 2
[4.185708 0.021968] 2D SRV Start
[4.192641 0.006933] INFO: Found a '7inch LCD' display(inactive) with size 800x480
[4.198814 0.006173] INFO: Found a 'HDMI' display(active) with size 1920x1080
[4.203851 0.005037] INFO: Found a 'FPDLink' display(inactive) with size 1280x720
[4.898737 0.694886] INFO: Creating plugin surr_view
[4.901703 0.002966] INFO: Creating chain surr_view:lvds-srv-2d-880x1080
[6.860440 1.958737] INFO: Chain surr_view:lvds-srv-2d-880x1080 has 2 eplinks
[6.865745 0.005305] INFO: Starting chain surr_view:lvds-srv-2d-880x1080
[6.869814 0.004069] INFO: Using properties 500x320 @ 10,200 on HDMI for surface 'Left Preview'
[6.876790 0.006976] INFO: Using properties 500x320 @ 10,600 on HDMI for surface 'Right Preview'
[6.883705 0.006915] INFO: Using properties 500x320 @ 1410,200 on HDMI for surface 'Front Preview'
[6.890687 0.006982] INFO: Using properties 500x320 @ 1410,600 on HDMI for surface 'Rear Preview'
[6.896866 0.006179] INFO: Using properties 880x1080 @ 520,0 on HDMI for surface '2-D stitched surround
[6.904808 0.007942] INFO: VIVI thread for chain surr_view:lvds-srv-2d-880x1080 started
```

3.2 From eMMC

The instructions to setup a DRA7xx EVM to boot from eMMC are described in the section Using eMMC Boot in the DRA7xx GLSDK Software Developers Guide. Once the basic boot has been verified, copy the prebuilt binaries `MLO`, `uImage`, `dra7-evm.dtb` and `initramfs.cpio.gz` from `prebuilt/boot_optimized` to the eMMC boot partition. Reboot the EVM.

The console logs are similar to the logs obtained from SD boot except for a reduction in the time spent in the boot loader. As transfer rates from eMMC are faster compared to SD, the time taken to load `uImage` and `initramfs.cpio.gz` comes down significantly.

```
[2.273209 2.273209] U-Boot SPL 2014.07-00003-ge488099 (Aug 21 2015 - 18:13:07)
[0.004925 0.004925] DRA752-GP ES1.0
[0.048798 0.043873] spl_mmc_load_image
[0.100806 0.052008] reading dra7-evm.dtb
[0.109763 0.008957] reading uImage
[0.112771 0.003008] reading uImage
[0.273797 0.161026] reading initramfs.cpio.gz
[0.839795 0.565998] Jumping to Linux
```

4 Package Organization

This package is structured as an overlay on top of the InfoADAS delivery. Most of the content is placed in the folder `boot_scripts` with minor customizations in other files.

It includes

1. Kernel patches
    1. to infoadas device tree for reducing initialization time.
    2. to parallelize initramfs loading.
    3. to reduce the sensor initialization time.
2. Kernel configuration file that modifies the infoadas kernel configuration to remove/modularize functionality to reduce the boot time.
3. Infoadas application patch to start 2D SRV automatically without user input.
4. Script to ease creating an initramfs for InfoADAS.

4.1 Kernel Patches

The kernel patches are located in `boot_scripts/patches/kernel` in the package.

File.name	Functionality
0001-arm-dts-pad-DTB-size-by-4K.patch	Pad DTB by 4 KB. This increased size is helpful when manipulating the DTB in SPL.
0002-i2c-ov1063x-Cleanup-and-optimize-sensor-configuratio.patch	Make the sensor configuration required for 2D SRV the default. Reduce configuration time per sensor by calibrating delays
0003-arch-arm-dra7-vision-disable-camera-and-serializers-.patch	Remove unused camera and serializers from DTB. This reduces time spent in probing for unused peripherals.
0004-crypto-omap-aes-Fix-module-build.patch	This patch is required if we want to make crypto-aes into a module
0005-Populate-rootfs-asynchronously-and-early.patch M	ake populate_rootfs asynchronous.

File.name	Functionality
0006-init-turn-populate_rootfs-into-a-postcore_initcall.patch	Invoke populate_rootfs earlier in the kernel boot process.

4.2 U-Boot patches

The kernel patches are located in `boot_scripts/patches/u-boot` in the package.

Table 5: boot loader patches

File.name	Functionality
0001-spl-dra7xx-Enable-loading-initramfs-from-mmc.patch	Load initramfs to RAM if file is found on eMMC
0002-spl-dra7xx-default-to-single-stage-boot.patch	Load kernel and DTB directly from SPL.
0003-spl-add-print-to-capture-kernel-start-time.patch	Add a debug print in SPL to capture end of SPL execution

4.3 Helper scripts

The helper scripts are located in 'boot_scripts'. The description of each of the files is below.

Table 6: Description of helper scripts for creating initramfs

File	Purpose
mkifs.sh	Top level file to create the initramfs
gen_ifs_cfg.pl	Perl script to resolve dependencies of binaries included in initramfs
modules.txt	Text file with list of modules that need to be included in initramfs
files.txt	Text file with list of files that need to be included in initramfs
initramfs.tt	Template file used by gen_ifs_cfg.pl to generate its output
install_deps.sh	bash script to install dependencies required for the scripts in this folder.
init	script to be used as init program for initramfs
Launch-infoadas.sh	Script used from initramfs to launch infoadas.

4.4 Modified files from the base infoadas package

Table 7: Description of modifications to existing files from the base package

File	Modification
Makefile.am	Include the boot optimization flag in application compile options
build.sh	Remove execute permissions on bmp file
build_config.txt	Define a variable to indicate boot optimizations
mk-sd-update.sh	Copy uImage also to the target folder. Document updates
plat/linux/app/app_main.c	Print a message when first buffer is sent to display. Auto start and stop 2D SRVif boot optimizations are turned on
plat/linux/kernel/ker-addon.sh	Strip symbols from modules. Apply optimized kernel config if boot optimizations are turned on
plat/linux/kernel/scripts/camnodes.sh	Redirect output to /dev/null to minimize run time

5 Building the boot time optimized binaries

This package is structured as an overlay on top of the InfoADAS delivery. While describing the instalation steps, we will assume the infoadas release is installed under `~/infoadas` and this package has been saved as `~/infoadas-boot-opt.tar.gz`. As this package replaces some of the files in the base infoadas package, it is recommended to take a backup of the base infoadas package or place it under version control.

```
host $ cd ~/infoadas/
host $ tar xf ../infoadas-boot-opt.tar.gz
```

Apply the boot optimization patches on top of the kernel. It is assumed that the patches required by the infoadas package have already been applied. Please create a branch so that you can quickly revert to the base infoadas kernel when desired.

```
host $ cd ~/infoadas/
host $ source build_config.txt
host $ cd $LINUX_KERNELDIR
host $ git am ~/infoadas/boot_scripts/patches/kernel/*.patch
```

Apply the boot optimization patches on top of U-boot. Please create a branch so that you can quickly revert to the base infoadas u-boot when desired. Sourcing the file `build_config.txt` is required only once in the shell running these commands. We will omit this from the next command listing.

```
host $ cd ~/infoadas/
host $ source build_config.txt
host $ cd $LINUX_UBOOTDIR
host $ git am ~/infoadas/boot_scripts/patches/u-boot/*.patch
```

Run the infoadas build. This generates the updated kernel,u-boot and application binaries necessary for optimized boot time.

```
host $ cd ~/infoadas/
host $ ./build.sh --all install
```

Install the infoadas binaries. It is assumed that an SD card is plugged in and the root partition is mounted at `/media/rootfs` and the boot partition is mounted at `/media/boot`. These are the default locations expected by the infoadas package.

```
host $ cd ~/infoadas/
host $ sudo ./mk-sd-update.sh prebuilt
```

Install the dependencies for the helper script to create initramfs. We use a Perl script to handle the dependencies while building the initramfs. The below script installs the prerequisites for the Perl script. It also installs the package 'device-tree-compiler' which provides the command `fdtput`. We use this command to manipulate the device tree for initramfs.

```
host $ cd ~/infoadas/boot_scripts
host $ sudo ./install_deps.sh
```

Run the scripts to generate the initramfs. This step will build initramfs and copy all the required binaries to the boot partition.

```
host $ cd ~/infoadas/boot_scripts
host $ ./mkifs.sh
```

Now the SD card is updated with the binaries for optimized infoadas boot. The SD card can be used to verify the optimized boot as described in the earlier chapter.

5.1 Undoing the boot optimization changes

To undo the boot optimization changes.

1. Revert the kernel patches applied above.
2. Revert the u-boot patches applied above.
3. Set the variable `INFOADAS_BOOT_OPT` to 0 in file `build_config.txt`.

```
# Set this variable to enable boot optimizations
export INFOADAS_BOOT_OPT=0
```

6 Optimization steps

This section describes the steps that we went through in optimizing the 2D SRV boot time.

6.1 Overview

The boot process of a Linux application is usually broken into four components.

1. Boot loader
2. Kernel
3. System services initialization
4. Application

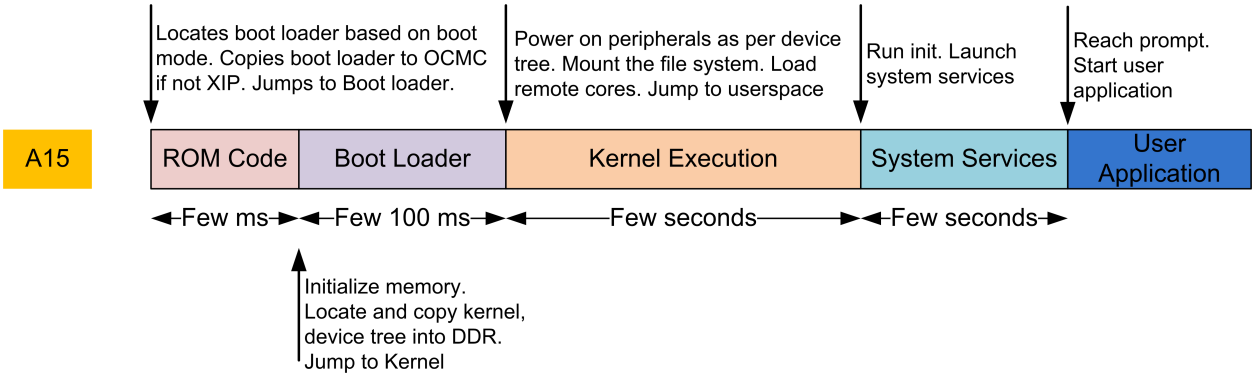

Figure 1: Linux normal boot flow

The optimization process also focusses on optimizing each of the four components of the boot process. In case of 2D SRV, we do not have dependency on (3) System services. So we will first launch the 2D SRV application and then launch the system services.

Most of the execution time in the boot loader is spent in copying the binaries into DDR or waiting for the DMA transfer of the binaries to be complete. As a result, there are very few avenues to optimize the boot loader except to increase the data transfer rates. We will tackle the boot loader optimizations in a separate application note.

In this document, we will focus on (2) Kernel and (4) Application i.e. 2D SRV.

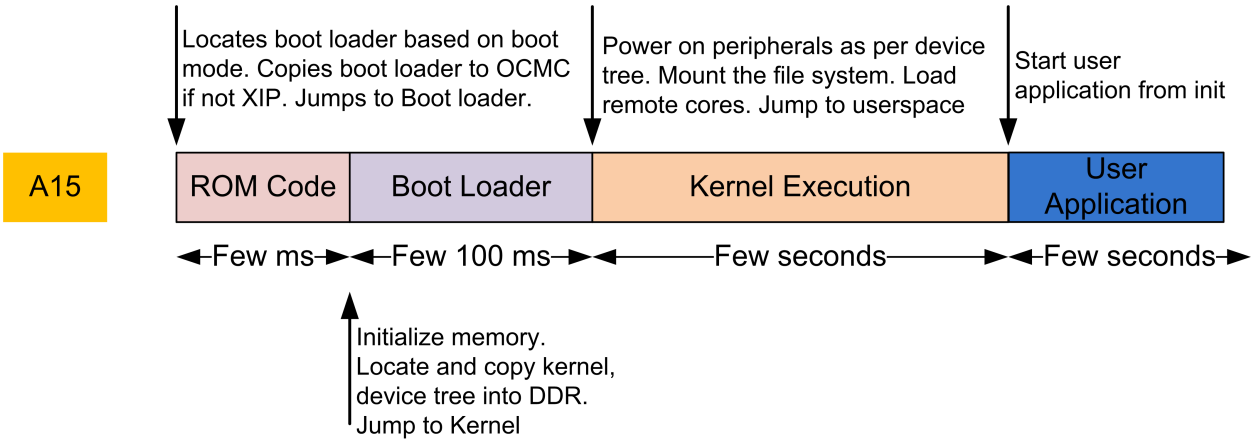

Figure 2: Linux boot without system services

6.2 Bootloader

In the scope of this document, we will keep the boot loader changes to minimum. "Boot loader" optimizations will be covered in a separate application note. For purposes of 2D SRV boot time optimization, we switch unconditionally to a single stage boot loader. This requires us to decide on the boot media before hand but saves time during the boot. We also added code to look for and load initramfs from MMC as it is expected that this feature is required. These patches can be found in `boot_scripts/patches/u-boot`.

Table 8: boot loader patches

File.name	Functionality
0001-spl-dra7xx-Enable-loading-initramfs-from-mmc.patch	Load initramfs to RAM if file is found on eMMC
0002-spl-dra7xx-default-to-single-stage-boot.patch	Load kernel and DTB directly from SPL.
0003-spl-add-print-to-capture-kernel-start-time.patch	Add a debug print in SPL to capture end of SPL execution

6.3 Initial benchmark

Before doing the initial benchmark for the optimization, we performed the following steps as basic optimization.

1. We modified the infoadas application to start the 2D SRV application automatically, run it for 10s and exit. This removes the variability introduced due to needing user inputs to start the use case. We added the below line in `plat/linux/app/app_main.c:main()`.

```
/* Start 2D SRV automatically, wait for 10 s and exit
* the application */
if(!ret)
    app_start_chain(get_plugin_data("surr_view"),
            "lvds-srv-2d-880x1080");

if(!ret)
    sleep(10);

app_exit();
```

2. Redirect the stdout of the infoadas application to `/dev/null`. This removes the delays in application execution caused by the UART prints.

```
target # ./infoadas-app 1>/dev/null
```

3. Added a print to infoadas application that indicates that the first frame has been sent to window system. We use this print as

5. /etc/inittab is modified to autologin the user. /home/root/.profile is modified to start the script that does all the operations to launch infoadas application.

6. Set the loglevel to 0 in the dtb file.This removes the delays in application execution caused by the UART prints.

With these changes, it took 25.535 s to start 2D SRV from the SPL console print. The log shows that kernel boot is complete by 7.325 s but application is being launched only around 20.24 s.

```
[2.307246 2.307246] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
[0.004834 0.004834] DRA752-GP ES1.0
[0.048815 0.043981] spl_mmc_load_image
[0.099836 0.051021] reading dra7-evm.dtb
[0.107781 0.007945] reading uImage
[0.110745 0.002964] reading uImage
[0.282815 0.172070] reading initramfs.cpio.gz
[0.329818 0.047003] Jumping to Linux
[7.325766 6.995948] INIT: version 2.88 booting
[7.564786 0.239020] Starting udev
...
[20.243668 0.011014]
[20.382675 0.139007] cam device major num is 247
[23.294633 2.911958] INFO: Found a '7inch LCD' display(inactive) with size 800x480
...
[25.535595 0.013972] INFO: First buffer sent to display
```

6.4 Launching 2D SRV as the first service

The time between kernel boot and 2D SRV application is consumed by the launch of system services. To remove this factor, we can launch the 2D SRV application as the first system service to be launched.

To do this operation, we create a file /etc/init.d/infoadas with the below content.

```
#!/bin/bash
if [ ! -d /sys/devices ]; then
        mount -t sysfs none /sys
fi
echo "Sensor Step 1"
modprobe -q snd-soc-davinci-mcasp
modprobe -q ov1063x
echo Remoteproc
FW_NAMES="dra7-dsp1-fw.xe66 dra7-dsp2-fw.xe66 dra7-ipu1-fw.xem4"
for FW in $FW_NAMES ; do
    if [ -e /sys/class/firmware/$FW/loading ]; then
        echo "Loading $FW"
        echo 1 > /sys/class/firmware/$FW/loading
        cat /opt/infoadas/fw/$FW > /sys/class/firmware/$FW/data
        echo 0 > /sys/class/firmware/$FW/loading
    else
        echo "$FW loaded already"
    fi
done
cd /opt/infoadas
echo "Sensor Step 2"
```

```
./camnodes.sh
./memcache_load.sh
cd bin
echo Weston
modprobe -a omapdrm_pvr
pvrsrvinit
source /etc/profile.d/wayland_env.sh
weston --tty=1 --idle-time=0 --connector=4 &>/dev/null &
echo "2D SRV Start"
./infoadas-app 1>/dev/null
```

This file `/etc/init.d/infoadas` is symlinked as `/etc/rcS.d/S01infoadas.sh` making it the first service to run.

One key change from earlier is that the firmware loading for the remotecores is being handled through the `sysfs` interface instead of `udev`. As we are starting infoadas as the first system service, `udev` has not been started yet. By making 2D SRV launch as the first service, we can send the first output frames to display in 14.3 s.

```
[ 2.257213 2.257213] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
[ 0.004874 0.004874] DRA752-GP ES1.0
[ 0.048760 0.043886] spl_mmc_load_image
[ 0.099759 0.050999] reading dra7-evm.dtb
[ 0.106758 0.006999] reading uImage
[ 0.108729 0.001971] reading uImage
[ 0.231730 0.123001] reading initramfs.cpio.gz
[ 0.278758 0.047028] Jumping to Linux
[ 8.232734 7.953976] INIT: version 2.88 booting
[ 8.424702 0.191968] Sensor Step 1
[ 8.456717 0.032015] Remoteproc
[ 8.457838 0.001121] Loading dra7-dsp1-fw.xe66
[ 8.628704 0.170866] Loading dra7-dsp2-fw.xe66
[ 8.780700 0.151996] Loading dra7-ipu1-fw.xem4
[ 8.913706 0.133006] Sensor Step 2
[ 8.956699 0.042993] cam device major num is 247
[11.803685 2.846986] Weston
[11.967680 0.163995] 2D SRV Start
[12.060767 0.093087] INFO: Found a '7inch LCD' display(inactive) with size 800x480
[12.065829 0.005062] INFO: Found a 'HDMI' display(active) with size 1920x1080
[12.070838 0.005009] INFO: Found a 'FPDLink' display(inactive) with size 1280x720
[12.578640 0.507802] INFO: Creating plugin surr_view
[12.581798 0.003158] INFO: Creating chain surr_view:lvds-srv-2d-880x1080
[14.265641 1.683843] INFO: Chain surr_view:lvds-srv-2d-880x1080 has 2 eplinks
[14.270665 0.005024] INFO: Starting chain surr_view:lvds-srv-2d-880x1080
[14.274838 0.004173] INFO: Using properties 500x320 @ 10,200 on HDMI for surface 'Left Preview'
[14.281708 0.006870] INFO: Using properties 500x320 @ 10,600 on HDMI for surface 'Right Preview'
[14.288570 0.006862] INFO: Using properties 500x320 @ 1410,200 on HDMI for surface 'Front Preview'
[14.294823 0.006253] INFO: Using properties 500x320 @ 1410,600 on HDMI for surface 'Rear Preview'
[14.301761 0.006938] INFO: Using properties 880x1080 @ 520,0 on HDMI for surface '2-D stitched surround
[14.309751 0.007990] INFO: VIVI thread for chain surr_view:lvds-srv-2d-880x1080 started
[14.317597 0.007846] INFO: First buffer sent to display
```

6.5 Optimizing Sensor configuration

2.86 s of the time listed above is spent in sensor configuration in user space. This is in addition to the spent during the module initialization. To reduce this time, we pick a patch from the GLSDK kernel GA branch that

1. Reduces the number of register writes to the sensor thereby reducing the time taken to configure the sensor.
2. Configures the sensor by default in the mode required by 2D SRV. This eliminates the need to configure the sensor from user space.

At the same time, we can also remove the camera nodes and serializers that are not required for the 2D SRV usecase from the device tree. Removing these nodes reduces unnecessary probing/initialization.

```
[2.328209 2.328208] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
[0.004870 0.004870] DRA752-GP ES1.0
...
[0.235763 0.125032] reading initramfs.cpio.gz
[0.282763 0.047000] Jumping to Linux
[5.629779 5.347016] INIT: version 2.88 booting
[5.817722 0.187943] Sensor Step 1
[5.844743 0.027021] Remoteproc
...
[6.368718 0.061993] Weston
[6.530732 0.162014] 2D SRV Start
...
[8.891676 0.017931] INFO: First buffer sent to display
```

With these changes, the spent in kernel initialization comes down to 5.6 s from 7.9 s. Also the 2.8s spent in sensor configuration in user space is removed. 2D SRV can now be shown on screen in 8.89 s.

6.6 Paring down the kernel

Even with this change, Kernel initialization takes 5.9s. The default GLSDK kernel includes builtin functionality for a wide range of use cases. We can optimize this by

1. moving functionality not required for 2D SRV into modules.
2. Disabling unused functionality
3. Deferring the initialization of functionality that is not required for the usecase but cannot be converted into a module.

We applied an additional kernel config fragment on top of the default infoadas configuration to keep only the required functionality in the kernel. This config fragment can be found at `boot_scripts/config_fragments/boot_opt.cfg`. With this change, we could see the kernel initialization time come down to 4.19 s from 5.62 s and the 2D SRV usecase launch time come down to 7.9 s from 8.89 s. The log can be seen below.

```
[2.281260 2.281260] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
...
[0.199772 0.088002] reading initramfs.cpio.gz
[0.246755 0.046983] Jumping to Linux
[4.438758 4.192003] INIT: version 2.88 booting
...
[7.909650 0.018948] INFO: First buffer sent to display
```

One point to note is that the mmc file system needed periodic cleanup and fsck to keep the boot time low. The kernel load time increased as modules from different kernel builds piled up in `/lib/modules`.

6.7 Profiling the kernel boot time

Even after this optimization, 4.19 s is being spent in kernel initialization. To find out where the time is being spent in the kernel, we profiled the kernel boot process using the `initcall_debug` boot option. To enable timing information, we also increased the size of the kernel log buffer and included all debug symbols in the kernel image temporarily.

The captured log was processed to calculate the time spent in initcalls using the below command

```
dmesg -s 128000 | grep "initcall" | sed "s/\(.*\)after\(.*\)/\2 \1/g" | sort -n | tail
```

The processed log is shown below. Out of the time shown below

1. 1.8 s is consumed by sensor initialization in `ov1063x_i2c_driver_init`.
2. 0.436 s is consumed by the `fpd3_serdes_driver_init`. This function configures the serializer and deserializers through which the cameras are connected to the EVM.
3. The MMC driver which is carrying the file system takes a 100 ms to initialize `omap_hsmmc_driver_init`.
4. Deferred probe initcalls i.e. the polling loop for modules whose dependencies were not available when they were first probed takes another 100 ms `deferrered_probe_initcall`.
5. `omap_drm_init` which does the display initialization takes 283 ms due to interaction with HDMI. If we were using an LCD for display instead of HDMI, this time would be atleast 100 ms lower.

```
87890 usecs [    0.521624] initcall omap_i2c_init_driver+0x0/0x20 returned 0
92147 usecs [    3.466059] initcall deferred_probe_initcall+0x0/0x8c returned 0
115197 usecs [    3.058203] initcall omap_hsmmc_driver_init+0x0/0x20 returned 0
195312 usecs [    0.281262] initcall __omap_hwmod_setup_all+0x0/0x48 returned 0
283446 usecs [    3.370357] initcall omap_drm_init+0x0/0x9c returned 0
436881 usecs [    1.035729] initcall fpd3_serdes_driver_init+0x0/0x20 returned 0
1812971 usecs [    2.936840] initcall ov1063x_i2c_driver_init+0x0/0x20 returned 0
```

We also plotted the boot graph to better visualize where the time is spent.

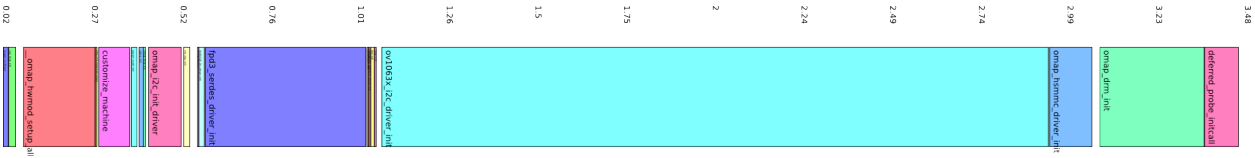

Figure 3: Bootgraph after first level optimization

Searching the `dmesg` logs for deferred call shows that most of the deferrals from regulators which are required to enabled mmc or from sound related modules.

```
bash-3.2# dmesg | grep defer
[    0.422615] platform fixedregulator-sd: Driver reg-fixed-voltage requests probe deferral
[   0.423133] platform fixedregulator-evm_3v3_sw: Driver reg-fixed-voltage requests probe deferral
[   0.423632] platform fixedregulator-aic_dvdd: Driver reg-fixed-voltage requests probe deferral
[    0.424531] platform fixedregulator-vtt: Driver reg-fixed-voltage requests probe deferral
[    2.941012] platform 4809c000.mmc: Driver omap_hsmmc requests probe deferral
[    2.941487] platform 480b4000.mmc: Driver omap_hsmmc requests probe deferral
[    3.068527] i2c 0-0019: Driver tlv320aic3x-codec requests probe deferral
[    3.068684] i2c 1-0018: Driver tlv320aic3x-codec requests probe deferral
[    3.068828] i2c 1-0019: Driver tlv320aic3x-codec requests probe deferral
[    3.068972] i2c 1-001a: Driver tlv320aic3x-codec requests probe deferral
```

```
[    3.069483] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
[    3.075025] platform primary_sound: Driver davinci_evm requests probe deferral
[    3.075555] platform jamr3_sound: Driver dra7xx-jamr3-snd requests probe deferral
[    3.371682] calling  deferred_probe_initcall+0x0/0x8c @ 1
[    3.449134] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
[    3.466005] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
[    3.466059] initcall deferred_probe_initcall+0x0/0x8c returned 0 after 92147 usecs
[    3.585056] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
[    3.629413] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
```

We can also see 200 ms being spent waiting for the mmc device carrying the root partition to become available. Including the time spent in the mmc driver init call, about 300 ms is being consumed by MMC related functionality.

```
[    3.467883] Waiting for root device /dev/mmcblk1p2...
[   3.568270] mmc1: host does not support reading read-only switch. assuming write-enable.
[    3.573908] mmc1: new high speed SDHC card at address aaaa
[    3.574375] mmcblk0: mmc1:aaaa SU16G 14.8 GiB
[    3.583799]  mmcblk0: p1 p2
[    3.585056] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
[    3.620890] mmc2: BKOPS_EN bit is not set
[    3.623481] mmc2: new high speed DDR MMC card at address 0001
[    3.623919] mmcblk1: mmc2:0001 MMC04G 3.52 GiB
[    3.624087] mmcblk1boot0: mmc2:0001 MMC04G partition 1 16.0 MiB
[    3.624258] mmcblk1boot1: mmc2:0001 MMC04G partition 2 16.0 MiB
[    3.625731]  mmcblk1: p1 p2
[    3.627517]  mmcblk1boot1: unknown partition table
[    3.628785]  mmcblk1boot0: unknown partition table
[    3.629413] platform bt_sco_card: Driver asoc-simple-card requests probe deferral
[    3.687771] async_waiting @ 1
[    3.687781] async_continuing @ 1 after 0 usec
[   3.729704] EXT4-fs (mmcblk1p2): mounted filesystem with ordered data mode. Opts: (null)
[    3.729736] VFS: Mounted root (ext4 filesystem) on device 179:10.
```

From the profiling information that we have gathered, the following are candidates for optimization.

1. OV1063x sensor initialization
2. fpd3_serdes_driver_init function.
3. MMC functionality converted into a module
4. making sound functionality into a module.

Items (1) and (2) listed above are being planned as part of further work. We will look into (3) and (4) below.

6.8 Using initramfs

As we are carrying the filesystem on MMC, removing the MMC driver from kernel requires us to provide an alternate mechanism to mount the root filesystem. Such mechanism available in the linux kernel is called initramfs i.e. Initial RAM filesystem. It is typically used for two purposes.

1. Linux distributions like Ubuntu use initramfs to support a variety of machines. A generic kernel is used alongside a machine specific initramfs to start the required drivers to mount the final filesystem.
2. In embedded systems, it is used to start an application before mounting the final system. initramfs is referred to in some documentation as “early userspace”.

Both these purposes are applicable to us. We can use initramfs to first launch 2D SRV and then mount the final filesystem from MMC.

Most of the complexity in using initramfs comes from the work required to include all the dependencies for the executables and modules into the initramfs image. To solve this issue, we have put together a script `boot_scripts/mkifs.sh`. The script and its input and outputs are described in the next section.

6.9 Building a initramfs image

This package includes a helper script to build the initramfs image. The script can be found in `boot_scripts/mkifs.sh`. The script internally invokes `boot_scripts/gen_ifs_cfg.pl`. `gen_ifs_cfg.pl` requires the following inputs.

1. `modules.txt` - This file should list the name of the modules that need to be included in the initramfs.
2. `files.txt` - This file should list the name of files and directories that need to tbe included in the initramfs. The script automatically includes any shared libraries that requires for binaries. If there are any libraries that are loaded using `dlopen()`, they need to specified in this file.

`gen_ifs_cfg.pl` expects two input arguments.

1. location of the targetfs containing the infoadas build output binaries. It is expected that the required kernel modules are also present in this rootfs in the right subdirectory.
2. Boot parition to place the final binaries.

It also accesses the linux kernel location via the environment variable `LINUX_KERNELDIR`.

With these inputs, `gen_ifs_cfg.pl` produces a text file that can be fed to the `gen_init_cpio` binary part of the kernel to produce a cpio archive. The cpio archive can be compressed and loaded via U-Boot or MLO.

`mkifs.sh` also updates the dtb file to indicate the location of the initramfs to the kernel. This is mandatory as the start and end of the initramfs must be indicated to the kernel via the device tree.

6.10 Testing basic initramfs build

We can use initramfs as the file system without removing MMC and Sound functionality from the kernel. Without mounting the file system from MMC, we expect to see some savings from the time spent in mounting the MMC partition. However this is offset by the time required to copy initramfs from eMMC to RAM before the kernel loads. The boot logs obtained are shown below.

```
[2.795054 2.425066] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
[0.005013 0.005013] DRA752-GP ES1.0
...
[0.109845 0.001958] reading uImage
[0.196914 0.087069] reading initramfs.cpio.gz
[0.715918 0.519004] Jumping to Linux
[5.477881 4.761963] INIT
...
[8.474847 0.016954] INFO: First buffer sent to display
```

As expected, we do not see any improvement in the boot time. Loading initramfs from eMMC to RAM takes ~520 ms. We see an increase of roughly the same amount in the boot time of the 2D SRV usecase from 7.9 s previously to 8.47 s.

6.11 Making MMC into a module

Next we made MMC into a module by applying an additional config fragment in `boot_scripts/config_fragments/boot_opt_initramfs.c`
When we profiled the result, making MMC into a module does not yield the expected improvement.

```
[2.262217 2.262217] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
[0.005765 0.005765] DRA752-GP ES1.0
...
[0.282788 0.169061] reading initramfs.cpio.gz
[0.875762 0.592974] Jumping to Linux
[5.426724 4.550962] INIT
...
[8.399760 0.016069] INFO: First buffer sent to display
```

We only see a 75 ms reduction in the boot time and time to usecase of 8.4 s . In fact this is 500 ms worse compared to the best time we achieved without using initramfs (7.9 s).

6.12 Profiling the initramfs boot

We profiled the kernel boot process again with `initcall_debug` option and found that decompressing the initramfs was consuming 1 s and all the execution was blocked on this decompression.

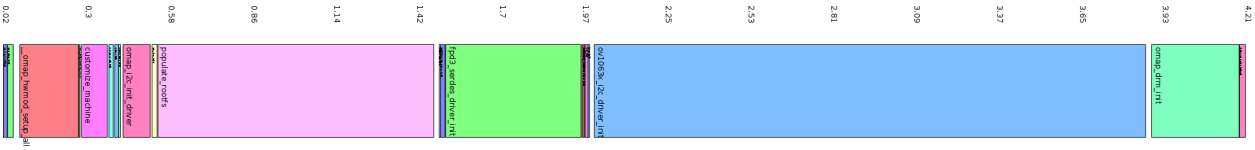

Figure 4: Bootgraph after enabling IFS

6.13 Optimizing initramfs boot

Fortunately the expansion and mounting of the initramfs is completely independent of other boot activities. The required data is already present at start of kernel boot and the filesystem is not required until we are ready to start init. If we make the initramfs decompression use some of the time spent waiting for the peripherals to respond, we can reduce the boot time.

Linux kernel provides a mechansim called `async` to perform operations in a worker thread. The expansion and mounting of initramfs is done from the `populate_rootfs` init call. By making the `populate_rootfs` run in an asynchronous manner, we can hide the latency of mounting initramfs behing some of the wait times in the other kernel initializations. The first patch listed below does this. This patch is based on http://patchwork.ozlabs.org/patch/40728/ posted to Ubuntu kernel team mailing list.

In addition to the above patch, we also changed the order of the process in kernel init so that the asynchronous populate_rootfs starts earlier and finishes in time to make it truly parallel.

File.name	Functionality
0005-Populate-rootfs-asynchronously-and-early.patch	Make populate_rootfs asynchronous.
0006-init-turn-populate_rootfs-into-a-postcore_initcall.patch	Invoke populate_rootfs earlier in the kernel boot process.

With this change, we are able to launch the 2D SRV usecase in **7.6s** from the first SPL console print. The console log is shown below. The initialization time for the kernel has come down to 3.73 s from 4.5s. leading to a corresponding decrease in the time

to launch the usecase.

```
[2.238247 2.238247] U-Boot SPL 2014.07-00005-g26a6466 (Aug 23 2015 - 16:02:39)
[0.004967 0.004967] DRA752-GP ES1.0
...
[0.266760 0.155035] reading initramfs.cpio.gz
[0.874764 0.608004] Jumping to Linux
[4.614732 3.739968] INIT
...
[5.580821 0.003096] INFO: Creating chain surr_view:lvds-srv-2d-880x1080
[7.538768 1.957947] INFO: Chain surr_view:lvds-srv-2d-880x1080 has 2 eplinks
[7.543729 0.004961] INFO: Starting chain surr_view:lvds-srv-2d-880x1080
...
[7.598692 0.015845] INFO: First buffer sent to display
```

From the boot graph, we can see that the time required to expand initramfs is hidden behind other operations.

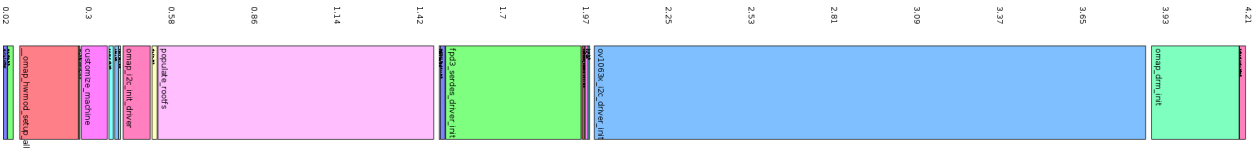

Figure 5: Bootgraph after parallelizing initramfs decompression

7 Final Optimized times

Based on the optimizations done so far

1. using a single stage boot loader
2. using an initramfs
3. using a pared down kernel
4. reducing the number of console accesses from the application

we were able to achieve a boot time of 7.6 s for 2D SRV when booting from eMMC. The time is measured from the first console print in the boot loader to the time at which the first buffer is handed over to the display.

7.1 Execution time breakdown

From the console log and boot graph, the major components that contribute to this 7.6s time are listed below with their contributions.

Table 10: Execution breakdown after optimization

Function	Time.consumed (s)
Bootloader	0.8
Kernel initialization excluding sensor configuration	1.8
Sensor initialization in kernel	1.9
Application: chain creation	1.9

These four components need to be profiled, optimized and parallelized to bring down the boot time for 2D SRV further.

7.2 Optimization approaches

Some of the optimization approaches we are considering to reduce the boot time further are listed below.

1. Configure the sensors using an external microcontroller as is usually done in production.
2. Set the A15 to the maximum operating frequency by default.
3. Leverage the 2nd A15 core as early as possible for Kernel operations.
4. Check the impact of various compression schemes on initramfs decompression times and choose the best scheme based on transfer time and decompression time.
5. Use LCD for displaying the output instead of HDMI. This saves time in configuring DSS and launching Weston.
6. Build binaries in initramfs with alternate `libc` to save on size.
7. Statically link the binaries in initramfs to reduce time spent searching for dependencies at load time.
8. Configure the library search path to reduce the number of system calls made by the binaries `pvrsrvinit`, `weston` and `infoadas-app`.
9. The application needs to be profiled with tools such as `perf` and `strace` to understand the bottleneck and optimize it.
10. Modify the OV1063x driver to use a worker thread i.e. async interface so that other kernel operations can proceed in parallel.

8 Summary

In this document, we described how we went about optimizing the boot time of the 2D SRV application part of the infoadas release. We also described the tools we used/developed during the optimization process and how our results can be reproduced. We achieved a boot time of 7.6s from the first SPL print to sending the first frame out to the HDMI display.

We are considering other optimization options to bring the boot time down further. We will update this document once the investigation is complete.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI's terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers' products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers' products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have not been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments,Post Office Box 655303,Dallas, Texas 75265