Processor SDK Linux Automotive Software Developers Guide

Last updated: **12/13/2019**

Welcome to the DRA7xx Processor SDK Linux Automotive Software Developer's Guide.

Thank you for choosing the DRA7xx EVM for your application. The purpose of this guide is to get you going with developing software for the DRA7xx on a Linux development host only.

NOTE! This is a live Wiki page that gets updated for every release . For the Software Developers Guide that is specific to the release you are using, please refer to the SW Dev Guide link in Processor SDK Linux Automotive Landing Page (http://processors.wiki.ti.com/index.php/Category:Processor_SDK_Linux_Automotive)

Contents

Introduction

Some commands are to be executed on the Linux development host, some on the Linux target and some on the u-boot (bootloader) prompt. The following conventions are used to distinguish the commands on a host and on the target:

```
host $ <this command is to be executed on the host>
target # <this command is to be executed on the target>
u-boot :> <this command is to be executed on the u-boot prompt>
```

Note! All instructions in this guide are for Ubuntu 14.04 (http://releases.ubuntu.com/14.04/) on 64-bit machine. At this time, this is the only supported Linux host distribution for development.

Starting your software development

Setup up ARM linux development Environment on the host. Please refer to this link to see how to set one up.

Configuration of ARM Linux development Environment (http://processors.wiki.ti.com/index.php/How_to_Build_a_Ubuntu_Linux_host_under_VirtualBox)

Step 1: Install the Processor SDK Linux Automotive release on the host machine.

- Download the Processor SDK Linux Automotive installer.
- If necessary make the installer executable manually by executing:

```
host $ chmod +x ti-processor-sdk-linux-automotive-dra7xx-evm-06.00.00.03-installer.bin
```

- Execute the installer on the host and follow the instructions:

```
host $ ./ti-processor-sdk-linux-automotive-dra7xx-evm-06.00.00.03-installer.bin
```

Step 2: Setup the INSTALL_DIR environment variable to the location where the Processor SDK Linux Automotive is installed (the following assumes that Processor SDK Linux Automotive was installed at default location):

```
host $ export INSTALL_DIR="${HOME}/ti-processor-sdk-linux-automotive_dra7xx-evm_06_00_00_03"
```

Step 3: Setup the Processor SDK Linux Automotive on host

Host Setup

- **Ubuntu**

The recommended Linux distribution is Ubuntu 14.04 or Ubuntu 16.04.

The following build host packages are required for Ubuntu. The following command will install the required tools on the Ubuntu Linux distribution.

For Ubuntu 14.04 and 16.04, please run the following:

```
$ sudo apt-get install git build-essential python diffstat texinfo gawk chrpath dos2unix wget unzip socat doxygen libc6:i386 libncurses5:i386 libstdc++6:i386 libz1:i386
```

The build also requires that `bash` is configured as the default system shell. The following steps will configure Ubuntu to use `bash` instead of the default `dash`.

```
$ sudo dpkg-reconfigure dash

(Select "no" when prompted)
```

Code Composer Studio

NOTE! This step is required only if you want to use Code Composer Studio IDE. In the latest Processor SDK Linux Automotive releases, M4/DSP firmware image build is handled as part of SDK Yocto build.

Code Composer Studio 6.1.3.00034 must be downloaded manually and placed into either the downloads directory, or placed on a mirror. Please see CCS v6 Download (http://processors.wiki.ti.com/index.php/Download_CCS#Code_Composer_Studio_Version_6_Downloads) for details and download the Linux off-line installation tarball, e.g., CCS6.1.3.00034_linux.tar.gz.

> **NOTE**
>
> If the CCS Linux tarball is copied to tisdk/downloads directory, please also run "touch tisdk/downloads/CCS6.1.3.00034_linux.tar.gz.done".

The Processor SDK Linux Automotive comes with a script for setting up your Ubuntu 14.04 LTS development host. It is an interactive script, but if you accept the defaults by pressing return you will use the recommended settings. This is recommended for first time users. Note that this script requires ethernet access as it will update your Ubuntu Linux development host with the packages required to develop using the Processor SDK Linux Automotive.
Note: Please make sure that the proxy settings are done for http, https, git, ftp and wget before proceeding further.
Execute the script in the Processor SDK Linux Automotive release directory using:

```
host $ cd ${INSTALL_DIR}
host $ ./setup.sh
```

The setup script would perform the following operations:

1. Installs all the necessary package on the host for the SDK.
2. Prepares the UART terminal to communicate with the target over Debug USB on Minicom. If you want to use a windows host for connecting to the target instead, see the #Setting_up_Tera_Term section
3. Setups the linaro cross compiler
4. Installs the dependencies for the repo tool.
5. Initialize the repo by pointing it to Processor SDK Linux Automotive release manifest location.

To start minicom on your Linux development host execute *minicom -w* (or Tera Term on Windows).

Step 4: Prepare SD card
To install the release image, you need a µSD Card (at least 4GB) with 2 partitions:

- boot (vfat) partition.
- rootfs (ext4 or ext3) partition.

The following procedure prepares the sdcard: (however, the user can choose to do it manually as well)

- Plug an SD card reader to your PC and insert a µSD card. It must be at least 4GB size.
- Identify which device corresponds to the SD card reader. `sudo fdisk -l` command can help you find out where the µSD Card is mapped. We will call it `/dev/sdY` here.

```
Note : If you are using NFS file system, then edit ${INSTALL_DIR}/board-support/pre-built-images/uenv.txt, add "ip=dhcp" in bootargs.
```

- Re-format your µSD card with this script mksdboot.sh from the bin directory in the Processor SDK Linux Automotive

```
$ sudo ${INSTALL_DIR}/bin/mksdboot.sh --device /dev/sdY --sdk ${INSTALL_DIR}
```

The above script would prepare the SD card with the prebuilt images and yocto filesystem for SD boot.

Step 5: Booting the board
To boot the board with the above created SD card, refer to the **Quick Start Guide** at the Processor SDK Linux Automotive download location *Link (http://software-dl.ti.com/infotainment/esd/jacinto6/processor-sdk-linux-automotive/latest/index_FDS.html)* to setup the board.
Then, power cycle the board and login with username as **root**.

Repo tool Usage

Starting source code development using repo tool

The Processor SDK Linux Automotive release uses the repo tool to effectively manage the different components.

NOTE :

1: The first step to the repo tool is the repo initialization and this is done as part of the $INSTALL_DIR/setup.sh script

2: The repo tool is downloaded into the bin folder in the Processor SDK Linux Automotive directory. Please ensure that this path is updated in the environment variable as shown below

```
host $  export PATH=${INSTALL_DIR}/bin:$PATH
```

The Processor SDK Linux Automotive release contains a helper script that sets up the development environment. Run the script as shown below:

```
host $ cd ${INSTALL_DIR}
host $ ./bin/fetch-sources.sh
```

The script does the following:

- Check for the repo tool.
- Perform repo sync
- Create a branch called **dev**
- Checkout the branch **dev**

It is expected that the development is done on the **dev** branch.

How to get updates

If there are changes in the remote repositories, it could be fetched using the same script.
However, please make note of these important points.

1. The script will fetch the latest changes, and switch back to the **dev** branch.
2. The new updates from the remote, will be available in the master branch.
3. The decision on whether to merge the changes to the local branch or merge the local branch to the master is left to the developer

Building Yocto Filesystem

Please ensure that the setup.sh script is run as described in Starting your software development section and answer 'yes' to fetch-sources prompt.
Before building the filesystem, **ensure that the svn, http, ftp and git proxies are set correctly**. Refer to the following link for these settings https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy.

Add the Linaro cross-compile toolchain path in the PATH environment variable.

```
host $ export PATH=<Path to Linaro cross-compile toolchain>/bin:$PATH
```

Run this command to as a one-time setup for the yocto build

```
host $ cd $INSTALL_DIR
```

```
NOTE:
# Prior to running the setup-yocto.sh script, make sure that GCC Linaro cross compiler installation path and the CCS binary is downloaded into a particular directory
# Update the ./bin/setup-yocto.sh file to indicate this path, the default path is set to /sdk/tools and this may vary on every machine.
```

```
host $ ./bin/setup-yocto.sh
```

For building core sdk , run the build-core-sdk.sh passing machine name as an argument.

```
host $ cd yocto-layers
```

Create a downloads directory (if building using Yocto for the first time), where the Yocto build will place the downloads. Note the path of the directory.

```
host $ mkdir downloads
```

NOTE: Please pass the same downloads directory path to the following build-script when prompted

```
host $ ./build-core-sdk.sh dra7xx-evm
```

These scripts will build the tisdk-rootfs-image.

After build is complete the generated images can be found in yocto-layers/build/arago-tmp-external-linaro-toolchain/deploy/images/

Generated images

Image name	Description
tisdk-rootfs-image-<*MACHINE-NAME*>-<*DATE*>.rootfs.tar.gz	This is the filesystem tarball. Copy and extract it on the rootfs partition of the boot media.
zImage-<*MACHINE-NAME*>.bin	zImage for the machine. Copy as zImage in /boot folder of the rootfs partition.
zImage-<*MACHINE-NAME*>.dtb	Copy dra7-evm.dtb in boot partition. Or choose to copy the appropriate dtb based on the display and additional boards that are connected.
u-boot-<*MACHINE-NAME*>.img	Copy as u-boot.img in boot partition.

Note: The build does not generate a uenv.txt. You need to copy it from the prebuilt binaries in the release.

Modifying source code and rebuilding a component

Once the Yocto setup is complete developers would like to modify a certain component source code and rebuild it. The source code for the generic components like omapdrmtest can be found in ${INSTALL_DIR}/yocto-layers/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-neon-linux-gnueabi/

Steps to rebuild:

1. Modify the source of the component in its work area

2. cd to yocto-layers

```
host $ cd ${INSTALL_DIR}/yocto-layers
```

3. export PATH if not done previously:

```
host $ export PATH=<Path to Linaro cross-compile toolchain>/bin:$PATH
```

4. Run one/combination of the folowing tasks:

a. To configure and compile with the latest changes in the work area:

```
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c configure <RECIPE-NAME>
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c compile <RECIPE-NAME>
```

For example, if the omapdrmtest source is modified and you want to generate only the binaries for dra7xx-evm, the compile task should suffice:

```
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c compile omapdrmtest
```

The binaries will be present in the same folder as the source files.

b. To generate .ipk package in addition to the binaries, following additional tasks need to be run after above steps:

```
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c install <RECIPE-NAME>
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c package <RECIPE-NAME>
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c package_write_ipk <RECIPE-NAME>
```

This will generate .ipk packages for the recipe. They can be found in the deploy-ipks folder of the work area of the recipe.
The .ipk packages can be copied and installed on the target by:

```
target $ opkg install <PACKAGE-NAME>
```

This approach is useful in the case of recipes that generate a large number of binaries, that are difficult to copy manually, like gst-plugins-bad.

c. During building and debugging the kernel or kernel modules, the compile_kmodules task needs to be executed that generates kernel modules.

```
host $ ./build-specific-recipe.sh <MACHINE-NAME> -f -c compile_kmodules <RECIPE-NAME>
```

For a complete set of tasks that a specific recipe executes during its build, please refer to the log.task_order in the temp folder of the component.
Pass the relevant tasks to build-specific-recipe.sh for required outcome.

5. For a robust solution , once the change in source area is tested with the above steps, please update the recipe.

a. create a patch of the change
b. copy it in the folder where the recipe is present
c. add the patch name in the SRC_URI variable in the recipe

The build-specific-recipe.sh recognizes the change in the recipe and builds it if required.

To build a specific component after recipe is updated, use the build-specific-recipe.sh

```
host $ ./build-specific-recipe.sh <MACHINE-NAME> <RECIPE NAME>
```

For example to build omapdrmtest:

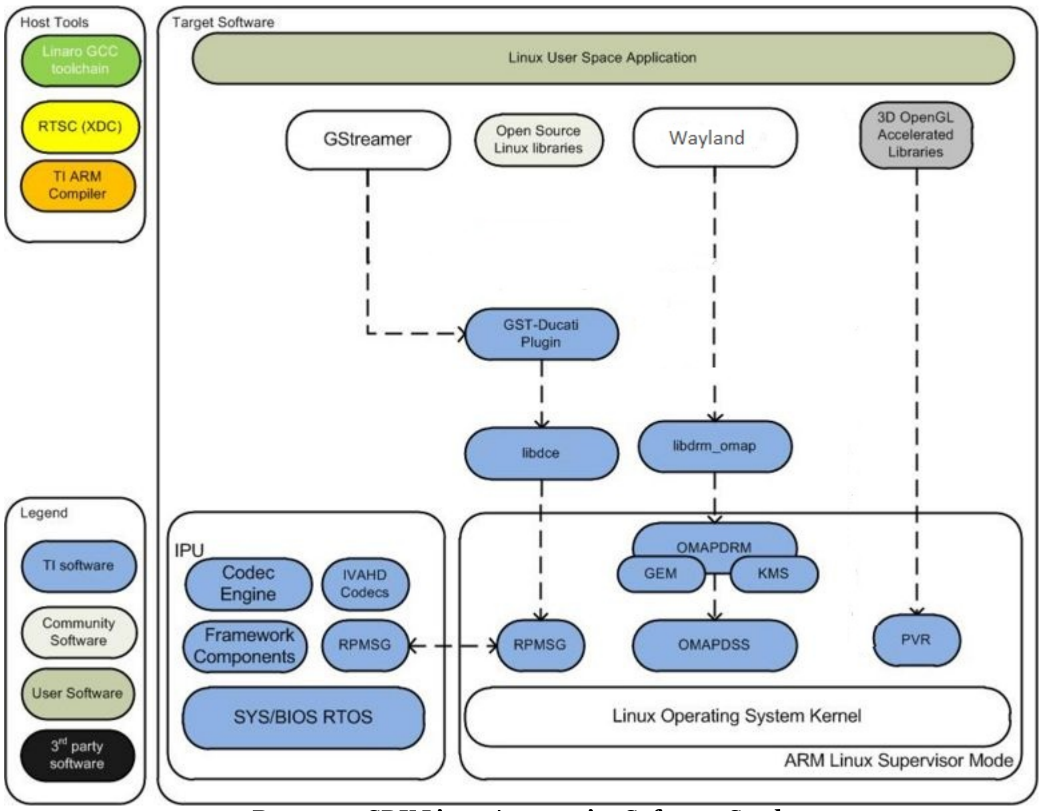
```
host $ ./build-specific-recipe.sh <MACHINE-NAME> omapdrmtest
```

MACHINE-NAME can be either omap5-evm or dra7xx-evm

To clean a specific component:

```
host $ ./clean-specific-recipe.sh <MACHINE-NAME> <RECIPE NAME>
```

Processor SDK Linux Automotive software overview



Processor SDK Linux Automotive Software Stack

The Processor SDK Linux Automotive contains many software components. Several components are developed by the open source community (white). A few components that leverage HW acceleration in SoC are developed by Texas Instruments. TI contributes, and sometimes even maintains, some of these open source community projects, but the support model is different from a project developed solely by TI.

Running Examples

Running OMAP DRM DSS Examples

The drmclone, drmextended, and modetest examples demonstrates how to create a CRTC (i.e. FB) and display planes (overlays) on the CRTC. Additionally, drmtest demonstrates similar functionality as the previously mentioned demos, along with dynamic plane updates for 2 CRTCs.

Retrieve the omapdrm-tests source

```
git clone https://github.com/tomba/omapdrm-tests.git
cd omapdrm-tests
```

Run (or example planescale)

```
./planescale
```

Graphics Demos from Command Line

The graphics driver and userspace libraries and binaries are distributed along with the SDK.

Graphic demos can also run from command line. In order to do so, exit Weston by pressing Ctrl-Alt-Backspace from the keyboard which connects to the EVM. Then, if the LCD screen stays in "Please wait...", press Ctrl-Alt-F1 to go to the command line on LCD console. After that, the command line can be used from serial console, SSH console, or LCD console. | Graphic demos can also run from command line. In order to do so, exit Weston by pressing Ctrl-Alt-Backspace from the keyboard which connects to the EVM. Then, if the LCD screen stays in "Please wait...", press Ctrl-Alt-F1 to go to the command line on LCD console. After that, the command line can be used from serial console, SSH console, or LCD console.

Please make sure the board is connected to atleast one display before running these demos.

Finding Connector ID

Note: Most of the applications used in the Demos would require the user to pass a connector id. A connector id is a number that is assigned to each of the display devices connected to the system. To get the list of the display devices connected and the corresponding connector id one can use the **modetest** application (shipped with the file system) as mentioned below:

```
 target #  modetest
```

Look for the display device for which the connector ID is required - such as HDMI, LCD etc.

```
Connectors:
id      encoder status          type    size (mm)       modes   encoders
4       3       connected       HDMI-A  480x270         20      3
  modes:
        name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1920x1080 60 1920 2008 2052 2200 1080 1084 1089 1125 flags: phsync, pvsync; type: preferred, driver
...
16      15      connected       unknown 0x0             1       15
  modes:
        name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  800x480 60 800 1010 1040 1056 480 502 515 525 flags: nhsync, nvsync; type: preferred, driver
```

Usually, LCD is assigned 16 (800x480), and HDMI is assigned 4 (multiple resolutions). |

```
Connectors:
id  encoder status      type    size (mm)   modes   encoders
4   3   connected   unknown 0x0     1   3
  modes:
    name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1280x800 60 1280 1328 1360 1404 800 804 811 823 flags: nhsync, nvsync; type: preferred, driver
...
16  11  connected   HDMI-A  700x390     31  11
  modes:
    name refresh (Hz) hdisp hss hse htot vdisp vss vse vtot)
  1280x720 60 1280 1390 1430 1650 720 725 730 750 flags: phsync, pvsync; type: preferred, driver
```

Usually LCD is assigned 4 (800x480), HDMI is assigned 16 (multiple resolutions).

Finding Plane ID

To find the Plane ID, run the modetest command:

```
 target #  modetest
```

Look for the section called Planes. (Sample truncated output of the Planes section is given below)

```
Planes:
id      crtc    fb      CRTC x,y        x,y     gamma size
19      0       0       0,0             0,0     0
 formats: RG16 RX12 XR12 RA12 AR12 XR15 AR15 RG24 RX24 XR24 RA24 AR24 NV12 YUYV UYVY
 props:
 ...
20      0       0       0,0             0,0     0
 formats: RG16 RX12 XR12 RA12 AR12 XR15 AR15 RG24 RX24 XR24 RA24 AR24 NV12 YUYV UYVY
 props:
 ...
```

kmscube

Run kmscube on default display (HDMI):

```
 target # kmscube
```

Run kmscube on default display (LCD):

```
 target # kmscube
```

Run kmscube on secondary display (HDMI):

```
 target # kmscube -c <connector-id>
 target # kmscube -c 16 #Usually, the connector id for HDMI is 16.
```

Run kmscube on all connected displays (LCD & HDMI & FPDLink(optional)):

```
target # kmscube -a
```

Run kmscube on default display (HDMI):

```
target # kmscube
```

Run kmscube on secondary display (LCD):

```
target # kmscube -c <connector-id>
target # kmscube -c 16 #Usually, the connector id for LCD is 16.
```

Run kmscube on all connected displays (LCD & HDMI):

```
target # kmscube -a
```

kmscube with video

This demo allows a video frame to be applied as a texture onto the surface of the kmscube. The user can invoke the demo by following the syntax below:

```
target # viddec3test <path_to_the_file> --kmscube --connector <connector_number>
```

This feature is not supported on OMAP5 based releases.

Run kmscube with video on default display (LCD):

```
target # viddec3test <path_to_the_file> --kmscube
```

Run kmscube with video on secondary display (HDMI):

```
target # viddec3test <path_to_the_file> --kmscube --connector 16 #Usually, the connector id for HDMI is 16.
```

Run kmscube with video on default display (HDMI):

```
target # viddec3test <path_to_the_file> --kmscube
```

Run kmscube with video on secondary display (LCD):

```
target # viddec3test <path_to_the_file> --kmscube --connector 16 #Usually, the connector id for HDMI is 16.
```

Additionally, to change the field of view of the rotating cube, the user can specify the same on the command line like below:

```
target # viddec3test <path_to_the_file> --kmscube --connector <connector_number> --fov <number>
```

Wayland/Weston

```
Wayland/Weston version brings in the multiple display support in extended desktop mode and the ability to drag-and-drop windows from one display to the other.
```

To execute the demos, the graphics driver must be initialized by running start weston, if this has not been done earlier.

```
target # /etc/init.d/weston start
```

To launch weston without using systemd init scripts, do the following:

On all connected displays (LCD, HDMI and FPDLink):

```
target # weston --tty=1 --backend=drm-backend.so
```

On default display (HDMI):

```
target # weston --tty=1 --connector=4
```

On secondary display (LCD):

```
target # weston --tty=1 --connector=16
```

On all connected displays (LCD and HDMI):

```
target # weston --tty=1
```

By default, the screensaver timeout is configured to 300 seconds.

The user can change the screensaver timeout using a command line option

```
--idle-time=<number of seconds>
```

To disable the screen timeout and to configure weston configured to display on all connectors, use the option with "0" as the input:

```
--idle-time=0
```

The filesystem comes with a preconfigured **weston.ini** file which will be located in

/etc/weston.ini | **/etc/weston.ini**

Running weston clients

Weston client examples can run from the command line on serial port console or SSH console. After launching weston, the user should be able to use the keyboard and the mouse for various controls.

```
      # /usr/bin/weston-flower
      # /usr/bin/weston-clickdot
      # /usr/bin/weston-cliptest
      # /usr/bin/weston-dnd
      # /usr/bin/weston-editor
      # /usr/bin/weston-eventdemo
      # /usr/bin/weston-image /usr/share/weston/terminal.png
      # /usr/bin/weston-resizor
      # /usr/bin/weston-simple-egl
      # /usr/bin/weston-simple-shm
      # /usr/bin/weston-simple-touch
      # /usr/bin/weston-smoke
      # /usr/bin/weston-info
      # /usr/bin/weston-terminal
```

| There is one icon on the top right hand corner of the weston desktop window which has been configured for

- weston-terminal

Clicking this icon should launch the applications on the Weston Desktop.

It is possible to add other icons by editing the weston.ini file.

There are several other applications that are included in the default filesystem. To invoke these applications, the user should launch the weston-terminal (top right hand corner of the desktop) and then invoke the client apps as described below from within the terminal window:

```
      wayland sh # /usr/bin/weston-flower
      wayland sh # /usr/bin/weston-clickdot
      wayland sh # /usr/bin/weston-cliptest
      wayland sh # /usr/bin/weston-dnd
      wayland sh # /usr/bin/weston-editor
      wayland sh # /usr/bin/weston-eventdemo
      wayland sh # /usr/bin/weston-image /usr/share/weston/terminal.png
      wayland sh # /usr/bin/weston-resizor
      wayland sh # /usr/bin/weston-simple-egl
      wayland sh # /usr/bin/weston-simple-shm
      wayland sh # /usr/bin/weston-simple-touch
      wayland sh # /usr/bin/weston-smoke
      wayland sh # /usr/bin/weston-info
      wayland sh # /usr/bin/weston-terminal
```

Running multimedia with Wayland sink

The GStreamer video sink for Wayland is the waylandsink. To use this video-sink for video playback:

```
target # gst-launch-1.0 playbin uri=file://<path-to-file-name> video-sink=waylandsink
```

Exiting weston

Terminate all Weston clients before exiting Weston. If you have invoked Weston from the serial console, exit Weston by pressing Ctrl-C.

It is also possible to invoke Weston from the native console, exit Weston by using pressing Ctrl-Alt-Backspace.

Using IVI shell feature

The SDK also has support for configuring weston ivi-shell. The default shell that is configured in the SDK is the desktop-shell.

To change the shell to ivi-shell, the user will have to add the following lines into the /etc/weston.ini.

To switch back to the desktop-shell can be done by commenting these lines in the /etc/weston.ini (comments begin with a '#' at the start of line).

```
[core]
shell=ivi-shell.so

[ivi-shell]
ivi-module=ivi-controller.so
ivi-input-module=ivi-input-controller.so
```

After the above configuration is completed, we can restart weston by running the following commands

```
target# /etc/init.d/weston stop
target# /etc/init.d/weston start
```

NOTE: When weston starts with ivi-shell, the default background is black, this is different from the desktop-shell that brings up a window with background.

With ivi-shell configured for weston, wayland client applications use ivi-application protocol to be managed by a central HMI window management. The wayland-ivi-extension provides ivi-controller.so to manage properties of surfaces/layers/screens and it also provides the ivi-input-controller.so to manage the input focus on a surface.

Applications must support the ivi-application protocol to be managed by the HMI central controller with an unique numeric ID.

Some important references to wayland-ivi-extension can be found at the following links: https://at.projects.genivi.org/wiki/display/WIE/01.+Quick+start https://at.projects.genivi.org/wiki/display/PROJ/Wayland+IVI+Extension+Design

Running weston’s sample client applications with ivi-shell

All the sample client applications in the weston package like weston-simple-egl, weston-simple-shm, weston-flower etc also have support for ivi-shell. The SDK includes the application called layer-add-surfaces which is part of the wayland-ivi-extension. This application allows the user to invoke the various functionalities of the ivi-shell and control the applications.

The following is an example sequence of commands and the corresponding effect on the target.

After launching the weston with the ivi-shell, please run the below sequence of commands:

```
target# weston-simple-shm &
```

At this point nothing is displayed on the screen, some additional commands are required.

```
target# layer_add_surfaces 0 1000 2 &
```

This command creates a layer with ID 1000 and to add maximum 2 surfaces to this layer on the screen 0 (which is usually the LCD).
At this point, the user can see weston-simple-shm running on LCD. This also prints the numericID (surfaceID) to which client’s surface is mapped as shown below:

```
 CreateWithDimension: layer ID (1000), Width (1280), Height (800)
 SetVisibility      : layer ID (1000), ILM_TRUE
 layer: 1000 created
 surface                : 10369 created
 SetDestinationRectangle: surface ID (10369), Width (250), Height (250)
 SetSourceRectangle     : surface ID (10369), Width (250), Height (250)
 SetVisibility          : surface ID (10369), ILM_TRUE
 layerAddSurface        : surface ID (10369) is added to layer ID (1000)
```

Here 10369 is the number to which weston-simple-shm application’s surface is mapped.

User can launch one more client application which allows layer_add_surfaces to add second surface to the layer 1000 as shown below.

```
target# weston-flower &
```

User can control the properties of the above surfaces using LayerManagerControl as shown below to set the position, resize, opacity and visibility respectively.

```
target# LayerManagerControl set surface 10369 position 100 100
target# LayerManagerControl set surface 10369 destination region 150 150 300 300
```

```
target# LayerManagerControl set surface 10369 opacity 0.5
target# LayerManagerControl set surface 10369 visibility 1
```

```
target# LayerManagerControl  help
```

The help option prints all possible control operations with the LayerManagerControl binary, please refer to the available options.

IMG PowerVR Demos

The Processor SDK Linux Automotive filesystem comes packaged with example OpenGLES applications. Both DRM and Wayland based applications are packaged as part of the filesystem.

The examples running on **Wayland** can be invoked using the below commands.

```
target # /usr/bin/SGX/demos/Wayland/OGLES2ChameleonMan
target # /usr/bin/SGX/demos/Wayland/OGLES2Navigation
```

The examples running on **DRM/KMS** can be invoked using the below commands.

```
target # /usr/bin/SGX/demos/Raw/OGLES2ChameleonMan
target # /usr/bin/SGX/demos/Raw/OGLES2Navigation
```

After you see the output on the display interface, hit *q* to terminate the application.

Using the PowerVR Tools

Please refer to http://community.imgtec.com/developers/powervr/graphics-sdk/ for additional details on the tools and detailed documentation.

The target file system includes tools such as PVRScope and PVRTrace recorder libraries from Imagination PowerVR SDK to profile and trace SGX activities. In addition, it also includes PVRPerfServerDeveloper toolfor Jacinto6 platform.

PVRTune

PVRPerfServerDeveloper tool can be used along with the PVRTune running on the PC to gather data on the SGX loading and activity threads. You can invoke the tool with the below command:

```
target # /opt/img-powervr-sdk/PVRHub/PVRPerfServer/PVRPerfServerDeveloper
```

PVRTrace

The default filesystem contains helper scripts to obtain the PVRTrace of the graphics application. This trace can then be played back on the PC using the PVRTrace Utility.

To start tracing, use the below commands as reference:

```
target # cp /opt/img-powervr-sdk/PVRHub/Scripts/start_tracing.sh ~/.
target # ./start_tracing.sh <log-filename> <application-to-be-traced>
```

Example:

```
target # ./start_tracing.sh westonapp weston-simple-egl
```

The above command will do the following:

1. Setup the required environment for the tracing
2. Create a directory under the current working directory called pvrtrace
3. Launch the application specified by the user
4. Start tracing the PVR Interactions and record the same to the log-filename

To end the tracing, user can invoke the Ctrl-C and the trace file path will be displayed.

The trace file can then be transferred to a PC and we can visualize the application using the host side PVRTrace utility. Please refer to the link at the beginning of this section for more details.

Testing DSS WB pipeline

Memory to Memory (M2M)

1. Identify the WB pipeline M2M device.

```
# ls /sys/class/video4linux/
Video0 video10 video11
# cat  /sys/class/video4linux/video10/name
omapwb-m2m
```

2.

Look at list of formats supported.

```
# v4l2-ctl -d /dev/video10 --list-formats
ioctl: VIDIOC_ENUM_FMT
        Index       : 0
        Type        : Video Capture Multiplanar
        Pixel Format: 'NV12'
        Name        : Y/CbCr 4:2:0

        Index       : 1
        Type        : Video Capture Multiplanar
        Pixel Format: 'YUYV'
        Name        : YUYV 4:2:2

        Index       : 2
        Type        : Video Capture Multiplanar
        Pixel Format: 'UYVY'
        Name        : UYVY 4:2:2

        Index       : 3
        Type        : Video Capture Multiplanar
        Pixel Format: 'XR24'
        Name        : 32-bit BGRX 8-8-8-8
```

3. Use `v4l2-ctl` command to test the input output. Below command converts from NV12 to YUYV using WB pipeline in M2M mode.

```
# v4l2-ctl -d /dev/video10 --set-fmt-video-out=width=1920,height=1080,pixelformat=NV12  \
--stream-from=test/BigBuckBunny_1920_1080_24fps_100frames.nv12 \
--set-fmt-video=width=1920,height=1080,pixelformat=YUYV \
--stream-to=out/video_test_file.yuyv --stream-mmap=3 --stream-out-mmap=3 --stream-count=70 --stream-poll
```

Capture

1. Identify the WB pipeline capture device.

```
# ls /sys/class/video4linux/
video0 video10 video11
# cat  /sys/class/video4linux/video11/name
omapwb-cap
```

2. Look at list of formats supported.

```
# v4l2-ctl -d /dev/video11 --list-formats
ioctl: VIDIOC_ENUM_FMT
        Index       : 0
        Type        : Video Capture Multiplanar
        Pixel Format: 'NV12'
        Name        : Y/CbCr 4:2:0

        Index       : 1
        Type        : Video Capture Multiplanar
        Pixel Format: 'YUYV'
        Name        : YUYV 4:2:2

        Index       : 2
        Type        : Video Capture Multiplanar
        Pixel Format: 'UYVY'
        Name        : UYVY 4:2:2

        Index       : 3
        Type        : Video Capture Multiplanar
        Pixel Format: 'XR24'
        Name        : 32-bit BGRX 8-8-8-8
```

3. Use `v4l2-ctl` command to test the input output. Below command converts from NV12 to YUYV using WB pipeline in M2M mode.

```
# v4l2-ctl -d /dev/video11 -i 0 --set-fmt-video=pixelformat=NV12 \
--stream-to=/test/video_test_file.yuv --stream-mmap=6 --stream-count=10 --stream-poll
Video input set to 0 (CRTC#0 - LCD1: ok)
<<<<<<<<< 7.84 fps
<
# v4l2-ctl -d /dev/video11 -i 1 --set-fmt-video=pixelformat=NV12 --stream-to=/test/video
_test_file.yuv --stream-mmap=6 --stream-count=10 --stream-poll
Video input set to 1 (CRTC#1 - DIGIT/TV: ok)
<<<<<<<<<< 8.65 fps
```

Running aplay and arecord application

Audio playback is supported on HDMI and via headset. By default, the audio playback takes place on the HDMI. To listen to audio via the HDMI, run the aplay application

```
target #  aplay <path_to_example_audio>.wav
```

If playback is required via headset, please make sure that the following amixer settings are done for the corresponding card (check the card no. by running the command cat /proc/asound/cards, assuming the card 1 is for headset here):

```
target #  amixer cset -c 1 name='Headset Left Playback' 1
target #  amixer cset -c 1 name='Headset Right Playback' 1
target #  amixer cset -c 1 name='Headset Playback Volume' 12
target #  amixer cset -c 1 name='DL1 PDM Switch' 1
target #  amixer cset -c 1 name='Sidetone Mixer Playback' 1
target #  amixer cset -c 1 name='SDT DL Volume' 120
target #  amixer cset -c 1 name='DL1 Mixer Multimedia' 1
target #  amixer cset -c 1 name='DL1 Media Playback Volume' 110
target #  amixer sset -c 1 'Analog Left',0 'Aux/FM Left'
target #  amixer sset -c 1 'Analog Right',0 'Aux/FM Right'
target #  amixer sset -c 1 'Aux FM',0 7
target #  amixer sset -c 1 'AUDUL Media',0 149
target #  amixer sset -c 1 'Capture',0 4
target #  amixer sset -c 1 MUX_UL00,0 AMic0
target #  amixer sset -c 1 MUX_UL01,0 AMic1
target #  amixer sset -c 1 'AMIC UL',0 120
```

Once these settings are done, one could do playback via headset using aplay by the following command:

```
target #  aplay -Dplughw:1,0 <path_to_example_audio>.wav
```

To playback/record on the evm via headset/mic, please make sure the following amixer settings are done:

- For playback via headset, enter the following at prompt **target#**

```
amixer sset 'Left DAC Mux',0 'DAC_L2'
amixer sset 'Right DAC Mux',0 'DAC_R2'
amixer cset name='HP Playback Switch' On
amixer cset name='Line Playback Switch' Off
amixer cset name='PCM Playback Volume' 127
```

Once these settings are successful, use aplay application for playback:

```
target #  aplay <path_to_example_audio>.wav
```

- For recording via Mic In

```
amixer cset name='Left PGA Mixer Mic3L Switch' On
amixer cset name='Right PGA Mixer Mic3L Switch' On
amixer cset name='Left PGA Mixer Line1L Switch' off
amixer cset name='Right PGA Mixer Line1R Switch' off
amixer cset name='PGA Capture Switch' on
amixer cset name='PGA Capture Volume' 6
```

Once these settings are successful, use arecord to record

```
target #  arecord -r 44.1 > <path_to_example_audio>.wav
```

To playback/record on the evm via Line Out/ Line In, please make sure the following amixer settings are done:

- For playback via Line Out, enter the following at prompt **target#**

```
amixer cset name='Line Playback Switch' On
amixer cset name='PCM Playback Volume' 127
```

Once these settings are successful, use aplay application for playback, e.g.,

```
target #  aplay <path_to_example_audio>.wav
```

- For recording via Line In

```
amixer cset name='Left PGA Mixer Mic2L Switch' On
amixer cset name='Right PGA Mixer Mic2L Switch' On
amixer cset name='Left PGA Mixer Line1L Switch' On
amixer cset name='Right PGA Mixer Line1R Switch' On
amixer cset name='PGA Capture Switch' On
amixer cset name='PGA Capture Volume' 50
```

Once these settings are successful, use arecord to record, e.g.,

```
target #  arecord -r 44100 -c 2 -f S16_LE <path_to_example_audio>.wav
```

Running GC320 application

GC320 is a 2D Graphic accelerator in DRA7xx. This IP can be utilized for the usecases like alpha blending, overlaying, bitBlit, color conversion, scaling, rotation etc.

SDK provides two sample GC320 testcases in the root filesystem. Before running the test, the gc320 kernel module needs to be inserted into system.

On 1.5GB RAM configuration

```
target# insmod /lib/modules/4.4.xx-gyyyyyyyy/extra/galcore.ko baseAddress=0x80000000 physSize=0x60000000
```

On 2GB RAM configuration

```
 target# insmod /lib/modules/4.4.xx-gyyyyyyyy/extra/galcore.ko baseAddress=0x80000000 physSize=0x80000000
```

Now follow these instrctions to execute the applications

```
target# cd /usr/bin/GC320/tests/unit_test
target# export LD_LIBRARY_PATH=$PWD
target# ./runtest.sh
```

This script executes two sample unit test cases of filling rectangles and GC320 rendered results will be stored in .bmp file in a directory "result" under /usr/bin/GC320/tests/unit_test.

Note: To run all GC320 unit testcases, clone ti-gc320-test package from git://git.ti.com/graphics/ti-gc320-test.git : branch:ti-5.0.11.p7, rebuild test application, libraries etc and install the package on target.

Running viddec3test application

viddec3test is a demo application for decoder/video playback using hardware accelerators. The application currently runs on the kms display. The application requires the connector information for display. One can get the information of the display connected to the board by running the *modetest* application in the filesystem, as described above. To execute the application "modetest" make sure the display is connected to the board.

Running a decode on a display

To run a hardware decode on a display connected to the board, execute the following command:

```
 target #  viddec3test -s <connector_id>:<display resolution> filename --fps 30

e.g.: target # viddec3test -s 4:1920x1080 file.h264 --fps 30
```

Running single decode on dual displays

To run the output of a single decode on the dual displays. Please make sure both the displays are connected and get the information about the connectors and the resolution associated with it for both the displays from the *modetest* application.

```
 target #  viddec3test -s <connector_id_1>:<display resolution> -s <connector_id_2>:<display resolution> filename --fps 30
    e.g.: target # viddec3test -s 4:1920x1080 -s 12:1024x768 file.h264 --fps 30
```

Running dual decode on dual displays

One can also run a dual decode and display their output on two different displays. Please make sure both the displays are connected and get the information about the connectors and the resolution associated with it for both the displays from the *modetest* application.

```
 target # viddec3test -s <connector_id_1>:<display resolution> filename1 -s <connector_id_2>:<display resolution> filename2
   e.g.: target # viddec3test -s 4:1920x1080 file1.h264 -- -s 12:1024x768 file2.h264
```

Running a gstreamer pipeline

GStreamer v1.14.4 is supported in Processor SDK | in Processor SDK Linux Automotive 6.00

Gstreamer pipelines can also run from command line. In order to do so, exit Weston by pressing Ctrl-Alt-Backspace from the keyboard which connects to the EVM. Then, if the LCD screen stays in "Please wait...", press Ctrl-Alt-F1 to go to the command line on LCD console. After that, the command line can be used from serial console, SSH console, or LCD console.

One can run an audio video file using the gstreamer playbin from the console. Currently, the supported Audio/video sink is kmssink, waylandsink and alsassink.

```
kmssink:
 target #  gst-launch playbin2 uri=file:///<path_to_file> video-sink=kmssink audio-sink="alsasink device=hw:1,0"
```

```
waylandsink:
  1. refer #Wayland/Weston to start the weston
  2. target # gst-launch playbin2 uri=file:///<path_to_file> video-sink=waylandsink audio-sink="alsasink device=hw:1,0"
```

```
dri2videosink:
  1. refer #X_Server to start the X11
  2. target # gst-launch playbin2 uri=file:///<path_to_file> video-sink=dri2videosink audio-sink="alsasink device=hw:1,0"
```

|

```
kmssink:
  target #  gst-launch-1.0 playbin uri=file:///<path_to_file> video-sink=kmssink audio-sink=alsasink
```

```
waylandsink:
  1. refer #Wayland/Weston to start the weston
  2. target #  gst-launch-1.0 playbin uri=file:///<path_to_file> video-sink=waylandsink audio-sink=alsasink
```

The following pipelines show how to use vpe for scaling and color space conversion.

```
  1. Decode-> Scale->Display
     target # gst-launch-1.0 -v filesrc location=example_h264.mp4 ! qtdemux ! h264parse ! \
 ducatih264dec ! vpe ! 'video/x-raw, format=(string)NV12, width=(int)720, height=(int)480' ! kmssink
```

```
  2. Color space conversion:
     target # gst-launch-1.0 -v videotestsrc ! 'video/x-raw, format=(string)YUY2, width= \
 (int)1280, height=(int)720' ! vpe ! 'video/x-raw, format=(string)NV12, width=(int)720, height=(int)480' \
 ! kmssink
```

```
Note:
  1. While using playbin for playing the stream, vpe plugin is automatically picked up. However vpe cannot be used with playbin for scaling.
For utilizing scaling capabilities of vpe, using manual pipeline given above is recommended.
  2. Waylandsink and Kmssink uses the cropping metadata set on buffers and does not require vpe plugin for cropping
```

The following pipelines show how to use v4l2src and ducatimpeg4enc elements to capture video from VIP and encode captured video respectively.

```
Capture and Display Fullscreen
  target #  gst-launch-1.0 v4l2src device=/dev/video1 num-buffers=1000 io-mode=4 ! 'video/x-raw, \
format=(string)YUY2, width=(int)1280, height=(int)720' ! vpe num-input-buffers=8 ! queue ! kmssink
```

```
Note:
 The following pipelines can also be used for NV12 capture-display usecase.
 Dmabuf is allocated by v4l2src if io-mode=4 and by kmssink and imported by v4l2src if io-mode=5
 target # gst-launch-1.0 v4l2src device=/dev/video1 num-buffers=1000 io-mode=4 ! 'video/x-raw, \
format=(string)NV12, width=(int)1280, height=(int)720' ! kmssink
 target # gst-launch-1.0 v4l2src device=/dev/video1 num-buffers=1000 io-mode=5 ! 'video/x-raw, \
format=(string)NV12, width=(int)1280, height=(int)720' ! kmssink
```

```
Capture and Display to a window in wayland
  1. refer #Wayland/Weston to start the weston
  2. target #  gst-launch-1.0 v4l2src device=/dev/video1 num-buffers=1000 io-mode=4 ! 'video/x-raw, \
format=(string)YUY2, width=(int)1280, height=(int)720' ! vpe num-input-buffers=8 ! queue ! waylandsink
```

```
Note:
 The following pipelines can also be used for NV12 capture-display usecase. Dmabuf is allocated by v4l2src
 if io-mode=4 and by waylandsink and imported by v4l2src if io-mode=5.
 Waylandsink supports both shm and drm. A new property use-drm is added to specify drm allocator based bufferpool to be used.
 When using ducati or vpe plugins, use-drm is set in caps as true.
 target # gst-launch-1.0 v4l2src device=/dev/video1 num-buffers=1000 io-mode=4 ! 'video/x-raw, \
format=(string)NV12, width=(int)1280, height=(int)720' ! waylandsink use-drm=true
 target # gst-launch-1.0 v4l2src device=/dev/video1 num-buffers=1000 io-mode=5 ! 'video/x-raw, \
format=(string)NV12, width=(int)1280, height=(int)720' ! waylandsink use-drm=true
```

```
Capture and Encode into a MP4 file.
  target #  gst-launch-1.0 -e v4l2src device=/dev/video1 num-buffers=1000 io-mode=4 ! 'video/x-raw, \
format=(string)YUY2, width=(int)1280, height=(int)720, framerate=(fraction)30/1' ! vpe num-input-buffers=8 ! \
queue ! ducatimpeg4enc bitrate=4000 ! queue ! mpeg4videoparse ! qtmux ! filesink location=x.mp4
```

```
Note:
  The following pipeline can be used in usecases where vpe processing is not required.
  target # gst-launch-1.0 -e v4l2src device=/dev/video1 num-buffers=1000 io-mode=5 ! 'video/x-raw, \
format=(string)NV12, width=(int)1280, height=(int)720, framerate=(fraction)30/1' ! ducatimpeg4enc bitrate=4000 ! \
queue ! mpeg4videoparse ! qtmux ! filesink location=x.mp4
```

```
Capture and Encode and Display in parallel.
  target #  gst-launch-1.0 -e v4l2src device=/dev/video1 num-buffers=1000 io-mode=4 ! 'video/x-raw, \
format=(string)YUY2, width=(int)1280, height=(int)720, framerate=(fraction)30/1' ! vpe num-input-buffers=8 ! tee name=t  ! \
 queue ! ducatimpeg4enc bitrate=4000 ! queue ! mpeg4videoparse ! qtmux ! filesink location=x.mp4 t. ! queue ! kmssink
```

|

Running VIP/VPE/CAL application

Video Input Port

Video Input Port is used to capture video frames from BT56/ BT601 Camera. Currently the VIP driver supports following features.

For more information on VIP driver and other features, please refer to **http://processors.wiki.ti.com/index.php/Processor_SDK_VIP_Driver**

- Standard V4L2 capture driver
- Supports single planar buffers
- Supports MMAP buffering method
- Supports DMABUF based buffering method
- Supports V4L2 endpoint standard way of specifying camera nodes
- Supports captures upto 60FPS
- Multi instance capture - All slices, ports supported
- Capture from a YUYV camera(8bit)
- NV12 capture format

Camera Adapter Layer

Camera Adapter Layer is used to capture video from CSI Camera. Currently the CAL driver supports following features.

- Standard V4L2 capture driver
- Supports single planar buffers
- Supports MMAP buffering method
- Supports DMABUF based buffering method
- Supports V4L2 endpoint standard way of specifying camera nodes (CSI bindings)
- Multi instance capture - 4data lane phy0 + 2data lane phy1

Supported cameras

Camera Adapter Layer is used to capture video from CSI Camera. Currently the CAL driver supports following features. Processor SDK Linux Automotive release supports following sensors/cameras/video inputs:-

- OV10633 sensor - YUYV sensor connected on J6 EVM
- OV10635 sensor - YUYV sensor on Vision board
- OV10635 sensor - YUYV sensor connected through LVDS
- TVP5158 decoder - Support for decoding single channel analog video
- OV10640/OV490 - 720p CSI2 raw camera connected to OV490 ISP in YUYV format

Processor SDK supports following sensors/cameras

- mt9t111 camera sensor

```
Note: This release of PSDKLA with kernel 4.19, supports only OV10633 sensor.
      OV10635 sensor capture on Vision Board and LVDS Capture are supported with VisionSDK v3.8.
      FPDLink serializer and deserializer support is also not available with this release.
      JAMR board support is not validated with this release
      CSI2 capture not validated with this release
```

Running dmabuftest

dmabuftest is a user space application which demonstrates capture display loopback. It can support multiple captures at the same time

Video buffers are allocated by libdrm and they are shared to VIP through dmabuf.

It interfaces with the VIP through standard v4l2 ioctls.

Filesystem from release has dmabuftest app preinstalled.

To capture and display on the LCD screen, run following command

```
target# dmabuftest -s 4:800x480 -d /dev/video1 -c 1280x720@YUYV
```

```
target# dmabuftest -s 16:800x480 -d /dev/video1 -c 1280x720@YUYV
```

To capture and display on the HDMI display, run following command

```
target# dmabuftest -s 32:1920x1200 -d /dev/video1 -c 1280x720@YUYV
```

```
target# dmabuftest -s 4:1920x1080 -d /dev/video1 -c 1280x720@YUYV
```

To capture video in NV12 format, run following command

```
target# dmabuftest -s 32:1920x1200 -d /dev/video1 -c 1280x720@NV12
```

```
target# dmabuftest -s 16:800x480 -d /dev/video1 -c 1280x720@NV12
```

To capture and display on KMScube backend (Video on a rotating cube), run following command

```
target# dmabuftest --kmscube --fov 20 -d /dev/video1 -c 1280x720@YUYV
This feature is currently not supported
```

To capture and display on wayland backend (Video in a wayland client window), run following command

```
target# dmabuftest -w 640x480 --pos 100x400 /dev/video1 -c 1280x720@YUYV
```

Capturing from OV10633 onboard camera

Linux kernel driver for OV1063x cameras support OV10633 sensor.

Video capture can be verified from the OV10633 sensor as follows

- Connect OV10633 sensor to the Leopard Imaging port on the J6 EVM
- Reboot the board and enable i2c2 as given above
- I2C device on Bus 2 slave address 0x37 should be probed successfully
- VIP should register a V4L2 video device (e.g. /dev/video1) using this i2c device
- Run dmabuftest with '1280x720@YUYV' as capture format

Capturing from OV10635 Vision board camera

Linux kernel driver for OV1063x cameras support OV10635 sensor.

Video capture can be verified from the OV10635 sensor as follows

- Connect OV10635 sensor to the OVcam port on the Vision board
- Change the SW3 switch setting on Vision board as SW3[1-8] = 01010101
- Reboot the board and enable i2c2 as given above
- I2C device on Bus 2 slave address 0x30 should be probed successfully
- VIP should register a V4L2 video device (e.g. /dev/video1) using this i2c device
- Run dmabuftest with '1280x720@YUYV' as capture format

Capturing through TVP decoder

Linux kernel supports TVP5158 NTSC/PAL decoder.

TVP5158 decoder is a TI chip which can decode upto 4 channels of NTSC/PAL analog video and multiplex it.

Video capture from 1 channel TVP5158 can be verified as follows.

- Connect analog camera to the Vin1 port of the JAMR3 board
- Change the SW2 switch setting on JAMR board as SW2[1-2] = [OFF, ON] - This is to select i2c4 for the IO expander
- Reboot the board and enable i2c2 as given above
- I2C device on Bus 2 slave address 0x58 should be probed successfully
- VIP should register a V4L2 video device (e.g. /dev/video1) using this i2c device
- Run dmabuftest with capture format of the analog camera (e.g. '720x240@YUYV')

Capturing through LVDS camera

LVDS camera is also a camera connected through a serializer and deserializer

Linux kernel has driver for FPDlink serializers and deserializers

For interfacing every LVDS camera with J6, an I2C slave for ser, deser and camera is needed. By default, all of the device tree nodes are disabled.

Following table shows mapping between all LVDS cameras on multi deserializer duaghter card for Vision Board.

```
LVDS camera    Camera address alias    Serializer address alias    Derializer address    VIP port
```

```
cam1           0x38                    0x74                        0x60                  Vin1a(VIP1 slice0 port A)
cam2           0x39                    0x75                        0x64                  Vin2a(VIP1 slice1 port A)
cam3           0x3A                    0x76                        0x68                  Vin3a(VIP2 slice0 port A)
cam4           0x3B                    0x77                        0x6C                  Vin5a(VIP3 slice0 port A)
cam5           0x3C                    0x78                        0x61                  Vin4b(VIP2 slice1 port B)
cam6           0x3D                    0x79                        0x69                  Vin6a(VIP3 slice1 port A)
```

Video capture from LVDS camera can be verified as follows.

- Connect a LVDS camera to cam1/2/3/4 port of Multides board.
- Change the SW3 switch setting on Vision board as SW3[1-8] = 00100101
- I2C device on Bus 2 slave address (e.g. 0x38 for cam1) should be probed successfully
- VIP should register a V4L2 video device (e.g. /dev/video1) using this i2c device
- Run dmabuftest with '1280x720@YUYV' as capture format

Capturing through OV10640/OV490 CSI camera/ISP

Linux kernel supports CSI capture from OV10640 RAW camera and OV490 ISP.

CAL works on the CSI2 protocol and supports both raw and YUYV capture. It is verified with the OV10640 raw camera and OV490 ISP. TI-EVM has support for capturing via two CSI phys. - phy0 (4data lanes) and phy1 (2data lanes)

Video capture from OV490 can be verified as follows.

- Connect OV10640 camera to the OV490 board
- Connect the OV490 board to the TI-EVM via the CSI2 dual 490 adaptor board
- I2C device on Bus 4 slave address 0x24 should be probed successfully
- VIP should register a V4L2 video device (e.g. /dev/video1) using this i2c device
- Run dmabuftest with capture format of the analog camera (e.g. '1280x720@YUYV')

Video Processing Engine(VPE)

VPE supports Scalar, Colour Space Conversion and Deinterlace.It uses V4L2 mem2mem API.

Supported Input formats: nv12, yuyv, uyvy

Supported Output formats: nv12, yuyv, uyvy, rgb24, bgr24, argb24, abgr24

Not Supported formats: yuv444, yvyu, vyuy, nv16, nv61, nv21

File to File

```
test-v4l2-m2m

Usage:
<SRCfilename> <SRCWidth> <SRCHeight> <SRCFormat> <DSTfilename> <DSTWidth> <DSTHeight> <DSTformat> <interlace> <translen>
```

Note:

<interlace> : set 1, If input is interlaced and want deinterlaced(progressive) output. output height should be twice of input height.

Deinterlace(DI):-

```
target# test-v4l2-m2m /dev/video0 frame-176-144-nv12-inp.yuv 176 144 nv12 progressive_output.nv12 176 288 nv12 1 1
```

Scalar(SC):-

```
target# test-v4l2-m2m /dev/video0 frame-176-144-nv12-inp.yuv 176 144 nv12 frame-1920-1080-nv12-out.nv12 1920 1080 nv12 0 1
```

Colour Space Conversion(CSC):-

```
target# test-v4l2-m2m /dev/video0 frame-720-240-yuyv-inp.yuv 720 240 yuyv frame-720-240-argb32-out.argb32 720 240 argb32 0 1
```

SC+CSC+DI:-

```
target# test-v4l2-m2m /dev/video0 frame-720-240-yuyv-inp.yuv 720 240 yuyv frame-1920-1080-rgb24-dei-out.rgb24 1920 1080 rgb24 1 1
```

File to Display

```
filevpedisplay

Usage:
<src_filename> <src_w> <src_h> <src_format> <dst_w> <dst_h> <dst_format> <top> <left> <w> <h> <inter> <trans> -s <conn_id>:<mode>
```

Input without crop:

```
target# filevpedisplay frame-176-144-nv12-inp.yuv 176 144 nv12 800 480 yuyv 0 0 176 144 0 1 -s 4:800x480
```

Input with crop:

```
target# filevpedisplay frame-176-144-nv12-inp.yuv 176 144 nv12 800 480 yuyv 16 32 128 128 0 1 -s 4:800x480
```

Input without crop:

```
target# filevpedisplay frame-176-144-nv12-inp.yuv 176 144 nv12 800 480 yuyv 0 0 176 144 0 1 -s 16:800x480
```

Input with crop:

```
target# filevpedisplay frame-176-144-nv12-inp.yuv 176 144 nv12 800 480 yuyv 16 32 128 128 0 1 -s 4:1280x720
```

VIP-VPE-Display

Camera captures the frames, which are processed by VPE(SC, CSC, Dei) then displays on LCD/HDMI.

```
capturevpedisplay

Usage:
<src_w> <src_h> <src_format> <dst_w> <dst_h> <dst_format> <inter> <trans> -s <conn_id>:<mode>
```

```
target# capturevpedisplay 640 480 yuyv 320 240 uyvy 0 1 -s 4:640x480
```

```
target# capturevpedisplay 640 480 yuyv 320 240 uyvy 0 1 -s 4:1280x720
```

Running DSS application

DSS applications are omapdrm based. These will demonstrate the clone mode, extended mode, overlay window, z-order and alpha blending features. To demonstrate clone and extended mode, HDMI display must be connected to board. Application requires the supported mode information of connected displays and plane ids. One can get these information by running the *modetest* application in the filesystem.

```
 target #  modetest
```

DSS application is omapdrm based. This will demonstrate the z-order and alpha blending features. HDMI display must be connected to board. Application requires the supported mode information of connected display and plane ids. One can get these information by running the *modetest* application in the filesystem.

```
 target #  modetest
```

Running drmclone application

This displays same test pattern on both LCD and HDMI (clone). Overlay window also displayed on LCD. To test clone mode, execute the following command:

```
 target #  drmclone -l <lcd_w>x<lcd_h> -p <plane_w>x<plane_h>:<x>+<y> -h <hdmi_w>x<hdmi_h>
```

```
e.g.: target # drmclone -l 1280x800 -p 320x240:0+0 -h 640x480
```

We can change position of overlay window by changing x+y values. eg. 240+120 will show @ center

Running drmextended application

This displays different test pattern on LCD and HDMI. Overlay window also displayed on LCD. To test extended mode, execute the following command:

```
 target # drmextended -l <lcd_w>x<lcd_h> -p <plane_w>x<plane_h>:<x>+<y> -h <hdmi_w>x<hdmi_h>
```

```
e.g.: target # drmextended -l 1280x800 -p 320x240:0+0 -h 640x480
```

Running drmzalpha application

Z-order:

It determines, which overlay window appears on top of the other.

Range: 0 to 3

  lowest value for bottom

  highest value for top

Alpha Blend:

It determines transparency level of image as a result of both global alpha & pre multiplied alpha value.

Global alpha range: 0 to 255

  0 - fully transparent

  127 - semi transparent

255 - fully opaque

Pre multipled alpha value: 0 or 1

0 - source is not premultiply with alpha

1 - source is premultiply with alpha

To test drmzalpha, execute the following command:

```
target # drmzalpha -s <crtc_w>x<crtc_h> -w <plane1_id>:<z_val>:<glo_alpha>:<pre_mul_alpha> -w <plane2_id>:<z_val>:<glo_alpha>:<pre_mul_alpha>
```

```
e.g.: target # drmzalpha -s 1280x800 -w 19:1:255:1 -w 20:2:255:1
```

To test drmzalpha, execute the following command:

```
target # drmzalpha -s <crtc_w>x<crtc_h> -w <plane1_id>:<z_val>:<glo_alpha>:<pre_mul_alpha> -w <plane2_id>:<z_val>:<glo_alpha>:<pre_mul_alpha>

e.g.: target # drmzalpha -s 640x480 -w 15:1:255:1 -w 16:2:255:1
```

Testing with FPDLink Display setup

NOTE! Support for FPDLink display is available upto K4.4 releases. PSDKLA6.0x release with k4.19 doesn't have support for FPDLInk. Check the release notes to see whether FPDLink is supported on the Processor SDK Linux Automotive release you are using.

For information on debugging FPDLink integration, please refer to Debugging FPDLink integration

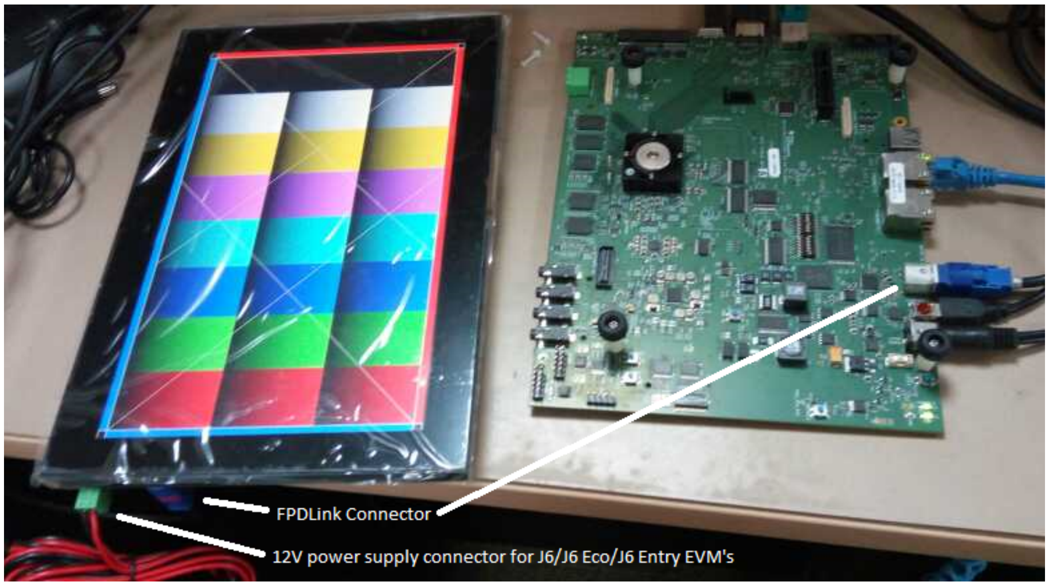
Current H/W setup

FPDLink display is currently supported with Spectrum Digital FPDLink display part number 703840-0001. This display includes a 1280x800 AUO LCD panel with Goodix touch screen connected over a DS90UB924Q1 deserializer.

To validate FPDLink with the current HW setup, below hardware is required.

- DRA7xx EVM + 12V supply for the EVM.
- FPDLink Cable between DRA7xx and FPDLink display
- 12 V power supply for the FPDLink display if using a J6/J6 Eco/J6 Entry EVM. J6 Plus EVM supplies power to the display over FPDLink. Power supply for display is not required in this case.

The picture below shows the overall setup.



Kernel Config modifications are not necessary as AUO panel support and fpdlink support are built into the kernel.

To test the FPDLink display,

1. Use the device tree `dra7-evm-fpd-auo-g101evn01.0.dtb` to boot.
2. Add `omapdrm.num_crtc=2` to the kernel boot arguments. The above device tree will enable both HDMI and FPDlink LCD.
3. Power on the EVM and the check the modetest output. You should see two connectors now, one for HDMI and another for FPDLink.

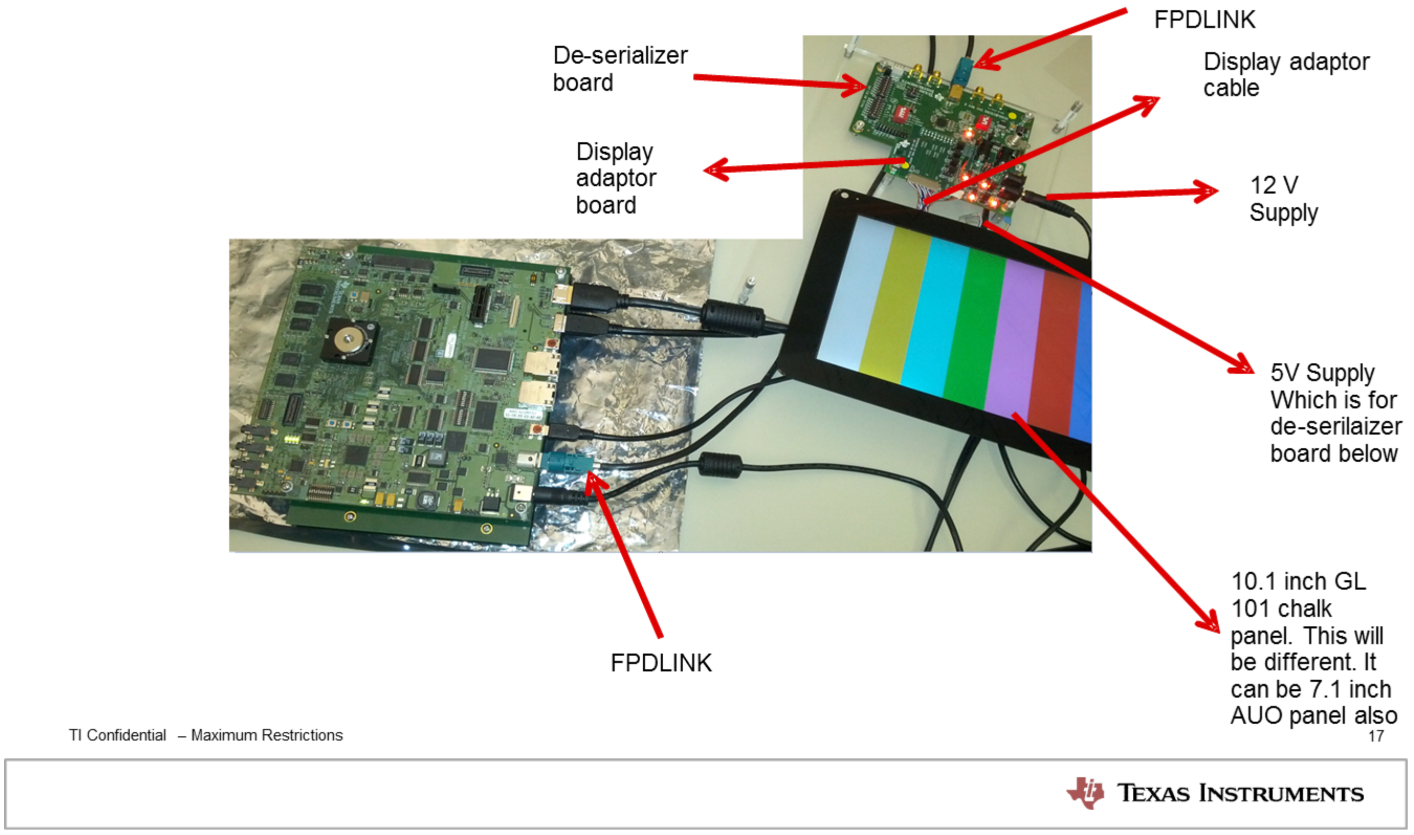
Legacy H/W setup

Please note that support for the below FPDLink hardware will be deprecated with the next release. This is due to availability of the single board FPDLink display listed above.

To validate FPDLink with the legacy HW setup, below hardware is required.

- DRA7xx EVM + 12V supply for the EVM.
- FPDLink Cable between DRA7xx and De-serilzer board (DS90UB928Q).
- 5V power supply for De-serializer board.
- LCD Adapter board (DS90UH928Q) that sits on De-serializer board.
- LCD Adapter cable which is between LCD panel and the Adapter board.
- 12V power supply for LCD Adapter board.
- The actual LCD panel (LG101(10.1in) or AUO (7.1 in))

The picture below shows the overall setup.


FPDLINK
De-serializer board
Display adaptor cable
Display adaptor board
12 V Supply
5V Supply Which is for de-serilaizer board below
10.1 inch GL 101 chalk panel. This will be different. It can be 7.1 inch AUO panel also
FPDLINK
TI Confidential – Maximum Restrictions
17
Texas Instruments

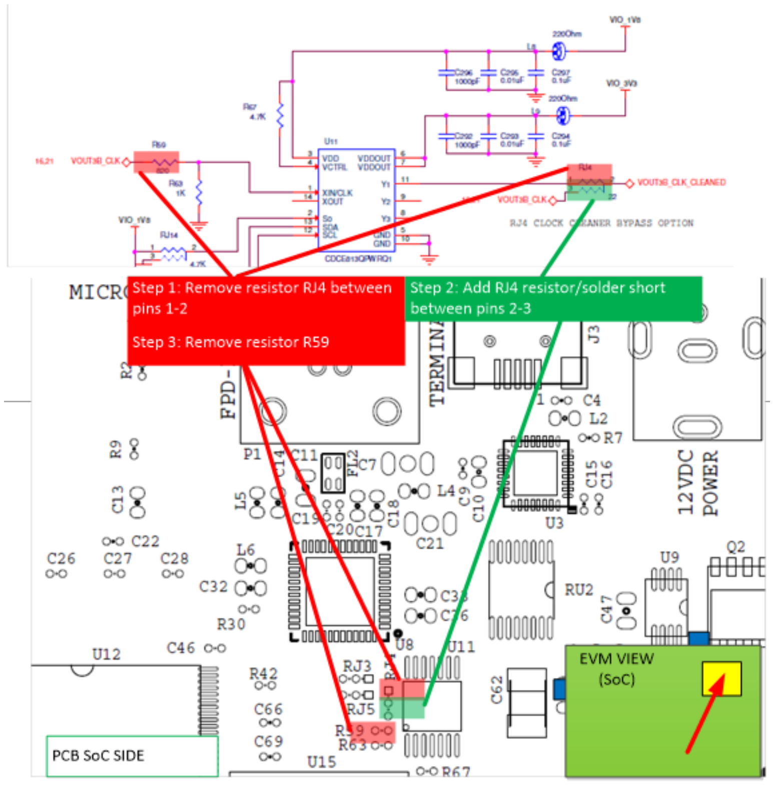
Kernel Config is not necessary as the supported panels and fpdlink are built into the kernel.

To test the FPDLink display,

1. Use the device tree `dra7-evm-fpd-lg.dtb` to boot.
2. Add `omapdrm.num_crtc=2` to the kernel boot arguments. The above device tree will enable both HDMI and FPDlink LCD.
3. Power on the EVM and the check the modetest output. You should see two connectors now, one for HDMI and another for FPDLink.

HW Modifications required

With the Rev B J6 Plus EVM's, a board modification is required to supply the pixel clock to the FPDLink connector. The modification required is shown in the below image.


Step 1: Remove resistor RJ4 between pins 1-2
Step 3: Remove resistor R59
Step 2: Add RJ4 resistor/solder short between pins 2-3
RJ4 CLOCK CLEANER BYPASS OPTION
VOUT3B_CLK_CLEANED
EVM VIEW (SoC)
PCB SoC SIDE
12VDC POWER

}}

Gsttestplayer

gsttestplayer is a gstreamer test application useful for testing some features not testable with gst-launch-1.0 such as:

1. Seek - Seeking to random points in a stream
2. Trick play - Playback at different speeds (fast forward, rewind)
3. Pause, Resume
4. Playing multiple streams simultaneously in the same process, in a loop or one after another.

Running gsttestplayer

Command line options:

```
target # gsttestplayer -h
      Usage: gsttestplayer <options>
              -s <sinkname>    Specify the video sink name to be used, default: kmssink
              -n               Do not use VPE, implies no scaling
              -r <width>x<height> Resize the output to widthxheight, no scaling if left blank
              -a               Play with no A/V Sync
              -c <cmds file>   Non-interactive mode, reading commands from <cmds file>
              --help-gst                        Show GStreamer Options
```

Example: To use waylandsink and resize the output video to 800x400.

```
target # gsttestplayer -s waylandsink -r 800x400
```

In normal mode, when -c option is not used, the application enters an command prompt at which the user enter various commands. Type "help" to print out the list of possible commands:

```
target # gsttestplayer -s waylandsink -r 800x400
      Scaling output to 800x400
      Using videosink=waylandsink
      <Enter ip> help
      Commands available:
      start  <instance num> <filename/capture device>
      stop   <instance num>
      pause  <instance num>
      resume <instance num>
      seek   <instance num> <seek to time in seconds> <optional: playback speed>
      sleep  <sleep time in seconds>
      msleep <sleep time in milliseconds>
      rewind <line number>
      exit
      <Enter ip>
```

Example commands:

```
start 0 KMS_MPEG4_D1.MP4  # Start playing the file "KMS_MPEG4_D1.MP4", using instance 0.
start 1 NTSC_h264.mp4     # Start playing the file "NTSC_h264.mp4" (simultaneously) using instance 1.
```

```
stop 0                  # Stop playback of instance 0.
seek 0 0 2              # Seek to "0"th second mark of the stream playing in instance 0,
                        #  and start playing back at speed 2x.
seek 0 300 -1           # Seek to "300"th second mark of the stream playing in instance 0,
                        #  and start playing back reserve at speed 1x.
start 2 /dev/video1     # Start capturing from /dev/video1 using the v4l2src plugin
```

All these commands could be put into a text file and given as input to gsttestplayer with the "-c" option. In this case, gsttestplayer runs non-interactively, reading commands from the text file one line after another. The commands *sleep* and *rewind* are useful for this mode, to introduce delays or to create a loop respectively.

Notes:

1. This application plays video only. Audio path is not used.
2. The input filename should have the correct file extension to indicate the type of file. The supported extensions are "mp4", "avi", "ts", "asf" & "wmv".
3. The input filename should contain the string "264", "mpeg2", "mpeg4" or "vc1"/"wmv" to indicate which video codec should be used for decoding - H.264, MPEG-2, MPEG-4 or Windows Media Video.
4. If the input filename is a video device which matches /dev/videoX pattern, v4l2src plugin would be used for video capture instead of playback.
5. Decode and capture can be run in parallel depending on the sink being used.

Running IPC examples

Processor SDK Linux Automotive includes IPC examples are part of the target filesystem.

User space sample application

MessageQ is the user space API provided for IPC. The sample application for MessageQ consists of an application "MessageQApp" running on the A15 and corresponding binaries running on the remotecore. The below table shows the paths under which the remotecore binaries can be found on the target filesystem. To ensure that these binaries are loaded by the kernel, please symbolic link them to the location shown in the below table.

Core	Binary path on target relative to /lib/firmware	Binary should be symlinked to
DSP2	./ipc/ti_platforms_evmDRA7XX_dsp2/messageq_single.xe66	/lib/firmware/dra7-dsp2-fw.xe66
IPU2	./ipc/ti_platforms_evmDRA7XX_ipu2/messageq_single.xem4	/lib/firmware/dra7-ipu2-fw.xem4
IPU1	./ipc/ti_platforms_evmDRA7XX_ipu1/messageq_single.xem4	/lib/firmware/dra7-ipu1-fw.xem4
DSP1	./ipc/ti_platforms_evmDRA7XX_dsp1/messageq_single.xe66	/lib/firmware/dra7-dsp1-fw.xe66

Boot the target and ensure that the lad daemon is running. Use the `ps` command to check if the lad daemon is already running. If not, please start the lad daemon.

```
target # /usr/bin/lad_dra7xx
```

Start the MessageQApp

```
target # /usr/bin/MessageQApp <number of messages> <core id>
```

The core id to be used is 1,2,3,4 for IPU2,IPU1,DSP2 and DSP1 respectively.

```
target # MessageQApp 10 3
Using numLoops: 10; procId : 3
Entered MessageQApp_execute
Local MessageQId: 0x80
Remote queueId  [0x30080]
Exchanging 10 messages with remote processor DSP2...
MessageQ_get #1 Msg = 0xb6400468
MessageQ_get #2 Msg = 0xb6400468
MessageQ_get #3 Msg = 0xb6400468
MessageQ_get #4 Msg = 0xb6400468
MessageQ_get #5 Msg = 0xb6400468
MessageQ_get #6 Msg = 0xb6400468
MessageQ_get #7 Msg = 0xb6400468
MessageQ_get #8 Msg = 0xb6400468
MessageQ_get #9 Msg = 0xb6400468
MessageQ_get #10 Msg = 0xb6400468
Exchanged 10 messages with remote processor DSP2
Sample application successfully completed!
Leaving MessageQApp_execute
```

RPMsg client sample application

RPMsg is the kernel space IPC and the building block for the user space MessageQ IPC API. The below wiki page illustrates how to build and run an rpmsg Linux kernel space client to communicate with a slave processor (e.g. DSP, IPU, etc) using IPC's RPMsg module.

RPMsg_Kernel_Client_Application (http://processors.wiki.ti.com/index.php/RPMsg_Kernel_Client_Application)

Running basic Wifi tests

To run this test, you would need to have the Wilink COM module connected to the EVM.

Check if the wlan0 interface is showing up:

```
target # ifconfig -a
```

Bring up the wlan0 inteface:

```
target # ifconfig wlan0 up
```

Search for the available Wifi networks:

```
target # iw wlan0 scan | grep -i ssid
```

Running basic Bluetooth tests

To run this test, you would need to have the Wilink COM module connected to the EVM. Make sure that this module supports the Bluetooth.

```
target # hciconfig hci0 up
target # hciconfig -a
```

Turn on the Bluetooth on the device that you want to pair and make it discoverable, then run the following command:

```
target # hcitool scan
```

How to bring up the GNSS driver and sample application

WL8 GNSS driver that is compatible with SDK is now available as part of the click wrap license at the following location. http://www.ti.com/tool/wilink-sw

Users are requested to register and obtain the package.

The package contains the driver source and the required documentation.

The document "Bring up manual for WiLink8 GNSS driver on Linux" is the starting point that contains the instructions for compiling and trying the sample application.

NOTE: These instructions are known to work if the user starts with the Processor SDK Linux Automotive installer and compiles the Linux kernel using the instructions provided in the Software Developers Guide.

Supported Boot modes

The same U-Boot image shipped with the SDK can be used to boot the system in following different modes based on the boot switch settings:

- MMC/SD
- eMMC
- Uart
- QSPI
- NAND

Booting EVM with different modes

Choosing the correct device tree

Booting Linux kernel needs to have a kernel image(zImage) and device tree blob(DTB) file. DTB file describes the hierarchy of the devices, and also describes various parameters about the devices.

Depending on which CPU board and application board you are trying to boot, there are different set of devices. For each permutation, there is a different DTB file describing all devices on baseboard as well as application board.

Release filesystem comes with most commonly used permutations of device trees. Following table shows the name of the device tree file to be used for each of the combination.

Device tree file (DTB) options

Board	DTB name
J6 EVM + 10inch LG LCD	dra7-evm-lcd-lg.dtb (Base board DTB works for JAMR3 as well)
J6 EVM + 10inch OSD LCD	dra7-evm-lcd-osd.dtb (Base board DTB works for JAMR3 as well)
J6 EVM + Vision app board	dra7-evm-vision.dtb
J6eco EVM upto revision C	dra72-evm-revc.dtb (Base board DTB works for JAMR3 as well)
J6eco EVM revision C + 10inch standard OSD panel	dra72-evm-revc-lcd-osd101t2045.dtb (Base board DTB works for JAMR3 as well)
J6eco EVM revision C + 10inch new OSD	dra72-evm-revc-lcd-osd101t2587.dtb (Base board DTB works for

panel	JAMR3 as well)
J6entry EVM	dra71-evm.dtb (Base board DTB works for JAMR3 as well)
J6entry EVM + 10inch AUO LCD panel	dra71-evm-lcd-auo-g101evn01.0.dtb (Base board DTB works for JAMR3 as well)
J6Plus EVM (HDMI only)	dra76-evm.dtb
J6Plus EVM + FPDLink display	dra76-evm-fpd-auo-g101evn01.0.dtb (Board modication required)

U-boot tries to detect the platform (J6/J6eco) and choose the appropriate dtb. For the DRA7xx boards, it only detects the baseboard. If you are using a stackup of baseboard plus LCD, then you must manually specify the proper dtb by defining fdtfile inside uenv.txt. For example, if you were using J6 EVM + 10inch LG LCD then you would add the following line to uenv.txt:

```
fdtfile=dra7-evm-lcd-lg.dtb
```

Identifying LCD Panels

Panel name	Resolution	Images	Remarks
LG LCD	1280x800	Media:lcd-lg-front.jpg, Media:lcd-lg-reverse.jpg, Media:lcd-lg-push-button.jpg	Has a circular hole on top for mounting camera. Has push buttons on the side. Device tree files end with lcd-lg.dtb
OSD Panel	1920x1200	Media:osd_2045_forward.jpg, Media:osd_2045_reverse.jpg	Back has notation "OSD 1080P Touch Display I/F board". Serial number start with OSD_. Device tree files end with lcd-osd.dtb or lcd-osd101t2045.dtb
New OSD Panel	1920x1200	TBD	Device tree files end with lcd-osd101t2587.dtb
AUO Panel	1280x800	Media:Auo-forward.jpg, Media:Auo_notation_reverse.jpg	Serial numbers on the back start with AU. Device tree files have AUO in name.

Choosing the correct bootloader config

All the padmux is performed from the first stage bootloader (MLO) instead of the kernel. Also, the bootloader takes care of configuring all the required PADs as well as the **IOdelay** configuration. Depending on the use case, boot loader needs to know the required pads to be configured at the boot time. Kernel would know the device information from DTB file but bootloader does not know which DTB file would be used by the kernel.

For this reason, we need to rebuild the bootloader (MLO) with a different config so that the appropriate PAD and IODELAY configuration is performed as required by the use case. Release prebuilt bootloader works for most of the use cases.

When using vision use cases, rebuild the bootloader with extra Kconfig CONFIG_TARGET_DRA7XX_EVM_VISION enabled

Using QSPI Boot

QSPI is a serial peripheral interface like SPI the major difference being the support for Quad read, uses 4 data lines for read compared to 2 lines used by the traditional SPI.

Supported boot modes

- QSPI Production Boot mode
- QSPI Development Boot mode

QSPI Production Boot Mode

This boot mode also called as 'spl_early_boot' in Processor SDK Linux Automotive.

- In this boot mode SPL(first stage of Uboot) directly boots the Linux kernel.
- The executables - MLO, dtb, uImage & IPU are stored in QSPI flash memory. Refer the "Memory Layout" section for offset details.
- eMMC partition 2 contains the filesystem & is mounted as rootfs.
- Build MLO and u-boot.img.

Note: By continuous press of character 'c' key on keyboard and resetting the EVM, SPL will load u-boot and enter into second stage boot loader.

SYS BOOT Switch Settings

change the boot switches to QSPI boot mode as:

```
SW2[5:0] = 110110 for development (jump to u-boot)
SW2[5:0] = 110111 for production mode (jump to MLO->kernel)
Also
SW5.4 = 0 (OFF)
```

Memory Layout of QSPI Flash

```
+----------------+ 0x00000
|      MLO       |
|                |
+----------------+ 0x040000
|   u-boot.img   |
|                |
+----------------+ 0x140000
|   DRA7-evm.dtb |
+----------------+ 0x1c0000
|   u-boot env   |
+----------------+ 0x1d0000
|   u-boot env   |
|    (backup)    |
+----------------+ 0x1e0000
|                |
|     uImage     |
|                |
|                |
+----------------+ 0x9e0000
|                |
|     IPU exe    |
|                |
+----------------+
```

Build kernel uImage for QSPI boot mode

In 'spl_early_boot' mode, While building kernel uImage, set CONFIG_CMDLINE bootargs appropriately and set rootfs=/dev/mmcblk0p2 (eMMC device) and set kernel CONFIG_CMDLINE bootargs @menuconfig->Boot options->"Default kernel command string".

for example:

```
 CONFIG_CMDLINE = "elevator=noop console=ttyO0,115200n8 root=<rootfs> rw rootwait earlyprintk fixrtc omapdrm.num_crtc=2 consoleblank=0  cma=64M rootfstype=ext4";
 where <rootfs> can be /dev/mmcblk0p2 for eMMC or /dev/mmcblk1p2 for mmc/sd.
```

alternate option: Add the bootargs in chosen node in DTB file, using fdtput utility.

```
  #fdtput -v -t s <DTB-FILE-PATH> "/chosen" bootargs "elevator=noop console=ttyO0,115200n8 root=<rootfs> rw rootwait earlyprintk fixrtc omapdrm.num_crtc=2 consoleblank=0  cma=64M
rootfstype=ext4"
 where <rootfs> can be /dev/mmcblk0p2 for eMMC or /dev/mmcblk1p2 for mmc/sd.
```

Flashing the Image to QSPI from Linux kernel

The mk-qspi-boot.sh will be available at home direcotry of "Processor SDK Linux Automotive" filesystem. The mk-qspi-boot.sh runs on the target. The scripts reads the binaries (MLO, u-boot.img, kernel/dtb, ipu images) from MMC/SD and flashes into QSPI flash memory at appropriate partition described in above table. Also format & creates rootfs partition in eMMC and copies the filesystem the eMMC rootfs parition.

```
Usage:
target#mk-qspi-boot.sh --device1 <mtd-device> --device2 <eMMC-device> --bootmode

--device1 - devfs entry for qspi flash as char device node
                        e.g /dev/mtd

--device2 - devfs entry for eMMC flash as block device node
                        e.g /dev/mmcblk0

--bootmode - 'spl_early_boot' & 'two_stage_boot'
                         spl_early_boot - ROM=>SPL=>uImage
                         two_stage_boot - ROM=>SPL=>u-boot.img=>uImage
e.g
target#: mk-qspi-boot.sh --device1 /dev/mtd --device2 /dev/mmcblk0 --bootmode spl_early_boot

Note: Defualt DTB is dra7-evm-lcd-osd.dtb. Do change DTB in script if you want to boot the board with different device tree.
You can modify the mk-qspi-boot.sh, make sure "emmc_dev" and "main_dev" in mk-qspi-boot.sh point to appropriate "/dev/mmcblkX" device node and
also set DTB_FILE_PATH, UIMAGE_FILE_PATH and IPU_FILE_PATH point to correct source.
```

QSPI Development Boot Mode

This boot mode also called as 'two_stage_boot' in "Processor SDK Linux Automotive"

- In this boot mode SPL(first stage of Uboot) brings-up u-boot.img(second part of u-boot).
- The second stage of u-boot, then, loads & boots the Linux kernel.
- Build MLO and u-boot.img for QSPI boot mode (use "dra7xx_evm_defconfig").
- Only MLO and u-boot.img are stored in QSPI flash memory.
- DTB & uImage are stored in MMC/SD boot parition.
- The rootfs is mounted from MMC/SD partition 2

Note: By continuous press of character 'c' key on keyboard and resetting the EVM, SPL will load u-boot and enter into second stage boot loader.

SYS BOOT Switch Settings

change the boot switches to QSPI boot mode as:

```
SW2[5:0] = 100110 for development (jump to u-boot)
SW2[5:0] = 100111 for production mode (single stage)
```

```
Also,
SW5.4 = 0 (OFF)
```

Flashing UBoot to QSPI from Linux kernel

The mk-qspi-boot.sh will be available at home direcotry of Processor SDK Linux Automotive filesystem. The mk-qspi-boot.sh runs on the target . It reads the executables from MMC/SD & flashes into QSPI flash memory for 'two_stage_boot'

```
For Usage of mk-spi-boot.sh refer to above section "Flashing_Image_to_QSPI_from_Linux"
```

Also QSPI flash can be accessed from kernel through MTD device.

Following commands demonstrate how to flash MLO and u-boot from SD card to QSPI.

```
target# mount /dev/mmcblk1p1 /mnt
target# cat /proc/mtd
target# flash_erase -N /dev/mtd0 0 4
target# flash_erase -N /dev/mtd1 0 16
target# mtd_debug write /dev/mtd0 0x0 $(ls -l /mnt/MLO {{!}} awk '{ print $5 }') /mnt/MLO
target# mtd_debug write /dev/mtd1 0x0 $(ls -l /mnt/u-boot.img {{!}} awk '{ print $5 }') /mnt/u-boot.img
```

Flashing UBoot to QSPI from UBoot

UBoot can be flashed to QSPI from any boot mode, here we are using MMC boot mode as an example
Boot from SD card and at the U-Boot prompt, Choose the mmc card to read from

```
uboot# mmc dev 0
uboot# mmc0 is current device
```

- Probe the flash to see if there is a device connected,

NOTE: Do NOT copy-paste the commands, please type them at the prompt

```
# sf probe 0
SF: Detected S25FL256S with page size 64 KiB, total 32 MiB, mapped at 5c000000
```

- Erase flash before writing the bin

NOTE: Do NOT copy-paste the commands, please type them at the prompt

```
# sf erase <offset> +<size>
where,
 offset - qspi flash offset address start from 0.
 size   - erase region size. This is automatically rounded up to the block size.
# sf write <memory address> <offset> <len>
where,
  memory-address - source data to read from
  offset - qspi offset location to write
  len    - length of the data to be written
```

- Load the MLO (to offset 0x0) and u-boot.img(to offset 0x40000) to QSPI flash

```
Flashing MLO to QSPI
uboot# fatload mmc 0 0x82000000 MLO.qspi
uboot# sf erase 0 +${filesize}
uboot# sf write 0x82000000 0x0 ${filesize}

Flashing u-boot.img to QSPI
uboot# fatload mmc 0 0x83000000 u-boot.img
uboot# sf erase 40000 60000;
uboot# sf write 0x83000000 0x40000 0x60000
```

- Build kernel to create uImage and dtb files. Load the uImage(to offset 0x1e0000) and DTB files (to offset 0x140000) to QSPI flash

```
Flashing kernel(uImage) to QSPI
uboot# fatload mmc 0 0x82000000 uImage
uboot# sf erase 1e0000 500000;
uboot# sf write 82000000 1e0000 500000;

Flashing dtb file to QSPI
uboot# fatload mmc 0 0x83000000 <dtb file>
uboot# sf erase 140000 20000;
uboot# sf write 0x83000000 0x140000 0x20000
```

After reset, you should be able to boot from QSPI flash.

Using eMMC Boot

eMMC boot method is the same as MMC/SD flashing and booting procedure. Format MMC/SD card and create boot/rootfs partition (refer to Software development setup section).

```
$ sudo ${INSTALL_DIR}/bin/mksdboot.sh --device /dev/sdY --sdk ${INSTALL_DIR}
```

The MMC/SD boot partition contains MLO,u-boot.img and uenv.txt. The rootfs partition contains ext4 file system with kernel & DTBs in rootfs/boot directory. Formatting of eMMC device can be done by using mk-eMMC-boot.sh script. Boot the EVM with MMC/SD boot mode first and then run the mk-eMMC-boot.sh script to create the eMMC partitions.

Partitioning and formatting eMMC

- Boot kernel using MMC/SD boot mode (refer to Software development setup section).
- mount the MMC/SD boot partition, create uenv-emmc.txt.

```
# mount /dev/mmcblk1p1 /mnt
# cp /mnt/uenv.txt /mnt/uenv-emmc.txt
Note: edit /mnt/uenv-emmc.txt and set rootfs=/dev/mmcblk0p2 in bootargs.
```

- consider /dev/mmcblk0 is eMMC device, then use the following script to partition the eMMC device.The mk-eMMC-boot.sh will create boot, rootfs partition on eMMC device and copy the all files from MMC/SD card to eMMC.

Note: Edit mk-eMMC-boot.sh script file and make sure "mmc_dev" point to appropriate MMC/SD device node (/dev/mmcblkX).

```
#./mk-eMMC-boot.sh --device /dev/mmcblk0
```

- reboot EVM to u-boot prompt, and set environment variables "mmcdev" and the "bootpart" point to eMMC device.

```
 #env default -a
 #setenv bootpart 1:2
 #setenv mmcdev 1
 #saveenv
```

- Set boot switches to EMMC boot mode.

```
Set SW2[7..0] = 00111000
```

- Remove MMC/SD card and reboot the EVM, now should boot from eMMC device.

Partitioning and formatting eMMC from Host PC through USB

The ums (USB Mass Storage feature) command in U-Boot can be used to expose the MMC as USB storage drive (/dev/sdX) to Host PC.

Syntax: ums <usb controller instance> mmc <mmc-instance>

- Setup: Connect USB0 port of the EVM to Ubuntu host PC.

From U-boot prompt

- Exposing MMC/SD as storage media

```
=> ums 0 mmc 0
```

- Exposing eMMC as storage media

```
=> ums 0 mmc 1
```

This command will expose the mmc device as storage device (/dev/sdX) to Ubuntu PC, further user can mount and partition, format and copy the files to device.

Booting Secure DRA7xx Devices SD/eMMC

This section has been updated and moved to this link: http://processors.wiki.ti.com/index.php/Processor_SDK_Linux_HS_Support

USB DRD (Dual Role Device) Support

The USB ports has capable of switching between device mode or host mode dynamically without removal of the USB cable.

- make sure dr_mode property in DTB is set to "otg" for DRD support.
- mount the debugfs node

```
# mount -t debugfs debugfs /mnt
```

- load the gadget module.

```
# modprobe <gadget.ko>
```

- To switch between host mode for USB1 and USB2 port.

```
# echo "host" > /mnt/48890000.usb/mode
# echo "host" > /mnt/488d0000.usb/mode
```

- To switch between host mode for USB1 and USB2 port.

```
# echo "device" > /mnt/48890000.usb/mode
# echo "device" > /mnt/48890000.usb/mode
```

Note: The role switching can be done dynamically based on board USB-ID (SW1 switch on TI-EVM).

```
# If host-type (A-type) usb cable is connected, the port switched to host mode.
# If device-type (B-type) usb cable is connected, the port switched to device mode.
```

Note:

- Refer DRD support (http://processors.wiki.ti.com/index.php/Template:GLSDK_USB_DRD#Introduction) for more information.
- Refer drive-vbus-using-gpio (http://processors.wiki.ti.com/index.php/Template:GLSDK_USB_DRD#Driving_VBUS_using_GPIO) in order to drive the vbus using gpio.

Device Firmware Upgrade (DFU)

The Device Firmware Upgrade (DFU) feature is used to program or flash the firmware to memory devices such as eMMC/MMCSD/QSPI/NOR/NAND/RAM devices. The u-boot has DFU support for eMMC, MMC/SD, QSPI and RAM devices.

DFU Supported Devices in u-boot

- eMMC
- MMC/SD
- RAM
- QSPI

SPL-DFU support (USB Peripheral boot mode)

This method is used to flash the binary images from ubuntu Host PC (using dfu-utils tool) to EVM over usb interface using Device Firmware Update (DFU) feature. This is used to flash the eMMC, or QSPI to fresh/factory boards.

- Use default "dra7xx_evm_defconfig" to build spl/u-boot-spl.bin, u-boot.img.

```
# make dra7xx_evm_defconfig
```

- select SPL->DFU support from menuconfig->"Boot Images"->"Enable SPL with DFU to load binares to memory device"
- Build spl/u-boot-spl.bin and u-boot.img

```
# make
```

- Set SYSBOOT SW2 switch to USB Peripheral boot mode

```
SW2[7..0] = 0000000
```

- Connect EVM Superspeed port (USB1 port) to PC (Ubuntu) through USB cable.
- From Ubuntu PC, load spl/u-boot-spl.bin to EVM. Issue below command and reset the board.

```
$ sudo ./usbboot -S spl/u-boot-spl.bin
```

- Load the u-boot.img to RAM.

```
# sudo dfu-util -l
```

```
Found DFU: [0451:d022] devnum=0, cfg=1, intf=0, alt=0, name="kernel"
Found DFU: [0451:d022] devnum=0, cfg=1, intf=0, alt=1, name="fdt"
Found DFU: [0451:d022] devnum=0, cfg=1, intf=0, alt=2, name="ramdisk"
```

```
$ sudo dfu-util c 1 -i 0 -a 0 -D "u-boot.img" -R
```

- Now EVM will boot to u-boot prompt.

DFU Usage from u-boot

Use default "dra7xx_evm_defconfig" to build MLO and u-boot.img.

```
make dra7xx_evm_defconfig
make
```

Connect the EVM with Ubuntu Host through micro USB device cable (USB1 port on EVM).

- From target u-boot prompt

Reset the default environment variable

```
# env default -a
```

set the dfu_alt_info environment variable to emmc dfu settings

```
# setenv dfu_alt_info ${dfu_alt_info_emmc}
```

- Run DFU command chosing eMMC as memory device

dfu <usb controller number> mmc <1-eMMC, 0-mmc/sd>

```
# dfu 0 mmc 1
set_config: high speed config #1: usb_dnload
```

- From PC host side (Ubuntu)

To list the available partition to load the file.

```
# sudo dfu-util -l
dfu-util 0.5
(C) 2005-2008 by Weston Schmidt, Harald Welte and OpenMoko Inc.
(C) 2010-2011 Tormod Volden (DfuSe support)
This program is Free Software and has ABSOLUTELY NO WARRANTY
dfu-util does currently only support DFU version 1.0
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=0, name="rawemmc"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=1, name="boot"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=2, name="rootfs"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=3, name="MLO"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=4, name="MLO.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=5, name="u-boot.img.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=6, name="spl-os-args.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=7, name="spl-os-image.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=8, name="spl-os-args"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=9, name="spl-os-image"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=10, name="u-boot.img"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=11, name="uEnv.txt
```

The above list shows list of dfu partitions or file name.

Flashing rawemmc partition

The "rawemmc" partition is used to update the disk image to raw eMMC sector 0. This is used to flash the eMMC to fresh boards. Please refer to section *Using USB peripheral boot mode*

Update existing MLO and u-boot.img of eMMC boot partition (vfat)

This method is used to update the MLO/u-boot.img files in boot(vfat) partion of eMMC

- To update the MLO file

```
#sudo dfu-util -D MLO -c 1 -i 0 -a 3
```

- To update the u-boot.img file

```
#sudo dfu-util -D u-boot.img -c 1 -i 0 -a 10
```

Update existing boot and rootfs partition (ext4) of eMMC

This method is used to update the existing boot partition (vfat) or rootfs partition (ext4) images in eMMC

- To update the boot partition disk image

```
#sudo dfu-util -D boot.img -c 1 -i 0 -a 1
```

- To update the rootfs partition disk image

```
#sudo dfu-util -D rootfs.img -c 1 -i 0 -a 2
```

Steps to create boot (vfat) and rootfs (ext4) raw disk image

In order to create raw boot and rootfs disk image along with partition structure follow the steps mentioned below.

1. Insert the mmc/sd card to Ubuntu PC, the size of sdcard shall be less than or equal to size of eMMC populated in EVM.
2. Prepare mmc/sd card mentioned in step4 of Processor SDK Linux Automotive User Guide (http://processors.wiki.ti.com/index.php/DRA7xx_Processor_SDK_Linux_Automotive_Software_Developers_Guide#Starting_your_software_development). By default the mksdboot.sh script creates two partitions, 64MB of boot partition with vfat filesystem and remaining size of sd card with rootfs partition with ext4 filesystem.
3. After sd card is created with boot and rootfs partition, mount the sdcard to update uenv.txt and overwrite uenv.txt by uenv-emmc.txt in "boot" partition. Copy any additional files required to boot/rootfs partition.

```
 #mount /dev/sdY1 /media/boot
 #mount /dev/sdY2 /media/rootfs
 #cp /media/boot/uenv-emmc.c /media/boot/uenv.txt
 #sync
```

Create the vfat raw boot disk image along with partition structure

```
#dd if=/dev/<sdY1> of=boot.img bs=1M &
```

Create the ext4 raw rootfs disk image along with partition structure

```
#dd if=/dev/<sdY2> of=rootfs.img bs=1M &
```

Flashing/Upgrading custom file to eMMC vfat/ext4 partition

1. If eMMC device is not partitoned, then first the eMMC device must be partitioned to vfat or ext4 filesystem. ReferFlashing_binaries_to_the_factory_boards_using_Device_Firmware_Upgrade_(DFU).pdf (http://processors.wiki.ti.com/index.php/File:Flashing_binariesto_the_factory_boards_using_Device_Firmware_Upgrade(DFU).pdf)
2. Once the eMMC device has been partitioned to vfat/ext4, user can write into existing file or partition. For example as shown in above list of dfu interfaces, user can upgrade the MLO or u-boot.img file from ubuntu PC. The alternate interface number 3 need to selected for upgrading MLO, and 10 need to selected for u-boot.img.

```
#sudo dfu-util -D MLO -c 1 -i 0 -a 3
#sudo dfu-util -D u-boot.img -c 1 -i 0 -a 10
```

Adding a custom file to DFU configuration

1. Add a custom/new file to the root directory of existing rootfs(ext4) or boot(vfat) partition of sdcard and flash the filesystem raw disk images to eMMC. For example consider the file zImage, dra7-evm-lcd10.dtb is added to root directory of rootfs(ext4) and README file in boot(vfat) partition. Make sure file read/write permission is set.
2. From u-boot source repository, edit the file include/configs/dra7xx_evm.h, add a new entry to DFU_ALT_INFO_EMMC macro.

```
#define DFU_ALT_INFO_EMMC \
       "dfu_alt_info_emmc=" \
       "rawemmc raw 0 3751936;" \
       "boot part 1 1;" \
       "rootfs part 1 2;" \
       "MLO fat 1 1;" \
       "ZImage ext4 1 2;" \
       "dra7-evm-lcd10.dtb ext4 1 2;" \
       "README fat 1 1;" \
       "MLO.raw raw 0x100 0x100;" \
       "u-boot.img.raw raw 0x200 0x400;" \
       "spl-os-args.raw raw 0x80 0x80;" \
       "spl-os-image.raw raw 0x900 0x2000;" \
       "spl-os-args fat 1 1;" \
       "spl-os-image fat 1 1;" \
       "u-boot.img fat 1 1;" \
       "uEnv.txt fat 1 1\0"
```

3. Compile the u-boot to create MLO, u-boot.img and load MLO/u-boot.img to EVM
4. Execute "dfu-util -l" from PC host, the new interfaces added will be listed

```
#sudo dfu-util -l
dfu-util 0.5
(C) 2005-2008 by Weston Schmidt, Harald Welte and OpenMoko Inc.
(C) 2010-2011 Tormod Volden (DfuSe support)
This program is Free Software and has ABSOLUTELY NO WARRANTY
dfu-util does currently only support DFU version 1.0
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=0, name="rawemmc"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=1, name="boot"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=2, name="rootfs"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=3, name="MLO"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=4, name="zImage"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=5, name="dra7-evm-lcd10.dtb"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=6, name="README"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=7, name="MLO.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=8, name="u-boot.img.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=9, name="spl-os-args.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=10, name="spl-os-image.raw"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=11, name="spl-os-args"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=12, name="spl-os-image"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=13, name="u-boot.img"
Found DFU: [0403:bd00] devnum=0, cfg=1, intf=0, alt=14, name="uEnv.txt"
```

6. Now you can upgrade the new added files by chosing the respective alternate interface number (alt=<number>)

```
Upgrade the zImage
#sudo dfu-util -c 1 -i 0 -a 4 -D zImage

Upgrade the dra7-evm-lcd10.dtb
#sudo dfu-util -c 1 -i 0 -a 4 -D dra7-evm-lcd10.dtb
```

Using dfu-utils from Windows PC Host

Follow the below steps to install the dfu-tools.
1. Download the dfu-utils dfu-util-0.6-win32.zip.bz2 (http://dfu-util.sourceforge.net/releases/) and unzip to some folder (say c:\dfu)
2. Download the zdaig application (https://sourceforge.net/projects/libwdi/files/zadig/) and execute zdaig application, install winUSB driver.
3. Go through DFU section mentioned above for setup the EVM for DFU.

```
Note: For flashing fresh boards, refer to peripheral usb boot mode section.
The initial bootloader (u-boot-spl.bin) shall be loaded to EVM using usbboot tool running from Ubuntu Host PC.
For more details refer Flashing_binaries_to_the_factory_boards_using_Device_Firmware_Upgrade_(DFU).pdf (http://processors.wiki.ti.com/index.php/File:Flashing_binariesto_the_factory_boards_using_Device_Firmware_Upgrade(DFU).pdf)
```

4. Connect the EVM (running DFU) to windows PC through superspeed usb cable. Windows host will detect the EVM as "download gadget".
5. Copy required binary files to flash c:\dfu folder. Open command prompt and execute dfu-util.exe.

```
c:\dfu> dfu-util –l
This will list all dfu interfaces
```

6. Follow DFU section mentioned above for flashing the binaries to eMMC.

```
c:\dfu> dfu-util -D <file-name> -c 1 -i 0 -a <alt number>
```

IPC

The table below shows the remote cores and their corresponding definitions in the kernel dtsi files (`${INSTALL_DIR}/board-support/linux/arch/arm/boot/dts/dra7.dtsi, and dra74x.dtsi`), as well as the argument to be used in the loading/unloading commands.

Remote Core	Definition in dtsi file	Argument in loading/unloading
IPU1	ipu@58820000	58820000.ipu
IPU2	ipu@55020000	55020000.ipu
DSP1	dsp@40800000	40800000.dsp
DSP2	dsp@41000000	41000000.dsp

For example, the argument of `55020000.ipu` corresponds to IPU2 as can be seen from `dra7.dtsi`.

```
   ipu2: ipu@55020000 {
        compatible = "ti,dra7-rproc-ipu";
```

In the sections below, `55020000.ipu` will be used as the example. For a specific use case, please select the corresponding argument which is applicable.

Unloading and loading remotecores at runtime

It is possible to unload and reload a remotecore at runtime from Linux using the `sysfs` interface.

```
target $ cd /sys/bus/platform/drivers/omap-rproc/
target $ echo 55020000.ipu > unbind
target $ echo 55020000.ipu > bind
```

The `echo 55020000.ipu > unbind` command tears down the communication channels between the A15 and the remotecore and unloads the remotecore. Any application level shutdown that needs to be performed needs to be handled by the system integrator.

The `echo 55020000.ipu > bind` loads the appropriate firmware binary onto the remotecore.

Changing the remotecore binary at runtime

To change the remotecore binary at runtime

1. Unload the remotecore using `unbind`.
2. Change the remotecore binary in the firmware folder. Default location is `/lib/firmware` on the target filesystem.
3. Load the remotecore using `bind`.

```
target $ cd /sys/bus/platform/drivers/omap-rproc/
target $ echo 55020000.ipu > unbind
target $ cp /home/root/new-binary.xem4 /lib/firmware/dra7-ipu2-fw.xem4
target $ echo 55020000.ipu > bind
```

If it is desirable to avoid overwriting the existing remote binaries, the method of symbolic links can be used instead of direct copy. For example, Processor SDK provides two types of DSP remotecore binaries: one for DSPDCE (dra7-dsp1-fw.xe66.dspdce-fw) and another one for OpenCL (dra7-dsp1-fw.xe66.opencl-monitor). dra7-dsp1-fw.xe66 is created as a symbolic link by default pointing to the OpenCL binary. When it is needed to switch to DSPDCE, the symbolic link of dra7-dsp1-fw.xe66 can be updated pointing to dra7-dsp1-fw.xe66.dspdce-fw.

```
target $ cd /sys/bus/platform/drivers/omap-rproc/
target $ echo 40800000.dsp > unbind
target $ rm /lib/firmware/dra7-dsp1-fw.xe66
target $ ln -s /lib/firmware/dra7-dsp1-fw.xe66.dspdce-fw /lib/firmware/dra7-dsp1-fw.xe66
target $ echo 40800000.dsp > bind
```

After the switch, copycodectest application can be run to verify that DSPDCE firmware is loaded. This application fills the input buffer with a number entered as the argument and after process the output buffer is tested for the same pattern.

usage: copycodectest pattern.

Example:

```
 target # copycodectest 123
```

Sample console output:

```
root@am57xx-evm:~# copycodectest 123
0x22070: Opening Engine..
Created dsp_universalCopy
Fill input buffer with pattern 123
Verifing the UniversalCopy algorithm
copycodectest executed successfully
```

Loading firmware during initial boot without using udev

During the default boot, firmware is supplied to the kernel by `udev`. Starting the `udev` service on boot causes a few seconds increase in boot time. In cases where a quick boot is required, the user may not start the `udev` service in boot. In such cases, firmware can be supplied to the kernel using the sysfs interface. An example script is shown below.

```
FW_NAMES="dra7-dsp1-fw.xe66 dra7-dsp2-fw.xe66 dra7-ipu1-fw.xem4 dra7-ipu2-fw.xem4"
for FW in $FW_NAMES ; do
    echo 1 > /sys/class/firmware/$FW/loading
    cat /lib/firmware/$FW > /sys/class/firmware/$FW/data
    echo 0 > /sys/class/firmware/$FW/loading
done
```

Handling the IPU bitbanding region

The ARM Cortex-M4 memory map includes a bit-banding region of memory from 0x4000:0000 to 0x400F:FFFF and 0x4200:0000 to 0x43FF:FFFF. Here is a Cortex-M4 memory map picture from ARM:

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CHDBIJJE.html

Many Vayu components running on the IPUs, including IPC, must access peripherals physically located in this bit-banding region. As a result, these accesses must be performed indirectly using a virtual memory address, mapped using the IPU's AMMU.

Many of the components aligned on mapping this memory using one Large AMMU page that maps 512M of physical memory beginning at 0x4000:0000 to virtual memory beginning at 0x6000:0000. Then the components (by default) access the peripherals using the 0x6XXX:XXXX address space.

IPC specifics

IPC follows the convention of, by default, accessing memory physically located at 0x4XXX:XXXX using virtual memory at 0x6XXX:XXXX. You can see an example of this here - note the mailbox addresses configured here are in the 06XXX:XXXX range:

http://git.ti.com/cgit/cgit.cgi/ipc/ipcdev.git/tree/packages/ti/sdo/ipc/family/vayu/InterruptIpu.xs

In that same file, you can see that these addresses are configurable, and the default 0x6XXX:XXXX addresses are only used if other addresses haven't already been configured by the system integrator (e.g. in a .cfg script). Users can override these default mailbox addresses using the ti.sdo.ipc.family.vayu.InterruptIpu module's mailboxBaseAddr[] array, documented here:

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/latest/docs/cdoc/ti/sdo/ipc/family/vayu/InterruptIpu.html#metamailbox.Base.Addr

Using the Late attach functionality

NOTE! This version of PSDKLA release supports Early Boot / Late Attach of remoteproc cores IPU1, IPU2, DSP1 and DSP2.

To satisfy the startup time requirements of specific use cases, one may need the boot loader to boot a remote core before booting the A15 with the linux kernel. The kernel then attaches to the already booted remote core for further communication. We refer to this feature as the "Early Boot - Late Attach" functionality. The "Early Boot" functionality is provided by the boot loader. The "Late Attach" functionality is a feature of the Linux Kernel. This functionality was added to Processor SDK - Linux Automotive from v3.01.

Below are the steps required to test early boot late attach functionality.

Configuring U-Boot

Please apply this patch http://processors.wiki.ti.com/index.php/File:0001-u-boot-configs-Enable-configs-for-Early-boot-support.zip on U-boot repository in board-support to enable Early Boot of remoteproc cores.

and rebuild u-boot using the configuration `dra7xx_evm_config`.

Building the Linux kernel device tree

Linux kernel contains the "Late Attach" feature builtin. The feature is enabled or disabled on a per remote core basis at boot time through device tree node attributes. To enable "Late Attach" for a remote core, 2 attributes need to be set on the remote core and each of the timer, mmu nodes used by the remotecore. These two attributes are

1. `ti,no-idle-on-init`
2. `ti,no-reset-on-init`.

These two attributes together signal to the kernel that

1. Late attach feature is in use for the remote core.
2. The remotecore and other nodes have been configured and are in use before the kernel boot. These should not be reset or idled during kernel boot.

The Linux kernel tree delivered with Processor SDK Linux Automotive includes a dts file that shows how to enable late attach feature for IPU2 on a DRA7xx evm . This file was built in the following manner. The timers used by IPU2 are found from `$PSDKLA/board-support/linux/arch/arm/boot/dts/dra7-evm.dts`

```
&ipu2 {
    status = "okay";
    memory-region = <&ipu2_cma_pool>;
    mboxes = <&mailbox6 &mbox_ipu2_legacy>;
    timers = <&timer3>;
    watchdog-timers = <&timer4>, <&timer9>;
};
```

The MMU used by IPU2 can be found from `$PSDKLA/board-support/linux/arch/arm/boot/dts/dra7.dtsi`.

```
        ipu2: ipu@55020000 {
            compatible = "ti,dra7-rproc-ipu";
            reg = <0x55020000 0x10000>;
            reg-names = "l2ram";
            ti,hwmods = "ipu2";
            iommus = <&mmu_ipu2>;
            ti,rproc-standby-info = <0x4a008920>;
            status = "disabled";
        };
```

IPU2 uses `timer3` to supply the OS tick and `timer4` and `timer9` as watchdog timers. IPU2 also uses an MMU referred to as `mmu_ipu2` in the device tree. All of these nodes including the IPU2 node need to have the three attributes described above set.

We also reserve a small region of memory(1 MB) for U-Boot to store the pagetables of the MMU's for the DSP's and the IPU's. The memory reservation prevents the kernel from reusing the memory.

```
&reserved_mem {
        mmu-early-page-tables@95700000 {
                reg = <0x0 0x95700000 0x0 0x100000>;
                no-map;
                status = "okay";
        };
};
```

The file `$PSDKLA/board-support/linux/arch/arm/boot/dts/dra7-evm-late-attach.dts` has the full example. This file is structured such that it includes an existing configuration first

```
#include "dra7-evm.dts"
```

and then sets only the attributes required for enabling late attach functionality for each remoteproc with the inclusion of dtsi files. Here dra7-ipu-common-early-boot.dtsi sets late-attach functionality for both IPU1 and IPU2, dra7-dsp-common-early-boot.dtsi sets late-attach functionality for DSP1 and dra74x-dsp-common-early-boot.dtsi sets late-attach functionality for DSP2.

```
 #include "dra7-ipu-common-early-boot.dtsi"
 #include "dra7-dsp-common-early-boot.dtsi"
 #include "dra74x-dsp-common-early-boot.dtsi"
```

If Late attach is not required for a specific core, delete the firmware from boot partition and remove the late-attach attributes for that cores in the dts and rebuild the dtb. Late attach can be enabled for other hardware configurations by choosing a different base `.dts` file. e.g. If the EVM has an OSD lcd instead of 10" LCD, you can change the include to "dra7-evm-lcd-osd.dts" instead of "dra7-evm-lcd10.dts".

Build the `.dtb` file and use it for linux boot instead of the regular dtb file.

Boot setup

Messageq_single firmwares from ipc have been used to validate Early-Boot-Late-Attach functionality.

1. Configure the EVM for boot from SD card.
2. Use the MLO and u-boot.img built above with the early boot functionality enabled.
3. Use the dtb file built with the late attach functionality enabled.
4. Sybolic link IPC firmware in lib/firmware of the rootfs with the command `ln -s ipc/ti_platforms_evmDRA7XX_ipu2/messageq_single.xem4 dra7-ipu2-fw.xem4` and follow the same for other 3 firmwares.

Core	Binary path on target relative to /lib/firmware	Binary should be symlinked to
DSP2	./ipc/ti_platforms_evmDRA7XX_dsp2/messageq_single.xe66	/lib/firmware/dra7-dsp2-fw.xe66
IPU2	./ipc/ti_platforms_evmDRA7XX_ipu2/messageq_single.xem4	/lib/firmware/dra7-ipu2-fw.xem4
IPU1	./ipc/ti_platforms_evmDRA7XX_ipu1/messageq_single.xem4	/lib/firmware/dra7-ipu1-fw.xem4
DSP1	./ipc/ti_platforms_evmDRA7XX_dsp1/messageq_single.xe66	/lib/firmware/dra7-dsp1-fw.xe66

Copy the IPU1, IPU2, DSP1 and DSP2 binaries into the boot partition of the SD card.

Boot the board and test the IPU2 functionality with this command

```
root@dra7xx-evm:~#MessageQApp 10 1
```

Confirming Late attach functionality

There is no default indication of whether late attach feature is in use or not. This is by design as use of late attach should be invisible to the user. For debug purposes, enable the `DEBUG` flag on `drivers/remoteproc/omap_remoteproc.c` during kernel compilation. If late attach functionality is in use, additional traces will now show up in the dmesg logs.

With messsageq_single firmwares provided with this release, timestamp is taken, when remoteproc core started up and entered main. Execute this command

```
root@dra7xx-evm:~#cat /sys/kernel/debug/remoteproc/remoteproc1/trace0
```

shows

```
[0][      0.000] Watchdog enabled: TimerBase = 0x6883e000 SMP-Core = 0 Freq = 19200000
[0][      0.000] Watchdog enabled: TimerBase = 0x688e6000 SMP-Core = 1 Freq = 19200000
[0][      0.000] Watchdog_restore registered as a resume callback
[0][      0.000] 18 Resource entries at 0x3000
[0][      0.000] messageq_single.c:main: MultiProc id = 1
[0][      0.000] Time at startup()  is 30172 ticks
[0][      0.000] Time at main()  is 30293 ticks
[0][      0.000] registering rpmsg-proto:rpmsg-proto service on 61 with HOST
[0][      0.000] tsk1Fxn: created MessageQ: SLAVE_IPU2; QueueID: 0x10080
[0][      0.000] Awaiting sync message from host...
```

When the core is bootstrapped from u-boot, the values of 'Time at startup()' and 'Time at main()' should be below 60,000 ticks since it typically takes less than 2s to load the firmware. These values are above 500,000 when the firmware is bootstrapped from kernel. Eary-Boot and Late-Attach validation can be verified for other cores IPU1, DSP1 and DSP2 also same way with messageq_single firmwares.

Customizing Early Boot for a Usecase

The Early boot code in U-Boot does the necessary configuration to bring up a remotecore. This includes the timers and the MMUs. It does not configure any other peripherals by default. Some usecases may require additional peripheral configuration before running the remotecore. U-Boot includes placeholder functions that can be populated for this purpose. These can be found in the file `$GLSDK/board-support/u-boot/board/ti/dra7xx/evm.c`.

```
/*
 * If the remotecore binary expects any peripherals to be setup before it has
 * booted, configure them here.
 *
 * These functions are left empty by default as their operation is usecase
 * specific.
 */

u32 ipu1_config_peripherals(u32 core_id, struct rproc *cfg)
{
    return 0;
}

u32 ipu2_config_peripherals(u32 core_id, struct rproc *cfg)
{
    return 0;
}

u32 dsp1_config_peripherals(u32 core_id, struct rproc *cfg)
{
    return 0;
}

u32 dsp2_config_peripherals(u32 core_id, struct rproc *cfg)
{
    return 0;
}
```

Validation and Limitations

Late attach functionality has been validated for all the remotecores IPU2,IPU1,DSP1 and DSP2. However there is the following limitation.

- U-Boot has only been updated to load remotecore binaries from the boot partition of the SD card or eMMC. If you desire to load remotecore binaries from other media (e.g. QSPI), the code to load the binaries needs to be added to U-Boot.

Debug guide

A more detailed user guide of Early Boot and Late Attach functionality can be found at Early Boot and Late Attach in Linux

Changing MPU Frequency (DVFS)

DRA7xx supports running MPU (CPU0/1) at more than one frequency.
User may chose to run at different frequencies based on performance requirements for the application/use-case targeted.
The Data Manual contains all the frequencies supported by the Hardware and the Processor SDK Linux_Automotive Datasheet available with current release contains details of supported frequencies in the Software.
Please refer to the following wiki for details on how to change CPU frequency and voltage DVFS UserGuide (http://processors.wiki.ti.com/index.php/GLSDK_DRA7xx_PM_DVFS_User_Guide)

Testing OpenCL

NOTE! Support for OpenCL is available only upto K4.4 releases. Check the release notes to see if this feature is available on the Processor SDK Linux Automotive release you are using.

OpenCL is supported in the SDK from the 3.02 release. To test OpenCL, please follow the below steps.

1. OpenCL requires reserving areas of memory to allow transfer of data between A15 and DSP. To reserve memory, apply the below kernel patch, rebuild the device tree file and boot with the updated device tree.

   http://review.omapzoom.org/38240 `dra7xx: dts: reserve memory for opencl`

2. Ensure that the OpenCL firmware is loaded to the DSP. In the default PSDKLA filesystem, OpenCL firmware is not loaded by default. Follow the below steps to select the right firmware and reboot.

   <source lang="bash">root@dra7xx-evm:~# cd /lib/firmware/ root@dra7xx-evm:/lib/firmware# ls -la *dsp*-fw.xe66 lrwxrwxrwx 1 1000 pulse 55 Feb 13 2017 dra7-dsp1-fw.xe66 -> ipc/ti_platforms_evmDRA7XX_dsp1/test_omx_dsp1_vayu.xe66 lrwxrwxrwx 1 1000 pulse 55 Feb 13 2017 dra7-dsp2-fw.xe66 -> ipc/ti_platforms_evmDRA7XX_dsp2/test_omx_dsp2_vayu.xe66 root@dra7xx-evm:/lib/firmware# rm dra7-dsp1-fw.xe66 dra7-dsp2-fw.xe66 root@dra7xx-evm:/lib/firmware# ln -s dra7-dsp1-fw.xe66.opencl-monitor dra7-dsp1-fw.xe66 root@dra7xx-evm:/lib/firmware# ln -s dra7-dsp2-fw.xe66.opencl-monitor dra7-dsp2-fw.xe66 root@dra7xx-evm:/lib/firmware# ls -la *dsp*-fw.xe66 lrwxrwxrwx 1 root root 32 Feb 24 2017 dra7-dsp1-fw.xe66 -> dra7-dsp1-fw.xe66.opencl-monitor lrwxrwxrwx 1 root root 32 Feb 24 2017 dra7-dsp2-fw.xe66 -> dra7-dsp2-fw.xe66.opencl-monitor

   root@dra7xx-evm:/lib/firmware# sync</source>

3. Ensure that the cmemk module is inserted in the kernel.

   <source lang="bash">root@dra7xx-evm:~# lsmod | grep cmemk

   cmemk 35110 2</source>

4. Ensure that CMEM memory reservations are in effect. Link (http://downloads.ti.com/mctools/esd/docs/opencl/memory/ddr-partition.html)

   <source lang="bash">root@dra7xx-evm:~# cat /proc/iomem | grep -i cmem 40500000-405fffff : CMEM

   a0000000-abffffff : CMEM</source>

5. OpenCL examples are present in the filesystem in `/usr/share/ti/examples/opencl`. The examples are prebuilt and can be executed on the target.

   <source lang="bash">root@dra7xx-evm:/usr/share/ti/examples/opencl# ls Makefile dgemm float_compute monte_carlo offline_embed sgemm vecadd_openmp buffer dsplib_fft make.inc null ooo_callback simple vecadd_openmp_t ccode edmamgr matmpy offline platforms vecadd

   root@dra7xx-evm:/usr/share/ti/examples/opencl# cd platforms/ root@dra7xx-evm:/usr/share/ti/examples/opencl/platforms# ls Makefile Makefile.rtos main.cpp main.o platforms

   root@dra7xx-evm:/usr/share/ti/examples/opencl/platforms# ./platforms PLATFORM: TI AM57x Version: OpenCL 1.1 TI product version 01.01.11.2 (Feb 19 2017 21:41:17) Vendor : Texas Instruments, Inc. Profile: FULL_PROFILE

   ```
   DEVICE: TI Multicore C66 DSP
   Type       : ACCELERATOR
   CompUnits  : 2
   Frequency  : 0.6 GHz
   Glb Mem    :  163840 KB
   GlbExt1 Mem:       0 KB
   GlbExt2 Mem:       0 KB
   Msmc Mem   :    1024 KB
   Loc Mem    :     128 KB
   Max Alloc  :  147456 KB
   ```

   root@dra7xx-evm:/usr/share/ti/examples/opencl/platforms# cd ../vecadd root@dra7xx-evm:/usr/share/ti/examples/opencl/vecadd# ./vecadd DEVICE: TI Multicore C66 DSP

   Offloading vector addition of 8192K elements...

   Kernel Exec : Queue to Submit: 13 us Kernel Exec : Submit to Start : 120 us Kernel Exec : Start to End : 33010 us

   Success! </source>

   Full instructions on building OpenCL examples are here.

   http://downloads.ti.com/mctools/esd/docs/opencl/examples/build_and_run.html

   Instructions on testing OpenCL examples are here. This page also includes a description of each example and the functionality demonstrated by each example.

   http://downloads.ti.com/mctools/esd/docs/opencl/examples/overview.html

Information on environment variables controlling the OpenCL runtime are described in the below link. The number of DSP's used for OpenCL and which DSP is used for OpenCL offloading can also be controlled using environment variables described in the below link.

http://downloads.ti.com/mctools/esd/docs/opencl/environment_variables.html

Testing OpenCV

NOTE! Support for OpenCV is available only upto K4.4 releases. Check the release notes to see if this feature is available on the Processor SDK Linux Automotive release you are using.

The 3.02 release also supports OpenCV. OpenCV is accelerated by using the C66x DSP's through OpenCL. **Please follow the steps to validate OpenCL before running the OpenCV tests**

1. Download OpenCV test data from https://github.com/Itseez/opencv_extra/archive/master.zip unzip this file in `/usr/share/OpenCV`. This will create a directory `opencv_extra-master` with the below contents.

   <source lang="bash">root@dra7xx-evm # ls /usr/share/OpenCV/opencv_extra-master/ 3d android_manager_internal_docs classifiers gpu_demos_pack learning_opencv_v2 README.md testdata</source>

   The testsuite expects the `testdata` folder in `/usr/share/OpenCV`. Create a symbolic link to the `testdata` folder in `/usr/share/OpenCV` or move the `testdata` to the required location.

   <source lang="bash">root@dra7xx-evm:/usr/share/OpenCV# ln -s opencv_extra-master/testdata .</source>
2. Run the tests with the following commands

   <source lang="bash">root@dra7xx-evm:/usr/share/OpenCV# cd titestsuite/ root@dra7xx-evm:/usr/share/OpenCV/titestsuite# ls runtests setupEnv.sh test_core_ocl test_imgproc_ocl root@dra7xx-evm:/usr/share/OpenCV/titestsuite# source setupEnv.sh root@dra7xx-evm:/usr/share/OpenCV/titestsuite# ./runtests</source>

   The tests run for ~70 mins on DRA75x platform which has 2 DSP's. Processing is offloaded transparently to the DSP where possible.

For more information on OpenCV and accelerating OpenCV using the C66x DSP's via OpenCL, please refer to the below links.

1. TI Processors Wiki OpenCV page (http://processors.wiki.ti.com/index.php/OpenCV)
2. How to add a new DSP Kernel (http://processors.wiki.ti.com/index.php/OpenCV#OpenCV_OpenCL_related_framework_details:_how_to_add_new_DSP_kernel)
3. Creating OpenCL C Kernel optimized for C66 core (http://processors.wiki.ti.com/index.php/OpenCV#Creating_OpenCL_C_kernel_optimized_for_C66_core)

OpenCV can be rebuilt and installed via the yocto pacakge opencv.

Ducati performance measurement

Ducati performance is assessed by factors like IVA use, IVA fps, cpu load etc. To obtain these values changes are required in ipumm.

Changes to be done in ipumm Makefile

1. Open the Makefile in ipumm (under component-sources of installer).
2. Set PROFILER to ENABLE (by default it's DISABLE).
3. Make sure that TRACELEVEL is set to 0 during performance measurement (High trace level effects the performance and will not give the best results).
4. Open ipumm/src/ti/utils/profile.c and set kpi_control=7(default it is set to 0)
5. Change directory to parent directory of component-sources.
6. Run make ipumm
7. dra7-ipu2-fw.xem4 is created in ipumm directory. Copy this to the /lib/firmware of rootfs and reboot
8. Execute video decode usecase with gstreamer or viddec3test.
9. Execute cat /sys/kernel/debug/remoteproc/remoteproc1/trace0 . Traces will show the iva load, fps achieved, average and Peak MHz etc.

SOC Performance monitoring tools

NOTE! Support for SoC performance monitoring tool is available only upto K4.4 releases. Check the release notes to see if this feature is available on the Processor SDK Linux Automotive release you are using.

Introduction

The SOC Performance monitoring tools are a set of tools that are included in the default filesystem that allow the user to visualize various SOC parameters real-time on the screen. Currently, there are two tools and a suite of scripts and utilities to use them.

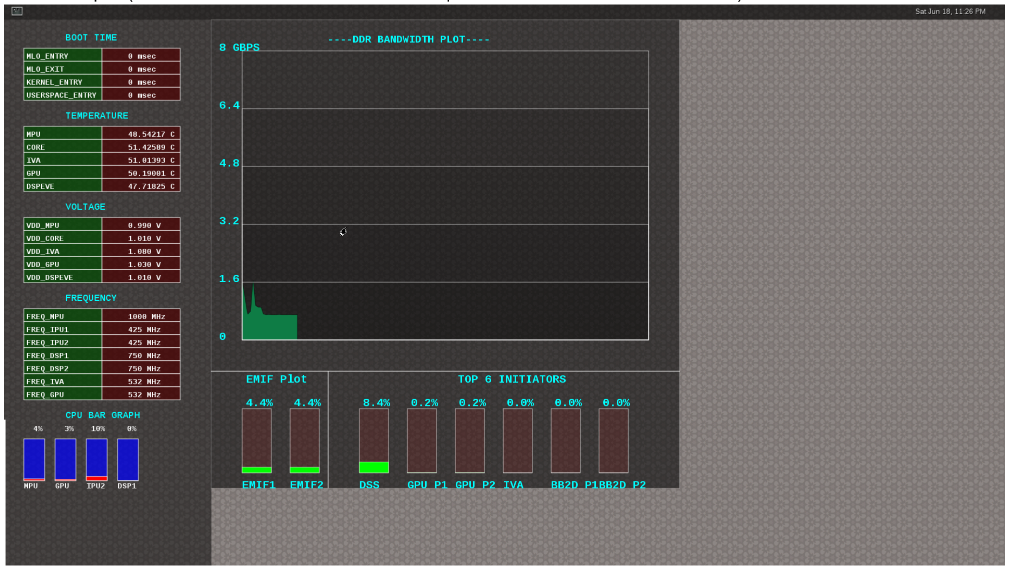
1. soc-performance-monitor
2. soc-ddr-bw-visualize

Both these applications are Wayland applications and need to be invoked after running Weston.

These tools bring in the capability to visualize the following:

1. DDR BW Utilization
   1. Overall DDR BW Usage
   2. Split of the traffic between the two EMIF's
   3. A real time "top" like functionality that depicts the list of "Top 6" initiators generating the traffic.
2. Voltage of the various rails
3. Frequency of the various cores
4. Temperature (read from on die temperature sensors)
5. CPU Load information of the various processor cores including the GPU and DSP.

6. Boot time results (requires rebuild of u-boot and kernel), refer instructions below.
7. Power plot (Will be available soon. Note that this requires board modification on the EVM)



BOOT TIME

MLO_ENTRY	0 msec
MLO_EXIT	0 msec
KERNEL_ENTRY	0 msec
USERSPACE_ENTRY	0 msec

TEMPERATURE

MPU	48.54217 C
CORE	51.42589 C
IVA	51.01393 C
GPU	50.19001 C
DSPEVE	47.71825 C

VOLTAGE

VDD_MPU	0.990 V
VDD_CORE	1.010 V
VDD_IVA	1.080 V
VDD_GPU	1.030 V
VDD_DSPEVE	1.010 V

FREQUENCY

FREQ_MPU	1000 MHz
FREQ_IPU1	425 MHz
FREQ_IPU2	425 MHz
FREQ_DSP1	750 MHz
FREQ_DSP2	750 MHz
FREQ_IVA	532 MHz
FREQ_GPU	532 MHz

Getting started

- Prepare the card with root filesystem provided with the installer.
- Boot up
- Start weston

```
target #  /etc/init.d/weston start
```

- Copy the required scripts into a temporary folder (this is to allow you to experiment with the settings later)

```
target # mkdir temp
target # cd temp
target # cp /etc/glsdkstatcoll/* .
target # cp /etc/visualization_scripts/* .
```

- You should see the following file in the directory after the above operation.

```
target # ls -al
drwxr-xr-x    2 root     root          4096 Mar 22 18:01 .
drwxr-xr-x    3 root     root          4096 Mar 22 18:01 ..
-rw-r--r--    1 root     root           114 Mar 22 18:01 config.ini
-rw-r--r--    1 root     root           265 Mar 22 18:01 dummy_boot_time_results.sh
-rw-r--r--    1 root     root           419 Mar 22 18:01 dummy_cpu_load.sh
-rw-r--r--    1 root     root           899 Mar 22 18:01 getFrequency.sh
-rw-r--r--    1 root     root          2293 Mar 22 18:01 getTemp.sh
-rw-r--r--    1 root     root           371 Mar 22 18:01 getVoltage.sh
-rw-r--r--    1 root     root           254 Mar 22 18:01 initiators.cfg
-rw-r--r--    1 root     root           143 Mar 22 18:01 list-boot-times.sh
-rw-r--r--    1 root     root           367 Mar 22 18:01 send_boot_times_to_monitor.sh
-rw-r--r--    1 root     root           496 Mar 22 18:01 soc_performance_monitor.cfg
-rw-r--r--    1 root     root           133 Mar 22 18:01 start_visualization_test.sh
```

- Running the soc-performance-monitor, this tool has two pre-requisites.

1. The name of the fifo configured in the file soc_performance_monitor.cfg needs to be created
2. The file soc_performance_monitor.cfg should be present in the current directory. This should be done in the above steps.

- Creating the fifo (mentioned in the soc_performance_monitor.cfg)

```
target # mkfifo /tmp/socfifo
```

- Run the tool for various performance metrics

```
target # soc-performance-monitor &
```

- Run the tool for DDR BW Visualization

```
target # mkfifo /tmp/statcollfifo
target # soc-ddr-bw-visualizer &
```

The following sections will talk about the how to populate the data into tools and further controls that are possible.

Quick guide to available plugins

Plugins are the entities (scripts/native binaries) that can be used to send commands to the SOC Performance Monitoring tools.

The main intent of this is to separate the visualization engine from the data collection part and allow full configuration of the application.

When the application (soc-performance-monitor) is invoked, it starts up with the default data which is set to zero. To populate the real values, the user can use the scripts provided in the prebuilt filesystem.

Temperature data

The temperature data is read from the on-die temperature registers and sent to the visualization tool. The file system comes with a script that does this functionality.

```
target # sh getTemp.sh
```

Invoking the above command will populate the temperature table with the current temperature.

Voltage data

The voltage data is read from the omapconf utility and then parsing out the required information to be later sent to the visualization tool. The file system comes with a script that does this functionality.

```
target # sh getVoltage.sh
```

Invoking the above command will populate the Temperature table with the configured voltage for the various rails.

Frequency data

The frequency data is read from the omapconf utility and then parsing out the required information to be later sent to the visualization tool. The file system comes with a script that does this functionality.

```
target # sh getFrequency.sh
```

Invoking the above command will populate the Frequency table with the configured frequency for the various cores.

CPU Load information

The CPU load information need individual plugin modules for each of the cores. This is envisioned to be different for different systems. The default filesystem contains the plugins required for reading the MPU(A15) and the GPU(SGX544 MP2). Other plugins for measuring the loads for the IPU1, IPU2, DSP1 and DSP2 will be available at a later time.

Measuring the MPU load

The filesystem is populated with a binary which is called "mpuload" that reads the /proc/stat interface and derives the load. The user can run the utility in the background with the

```
target # mpuload FIFO

Example usage:

target # mpuload /tmp/socfifo 1000 &
```

After running this binary the MPU load in the Bar Graph of the CPU load will be updated dynamically at an interval of 1 second.

Measuring the GPU load

The filesystem is populated with a binary called as "pvrscope" that reads the SGX registers via a library called libPVRScopeDeveloper.a This utility invokes the APIs provided by IMG as part of the Imagination PowerVR SDK and then populates the required FIFO.

Usage instructions:

```
target # pvrscope <option> <time_seconds>

options:
          -f    write into the FIFO (/tmp/socfifo)
          -c    output to console

time:
          1-n   specified in seconds
          0     run forever
```

After running this utility, the GPU load in the BAR Graph of the CPU load area will be updated at an interval of 1 second.

Boot time measurement

This feature will be provided at future release.

Order of execution

The performance visualization tools have to be executed in the following order.

- Launch weston
- Create required FIFOs
- Configure the .cfg file to suit the required settings
- Run the soc-performance-monitor and/or soc-ddr-bw-visualizer
- Run the plugins to populate data

Config file format

The config file has the following format.
There are 3 different kinds of sections that can be defined, please refer to the particular section for more details.

The generic format is:

```
[SECTION_NAME]
VALUE_1
VALUE_2
..
..
VALUE_N
SPECIAL VALUE
<blank line>
```

Types of sections

1. GLOBAL
2. TABLE
3. BAR GRAPH

GLOBAL section:

The SECTION_NAME is specified as GLOBAL followed by a sequence of key value pairs.

```
[GLOBAL]
KEY_1=VALUE_1
KEY_2=VALUE_2
..
..
KEY_n=VALUE_n
<blank>
```

Global configurations

The list of recognized global values are:

- REFRESH_TIME_USECS
- FIFO
- MAX_HEIGHT
- MAX_WIDTH
- X_POS
- Y_POS

REFRESH_TIME_USECS:

- This will dictate the interval at which the utility is going to run.
- The value is specified in micro seconds
- This value decides a major trade-off, lower rate will increase the CPU load and GPU load.
- The ideal value is about 100000 usecs

FIFO:

- The value of this field is the named pipe or fifo that can be used to communicate with the application.
- User would need to create a fifo (application will prompt if it doesn't exist)

MAX_HEIGHT, MAX_WIDTH:

- The width and height of the application.
- This can be adjusted based on the number of tables and bar graph entities.

X_POS, Y_POS:

- Decide the starting offset of the application.
- Note that there are commands to move the application (Refer commands section).

TABLE section:

The section name can be one of the following:

- BOOT_TIME
- TEMPERATURE
- VOLTAGE
- FREQUENCY

```
[TABLE_NAME]
 VALUE_1
 VALUE_2
 ..
 ..
 VALUE_N
TITLE="TABLE TITLE",UNIT="unit to be displayed"
<blank line>
```

NOTE: The TITLE=list is a list of comma separated values and TITLE and UNIT are the only supported values.

BAR GRAPH section:

This section is the simplest section and does not allow much configuration other than the names and the title.
It follows the following format:

```
[GRAPH_NAME]
 VALUE_1
 VALUE_2
 ..
 ..
 VALUE_N
 TITLE OF THE GRAPH
 <blank line>
```

Commands:

The **FIFO** can be used to communicate with the soc-performance-monitor application and pass data from the command line or from other applications.
There are a few commands that have been implemented to aid in modifying the running application via the FIFO.

The commands in general have the following format:

```
"INSTRUCTION: DATA_1 ... DATA_N"
```

and they can be sent to the soc-performance-monitor by simply doing an echo:

```
echo "INSTRUCTION: DATA_1 ... DATA_N" > FIFO
```

The currently supported list of supported commands are:

1. TABLE
2. CPULOAD

NOTE: To execute a sequence of commands in a sequence, it is advised that a delay of REFRESH_TIME_USECS be inserted between two commands.

TABLE command

The format of the TABLE command is:

```
"TABLE: ROW_NAME value unit"
```

When this command is issued, the tool will find a table entry with the ROW_NAME in Column 0 and then update the Column 1 of the table with "value unit"
If the ROW_NAME is not found, then this command will have no effect. Please note that this brings in a restriction that all the tables rows will need to have a unique name. In order to ensure this, the soc_performance_monitor.cfg file will have to be reviewed to ensure unique names.

Example: To update the FREQUENCY table for MPU, the user can send the following command:

```
echo "TABLE: FREQ_MPU 1500 MHz" > /tmp/socfifo
```

CPULOAD command

The format of the CPULOAD command is:

```
"CPULOAD: CORE_NAME value" > FIFO

 CORE_NAME has to be one of the names specified in the soc_performance_monitor.cfg.
 value is in the range 0 to 100
```

Usually, the CPULOAD command is invoked through an application monitors the load of a specific core.
In each system, the mechanism to retrieve the CPULOAD of a particular core can vary and it is for this reason that several plugins have been provided and serve as an example for further extension.

Example: To update the CPULOAD table for GPU, the user can send the following command:

```
echo "CPULOAD: GPU 87" > /tmp/socfifo
```

Executing in debug mode

To launch the application in debug mode for very verbose data on the internal working of the tool, launch the tool with the following option:

```
# soc-performance-monitor 1
```

Build instructions

The full source of the tool is available and the required recipes have been updated as part of the recipes and upstreamed to meta-arago.
Essentially, if the user builds the Yocto filesystem as documented in the SDG, the tool will get recompiled as part of it.

Configuration of the soc-ddr-bw-visualizer

Refer to #Using_the_statistics_collector_.28bandwidth_application.29

- The total time that the tool runs is configured using config.ini.
- To allow finer granularity of control to choose the initiators of interest, the user will have to modify the initiators.cfg.

The tool will have to relaunched for the new settings to take effect.

Additional Procedures

Build Environment Setup

NOTE: From this release, each component i.e kernel, u-boot or any userspace application should be cross-compiled

Cross Compiler setup

The cross compiler setup for the Rules.make is done as part of the setup.sh script in the SDK installation folder. The script for cross compiler ensures that the Linaro cross compiler toolchain is installed in the ${HOME} (or user specified) folder of the host machine.
Note: Please ensure that the PATH variable is set in your machine to point to the cross compiler setup.
To compile the code (if not using the top level Makefile to build kernel and u-boot), please ensure the environment variables ARCH and CROSS_COMPILE is set to arm and to the linaro cross compiler path respectively.

Rebuilding the SDK components

The installer contains a top level Makefile to allow the re-building of the various components.

Rebuild the SDK components by first entering the SDK directory using:

```
host $ cd ${INSTALL_DIR}
```

The installer Makefile has a number of build targets which allows you to rebuild the various components. For a complete list execute:

```
host $ make help
```

After that, each of the build targets listed by 'make help' can then be executed using:

```
host $ make <target>_clean
host $ make <target>
host $ make <target>_install
```

For example, to compile the Linux Kernel, you can use the following commands

```
host $ make linux_clean
host $ make linux
host $ make linux_install
```

In order to install the resulting binaries on your target, execute one of the "install" targets. Where the binaries are copied is controlled by the `EXEC_DIR` variable in `${INSTALL_DIR}/Rules.make`. By default, this variable is set up to point to your NFS mounted target file system when you execute the installer setup (`setup.sh`) script, but can be manually changed to fit your needs.

You can remove all components generated files at any time using:

```
host $ make clean
```

And you can rebuild all components using:

```
host $ make all
```

You can then install all the resulting target files using:

```
host $ make install
```

Creating your own Linux kernel image

The pre-built Linux kernel image (zImage) provided with in the SDK installation folder is compiled with a default configuration. You may want to change this configuration for your application, or even alter the kernel source itself. This section shows you how to recompile the Linux kernel provided with the SDK, and shows you how to boot it instead of the default Linux kernel image.

1. If you haven't already done so, follow the instructions in #Starting_your_software_development to setup your build environment.

2. Recompile the kernel provided with the SDK by executing the following:

```
host $ cd ${INSTALL_DIR}
host $ make linux_clean
host $ make linux
host $ make linux_install
```

3. You will need a way for the boot loader (u-boot) to be able to reach your new zImage. Copy the new zImage that is generated in arch/arm/boot/ directory to the rootfs/boot folder.

4. Copy the exported Linux kernel modules from the EXEC_DIR to the /lib/modules directory to the root file system

Using the statistics collector (bandwidth application)

NOTE! Support for SoC performance monitoring tool is available only upto K4.4 releases. Check the release notes to see if this feature is available on the Processor SDK Linux Automotive release you are using.

The default filesystem includes a utility to determine the bandwidth statistics of each initiator in the J6 SoC.

This section will give an overview of how the user can use this effectively.

Target side

The tool is called glsdkstatcoll and is present under the /usr/bin folder. This tool requires a config file as an input which is also part of the filesystem

Copy the sample config file into your working directory:

```
target # cp /etc/glsdkstatcoll/* .
target # glsdkstatcoll -h
```

There are two levels of configuration that the tool allows:

1. Configuration of the interval and the total time can be controlled via **config.ini**
2. The list of initiators can be configured using the **initiators.cfg** file.

Note that both these files have to be present under the current directory.

After configuration, the user can launch the tool as below:

```
target # glsdkstatcoll -f config.ini
```

The tool will run for the specified duration in seconds "TOTAL_TIME" in config.ini and will sample every "INTERVSAL_US" microseconds.

The results from this tool will be available in statcollector.csv

```
target # ls -al statcollector.csv
```

Host side

To visualize this data, first install matplotlib:

```
host # sudo apt-get install python-matplotlib
host # git clone git://git.ti.com/glsdk/example-applications.git
host # cd example-applications/bandwidth-tool/host
```

Since the analysis of the data will be done over many iterations, the user has been provided with a config file in which some basic configuration can be done.

```
host # vi configstat.ini # Set the IP Address of the target and also the directory where the statcollector.csv was generated
```

Once the above setup is completed, the data can be visualized as below:

```
host # python statcoll_plot.py
```

This will tool will do the following:

1. Fetch the file from the target
2. Figure out all the initiators that did not have any traffic and exclude them from the plot.
3. Plot the total traffic from EMIF1_SYS and EMIF2_SYS ports
4. Display the peak, average and average(active) traffic on the various ports.
5. Note that if the INTERVAL_US is other than 30000 microseconds, user will have to edit the statcoll_plot.py

If the target does not have the Ethernet capabilities, then the file can be fetched and the same tool can be run like below:

```
host # python statcoll_plot.py -f <name of the file>
```

There are some further enhancements planned, they will be posted in the git place holder.

Setting up Tera Term

Tera Term is a commonly used terminal program on Windows. If you prefer to use it instead of Minicom, you can follow these steps to set it up.

1. Download Tera Term from this location (http://logmett.com/index.php?/download/tera-term-477-freeware.html), and start the application.

2. In the menu select *Setup->General...* and set:

```
Default port: COM1
```

3. In the menu select *Setup->Serial Port...* and set the following:

```
Port:         COM1
Baud rate:    115200
Data:         8 bits
Parity:       none
Stop:         1 bit
Flow control: none
```

{{ 1. switchcategory:MultiCore= • For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum • For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **Processor SDK Linux Automotive Software Developers Guide** here.	Keystone= • For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum • For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum Please post only comments related to the article **Processor SDK Linux Automotive Software Developers Guide** here.	C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article* ***Processor SDK Linux Automotive Software Developers Guide*** *here.*	DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article* ***Processor SDK Linux Automotive Software Developers Guide*** *here.*	MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article* ***Processor SDK Linux Automotive Software Developers Guide*** *here.*	OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***Processor SDK Linux Automotive Software Developers Guide*** *here.*	OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article* ***Processor SDK Linux Automotive Software Developers Guide*** *here.*	MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article* ***Processor SDK Linux Automotive Software Developers Guide*** *here.*	*For technical s please post yo questions at http://e2e.ti.co Please post o comments abo article* ***Proces SDK Linux Automotive S Developers G*** *here.* }}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters

DLP & MEMS
High-Reliability
Interface
Logic
Power Management

Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity

Retrieved from "https://processors.wiki.ti.com/index.php?title=Processor_SDK_Linux_Automotive_Software_Developers_Guide&oldid=237122"

This page was last edited on 13 December 2019, at 07:00.