MCSDK 3.x

User Guide

PDF generated using the open source mwlib toolkit. See http://code.pediapress.com/ for more information.
PDF generated at: Sun, 20 Sep 2015 19:43:40 CDT

Contents

References

MCSDK User Guide for KeyStone II

Multicore Software Development Kit User Guide (KeyStone II Devices)

Contents

(*Last updated: 09/20/2015*)

If you are reading this on the PDF version of this guide, the most recent version can be found online at: http://processors.wiki.ti.com/index.php/MCSDK_User_Guide_for_KeyStone_II

MCSDK UG Chapter Getting Started

MCSDK User Guide: Getting Started with the MCSDK

Last updated: **09/20/2015**

Overview

The Multicore Software Development Kit (MCSDK) provides foundational software for TI KeyStone II device platforms. It encapsulates a collection of software elements and tools intended to enable customer application development and migration.

This chapter provides information on installing the MCSDK, loading the EVM flash images via JTAG, and running the out-of-box demonstration application using the updated software.

By the end of this *Getting Started* chapter, the user will have

- Installed Code Composer Studio
- Installed latest Emulation package
- Installed the MCSDK software
- Update EVM flash with release images
- Executed the out-of-box demonstration application using latest images

Useful Tip

The EVM comes with a *Quick Start Guide* that provides instructions to setup hardware and boot the out-of-box demonstration loaded on the EVM.

Acronyms

The following acronyms are used throughout this chapter.

Acronym	Meaning
CCS	Texas Instruments Code Composer Studio
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol
DSP	Digital Signal Processor
EVM	Evaluation Module, hardware platform containing the Texas Instruments DSP
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
JTAG	Joint Test Action Group
MCSDK	Texas Instruments Multi-Core Software Development Kit
PDK	Texas Instruments Programmers Development Kit
RAM	Random Access Memory
TI	Texas Instruments
UBIFS	Unsorted Block Image File System

Supported Devices and Platforms

This release supports the following Texas Instrument devices/platforms:

Platform	Supported Devices	Supported EVM
[K2H [1]]	TCI6636K2H [2], 66AK2H06 [3], 66AK2H12 [4], 66AK2H14 [5]	XTCIEVMK2X, EVMK2H [6]
[K2K [1]]	TCI6638K2K [7]	XTCIEVMK2X
[K2E [1]]	66AK2E02 [8], 66AK2E05 [9], AM5KE02 [10], AM5K2E04 [11]	EVMK2E [12]
[K2L [1]]	TCI6630K2L [13], 66AK2L06 [14]	TCIEVMK2L

What's in this Kit?

The kit contains

- **EVM Board**: This board contains a Multicore System-on-Chip
- **Universal Power Supply**: Both U.S. and European power specifications are supported
- **Cables**: USB, Serial, and Ethernet cables are included to allow for host development
- **Software**: Software installers, documentation, factory images
- **EVM Quick Start Guide**: Single page guide to get started run demos pre-flashed on board

The software media (e.g., DVD, SD card, or USB) contains

- Multicore Software Development Kit source packages
- Code Composer Studio installer
- Factory image restoration process
- Hardware documentation

All EVMs have both NAND and NOR flash, though they may be different in size. Typical use case is to load uboot on NOR and boot from NOR. Due to greater NAND flash size, the software file system is loaded on NAND in a UBIFS format.

Getting Started

Setup Hardware

The EVM provides the ability to utilize a variety of capabilities of the SoC. Follow instructions in one of the links below that correspond to the EVM that you are using for information on configuration, board firmware updates, and further pertinent information:

- XTCIEVMK2X, EVMK2H : Use EVMK2H Hardware Setup [15]
- EVMK2E: Use EVMK2E Hardware Setup [16]
- TCIEVMK2L : Use TCIEVMK2L Hardware Setup [17]

Setup Software

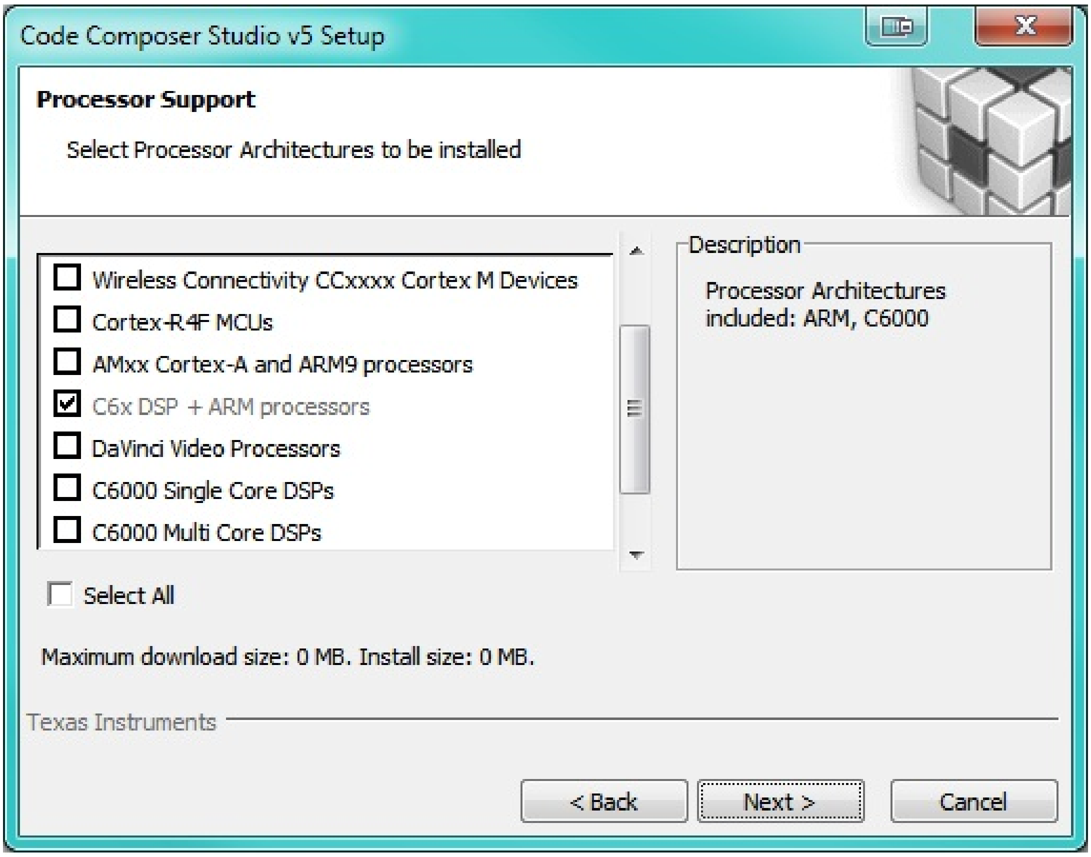
Code Composer Studio

The MCSDK uses *Code Composer Studio* as the host integrated development environment for development and debug. Install the version of CCSv5 indicated in the MCSDK Release Notes. This is the version of CCS that was validated with the software provided in the MCSDK and will provide the best user experience. To install CCS, please refer to the instructions provided in the CCSv5 Getting Started Guide [18].

Note: MCSDK version 3.1.2 an later have been validated with CCS version 6.0.1. The instructions can be found in the CCSv6 Getting Started Guide [19].

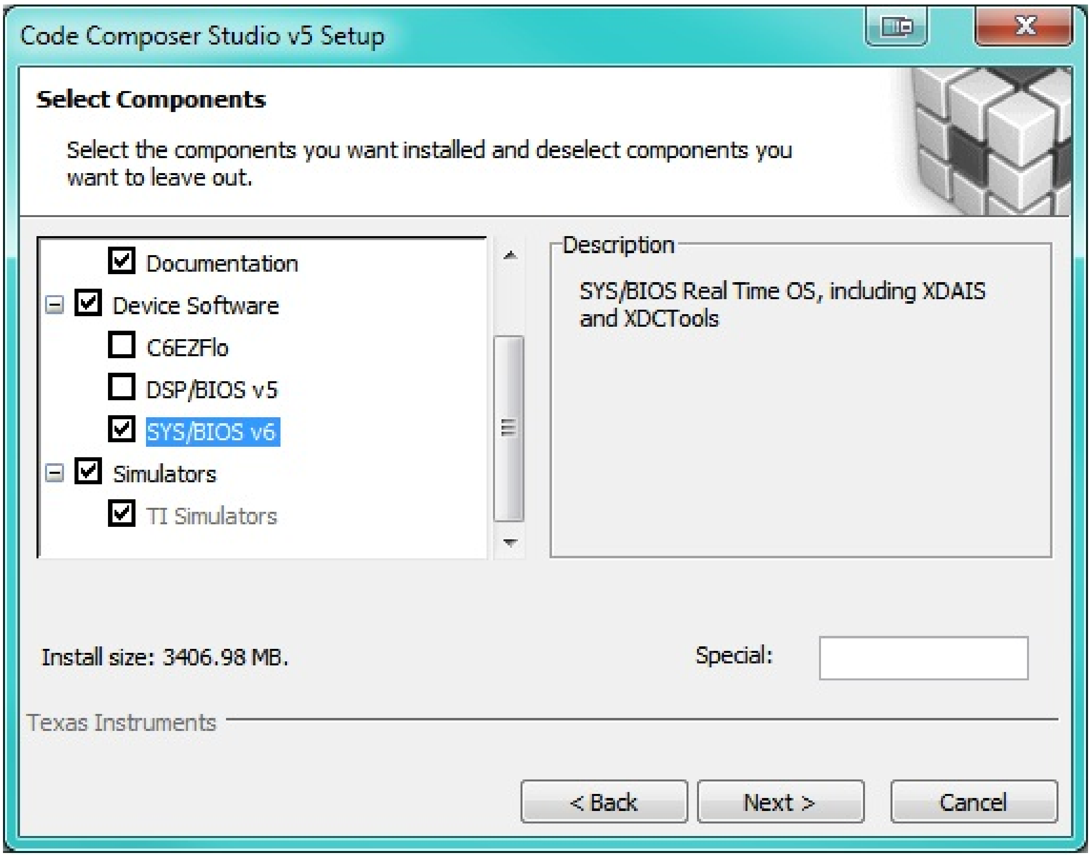
When installing CCS you can choose to control what is installed for processor architecture and components. The minimum required for MCSDK are

Processor Architecture option:


Code Composer Studio v5 Setup
Processor Support
Select Processor Architectures to be installed
□ Wireless Connectivity CCxxxx Cortex M Devices
□ Cortex-R4F MCUs
□ AMxx Cortex-A and ARM9 processors
☑ C6x DSP + ARM processors
□ DaVinci Video Processors
□ C6000 Single Core DSPs
□ C6000 Multi Core DSPs
Description
Processor Architectures included: ARM, C6000
□ Select All
Maximum download size: 0 MB. Install size: 0 MB.
Texas Instruments
< Back
Next >
Cancel

Note: If CCS v5.5 is used, please select "C6000 Multi Core DSPs" option, and if ARM environment is desired, please also select "Sitara AMxxxx processors" so the ARM compiler will be installed. Though Sitara A8 ARM compiler works for A15 family processor, it is suggested that Linaro GCC compiler should be installed and used for ARM A15 family processor.

Component option:



Code Composer Studio v5 Setup

Select Components

Select the components you want installed and deselect components you want to leave out.

- ☑ Documentation
- ⊟ ☑ Device Software
  - ☐ C6EZFlo
  - ☐ DSP/BIOS v5
  - ☑ SYS/BIOS v6
- ⊟ ☑ Simulators
  - ☑ TI Simulators

Description

SYS/BIOS Real Time OS, including XDAIS and XDCTools

Install size: 3406.98 MB.

Special:

Note: There is a known issue with ARM debug on CCS v5.5, see this forum post [20] for instructions on applying a patch to fix the issue.

Emulator Support

The GEL and XML files for connecting a JTAG to the KeyStone devices supported in MCSDK are provided in the *TI KeyStone2 Emupack* release. See the Downloads [21] chapter for information on how to get the latest version of this package. This software package is required for using an emulator to debug software over JTAG as well as the program_evm tool to update the EVM flash with latest images.

Multicore Software Development Kit

The final step is to install the MCSDK. See the Downloads [21] chapter for information on how to get the latest version of this software. The release contents are delivered as installers and source tarball. Some notes:

- To avoid manually updating file locations, it is recommended to install the MCSDK in the same directory as CCSv5.
- For a Linux host, run the mcsdk_<build-id>_setuplinux.bin executable. Starting with MCSDK 3.1.0, the GUI interface has been replaced with a command line installer.
- For a Windows host, run the mcsdk_<build-id>_setupwin32.exe executable. Starting with MCSDK 3.1.0, this package is focused on DSP source; use the above Linux package for Linux sources and images.
- Starting with MCSDK 3.1.x, the release includes a mcsdk_<build-id>.tar.gz, which can be untarred on any platform.
- Some of the components installed by CCS may be updated by the MCSDK installer (e.g., SYS/BIOS); see the MCSDK release notes for a mapping of each component to the software package (CCS or MCSDK).

Note: For *MCSDK 3.0.x* Linux installer, mcsdk_x_xx_xx_xx_setuplinux.bin, is a 32-bit binary. To run this installer on a 64-bit machine, an adaption/compatibility library, ia32-libs, needs to be installed. On Ubuntu 12.04 machine, the following command is used to install ia32-libs:

```
   sudo apt-get install ia32-libs
```

Note: For *MCSDK 3.1.x* or later releases: The source tarball included in the release can be installed on a Linux machine using the steps as follows. To see the list of packages in the MCSDK release tar ball:

```
   tar tzvf mcsdk_<build-id>.tar.gz --exclude="*/*"
```

To install the full package in a specified directory (Say ~/ti)

```
  tar xzvf mcsdk_<build-id>.tar.gz -C ~/ti
```

This will install the different packages under ~/ti, with their respective package_<version> directories). In addition, the XDC package xdc_<version>_core6x in the tar package is meant for ARM native compilation only. If using x86 based host, this can be removed to avoid issues with XDC compilation.

To install a specific package only from the tar ball:

```
  tar xzvf mcsdk_<build-id>.tar.gz -C ~/ti --wildcards "mcsdk_linux*
```

Update EVM On-Board Flash

To update on-board flash media on the EVM (NOR, NAND, EEPROM), use the *program_evm* script located in the mcsdk_bios_3_xx_xx_xx/tools/ directory that updates all EVM images automatically via command line. The script uses CCS DSS scripts to load and run a utility from a SoC C6x DSP core to load and burn flash.

The steps for this are described in the Program EVM User Guide. To use *program_evm* you need to install CCS and have a connection between your host PC and the EVM via JTAG. When running the *program_evm*, make sure the CCS GUI is not connected to the target.

Useful Tip

The images that are loaded by *program_evm* are kept in the /program_evm/binaries/evmk2x/ directory. Since you can substitute any image here, this is a quick way to update flash with custom images.

Run Out of Box Demonstrations

Please follow the below steps to run the Matrix Application demos, the demos can be launched by a GUI Matrix launcher from a remote browser.

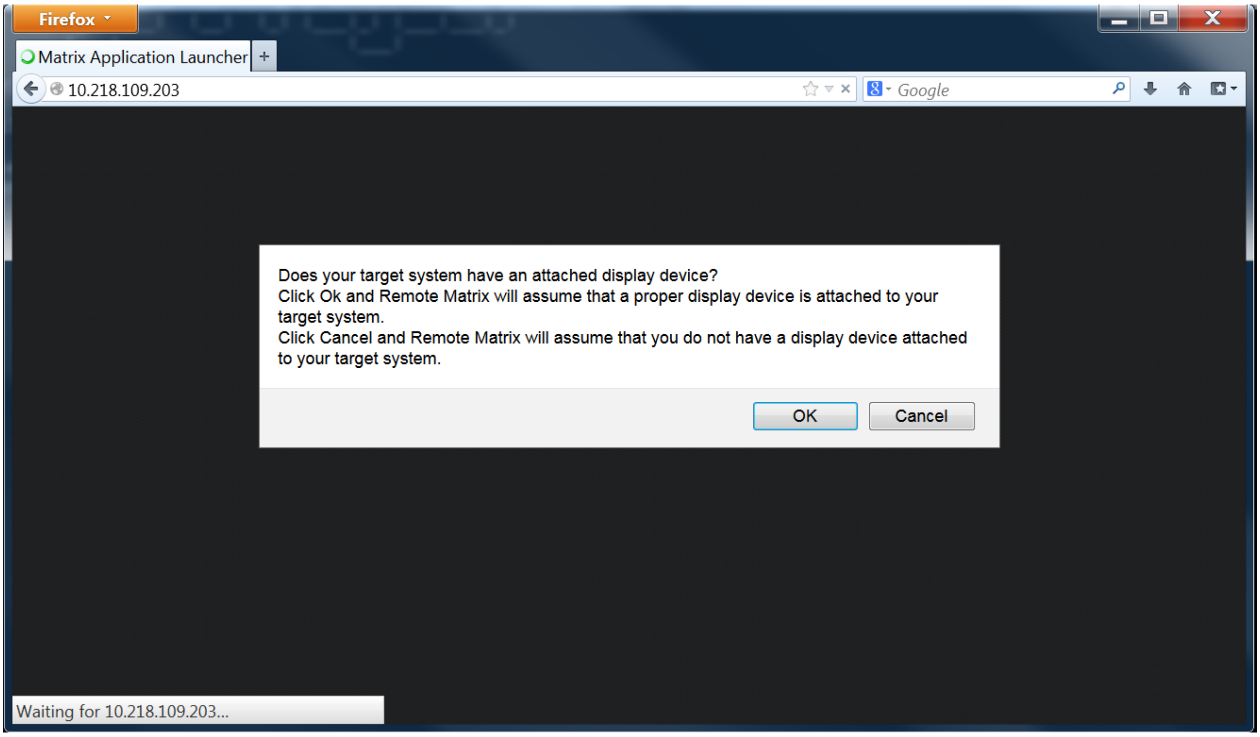
1. Configure uboot to set the DSP load reserve area to match pre-build DSP libraries:

```
env default -f -a
setenv mem_reserve 1536M
saveenv
boot
```

2. Connect the EVM Ethernet port 0 to a corporate or local network with DHCP server running, when the Linux kernel boots up, the rootfs start up scripts will get an IP address from the DHCP server and print the IP address to the EVM on-board LCD.

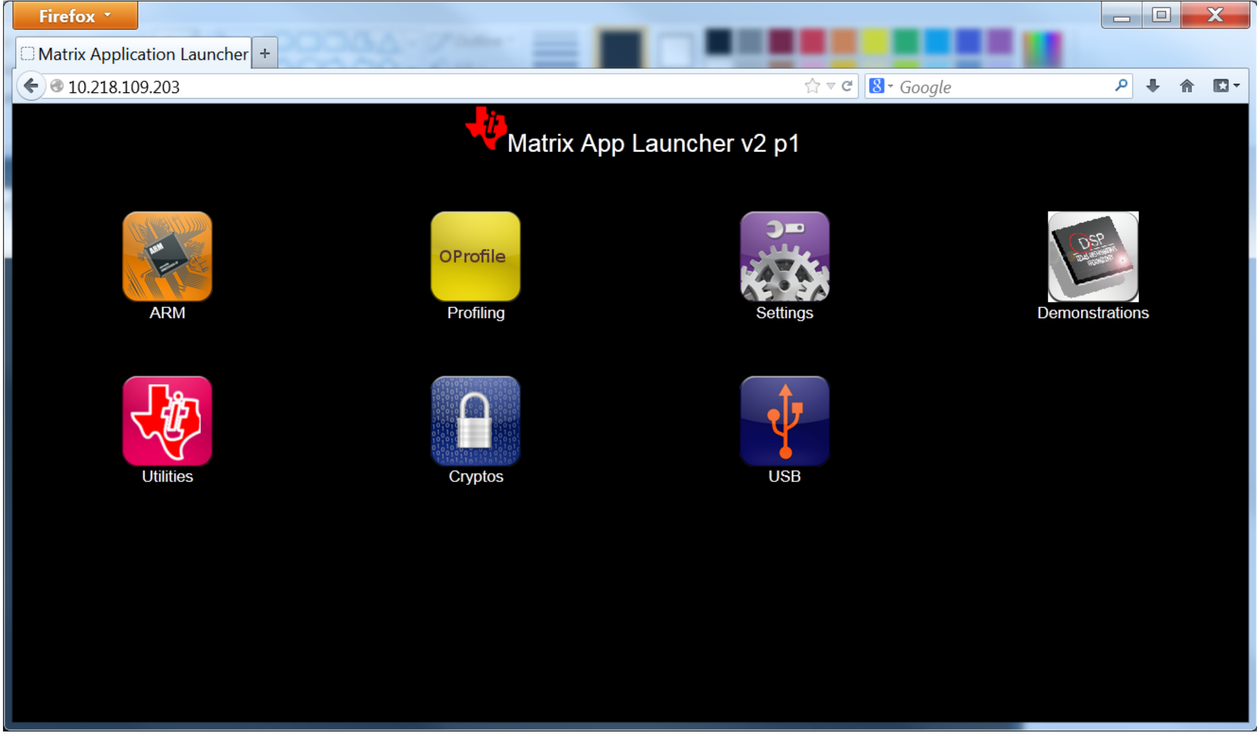
3. Open an Internet browser (e.g. Mozilla Firefox) on a remote computer that connects with the same network as the EVM.

4. Type the IP address displayed on EVM LCD to the browser and click cancel button to launch the Matrix launcher in the remote access mode instead of on the on-board display device.



Does your target system have an attached display device?
Click Ok and Remote Matrix will assume that a proper display device is attached to your target system.
Click Cancel and Remote Matrix will assume that you do not have a display device attached to your target system.

5. Click into each Matrix app icon on the launcher page to run the different demos.



Note: Linux kernel vmlinux image currently is not included in the rootfs due to the NAND flash size limitation, to run the Oprofile without profiling the kernel, enter the following command in the Linux command shell:

- opcontrol --no-vmlinux

Please refer to the Oprofile Init demo instructions for details.

Next Steps

See Software Design Guide [22] on timeline guidance for your Keystone II software development.
See the "Developing with MCSDK" chapter of this User Guide on a section to create a simple "Hello World" application. More advanced topics are covered in different sections of that chapter.

How To

Manually get IP address from Linux

If IP address is not known, please connect USB console and get IP address form Linux using *ifconfig* command. Log in as root (no password) if prompt for user.

Set up TFTP Server

If loading images using TFTP, an example on Ubuntu TFTP server set up can be found here [23].

On Ubuntu 12.04, it is recommended to use tftpd-hpa server as the TFTP server since the default one has shown some slower transfer speeds.

```
sudo  apt-get install tftpd-hpa tftp-hpa xinetd
```

You will need to update the /etc/default/tftpd-hpa config file

```
# /etc/default/tftpd-hpa
```

```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/tftpboot"
TFTP_ADDRESS="0.0.0.0:69"
TFTP_OPTIONS="--secure"
```

U-boot quick update for K2H EVM

The following instructions is an example of loading U-boot to a **K2H EVM** in a **Linux** environment.

The main assumption here is that you are familiar with the overall process, but you should review the following related subjects:

http://processors.wiki.ti.com/index.php/MCSDK_UG_Chapter_Exploring#Loading_and_running_U-Boot_on_EVM_through_SPI_Boot

http://processors.wiki.ti.com/index.php/Program_EVM_UG

Procedures

[1] Install Code Composer Studio (**CCS**) from TI download website to your Linux PC.

[2] Install **CCS EMUPACK files** from: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/index_FDS.html

[3] Install **MCSDK files** from: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/index_FDS.html

[4] Note: You may need to change the MCSDK file ownership and rights for all the installed files in your environment.

[5] Put the board in **no boot mode** and connect emulator USB-mini cable to the top USB daughter card (emulator board).

```
You should record your SW1 Normal switch settings on the board and then
push all the switches away from the edge of the board for programming U-boot
into NOR in No Boot Mode.

For Normal Operations:
SW 1   SW2   SW3   Sw4
OFF    OFF   ON    OFF

For No Boot Mode operations:
SW 1   SW2   SW3   Sw4
ON     ON    ON    ON  (Note: Away from the edge of the board)
```

[6] cd **<MCSDK_INSTALL_PATH>/**mcsdk_bios_3_01_xx_yy/tools/program_evm/

[7] Execute the following command in the **program_evm** folder:

```
<CCS_INSTALL_PATH>/ccsvX/ccs_base/scripting/bin/dss.sh program_evm.js evmk2h nor
```

[8] **Power off EVM and place the configuration switches back to the normal mode** recorded earlier.

```
Typically from left to right OFF, OFF, ON, OFF
Connect your serial port to Normal console port (marked as SOC which is closest
to the board edge) and power on the EVM to see the normal boot operations.
```

Address basic issues with out of box demonstration

If you run into any issues with running out of box demonstration, be sure the following was done properly:

- BMC update from HW Setup Guide
- UCD update from HW Setup Guide
- Correct boot mode setting from HW Setup Guide (typically SPI NOR Little Endian)

EVMK2H Hardware Setup

Getting Started with MCSDK: EVMK2H Hardware Setup

Last updated: **09/20/2015**

Hardware Setup

Warning: The EVM board is sensitive to electrostatic discharges (ESD). Use a grounding strap or other device to prevent damaging the board. Be sure to connect communication cables before applying power to any equipment.

FTDI Driver Installation on PC Host

The K2 EVM has a FTDI FT2332HL device on board. A driver must be installed on the PC Host in order to be able to communicate with the EVM using the FTDI mini-USB connector located under the mini-USB connector of the emulator daughter card. The driver can be downloaded from here FTDI Driver [1].

Note: Before testing the usb connection, make sure that the mini-usb cable is plugged into the port on the base board. (and not connected to the daughter card).

After installing the driver and connecting the USB cable, two COM ports should be visible in the list of COM ports available to connect to in the PC Host terminal console. The lower COM port corresponds to the SoC UART and the higher one corresponds to the MCU UART.

BMC Version Check and Update

Read BMC_In-Field_Update [2] to check BMC version and update if necessary.

UCD Power Management Modules In-Field Update

There are three power management modules (sometimes called the UCDs) located on the EVM. Each module can be identified by it's address: 104(68h), 52(34h), and 78(4Eh). Each module contains non-volatile registers that determine it's operation. It may be necessary to update these registers after the board has been shipped. This update can be performed through the BMC, which can issue commands to the UCD modules to update the register settings. The **Power Management Configuration Update Tool** performs the task of sending commands to the BMC to get the current module versions, and perform updates using configuration files. The latest version of the tool is available here [3], along with instructions on using the tool, and the latest configuration files (txt files). Please follow the instructions provided to check the current module versions, and update them accordingly.

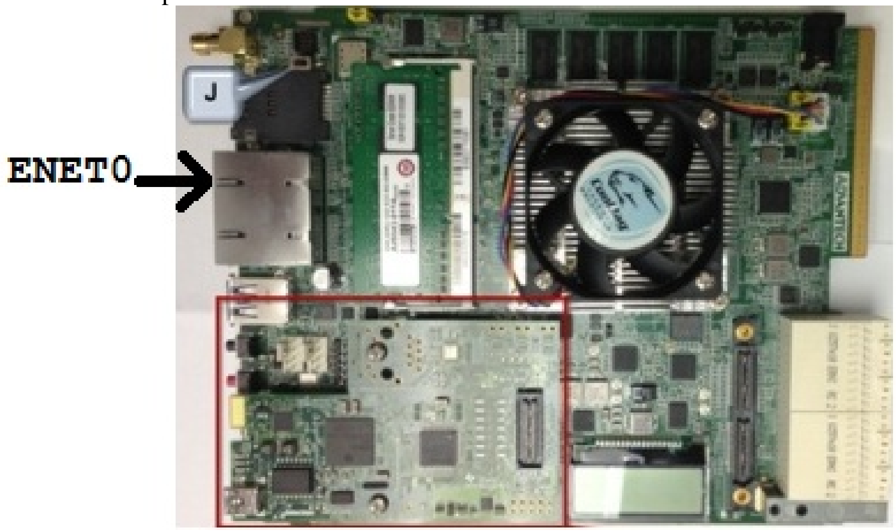
Note: The DIP switch configuration of the board when running the update is irrelevant.

Note: BMC versions 1.0.1.3a and earlier will not work properly with the **Get Versions** feature of the Update Tool. Upgrade to a more recent version of the BMC to use this functionality.

Attach the Ethernet cable

Using the Ethernet cable supplied, connect one end of the cable to the Ethernet Port 0 (marked ENET0 on the board) on the EVM and the other end to your PC.

This picture shows which Ethernet Port is 0:



Connect the JTAG interface

Use the USB to USB mini-B cable provided. Connect the USB mini-B connector to the USB mini-B interface on the XDS-2xx daughter card on the EVM, and the USB connector to your PC. This enables XDS-2xx emulation and is directly useable by CCS. If you are using a different JTAG, connect it now.

Set the boot mode switch SW1

No Boot/JTAG DSP Little Endian Boot mode

For **Rev 0B EVM**, the setting is as follows

```
SW1 - 4(ON) 3(ON) 2(ON) 1(OFF)
```

For **Rev 1.0 EVM**, the setting is as follows:

```
SW1 - 1(OFF) 2(OFF) 3(OFF) 4(ON)
```

SPI Little Endian Boot mode

For **Rev 0B EVM**, the setting is as follows:

```
SW1 - 4(ON) 3(ON) 2(OFF) 1(ON)
```

For **Rev 1.0 EVM**, the setting is as follows:

```
SW1 - 1(OFF) 2(OFF) 3(ON) 4(OFF)
```

DDR Configuration (Rev 0B EVM only)

For **Rev 0B EVM**, the following procedure is required for proper DDR configuration:

- Connect the MCU UART port to PC using the serial cable provided with the EVM. The MCU UART port is the 4-pin white connector farthest from the edge of the EVM. Alternatively it is also possible to connect a mini-USB cable to the FTDI mini-USB connector of the EVM. This will provide access to both the SoC and the MCU

UART ports.

- Start Tera Term or Hyper terminal and set to 115200 board rate, 8-bit data, 1-bit stop and no parity/flow control.
- Power on the EVM. MCU UART console will show user prompt once MCU boot up is complete. Type the following commands at the console to setup DDR3A. Ethernet requires DDR3A and will not work with DDR3B which is default in Rev 0B EVMs.

```
BMC> setboot 100001
BMC> fullrst
```

Attach the serial port cable to the SoC UART port

Connect the SoC UART port to PC using the serial cable provided with the EVM. The SoC UART port is the 4-pin white connector closest to the edge of the EVM.

Start Tera Term or Hyper terminal and set to 115200 board rate, 8-bit data, 1-bit stop and no parity/flow control.

Connect the power cable

Connect the power cable to the EVM power jack on the board. To be ESD safe, plug in the other end of the power cable only after you have connected the power cord to the board. Then turn on the board.

BMC In-Field Update

BMC, or Board Management Controller, takes care of the power, clocks, resets, bootmodes, etc. of the EVM.

For Rev1.0 EVMs an in-field update may be necessary as a very small quantity were delivered with an old revision of the BMC. If your EVM is using version 1.0.1.3 then it should be updated to version 1.0.1.3a. The update corrects the way that the boot mode pins are interpreted.

You can check the version by:

1. Opening a hyperterminal or another similar type of console application.

2. Set COM Port to higher value

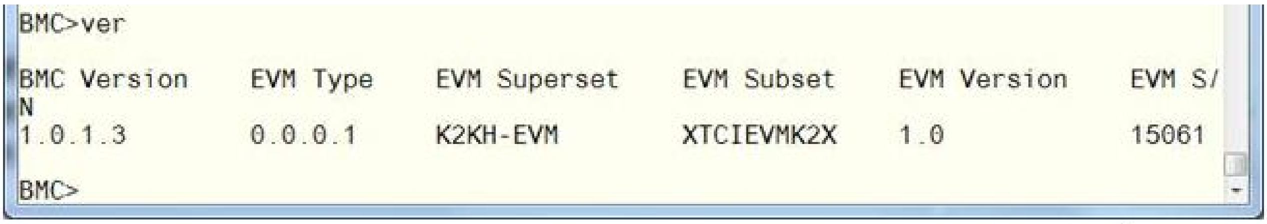
- When you connect to FTDI mini-USB on the EVM it will provide 2 COM port connections, one to the SOC UART and one to BMC UART.
- The SOC UART will always be the lowest value COM port, for example COM5, and the BMC UART will always be the higher value COM port, for example COM6. (Actual COM PORT values will vary).

3. Set COM port properties appopriately:

- Baud Rate or Bits per second: 115200
- Data Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

4. At BMC prompt typer 'ver' (no quotes)

5. Check BMC Version



```
BMC>ver

BMC Version     EVM Type    EVM Superset    EVM Subset    EVM Version    EVM S/
N
1.0.1.3         0.0.0.1     K2KH-EVM        XTCIEVMK2X    1.0            15061

BMC>
```

Bmc ver screenshot.JPG

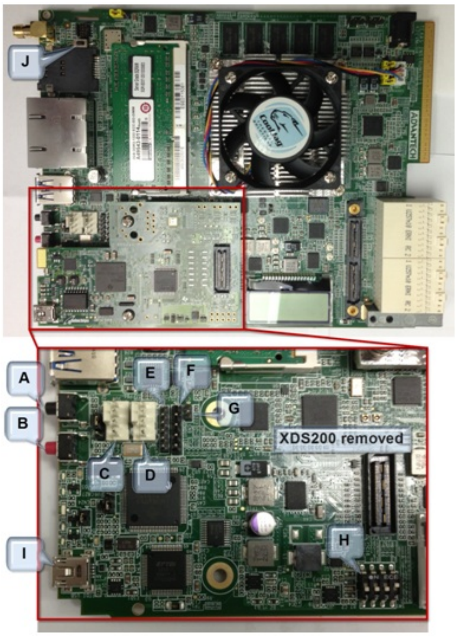
If an in-field update is needed, downloaded the latest version here [3] (labeled **Board Management Controller Binaries (BMC)**) and continue with the following steps.

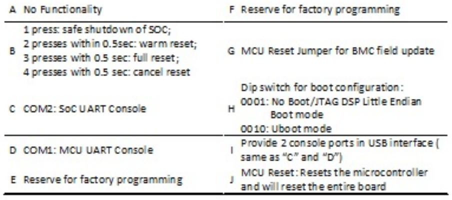
Prepare EVM for in-field update:

1. Remove power to the EVM.
2. Set boot mode to "No Boot mode" (See link [4])
3. Remove the MCU_BOOTSELECT (CN9) jumper (see picture below for location of jumper: Referred as "**G: MCU Reset Jumper for BMC field update**").
4. Make sure your USB cable is connected to FTDI mini-USB (not XDS200 Emulator USB) OR connect 4pin UART cable to COM1: MCU UART connector.
5. Make sure no HyperTerminal/Console connected to BMC COM port are open or active.
6. Use the LM Flash Programmer (available here [5]) to update the firmware, as detailed in the steps below.

Perform in-field update:

1. Apply power to the EVM. No LED's will be illuminated and no LCD backlight or characters will be on because the BMC is waiting for a command rather than executing from Flash.
2. Open the LM Flash programmer utility. (Default location Start Menu -> All Programs -> Texas Instruments -> Stellaris -> LM Flash Programmer -> LM Flash Programmer)
3. In the LM Flash Programmer Utility ‘Configuration’ tab, in the interface section, select ‘Serial (UART)’ from the drop-down box on the left.
4. Select the BMC COM Port (the same COM port used to issue the ver command earlier), and set the ‘Baud Rate’ to 115200.
5. Set ‘Transfer Size’ to 60, and make sure ‘Disable Auto Baud Support’ is unchecked.
6. In the ‘Program’ tab, Browse to the location of the binary file containing the firmware update, and select it.
7. Leave all other options as default, and press the ‘Program’ button.
8. After the programming is complete, power off the board.
9. Reconnect the jumper.
10. Open the HyperTerminal/Console for the BMC COM port.
11. Apply power to the EVM. When BMC completes initialization of board it will show latest version of BMC in Console.
12. If step 9 was done after power was applied, just type "ver" at BMC prompt.





A	No Functionality	F	Reserve for factory programming

B	1 press: safe shutdown of SOC;
	2 presses within 0.5sec: warm reset;
	3 presses with 0.5 sec: full reset;
	4 presses with 0.5 sec: cancel reset
		G	MCU Reset Jumper for BMC field update

C	COM2: SoC UART Console	H	Dip switch for boot configuration: 0001: No Boot/JTAG DSP Little Endian Boot mode 0010: Uboot mode
D	COM1: MCU UART Console	I	Provide 2 console ports in USB interface (same as "C" and "D")
E	Reserve for factory programming	J	MCU Reset: Resets the microcontroller and will reset the entire board

DIP Switch and Bootmode Configurations

Rev 1.0 EVM SW1 switch Bootmode Configuration

The table below shows the bootmode combinations for the BMC v1.0.1.3a.

DIP Switch (p1, p2, p3, p4)	Bootmode
0000	ARM NAND
0001	DSP no-boot
0010	ARM SPI
0011	ARM I2C
0100	ARM UART
0101	Reserved
0110	Reserved
0111	Reserved
1000	Reserved
1001	Reserved[1]
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved

1110	Reserved
1111	Reserved

[1] In revision BMC v1.0.1.4 this is 10 MHz SPI NOR. This will not continue in future versions.

Changing the Bootmode

In BMC v1.0.1.x the only way to use a bootmode that is not supported by the DIP switch combinations is to use the 'setboot' and 'fullrst' commands. To set the bootmode use the setboot command, which takes a 32 bit value in hex as its only argument:

```
setboot 00110CE7
```

Then use the fullrst command to boot the SoC into this bootmode:

```
fullrst
```

This process is volatile, and will have to be repeated every time the board is power cycled.

In BMC v1.0.2.x the setboot command has been removed. It has been replaced with the bootmode command, which performs various functions depending on the way in which the command is used. The command works with 16 bootmodes, which are representative of the various DIP switch combinations; the bootmodes are numbered 0 - 15. Bootmodes 8 - 15 are User-Defined, and may be altered and stored using the command (explained below). Each bootmode consists of a title, a high value, and a low value. The high value is currently not used. The low value is a 32 bit value in hex, and is the same as the value previously used by setboot. The bits of low value (and the setboot argument) are shown in the table below).

Bit	Devstat Bit	Config Pin Function	Normal Pin Function	Comments
31				Reserved
30				Reserved
29				Reserved
28				Reserved
27				Reserved
26				Reserved
25		PACLKSEL	PACLKSEL	
24		CORECLKSEL	CORECLKSEL	
23				Reserved
22		AVSIFSEL1	TIMI1	Reserved: EVM forces these bits to strap values during reset
21		AVSIFSEL0	TIMI0	Reserved: EVM forces these bits to strap values during reset
20		DDR3_REMAP_EN	GPIO16	
19		ARM_LENDIAN	GPIO15	0 = little, 1 = is not supported; do in SW
18		MAINPLLODSEL	CORESEL3	
17		ARMAVSSHARED	CORESEL2	
16	16	BOOTMODE15	CORESEL2	
15	15	BOOTMODE14	CORESEL1	Element
14	14	BOOTMODE13	CORESEL0	
13	13	BOOTMODE12	GPIO13	

12	12	BOOTMODE11	GPIO12	
11	11	BOOTMODE10	GPIO11	
10	10	BOOTMODE9	GPIO10	
9	9	BOOTMODE8	GPIO9	
8	8	BOOTMODE7	GPIO8	
7	7	BOOTMODE6	GPIO7	
6	6	BOOTMODE5	GPIO6	
5	5	BOOTMODE4	GPIO5	
4	4	BOOTMODE3	GPIO4	
3	3	BOOTMODE2	GPIO3	
2	2	BOOTMODE1	GPIO2	
1	1	BOOTMODE0	GPIO1	
0	0	LENDIAN	GPIO0	

There are 5 different formats to the bootmode command:

```
bootmode
```

When the command is entered with no arguments the current bootmode will be displayed.

```
bootmode all
```

This format will display all 16 bootmodes and mark the currently selected bootmode.

```
bootmode #x
```

This will change the currently selected bootmode to the bootmode represented by x. For example, if the board is booted with DIP switch setting 1(OFF) 2(OFF) 3(OFF) 4(ON) then the bootmode would be 1 - DSP NO-BOOT. If 'bootmode #2' is entered, the bootmode represented by DIP switch setting 1(OFF) 2(OFF) 3(ON) 4(OFF) would become the current bootmode (ARM SPI-NOR BOOT). If the 'reboot' command is given, the SoC will be rebooted using this new bootmode. This format is volatile, meaning once power is removed, the bootmode at the next power up will be determined by the DIP switch.

```
bootmode read
```

This format reads the current value of the DIP switch, and changes the current bootmode to this value. For example, if the board is booted with DIP switch setting 1(OFF) 2(OFF) 3(OFF) 4(ON) then the bootmode would be 1 - DSP NO-BOOT. If the DIP switch is changed to 1(OFF) 2(OFF) 3(ON) 4(OFF) and then the command 'bootmode read' is given, the board will change to bootmode 2 - ARM SPI-NOR BOOT. If the 'reboot command is then given, the SoC will be rebooted using this new bootmode.

```
bootmode [#]x hi_value lo_value title
```

This bootmode is used to alter User-Defined bootmodes (bootmodes 8 - 15). x is the index of the bootmode to be set, as such its appropriate value range is 8 - 15, any other value will return an error. hi_value is not currently used, and should always be set to 0. lo_value is a 32 bit hex value whose bits are representative of the table above. title is a given string that is displayed by the bootmode command to help better understand what the bootmode does, it has no purpose within the actual booting of the board. The option '#' is used to determine whether the bootmode will be saved. If # is used, the bootmode will be saved to flash, meaning the new value is tied to the DIP switch, and will remain even if power is removed from the board. If # is not given, then the change will be lost as soon as power is removed. Some examples with descriptions are given below:

```
bootmode 8 0 112005 ARM_SPI
```

bootmode 8 will be set to 112005 and given the title ARM_SPI. To boot into this bootmode, the command 'bootmode #8' followed by 'reboot' would be given (or changing the DIP switch to 1(ON) 2(OFF) 3(OFF) 4(OFF) without removing power, then entering 'bootmode read' followed by 'reboot'). Once power is removed, this change is lost.

```
bootmode #10 0 12cba1 RANDOM_BOOT
```

bootmode 10 will be set to 12cba1 and given the title RANDOM_BOOT. This is now the permanent value of bootmode 10; the change will persist even if power is removed.

EVMK2E Hardware Setup

Getting Started with MCSDK: EVMK2E Hardware Setup

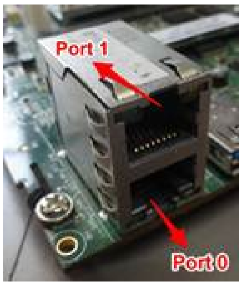
Last updated: **09/20/2015**

Hardware Setup

Note: The EVM board is sensitive to electrostatic discharges (ESD). Use a grounding strap or other device to prevent damaging the board. Be sure to connect communication cables before applying power to any equipment.

Attach the Ethernet cable

Using the Ethernet cable supplied, connect one end of the cable to the Ethernet Port 0 (At bottom one) on the EVM and the other end to your PC. The below picture shows which Ethernet Port is port 0:



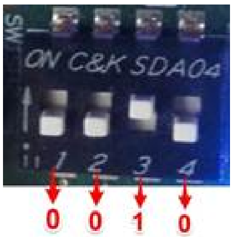
Connect the JTAG interface

Use the USB to USB mini-B cable provided. Connect the USB mini-B connector to the USB mini-B interface near to the RST_PWR1 (Red color) button on the EVM, and the USB connector to your PC. This enables XDS-2xx emulation and is directly useable by CCS. If you are using a different JTAG, you can connect it at MIPI60 connector (EMU1).

Set the boot mode switch SW1

SPI Little Endian Boot mode (Default factory setting)

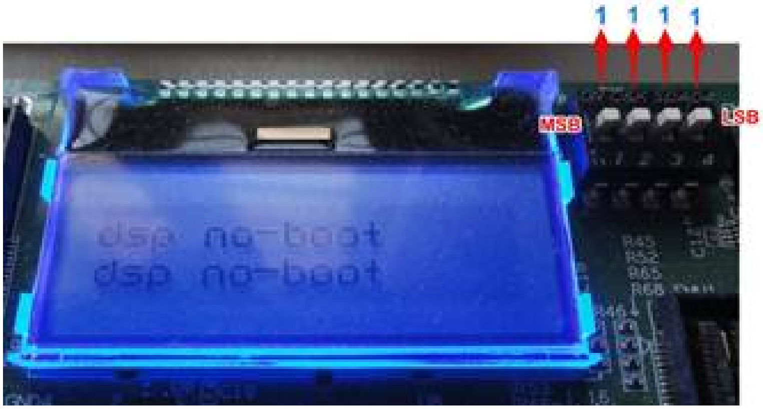
```
MSB                               LSB
SW1 - 1(OFF) 2(OFF) 3(ON) 4(OFF)
```



Note: Here a switch on “ON” position should be considered as “1”.

No Boot/JTAG DSP Little Endian Boot mode

```
MSB                           LSB
SW1 - 1(ON) 2(ON) 3(ON) 4(ON)
```



Attach the serial port cable to the SoC UART port

Connect the SoC UART port to PC using the serial cable provided with the EVM. The SoC UART port is the 4-pin white connector COM1 of the EVM.

Start TeraTerm or HyperTerminal and set configuration to

- Baud Rate or Bits per second: 115200
- Data Bits: 8
- Parity: None
- Stop Bits: 1
- Flow Control: None

Connect the power cable

Connect the power cable to the EVM power jack on the board. To be ESD safe, plug in the other end of the power cable only after you have connected the power cord to the board. Then turn on the board.

DIP Switch and Bootmode Configurations

EVM SW1 switch Bootmode Configuration

The table below shows the bootmode combinations for the BMC v1.1.0.x. and value selected from internal flash memory of LM3s2d93.

DIP Switch settings Selected	High_value of that bootmode	Low_value of that bootmode	Selected bootmode
0000	0x00000000	0x00010067	ARM NAND
0001	0X00000000	0x00100001	DSP No Boot
0010	0x00000000	0x00008005	ARM SPI
0011	0x00000000	0x00100003	ARM I2C Master
0100	0x00000000	0x0000006F	ARM UART Master
0101	0x00000000	0x0001506B	ARM RBL EthNet
0110	0x00000000	0x00001061	Sleep with Max PLL and ARM Bypass
0111	0x00000000	0x00001061	Sleep with Max PLL
1000	0x00000000	0x00010167	DSP NAND
1001	0x00000000	0x00001061	Sleep with Slow PLL and ARM Bypass
1010	0x00000000	0x00008105	DSP SPI-boot
1011	0x00000000	0x00100103	ARM I2C Master
1100	0x00000000	0x0000016F	DSP UART boot
1101	0x00000000	0x0001516B	DSP RBL ENET
1110	0x00000000	0x00003661	Sleep with Slow PLL and Slow ARM PLL
1111	0x00000000	0x00100001	DSP No-Boot

How To Guides

Host driver for on-board mini-USB connector

The K2E EVM has a CP2105 device on-board. A driver must be installed on the host PC in order to be able to communicate with the EVM using the CP2105 mini-USB connector located at the corner edge of the EVM. The driver can be downloaded from CP2105 driver download [1].

Both Linux and Windows host machine drivers can be downloaded from this page. For Linux host machine, please follow the instructions given in the release notes. There are two versions of drivers for Linux kernel version 3.x.x and 2.6.x. Please download appropriate drivers after identifying the correct kernel version of the user’s host machine.

Note: Before testing the USB connection, make sure that the mini-USB cable is plugged into the port on the base board.

After installing the driver and connecting the USB cable, two COM ports should be visible in the list of COM ports available to connect to in the PC Host terminal console. The lower COM port (Enhanced) corresponds to the SoC

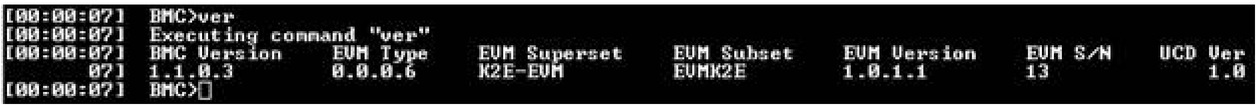
UART and the higher (Standard) one corresponds to the MCU UART.

BMC Version Check and Update

BMC, or Board Management Controller, takes care of the power, clocks, resets, bootmodes, etc. of the EVM.

You can check the version by:

1. Opening a hyperterminal or another similar type of console application.
2. Set COM Port to higher value.
   - When you connect to CP2105 mini-USB on the EVM it will provide two COM port connections, one to the SOC UART and one to BMC UART.
   - The SOC UART will always be the “Enhanced” COM port, for example COM6 (actual COM PORT values will vary.)
3. Set COM port properties appropriately:
   - Baud Rate or Bits per second: 115200
   - Data Bits: 8
   - Parity: None
   - Stop Bits: 1
   - Flow Control: None
4. At BMC prompt type 'ver' (no quotes).
5. Check BMC version

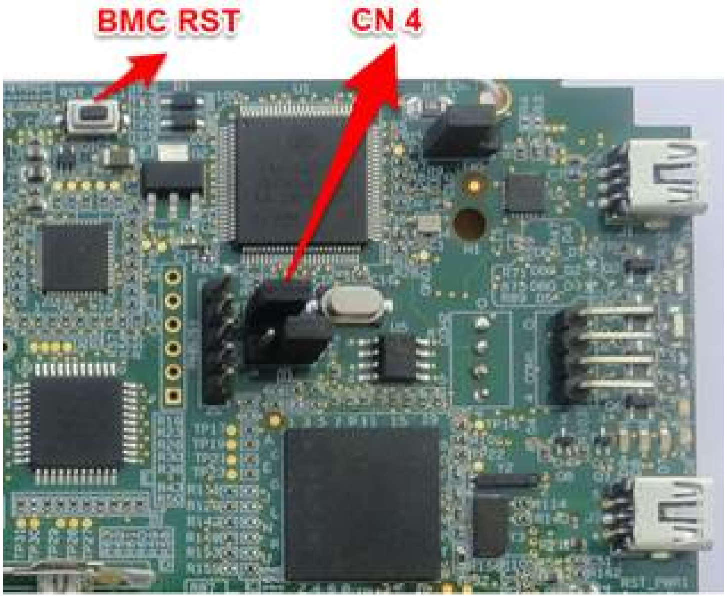


```
[00:00:07]  BMC>ver
[00:00:07]  Executing command "ver"
[00:00:07]  BMC Version    EVM Type    EVM Superset    EVM Subset    EVM Version    EVM S/N    UCD Ver
        07]  1.1.0.3        0.0.0.6     K2E-EVM         EVMK2E        1.0.1.1        13             1.0
[00:00:07]  BMC>
```

If an in-field update is needed, downloaded the latest version here [2] (labeled “BMC”) and follow instructions below.

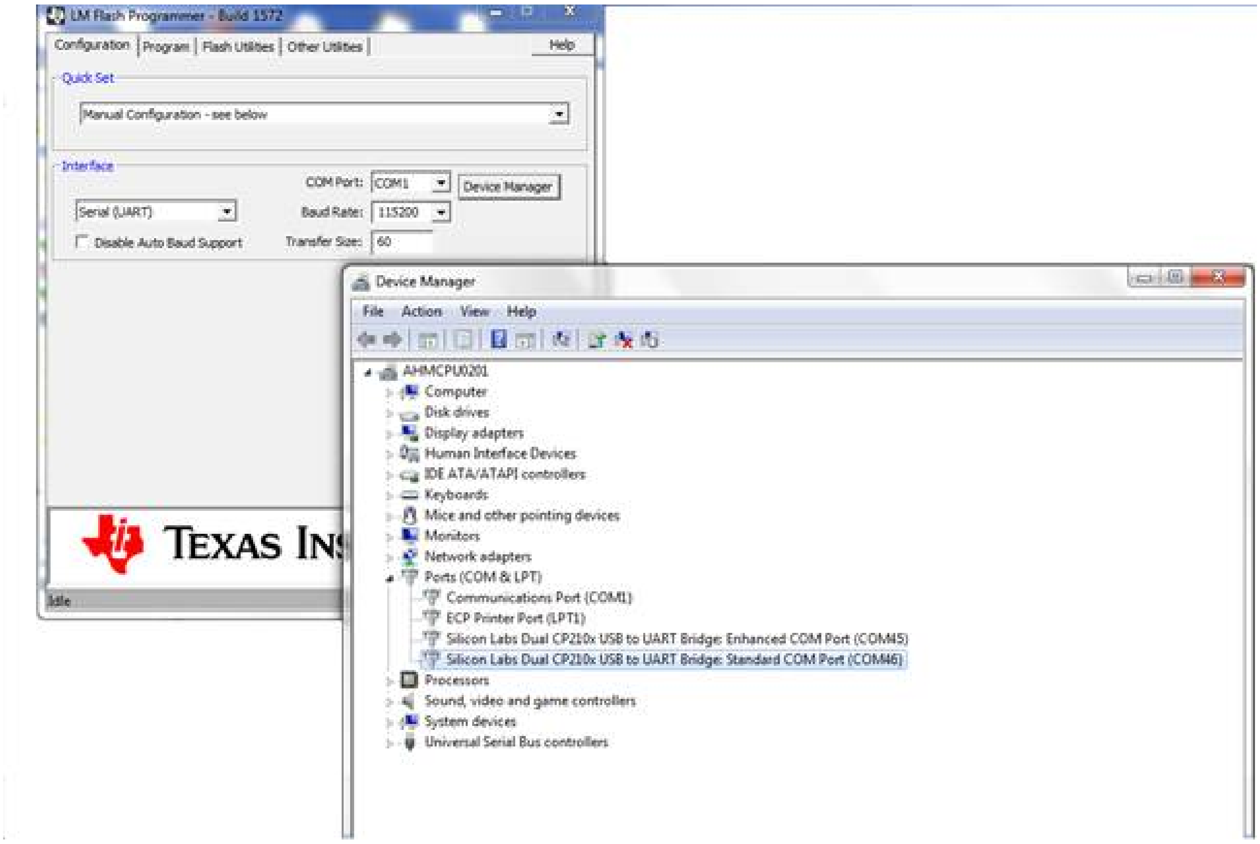
Prepare EVM for in-field update

1. Remove power to the EVM.
2. Set boot mode to "No Boot mode” (see above).
3. Remove the MCU_BOOTSELECT (CN4) jumper (see picture below for location of jumper.)
4. Make sure your
   - USB cable is connected to CP2105 mini-USB (not XDS200 Emulator USB) or
   - Connect 4pin UART cable to COM1: MCU UART connector
5. Make sure no HyperTerminal/Console connected to BMC COM port are open or active.
6. Use the LM Flash Programmer (available here [5]) to update the firmware, as detailed in the steps below.



Perform in-field update

1. Apply power to the EVM. No LED's will be illuminated and no LCD backlight or characters will be on because the BMC is waiting for a command rather than executing from Flash.
2. Open the LM Flash programmer utility. (Default location Start Menu -> All Programs -> Texas Instruments -> Stellaris -> LM Flash Programmer -> LM Flash Programmer)
3. In the LM Flash Programmer Utility ‘Configuration’ tab, in the interface section, select ‘Serial (UART)’ from the drop-down box on the left.
4. Select the BMC COM Port (the same COM port used to issue the ver command earlier), and set the ‘Baud Rate’ to 115200.
5. Set ‘Transfer Size’ to 60, and make sure ‘Disable Auto Baud Support’ is unchecked. See image below.
6. In the ‘Program’ tab, Browse to the location of the binary file containing the firmware update, and select it.
7. Leave all other options as default, and press the ‘Program’ button.
8. After the programming is complete, power off the board.
9. Reconnect the jumper.
10. Open the HyperTerminal/Console for the BMC COM port.
11. Apply power to the EVM. When BMC completes initialization of board it will show latest version of BMC in Console.
12. If step 9 was done after power was applied, just type "ver" at BMC prompt.



Quick Set: Manual Configuration - see below

Interface: Serial (UART)

□ Disable Auto Baud Support

COM Port: COM1

Baud Rate: 115200

Transfer Size: 60

- AHMCPU0201
  - Computer
  - Disk drives
  - Display adapters
  - Human Interface Devices
  - IDE ATA/ATAPI controllers
  - Keyboards
  - Mice and other pointing devices
  - Monitors
  - Network adapters
  - Ports (COM & LPT)
    - Communications Port (COM1)
    - ECP Printer Port (LPT1)
    - Silicon Labs Dual CP210x USB to UART Bridge: Enhanced COM Port (COM45)
    - Silicon Labs Dual CP210x USB to UART Bridge: Standard COM Port (COM46)
  - Processors
  - Sound, video and game controllers
  - System devices
  - Universal Serial Bus controllers

UCD Power Management Update

There is one power management module (a.k.a. UCD) located on the EVM. It can be identified by its address: 104(68h). Each module contains non-volatile registers that determine its operation. It may be necessary to update these registers in the field after the board has been shipped.

This update can be performed through the BMC, which can issue commands to the UCD modules to update the register settings. The **Power Management Configuration Update Tool** performs the task of sending commands to the BMC to get the current module versions, and perform updates using configuration files.

The latest version of the tool is available from here [2] along with instructions on using the tool, and the latest configuration files (txt files). Please follow the instructions provided to check the current module versions, and update them accordingly.

Note: The DIP switch configuration of the board when running the update is irrelevant.

TCIEVMK2L Hardware Setup

Getting Started with MCSDK: TCIEVMK2L Hardware Setup

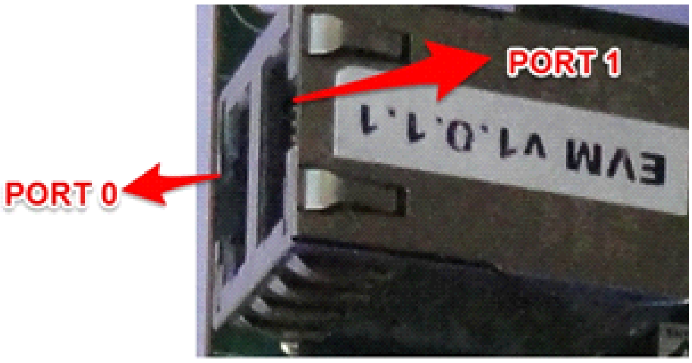
Last updated: **09/20/2015**

Hardware Setup

Note: The EVM board is sensitive to electrostatic discharges (ESD). Use a grounding strap or other device to prevent damaging the board. Be sure to connect communication cables before applying power to any equipment.

Attach the Ethernet cable

Using the Ethernet cable supplied, connect one end of the cable to the Ethernet Port 0 (At bottom one) on the EVM and the other end to your PC. The below picture shows which Ethernet Port is port 0:



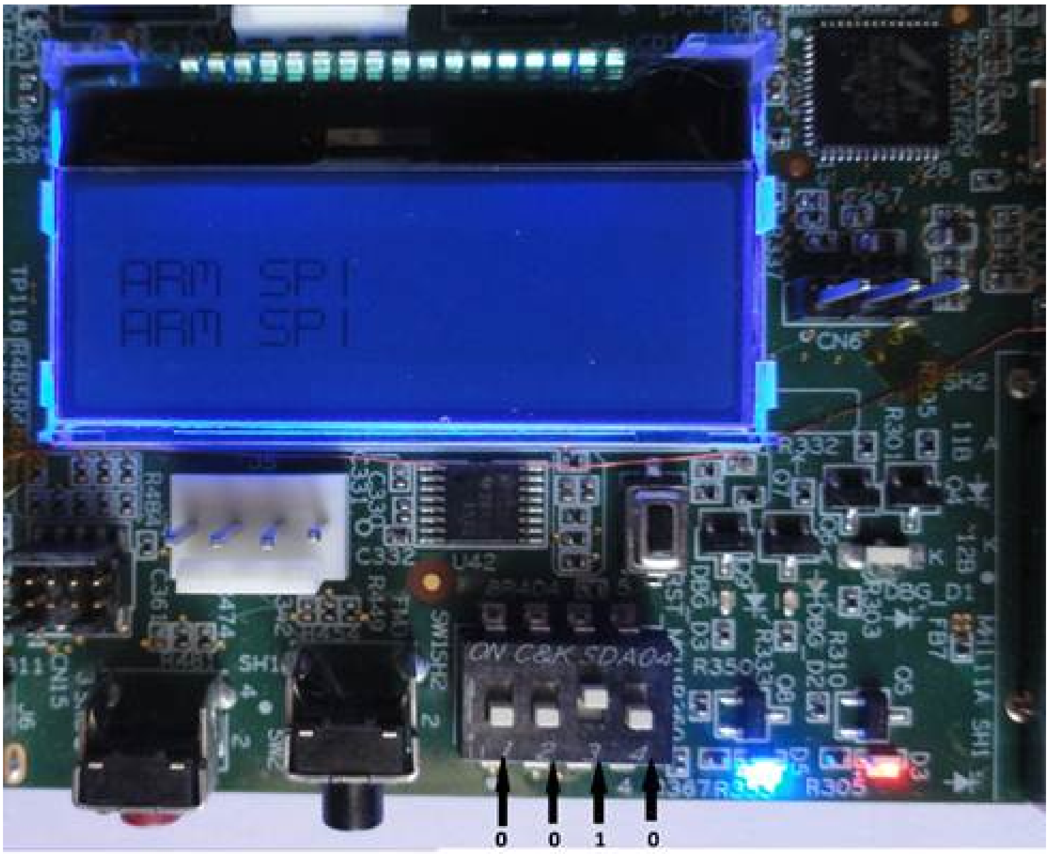
Connect the JTAG interface

Use the USB to USB mini-B cable provided. Connect the USB mini-B connector to the USB mini-B interface near to the RST_PWR (Red color) button on the EVM, and the USB connector to your PC. This enables XDS-2xx emulation and is directly useable by CCS. If you are using a different JTAG, you can connect it at MIPI60 connector (EMU1).

Set the boot mode switch SW1

SPI Little Endian Boot mode (Default factory setting)

```
SW1 - 4(ON) 3(ON) 2(OFF) 1(ON)
```



Note: Here a switch on “ON” position should be considered as “1”.

No Boot/JTAG DSP Little Endian Boot mode

```
SW1 - 4(ON) 3(ON) 2(ON) 1(ON)
```

Attach the serial port cable to the SoC UART port

Connect the SoC UART port to PC using the serial cable provided with the EVM. The SoC UART port is the 4-pin white connector COM1 of the EVM.

Start Tera Term or HyperTerminal and set configuration to

- Baud Rate or Bits per second: 115200
- Data Bits: 8
- Parity: None
- Stop Bits: 1

- Flow Control: None

Connect the power cable

Connect the power cable to the EVM power jack on the board. To be ESD safe, plug in the other end of the power cable only after you have connected the power cord to the board. Then turn on the board.

DIP Switch and Bootmode Configurations

EVM SW1 switch Bootmode Configuration

The table below shows the bootmode combinations for the BMC v1.1.0.x. and value selected from internal flash memory of LM3s2d93. Bootmodes 0-7 are read-only, and 8-15 are user-configurable.

DIP Switch settings Selected	**High_value of that bootmode**	**Low_value of that bootmode**	**Selected bootmode**
0000	0x00000000	0x00100EEB	ARM NAND With CSISC2_0_MUX_SEL set to 0
0001	0X00000000	0x00100001	DSP No Boot
0010	0x00000000	0x00108087	ARM SPI
0011	0x00000000	0x00100005	ARM I2C Master
0100	0x00000000	0x00100EEF	ARM UART Master
0101	0x00000000	0x0010DEED	ARM RBL EthNet
0110	0x00000000	0x001010E1	Sleep with Max PLL and ARM Bypass
0111	0x00000000	0x00103EE1	Sleep with Max PLL
1000	0x00000000	0x00100FEB	DSP NAND
1001	0x00000000	0x001010C1	Sleep with Slow PLL and ARM Bypass
1010	0x00000000	0x00108187	DSP SPI-boot
1011	0x00000000	0x00100105	ARM I2C Master
1100	0x00000000	0x00100FEF	DSP UART boot
1101	0x00000000	0x0010DFED	DSP RBL ENET
1110	0x00000000	0x00103CC1	Sleep with Slow PLL and Slow ARM PLL
1111	0x00000000	0x08100001	DSP No-Boot – With CSISC2_0_MUX_SEL set to 1

How To Guides

Host driver for on-board mini-USB connector

The K2L EVM has a CP2105 device on-board. A driver must be installed on the host PC in order to be able to communicate with the EVM using the CP2105 mini-USB connector located at the corner edge of the EVM. The driver can be downloaded from CP2105 driver download [1].

Both Linux and Windows host machine drivers can be downloaded from this page. For Linux host machine, please follow the instructions given in the release notes. There are two versions of drivers for Linux kernel version 3.x.x and 2.6.x. Please download appropriate drivers after identifying the correct kernel version of the user’s host machine.

Note: Before testing the USB connection, make sure that the mini-USB cable is plugged into the port on the base board.

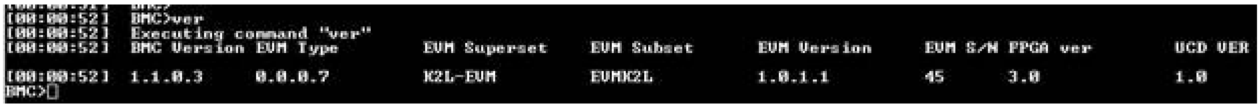
After installing the driver and connecting the USB cable, two COM ports should be visible in the list of COM ports available to connect to in the PC Host terminal console. The lower COM port (Enhanced) corresponds to the SoC UART and the higher (Standard) one corresponds to the MCU UART.

BMC Version Check and Update

BMC, or Board Management Controller, takes care of the power, clocks, resets, bootmodes, etc. of the EVM.

You can check the version by:

1. Opening a hyperterminal or another similar type of console application.
2. Set COM Port to higher value
   - When you connect to CP2105 mini-USB on the EVM it will provide 2 COM port connections, one to the SOC UART and one to BMC UART.
   - The SOC UART will always be the “Enhanced” COM port, for example COM6. (Actual COM PORT values will vary).
3. Set COM port properties appropriately:
   - Baud Rate or Bits per second: 115200
   - Data Bits: 8
   - Parity: None
   - Stop Bits: 1
   - Flow Control: None
4. At BMC prompt type 'ver' (no quotes)
5. Check BMC version

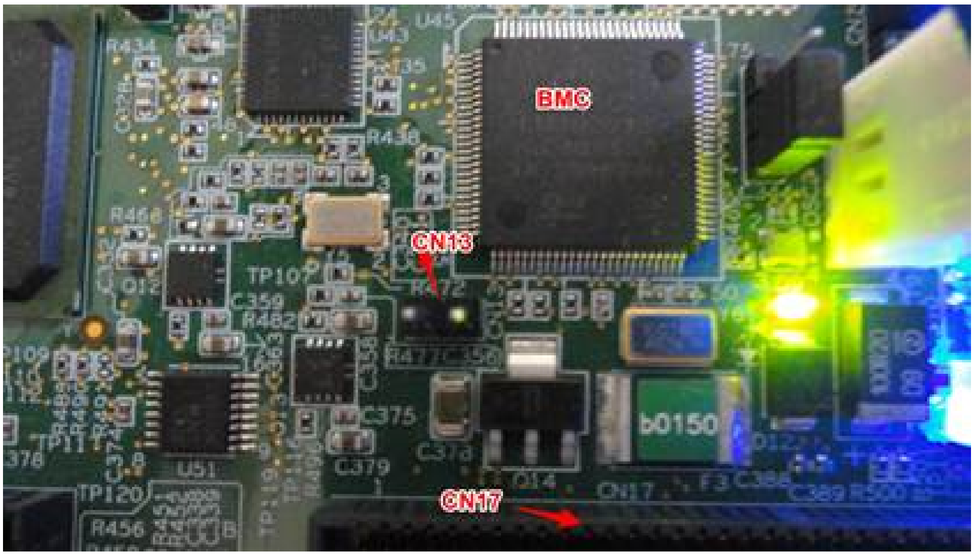


```
[00:00:51]  BMC>
[00:00:52]  BMC>ver
[00:00:52]  Executing command "ver"
[00:00:52]  BMC Version EVM Type        EVM Superset    EVM Subset      EVM Version     EVM S/N FPGA ver        UCD VER

[00:00:52]  1.1.0.3     0.0.0.7         K2L-EVM         EVMK2L          1.0.1.1         45      3.0             1.0
BMC>
```

If an in-field update is needed, downloaded the latest version **TBD** (labeled “BMC”) and follow instructions below.

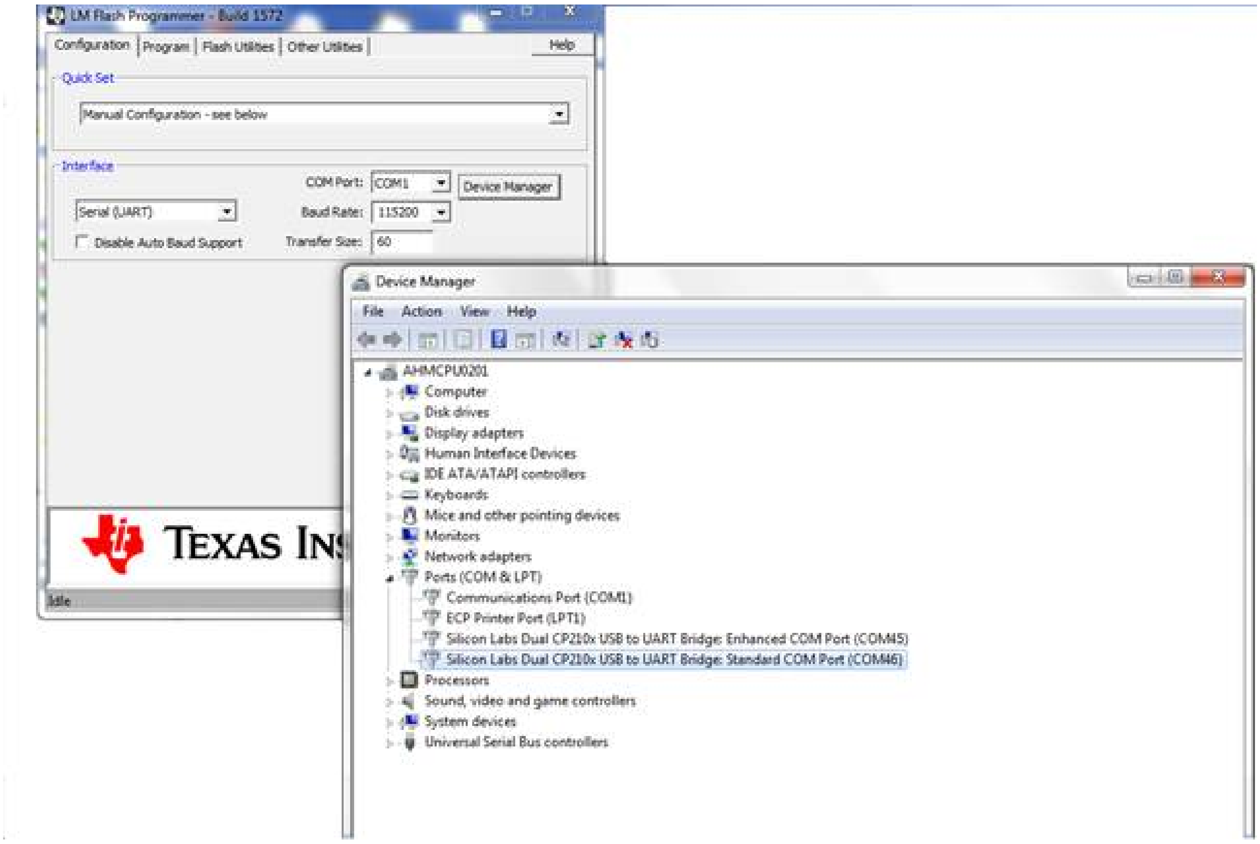
Prepare EVM for in-field update

1. Remove power to the EVM.
2. Set boot mode to "No Boot mode” (see above)
3. Remove the MCU_BOOTSELECT (CN13) jumper (see picture below for location of jumper.)
4. Make sure your USB cable is connected to CP2105 mini-USB (not XDS200 Emulator USB) OR connect 4pin UART cable to J1: MCU UART connector.
5. Make sure no HyperTerminal/Console connected to BMC COM port are open or active.
6. Use the LM Flash Programmer (available here [5]) to update the firmware, as detailed in the steps below.



Perform in-field update

1. Apply power to the EVM. No LED's will be illuminated and no LCD backlight or characters will be on because the BMC is waiting for a command rather than executing from Flash.
2. Open the LM Flash programmer utility. (Default location Start Menu -> All Programs -> Texas Instruments -> Stellaris -> LM Flash Programmer -> LM Flash Programmer)
3. In the LM Flash Programmer Utility ‘Configuration’ tab, in the interface section, select ‘Serial (UART)’ from the drop-down box on the left.
4. Select the BMC COM Port (the same COM port used to issue the ver command earlier), and set the ‘Baud Rate’ to 115200.
5. Set ‘Transfer Size’ to 60, and make sure ‘Disable Auto Baud Support’ is unchecked. See figure below.
6. In the ‘Program’ tab, Browse to the location of the binary file containing the firmware update, and select it.
7. Leave all other options as default, and press the ‘Program’ button.
8. After the programming is complete, power off the board.
9. Reconnect the jumper.
10. Open the HyperTerminal/Console for the BMC COM port.
11. Apply power to the EVM. When BMC completes initialization of board it will show latest version of BMC in Console.
12. If step 9 was done after power was applied, just type "ver" at BMC prompt.


LM Flash Programmer - Build 1572
Configuration
Program
Flash Utilities
Other Utilities
Help
Quick Set
Interface
Device Manager
Texas Ins
Idle
Device Manager
File
Action
View
Help
AHMCPU0201
Computer
Disk drives
Display adapters
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Ports (COM & LPT)
Communications Port (COM1)
ECP Printer Port (LPT1)
Silicon Labs Dual CP210x USB to UART Bridge: Enhanced COM Port (COM45)
Silicon Labs Dual CP210x USB to UART Bridge: Standard COM Port (COM46)
Processors
Sound, video and game controllers
System devices
Universal Serial Bus controllers

Manual Configuration - see below

Serial (UART)

□ Disable Auto Baud Support

COM Port: COM1

Baud Rate: 115200

Transfer Size: 60

UCD Power Management Update

There is one power management module (a.k.a. UCD) located on the EVM. It can be identified by its address: 104(68h). Each module contains non-volatile registers that determine its operation. It may be necessary to update these registers in the field after the board has been shipped.

This update can be performed through the BMC, which can issue commands to the UCD modules to update the register settings. The **Power Management Configuration Update Tool** performs the task of sending commands to the BMC to get the current module versions, and perform updates using configuration files.

The latest version of the tool is available from **TBD** along with instructions on using the tool, and the latest configuration files (txt files). Please follow the instructions provided to check the current module versions, and update them accordingly.

Note: The DIP switch configuration of the board when running the update is irrelevant.

Configure EVM to DSP Big Endian Mode

To set DSP as Big Endian the LSB of the lowest word of the BMC bootmode must be set to 0. BMC bootmodes 0 to 7 are read-only; bootmodes 8 to 15 are customizable.

Here are the steps for setting Big Endian for DSP No-Boot bootmode. These steps will overwrite the bootmode setting that is currently there. This procedure uses custom bootmode 9 but you can select any custom bootmode that you are unlikely to use.

1. Connect to the EVM's BMC console port.
2. Type "bootmode all" to list all bootmodes available.
3. Please view bootmode #1 setting (would be entred as "bootmode #1 0x0 0x100001 DSP No-Boot"). This is DSP No-Boot for Little Endian.

4. Type "bootmode #9 0x0 0x100000 DSP No-Boot Big Endian". As you can see only the LSB of the lowest word has been modified to have DSP as Big Endian.
5. Type "bootmode #9". This sets BMC to use bootmode #9
6. Type "reboot". This reboots the EVM to come up in the bootmode just set. In this case DSP No-Boot Big Endian.

Note: If the unit is power cycled the bootmode will be based on the bootmode EVM switches. If you would like the bootmode to come up as the default after power cycle as bootmode #9 then set the bootmode switches to 9.

Change And Configure The Boot Mode

There are 5 different formats to the bootmode command:

```
bootmode
```

When the command is entered with no arguments the current bootmode will be displayed.

```
bootmode all
```

This format will display all 16 bootmodes and mark the currently selected bootmode.

```
bootmode #x
```

This will change the currently selected bootmode to the bootmode represented by x. For example, if the board is booted with DIP switch setting 1(OFF) 2(OFF) 3(OFF) 4(ON) then the bootmode would be 1 - DSP NO-BOOT. If 'bootmode #2' is entered, the bootmode represented by DIP switch setting 1(OFF) 2(OFF) 3(ON) 4(OFF) would become the current bootmode (ARM SPI-NOR BOOT). If the 'reboot' command is given, the SoC will be rebooted using this new bootmode. This format is volatile, meaning once power is removed, the bootmode at the next power up will be determined by the DIP switch.

```
bootmode read
```

This format reads the current value of the DIP switch, and changes the current bootmode to this value. For example, if the board is booted with DIP switch setting 1(OFF) 2(OFF) 3(OFF) 4(ON) then the bootmode would be 1 - DSP NO-BOOT. If the DIP switch is changed to 1(OFF) 2(OFF) 3(ON) 4(OFF) and then the command 'bootmode read' is given, the board will change to bootmode 2 - ARM SPI-NOR BOOT. If the 'reboot command is then given, the SoC will be rebooted using this new bootmode.

```
bootmode [#]x hi_value lo_value title
```

Of the 16 total bootmodes to select from, 0-7 are read-only and 8-15 are user configurable. x is the index of the bootmode to be set, as such its appropriate value range is 8 - 15, any other value will return an error. hi_value is not currently used, and should always be set to 0. lo_value is a 32 bit hex value whose bits are representative of the table above. title is a given string that is displayed by the bootmode command to help better understand what the bootmode does, it has no purpose within the actual booting of the board. The option '#' is used to determine whether the bootmode will be saved. If # is used, the bootmode will be saved to flash, meaning the new value is tied to the DIP switch, and will remain even if power is removed from the board. If # is not given, then the change will be lost as soon as power is removed. Some examples with descriptions are given below:

```
bootmode 8 0 112007 ARM_SPI
```

bootmode 8 will be set to 112007 and given the title ARM_SPI. To boot into this bootmode, the command 'bootmode #8' followed by 'reboot' would be given (or changing the DIP switch to 1(ON) 2(OFF) 3(OFF) 4(OFF) without removing power, then entering 'bootmode read' followed by 'reboot'). Once power is removed, this change is lost.

```
bootmode #10 0 12cba1 RANDOM_BOOT
```

bootmode 10 will be set to 12cba1 and given the title RANDOM_BOOT. This is now the permanent value of bootmode 10; the change will persist even if power is removed.

Program EVM UG

Getting Started with MCSDK: Program EVM Guide

Last updated: **09/20/2015**

Overview

This release provides the images for the factory to program on the eeprom, nand and nor for EVMK2H, EVM2E & EVMK2L

Files Provided

EVMK2H Files

EVMK2H uses the factory default images under program_evm\binaries\evmk2h.

File Name	Description
nand.bin	Nand UBIFS image (keystone-evm-ubifs.ubi or k2hk-evm-ubifs.ubi)
nandwriter_evmk2h.out	Nand Writer DSP executable
nand_writer_input.txt	nand image writer input file
nor.bin	SPI NOR file for U-Boot (u-boot-spi-keystone-evm.gph)
norwriter_evmk2h.out	NOR image writer DSP executable
nor_writer_input.txt	NOR image writer input file

EVMK2E Files

EVMK2E uses the factory default images under program_evm\binaries\evmk2e.

File Name	Description
nand.bin	Nand UBIFS image (k2e-evm-ubifs.ubi)
nandwriter_evmk2e.out	Nand Writer DSP executable
nand_writer_input.txt	nand image writer input file
nor.bin	SPI NOR file for U-Boot (u-boot-spi-k2e-evm.gph)
norwriter_evmk2e.out	NOR image writer DSP executable
nor_writer_input.txt	NOR image writer input file

EVMK2L Files

EVMK2L uses the factory default images under program_evm\binaries\evmk2l.

File Name	Description
nand.bin	Nand UBIFS image (k2l-evm-ubifs.ubi)
nandwriter_evmk2l.out	Nand Writer DSP executable
nand_writer_input.txt	nand image writer input file
nor.bin	SPI NOR file for U-Boot (u-boot-spi-k2l-evm.gph)
norwriter_evmk2l.out	NOR image writer DSP executable
nor_writer_input.txt	NOR image writer input file

MD5SUM utility used

Please use the md5sum utility from the following link: http:/ / www. pc-tools. net/ files/ win32/ freeware/ md5sums-1.2.zip [1]

Device Support

- The images provided support EVMK2H in Little endian Mode.
- Starting from MCSDK 3.1.0 release: EVMK2L, EVMK2E in little endian mode

Directory Structure

- The program_evm (top-level) directory is intended to hold the *DSS* script for the Code Composer Studio which programs the default images to NAND/NOR/EEPROM.
- The binaries/evmxxxx directory is intended to hold all the factory default images and the respective writers.
- The configs/evmxxxx directory is intended to hold the "CCS target configuration files". For evmk2h/evmk2l/evmk2e, configuration file is present for Windows, and one is present for linux per platform.
- The gel directory holds custom GEL files for the boards present. It also contains a README.txt for the gel files usage.
- The logs directory is empty and will be used to store logs. Logs are automatically generated when using program_evm.js to flash evmxxxx devices.

```
├───program_evm
│   │   program_evm_userguide.pdf
│   │   program_evm.js
│   │
│   ├───binaries
│   │   └───evmk2h
│   │           eepromwriter_evmk2h.out
│   │           eepromwriter_input.txt
│   │           eepromwriter_input50.txt
│   │           eepromwriter_input51.txt
│   │           nand.bin
│   │           nandwriter_evmk2h.out
│   │           nand_writer_input.txt
│   │           nor.bin
│   │           norwriter_evmk2h.out
│   │           nor_writer_input.txt
│   │   └───evmk2l
│   │           eepromwriter_evmk2l.out
│   │           eepromwriter_input.txt
│   │           eepromwriter_input50.txt
│   │           eepromwriter_input51.txt
│   │           nand.bin
│   │           nandwriter_evmk2l.out
│   │           nand_writer_input.txt
│   │           nor.bin
│   │           norwriter_evmk2l.out
│   │           nor_writer_input.txt
│   │   └───evmk2e
│   │           eepromwriter_evmk2e.out
│   │           eepromwriter_input.txt
│   │           eepromwriter_input50.txt
│   │           eepromwriter_input51.txt
│   │           nand.bin
│   │           nandwriter_evmk2e.out
│   │           nand_writer_input.txt
│   │           nor.bin
│   │           norwriter_evmk2e.out
│   │           nor_writer_input.txt
│   ├───configs
│   │   └───evmk2h
│   │           evmk2h.ccxml
│   │           evmk2h-linuxhost.ccxml
│   │   └───evmk2l
│   │           evmk2l.ccxml
│   │           evmk2l-linuxhost.ccxml
│   │   └───evmk2e
│   │           evmk2e.ccxml
│   │           evmk2e-linuxhost.ccxml
│   ├───gel
│   │           xtcievmk2x.gel
│   │           evmk2e.gel
│   │           tcievmk2l.gel
│   │           README.txt
│   └───logs (empty directory)
```

Programming the bin files

This section assumes you have installed BIOS-MCSDK (MCSDK 3.x for EVMK2H/EVMK2L/EVMK2E) and Code Composer Studio.

Set the EVM for no-boot mode

For EVMK2H

Due to hardware NAND issue, EVM of certain revisions need to use alternative programming method:

EVM Revision	NAND Known Issue	Programming Method
Rev 3.0	Yes	Alternative U-Boot command
Rev 2.0	Yes	Alternative U-Boot command
Rev 1.x	No	Program_evm

For Rev 2.0 and 3.0 EVM using alternative U-Boot command to program NAND, please follow the steps below:
Set the boot mode to SPI boot mode: Set_SPI_boot [2]

```
 1. Set up TFTP server
 2. Have Ethernet cable connected to the EVM and verify the connection to the TFTP server
 3. Boot up the EVM to the U-boot prompt and type the following commands:
```

```
    u-boot# setenv serverip <TFTP server IP address>
    u-boot# setenv tftp_root <tftp directory>
    u-boot# setenv addr_fs 0x82000000
    u-boot# nand erase.part ubifs
    u-boot# dhcp ${addr_fs} ${tftp_root}/keystone-evm-ubifs.ubi
    u-boot# nand write ${addr_fs} ubifs ${filesize}
    u-boot# env default -f -a
    u-boot# setenv boot ubi
    u-boot# saveenv
```

Once it is completed, the EVM is ready to use.

For Rev 1.0 EVM, make sure the EVM dip switches are set for no-boot mode and continue.

SWITCH	Pin1	Pin2	Pin3	Pin4
SW1	**Off**	**Off**	**Off**	On

See instruction here for K2H no boot mode for reference K2H_Noboot [4]

For EVMK2E

Make sure the EVM dip switches are kept as below to put the board in no-boot mode, and continue

SWITCH	Pin1	Pin2	Pin3	Pin4
SW1	**Off**	**Off**	**Off**	On

See instruction here for K2E no boot mode for reference: K2E_NoBoot [3]

For EVMK2L

Make sure the EVM dip switches are kept as below, and continue

SWITCH	Pin1	Pin2	Pin3	Pin4
SW1	**Off**	**Off**	**Off**	On

See instructions here for K2L no boot mode for reference: K2L_NoBoot [4]

Set the Environment Variables

Please make sure the below environment variables needs to be set. Otherwise there could be some unexpected behavior experienced.

Windows

1. Set the **DSS_SCRIPT_DIR** environment variable (Mandatory) to Code Composer Studio scripting bin directory. Example:

```
set DSS_SCRIPT_DIR="C:\ti\ccsv5\ccs_base\scripting\bin"
```

2. Set the **PROGRAM_EVM_TARGET_CONFIG_FILE** environment variable: to the respective configuration file for the platform.
(e.g)

```
set PROGRAM_EVM_TARGET_CONFIG_FILE=configs\evmk2h\evmh2h.ccxml
```

Note: Please note that
1. The DSS_SCRIPT_DIR env variable needs the opening/closing quotes,
2. The PROGRAM_EVM_TARGET_CONFIG_FILE variable, on the other hand, ***should not*** have the quotes around it if the path has spaces in it.

Linux

1. Set the **DSS_SCRIPT_DIR** environment variable (Mandatory) to your Code Composer Studio scripting bin directory. Example:

```
 export DSS_SCRIPT_DIR=~/ti/ccsv5/ccs_base/scripting/bin
```

2. Set the **PROGRAM_EVM_TARGET_CONFIG_FILE** environment variable Please provide the path for the ccxml file that is created for the EVM for the CCS. Please note that this step can be skipped for CCS 5.1.0 users since the program evm has a default target configuration files bundled for xds100 and xds560 mezzanine cards. Example:

```
 export PROGRAM_EVM_TARGET_CONFIG_FILE=configs/evmk2h/evmk2h-linuxhost.ccxml
```

DSS Script Arguments

General Script Usage

Script Usage:
Windows:

```
[MCSDK]\tools\program_evm>%DSS_SCRIPT_DIR%\dss.bat program_evm.js evm(k2h|k2e|k2l)-le
```

Linux:

```
[MCSDK]/tools/program_evm>$DSS_SCRIPT_DIR/dss.sh program_evm.js evm(k2h|k2e|k2l)[-le]
```

MCSDK: refers to the MCSDK installation directory, eg. C:\Program Files\Texas Instruments\mcsdk_bios_3_01_00_03\
k2h:TCI6638 device
k2e:C66AK2E device
k2l:TCI6630 device
-le (optional): Little Endian (default)
-be (optional): Big Endian

Formatting the NAND Flash

Note: Sometimes, NAND flash could be corrupted (e.g. EVM boots from UBI and does not gracefully shut down), NAND flash needs to be formatted before loading the program using program_evm utility.

The program_evm supports formatting the NAND device as below.
Windows:

```
[MCSDK]\tools\program_evm>%DSS_SCRIPT_DIR%\dss.bat program_evm.js evm(k2h|k2e|k2l)-le format-nand
```

Linux:

```
[MCSDK]/tools/program_evm>$DSS_SCRIPT_DIR/dss.sh program_evm.js evm(k2h|k2e|k2l)-le format-nand
```

Warning: Please note that this would erase all the nand blocks.

Executing the DSS script to restore factory default images.

Windows

1. cd "\program_evm" directory
2. Set the necessary environment variables as described in section 7.2.1.
3. Using the DSS Script batch file, run the "program_evm.js" script command from program_evm directory.

Example:

```
\program_evm>%DSS_SCRIPT_DIR%\dss.bat program_evm.js evmk2h-le
```

This will write all the little endian images to K2H EVM.

Linux

1. cd "program_evm" directory
2. Set the necessary environment variables as described under section 7.2.2.
3. Using the DSS Script batch file, run the "program_evm.js" script command from program_evm directory.

Example:

```
/program_evm> $DSS_SCRIPT_DIR/dss.sh program_evm.js evmk2h-le
```

This will write all the little endian images to K2H EVM.

Sample DSS Script output for Windows and Linux

The sample output after running the DSS Script is as below.

Note: The loading of nand.bin can take up to a few minutes depending on the image size.

```

board: evm6657l
endian: Little
emulation: XDS200 emulator
binaries: C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/
ccxml: C:\Documents and
Settings\xxxxxxx\user\CCSTargetConfigurations\evmc6657.ccxml
C66xx_0: GEL Output:
Connecting Target...

C66xx_0: GEL Output: DSP core #0

C66xx_0: GEL Output: C6657L GEL file Ver is 1.0

C66xx_0: GEL Output: Global Default Setup...

C66xx_0: GEL Output: Setup Cache...

C66xx_0: GEL Output: L1P = 32K

C66xx_0: GEL Output: L1D = 32K

C66xx_0: GEL Output: L2 = ALL SRAM

C66xx_0: GEL Output: Setup Cache... Done.

C66xx_0: GEL Output: Main PLL (PLL1) Setup ...

C66xx_0: GEL Output: PLL1 Setup for DSP @ 1000.0 MHz.

C66xx_0: GEL Output:            SYSCLK2 = 333.3333 MHz, SYSCLK5 = 200.0
 MHz.

```

```
C66xx_0: GEL Output:            SYSCLK8 = 15.625 MHz.

C66xx_0: GEL Output: PLL1 Setup... Done.

C66xx_0: GEL Output: Power on all PSC modules and DSP domains...

C66xx_0: GEL Output: Power on all PSC modules and DSP domains... Done.

C66xx_0: GEL Output: DDR3 PLL (PLL2) Setup ...

C66xx_0: GEL Output: DDR3 PLL Setup... Done.

C66xx_0: GEL Output: DDR3 Init begin (1333 auto)

C66xx_0: GEL Output: XMC Setup ... Done

C66xx_0: GEL Output:
DDR3 initialization is complete.

C66xx_0: GEL Output: DDR3 Init done

C66xx_0: GEL Output: DDR3 memory test... Started

C66xx_0: GEL Output: DDR3 memory test... Passed

C66xx_0: GEL Output: PLL and DDR3 Initialization completed(0) ...

C66xx_0: GEL Output: configSGMIISerdes Setup... Begin

C66xx_0: GEL Output: SGMII SERDES has been configured.

C66xx_0: GEL Output: Enabling EDC ...

C66xx_0: GEL Output: L1P error detection logic is enabled.

C66xx_0: GEL Output: L2 error detection/correction logic is enabled.

C66xx_0: GEL Output: MSMC error detection/correction logic is enabled.

C66xx_0: GEL Output: Enabling EDC ...Done

C66xx_0: GEL Output: Global Default Setup... Done.

Start writing eeprom50
Writer:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/eepromwriter_evm
```

```
Image:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/eeprom50.bin

C66xx_0: GEL Output: Invalidate All Cache...

C66xx_0: GEL Output: Invalidate All Cache... Done.

C66xx_0: GEL Output: GEL Reset...

C66xx_0: GEL Output: GEL Reset... Done.

C66xx_0: GEL Output: Disable all EDMA3 interrupts and events.

EEPROM Writer Utility Version 01.00.00.05

Writing 59128 bytes from DSP memory address 0x0c000000 to EEPROM bus
address 0x0050 starting from device address 0x0000 ...
Reading 59128 bytes from EEPROM bus address 0x0050 to DSP memory
address 0x0c010000 starting from device address 0x0000 ...
Verifying data read ...
EEPROM programming completed successfully
Start writing eeprom51
Writer:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/eepromwriter_evm

Image:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/eeprom51.bin

C66xx_0: GEL Output: Invalidate All Cache...

C66xx_0: GEL Output: Invalidate All Cache... Done.

C66xx_0: GEL Output: GEL Reset...

C66xx_0: GEL Output: GEL Reset... Done.

C66xx_0: GEL Output: Disable all EDMA3 interrupts and events.

EEPROM Writer Utility Version 01.00.00.05

Writing 47848 bytes from DSP memory address 0x0c000000 to EEPROM bus
address 0x0051 starting from device address 0x0000 ...
Reading 47848 bytes from EEPROM bus address 0x0051 to DSP memory
address 0x0c010000 starting from device address 0x0000 ...
Verifying data read ...
EEPROM programming completed successfully
Writer:C:\Program Files\Texas
```

```
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/nandwriter_evm66

NAND:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/nand.bin

Required NAND files does not exist in C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/

Writer:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/norwriter_evm665

NOR:C:\Program Files\Texas
Instruments\mcsdk_2_01_00_03\tools\program_evm/binaries/evm6657l/nor.bin

C66xx_0: GEL Output: Invalidate All Cache...

C66xx_0: GEL Output: Invalidate All Cache... Done.

C66xx_0: GEL Output: GEL Reset...

C66xx_0: GEL Output: GEL Reset... Done.

C66xx_0: GEL Output: Disable all EDMA3 interrupts and events.

Start loading nor.bin
Start programming NOR
2012_08_23_165121
NOR Writer Utility Version 01.00.00.03

Flashing sector 0 (0 bytes of 664115)
Flashing sector 1 (65536 bytes of 664115)
Flashing sector 2 (131072 bytes of 664115)
Flashing sector 3 (196608 bytes of 664115)
Flashing sector 4 (262144 bytes of 664115)
Flashing sector 5 (327680 bytes of 664115)
Flashing sector 6 (393216 bytes of 664115)
Flashing sector 7 (458752 bytes of 664115)
Flashing sector 8 (524288 bytes of 664115)
Flashing sector 9 (589824 bytes of 664115)
Flashing sector 10 (655360 bytes of 664115)
Reading and verifying sector 0 (0 bytes of 664115)
Reading and verifying sector 1 (65536 bytes of 664115)
Reading and verifying sector 2 (131072 bytes of 664115)
Reading and verifying sector 3 (196608 bytes of 664115)
Reading and verifying sector 4 (262144 bytes of 664115)
Reading and verifying sector 5 (327680 bytes of 664115)
Reading and verifying sector 6 (393216 bytes of 664115)
Reading and verifying sector 7 (458752 bytes of 664115)
Reading and verifying sector 8 (524288 bytes of 664115)
Reading and verifying sector 9 (589824 bytes of 664115)
Reading and verifying sector 10 (655360 bytes of 664115)
NOR programming completed successfully
End programming NOR
```

Note: For EVMs without Security Accelerator components, PSC errors will show up due to a known issue in GEL file. The PSC errors can be ignored and are not fatal. The program EVM will proceed and complete successfully.

```

program_evm>%DSS_SCRIPT_DIR%\dss.bat program_e
vm.js evmk2h-le nor
board: evmk2h
endian: Little
emulation: XDS2xx emulator
binaries:
C:\TI\mcsdk_bios_3_01_01_04\tools\program_evm/binaries/evmk2h/
ccxml: C:\Users\z1105417\ti\CCSTargetConfigurations\hawking_evm.ccxml
C66xx_0: GEL Output:
Connecting Target...

C66xx_0: GEL Output: TCI6638K2K GEL file Ver is 1.3

C66xx_0: GEL Output: Detected PLL bypass enabled: SECCTL[BYPASS] =
0x00800000

C66xx_0: GEL Output: (2a) MAINPLLCTL1 = 0x00000040

C66xx_0: GEL Output: (2b) PLLCTL = 0x00000048

C66xx_0: GEL Output: (2c) PLLCTL = 0x00000048

C66xx_0: GEL Output: (2d) Delay...

C66xx_0: GEL Output: (2e) SECCTL = 0x00810000

C66xx_0: GEL Output: (2f) PLLCTL = 0x0000004A

C66xx_0: GEL Output: (2g) Delay...

C66xx_0: GEL Output: (2h) PLLCTL = 0x00000048

C66xx_0: GEL Output: (4)PLLM[PLLM] = 0x0000000F

C66xx_0: GEL Output: MAINPLLCTL0 = 0x05000000
```

```
C66xx_0: GEL Output: (5) MAINPLLCTL0 = 0x07000000

C66xx_0: GEL Output: (5) MAINPLLCTL1 = 0x00000040

C66xx_0: GEL Output: (6) MAINPLLCTL0 = 0x07000000

C66xx_0: GEL Output: (7) SECCTL = 0x00890000

C66xx_0: GEL Output: (8a) Delay...

C66xx_0: GEL Output: PLL1_DIV3 = 0x00008002

C66xx_0: GEL Output: PLL1_DIV4 = 0x00008004

C66xx_0: GEL Output: PLL1_DIV7 = 0x00000000

C66xx_0: GEL Output: (8d/e) Delay...

C66xx_0: GEL Output: (10) Delay...

C66xx_0: GEL Output: (12) Delay...

C66xx_0: GEL Output: (13) SECCTL = 0x00090000

C66xx_0: GEL Output: (Delay...

C66xx_0: GEL Output: (Delay...

C66xx_0: GEL Output: (14) PLLCTL = 0x00000041

C66xx_0: GEL Output: PLL has been configured (CLKIN * PLLM / PLLD /
PLLOD = PLLO
UT):

C66xx_0: GEL Output: PLL has been configured (122.88 MHz * 16 / 1 / 2 =
 983.04 M
Hz)

C66xx_0: GEL Output: Power on all PSC modules and DSP domains...

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=2, md=9!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=17, md=25!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=17, md=26!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=18, md=27!
```

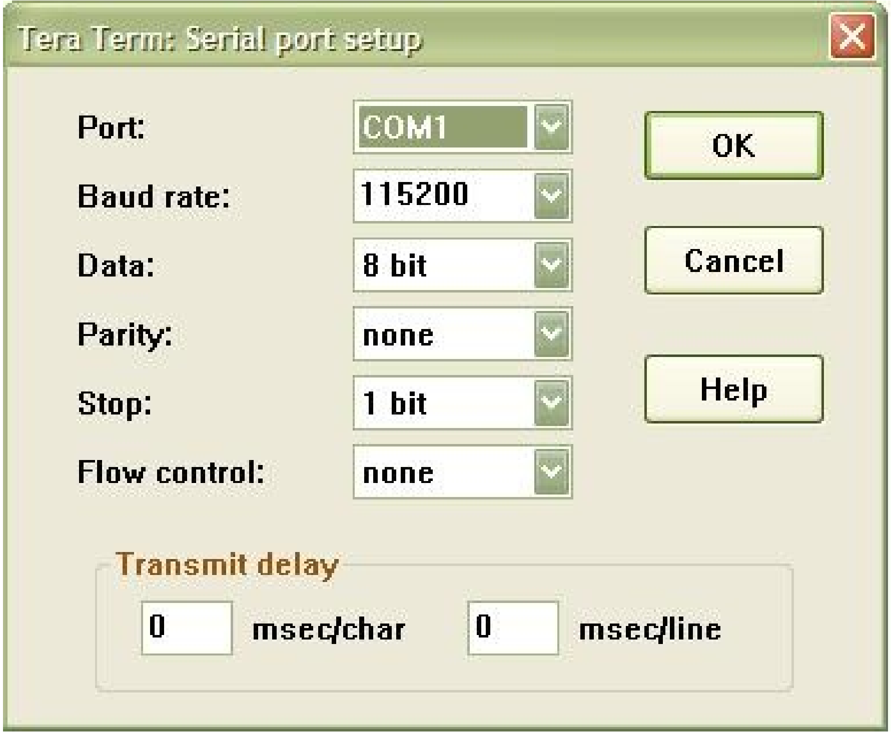
```
C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=19, md=28!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=19, md=29!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=20, md=30!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=20, md=31!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=20, md=32!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=20, md=33!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=21, md=34!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=22, md=35!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=22, md=36!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=23, md=37!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=23, md=38!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=24, md=39!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=24, md=40!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=24, md=41!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=24, md=42!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=25, md=43!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=25, md=44!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=25, md=45!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=25, md=46!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=26, md=47!

C66xx_0: GEL Output: Set_PSC_State... Timeout Error #03 pd=27, md=48!

C66xx_0: GEL Output: Power on all PSC modules and DSP domains... Done.

C66xx_0: GEL Output: WARNING: SYSCLK is the input to the PA PLL.
```

```
C66xx_0: GEL Output: Completed PA PLL Setup

C66xx_0: GEL Output: PAPLLCTL0 - before: 0x0x098804C0    after:
0x0x07080400

C66xx_0: GEL Output: PAPLLCTL1 - before: 0x0x00000040    after:
0x0x00002040

C66xx_0: GEL Output: DDR begin

C66xx_0: GEL Output: XMC setup complete.

C66xx_0: GEL Output: DDR3 PLL (PLL2) Setup ...

C66xx_0: GEL Output: DDR3 PLL Setup complete, DDR3A clock now running
at 666 MHz
.

C66xx_0: GEL Output: DDR3A initialization complete

C66xx_0: GEL Output: DDR3 PLL Setup ...

C66xx_0: GEL Output: DDR3 PLL Setup complete, DDR3B clock now running
at 800MHz.


C66xx_0: GEL Output: DDR3B initialization complete

C66xx_0: GEL Output: DDR done

Writer:C:\TI\mcsdk_bios_3_01_01_04\tools\program_evm/binaries/evmk2h/norwriter_e
vmk2h.out

NOR:C:\TI\mcsdk_bios_3_01_01_04\tools\program_evm/binaries/evmk2h/nor.bin

Start loading nor.bin
Start programming NOR
2014_12_15_81009
NOR Writer Utility Version 01.00.00.03

Flashing sector 0 (0 bytes of 393216)
Flashing sector 1 (65536 bytes of 393216)
Flashing sector 2 (131072 bytes of 393216)
Flashing sector 3 (196608 bytes of 393216)
Flashing sector 4 (262144 bytes of 393216)
Flashing sector 5 (327680 bytes of 393216)
Reading and verifying sector 0 (0 bytes of 393216)
Reading and verifying sector 1 (65536 bytes of 393216)
Reading and verifying sector 2 (131072 bytes of 393216)
Reading and verifying sector 3 (196608 bytes of 393216)
Reading and verifying sector 4 (262144 bytes of 393216)
Reading and verifying sector 5 (327680 bytes of 393216)
NOR programming completed successfully
End programming NOR
```

Verification

Serial Port Setup

Connect the RS232 Serial cable provided in the box to the serial port of the Host PC. If Host is running Windows OS, start tera term and configure the serial port settings as follows.



Tera Term: Serial port setup

Port: COM1

Baud rate: 115200

Data: 8 bit

Parity: none

Stop: 1 bit

Flow control: none

Transmit delay

0 msec/char

0 msec/line

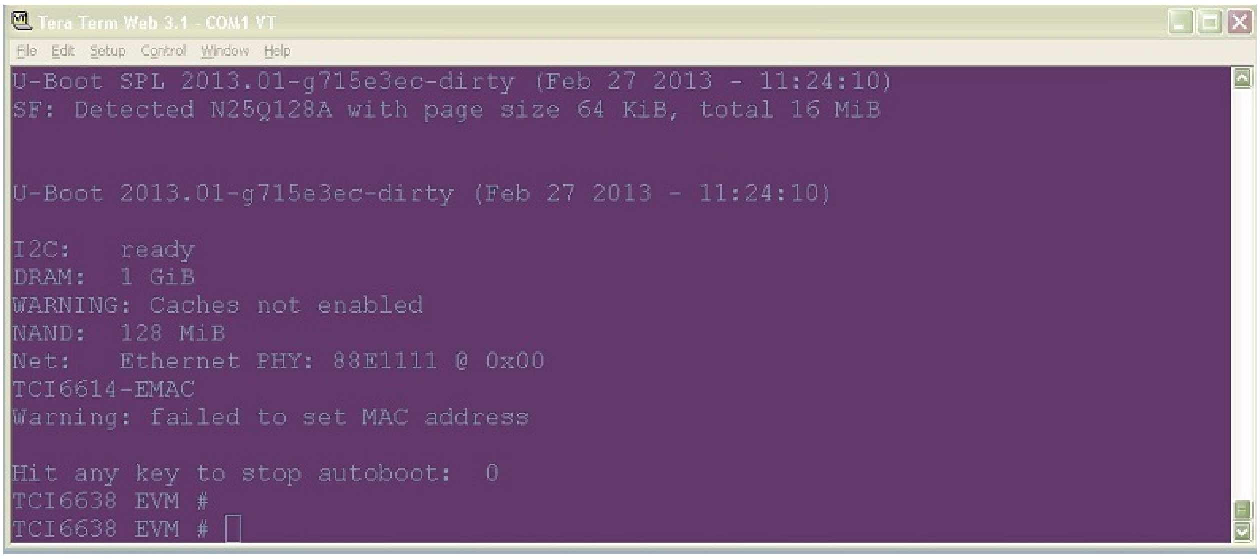
Verifying NOR

EVMK2H, EVMK2E, EVMK2L

1. Set the dip switches as below to set SPI boot mode

SWITCH	Pin1	Pin2	Pin3	Pin4
SW1	**Off**	**Off**	On	**Off**

2. Power Cycle the board.
3. Make sure the evm is connected to the DHCP server.
4. U-Boot will show up on the UART. A sample screen is shown below.


Tera Term Web 3.1 - COM1 VT
File Edit Setup Control Window Help

```
U-Boot SPL 2013.01-g715e3ec-dirty (Feb 27 2013 - 11:24:10)
SF: Detected N25Q128A with page size 64 KiB, total 16 MiB


U-Boot 2013.01-g715e3ec-dirty (Feb 27 2013 - 11:24:10)

I2C:   ready
DRAM:  1 GiB
WARNING: Caches not enabled
NAND:  128 MiB
Net:   Ethernet PHY: 88E1111 @ 0x00
TCI6614-EMAC
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
TCI6638 EVM #
TCI6638 EVM #
```

Verifying NAND

EVMK2H, EVMK2E & EVMK2L

NOR(U-Boot) must be flashed prior to performing any of the steps below.

For EVMK2H, EVMK2E & EVMK2L it is necessary to reformat the NAND Flash prior to burning the image. To do this, use the following command:

```
%DSS_SCRIPT_DIR%\dss.bat program_evm.js evmk2h format-nand
```

After entering the command, burn the NAND(Linux) and perform the steps below to verify.

1. Set the dip switches as below to set SPI boot mode

SWITCH	Pin1	Pin2	Pin3	Pin4
SW1	**Off**	**Off**	On	**Off**

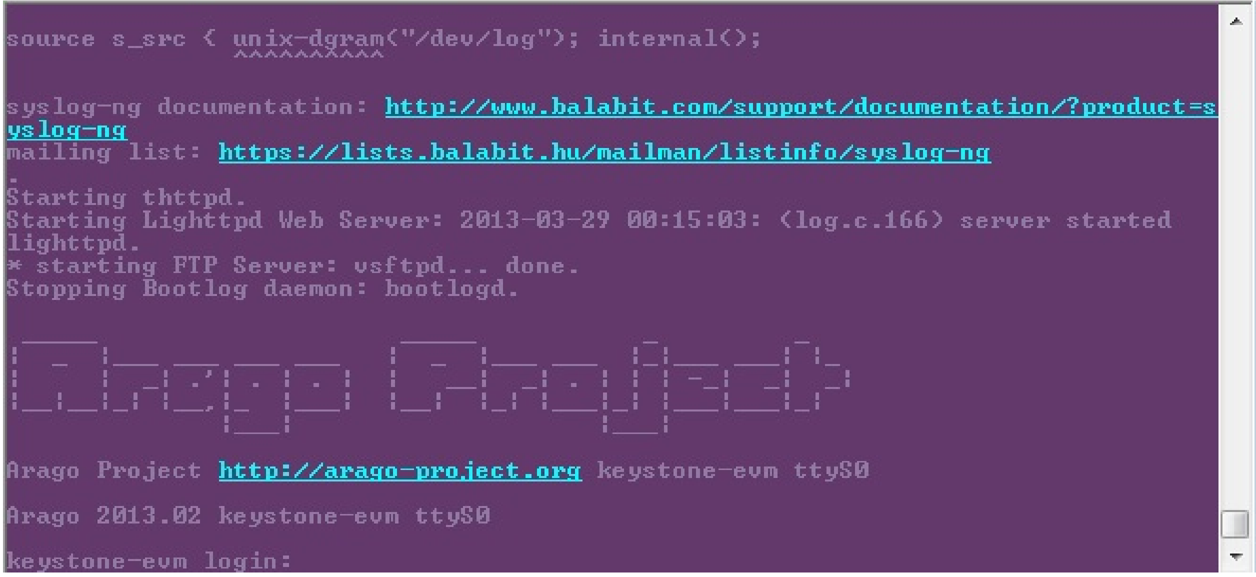
2. Power Cycle the board. U-Boot will show up on the UART.
3. Type the following commands into U-Boot:

```
env default —f —a
setenv boot ubi
boot
```

Note: If you are using EVM rev >= 2.0, also make sure to set the following commands to allow DSP reserved memory to be from 0xA000_0000 – 0xFFFF_FFFF. This is to match the DSP memory map reserved in default device tree and used by demo applications. (The mem_reserve variable is used to set reserved area from available memory within the 32 bit address space. The default mem_reserve boot variable is set to 512M. On a 2.0 EVM with >= 2GB memory, this will make the reserved memory to be 0xE000_0000 - 0xFFFF_FFFF. This will not work with the Out of box demos and default device tree currently. Need to change the reservation to 1536 M to reserve 0xA000_0000 - 0xFFFF_FFFF to allow this to work.)

```
setenv mem_reserve 1536M
saveenv
```

4. A sample screen is shown below.



```
source s_src { unix-dgram("/dev/log"); internal();
               ^^^^^^^^^^

syslog-ng documentation: http://www.balabit.com/support/documentation/?product=s
yslog-ng
mailing list: https://lists.balabit.hu/mailman/listinfo/syslog-ng
.
Starting thttpd.
Starting Lighttpd Web Server: 2013-03-29 00:15:03: (log.c.166) server started
lighttpd.
* starting FTP Server: vsftpd... done.
Stopping Bootlog daemon: bootlogd.

 _____                    _____           _         _
|  _  |___ ___ ___ ___   |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . |  |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___|  |__|  |_| |___|_| |___|___|_|
              |___|                    |___|

Arago Project http://arago-project.org keystone-evm ttyS0

Arago 2013.02 keystone-evm ttyS0

keystone-evm login:
```

MCSDK UG Chapter Tools

MCSDK User Guide: Tools and Development Environment

Last updated: **09/20/2015**

Overview

This Chapter provides an overview of the various tools and development environment used by the MCSDK along with how to install and configure them. Topics include how to build the MCSDK for ARM/DSP and the different debuggers that can be used for development.

Acronyms

The following acronyms are used throughout this chapter.

Acronym	Meaning
CCS	Texas Instruments Code Composer Studio
DSP	Digital Signal Processor
DVT	Texas Instruments Data Analysis and Visualization Technology
EVM	Evaluation Module, hardware platform containing the Texas Instruments DSP
JTAG	Joint Test Action Group
MCSA	Texas Instruments Multi-Core System Analyzer
MCSDK	Texas Instruments Multi-Core Software Development Kit
RTSC	Eclipse Real-Time Software Components
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter

Development Environments

For ARM/DSP development on Linux, **Ubuntu 12.04 64-bit** is the recommended host OS. Other recent 64-bit Linux distributions are expected to work but may require extra Linux knowledge. Older Linux distributions (e.g., Red Hat Enterprise Linux 4) would require extra work and updated software and is not recommended. Building on non-Linux hosts is not supported at all; use a virtual Linux host if you must use a Mac.

For DSP development on Windows, **Windows 7** is the recommended host OS.

Linux

Linaro toolchain

To build Linux kernel and filesystem, you will need to install the Linaro GCC 4.7 toolchain version 13.03 into your Linux host.

Download the **Linaro tool chain** from https:/ / launchpad. net/ linaro-toolchain-binaries/ trunk/ 2013. 03/ + download/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2

Untar the **gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2** into your HOME directory

```
$cd ~/
$tar xjf gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux.tar.bz2
```

To use this toolchain you need to set two environment variables:

```
export CROSS_COMPILE=arm-linux-gnueabihf-
```

Note: Use the comand above to set the CROSS_COMPILE variable if it was changed to build file system with arago tools.

```
export ARCH=arm
```

and add the toolchain path:

```
PATH=$HOME/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux/bin:$PATH
```

To help you get started quickly, the MCSDK package comes with pre-built binaries. However, after making any changes to the Linux Kernel you need to cross-compile them using the Linaro toolchain and use the new binaries that are generated.

Installation

Installing the Multicore Software Development Kit

After installing CCS, the next step is to install the MCSDK. See the Downloads [21] chapter for information on how to get the latest version of this software. The release contents are delivered as installers. Some notes:

- To avoid manually updating file locations, it is recommended to install the MCSDK in the same directory as CCSv5
- For a Linux host, run the mcsdk_<build-id>_setuplinux.bin executable.
- For a Windows host, run the mcsdk_<build-id>_windows_setup.exe executable.
- Some of the components installed by CCS may be updated by the MCSDK installer (e.g., SYS/BIOS); see the MCSDK release notes for which installer each component is mapped to.

The pre-built binaries (u-boot, linux kernel, filesystem ...) are provided in the release under the folder mcsdk_linux_<version>.

The following is a more detailed listing of the contents of this folder:

```
release_folder
      ti
            mcsdk_linux_<version>
                  bin
                  board-support
                        linux-src
                        uboot-src
                  boot
                  docs
                  example-applications
                  host-tools
                        loadlin
                        u-boot
                  images
                  linux-devkit
                  patches
                  post
```

- bin
- host-tools/loadlin

contains dss scripts and collaterals for loading and running Linux kernel on the Simulator

- loadlin.xsl
- tci6638-evm.ccxml- EVM configuration file for ccs
- loadlin-evm-kern.js - dss script that loads and runs Linux kernel on EVM.
- loadlin-evm-uboot.js - dss script that loads and runs U-boot on EVM.
- tracelog.xml - trace file

- images

uImage-keystone-evm.bin	A precompiled Linux kernel image suitable for loading on to the EVM (image built with CONFIG_PREEMPT_NONE=y Kconfig option)
uImage-k2hk-evm.dtb (DTB corresponding to CONFIG_PREEMPT_NONE image)	device Tree blob for EVM
u-boot-keystone-evm.bin	A precompiled U-boot image suitable for loading on to the EVM through CCS
u-boot-spi-keystone-evm.gph	SPI NOR flash combined SPL and Second stage u-boot image
u-boot-spl-keystone-evm.bin	SPI NOR flash SPL (first stage) u-boot binary.
u-boot-keystone-evm.img	SPI NOR flash Second stage u-boot image.
arago-console-image.tar.gz	A console root file system (min file system) gzipped tar file
arago-console-image.cpio.gz	A console root file system image (min file system) for ramfs boot
tisdk-rootfs.tar.gz	ti root file system gzipped tar file (may be used as NFS rootfs)
tisdk-rootfs-keystone-evm.ubifs	ti rootfs ubifs image
keystone-evm-ubifs.ubi	ubi image that includes rootfs and boot (kernel, dtb and boot monitor) ubifs images
keystone-evm-boot.ubifs	boot volume ubifs image with uImage, dtb and boot monitor images
skern-keystone-evm.bin	boot monitor image

Following images are for Linux Kernel images built with CONFIG_PREEMPT_RT_FULL=y Kconfig option set:-

- uImage-rt-keystone-evm.bin

  A precompiled Linux kernel image suitable for loading on to the EVM

- uImage-rt-k2hk-evm.dtb

  device Tree blob for EVM

- tisdk-rootfs-rt.tar.gz

  ti root file system gzipped tar file (may be used as NFS rootfs)

- tisdk-rootfs-rt.ubifs

  ti rootfs ubifs image

- keystone-evm-ubifs-rt.ubi

  ubi image that includes rootfs and boot (kernel, dtb and boot monitor) ubifs images

- keystone-evm-boot-rt.ubifs

  boot volume ubifs image with uImage, dtb and boot monitor images

- boot
- host-tools

Other folders doesn't have any contents as of now, but may be filled in future releases.

Loading and Running Application on the target

ARM

Loading and Running U-Boot using CCS

This section describes how to boot up U-Boot using CCS and is useful when the U-Boot image is not programmed on the SPI NOR flash.

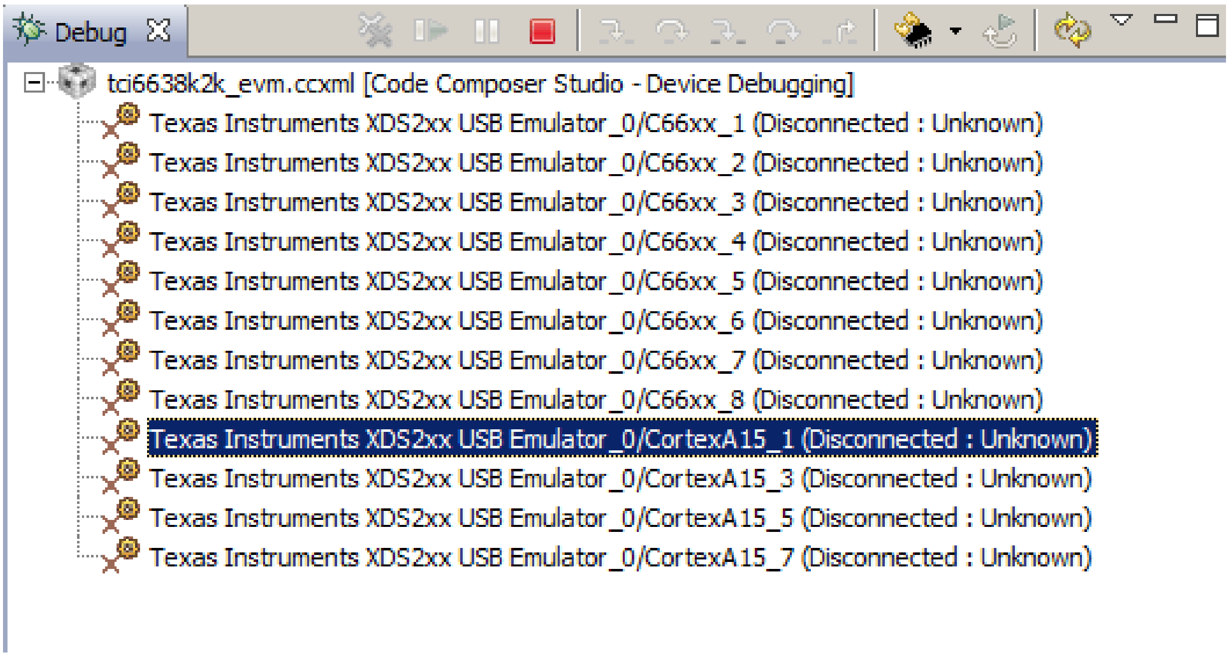
- Rev 1.0 EVMs have a U-Boot image programmed on the SPI NOR flash. This section may be skipped for Rev 1.0 EVM users
- Rev 0B EVMs do not have a U-boot programmed on the SPI NOR flash. The procedure described in this section must be followed to load U-boot to the target.

HW Configuration

It is assumed that the user has configured the EVM with "No Boot/JTAG DSP Little Endian Boot mode" as described in [4]

Loading U-Boot with CCS

- Power on the EVM
- Once the "Boot Complete" message is shown at the console (on the EVM), Start CCS Application and Launch target configuration as shown.



```
tci6638k2k_evm.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS2xx USB Emulator_0/C66xx_1 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_2 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_3 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_4 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_5 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_6 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_7 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_8 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_1 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_3 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_5 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_7 (Disconnected : Unknown)
```

If you do not have a target configuration for the EVM, you have to create one. Create a new target configuration for the TCI6638 device using the "Texas Instruments XDS2xx USB Emulator" connection and save the same. You have to do that only once. Connect CCS to the CortexA15_1 target.

Load the U-Boot binary, u-boot-keystone-evm.bin available under release folder to MSMC SRAM memory at 0xc001000 through CCS as follows:

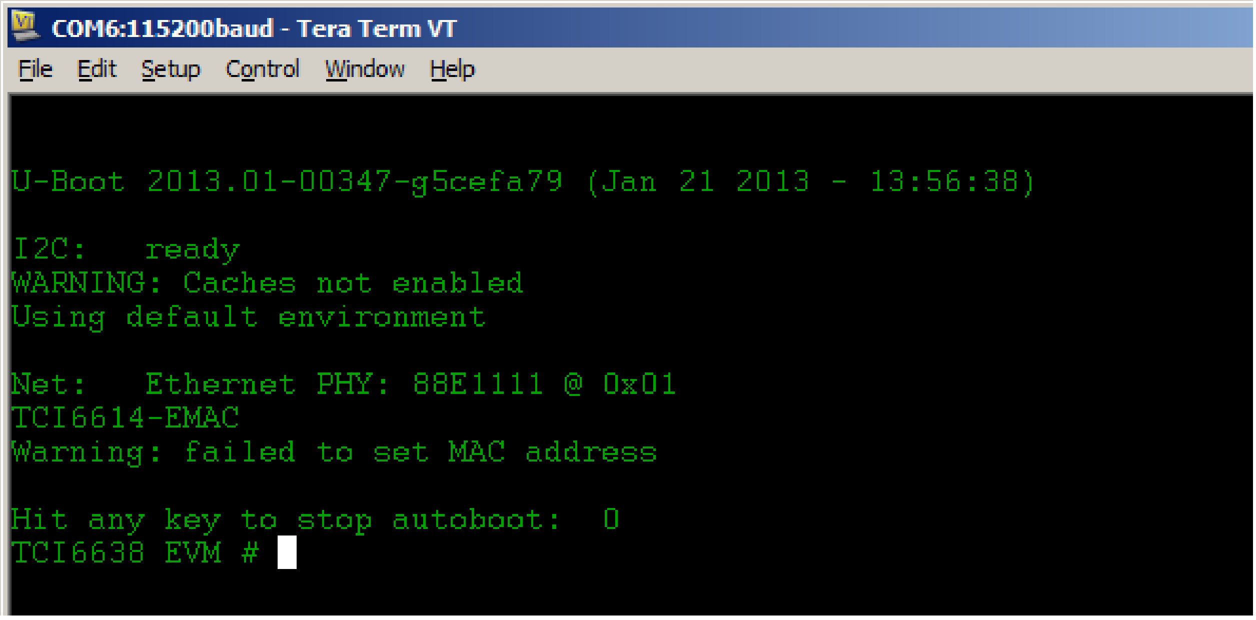
- In CCS menu:Tools->Load Memory
- Set path to u-boot-keystone-evm.bin, click Next
- Start Address:0xc001000
- Type-size: 32 bits. Click Finish

Set PC to 0xc001000 as follows:

- View->Registers
- Expand Core Registers to see PC

Run U-Boot:

- Click Resume(F8)
- In the terminal window, type any key to stop autoboot and get a U-Boot prompt.


COM6:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
U-Boot 2013.01-00347-g5cefa79 (Jan 21 2013 - 13:56:38)

I2C:   ready
WARNING: Caches not enabled
Using default environment

Net:   Ethernet PHY: 88E1111 @ 0x01
TCI6614-EMAC
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
TCI6638 EVM #
```

Programming SPI NOR flash with U-Boot GPH image

This section describes how to erase the SPI NOR flash and program the same with U-Boot image to do two stage SPI boot of EVM. It is mandatory that user updates the U-Boot image to the latest version provided in the release before booting up Linux provided in the release. In this section Both SPI boot mode and Uboot mode (term user in QSG) refers to the booting U-Boot using U-Boot image programmed on SPI NOR flash. Two stage SPI boot is assumed in this section.

Two stage SPI boot consists of booting first the U-Boot SPL binary from SPI NOR flash (located at offset 0) which will then load and run the second level boot loader (Full version of U-Boot) image from SPI NOR flash located at offset 0x10000. Once the boot mode is set to SPI boot mode and EVM is powered ON, RBL (ROM Boot loader) first loads the small (about 10K) SPL binary from SPI NOR flash to MSMC SRAM at address 0xc200000 and execute. This SPL U-Boot then loads the full version U-Boot image (u-boot-keystone-evm.img) from offset 0x10000 of SPI NOR flash to MSMC SRAM at address 0xc001000 and execute. Two stage SPI boot is preferred over single stage boot since it reduces the boot time. Single stage boot is slow since the RBL runs at a slower CORE PLL clock in SPI boot mode and hence the SPI data rate is low. So a small U-Boot SPL binary is loaded by RBL at the first stage. The SPL U-Boot code programs the ARM Tetris and Core PLL to higher clock rate. After this it loads the second stage boot loader from SPI NOR flash at a higher SPI data rate. As the bigger image is transferred at a higher SPI data rate, overall boot up time is reduced compared to one stage boot. One stage boot may take around 15 seconds.

There are two cases to deal with:-

- EVM doesn't have a u-boot image flashed on SPI NOR flash (either Rev 0B board or board is Rev 1.0, but board doesn't boot to u-boot prompt. If this is the case, goto the previous section to load u-boot image through CCS)
- EVM has u-boot image on SPI NOR flash(Rev 1.0). Configure the SW1 switch on the board for SPI Little Endian Boot mode as described in [2]. Connect SoC and BMC UART ports of EVM to PC and start a Tera Term to get access to the COM port consoles as per instructions in [1]. Please note that U-Boot software is

configured at build time to use Ethernet Port 0 (assuming 2 ports named as Port 0 and Port 1) in the default u-boot image on Rev 1.0 boards. So connect Ethernet cable to Port 0. (ENET0) If user is running U-Boot images, Port 0 should be used for connecting Ethernet cable. Power on the EVM. U-Boot boot log can be observed at the SoC UART port. A sample boot log is shown in the previous section.

At this point, it is assumed that user is running U-Boot on the EVM either through CCS or using the SPI NOR boot mode and U-Boot console is available to accept commands from the user.

Following steps are required to program the combined SPI GPH image onto SPI NOR flash.

Copy u-boot-spi-keystone-evm.gph from Release to tftp server root directory. Start tftp server.

Issue following commands to U-Boot console. In the below commands tftp server ip is assumed to be 192.168.1.10. Replace it with the user's tftp server IP address before issuing the command.

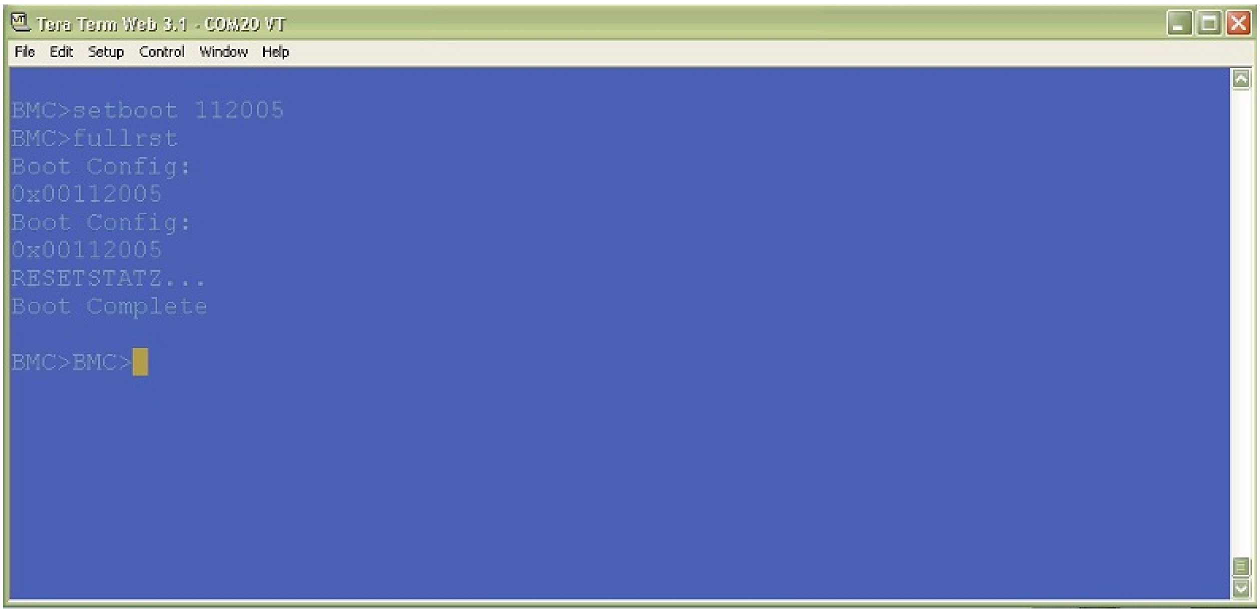
```
setenv serverip 192.168.1.10
dhcp 0xc300000 u-boot-spi-keystone-evm.gph
sf probe
sf erase 0 <size of u-boot-spi-keystone-evm.gph in hex rounded to
             sector boundary of 0x10000>
sf write 0xc300000 0 <size of u-boot-spi-keystone-evm.gph image in
             hex>
```

Note that size of the image will be be displayed as part of the dhcp command. EVM now can be booted using SPI NOR boot mode. Power off the EVM and follow instruction above to boot EVM in SPI NOR boot mode. Note that Ethernet port should now be switched to Port 0.

If you already have an Alpha9 or later u-boot image on the NOR flash, upgrade procedure for u-boot is simplified as follows:-

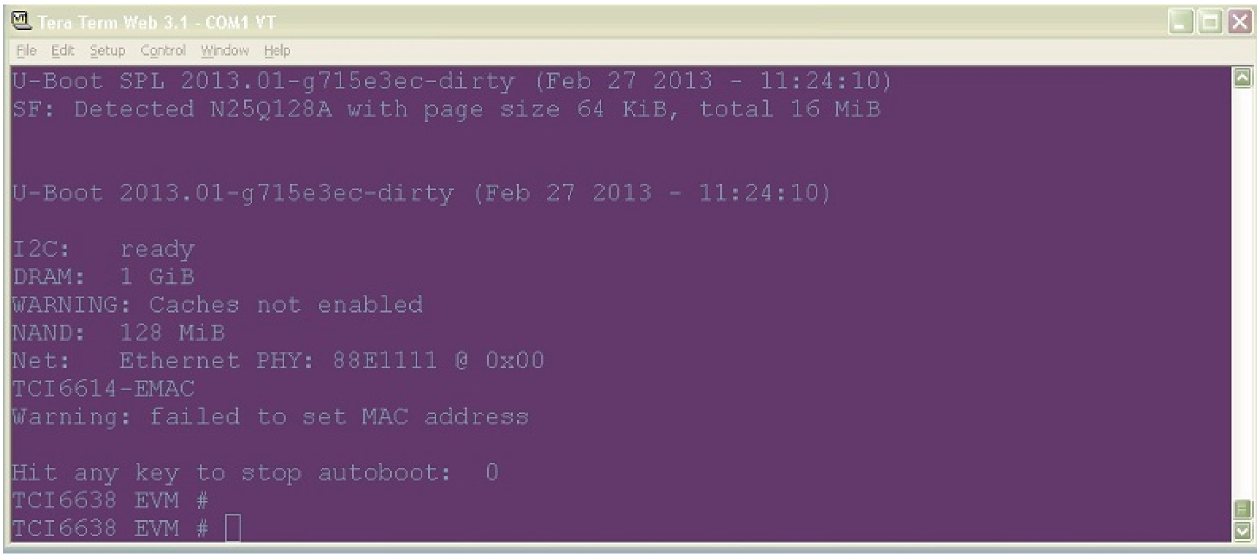
```
  env default -f -a
  setenv serverip 192.168.1.10
  setenv tftp_root <tftp root directory>
  setenv name_uboot <name of u-boot gph image under the tftp root>
  run get_uboot_net
  run burn_uboot
```

On a Rev 0B board, Ethernet (For example dhcp command) will not function unless following command is issued at the BMC UART console as shown in the figure below. For Rev 1.0 boards, please follow the BMC field updated procedure at [2] if Ethernet doesn't function.


Tera Term Web 3.1 - COM20 VT
File Edit Setup Control Window Help

```
BMC>setboot 112005
BMC>fullrst
Boot Config:
0x00112005
Boot Config:
0x00112005
RESETSTATZ...
Boot Complete

BMC>BMC>
```

A sample u-boot console boot up log is shown below.


Tera Term Web 3.1 - COM1 VT
File Edit Setup Control Window Help

```
U-Boot SPL 2013.01-g715e3ec-dirty (Feb 27 2013 - 11:24:10)
SF: Detected N25Q128A with page size 64 KiB, total 16 MiB


U-Boot 2013.01-g715e3ec-dirty (Feb 27 2013 - 11:24:10)

I2C:   ready
DRAM:  1 GiB
WARNING: Caches not enabled
NAND:  128 MiB
Net:   Ethernet PHY: 88E1111 @ 0x00
TCI6614-EMAC
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
TCI6638 EVM #
TCI6638 EVM #
```

Loading and Running Linux Kernel

The U-Boot comes with a set of default environment variables. Using those variables simplify several ways to boot kernel. Use the following command to set the default environment variables:

```
>env default -f —a
>saveenv
```

U-Boot supports three ways to boot kernel and is configured through 'boot' env variable. For each of this method, the boot env variable value to be used is provided in parenthesis.

- tftp kernel and initrd ramfs (ramfs);
- tftp kernel and use nfs mounted filesystem (net);
- using ubifs (ubi);

You would need to add a few more variable specific to you setup:

- serverip - IP address of your tftp server
- tftp_root - Relative path from tftp server root directory where the image files are stored
- boot - one of three ways to boot kernel (ramfs, net, ubi)

- nfs_root - root directory exported by the NFS server (where the rootfs tar file was untarred)

Other defaults may be changed as needed. By default, these are set to use the image names from the Release. However for Preempt RT Full images, change the following env variables (applicable only for tftp boot of Linux and DTB):-

```
>setenv name_kern uImage-rt-keystone-evm.bin
>setenv name_fdt uImage-rt-k2hk-evm.dtb
```

Loading and Running Linux Kernel using tftp with initrd file system

This section describes how to load and boot Linux over ethernet using tftp protocol. The initrd is used as the rootfs for this boot.

Set the "boot" environment variable to "ramfs". Copy the images from the release folder to tftp server root directory (in this example, "release" is the root directory) and start the tftp server. Tftp server IP of 192.168.1.10 is used the example which is to be replaced with the ip address of the user's tftp server. Note also that name_fs env variable value has to be changed to the name of the initrd file used (use file from the release folder). Below commands assume the rootfs file name as 'rootfs-cpio.gz' and size as less than 9M. If the size is different user needs to update the value in the args_ramfs.

If the user is provided with a .cpio file in the SDK, following commands at the Linux shell can be used to convert it to .gz format:-

```
>gzip <cpio file name>
```

For example

```
>gzip test.cpio
```

This will produce test.cpio.gz.

Make sure that DDR3A is configured using the following commands at the BMC console on a REV 0B board (This may be skipped for Rev 1.0 EVM).

```
>setboot 112005
>fullrst
```

At this time, U-Boot boots up. Type esc to stop auto boot. Type the following commands at the U-Boot console.

```
>setenv serverip 192.168.1.10
>setenv boot ramfs
>setenv tftp_root release
```

For Preempt RT Full images, change the following env variables:-

```
>setenv name_kern uImage-rt-keystone-evm.bin
>setenv name_fdt uImage-rt-k2hk-evm.dtb
```

```
>saveenv
>boot
```

A sample log is shown below for reference

```
TCI6638 EVM # boot
BOOTP broadcast 1
DHCP client bound to address 192.168.50.101
Using TCI6614-EMAC device
```

```
TFTP from server 192.168.50.100; our IP address is 192.168.50.101
Filename '//rootfs.cpio.gz'.
Load address: 0x82000000
Loading:
#################################################################

#################################################################

#################################################################

#################################################################
         ################
         720.7 KiB/s
done
Bytes transferred = 4039802 (3da47a hex)
BOOTP broadcast 1
DHCP client bound to address 192.168.50.101
Using TCI6614-EMAC device
TFTP from server 192.168.50.100; our IP address is 192.168.50.101
Filename '//k2hk-evm.dtb'.
Load address: 0x87000000
Loading: ##
         538.1 KiB/s
done
Bytes transferred = 29233 (7231 hex)
BOOTP broadcast 1
DHCP client bound to address 192.168.50.101
Using TCI6614-EMAC device
TFTP from server 192.168.50.100; our IP address is 192.168.50.101
Filename '//skern.bin'.
Load address: 0xc5f0000
Loading: ####
         676.8 KiB/s
done
Bytes transferred = 45056 (b000 hex)
BOOTP broadcast 1
DHCP client bound to address 192.168.50.101
Using TCI6614-EMAC device
TFTP from server 192.168.50.100; our IP address is 192.168.50.101
Filename '//uImage'.
Load address: 0x88000000
Loading:
#################################################################

#################################################################

#################################################################
```

```
         ##########################
         719.7 KiB/s
done
Bytes transferred = 3240032 (317060 hex)
## installed monitor, freq [133120000], status 133120000
## Booting kernel from Legacy Image at 88000000 ...
   Image Name:   Linux-3.6.6-rt17-24126-g74d6a5f-
   Created:      2013-03-25  13:55:15 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3239968 Bytes = 3.1 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
   Booting using the fdt blob at 0x87000000
   Loading Kernel Image ... OK
OK
   Using Device Tree in place at 87000000, end 8700a230

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0
[    0.000000] Linux version 3.6.6-rt17-24126-g74d6a5f-dirty
(a0794637@uvma0794637) (gcc version 4.7.2 20120701 (prerelease)
(crosstool-NG linaro-1.13.1-2012.07-20120720 - Linaro GCC 2012.07) ) #7
 SMP PREEMPT Mon Mar 25 09:55:03 EDT 2013
[    0.000000] CPU: ARMv7 Processor [412fc0f4] revision 4 (ARMv7),
cr=30c7387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT
instruction cache
[    0.000000] Machine: KeyStone2, model: Texas Instruments Keystone 2
SoC
[    0.000000] switching to high address space at 0x800000000
[    0.000000] cma: CMA: reserved 16 MiB at 1f000000
[    0.000000] Memory policy: ECC disabled, Data cache writealloc
[    0.000000] PERCPU: Embedded 8 pages/cpu @c0a49000 s11840 r8192
d12736 u32768
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.
Total pages: 130048
[    0.000000] Kernel command line: console=ttyS0,115200n8 rootwait=1
earlyprintk rdinit=/sbin/init rw root=/dev/ram0 initrd=0x802000000,9M
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144
 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072
bytes)
[    0.000000] Memory: 512MB = 512MB total
[    0.000000] Memory: 487548k/487548k available, 36740k reserved, 0K
highmem
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff000000   ( 488 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc059eca0   (5724 kB)
[    0.000000]       .init : 0xc059f000 - 0xc05d6e40   ( 224 kB)
[    0.000000]       .data : 0xc05d8000 - 0xc060e6c8   ( 218 kB)
[    0.000000]        .bss : 0xc060e6ec - 0xc0637f8c   ( 167 kB)
[    0.000000] SLUB: Genslabs=11, HWalign=64, Order=0-3, MinObjects=0,
CPUs=4, Nodes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] ipc irq: irqchip registered, range 512-539
[    0.000000] main_pll_clk rate is 798720000, postdiv = 2, pllm =
12,plld = 0
[    0.000000] tci6614-timer: no matching node
[    0.000000] arch_timer: found timer irqs 29 30
[    0.000000] Architected local timer running at 133.66MHz.
[    0.000000] Switching to timer-based delay loop
[    0.000000] sched_clock: 32 bits at 133MHz, resolution 7ns, wraps
every 32131ms
[    0.000000] Console: colour dummy device 80x30
[    0.000094] Calibrating delay loop (skipped), value calculated using
 timer frequency.. 267.33 BogoMIPS (lpj=1336666)
[    0.000112] pid_max: default: 4096 minimum: 301
[    0.000357] Mount-cache hash table entries: 512
[    0.010654] CPU: Testing write buffer coherency: ok
[    0.010847] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.010881] hw perfevents: enabled with ARMv7 Cortex-A15 PMU driver,
 7 counters available
[    0.010935] Setting up static identity map for 0x8040f478 -
0x8040f4ac
[    0.098773] CPU1: Booted secondary processor
[    0.098813] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.138592] CPU2: Booted secondary processor
[    0.138633] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.178671] CPU3: Booted secondary processor
[    0.178709] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.178878] Brought up 4 CPUs
[    0.178914] SMP: Total of 4 processors activated (1069.33 BogoMIPS).
[    0.202614] NET: Registered protocol family 16
[    0.220858] DMA: preallocated 256 KiB pool for atomic coherent
```

```
allocations
[    0.235019] hw-breakpoint: found 5 (+1 reserved) breakpoint and 4
watchpoint registers.
[    0.235034] hw-breakpoint: maximum watchpoint size is 8 bytes.
[    0.251719] bio: create slab <bio-0> at 0
[    0.252772] SCSI subsystem initialized
[    0.253498] usbcore: registered new interface driver usbfs
[    0.253663] usbcore: registered new interface driver hub
[    0.253883] usbcore: registered new device driver usb
[    0.255743] keystone-hwqueue hwqueue.2: qmgr start queue 0, number
of queues 8192
[    0.255940] keystone-hwqueue hwqueue.2: added qmgr start queue 0,
num of queues 8192, reg_peek e0840000, reg_status e0804000, reg_config
e0806000, reg_region e0808000, reg_push e0880000, reg_pop e08c0000
[    0.255958] keystone-hwqueue hwqueue.2: qmgr start queue 8192,
number of queues 8192
[    0.256140] keystone-hwqueue hwqueue.2: added qmgr start queue 8192,
 num of queues 8192, reg_peek e0900000, reg_status e080a400, reg_config
 e080c000, reg_region e080e000, reg_push e0940000, reg_pop e0980000
[    0.256982] keystone-hwqueue hwqueue.2: added pool pool-net: 2048
descriptors of size 128
[    0.257001] keystone-hwqueue hwqueue.2: added pool pool-rio: 128
descriptors of size 256
[    0.257020] keystone-hwqueue hwqueue.2: added pool pool-udma: 1636
descriptors of size 256
[    0.261097] keystone-hwqueue hwqueue.2: registered queues 0-16383
[    0.268032] keystone-pktdma 2004000.pktdma: registered 8 logical
channels, flows 32, tx chans: 9, rx chans: 24
[    0.275212] keystone-pktdma 2a08000.pktdma: registered 24 logical
channels, flows 32, tx chans: 32, rx chans: 32, loopback
[    0.275456] Switching to clocksource arch_sys_counter
[    0.302522] NET: Registered protocol family 2
[    0.303140] TCP established hash table entries: 16384 (order: 5,
131072 bytes)
[    0.303537] TCP bind hash table entries: 16384 (order: 5, 131072
bytes)
[    0.303943] TCP: Hash tables configured (established 16384 bind
16384)
[    0.303975] TCP: reno registered
[    0.303989] UDP hash table entries: 256 (order: 1, 8192 bytes)
[    0.304025] UDP-Lite hash table entries: 256 (order: 1, 8192 bytes)
[    0.304331] NET: Registered protocol family 1
[    0.304645] RPC: Registered named UNIX socket transport module.
[    0.304656] RPC: Registered udp transport module.
[    0.304665] RPC: Registered tcp transport module.
[    0.304674] RPC: Registered tcp NFSv4.1 backchannel transport
module.
```

```
[    0.304902] Unpacking initramfs...
[    0.697637] Initramfs unpacking failed: no cpio magic
[    0.705712] Freeing initrd memory: 9216K
[    0.826791] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[    0.827330] NTFS driver 2.1.30 [Flags: R/O].
[    0.827878] jffs2: version 2.2. (NAND) © 2001-2006 Red Hat, Inc.
[    0.830350] NET: Registered protocol family 38
[    0.830779] Block layer SCSI generic (bsg) driver version 0.4 loaded
 (major 254)
[    0.830792] io scheduler noop registered
[    0.830803] io scheduler deadline registered
[    0.831088] io scheduler cfq registered (default)
[    0.833388] keystone-udma udma0.3: registered udma device udma0
[    0.928798] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    0.930809] 2530c00.serial: ttyS0 at MMIO 0x2530c00 (irq = 309) is a
 16550A
[    1.546206] console [ttyS0] enabled
[    1.555595] loop: module loaded
[    1.558966] at24 0-0050: 131072 byte 24c1024 EEPROM, writable, 1
bytes/write
[    1.567633] Generic platform RAM MTD, (c) 2004 Simtec Electronics
[    1.586227] ONFI param page 0 valid
[    1.589690] ONFI flash detected
[    1.592811] NAND device: Manufacturer ID: 0x2c, Chip ID: 0xac
(Micron MT29F4G08ABBDAHC), page size: 2048, OOB size: 64
[    1.605082] Bad block table found at page 262080, version 0x01
[    1.614024] Bad block table found at page 262016, version 0x01
[    1.621599] 3 ofpart partitions found on MTD device 30000000.nand
[    1.627664] Creating 3 MTD partitions on "30000000.nand":
[    1.633030] 0x000000000000-0x000000100000 : "u-boot"
[    1.639264] 0x000000100000-0x000000180000 : "params"
[    1.645369] 0x000000180000-0x000008000000 : "ubifs"
[    1.651509] davinci_nand 30000000.nand: controller rev. 2.5
[    1.658339] spi_davinci 21000400.spi: master is unqueued, this is
deprecated
modprobe: FATAL: Could not load
/lib/modules/3.6.6-rt17-24126-g74d6a5f-dirty/modules.dep: No such file
or directory

[    1.696448] m25p80 spi32766.0: n25q128 (16384 Kbytes)
[    1.701482] 2 ofpart partitions found on MTD device spi32766.0
[    1.707291] Creating 2 MTD partitions on "spi32766.0":
[    1.712393] 0x000000000000-0x000000080000 : "u-boot-spl"
[    1.718867] 0x000000080000-0x000001000000 : "test"
[    1.724807] spi_davinci 21000400.spi: Controller at 0xe0836400
[    1.775558] davinci_mdio 2090300.mdio: davinci mdio revision 1.5
[    1.781519] libphy: 2090300.mdio: probed
[    1.792851] davinci_mdio 2090300.mdio: phy[0]: device
2090300.mdio:00, driver Marvell 88E1111
[    1.801333] davinci_mdio 2090300.mdio: phy[1]: device
2090300.mdio:01, driver Marvell 88E1111
[    1.812674] keystone-netcp 2090000.netcp: Created interface "eth0"
[    1.818844] keystone-netcp 2090000.netcp: dma_chan_name nettx0
[    1.825936] keystone-netcp 2090000.netcp: Created interface "eth1"
[    1.832081] keystone-netcp 2090000.netcp: dma_chan_name nettx1
[    1.839367] keystone-dwc3 2690000.dwc: usbss revision 47914300
[    1.845201] keystone-dwc3 2690000.dwc: mapped irq 425 to virq 572
[    2.055313] xhci-hcd xhci-hcd: xHCI Host Controller
[    2.060231] xhci-hcd xhci-hcd: new USB bus registered, assigned bus
number 1
[    2.068463] xhci-hcd xhci-hcd: irq 572, io mem 0x02690000
[    2.073962] usb usb1: New USB device found, idVendor=1d6b,
idProduct=0002
[    2.080724] usb usb1: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    2.087911] usb usb1: Product: xHCI Host Controller
[    2.092754] usb usb1: Manufacturer: Linux
3.6.6-rt17-24126-g74d6a5f-dirty xhci-hcd
[    2.100281] usb usb1: SerialNumber: xhci-hcd
[    2.105216] hub 1-0:1.0: USB hub found
[    2.108975] hub 1-0:1.0: 1 port detected
[    2.113182] xhci-hcd xhci-hcd: xHCI Host Controller
[    2.118056] xhci-hcd xhci-hcd: new USB bus registered, assigned bus
number 2
[    2.125189] usb usb2: New USB device found, idVendor=1d6b,
idProduct=0003
[    2.131948] usb usb2: New USB device strings: Mfr=3, Product=2,
SerialNumber=1
[    2.139132] usb usb2: Product: xHCI Host Controller
[    2.143974] usb usb2: Manufacturer: Linux
3.6.6-rt17-24126-g74d6a5f-dirty xhci-hcd
[    2.151504] usb usb2: SerialNumber: xhci-hcd
[    2.156443] hub 2-0:1.0: USB hub found
[    2.160176] hub 2-0:1.0: 1 port detected
[    2.164460] Initializing USB Mass Storage driver...
[    2.169524] usbcore: registered new interface driver usb-storage
[    2.175484] USB Mass Storage support registered.
[    2.180477] mousedev: PS/2 mouse device common for all mice
[    2.186425] i2c /dev entries driver
[    2.191517] usbcore: registered new interface driver usbhid
[    2.197068] usbhid: USB HID core driver
[    2.201383]  remoteproc0: 2620040.dsp0 is available
[    2.206289]  remoteproc0: Note: remoteproc is still under
development and considered experimental.
```

```
[    2.215190]  remoteproc0: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.225561]  remoteproc0: no firmware found
[    2.230123]  remoteproc1: 2620044.dsp1 is available
[    2.234964]  remoteproc1: Note: remoteproc is still under
development and considered experimental.
[    2.243874]  remoteproc1: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.254203]  remoteproc1: no firmware found
[    2.258829]  remoteproc2: 2620048.dsp2 is available
[    2.263671]  remoteproc2: Note: remoteproc is still under
development and considered experimental.
[    2.272581]  remoteproc2: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.282912]  remoteproc2: no firmware found
[    2.287502]  remoteproc3: 262004c.dsp3 is available
[    2.292343]  remoteproc3: Note: remoteproc is still under
development and considered experimental.
[    2.301254]  remoteproc3: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.311579]  remoteproc3: no firmware found
[    2.316186]  remoteproc4: 2620050.dsp4 is available
[    2.321027]  remoteproc4: Note: remoteproc is still under
development and considered experimental.
[    2.329943]  remoteproc4: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.340272]  remoteproc4: no firmware found
[    2.344823]  remoteproc5: 2620054.dsp5 is available
[    2.349684]  remoteproc5: Note: remoteproc is still under
development and considered experimental.
[    2.358589]  remoteproc5: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.368921]  remoteproc5: no firmware found
[    2.373476]  remoteproc6: 2620058.dsp6 is available
[    2.378337]  remoteproc6: Note: remoteproc is still under
development and considered experimental.
[    2.387249]  remoteproc6: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.397587]  remoteproc6: no firmware found
[    2.402136]  remoteproc7: 262005c.dsp7 is available
[    2.407002]  remoteproc7: Note: remoteproc is still under
development and considered experimental.
[    2.415909]  remoteproc7: THE BINARY FORMAT IS NOT YET FINALIZED,
and backward compatibility isn't yet guaranteed.
[    2.426234]  remoteproc7: no firmware found
[    2.431285] oprofile: using timer interrupt.
[    2.435882] Netfilter messages via NETLINK v0.30.
[    2.440572] nf_conntrack version 0.5.0 (8017 buckets, 32068 max)
[    2.447049] ctnetlink v0.93: registering with nfnetlink.
[    2.452818] IPv4 over IPv4 tunneling driver
[    2.457770] gre: GRE over IPv4 demultiplexor driver
[    2.462612] ip_gre: GRE over IPv4 tunneling driver
[    2.468398] ip_tables: (C) 2000-2006 Netfilter Core Team
[    2.473864] ipt_CLUSTERIP: ClusterIP Version 0.8 loaded successfully
[    2.480248] arp_tables: (C) 2002 David S. Miller
[    2.484888] TCP: cubic registered
[    2.488196] Initializing XFRM netlink socket
[    2.493505] NET: Registered protocol family 10
[    2.499061] NET: Registered protocol family 17
[    2.503495] NET: Registered protocol family 15
[    2.507923] 8021q: 802.1Q VLAN Support v1.8
[    2.512995] sctp: Hash tables configured (established 16384 bind
16384)
[    2.520315] NET: Registered protocol family 40
[    2.524956] VFP support v0.3: implementor 41 architecture 4 part 30
variant f rev 0
[    2.532601] Registering SWP/SWPB emulation handler
[    2.544507] Freeing init memory: 220K
INIT: version 2.86 booting
Please wait: booting...
Starting udev
WARNING: -e needs -E or -F
[    2.715249] udevd (1148): /proc/1148/oom_adj is deprecated, please
use /proc/1148/oom_score_adj instead.
[    4.245964] alignment: ignoring faults is unsafe on this CPU.
Defaulting to fixup mode.
Root filesystem already rw, not remounting
Caching udev devnodes
Populating dev cachemv: cannot rename '/tmp/devices': No such file or
directory
root: mount: mounting rootfs on / failed: No such file or directory
root: mount: mounting usbfs on /proc/bus/usb failed: No such file or
directory
Configuring network interfaces... [    5.026077] keystone-netcp
2090000.netcp: initializing cpsw version 1.3 (1) SGMII identification
value 0x4ed1
[    5.037459] keystone-netcp 2090000.netcp: Created a cpsw ale engine
[    5.043691] keystone-netcp 2090000.netcp: initialized cpsw ale
revision 1.3
[    5.053692] davinci_mdio 2090300.mdio: resetting idled controller
[    5.060920] keystone-netcp 2090000.netcp: phy found: id is:
0x2090300.mdio:00
[    5.120210] keystone-netcp 2090000.netcp: Using Packet Accelerator
Firmware version 0x010300e8
```

```
[    5.131976] net eth0: netcp device eth0 opened
[    5.139053] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[    5.144843] 8021q: adding VLAN 0 to HW filter on device eth0
[    5.150479] net eth0: adding rx vlan id: 0
udhcpc (v1.13.2) started
Sending discover...
[    7.056162] libphy: 2090300.mdio:00 - Link is Down
Sending discover...
[    9.056205] libphy: 2090300.mdio:00 - Link is Up - 1000/Full
[    9.061837] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
Sending discover...
Sending select for 192.168.50.101...
Lease of 192.168.50.101 obtained, lease time 604800
adding dns 192.168.50.40
done.
Setting up IP spoofing protection: rp_filter.
hwclock: can't open '/dev/misc/rtc': No such file or directory
Wed Aug 29 18:38:00 UTC 2012
hwclock: can't open '/dev/misc/rtc': No such file or directory
INIT: Entering runlevel: 3
Starting telnet daemon.
Starting syslogd/klogd: done

 _____                    _____           _         _
|  _  |___ ___ ___ ___   |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . |  |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___|  |__|  |_| |___|_| |___|___|_|
              |___|                    |___|

Arago Project http://arago-project.org tci6614-evm ttyS0

Arago 2011.06 tci6614-evm ttyS0

tci6614-evm login:
```

Loading and Running Linux Kernel using tftp with NFS file system

Instead of using initrd file system described in the previous section you may use NFS mounted file system. You will still use tftp server to load kernel, device tree and boot monitor. So, you would need to set the same “serverip” and “tftp_root” environment variables as described in the previous section. In addition to them set the “nfs_root” environment variable, which should refer to root directory of your target file system. The default environment variables assume that the target file system is on the same server as tftp server.

```
>setenv serverip 192.168.1.10
>setenv tftp_root release
>setenv nfs_root export/release
>setenv boot net
>saveenv
>boot
```

If the NFS server IP is different, update the value in args_net env variable. For example do

```
>setenv nfs_serverip 192.128.1.11
```

and replace serverip with nfs_serverip for args_net.

Note that if you don't already have NFS server setup on your Ubuntu PC, please see http://processors.wiki.ti.com/index.php/NFS_Setup

Once booting completes, you can login to the EVM as 'root' with no password

Loading and Running Linux Kernel using UBIFS

Please see the UBI [2] section in the User Guide how to create and flash ubifs image.

Set "boot" environment variable to "ubi" to set the ubi boot mode.

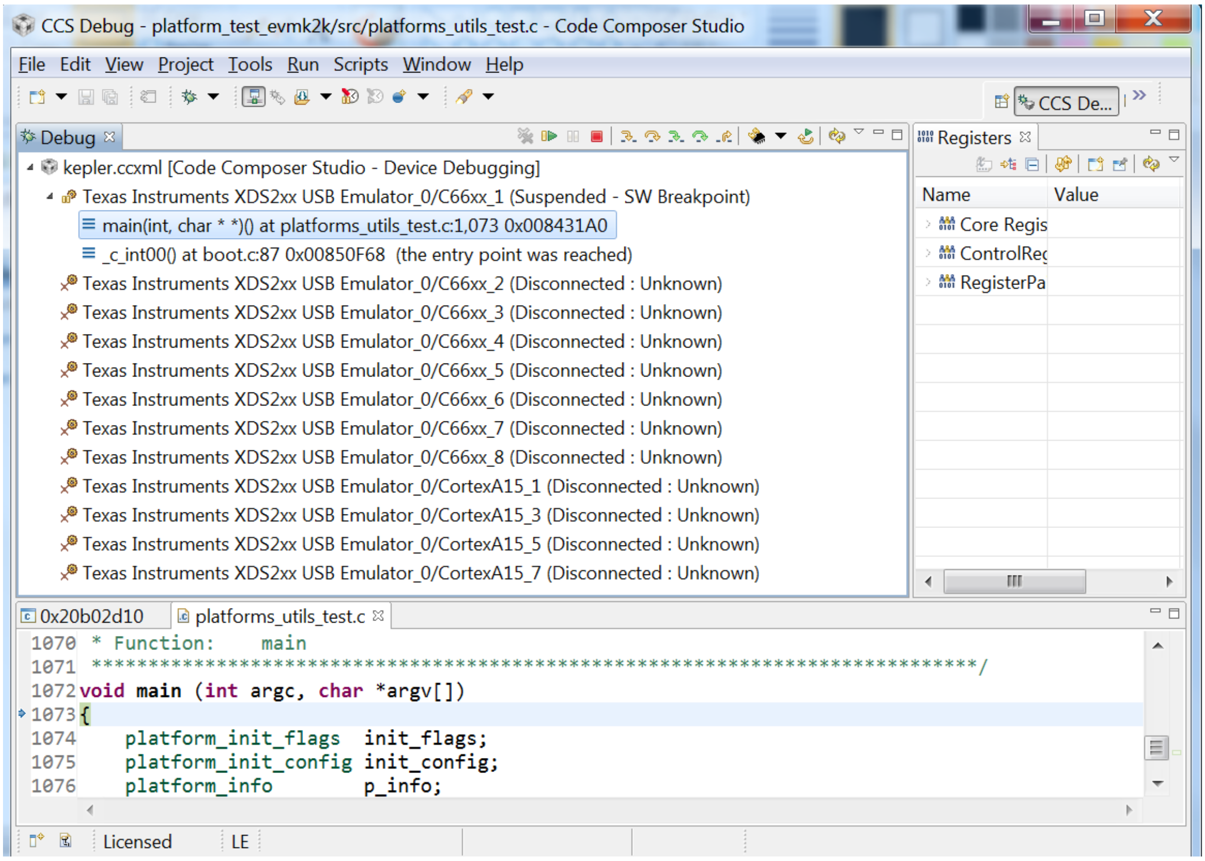
```
>setenv boot ubi
```

DSP

Loading and Running DSP applicaton

Start the Code Composer Studio. Open the "Target Configurations" tab via View->Target Configurations. Create a new target "tci6638k2k.ccxml" with TI XDS2xx USB emulator and TCI6638 device.

Right click on the tci6638k2k.ccxml configuration and launch it. Load any DSP application .out to DSP core 1 (C66x_1), and run the application



```
kepler.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS2xx USB Emulator_0/C66xx_1 (Suspended - SW Breakpoint)
    main(int, char * *)() at platforms_utils_test.c:1,073 0x008431A0
    _c_int00() at boot.c:87 0x00850F68  (the entry point was reached)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_2 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_3 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_4 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_5 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_6 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_7 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/C66xx_8 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_1 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_3 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_5 (Disconnected : Unknown)
  Texas Instruments XDS2xx USB Emulator_0/CortexA15_7 (Disconnected : Unknown)
```

```
 * Function:    main
 ***************************************************************************/
void main (int argc, char *argv[])
{
    platform_init_flags  init_flags;
    platform_init_config init_config;
    platform_info        p_info;
```

How To

Relevant How To's are placed here.

How to Use the Linux Devkit

To install the devkit, just run the script arago-2013.04-armv7a-linux-gnueabi-mcsdk-sdk-i686.sh under mcsdk_linux_*/linux-devkit/.

The header files are under

path/to/devkit/install/directory/sysroots/armv7ahf-vfp-neon-oe-linux-gnueabi/usr/include/

The libraries are under

path/to/devkit/install/directory/sysroots/armv7ahf-vfp-neon-oe-linux-gnueabi/usr/lib/

MCSDK UG Chapter Exploring

MCSDK User Guide: Exploring the MCSDK

Last updated: **09/20/2015**

Overview

The Multicore Software Development Kit (MCSDK) provides foundational software for TI KeyStone II device platforms. It encapsulates a collection of software elements and tools intended to enable customer application development and migration.

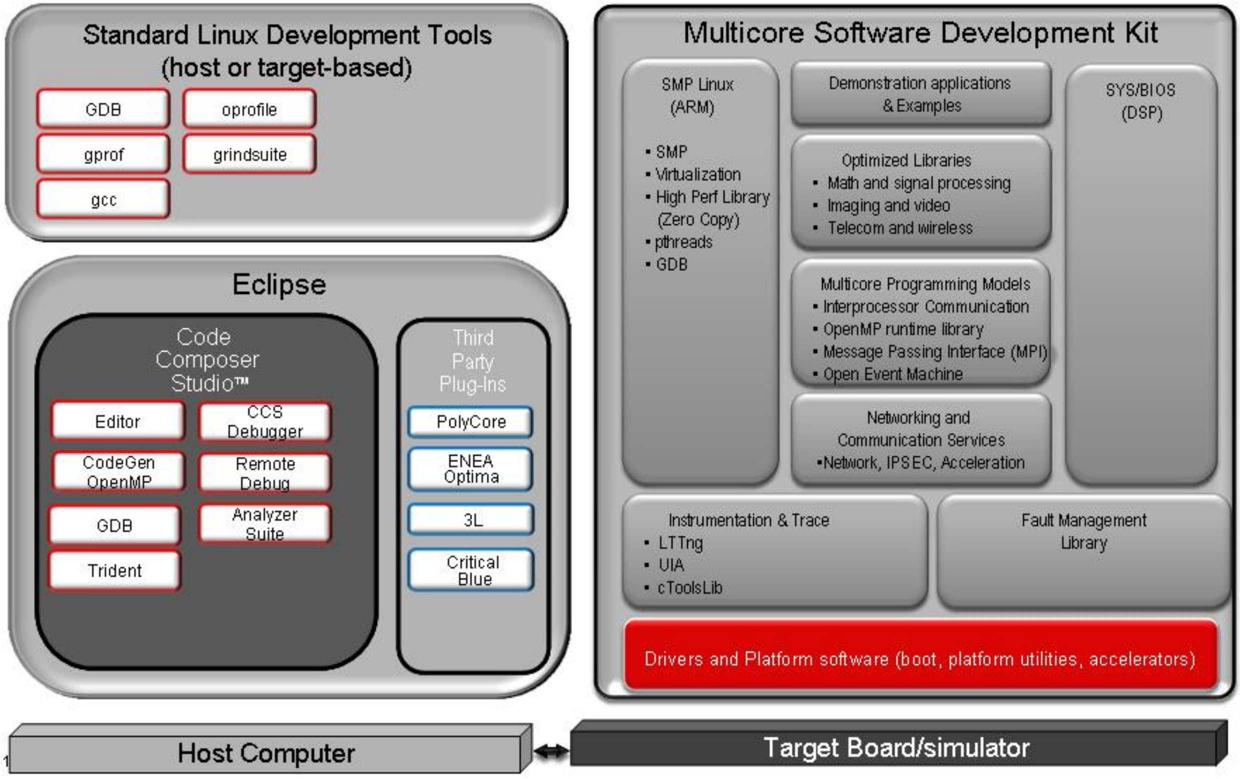
The foundational components include:

- SYS/BIOS real-time embedded operating system on DSP cores
- Linux high-level operating system running on ARM A15 cluster (SMP mode)
- DSP chip support libraries, DSP/ARM drivers, and basic platform utilities
- Interprocessor communication for communication across cores and devices
- SoC resource management
- Optimized application-specific (transport) and application non-specific algorithm libraries
- Trace debug and instrumentation
- Bootloaders and boot utilities, power-on self test

- Demonstrations and examples
- ARM software libraries available in Linux devkit or via Arago/Yocto
- Latest toolchain (ARM Linaro, DSP TI CodeGen)
- Host tools, integrated development environment

This Chapter provides a high level overview of the different pieces so you will gain a sense of what they are and what they do. After reading this chapter you should have a sense of the different pieces that make up the MCSDK. You can then refer to the chapter, Developing with the MCSDK to get the details.

Here is a high-level picture of the software ecosystem that MCSDK is a part of:



Note: Not all items in the genric picture above applies to all parts.

Please see the release notes for the comprehensive list of components and version; the latest release notes is MCSDK Release Notes [1]. For licensing information, please see the software manifest; the latest software manifest is MCSDK Software Manifest [2].

IS NOT

- Support for BIOS5 or older releases
- Support for CCS 4.x or older releases
- Support for platforms not listed here [3]
- DSP image format other than ELF (e.g., COFF)
- Big endian ARM
- Big endian DSP with ARM Little endian configuration

Acronyms

Put a condensed acronym table here. That is one that addresses acronyms that this Chapter uses. This means you should add/remove from the Table below.

The following acronyms are used throughout this chapter.

Acronym	Meaning
AMC	Advanced Mezzanine Card
CCS	Texas Instruments Code Composer Studio
CSL	Texas Instruments Chip Support Library
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol
DSP	Digital Signal Processor
DVT	Texas Instruments Data Analysis and Visualization Technology
EDMA	Enhanced Direct Memory Access
EEPROM	Electrically Erasable Programmable Read-Only Memory
EVM	Evaluation Module, hardware platform containing the Texas Instruments DSP
HUA	High Performance Digital Signal Processor Utility Application
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
IPC	Texas Instruments Inter-Processor Communication Development Kit
JTAG	Joint Test Action Group
MCSA	Texas Instruments Multi-Core System Analyzer
MCSDK	Texas Instruments Multi-Core Software Development Kit
NDK	Texas Instruments Network Development Kit (IP Stack)
NIMU	Network Interface Management Unit
PDK	Texas Instruments Programmers Development Kit
RAM	Random Access Memory
RTSC	Eclipse Real-Time Software Components
SPL	Secondary Program Loader
SRIO	Serial Rapid IO
TCP	Transmission Control Protocol
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UIA	Texas Instruments Unified Instrumentation Architecture
USB	Universal Serial Bus

Note: We use the abbreviation TMS when referring to a specific TI device (processor) and the abbreviation TMD when referring to a specific platform that the processor is on. For example, TMS320C6678 refers to the C6678 DSP processor and TMDSEVM6678L refers to the actual hardware EVM that the processor is on.

ARM Subsystem

Overview

ARM subsystem runs following software components:-

- U-Boot - Boot loader
- Boot Monitor - Monitor and other secure functions
- SMP Linux - ARM A15 port of SMP Linux

This section describes details of these components delivered as part the Linux ecosystem in MCSDK.

Linux

This section provides information on the features, functions, delivery package and compile tools for the Linux Kernel release for KeyStone II. This document describes how to install and work with Texas Instruments' Linux Kernel for the KeyStone II platform. The MCSDK provides a fundamental Linux based software platform for development, deployment and execution on the ARM A15 processor of the KeyStone II architecture. In this context, the document contains instructions to:

- Install the release
- Build the sources contained in the release and run it on the target keystone II EVM
- Kernel and peripheral driver design and/or implementation details

Changes impacting users

It is strongly recommended to upgrade all images to the same release which includes: uboot, boot monitor, kernel and file system.

This section describes the list of changes that impacts the user

K2_RT_LINUX_03.10.10_13.11/K2_LINUX_03.10.10_13.11

Starting with this release, the git repo for linux, boot monitor and u-boot has been moved to git.ti.com (previously on Arago). Also Linux kernel is rebased to v3.10.10. Please see the build location for the URL for these repos.

K2_RT_LINUX_03.08.04_13.09_01/K2_LINUX_03.08.04_13.09_01

In these tags, kernel started using 512M capacity of the NAND available on the EVM. The ubifs partion size in dts file is changed accordingly to use the increased size. The u-boot is upgraded to increase the mtd partition size to reflect the new size. The u-boot tag for this is K2_UBOOT_2013-01_13.09_01. There are dts changes that requires the kernel to be in sync with DTS.

DEV.MCSDK-03.08.04.11

- DTS file name is changed to match with EVM name, k2hk-evm.dts. So the DTB file name has k2hk to indicate the EVM name.

DEV.MCSDK-03.08.04.10

- Linux upstream kernel changed to 3.8.4. Last release was based on 3.6.6. As a side effect, the location of dtb file generated by the Linux kernel build is now at arch/arm/boot/dts folder.
- Changed the name of dtb file populated in the release to uImage-tci6638-evm.dtb.

DEV.MCSDK-03.06.06.08

- Changed the name of u-boot command install_skern() to mon_install()
- Changed the name of dtb file populated in the release binaries from tci6638-evm.dtb to uImage-keystone-evm.dtb

Installation Guide

Prerequisites for MCSDK 3.0 Installation

Before you begin with the installation of this package please make sure you have met the following system requirements:

- CCS v5.3.0 or later: Available on Windows or Linux Host
- Keystone II Simulator: Available on Windows or Linux Host
- Toolchain: **Must** be installed on Linux Host. Ubuntu 12.04 recommended.
- Terminal Emulator: Tera Term on Windows or Minicom on Linux Host can be used.

For CCS and Simulator installation

please refer to corresponding documentation, which comes with CCS and/or Simulator. Please see CCS Getting started guide [18]

Toolchain Installation

Please refer to Linaro toolchain [4]

Linux Software Installation

The Linux kernel related utilities and pre-built binaries are provided in the release under the folder mcsdk_linux_<version>.

Please refer the "Getting Started" section for more details on installation.

Tag and Branch naming convention

U-Boot

U-Boot release tag is named in the following format:-

- K2_UBOOT_<upstream release version><year_> <month_><[iteration]>

For Example, K2_UBOOT_2013-01_13.08 indicates, the u-boot is based on upstream version 2013-01 and the release is done in August 2013. This done to make it intuitive for anyone to identify the baseline upstream u-boot version used in a release from the tag name.

Linux

Linux release tag is named in the following format:-

- K2_LINUX_<rt></rt>-<upstream release version_><year_> <month_><[iteration]>

For example K2_LINUX_03.08.04_13.08_01 indictaes the Linux kernerl is based on upstream kernel version 03.08.04 and the release is done in August 2013 and iteration is 1. Iteration is optional and is used when multiple tags are to be used for the same month for what ever reason.

For RT Preempt patched kernel, the tag name also include the word "RT"

RT Preempt patched Linux kernel is available on a seperate master branch (master-rt). master branch is without RT patches. This will be available under release_<build number>/master-rt and release_<build number>/master branches respectively on Arago git repo.

Boot Monitor

Boot Monitor release tag is named in the following format:-

- K2_BM_<year_> <month_><[iteration]>

For example K2_BM_13.08 is released in August 2013

Build Instructions

There are two possible ways to build the linux kernel:

- The first is to build the standalone kernel by cloning the kernel Yocto git repository on the local host. Please refer the Yocto section below for details.
- The second is to build the kernel together with the filesystem using Yocto project on Arago

Build Prerequisites

- Ubuntu 12.04 LTS distribution and sudo access should be available to the user.
- Install the tool chain as described in the section **Toolchain Installation** above for cross compiling.
- Install and configure git in the Ubuntu machine. Use the following command to install git:

```
 $ apt-get install git-core
```

- To configure git please refer here[5]. If your network is behind a proxy, those settings need to be configured as well.
- Packages needed at build-time can be fetched with a simple command on Ubuntu 12.04:

```
 $ sudo apt-get install build-essential subversion ccache sed wget cvs
coreutils unzip texinfo docbook-utils gawk help2man diffstat file g++
texi2html bison flex htmldoc chrpath libxext-dev xserver-xorg-dev
doxygen bitbake uboot-mkimage libncurses5-dev
```

Note: If you are running a distribution other than Ubuntu 12.04 LTS, please refer to your distribution documentation for instructions on installing these required packages.

Note: From the MCSDK 3.0.4 release onward, the size of the the rootfs has increased beyond 80MB. As a result use of ramfs is not possible. The filesystem needs to be under 80M for use with ramfs

Proxy Setup

If your network is behind a firewall/proxy additional settings are needed for bitbake to be able to download source code repositories for various open source projects. Some of these configuration items are:

- wgetrc: A ".wgetrc" needs to be created under the $HOME directory. A sample wgetrc can be found here[6]. Please update configuration variables http_proxy, https_proxy and ftp_proxy as per your network environment.
- Set proxy environment variables. These may be added to your .bashrc or shell initialization script:

```
export http_proxy="http://&lt;your_proxy&gt;:&lt;port&gt;"
export ftp_proxy="http://&lt;your_proxy&gt;:&lt;port&gt;"
export https_proxy="http://&lt;your_proxy&gt;:&lt;port&gt;"
```

- $HOME/.subversion/servers needs to be updated if the network is behind a proxy. The following lines need to be modified as per settings for your network:

```
http-proxy-exceptions = “exceptions”
http-proxy-host = “proxy-host-for-your-network”
http-proxy-port = 80
```

U-Boot Build instructions

First clone the U-Boot source tree from Arago git repository

```
$ git clone git://git.ti.com/keystone-linux/u-boot.git u-boot-keystone
$ cd u-boot-keystone
$ git reset --hard <Release tag>
```

where release tag can be obtained from the release notes. For example, release tag used is DEV.MCSDK-2013-01.11

1. To build u-boot.bin that can be loaded and run from MSMC SRAM, do the following

```
make <soc>_evm_config
make
```

To do 2 stage SPI NOR boot, following images are to be built.

2. To build u-boot-spl.bin that can be booted from SPI NOR flash, do the following:-

```
make <soc>_evm_config
make spl/u-boot-spl.bin
```

The u-boot-spl.bin will be available at spl/ folder

3. To build secondary boot u-boot.img (in uImage firmware format) that can be flashed and booted through SPI NOR flash do the following:-

```
make <soc>_evm_config
make u-boot.img
```

The u-boot.img will be created at root directory of u-boot source tree.

Alternately both u-boot-spl.bin and u-boot.img can be combined and flashed to SPI NOR flash. To do so, use the following command to build a single gph image for programming on SPI NOR flash

```
make <soc>_evm_config
make u-boot-spi.gph
```

The u-boot-spi.gph image will be created in the root folder.

4. To build a single u-boot-nand.gph image that can be programmed on EMIF16 NAND flash, do the following:-

```
make <soc>_evm_config
make u-boot-nand.gph
```

Note: <soc> is "k2hk", "k2l" or "k2e"; for MCSDK 3.0.x, <soc> is "tci6638"

Boot Monitor Build instructions

To build boot monitor code, first clone the git repository as

```
$ git clone git://git.ti.com/keystone-linux/boot-monitor.git
$ cd boot-monitor
$ git reset --hard <Release tag>
```

where release tag can be obtained from the release notes. For example, DEV.MCSDK-03.00.00.11 is the release tag.

```
$ make clean
$ make
```

skern-<soc>.bin (for MCSDK 3.0.x, skern.bin) will be created in the current working directory.

Note: <SOC> is "k2hk", "k2l" or "k2e"

Linux kernel and the device tree blob build instructions

This section assumes that the Linaro toolchain for ARM is installed and environment variables CROSS_COMPILE, ARCH are set up as per instructions given in section Toolchain Installation Linaro_toolchain [4].
(e.g)

```
export CROSS_COMPILE=arm-linux-gnueabihf-
export ARCH=arm
PATH=<path to installed toolchain>/bin:$PATH
```

The first step in building the kernel is downloading the kernel source code from the git repositories to the local Ubuntu 12.04 host. This is performed as follows:

```
$ git clone git://git.ti.com/keystone-linux/linux.git linux-keystone
$ cd linux-keystone
```

$ git reset --hard <Release tag> where release tag can be obtained from Release notes. For example tag is DEV.MCSDK-03.08.04.11 for the release.

Before building the linux kernel for Simulator, user has to modify the source file linux-keystone/drivers/net/ethernet/ti/keystone_ethss.c

The line u8 mac_addr_nic[6] = {0x5c, 0x26, 0x0a, 0x80, 0x0d, 0x43}; /* NIC addr */ has to be modified

The NIC address should be appropriately modified with the NIC address of the host where CCS and the simulator are installed.

To build the kernel for execute the following commands inside the linux-keystone folder:

```
$ make keystone2_defconfig
$ make uImage
```

The next step is to build the device tree file:

For building DTB for EVM use the following command:-

```
$ make <soc>-evm.dtb
```

Note: <soc> is "k2hk", "k2l" or "k2e"

For building DTB for simulator use the following command:-

$ make keystone-sim.dtb

Note that starting 3.8.4, location of dtb binary is moved to arch/arm/boot/dts.

To Build Linux for Full RT Preempt mode, do the following

$ git reset --hard <Release tag> where release tag can be obtained from Release notes. For example tag for RT kernel is DEV.MCSDK-RT-03.08.04.11 for the release.

$ make keystone2_fullrt_defconfig

$ make uImage

TransportNetLib Software Library and Build instructions

TransportNetLib software package provides a library for ARM user space application to gain direct access to networking h/w. This is for such use cases as fast path packet processing. The TransportNetLib includes HighPerformanceLib(HPLIB) and Network API(NetAPI) modules. Detailed description of these modules and build insructions are available at http://processors.wiki.ti.com/index.php/TransportNetLib_UsersGuide.

Running U-Boot, Boot Monitor and Linux Kernel on EVM

Loading and Running U-Boot on EVM through UART

Before loading the u-boot to EVM you need to create u-boot.uart blob. That is the u-boot.bin binary with preceding 4094 zeros. Assuming you are in the u-boot source top level directory execute the following commands after building u-boot:

```
> dd if=/dev/zero of=4k_zeros bs=4096 count=1
> cat 4k_zeros u-boot.bin > u-boot.uart
```

You should run two terminals on a PC. One connected to the BMC and another to the UART0 of the K2HK SOC. The terminal connected the the SOC UART0 port must support xmodem protocol. For example to use minicom on Linux, follow the steps at [[7]]

At the UART0 terminal, user should initiate transfer of the u-boot.uart file using the xmodem protocol. User should see

```
's: Give your local XMODEM receive command now."
```

Using BMC console set the ARM UART boot mode and reboot the K2HK SOC.

```
BMC> bootmode #4
BMC> reboot
```

This would initiate the transfer. Once transfer is complete, user would see

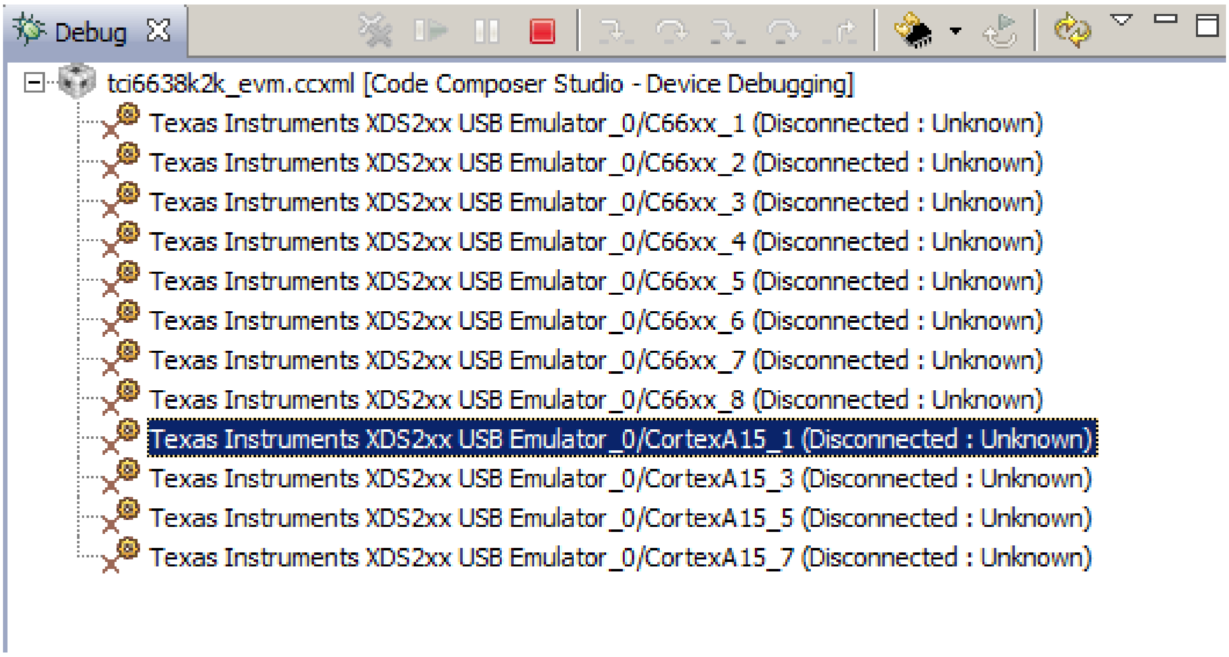
```
Transfer complete
READY: press any key to continue...
```

Once any key is pressed, u-boot starts up and boot up log is seen at the UART0 terminal

Loading and Running U-Boot on EVM through CCS

If you do not have a target configuration for the EVM, you have to create one. Create a new target configuration for the TCI6638 device using the "Texas Instruments XDS2xx USB Emulator" connection. You have to do that only once.

Launch the target configuration.



tci6638k2k_evm.ccxml [Code Composer Studio - Device Debugging]
Texas Instruments XDS2xx USB Emulator_0/C66xx_1 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_2 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_3 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_4 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_5 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_6 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_7 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/C66xx_8 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/CortexA15_1 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/CortexA15_3 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/CortexA15_5 (Disconnected : Unknown)
Texas Instruments XDS2xx USB Emulator_0/CortexA15_7 (Disconnected : Unknown)

Power on the EVM and connect CCS to the CortexA15_1 target.

Connect serial cable between the PC and the EVM. Open Teraterm or Hyper Terminal, create a connection with 115200 baud rate, 8 data bits, no parity, 1 stop bits and no flow control.

Copy the images from the images folder of the installed release directory to the loadlin folder. Copy the latest tci6638-evm.ccxml to the loadlin folder. Also the script may need tweaking to suite the environment on which you are running this script. The example given here is for reference only.

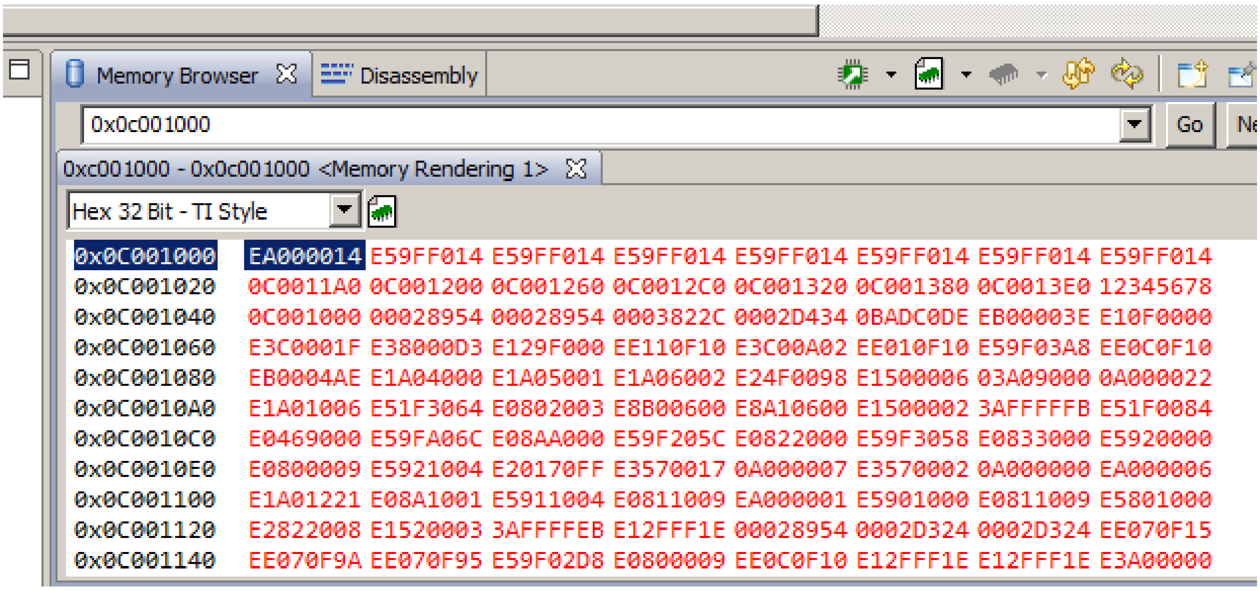
Edit the loadlin-evm-uboot.js java script from the <release folder>/host-tools/loadlin folder for the following

```
PATH_LOADLIN = <path of release folder>/host-tools/loadlin
var pathUboot = PATH_LOADLIN + "/u-boot-keystone-evm.bin";
var pathSKern = PATH_LOADLIN + "/skern-keystone-evm.bin";
```

Save the file. Copy the image files (u-boot-keystone-evm.bin and skern-keystone-evm.bin) from <release folder>/images to <release folder>/host-tools/loadlin. Open the scripting console and type

- loadJSFile "<path of release folder>/host-tools/loadlin/loadlin-evm-uboot.js"

This will load, u-boot image to MSMC RAM at 0xc001000 and boot monitor image to MSMC RAM at address 0xc5f0000. Make sure PC is currently pointing to 0xc001000. Click Resume button on the CCS window to run u-boot.

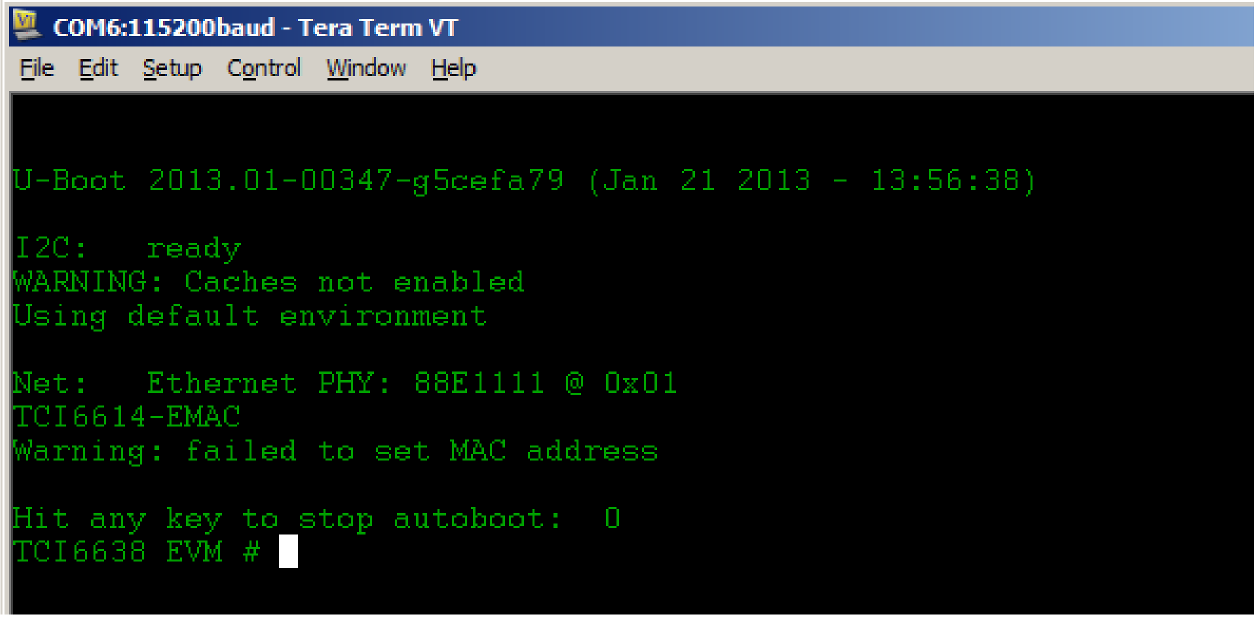


0x0c001000

0xc001000 - 0x0c001000 <Memory Rendering 1>

Hex 32 Bit - TI Style

```
0x0C001000  EA000014 E59FF014 E59FF014 E59FF014 E59FF014 E59FF014 E59FF014 E59FF014
0x0C001020  0C0011A0 0C001200 0C001260 0C0012C0 0C001320 0C001380 0C0013E0 12345678
0x0C001040  0C001000 00028954 00028954 0003822C 0002D434 0BADC0DE EB00003E E10F0000
0x0C001060  E3C0001F E38000D3 E129F000 EE110F10 E3C00A02 EE010F10 E59F03A8 EE0C0F10
0x0C001080  EB0004AE E1A04000 E1A05001 E1A06002 E24F0098 E1500006 03A09000 0A000022
0x0C0010A0  E1A01006 E51F3064 E0802003 E8B00600 E8A10600 E1500002 3AFFFFFB E51F0084
0x0C0010C0  E0469000 E59FA06C E08AA000 E59F205C E0822000 E59F3058 E0833000 E5920000
0x0C0010E0  E0800009 E5921004 E20170FF E3570017 0A000007 E3570002 0A000000 EA000006
0x0C001100  E1A01221 E08A1001 E5911004 E0811009 EA000001 E5901000 E0811009 E5801000
0x0C001120  E2822008 E1520003 3AFFFFEB E12FFF1E 00028954 0002D324 0002D324 EE070F15
0x0C001140  EE070F9A EE070F95 E59F02D8 E0800009 EE0C0F10 E12FFF1E E12FFF1E E3A00000
```



```
U-Boot 2013.01-00347-g5cefa79 (Jan 21 2013 - 13:56:38)

I2C:   ready
WARNING: Caches not enabled
Using default environment

Net:   Ethernet PHY: 88E1111 @ 0x01
TCI6614-EMAC
Warning: failed to set MAC address

Hit any key to stop autoboot:  0
TCI6638 EVM #
```

Loading and running U-Boot on EVM through SPI Boot

Two Stage SPI Flash Boot

The two stage SPI flash u-boot image consists of a first stage u-boot SPL binary with a header and a footer, and a second stage full u-boot binary. These two binaries can be programmed to SPI flash in two ways.

- Using a combined single GPH image (u-boot-spi-<soc>-evm.gph from release) to program the SPI NOR flash.

Where <soc> is k2hk/k2e/k2l. If you build u-boot from source code, u-boot-spi.gph will be created in the root source directory and is to be used as u-boot-spi-<soc>.gph. This is the quick and easiest way to upgrade u-boot image if you have u-boot running already on the evm. This implemented using a u-boot env script which is added to very recent version of the u-boot. To see if current u-boot on EVM support this, do

```
>printenv
```

and check if get_uboot_net and burn_uboot env variables are present. Also name_uboot is set to u-boot-spi-<soc>-evm.gph. If so do following to upgrade u-boot assuming gph image is copied to root directory of tftp server

```
env default -f -a
setenv serverip <ip address of tftp server>
setenv tftp_root <tftp root directory>
run get_uboot_net
run burn_uboot
reset
```

The EVM now boots with latest version of u-boot image.

'NOTE:' User might want to check the value of mem_reserve env variable to ensure only reserved memory is used by applications as described in Reserve DDR memory

- Program the two stage binaries separately, follow the steps in the below section

Burning First Stage u-boot-spl.bin to SPI Flash

Follow the same procedure described in section Burning Single Stage U-Boot to SPI Flash, but use the u-boot-spl.bin image instead. The SPL U-Boot reside in SPI flash at offset 0.

Burning Second Stage u-boot.img to SPI Flash

First step is to load u-boot.img to MSMC SRAM address 0xc300000 through CCS or other means. Then do the following steps:-

- sf probe
- sf erase 0x10000 <size of image in hex rounded to sector boundary of 0x10000>

  for example if the size of image is hex 0x4500b, round it to 0x50000

- sf write 0xc300000 0x10000 <size of image in hex>

Single Stage SPI Flash Boot

In the single stage SPI flash boot, the image to be burnt into the SPI flash contains the u-boot.bin with a header and a footer added.

Burning Single Stage U-Boot to SPI Flash

This section assumes that u-boot is booted up and running on EVM, e.g. using CCS (see section above)

Load u-boot.bin to memory to be saved to SPI flash through CCS

```
* Have a copy of u-boot.bin ready in a directory on host PC running CCS
* Start and connect CCS to ARM core 0
* CCS > Run > "Suspend"
* CCS > Tools > Load Memory > browse to the copy of u-boot.bin and load it to address 0x0c300000
* CCS > Run > "Resume"
where 0x0c300000 is an example address to temporarily keep the u-boot.bin image.
```

Now a copy of u-boot.bin is loaded to 0x0c300000.

At the u-boot prompt:

- Format the u-boot.bin image loaded in memory to SPI boot format

```
> fmtimg spi 0x0c001000 0x0c300000 <hex_size>
where hex_size is the size of the u-boot.bin in hex bytes,
and 0x0c001000 is the address that u-boot.bin will be loaded by RBL and starts execution.
Note down the formatted image hex size printed out at the end of the command, call this fmt_img_size.
```

- Alternatively issue following commands to perform the formatting

```
> setenv fileaddr 0x0c300000
> setenv filesize <hex_size>
> fmtimg spi 0x0c001000
where command fmtimg takes the hex size of u-boot.bin from env variable "filesize",
and the address where u-boot.bin is situated is taken from env variable "fileaddr".
This is useful if u-boot.bin is loaded to memory by using tftp. In that case, filesize and fileaddr are set in the process.
Note down the formatted image hex size printed out at the end of the command, call this fmt_img_size.
```

- Prepare SPI flash for saving formatted u-boot.bin later

```
> sf probe
> sf erase 0 <hex_len>
erases hex_len bytes in SPI flash starting from offset 0,
where hex_len must be at least the formatted image size shown in the fmtimg command and on flash sector boundary,
sector size of SPI flash on EVM is 0x10000
e.g. if formatted image size is 0x2FBA0, then "sf erase 0 0x30000"
```

- Save formatted u-boot.bin image to SPI flash

```
> sf write 0x0c300000 0 <fmt_img_size>
where fmt_img_size is the hex size of the formatted image printed out in the fmtimg command above.
```

Booting U-Boot from SPI Flash on EVM

```
* Power off EVM
* Set Boot Setting DIP Switch (SW1) on EVM to 0010 (ARM SPI boot mode)
     - i.e SW1: 1(OFF) 2(OFF) 3(ON) 4(OFF)
* If any, disconnect CCS from EVM
* Power up EVM
```

Loading and running U-Boot on EVM through NAND boot (Keystone2 Rev2.0 or Later EVM ONLY)

Burning U-Boot to NAND Flash

This section assumes that u-boot is booted up and running on EVM, e.g. using CCS or SPI boot. Also the EVM is connected to a network where a tftp server is available.

- compile the nand u-boot image on host (refer to U-Boot Build instructions section)
- place u-boot-nand.gph in tftp server
- under u-boot prompt, do

```
> setenv serverip <tftp-server-ip-address>
> setenv bootfile '/tftp/server/path/to/u-boot-nand.gph'
> dhcp ${fileaddr}
> nand ecclayout set 1
> nand erase.part bootloader
> # For K2HK EVM
> nand write ${fileaddr} bootloader ${filesize}
> # For K2E or K2L EVM
> nand write ${fileaddr} <offset> ${filesize}
> nand ecclayout set 0
```

'NOTE:' There is an errata for K2L and K2E PG 1.0 that NAND boot does not distinguish between a block that has been pre-marked as bad vs one that is declared bad due to error correction failure. When a bad block is found, the ROM bootloader moves to the next block and re-initializes the boot data processor. When the block is pre-marked as bad the boot data processor should not be reset.

The workaround to this problem is to burn a multi-block image in the NAND with contiguous good blocks. Users can check the bad block list using "nand bad" command and skip any bad blocks.

The NAND device on K2L EVM has a block size of 256KB, and u-boot-nand.gph is over 300KB which will uses 2 blocks, starting from block 0 (offset address 0). Similarly the nand device on K2E EVM has a block size of 128KB, and u-boot-nand.gph will uses 3 blocks. If any of the 2 or 3 blocks are marked bad, we should skip the bad block and use the next 2 or 3 good blocks.

E.g. block 0 is good but block 1 is bad, and block 2 - 4 are good, the <offset> should be set to 0x80000 (starting offset address of block 2) for K2L EVM, and 0x40000 for K2E EVM.

'NOTE:' User might want to check the value of mem_reserve env variable to ensure only reserved memory is used by applications as described in Reserve DDR memory

Booting U-Boot from NAND Flash on EVM

To boot u-boot from NAND flash, do one of the following:

- By setting boot pin:

```
* Power off EVM
* Set Boot Setting DIP Switch (SW1) on EVM to 0000 (ARM NAND boot mode)
     - i.e SW1: 1(OFF) 2(OFF) 3(OFF) 4(OFF)
* Power up EVM
```

- From BMC terminal on Rev2.0 EVM

```
BMC> bootmode #0
BMC> reboot
```

Loading and Running Linux Kernel on EVM

Loading and Running Linux Kernel on EVM using CCS

First step is to load **u-boot** (eg. u-boot-keystone-evm.bin) and **boot monitor** (eg. skern-keystone-evm.bin) images onto MSMC RAM and run u-boot to get the u-boot prompt. See the instructions in section Loading and Running U-boot on EVM using CCS.

Now follow the steps below to load and run Linux through CCS.

Copy the images from the images folder of the release directory to the loadlin folder.

Edit the loadlin-evm-kern.js file and make sure the following variables points to valid paths for the images.

```
PATH_LOADLIN = <path of lsp-release-folder>/host-tools/loadlin
var pathKernel            = PATH_LOADLIN + "/uImage-keystone-evm.bin";
var pathDtb            = PATH_LOADLIN + "/uImage-k2hk-evm.dtb";
var pathInitrd           = PATH_LOADLIN + "/<name of min root file system, eg. tisdk-rootfs.cpio.gz>".
If the root fs image size is greater than 9M, make sure the bootargs in u-boot is set correctly to reflect the size.
Currently it defaults to 9M.
```

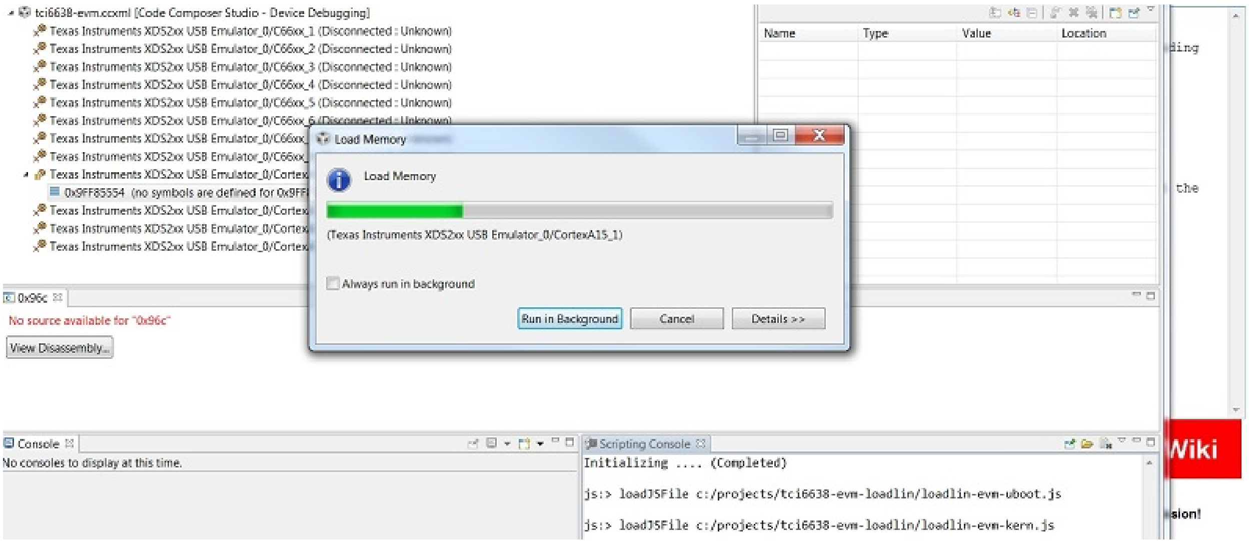
> **Note** For RT Preempt images, use image file names uImage-rt-keystone-evm.bin and uImage-rt-k2hk-evm.dtb for kernel and dtb files.

Save the file and exit.

- Click onto the suspend button or Alt+F8

Open scripting console and run

- loadJSFile "<path of loadlin from release folder>/loadlin-evm-kern.js"



```
tci6638-evm.ccxml [Code Composer Studio - Device Debugging]
    Texas Instruments XDS2xx USB Emulator_0/C66xx_1 (Disconnected : Unknown)
    Texas Instruments XDS2xx USB Emulator_0/C66xx_2 (Disconnected : Unknown)
    Texas Instruments XDS2xx USB Emulator_0/C66xx_3 (Disconnected : Unknown)
    Texas Instruments XDS2xx USB Emulator_0/C66xx_4 (Disconnected : Unknown)
    Texas Instruments XDS2xx USB Emulator_0/C66xx_5 (Disconnected : Unknown)
    Texas Instruments XDS2xx USB Emulator_0/C66xx_6 (Disconnected : Unknown)
    Texas Instruments XDS2xx USB Emulator_0/C66xx_
    Texas Instruments XDS2xx USB Emulator_0/C66xx_
    Texas Instruments XDS2xx USB Emulator_0/Cortex
        0x9FF85554  (no symbols are defined for 0x9FF
    Texas Instruments XDS2xx USB Emulator_0/Cortex
    Texas Instruments XDS2xx USB Emulator_0/Cortex
    Texas Instruments XDS2xx USB Emulator_0/Cortex
```

No source available for "0x96c"

Load Memory

(Texas Instruments XDS2xx USB Emulator_0/CortexA15_1)

□ Always run in background

No consoles to display at this time.

```
Initializing .... (Completed)

js:> loadJSFile c:/projects/tci6638-evm-loadlin/loadlin-evm-uboot.js

js:> loadJSFile c:/projects/tci6638-evm-loadlin/loadlin-evm-kern.js
```

- The script loads Linux kernel image, DTB and file system image to DDR. Once script execution is complete, click onto the Resume button on CCS.
- At u-boot prompt, update the env variable bootargs, for example:

```
> setenv bootargs 'console=ttyS0,115200n8 rootwait=1 earlyprintk rdinit=/sbin/init rw root=/dev/ram0 initrd=0x802000000,9M'

> saveenv
```

If the size of the file system image is greater than 9M, update 9M to the correct size.

Next, type the following command to install boot monitor code.

```
> mon_install 0x0c5f0000
```

Now boot Linux using the u-boot command below.

```
> bootm 0x88000000 - 0x87000000
```

A sample SMP Linux boot log is shown below for reference

```
## Started boot kernel successfully
TCI6638 EVM # bootm 0x88000000 - 0x87000000
## Booting kernel from Legacy Image at 88000000 ...
   Image Name:   Linux-3.6.6-rt17-01071-g898535d-
   Created:      2013-01-28  16:17:47 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3215872 Bytes = 3.1 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
   Booting using the fdt blob at 0x87000000
   Loading Kernel Image ... OK
OK
```

```
   Loading Device Tree to 87ff8000, end 87fff2ea ... OK
```

```
Starting kernel ...
```

```
>>>> skern_poweron_cpu >>>>
Message2 from Secure Mode
>>>> skern_poweron_cpu >>>>
Message2 from Secure Mode
>>>> skern_poweron_cpu >>>>
Message2 from Secure Mode
[    0.000000] Booting Linux on physical CPU 0
[    0.000000] Linux version 3.6.6-rt17-01071-g898535d-dirty (a0868495@ares-ubun
tu.am.dhcp.ti.com) (gcc version 4.7.2 20120701 (prerelease) (crosstool-NG linaro
-1.13.1-2012.07-20120720 - Linaro GCC 2012.07) ) #1 SMP PREEMPT Mon Jan 28 11:17
:38 EST 2013
[    0.000000] CPU: ARMv7 Processor [412fc0f4] revision 4 (ARMv7), cr=10c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] Machine: KeyStone2, model: Texas Instruments Keystone 2 SoC
[    0.000000] cma: CMA: reserved 16 MiB at 86c00000
[    0.000000] Memory policy: ECC disabled, Data cache writealloc
[    0.000000] On node 0 totalpages: 32768
[    0.000000] free_area_init_node: node 0, pgdat c05ff440, node_mem_map c062b00
0
[    0.000000]   Normal zone: 256 pages used for memmap
[    0.000000]   Normal zone: 0 pages reserved
[    0.000000]   Normal zone: 32512 pages, LIFO batch:7
[    0.000000] PERCPU: Embedded 8 pages/cpu @c0735000 s11776 r8192 d12800 u32768
[    0.000000] pcpu-alloc: s11776 r8192 d12800 u32768 alloc=8*4096
[    0.000000] pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pag
es: 32512
[    0.000000] Kernel command line: console=ttyS0,115200n8 debug earlyprintk  rd
init=/bin/ash rw root=/dev/ram0 initrd=0x85000000,9M
[    0.000000] PID hash table entries: 512 (order: -1, 2048 bytes)
[    0.000000] Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
[    0.000000] Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
[    0.000000] Memory: 128MB = 128MB total
[    0.000000] Memory: 97800k/97800k available, 33272k reserved, 0K highmem
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
[    0.000000]     vmalloc : 0xc8800000 - 0xff000000   ( 872 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xc8000000   ( 128 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc0596c08   (5692 kB)
[    0.000000]       .init : 0xc0597000 - 0xc05cbe00   ( 212 kB)
[    0.000000]       .data : 0xc05cc000 - 0xc0601ea8   ( 216 kB)
[    0.000000]       .bss : 0xc0601ecc - 0xc062a1cc   ( 161 kB)
[    0.000000] SLUB: Genslabs=11, HWalign=64, Order=0-3, MinObjects=0, CPUs=4, N
odes=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] main_pll_clk rate is 983040000, postdiv = 2, pllm = 15,plld = 0
[    0.000000] tci6614-timer: no matching node
[    0.000000] arch_timer: found timer irqs 29 30
[    0.000000] Architected local timer running at 166.66MHz.
[    0.000000] Switching to timer-based delay loop
[    0.000000] sched_clock: 32 bits at 166MHz, resolution 6ns, wraps every 25769
ms
[    0.000000] Console: colour dummy device 80x30
[    0.000092] Calibrating delay loop (skipped), value calculated using timer fr
equency.. 333.33 BogoMIPS (lpj=1666666)
[    0.000113] pid_max: default: 4096 minimum: 301
[    0.000364] Mount-cache hash table entries: 512
[    0.008205] CPU: Testing write buffer coherency: ok
[    0.008399] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.008435] hw perfevents: enabled with ARMv7 Cortex-A15 PMU driver, 7 counte
rs available
[    0.008503] Setting up static identity map for 0x804074b0 - 0x804074e4
[    0.104891] CPU1: Booted secondary processor
[    0.104932] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.154985] CPU2: Booted secondary processor
[    0.155031] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.205092] CPU3: Booted secondary processor
[    0.205141] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.205335] Brought up 4 CPUs
[    0.205375] SMP: Total of 4 processors activated (1333.33 BogoMIPS).
[    0.232209] NET: Registered protocol family 16
[    0.247428] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.256104] hw-breakpoint: found 5 (+1 reserved) breakpoint and 4 watchpoint
registers.
[    0.256114] hw-breakpoint: halting debug mode enabled. Assuming maximum watch
point size of 4 bytes.
[    0.275902] bio: create slab <bio-0> at 0
[    0.277188] SCSI subsystem initialized
[    0.278040] usbcore: registered new interface driver usbfs
[    0.278235] usbcore: registered new interface driver hub
[    0.278478] usbcore: registered new device driver usb
[    0.282607] Switching to clocksource arch_sys_counter
[    0.313194] NET: Registered protocol family 2
[    0.313891] TCP established hash table entries: 4096 (order: 3, 32768 bytes)
[    0.314031] TCP bind hash table entries: 4096 (order: 3, 32768 bytes)
[    0.314165] TCP: Hash tables configured (established 4096 bind 4096)
[    0.314195] TCP: reno registered
[    0.314212] UDP hash table entries: 128 (order: 0, 4096 bytes)
[    0.314237] UDP-Lite hash table entries: 128 (order: 0, 4096 bytes)
[    0.314570] NET: Registered protocol family 1
[    0.314914] RPC: Registered named UNIX socket transport module.
[    0.314927] RPC: Registered udp transport module.
[    0.314938] RPC: Registered tcp transport module.
[    0.314949] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.315232] Unpacking initramfs...
[    0.357371] Initramfs unpacking failed: junk in compressed archive
[    0.366886] Freeing initrd memory: 9216K
[    0.512036] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[    0.512661] NTFS driver 2.1.30 [Flags: R/O].
[    0.513335] jffs2: version 2.2. (NAND) Â© 2001-2006 Red Hat, Inc.
[    0.516214] NET: Registered protocol family 38
[    0.516738] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 2
54)
[    0.516754] io scheduler noop registered
[    0.516766] io scheduler deadline registered
[    0.517099] io scheduler cfq registered (default)
[    0.629016] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    0.631409] 2530c00.serial: ttyS0 at MMIO 0x2530c00 (irq = 309) is a 16550A
[    1.272067] console [ttyS0] enabled
[    1.282480] loop: module loaded
[    1.285920] at24 0-0050: 131072 byte 24c1024 EEPROM, writable, 1 bytes/write
[    1.294796] Generic platform RAM MTD, (c) 2004 Simtec Electronics
[    1.317849] ONFI param page 0 valid
[    1.321286] ONFI flash detected
[    1.324406] NAND device: Manufacturer ID: 0x2c, Chip ID: 0xa1 (Micron MT29F1G
08ABBDAHC), page size: 2048, OOB size: 64
[    1.335350] Bad block table found at page 65472, version 0x01
[    1.341652] Bad block table found at page 65408, version 0x01
[    1.347821] 3 ofpart partitions found on MTD device 30000000.nand
[    1.353839] Creating 3 MTD partitions on "30000000.nand":
[    1.359157] 0x000000000000-0x000000100000 : "u-boot"
[    1.365625] 0x000000100000-0x000000180000 : "params"
[    1.371894] 0x000000180000-0x000008000000 : "ubifs"
[    1.378210] davinci_nand 30000000.nand: controller rev. 2.5
[    1.385151] spi_davinci 21000400.spi: master is unqueued, this is deprecated
[    1.396627] m25p80 spi32766.0: unrecognized JEDEC id 20bb18
[    1.402144] spi_davinci 21000400.spi: Controller at 0xc8886400
[    1.411193] Initializing USB Mass Storage driver...
[    1.416251] usbcore: registered new interface driver usb-storage
[    1.422163] USB Mass Storage support registered.
[    1.427169] mousedev: PS/2 mouse device common for all mice
[    1.433274] i2c /dev entries driver
[    1.438539] usbcore: registered new interface driver usbhid
[    1.444048] usbhid: USB HID core driver
[    1.448904] oprofile: using timer interrupt.
[    1.453217] Netfilter messages via NETLINK v0.30.
[    1.457862] nf_conntrack version 0.5.0 (1928 buckets, 7712 max)
[    1.464255] ctnetlink v0.93: registering with nfnetlink.
[    1.470064] IPv4 over IPv4 tunneling driver
[    1.475125] gre: GRE over IPv4 demultiplexor driver
[    1.479929] ip_gre: GRE over IPv4 tunneling driver
[    1.485851] ip_tables: (C) 2000-2006 Netfilter Core Team
[    1.491305] ipt_CLUSTERIP: ClusterIP Version 0.8 loaded successfully
[    1.497642] arp_tables: (C) 2002 David S. Miller
[    1.502261] TCP: cubic registered
[    1.505545] Initializing XFRM netlink socket
[    1.510969] NET: Registered protocol family 10
[    1.516455] NET: Registered protocol family 17
[    1.520858] NET: Registered protocol family 15
[    1.525257] 8021q: 802.1Q VLAN Support v1.8
[    1.529761] sctp: Hash tables configured (established 4096 bind 4096)
[    1.536950] NET: Registered protocol family 40
[    1.541512] rpmsg_proto_init: registered rpmsg network protocol
[    1.547471] VFP support v0.3: not present
[    1.551440] Registering SWP/SWPB emulation handler
[    1.564498] Freeing init memory: 208K
/bin/ash: can't access tty; job control turned off
/ #
```

Design/Implementation Notes

U-Boot

The MCSDK U-boot is based on the Universal Boot Loader (U-boot) public project. The documentation is available on http://www.denx.de/wiki/U-Boot/

The release is based on upstream v2013.01 U-boot release. This section gives information specific to the TCI6638EVM board and drivers only.

Board Support

The Keystone EVM board file islocated at board/ti/ks2_evm (for MCSDK 3.0.x, it is located at board/ti/tci6638_evm directory). It has PLL, DDR3 configurations and initialization functions.

SoC Support

The Keystone chip specific code is located at arch/arm/cpu/armv7/keystone directory. It has the following files:

- aemif.c – Asynchronous EMIF configuration.
- init.c – chip configuration
- clock.c – clock related functions
- cmd_clock.c, cmd_mon.c, cmd_fmtimg.c – implementation of “psc”, “getclk”, “pllset”, “fmtimg” and “mon_install” shell commands.
- psc.c – PSC driver.
- cppi_dma.c – simple CPPI_DMA driver.

Drivers

The Keystone EVM uses the following drivers:

- SPI - drivers/spi/davinci_spi.c.
- I2C - drivers/i2c/keystone_i2c.c.
- UART - serial.c, ns16550.c and serial_ns16550.c at drivers/serial directory.
- NAND - drivers/mtd/nand/davinci_nand.c
- ETH - drivers/net/keystone_net.c
- USB - generic xhci support from patches patchwork.ozlabs.org/patch/193476/ and patchwork.ozlabs.org/patch/193477/. Platform specific xhci support in drivers/usb/host/xhci-keystone.c

Ethernet Driver

The K2HK EVM has 4 Ethernet ports (port 0 - 3) with K2HK_EMAC, K2HK_EMAC1, K2HK_EMAC2, K2HK_EMAC3 interface names. The K2L EVM has 4 Ethernet ports (port 0 - 3) with K2L_EMAC0, K2L_EMAC1, K2L_EMAC2, K2L_EMAC3 interface names. The "ethact" environment variable selects the currently active Ethernet interface. For example:

```
set ethact K2HK_EMAC1
```

selects the port 1 to be used for following network commands.

You may change the active interface on runtime.

The **ethact** variable is set on start of u-boot to the name of the first registered interface. The TCI6638_EMAC is registered first and therefore ethact is always set to that name. You may want to use another environment variable to set desirable interface. You can do that extending init_${boot} variables.

MAC address of the port 0 is taken from SOC e-fuse. You may want to overwrite it setting the “ethaddr” environment variable. The SoC doesn’t have an e-fuse for second port MAC address. You have to set the “eth1addr” explicitly for that port.

NOTE: U-boot does not allow to change the "ethaddr" variable if it is already set. If you want to change it you need to use the "env default -a" command, which resets all environment variabels to default values and deletes the "ethaddr" variable as well.

The board_k2x.c file has the eth_priv_cfg array, which has the default configuration of the both interfaces.

```
eth_priv_t eth_priv_cfg[] = {
        {
                .int_name       = "K2HK_EMAC",
                .rx_flow        = CPSW_PORT_RX_FLOW(0),
                .phy_addr       = 0,
                .slave_port     = 1,
                .sgmii_link_type = SGMII_LINK_MAC_PHY,
        },
        {
                .int_name       = "K2HK_EMAC1",
                .rx_flow        = CPSW_PORT_RX_FLOW(1),
                .phy_addr       = 1,
                .slave_port     = 2,
                .sgmii_link_type = SGMII_LINK_MAC_PHY,
        },
        {
                .int_name        = "K2HK_EMAC2",
                .rx_flow         = CPSW_PORT_RX_FLOW(2),
                .phy_addr        = 2,
                .slave_port      = 3,
                .sgmii_link_type = SGMII_LINK_MAC_MAC_FORCED,
        },
        {
                .int_name        = "K2HK_EMAC3",
                .rx_flow         = CPSW_PORT_RX_FLOW(3),
                .phy_addr        = 3,
                .slave_port      = 4,
                .sgmii_link_type = SGMII_LINK_MAC_MAC_FORCED,
        },
};
```

where

- "phy_addr" is the phy address on mdio bus.
- "slave_port" is the port number starting from "1"
- "sgmii_link_type" - type of SGMII link.

The emac_def.h defines possible sgmii link types

```
#define SGMII_LINK_MAC_MAC_AUTONEG      0
#define SGMII_LINK_MAC_PHY            1
#define SGMII_LINK_MAC_MAC_FORCED      2
#define SGMII_LINK_MAC_FIBER            3
#define SGMII_LINK_MAC_PHY_FORCED      4
```

By default Ethernet driver assumes that phys are not connected to MDIO bus. Using the “has_mdio” environment variable you may override this default:

```
“setenv has_mdio 1”—tells the driver that PHYs are connected to MDIO.
```

If has_mdio=0 or there is no such environment variable the sgmii_link_type for each port is set to SGMII_LINK_MAC_PHY_FORCED. Be aware that the sgmii_link_type for each interface may be overridden by setting the sgmiiN_link_type environment variable. For example:

```
“setenv sgmii1_link_type 2”-  sets the SGMII_LINK_MAC_MAC_FORCED link type for port 1.
```

Driver doesn't perform sanity check of the settings. It is your responsibility to set correct values.

The K2 SOC has 4 SGMII port and you can add configurations for port2 and port3 to the eth_priv_cfg[]. Though driver supports four ports, as K2K EVM has only 2 ports the port2 and port3 functionality wasn’t tested.

Network driver supports Multicast tftp. To enable this feature, add following in the respective config.h file (example include/configs/k2hk_evm.h for K2HK)

```
#define CONFIG_MCAST_TFTP
```

10G Ethernet Driver

K2HK and K2E supports 10G ethernet connections through a 3-port (1 host and 2 slaves) 10G ethernet switch. On K2HK and K2E EVMs, the physical 10G ethernet connection is brought out to a Rear Transition Module-Break Out Card (RTM-BOC). This implementation **supports only the Rev.C RTM-BOC with Dual Retimers**.

The U-boot implementation of the 10G ethernet driver can be considered as an extension of the 1G ethernet driver framework. It adds two interfaces to the data structure eth_priv_cfg[] in the corresponding board_k2x.c files:

```
eth_priv_t eth_priv_cfg[] = {
      ...
      {
            .int_name        = "K2HK_XMAC0",
            .rx_flow         = 0,
            .slave_port      = 1,
            .sgmii_link_type = XGMII_LINK_MAC_MAC_FORCED,
      },
      {
            .int_name        = "K2HK_XMAC1",
            .rx_flow         = 8,
            .slave_port      = 2,
            .sgmii_link_type = XGMII_LINK_MAC_MAC_FORCED,
      },
};
```

with a new interface sgmii_link_type of XGMII_LINK_MAC_MAC_FORCED. The interface names are K2HK_XMAC0 and K2HK_XMAC1 on K2HK; and K2E_XMAC0 and K2E_XMAC1 on K2E.

From the user perspective, there is no difference in configuring and using the 10G interfaces except that there is no has_mdio or sgmii_link_type setenv settings. See Ethernet Driver for more configuration details.

When U-boot boots up, it will only show the 10G interface names if the proper RTM-BOC is connected to the EVM. If so, they are the 4th and 5th (0-based) interfaces on K2HK; and 8th and 9th on K2E. Hence to properly configure the corresponding MAC addresses, eth4addr and eth5addr on K2HK should be used, while they are eth8addr and eth9addr on K2E .

I2C Driver

The keystone_i2c.c driver supports multiple I2C buses. Thus K2 SoC has three buses 0, 1 and 2. Only one bus can be uses at the same time. Use the

```
int i2c_set_bus_num(unsigned int bus)
```

function to set desired bus number. You may get the current bus number by calling the

```
unsigned int i2c_get_bus_num(void)
```

function.

In order to set/get current bus number from u-boot shell use “i2c dev” command.

```
TCI6638 EVM # i2c dev
Current bus is 0
TCI6638 EVM # i2c dev 1
Setting bus to 1
TCI6638 EVM #
```

NAND Driver

The NAND driver used is davinci_nand.c. For K2L, NAND part is 2G vs other EVMs that has 512M part. The EVM by default create 3 partitions.

1. bootloader 2. params (env) 3. ubifs. This has boot volume to hold the images and rootfs to volume to hold the file system

UBI boot time is dependent on the ubifs partition size. So on K2L since the size is much bigger, it takes more time because the mount is done on a bigger nand parition. To reduce the boot time customer may resize the partition to a smaller to hold the file system. Additional partitions can be defined to use as data storage or other applications. Same change needs to be reflected in the DTS as well as mtdparts env variable in u-boot.

SPL

SPL stands for Secondary Program Loader. This is a framework added to unpstream U-Boot in version 2012-10. SPL is supported in U-Boot for keystone II devices. This feature is configured using the config option CONFIG_SPL. It allows creation of a small first stage boot loader (SPL) that can be loaded by ROM Boot loader (RBL) which would then load and run the second stage boot loader (full version of U-Boot) from NOR or NAND. The required config options are added to tci6638_evm.h. User may refer the README text file in the U-Boot root source directory for more details.

For keystone II, RBL loads SPL image from offset 0 of the SPI NOR flash in spi boot mode. The first 64K of the SPI NOR flash is flashed with SPL followed by u-boot.img. The initial 64K is padded with zeros.

Thump mode

U-Boot build uses ARM thump mode. This is configured using CONFIG_SYS_THUMB_BUILD option.

Power-On Self-Test (POST)

POST will perform the following functional tests:

- EEPROM read test
- NAND read test
- External memory read/write test

By default the POST is enabled. Set "no_post" environment variable to disable it.

```
set no_post 1
```

Note: If needed to disable POST on next reboot, this variable can be saved using saveenv.

FDT command

Using FDT commands, user will be able to tweak the FDT before boot to Linux. An example of this command usage to enable the PSC module and Power domain for SRIO and Hyperlink IP is discussed here. The device tree bindings for the clock node for these IPs (functional blocks) have the status property defined and set to "disabled" by default. If user wants to enabled this so that user space drivers can be used for these IPs, the status property can be updated in the DTB blob through fdt command before boot to Linux. Following env variables can be set in u-boot to achieve this:-

```
setenv modify_fdt 'fdt addr ${addr_fdt}; fdt set /soc/clocks/clkhyperlink0
status "enabled"; fdt set /soc/clocks/clkhyperlink1 status "enabled"; fdt
set /soc/clocks/clksrio status "enabled";'
setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot}
modify_fdt run_mon run_kern'
```

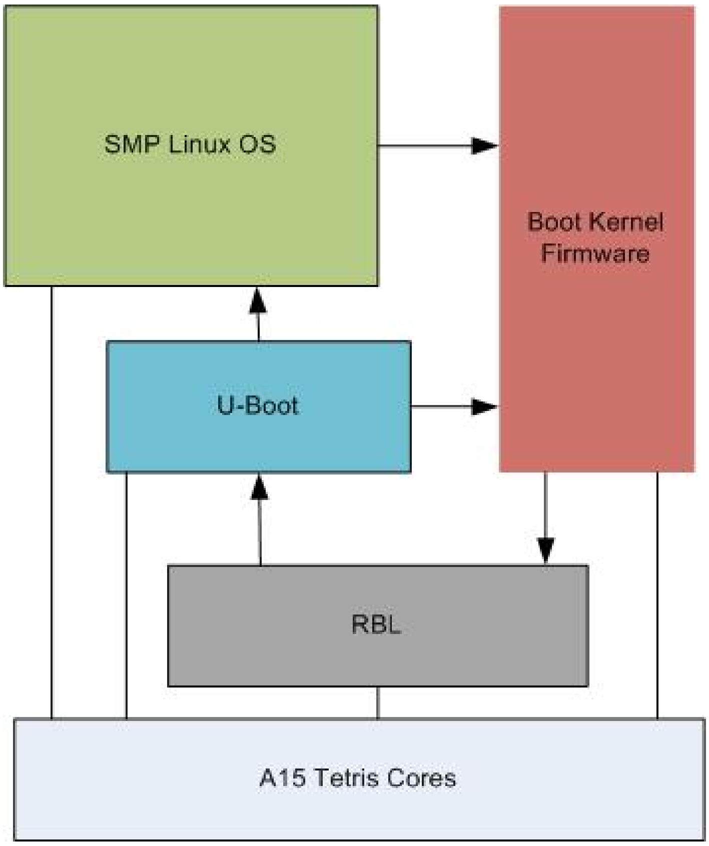
User may modify the above based on which IP is to be enabled. In the above example, two hyperlink and one srio IPs are enabled (LPSC module is enabled as well the Power Domain). This command will be useful to fix up DT bindings in the DTB before booting to Linux.

Reserve DDR memory

U-Boot has mem_reserve env variable to reserve DDR3 memory at the end of the 32 bit address space. This will be useful for reserving memory for DSP. Based on the memory availability on the board, the address range of this region will change. So any users of this feature need to make sure the address match with what is reserved through this mechanism. Otherwise the user application can step into kernel memory space and cause kernel crash during system operation. By default 512M memory is reserved at the end of the address space. To change the default size, user need to update this env variable and save the configuration using saveenv command.

Boot Monitor

Boot Monitor software provides secure privilege level execution service for Linux kernel code through SMC calls. ARM cortex A15 requires certain functions to be executed in the PL1 privilege level. Boot monitor code provides this service. A high level architecture of the boot moinitor software is shown below



Boot monitor code is built as a standalone image and is loaded into MSMC SRAM at 0xc5f0000 and occupies top 64K of the memory. The image has to be loaded to the at the above address either through CCS or through tftp or other means. It gets initialized through the u-boot command install_skern. The command takes the load address above as the argument.

Boot sequence of primary core

In the primary ARM core, ROM boot loader (RBL) code is run on Power on reset. After completing its task, RBL load and run u-boot code in the non secure mode. Boot monitor gets install through the command mon_install(). As part of this following will happen

- boot monitor primary core entry point is entered via the branch address 0xc5f0000
- As part of non secure entry, boot monitor calls the RBL API (smc #0) through SMC call passing the _skern_init() as the argument. This function get called as part of the RBL code
- _skern_init() assembly function copies the RBL stack to its own stack. It initialize the monitor vector and SP to point to its own values. It then calls skern_init() C function to initialize to do Core or CPU specific initialization. r0 points to where it enters from primary core or secondary core, r1 points to the Tetris PSC base address and r2 points to the ARM Arch timer clock rate. RBL enters this code in monitor mode. skern_init() does the following:-
- Initialize the arch timer CNTFREQ
- Set the secondary core non secure entry point address in the ARM magic address for each core
- Configure GIC controller to route IPC interrupts

Finally the control returns to RBL and back to non secure primary core boot monitor entry code.

- On the primary core, booting of Linux kernel happens as usual through the bootm command.
- At Linux start up, primary core make smc call to power on each of the secondary core. smc call is issued with r0 pointing to the command (0 - power ON). r1 points to the CPU number and r2 to secondary core kernel entry point address. Primary core wait for secondary cores to boot up and then proceeds to rest of booting sequence.

Boot sequence of secondary cores

At the secondary core, following squence happens

- On power ON reset, RBL initializes. It then enters the secondary entry point address of the boot monitor core. The init code calls smc #0 to invoke _skern_init() as a privilege function and in turn sets its own stack, vectors and smc service functions. _skern_init() calls skern_init() C function to initialize per CPU variables. It initialize the arch timer CNTFREQ to desired value.
- On return from _skern_init(), it jumps to the secondary kernel entry point address and start booting secondary instance of Linux kernel.

LPAE

The DDR3A memory address space is 0x800000000-0x9FFFFFFFF. That range is outside the first 4GB of address space. Even if Cortex-A15 supports Large Physical Address Extension (LPAE), which allows 40 bits address bus, and may access the DDR3A address space, that is possible only when ARM turns MMU on. Before that ARM has 32-bits effective address bus and can access only first 4GB of address space.

In order to give access to the DDR3A when MMU is off, MSMC maps first 2GB of DDR3A to the 0x0080000000-0x00FFFFFFFF address space. That aliased address space of the first 2GB of the DDR3A may be used not only when MMU is off, but when it is on as well. Be aware that Keystone2 doesn’t support cache coherency for the aliased address space.

The memory property in the tci6638-evm.dts file specifies the memory range available for kernel. The default sets 0x80000000 as start address and 0x20000000 as size. That corresponds to the aliased DDR3A memory range.

When u-boot boots the kernel, it can modify the start address of the “memory” in the DTB. If the “**mem_lpae**” environment variable is “0”, u-boot doesn’t modify the start address. If the **mem_lpae=1**, u-boot sets the start address to 0x800000000, which corresponds to non-aliased start address of the DDR3A.

Using more than 2GB of DDR3A memory

U-boot works without MMU enabled and its address range is limited to the first 4GB. The 2GB of that range is the DDR3A aliased address range. Therefore u-boot without MMU enabled cannot detect more than 2GB of DDR3A size. Kernel works with MMU enabled and when LPAE is enable may use up to 8GB of the DDR3A memory.

To pass the memory size information to the kernel u-boot fixes the memory node of the DTB. If the SO-DIMM size is less or equal 2GB u-boot creates memory node with one bank only:

```
memory {
      reg = <0x00000008 0x00000000 0x00000000 0x80000000>;
};
```

U-boot may reserve part of the memory at the beginning or end of the first 2GB. If the "mem_reserve_head" variable is set, u-boot modifies the start address and the size of the memory bank on the mem_reserve_head. For example if the mem_reserve_mem=256M u-boot creates the following memory node:

```
memory {
      reg = <0x00000008 0x10000000 0x00000000 0x70000000>;
};
```

If the mem_reserve variable is set, u-boot reserves the memory at the end of the first bank. For example if the mem_reserve=512M u-boot creates the following memory node:

```
memory {
      reg = <0x00000008 0x00000000 0x00000000 0x60000000>;
};
```

If the board has SO-DIMM bigger than 2GB user has to set the ddr3a_size environment variable. The variable may be set to 4 or 8. If u-boot detects the 2GB of memory (maximum it may detect), it checks the "mem_lpae" and "ddr3a_size" variables. If the mem_lpae=1 and ddr3a_size=4 or 8, u-boot creates the memory node with two banks. The first bank represents first 2GB of memory with possible reserved memory. The second bank represent the remaining memory. That memory starts at the fixed address and size equal "ddr3a_size - 2".

For example if mem_lpae=1, mem_reserve=512M and ddr3a_size=8 u-boot creates the following memory node:

```
memory {
        reg = <0x00000008 0x00000000 0x00000000 0x60000000
               0x00000008 0x80000000 0x00000001 0x80000000>;
};
```

It is important that u-boot creates two memory banks. That allows to reserve memory at the first bank (first 2GB of memory). There are SOC masters which have only 32 bit address bus and may access the first 2GB of the DDR3A only.

It is possible to reserve memory at the second bank. A customer has to modify the u-boot ft_board_setup() function to do that.

DDR3 ECC

DDR3 error detection and correction feature is enabled on K2H/K2K PG 2.0 devices and K2L/K2E devices. The DDR3 controller supports ECC on the data written or read from the SDRAM and is enabled by programming the ECC Control register. 8-bit ECC is calculated over 64-bit data quanta. The ECC is calculated for all accesses that are within the address ranges protected by ECC. 1-bit error is correctable by ECC and 2-bit error is not correctable and will be treated as unrecoverable error by sotware and trigger the reset of the device.

DDR3 ECC Handling in U-boot

U-boot checks if the DDR3 controller supports ECC RMW or not. If ECC RMW is not supported (in K2H/K2K PG1.x devices), U-boot will disable the ECC by default, otherwise it always enables ECC (in K2H/K2K PG2.0 devices and K2L/K2E devices)

During the ECC initialization, U-boot fills the entire memory (up to 8GB) to zeros using an EDMA channel after ECC is enabled. For K2H/K2K/K2L device, U-boot configures the chip level interrupt controller to route the DDR3 ECC error interrupt to ARM interrupt controller. For K2E device, since DDR3 ECC error interrupt is directly routed to ARM interrupt controller, there is no need to configure the chip level interrupt controller.

A new DDR3 command is added to simulate the ECC error, the command format is:

```
ddr ecc_err <addr in hex> <bit_err in hex> - generate bit
errors in DDR data at <addr>, the command will read a 32-bit data
 from <addr>, and write (data ^ bit_err) back to <addr>

E.g.:
ddr ecc_err 0x90000000 0x1 (this will genereate a 1-bit error on bit 0
of the data in ddr address 0x9000_0000)
ddr ecc_err 0xa0000000 0x1001 (this will genereate 2-bit error on bit 0
 & 3 of the data in ddr address 0xa000_0000)
```

A new environment variable "ecc_test" is also introduced to test ECC. By default, ecc_test = 0, and any detection of 2-bit error will reset the device. If ecc_test = 1, U-boot will bypass the error and continues to boot Linux kernel so that Linux kernel can handle the error in interrupt service.

DDR3 ECC Handling in Linux kernel

Linux kernel requests an IRQ handler for DDR3 ECC error interrupt, the handler checks the DDR3 controller interrupt status register, if the error is 2-bit error, Linux kernel will reboot the device. User can also use a user mode command to read the DDR3 ECC registers (e.g. 1-bit error count register, etc.), the DDR3 controller register and interrupt mapping are defined in the sysctrl node of device tree binding:

```
E.g. K2HK SOC device tree:
sysctrl {
      reg = <0x21010000 0x0200>; /* DDR3 controller reg */
      interrupts = <0 24 0xf01    /* L1L2 ECC error interrupt */
                  0 448 0xf01>; /* DDR3 ECC error interrupt */
};
```

Linux Kernel

The Keystone II devices runs SMP Linux and is based on upstream Linux version. The Linux port uses arch/arm/mach-keystone for the machine specific intialization code. Linux is boot up through U-boot on the primary ARM core. Linux depends on the Boot Monitor software that gets installed through U-Boot for SMP boot up. Please refer the Boot Monitor section for more details on Linux boot up sequence.

Device Tree bindings are used to define the configuration of the board to use for Linux boot up. This makes it easy for users to define bindings for their custom board so that same image provided in the release may be used to run on their board. Note that any additional peripheral drivers required on the custom board is ther responsobility of the user. User will be able to use the Device Tree bindings to configure the driver. The traditional machine setup code based device initialization support is not available for Keystone devices.

SoC Support

SoC specifc initialization and setup code is located under arch/arm/mach-keystone folder. Here are the basic init done for Keystone II devices.

- pmc.c - Power Management Run time init code
- platsmp.c - Provides smp_operations for SMP Linux operation. This is where code to Power ON secondary cores resides.
- keystone.c - DT_MACHINE_START macro for Keystone I and Keystone II devices are defined in this file. This also does clock, arch timers and Device Tree specific initialization. It provides machine specific restart() function for Linux.

GIC irq chip driver

gic irqchip driver is re-used for the Keystone II devices. This the parent irq controller. Most of the peripheral driver bindings include phandle for this irq controller. This driver is currently located under arch/arm/common folder and is moved to drivers/irqchip folder in recent upstream kernel releases. Device bindings for the driver can be found in Linux kernel tree under Documentation/arm/gic.txt

Keystone IPC irq chip driver

This is the irq controller driver for receiving IPC interrupts from DSPs. It manages the IPCGRx and IPCARx registers for the Host. It uses GIC irq controller as the parent. The bindings are similar to that for GIC. However it uses "interrupts" property in the bindings to define the parent interrupt line to the GIC interrupt controller. Two cells are used to identify an interrupt line to IPC IRQ chip. Up to 28 interrupt lines (Logical) are terminated on this device and maps to bits of the above IPC registers. One user of this driver is the remote proc user driver that uses these interrupts to receive interrupts from DSPs. The bindings for rproc user driver provides "interrupts" property to identify the specific interrupts terminated on this device.

SMP

Boot Setup

SMP Linux boot support is implemented in U-Boot, Boot Monitor and Linux. In the U-Boot, a command mon_install() is added to allow initialization of boot monitor. This command is to be included in bootcmd as part of U-Boot environment setup. This command assumes that boot monitor image is loaded in MSMC Memory prior to the invokation of the command. More details on Boot Monitor is provided elsewhere in this document. Linux platform code uses the SMC calls to inoke monitor services provided by boot monitor, for example smc call to Power ON secondary core is one such service.

Common Clock framework

In this release, all of the clock hardware nodes in the platform are represented in device tree. Here is the high level view of the clock tree

```
refclkmain (fixed clock)
  - mainpllclk (Main PLL clock)
     - mainmuxclk (Mux clock)
        - chipclk1 (chip fixed factor clocks)
           - clkusb (PSC clocks)
```

Please refer the Device data manual for the description of various clocks available on the SoC. The devices gets the Main PLL input clock from the external source. The refclkmain represents this fixed clock. Main PLL output clock is represented bu mainpllclk node. The output of this clock is fed to the mainmuxclk which can either select refclk directly (bypassing the Main PLL) or the output of the Main PLL to pass to the next stage which are divider clocks. The SYSCLKx are the divider clocks that then gets distributed to various IPs. The PSC clocks represents the LPSC (local power sleep controller) availale for individual IPs.

The driver for each of the above clock device is shown below

1. fixed clock - example refclkmain

```
drivers/clk/clk-fixed-rate.c
```

2. Main PLL clock - example mainpllclk

```
drivers/clk/clk-keystone-pll.c
```

3. (Mux clock) - example mainmuxclk

```
drivers/clk/clk-mux.c
```

4. fixed factor clocks (example chipclk1)

```
drivers/clk/clk-fixed-factor.c
```

5. Psc clocks (example - clkusb)

```
drivers/clk/davinci/clk-davinci-psc.c
```

6. Common clock init code

```
drivers/clk/davinci/davinci-clock.c
```

More details on common clock framework is part of Linux kernel documentation at Documentation/clk.txt Device clock bindings are documented at Documentation/devicetree/bindings/clock/*

The common clock code is enable through the CONFIG_COMMON_CLK KConfig option.

There are debugfs entries for each clock node to display the rates, usecounts etc. This replace the old debugfs entry to display the clock information. For example to show the main PLL clock rate, use the following command

```
root@tci6614-evm:~# mount -t debugfs debugfs /debug
root@tci6638-evm:~# cat /sys/kernel/debug/clk/refclk-main/mainpllclk/clk_rate
798720000
```

AEMIF Driver

DaVinci AEMIF driver is re-used for Keystone devices. The driver is implemented in drivers/memory/davinci-aemif.c. The binding definitions are given in Documentation/devicetree/bindings/memory/davinci-aemif.txt. This driver allows configuration of the AEMIF bus connected to slave devices such as NAND. The bindings exposes the timing parameters required for the slave device.

NAND Driver

Davinci NAND controller driver is re-used for the Keystone devices. The required bindings are defined under Documentation/devicetree/bindings/mtd/davinci-nand.txt. Driver file is drivers/mtd/nand/davinci_nand.c

SPI and SPI NOR Flash Drivers

DaVinci SPI controller driver is re-used for keystone devices. Driver is implemented in drivers/spi/spi-davinci.c and the bindings in Documentation/devicetree/bindings/spi/spi-davinci.txt. The SPI NOR flash driver used is m25p80.c under drivers/mtd/devices/m25p80.c. The bindings for this driver is available under Documentation/devicetree/bindings/mtd/st-m25p.txt

I2C and EEPROM Drivers

DaVinci I2C controller driver is re-used for Keystone devices. The driver file is drivers/i2c/chip/i2c-davinci.c and the binding is defined in Documentation/devicetree/bindings/i2c/davinci.txt. EEPROM driver is a i2c client driver and is implemented in drivers/misc/eeprom/at24.c. The binding for this is also part of the davinci i2c driver bindings.

Keystone GPIO Driver

GPIO driver is implemented in drivers/gpio/gpio-keystone.c. Each instance of the driver support 32 GPIO pins (2 Banks). Currently only 2 banks are tested on keystone devices. The driver also implements gpio irq chip driver that exposes gpio pin as irq pins and maps them to the corresponding irq pins input of GIC. This driver allow user drivers to define the GPIO pins that it uses as bindings and points to the bindings of this device. Also allows user driver to use a GPIO pin as interrupt. The irq chip driver supports upto 32 irq pins. Driver uses the pin as irq or GPIO, not both. Please refer to bindings documenation at Documentation/devicetree/gpio/gpio-keystone.txt for details of usage. Note that user driver can also use the GPIO pins using the GPIO LIB APIs. User needs to choose either defining the GPIOs through bindings or using API, and not both for a specific pin.

Keystone IPC GPIO Driver

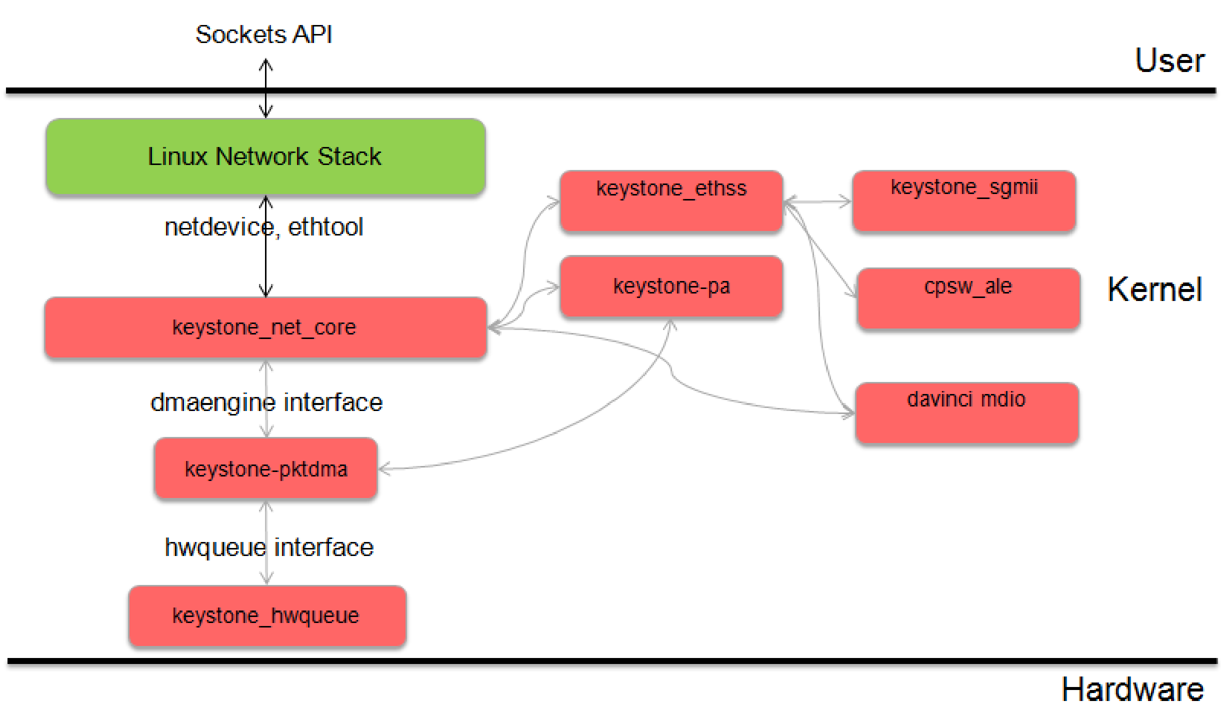
Keystone IPC GPIO driver exposes IRQs to DSPs (using IPCGRx register of each DSP) as GPIO pins. Software users allocates a GPIO pin using GPIO lib API or using a gpio bindings in the driver using phandle to the corresponding gpio ipc device binding. This driver allows defining one instance of the GPIO driver per DSP and allows up to 28 GPIO pins per GPIO instance. The bindings for each instance is defined in the dts file for the platform. The driver is implemented in drivers/gpio/gpio-keystone-ipc.c. Documentation/devicetree/bindings/gpio/gpio.txt has description on how to define bindings in the user driver to use a GPIO pin. The user driver uses GPIO lib API calls to write a value of 1 to the GPIO output pin that will raise IRQ to the DSP.

Watchdog Driver

The Watchdog driver is implemented in drivers/watchdog/davinci_wdt.c. For an explanation on the device tree bindings required see Documentation/devicetree/bindings/watchdog/davinci-wdt.txt.

Network Driver

NetCP Linux Driver Stack



- The NETCP driver code is at **/drivers/net/ethernet/ti/keystone_net_core.c** in the linux kernel.

Model:

1) Open:

On device open, we request DMA channels using standard dma engine APIs provided by the dmaengine layer.
The capabilities of the DMA are appropriately set and the appropriate channels are requested by name.
We invoke the dma_request_channel_by_name API to acquire dma engine channels.
The netdev open also sets up the streaming switch, SGMII, Serdes, switch and mac sliver. On Open we also configure the packet accelerator module.

2) Stop:

A standard netdev stop operation that stops the netif queue.
It sets the receive state to teardown and calls dmaengine_pause APIs from the dmaengine layer to pause both the RX and TX channel.
After NAPI is disabled, the operation goes onto release the dma channels by calling the dma_release_channel API.
The RX state is set to invalid and the ethernet susbsystem is stopped

3) Polling

A NAPI poll function is implemented, which sets the RX state to poll and resume the dma engine.
It will then go onto initialize the queues. We allocate memory for each packet and initialize the scatter list.
We iterate in a loop till we run out memory for the descriptors. These dma descriptors are acquired using a dmaengine API called device_prep_slave_sg.
We pass appropriate arguments to the API. At this instant we are not concerned about the Software info and PS info

associated with each descriptor.
netcp_rx_complete is the callback associated with the dma engine on the RX side.
The callback function checks for the correct DMA and RX state and warns the user on seeing abnormal behavior.
It then goes onto stahs the received packet to the tail of a linked list.

4) Xmit

netcp_ndo_start_xmit is the standard netdev operation that is used to push an outgoing packet.
Again we have a structure for each packet. This will contain an array of scatterlists and the number of scatterlist entries which at this point should be 1.
We then call the device_prep_slave_sg API with appropriate parameters to acquire a descriptor for the TX operation.
'netcp_tx_complete' is the callback fucntion attached to each TX dma callback.
All the network statistics are appropriately updated on a successful transfer. The callback then proceeds to free the skb with the dev_kfree_skb_any API.

5) Receive

In the napi_poll netdev operation, we call the netcp_refill_rx function which will allocate skbs and attch these skbs to a descriptor until we run out of descriptor memory. the deliver_stash routine fills the appropriate RX

DMA Engine

The PktDMA interface is an extension of the Linux DMA Engine subsystem.
It is worth reviewing the following documentation found in the kernel source tree

- Documentation/dmaengine.txt
- Documentation/devicetree/bindings/dma/keystone-pktdma.txt

Device Tree Requirements

Each PktDMA channel must be defined in the device tree prior to use.
The device tree channel description associates the channel with one or more hardware queues that must also be described in the device tree.
The PktDMA code generally identifies a device tree description by the channel label string.
For example, the Keystone Ethernet driver uses two channels, named “nettx” and “netrx”, used for outbound and inbound traffic respectively.
Transmit channels must define the “submit-queue” tag, while receive channels must define the “flow” tag.

DMA Channel Configuration

First, open a DMA channel by calling dma_request_channel_by_name(), passing in a mask requesting the DMA_SLAVE
capability and the desired channel name.
The returned pointer is used as a handle for all subsequent operations on the channel.
Note that while a NULL pointer indicates an error, there are non-NULL values that also indicate errors;
use the IS_ERR_OR_NULL() macro to test the returned value. <The next step is to configure the channel using
the dma_keystone_config() function, which takes a dma_keystone_info structure.
The fields that must be filled in depend on whether the channel is to be used as a transmit or receive channel.
In both cases, all unused fields must be zero.

For a transmit channel, set these fields:

- Set the direction field to DMA_MEM_TO_DEV.
- Set the tx_queue_depth field to the the number of transmit buffers that can be outstanding at any one time.

Attempting to prepare a buffer or chain of buffers for transmission when there is insufficient depth remaining will result in an error.

For a receive channel, set these fields:

- Set the direction field to DMA_DEV_TO_MEM.
- Set the scatterlist_size field to the number of entries in the array of scatterlist structures that will be used for reception. If the application does not support scatter/gather, then this will be between 1 and 3.
- Set the rxpool_allocator field to be a pointer to a function that the DMA engine will call to replenish the pool of free receive buffers.
- Set the rxpool_destructor field to be a pointer to a function that the DMA engine will call to free the pool of free receive buffers during channel shutdown and certain error conditions.
- The rxpool_param field is a pointer to a void that is passed to the rxpool allocator and destructor functions to make them easier to write. The PktDMA subsystem makes no other use of this field.
- Set the rxpool_thresh_enable field to DMA_THRESH_NONE. In the future the PktDMA interface will support selection of free buffer queue by packet size, but this is not currently supported.
- The rxpools array determines how the four free buffer queues are configured. If neither scatter/gather nor size thresholds are being used, fill in only the first entry in the array (rxpools[0]); this is likely to be the normal case. Each entry contains a buffer size and desired pool depth. If using packet size thresholds, the array must be filled in from smallest to largest.
- Set the rxpool_count to the number of rxpools array entries being used.

The dma_keystone_config() function will handle the opening and initialization of hardware queues, and allocation of hwqueue descriptors.
For transmit channels the number of descriptors is given by the tx_queue_depth field.
For receive channels the number of descriptors is the sum of the pool_depth fields in the rxpools array.

Notification and Completion Handlers

When an outgoing packet has been transmitted, the hardware will place the descriptor on a completion queue.
Similarly, when an incoming packet has been received, the hardware places the descriptor on a receive queue.
In either case, an interrupt may be generated.
The driver can register to be notified of the interrupt by establishing a notification handler using the dma_set_notify() function.
Note that the notification handler is called from within the hardware interrupt service routine,
and thus runs in hard interrupt context with the IRQ disabled.
Only a minimum of processing should be done in this context; typically disabling further interrupts
on the channel using the dmaengine_pause() function, and scheduling deferred work in a tasklet or work queue.

When the deferred work function is invoked, it should invoke the dma_poll() function.
This will cause the PktDMA subsystem to pop packet descriptors from the completion queue and invoke the associated completion functions.
This process continues until the completion queue is empty or the “budget” of descriptors has been exhausted.
The return value from dma_poll() is the number of packets processed.

The hwqueue subsystem configures the queues with interrupts enabled.
If a notification handler is not established for a channel and interrupts remain enabled,
the packet completion functions will be called from hard interrupt context.
This is generally not desirable. It is recommended that either a notification function and a deferred work mechanism be established,
or the completion interrupt should be disabled and the dma_poll() function called periodically.

Packet Transmission Initiation

To transmit a packet, initialize an array of scatterlist structures. If the packet descriptor is to contain an Extended Packet Information Block,
this should be described by the first entry in the scatterlist array. If the descriptor is to contain a Protocol Specific data block, this should be described next.

These areas a copied into the descriptor from system virtual address space by the CPU and are not subject to DMA requirements.

The remaining entries in the scatterlist array describe the data buffer.

A data buffer for transmission must reside in an area of memory that is (1) in an address space accessible to the PktDMA hardware, and (2) will not be

modified or reallocated until the operation has completed. If the entire data area is not physically contiguous,

each physically contiguous section must be described by a separate scatterlist array entry – a so-called “gather” operation.

Once the scatterlist array is complete, the data buffer segments must be mapped for the DMA_TO_DEVICE transfer.

If it is known that there will be only one data segment this done using the dma_map_single() function, or dma_map_sg() function if there may be multiple segments.

Do not access the buffer after it has been mapped; this transfers “ownership” of the buffer to the PktDMA device and doing so will result in unpredictable behavior.

Do not map the EPIB or PSINFO segments.

After mapping the data buffer segments, pass the scatterlist array to the dmaengine_prep_slave_sg() function,

the number of scatterlist array entries used, and flags indicating whether scatterlist array includes EPIB or PSINFO segments.

If there are no problems, this function returns the address of a dma_async_tx_descriptor structure.

Each packet should be associated with a completion routine that will be called by the PktDMA subsystem after the packet has been accepted.

To do this, fill in the callback field of the dma_async_tx_descriptor structure with the address of the completion routine.

To make coding easier, the content of the callback_param field of this structure (a void pointer) is passed to the completion routine as a parameter.

The packet is now ready for transmission. Call dmaengine_submit(), passing the address of the dma_async_tx_descriptor.

The return value is a “cookie” that will be used later to check the completion status of the packet.

Packet Transmission Completion

The packet completion routine for transmission is relatively simple.

The completion status should be checked by calling the dma_async_is_tx_complete() function.

Unfortunately, this routine requires the “cookie” returned by dmaengine_submit(), which makes for a possible race condition.

The data buffers, which were mapped for DMA earlier, must be unmapped using the dma_unmap_sg() function or equivalent.

Receive Free Queue Management

Before the PktDMA hardware can transfer a received packet to the host, the receive free queue or queues must be populated.

The desired buffer sizes and depths of these queues is established when the channel is configured.

To populate the queue, call the dma_rxfree_refill() function.

This will call the rxpool_allocator function repeatedly until the pool reaches its configured depth, or until the allocator function returns a NULL pointer.

This is done for each configured queue.

The first is the value of rxpool_param set during configuration.

The second is the free queue number, an integer value between 0 and 3, indicating which free queue is being populated.

The third is the size of the buffer to be allocated, as passed in the rxpools array.

The buffer size is not used by the PktDMA code, and is provided only for the convenience of the allocator and destructor functions.

The rxpool_allocator function must return the address of a dma_async_tx_descriptor structure (or NULL).
The sequence for this is very similar to transmission, with these exceptions:

- Only single buffers may be returned, no multi-buffer chains.
- The scatterlist arrays used for preparing buffers that may be used as the first buffer in a packet (typically queue number 0) must be discrete and persistent until the buffer is returned in a completion queue or passed to the rxpool_destructor function.
- The scatterlist arrays used for preparing buffers that may be used as the first buffer in a packet must contain at least the number of elements specified in the scatterlist_size configuration field.
- If the DMA_HAS_EPIB or DMA_HAS_PSINFO flag bits are set, the first one or two entries in the scatterlist array must be filled in with info about these buffers. The buffers themselves must be writable and persistently allocated, as they will be written to before the completion function is called.
- Bits 27:24 of the flags argument to dmaengine_prep_slave_sg() function must contain the queue number passed in as parameter 2.

The receive code must call dma_rxfree_refill() to replenish the free buffer pools as needed.
This probably is best done at the end of the deferred work routine that also calls dma_poll(), but this depends on the circumstances.
When a receive channel is closed, and in certain exception situations, the rxpool_destructor function will be called to release the buffer.
This function is not called when a packet is returned through a completion queue.
The rxpool_destructor must unmap the buffer using dma_unmap_single(), dma_unmap_sg(), or similar functions. The buffer should then be freed.

Packet Reception Completion

When the PktDMA hardware has assembled a packet it places the address of the packet descriptor for the first buffer in the completion queue.
This may trigger an interrupt, causing the notification routine to be called, which may schedule deferred work.
The deferred work routine or other mechanism must then call the dma_poll() function, which will dequeue packet descriptors
from the completion queue and invoke the per-packet completion functions.
Note that in the case of a packet formed from a chain of buffers, only the completion routine for the first buffer in the chain will be invoked.

The per-packet completion function is passed a parameter whose value was set when the packet was allocated.
This value is not used by the PktDMA subsystem; however, it is the only mechanism for the completion function to find the received packet.
The Ethernet driver passes the address of a per-packet data structure that contains the scatterlist array, the “cookie”, and other critical data.

The completion function must perform several operations:

- Check the completion status of the packet using dma_async_is_tx_complete().
- Process the content of the EPIB and/or PSinfo sections.
- Unmap the buffer or chain of buffers. If a chain of buffers has been returned, the scatterlist array for the first buffer in the chain will be populated with the addresses and sizes of all the buffers in the chain.
- Free the buffer or buffers, or pass them to another subsystem that will.

DMA Channel Teardown

To tear down a PktDMA channel, bring the channel to a quiescent state, then call the dma_release_channel() function.
Any packets queued for transmission in a TX queue will be flushed by passing them to the completion routine with an error status.
Buffers queued to the RX free buffer pools will be passed to the rxpool_destructor function for disposal.

Packet Accelerator

- The Packet Accelerator driver is at **drivers/net/ethernet/ti/keystone_pa.c**.
  - The Packet Accelerator has 6 PDSPs.
  - The packet classifier image 1 is downloaded to PDSP0, PDSP1 and PDSP2.
  - The packet classifier image 2 is downloaded to PDSP3.
  - The packet modifier image is downloaded to PDSP4 and PDSP5.

PA Timestamping

- PA timestamping has been implemented in the network driver.
- All Receive packets will be timestamped and this timestamped by PDSP0 and this timestamp will be available in the timestamp field of the descriptor itself.
- To obtain the TX timestamp, we call a PA API to format the TX packet. Essentially what we do is to add a set of params to the "PSDATA" section of the descriptor. This packet is then sent to PDSP5. Internally this will route the packet to the switch. The TX timestamp is obtained in a return packet and we have designed the network driver in such a way to obtain both ethernet RX packet and the TX timestamp packet on the same queue and flow.
- The way we differentiate between an Ethernet RX packet and a TX timestmap packet is based on the "Software info 0" field of the descriptor.
- To obtain the timestamps itself, we use generic kernel APIs and features.
- Appropriate documentation for this can be found at Timestamping Documentation [8]
- The timestamping was tested with open source timestamping test code found at Timestamping Test Code [9]

```
./timestamping eth0 SOF_TIMESTAMPING_TX_HARDWARE SOF_TIMESTAMPING_SYS_HARDWARE SOF_TIMESTAMPING_RAW_HARDWARE
```

- Please refer to this [10] document for testing PA timestamps in MCSDK3.0

Mark_mcast_match Special Packet Processing Feature

This feature provide for special packet egress processing for specific marked packets. The intended use is:

```
1) SOC Configured in multiple-interface mode
2) CPSW ALE re-enabled via /sys/class/net/eth0/device/ale_control (so that SOC switch is
   active behind the scenes)
3) NetCP interfaces slaved to a bridge
4) NetCP interfaces feed a common QoS tree
5) Bridge forwarding disabled via "ebtables -P FORWARD DROP" (because CPSW is
   doing the port to port forwarding)
```

In this rather odd situation, the bridge will transmit locally generated multicast (and broadcast) packets by sending one on each of the slaved interfaces (i.e. bridge flooding). This has two ramifications:

```
  (a) This results in multiple packets (copies of these locally generated
      mulitcasts) through a common QoS, which is considered "bad"
      because the common QOS tree is configured assuming only one copy.
  (b) even if QOS is not present, sending multiple copies of these multicasts is
      sub-optimal since the CPSW switch is capable of doing the forwarding itself given
      just one copy of the original packet.
```

To avoid these ramifications, such local multicast packets can be marked via ebtables for special processing in the NetCP PA module before the packets are queued for transmission. Packets thus recognized are NOT marked for egress via a specific slave port, and thus will be transmitted through all slave ports by the CPSW h/w forwarding logic.

To do this, a new DTS parameter "mark_mcast_match" has been added. This parameter takes two u32 values: a "match" value and a "mask" value.

When the NetCP PA module encounters a packet with a non-zero skb->mark field, it bitwise-ANDs the skb->mark value with the "mask" value and then compares the result with the "match" value. If these do not match, the mark is ignored and the packet is processed normally.

However, if the "match" value matches, then the low-order 8 bits of the skb->mark field is used as a bitmask to determine whether the packet should be dropped. If the packet would normally have been directed to slave port 1, then bit 0 of skb->mark is checked; slave port 2 checks bit 1, etc. If the bit is set, then the packet is enqueued for ALE processing but *with the CPSW engress port field in the descriptor set to 0* (indicating that CPSW is responsible for selecting the egress port(s) to forward the packet too) ; if the bit is NOT set, the packet is silently dropped.

An example...

The device tree contains this PA definition:

mark_mcast_match = <0x12345a00 0xffffff00>;

The runtime configuration scripts execute this command:

ebtables -A OUTPUT -d Multicast -j mark \ --mark-set 0x12345a01 --mark-target ACCEPT

When the bridge attempts to send an ARP (broadcast) packet, it will send one packet to each of the slave interfaces. The packet sent by the bridge to slave interface eth0 (CPSW slave port 1) will be passed to the CPSW, and the ALE will broadcast this packet on all slave ports. The packets sent by the bridge to other slave interfaces (eth1, CPSW slave port 2) will be silently dropped.

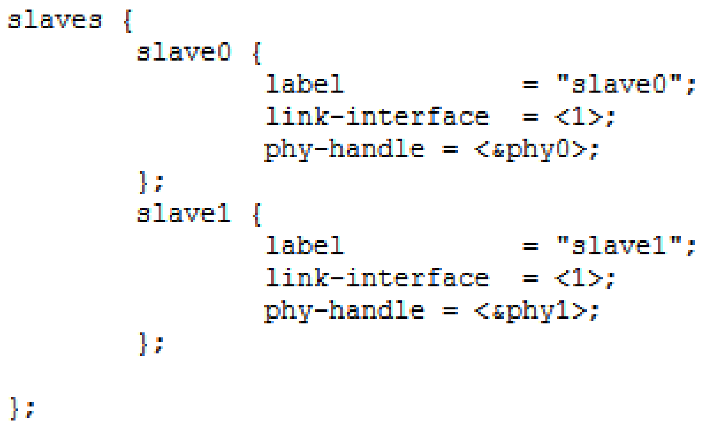
SGMII

- The SGMII driver code is at **drivers/net/ethernet/ti/keystone_sgmii.c**.

The SGMII module on Keystone 2 devices can be configured to operate in various modes.
The modes are as follows

- mac mac autonegotiate
- mac phy
- mac mac forced
- mac fiber

The mode of operation can be decided through the device tree bindings. An example is shown below



```
slaves {
        slave0 {
                label           = "slave0";
                link-interface  = <1>;
                phy-handle = <&phy0>;
        };
        slave1 {
                label           = "slave1";
                link-interface  = <1>;
                phy-handle = <&phy1>;
        };

};
```

AS we can see in the above diagrarm, the **link-interface** attribute must be appropriately changed to decide the mode of operation.

- For mac mac auto negotiate use 0
- For mac phy use 1
- For mac mac forced use 2
- For mac fiber use 3

Setting up an NFS filesystem

An NFS filesystem can be setup during development.

To setup an NFS filesystem, first obtain a tarball of the filesystem. This should be untarred on the linux host machine.
This section will explain how to setup an NFS server on a Ubuntu linux host.

- Untar the filesystem that is made as part of the release into location **/opt/filesys**
- On the linux host open file **/etc/exports** and add the following line

```
/opt/filesys *(rw,subtree_check,no_root_squash,no_all_squash,sync)
```

- Please note that the location of the fielsystem on the host should be the same as that in /etc/exports
- Next we have to start the nfs server and this is achieved by giving the following command

```
/etc/init.d/nfs-kernel-server restart
```

Modifying the command line for NFS filesystem boot

The kernel command line should be appropriately modified to enable kernel boot up with an NFS filesystem

Add the following to the kernel command line **root=/dev/nfs nfsroot=192.168.1.140:/opt/filesys,v3,tcp rw ip=dhcp**

To use eth1 set ***ip=dhcp:eth1***

During kernel boot up, in the boot up log we should be able to see the following

Kernel command line: earlyprintk debug console=ttyS0,115200n8 ip=dhcp mem=512M rootwait=1 rootfstype=nfs root=/dev/nfs rw nfsroot=192.168.1.140:/opt/filesys,v3,tcp

Modifying CPSW Configurations Through Sysfs User Interface

Through sysfs, an user can show or modify some ALE control, ALE table and CPSW control configurations from user space by using the commands described in the following sub-sections.

1. Showing ALE Table

- Command to show the table entries.

```
$ cat /sys/devices/soc.0/2090000.netcp/ale_table
```

- One execution of the command may show only part of the table. Consecutive executions of the command will show the remaining parts of the table (see example below).
- The '+' sign at the end of the show indicates that there are entries in the remaining table not shown in the current execution of the command (see example below).

2. Showing RAW ALE Table

- Command to show the raw table entries.

```
$ cat /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

- Command to set the start-showing-index to n.

```
$ echo n > /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

- Only raw entries (without interpretation) will be shown.
- Depending on the number of occupied entries, it is more likely to show the whole table with one execution of the raw table show command. If not, consecutive executions of the command will show the remaining parts of the table.
- The '+' sign at the end of the show indicates that there are entries in the remaining table not shown in the current execution of the command (see example below).

3. Showing ALE Controls

- Command to show the ale controls.

```
$ cat /sys/devices/soc.0/2090000.netcp/ale_control
```

4. Showing CPSW Controls

- Command to show various CPSW controls

```
$ cat /sys/devices/soc.0/2090000.netcp/cpsw/file_name
```

where file_name is a file under the directory /sys/devices/soc.0/2090000.netcp/cpsw.

- Files or directories under the cpsw directory are
  - control
  - flow_control
  - port_tx_pri_map/
  - port_vlan/
  - priority_type
  - version
- For example, to see the CPSW version, use the command

```
$ cat /sys/devices/soc.0/2090000.netcp/cpsw/version
```

5. Adding/Deleting ALE Table Entries

- In general, the ALE Table add command is of the form

```
   $ echo "add_command_format" > /sys/devices/soc.0/2090000.netcp/ale_table
```

or

```
   $ echo "add_command_format" > /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

- The delete command is of the form

```
   $ echo "n:" > /sys/devices/soc.0/2090000.netcp/ale_table
```

or

```
   $ echo "n:" > /sys/devices/soc.0/2090000.netcp/ale_table_raw
```

where n is the index of the table entry to be deleted.

- Command Formats
  - Adding VLAN command format

```
v.vid=(int).force_untag_egress=(hex 3b).reg_fld_mask=(hex 3b).unreg_fld_mask=(hex 3b).mem_list=(hex 3b)
```

  - Adding OUI Address command format

```
o.addr=(aa:bb:cc)
```

  - Adding Unicast Address command format

```
u.port=(int).block=(1|0).secure=(1|0).ageable=(1|0).addr=(aa:bb:cc:dd:ee:ff)
```

  - Adding Multicast Address command format

```
m.port_mask=(hex 3b).supervisory=(1|0).mc_fw_st=(int 0|1|2|3).addr=(aa:bb:cc:dd:ee:ff)
```

  - Adding VLAN Unicast Address command format

```
vu.port=(int).block=(1|0).secure=(1|0).ageable=(1|0).addr=(aa:bb:cc:dd:ee:ff).vid=(int)
```

  - Adding VLAN Multicast Address command format

```
vm.port_mask=(hex 3b).supervisory=(1|0).mc_fw_st=(int 0|1|2|3).addr=(aa:bb:cc:dd:ee:ff).vid=(int)
```

  - Deleting ALE Table Entry

```
entry_index:
```

  - Remark: any field that is not specified defaults to 0, except vid which defaults to -1 (i.e. no vid).
- Examples
  - Add a VLAN with vid=100 reg_fld_mask=0x7 unreg_fld_mask=0x2 mem_list=0x4

```
$ echo "v.vid=100.reg_fld_mask=0x7.unreg_fld_mask=0x2.mem_list=0x4" > /sys/class/net/eth0/device/ale_table
```

  - Add a persistent unicast address 02:18:31:7E:3E:6F

```
   $ echo "u.addr=02:18:31:7E:3E:6F" > /sys/class/net/eth0/device/ale_table
```

  - Delete the 100-th entry in the table

```
   $ echo "100:"  > /sys/class/net/eth0/device/ale_table
```

5. Modifying ALE Controls

- Access to the ALE Controls is available through the /sys/class/net/eth0/device/ale_control pseudo file. This file contains the following:
- • version: the ALE version information
- • **enable**: 0 to disable the ALE, 1 to enable ALE (should be 1 for normal operations)
- • **clear**: set to 1 to clear the table (refer to [1] for description)
- • **ageout** : set to 1 to force age out of entries (refer to [1] for description])
- • **p0_uni_flood_en** : set to 1 to enable unknown unicasts to be flooded to host port. Set to 0 to not flood such unicasts. Note: if set to 0, CPSW may delay sending packets to the SOC host until it learns what mac addresses the host is using.
- • **vlan_nolearn** : set to 1 to prevent VLAN id from being learned along with source address.
- • **no_port_vlan** : set to 1 to allow processing of packets received with VLAN ID=0; set to 0 to replace received packets with VLAN ID=0 to the VLAN set in the port's default VLAN register.
- • **oui_deny** : 0/1 (refer to [1] for a description of this bit)
- • **bypass**: set to 1 to enable ALE bypass. In this mode the CPSW will not act as switch on receive; instead it will forward all received traffic from external ports to the host port. Set to 0 for normal (switched) operations.
- • **rate_limit_tx**: set to 1 for rate limiting to apply to transmit direction, set to 0 for receive direction. Refer to [1] for a description of this bit.
- • **vlan_aware**: set to 1 to force the ALE into VLAN aware mode
- • **auth_enable**: set to 1 to enable table update by host only. Refer to [1] for more details on this feature
- • **rate_limit**: set to 1 to enable multicast/broadcast rate limiting feature. Refer to [1] for more details.
- • **port_state.0=** set the port 0 (host port) state. State can be:
- o 0: disabled
- o 1: blocked
- o 2: learning
- o 3: forwarding
- • **port_state.1**: set the port 1 state.
- • **port_state.2**: set the port 2 state
- • **drop_untagged.0** : set to 1 to drop untagged packets received on port 0 (host port)
- • **drop_untagged.1** : set to 1 to drop untagged packets received on port 1
- • **drop_untagged.2** : set to 1 to drop untagged packets received on port 2
- • **drop_unknown.0** : set to 1 to drop packets received on port 0 (host port) with unknown VLAN tags. Set to 0 to allows these to be processed
- • **drop_unknown.1** : set to 1 to drop packets received on port 1 with unknown VLAN tags. Set to 0 to allow these to be processed.
- • **drop_unknown.2** : set to 1 to drop packets received on port 2 with unknown VLAN tags. Set to 0 to allow these to be processed.
- • **nolearn.0** : set to 1 to disable address learning for port 0
- • **nolearn.1** : set to 1 to disable address learning for port 1
- • **nolearn.2** : set to 1 to disable address learning for port 2
- • **unknown_vlan_member** : this is the port mask for packets received with unknown VLAN IDs. The port mask is a 5 bit number with a bit representing each port. Bit 0 refers to the host port. A '1' in bit position N means include the port in further forwarding decision. (e.g., port mask = 0x7 means ports 0 (internal), 1 and 2 should be included in the forwarding decision). Refer to [1] for more details.
- • **unknown_mcast_flood=** : this is the port mask for packets received with unkwown VLAN ID and unknown (un-registered) destination multicast address. This port_mask will be used in the multicast flooding decision. unknown multicast flooding.

- • **unknown_reg_flood**: this is the port mask for packets received with unknown VLAN ID and registered (known) destination multicast address. It is used in the multicast forwarding decision.
- • **unknown_force_untag_egress**: this is a port mask to control if VLAN tags are stripped off on egress or not. Set to 1 to force tags to be stripped by h/w prior to transmission
- • **bcast_limit.0** : threshold for broadcast pacing on port 0 .
- • **bcast_limit.1**: threshold for broadcast pacing on port 1.
- • **bcast_limit.2** : threshold for broadcast pacing on port 2 .
- • **mcast_limit.0**: threshold for multicast pacing on port 0 .
- • **mcast_limit.1**: threshold for multicast pacing on port 1 ..
- • **mcast_limit.2**: threshold for multicast pacing on port 2 .
- Command format for each modifiable ALE control is the same as what is displayed for that field from showing the ALE table.
- For example, to disable ALE learning on port 0, use the command

```
  $ echo "nolearn.0=0" > cat /sys/devices/soc.0/2090000.netcp/ale_control
```

6. Modifying CPSW Controls

- Command format for each modifiable CPSW control is the same as what is displayed for that field from showing the CPSW controls.
- For example, to enable flow control on port 2, use the command

```
  $ echo "port2_flow_control_en=1" > /sys/devices/soc.0/2090000.netcp/cpsw/flow_control
```

7. Resetting CPSW Statistics

- Use the command

```
  $ echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/A
```

or

```
  $ echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/B
```

to reset statistics module A or B counters.

8. Additional Examples

To enable CPSW:

//enable unknown unicast flood to host, disable bypass, enable VID=0 processing

echo “p0_uni_flood_en=1” > /sys/class/net/eth0/device/ale_control
echo “bypass=0” > /sys/class/net/eth0/device/ale_control
echo “no_port_vlan=1” > /sys/class/net/eth0/device/ale_control

To disable CPSW:

// disable port 0 flood for unknown unicast;
//enable bypass mode
echo “p0_uni_flood_en=0” > /sys/class/net/eth0/device/ale_control
echo “bypass=1” > /sys/class/net/eth0/device/ale_control

To set port 1 state to forwarding:

echo “port_state.1=3” > /sys/class/net/eth0/device/ale_control

To set CPSW to VLAN aware mode:

echo “vlan_aware=1” > /sys/class/net/eth0/device/cpsw/control
echo “ale_vlan_aware=1” > /sys/class/net/eth0/device/ale_control
(set these to 0 to disable vlan aware mode)

To set port 1's Ingress VLAN defaults:

echo "port.1._vlan_id=5" > /sys/class/net/eth0/device/cpsw/port_vlan/1

echo "port.1._cfi=0" > /sys/class/net/eth0/device/cpsw/port_vlan/1

echo "port.1._vlan_pri=0" > /sys/class/net/eth0/device/cpsw/port_vlan/1

To set port 1 to use the above default vlan id on ingress:

echo "port.1.pass_pri_tagged=0" > /sys/class/net/eth0/device/cpsw/control

To set port 1's Egress VLAN defaults:

• For registered VLANs, the egress policy is set in the "force_untag_egress field" of the ALE entry for that VLAN. This field is a bit map with one bit per port. Port 0 is the host port. For example, to set VLAN #100 to force untagged egress on port 2 only:

echo "v.vid=100.force_untag_egress=0x4.reg_fld_mask=0x7.unreg_fld_mask=0x2.mem_list=0x4" > /sys/class/net/eth0/device/ale_table

• For un-registered VLANs, the egress policy is set in the ALE unknown vlan register, which is accessed via the ale_control pseudo file. The value is a bit map, one bit per port (port 0 is the host port). for example, set every port to drop unknown VLAN tags on egress

echo "unknown_force_untag_egress=7" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Admit tagged" (i.e. drop un-tagged) :

echo "drop_untagged.1=1" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Admit all" :

echo "drop_untagged.1=0" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Admit unknown VLAN":

echo "drop_unknown.1=0" > /sys/class/net/eth0/device/ale_control

To set to Port 1 to "Drop unknown VLAN":

echo "drop_unknown.1=1" > /sys/class/net/eth0/device/ale_control

9. Sample Display

```
root@keystone-evm:~#

root@keystone-evm:~# ls -l /sys/devices/soc.0/2090000.netcp/

-rw-r--r--    1 root     root          4096 Feb 28 20:32 ale_control

-r--r--r--    1 root     root          4096 Feb 28 20:30 ale_table

drwxr-xr-x    2 root     root             0 Feb 28 20:46 cpsw

lrwxrwxrwx    1 root     root             0 Jan  1  1970 driver -> ../../../bus/platform/drivers/keystone-netcp

-r--r--r--    1 root     root          4096 Feb 28 20:46 modalias

drwxr-xr-x    4 root     root             0 Jan  1  1970 net

drwxr-xr-x    2 root     root             0 Feb 28 20:46 power

lrwxrwxrwx    1 root     root             0 Jan  1  1970 subsystem -> ../../../bus/platform

-rw-r--r--    1 root     root          4096 Jan  1  1970 uevent

root@keystone-evm:~#

root@keystone-evm:~#

root@keystone-evm:~# ls -l /sys/devices/soc.0/2090000.netcp/cpsw

-rw-r--r--    1 root     root          4096 Feb 28 20:31 control

-rw-r--r--    1 root     root          4096 Feb 28 20:31 flow_control

drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_tx_pri_map

drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_vlan

-rw-r--r--    1 root     root          4096 Feb 28 20:31 priority_type

drwxr-xr-x    2 root     root             0 Feb 28 20:31 stats
```

```
-r--r--r--    1 root     root          4096 Feb 28 20:31 version

root@keystone-evm:~#

root@keystone-evm:~# ls -l /sys/class/net/eth0/device/

-rw-r--r--    1 root     root          4096 Feb 28 20:32 ale_control

-r--r--r--    1 root     root          4096 Feb 28 20:32 ale_table

drwxr-xr-x    2 root     root             0 Feb 28 20:30 cpsw

lrwxrwxrwx    1 root     root             0 Jan  1  1970 driver -> ../../../bus/platform/drivers/keystone-netcp

-r--r--r--    1 root     root          4096 Feb 28 20:32 modalias

drwxr-xr-x    4 root     root             0 Jan  1  1970 net

drwxr-xr-x    2 root     root             0 Feb 28 20:32 power

lrwxrwxrwx    1 root     root             0 Jan  1  1970 subsystem -> ../../../bus/platform

-rw-r--r--    1 root     root          4096 Jan  1  1970 uevent

root@keystone-evm:~#

root@keystone-evm:~# ls -l /sys/class/net/eth0/device/cpsw/

-rw-r--r--    1 root     root          4096 Feb 28 20:31 control

-rw-r--r--    1 root     root          4096 Feb 28 20:31 flow_control

drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_tx_pri_map

drwxr-xr-x    2 root     root             0 Feb 28 20:31 port_vlan

-rw-r--r--    1 root     root          4096 Feb 28 20:31 priority_type

drwxr-xr-x    2 root     root             0 Feb 28 20:31 stats

-r--r--r--    1 root     root          4096 Feb 28 20:31 version

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/version

cpsw version 1.3 (1) SGMII identification value 0x4ed1

root@keystone-evm:~#

root@keystone-evm:~#

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/control

fifo_loopback=0

vlan_aware=1

p0_enable=1

p0_pass_pri_tagged=0

p1_pass_pri_tagged=0

p2_pass_pri_tagged=0

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/flow_control

port0_flow_control_en=1

port1_flow_control_en=0

port2_flow_control_en=0

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/priority_type

escalate_pri_load_val=0

port0_pri_type_escalate=0

port1_pri_type_escalate=0

port2_pri_type_escalate=0

root@keystone-evm:~#

root@keystone-evm:~#
```

```
root@keystone-evm:~# ls -l /sys/class/net/eth0/device/cpsw/port_tx_pri_map/

-rw-r--r--    1 root     root          4096 Feb 28 21:06 1

-rw-r--r--    1 root     root          4096 Feb 28 21:06 2

-rw-r--r--    1 root     root          4096 Feb 28 21:06 3

-rw-r--r--    1 root     root          4096 Feb 28 21:06 4

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/1

port_tx_pri_0=1

port_tx_pri_1=0

port_tx_pri_2=0

port_tx_pri_3=1

port_tx_pri_4=2

port_tx_pri_5=2

port_tx_pri_6=3

port_tx_pri_7=3

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/2

port_tx_pri_0=1

port_tx_pri_1=0

port_tx_pri_2=0

port_tx_pri_3=1

port_tx_pri_4=2

port_tx_pri_5=2

port_tx_pri_6=3

port_tx_pri_7=3

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/3

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_tx_pri_map/4

root@keystone-evm:~#

root@keystone-evm:~# ls -l /sys/class/net/eth0/device/cpsw/port_vlan/

-rw-r--r--    1 root     root          4096 Feb 28 20:30 0

-rw-r--r--    1 root     root          4096 Feb 28 20:30 1

-rw-r--r--    1 root     root          4096 Feb 28 20:30 2

-rw-r--r--    1 root     root          4096 Feb 28 20:30 3

-rw-r--r--    1 root     root          4096 Feb 28 20:30 4

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/0

port_vlan_id=0

port_cfi=0

port_vlan_pri=0

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/1

port_vlan_id=0

port_cfi=0

port_vlan_pri=0

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/2
```

```
port_vlan_id=0
port_cfi=0
port_vlan_pri=0
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/3
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/cpsw/port_vlan/4
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_control
version=(ALE_ID=0x0029) Rev 1.3
enable=1
clear=0
ageout=0
p0_uni_flood_en=0
vlan_nolearn=0
no_port_vlan=1
oui_deny=0
bypass=1
rate_limit_tx=0
vlan_aware=1
auth_enable=0
rate_limit=0
port_state.0=3
port_state.1=3
port_state.2=0
drop_untagged.0=0
drop_untagged.1=0
drop_untagged.2=0
drop_unknown.0=0
drop_unknown.1=0
drop_unknown.2=0
nolearn.0=0
nolearn.1=0
nolearn.2=0
unknown_vlan_member=7
unknown_mcast_flood=6
unknown_reg_flood=7
unknown_force_untag_egress=7
bcast_limit.0=0
bcast_limit.1=0
bcast_limit.2=0
mcast_limit.0=0
mcast_limit.1=0
mcast_limit.2=0
root@keystone-evm:~# echo "p0_uni_flood_en=1" >  /sys/class/net/eth0/device/ale_control
root@keystone-evm:~#
root@keystone-evm:~#
```

```
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_control | grep p0_uni_flood_en
p0_uni_flood_en=1
root@keystone-evm:~#
root@keystone-evm:~# echo "drop_unknown.1=1" > /sys/class/net/eth0/device/ale_control
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_control | grep drop_unknown
drop_unknown.0=0
drop_unknown.1=1
drop_unknown.2=0
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table
index 0, raw: 0000001c d000ffff ffffffff, type: addr(1), addr: ff:ff:ff:ff:ff:ff, mcstate: f(3), port mask: 7, no super
index 1, raw: 00000000 10000017 eaf4323a, type: addr(1), addr: 00:17:ea:f4:32:3a, uctype: persistant(0), port: 0
index 2, raw: 0000001c d0003333 00000001, type: addr(1), addr: 33:33:00:00:00:01, mcstate: f(3), port mask: 7, no super
index 3, raw: 0000001c d0000100 5e000001, type: addr(1), addr: 01:00:5e:00:00:01, mcstate: f(3), port mask: 7, no super
index 4, raw: 00000004 f0000001 297495bf, type: vlan+addr(3), addr: 00:01:29:74:95:bf, vlan: 0, uctype: touched(3), port: 1
index 5, raw: 0000001c d0003333 fff4323a, type: addr(1), addr: 33:33:ff:f4:32:3a, mcstate: f(3), port mask: 7, no super
index 6, raw: 00000004 f0000000 0c07acca, type: vlan+addr(3), addr: 00:00:0c:07:ac:ca, vlan: 0, uctype: touched(3), port: 1
index 7, raw: 00000004 7000e8e0 b75db25e, type: vlan+addr(3), addr: e8:e0:b7:5d:b2:5e, vlan: 0, uctype: untouched(1), port: 1
index 9, raw: 00000004 f0005c26 0a69440b, type: vlan+addr(3), addr: 5c:26:0a:69:44:0b, vlan: 0, uctype: touched(3), port: 1
index 11, raw: 00000004 70005c26 0a5b2ea6, type: vlan+addr(3), addr: 5c:26:0a:5b:2e:a6, vlan: 0, uctype: untouched(1), port: 1
index 12, raw: 00000004 f000d4be d93db6b8, type: vlan+addr(3), addr: d4:be:d9:3d:b6:b8, vlan: 0, uctype: touched(3), port: 1
index 13, raw: 00000004 70000014 225b62d9, type: vlan+addr(3), addr: 00:14:22:5b:62:d9, vlan: 0, uctype: untouched(1), port: 1
index 14, raw: 00000004 7000000b 7866c6d3, type: vlan+addr(3), addr: 00:0b:78:66:c6:d3, vlan: 0, uctype: untouched(1), port: 1
index 15, raw: 00000004 f0005c26 0a6952fa, type: vlan+addr(3), addr: 5c:26:0a:69:52:fa, vlan: 0, uctype: touched(3), port: 1
index 16, raw: 00000004 f000b8ac 6f7d1b65, type: vlan+addr(3), addr: b8:ac:6f:7d:1b:65, vlan: 0, uctype: touched(3), port: 1
index 17, raw: 00000004 7000d4be d9a34760, type: vlan+addr(3), addr: d4:be:d9:a3:47:60, vlan: 0, uctype: untouched(1), port: 1
index 18, raw: 00000004 70000007 eb645149, type: vlan+addr(3), addr: 00:07:eb:64:51:49, vlan: 0, uctype: untouched(1), port: 1
index 19, raw: 00000004 f3200000 0c07acd3, type: vlan+addr(3), addr: 00:00:0c:07:ac:d3, vlan: 800, uctype: touched(3), port: 1
index 20, raw: 00000004 7000d067 e5e7330c, type: vlan+addr(3), addr: d0:67:e5:e7:33:0c, vlan: 0, uctype: untouched(1), port: 1
index 22, raw: 00000004 70000026 b9802a50, type: vlan+addr(3), addr: 00:26:b9:80:2a:50, vlan: 0, uctype: untouched(1), port: 1
index 23, raw: 00000004 f000d067 e5e5aa12, type: vlan+addr(3), addr: d0:67:e5:e5:aa:12, vlan: 0, uctype: touched(3), port: 1
index 24, raw: 00000004 f0000011 430619f6, type: vlan+addr(3), addr: 00:11:43:06:19:f6, vlan: 0, uctype: touched(3), port: 1
index 25, raw: 00000004 7000bc30 5bde7ee2, type: vlan+addr(3), addr: bc:30:5b:de:7e:e2, vlan: 0, uctype: untouched(1), port: 1
index 26, raw: 00000004 7000b8ac 6f92c3d3, type: vlan+addr(3), addr: b8:ac:6f:92:c3:d3, vlan: 0, uctype: untouched(1), port: 1
index 28, raw: 00000004 f0000012 01f7d6ff, type: vlan+addr(3), addr: 00:12:01:f7:d6:ff, vlan: 0, uctype: touched(3), port: 1
index 29, raw: 00000004 f000000b db7789a5, type: vlan+addr(3), addr: 00:0b:db:77:89:a5, vlan: 0, uctype: touched(3), port: 1
index 31, raw: 00000004 70000018 8b2d9433, type: vlan+addr(3), addr: 00:18:8b:2d:94:33, vlan: 0, uctype: untouched(1), port: 1
index 32, raw: 00000004 70000013 728a0dc0, type: vlan+addr(3), addr: 00:13:72:8a:0d:c0, vlan: 0, uctype: untouched(1), port: 1
index 33, raw: 00000004 700000c0 b76f6e82, type: vlan+addr(3), addr: 00:c0:b7:6f:6e:82, vlan: 0, uctype: untouched(1), port: 1
index 34, raw: 00000004 700014da e9096f9a, type: vlan+addr(3), addr: 14:da:e9:09:6f:9a, vlan: 0, uctype: untouched(1), port: 1
index 35, raw: 00000004 f0000023 24086746, type: vlan+addr(3), addr: 00:23:24:08:67:46, vlan: 0, uctype: touched(3), port: 1
index 36, raw: 00000004 7000001b 11b4362f, type: vlan+addr(3), addr: 00:1b:11:b4:36:2f, vlan: 0, uctype: untouched(1), port: 1
[0..36]: 32 entries, +
root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table
```

```
index 37, raw: 00000004 70000019 b9382f7e, type: vlan+addr(3), addr: 00:19:b9:38:2f:7e, vlan: 0, uctype: untouched(1), port: 1

index 38, raw: 00000004 f3200011 93ec6fa2, type: vlan+addr(3), addr: 00:11:93:ec:6f:a2, vlan: 800, uctype: touched(3), port: 1

index 40, raw: 00000004 f0000012 01f7a73f, type: vlan+addr(3), addr: 00:12:01:f7:a7:3f, vlan: 0, uctype: touched(3), port: 1

index 41, raw: 00000004 f0000011 855b1f3c, type: vlan+addr(3), addr: 00:11:85:5b:1f:3c, vlan: 0, uctype: touched(3), port: 1

index 42, raw: 00000004 7000d4be d900d37e, type: vlan+addr(3), addr: d4:be:d9:00:d3:7e, vlan: 0, uctype: untouched(1), port: 1

index 45, raw: 00000004 f3200012 01f7d6ff, type: vlan+addr(3), addr: 00:12:01:f7:d6:ff, vlan: 800, uctype: touched(3), port: 1

index 46, raw: 00000004 f0000002 fcc039df, type: vlan+addr(3), addr: 00:02:fc:c0:39:df, vlan: 0, uctype: touched(3), port: 1

index 47, raw: 00000004 f0000000 0c07ac66, type: vlan+addr(3), addr: 00:00:0c:07:ac:66, vlan: 0, uctype: touched(3), port: 1

index 48, raw: 00000004 f000d4be d94167da, type: vlan+addr(3), addr: d4:be:d9:41:67:da, vlan: 0, uctype: touched(3), port: 1

index 49, raw: 00000004 f000d067 e5e72bc0, type: vlan+addr(3), addr: d0:67:e5:e7:2b:c0, vlan: 0, uctype: touched(3), port: 1

index 50, raw: 00000004 f0005c26 0a6a51d0, type: vlan+addr(3), addr: 5c:26:0a:6a:51:d0, vlan: 0, uctype: touched(3), port: 1

index 51, raw: 00000004 70000014 22266425, type: vlan+addr(3), addr: 00:14:22:26:64:25, vlan: 0, uctype: untouched(1), port: 1

index 53, raw: 00000004 f3200002 fcc039df, type: vlan+addr(3), addr: 00:02:fc:c0:39:df, vlan: 800, uctype: touched(3), port: 1

index 54, raw: 00000004 f000000b cd413d26, type: vlan+addr(3), addr: 00:0b:cd:41:3d:26, vlan: 0, uctype: touched(3), port: 1

index 55, raw: 00000004 f3200000 0c07ac6f, type: vlan+addr(3), addr: 00:00:0c:07:ac:6f, vlan: 800, uctype: touched(3), port: 1

index 56, raw: 00000004 f000000b cd413d27, type: vlan+addr(3), addr: 00:0b:cd:41:3d:27, vlan: 0, uctype: touched(3), port: 1

index 57, raw: 00000004 f000000d 5620cdce, type: vlan+addr(3), addr: 00:0d:56:20:cd:ce, vlan: 0, uctype: touched(3), port: 1

index 58, raw: 00000004 f0000004 e2fceead, type: vlan+addr(3), addr: 00:04:e2:fc:ee:ad, vlan: 0, uctype: touched(3), port: 1

index 59, raw: 00000004 7000d4be d93db91b, type: vlan+addr(3), addr: d4:be:d9:3d:b9:1b, vlan: 0, uctype: untouched(1), port: 1

index 60, raw: 00000004 70000019 b9022455, type: vlan+addr(3), addr: 00:19:b9:02:24:55, vlan: 0, uctype: untouched(1), port: 1

index 61, raw: 00000004 f0000027 1369552b, type: vlan+addr(3), addr: 00:27:13:69:55:2b, vlan: 0, uctype: touched(3), port: 1

index 62, raw: 00000004 70005c26 0a06d1cd, type: vlan+addr(3), addr: 5c:26:0a:06:d1:cd, vlan: 0, uctype: untouched(1), port: 1

index 63, raw: 00000004 7000d4be d96816aa, type: vlan+addr(3), addr: d4:be:d9:68:16:aa, vlan: 0, uctype: untouched(1), port: 1

index 64, raw: 00000004 70000015 f28e329c, type: vlan+addr(3), addr: 00:15:f2:8e:32:9c, vlan: 0, uctype: untouched(1), port: 1

index 66, raw: 00000004 7000d067 e5e53caf, type: vlan+addr(3), addr: d0:67:e5:e5:3c:af, vlan: 0, uctype: untouched(1), port: 1

index 67, raw: 00000004 f000d4be d9416812, type: vlan+addr(3), addr: d4:be:d9:41:68:12, vlan: 0, uctype: touched(3), port: 1

index 69, raw: 00000004 f3200012 01f7a73f, type: vlan+addr(3), addr: 00:12:01:f7:a7:3f, vlan: 800, uctype: touched(3), port: 1

index 75, raw: 00000004 70000014 22266386, type: vlan+addr(3), addr: 00:14:22:26:63:86, vlan: 0, uctype: untouched(1), port: 1

index 80, raw: 00000004 70000030 6e5ee4b4, type: vlan+addr(3), addr: 00:30:6e:5e:e4:b4, vlan: 0, uctype: untouched(1), port: 1

index 83, raw: 00000004 70005c26 0a695379, type: vlan+addr(3), addr: 5c:26:0a:69:53:79, vlan: 0, uctype: untouched(1), port: 1

index 85, raw: 00000004 7000d4be d936b959, type: vlan+addr(3), addr: d4:be:d9:36:b9:59, vlan: 0, uctype: untouched(1), port: 1

index 86, raw: 00000004 7000bc30 5bde7ec2, type: vlan+addr(3), addr: bc:30:5b:de:7e:c2, vlan: 0, uctype: untouched(1), port: 1

[37..86]: 32 entries, +

root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table

index 87, raw: 00000004 7000b8ac 6f7f4712, type: vlan+addr(3), addr: b8:ac:6f:7f:47:12, vlan: 0, uctype: untouched(1), port: 1

index 88, raw: 00000004 f0005c26 0a694420, type: vlan+addr(3), addr: 5c:26:0a:69:44:20, vlan: 0, uctype: touched(3), port: 1

index 89, raw: 00000004 f0000018 8b2d92e2, type: vlan+addr(3), addr: 00:18:8b:2d:92:e2, vlan: 0, uctype: touched(3), port: 1

index 93, raw: 00000004 7000001a a0a0c9df, type: vlan+addr(3), addr: 00:1a:a0:a0:c9:df, vlan: 0, uctype: untouched(1), port: 1

index 94, raw: 00000004 f000e8e0 b736b25e, type: vlan+addr(3), addr: e8:e0:b7:36:b2:5e, vlan: 0, uctype: touched(3), port: 1

index 96, raw: 00000004 70000010 18af5bfb, type: vlan+addr(3), addr: 00:10:18:af:5b:fb, vlan: 0, uctype: untouched(1), port: 1

index 99, raw: 00000004 70003085 a9a63965, type: vlan+addr(3), addr: 30:85:a9:a6:39:65, vlan: 0, uctype: untouched(1), port: 1

index 101, raw: 00000004 70005c26 0a695312, type: vlan+addr(3), addr: 5c:26:0a:69:53:12, vlan: 0, uctype: untouched(1), port: 1

index 104, raw: 00000004 7000f46d 04e22fc9, type: vlan+addr(3), addr: f4:6d:04:e2:2f:c9, vlan: 0, uctype: untouched(1), port: 1

index 105, raw: 00000004 7000001b 788de114, type: vlan+addr(3), addr: 00:1b:78:8d:e1:14, vlan: 0, uctype: untouched(1), port: 1

index 109, raw: 00000004 7000d4be d96816f4, type: vlan+addr(3), addr: d4:be:d9:68:16:f4, vlan: 0, uctype: untouched(1), port: 1

index 111, raw: 00000004 f0000010 18a113b5, type: vlan+addr(3), addr: 00:10:18:a1:13:b5, vlan: 0, uctype: touched(3), port: 1

index 115, raw: 00000004 f000f46d 04e22fbd, type: vlan+addr(3), addr: f4:6d:04:e2:2f:bd, vlan: 0, uctype: touched(3), port: 1
index 116, raw: 00000004 7000b8ac 6f8ed5e6, type: vlan+addr(3), addr: b8:ac:6f:8e:d5:e6, vlan: 0, uctype: untouched(1), port: 1

index 118, raw: 00000004 7000001a a0b2ebee, type: vlan+addr(3), addr: 00:1a:a0:b2:eb:ee, vlan: 0, uctype: untouched(1), port: 1

index 119, raw: 00000004 7000782b cbab87d4, type: vlan+addr(3), addr: 78:2b:cb:ab:87:d4, vlan: 0, uctype: untouched(1), port: 1

index 126, raw: 00000004 70000018 8b09703d, type: vlan+addr(3), addr: 00:18:8b:09:70:3d, vlan: 0, uctype: untouched(1), port: 1

index 129, raw: 00000004 70000050 b65f189e, type: vlan+addr(3), addr: 00:50:b6:5f:18:9e, vlan: 0, uctype: untouched(1), port: 1

index 131, raw: 00000004 f000bc30 5bd07ed1, type: vlan+addr(3), addr: bc:30:5b:d0:7e:d1, vlan: 0, uctype: touched(3), port: 1

index 133, raw: 00000004 f0003085 a9a26425, type: vlan+addr(3), addr: 30:85:a9:a2:64:25, vlan: 0, uctype: touched(3), port: 1

index 147, raw: 00000004 f000b8ac 6f8bae7f, type: vlan+addr(3), addr: b8:ac:6f:8b:ae:7f, vlan: 0, uctype: touched(3), port: 1

index 175, raw: 00000004 700090e2 ba02c6e4, type: vlan+addr(3), addr: 90:e2:ba:02:c6:e4, vlan: 0, uctype: untouched(1), port: 1

index 186, raw: 00000004 70000013 728c27fd, type: vlan+addr(3), addr: 00:13:72:8c:27:fd, vlan: 0, uctype: untouched(1), port: 1

index 197, raw: 00000004 f0000012 3f716cb1, type: vlan+addr(3), addr: 00:12:3f:71:6c:b1, vlan: 0, uctype: touched(3), port: 1

index 249, raw: 00000004 7000e89d 877c862f, type: vlan+addr(3), addr: e8:9d:87:7c:86:2f, vlan: 0, uctype: untouched(1), port: 1

[87..1023]: 25 entries

root@keystone-evm:~#

root@keystone-evm:~# cat /sys/class/net/eth0/device/ale_table_raw

0: 1c d000ffff ffffffff

1: 00 10000017 eaf4323a

2: 1c d0003333 00000001

3: 1c d0000100 5e000001

4: 04 f0000001 297495bf

5: 1c d0003333 fff4323a

6: 04 f0000000 0c07acca

7: 04 7000e8e0 b75db25e

9: 04 f0005c26 0a69440b

11: 04 70005c26 0a5b2ea6

12: 04 f000d4be d93db6b8

13: 04 f0000014 225b62d9

14: 04 7000000b 7866c6d3

15: 04 f0005c26 0a6952fa

16: 04 f000b8ac 6f7d1b65

17: 04 7000d4be d9a34760

18: 04 70000007 eb645149

19: 04 f3200000 0c07acd3

20: 04 7000d067 e5e7330c

22: 04 70000026 b9802a50

23: 04 f000d067 e5e5aa12

24: 04 f0000011 430619f6

25: 04 f000bc30 5bde7ee2

26: 04 f000b8ac 6f92c3d3

28: 04 f0000012 01f7d6ff

29: 04 f000000b db7789a5

31: 04 70000018 8b2d9433

32: 04 70000013 728a0dc0

33: 04 700000c0 b76f6e82

34: 04 700014da e9096f9a

35: 04 f0000023 24086746

36: 04 7000001b 11b4362f
```

```
37: 04 f0000019 b9382f7e

38: 04 f3200011 93ec6fa2

39: 04 f0005046 5d74bf90

40: 04 f0000012 01f7a73f

41: 04 f0000011 855b1f3c

42: 04 f000d4be d900d37e

45: 04 f3200012 01f7d6ff

46: 04 f0000002 fcc039df

47: 04 f0000000 0c07ac66

48: 04 f000d4be d94167da

49: 04 f000d067 e5e72bc0

50: 04 f0005c26 0a6a51d0

51: 04 70000014 22266425

53: 04 f3200002 fcc039df

54: 04 f000000b cd413d26

55: 04 f3200000 0c07ac6f

56: 04 f000000b cd413d27

57: 04 f000000d 5620cdce

58: 04 f0000004 e2fceead

59: 04 7000d4be d93db91b

60: 04 70000019 b9022455

61: 04 f0000027 1369552b

62: 04 70005c26 0a06d1cd

63: 04 7000d4be d96816aa

64: 04 70000015 f28e329c

66: 04 7000d067 e5e53caf

67: 04 f000d4be d9416812

69: 04 f3200012 01f7a73f

75: 04 70000014 22266386

80: 04 70000030 6e5ee4b4

83: 04 70005c26 0a695379

85: 04 7000d4be d936b959

86: 04 7000bc30 5bde7ec2

87: 04 7000b8ac 6f7f4712

88: 04 f0005c26 0a694420

89: 04 f0000018 8b2d92e2

93: 04 7000001a a0a0c9df

94: 04 f000e8e0 b736b25e

96: 04 70000010 18af5bfb

99: 04 f0003085 a9a63965

101: 04 70005c26 0a695312

104: 04 7000f46d 04e22fc9

105: 04 7000001b 788de114

109: 04 7000d4be d96816f4

111: 04 f0000010 18a113b5

115: 04 f000f46d 04e22fbd

116: 04 7000b8ac 6f8ed5e6
```

```
118: 04 7000001a a0b2ebee
119: 04 7000782b cbab87d4
126: 04 70000018 8b09703d
129: 04 f0000050 b65f189e
131: 04 f000bc30 5bd07ed1
133: 04 f0003085 a9a26425
147: 04 f000b8ac 6f8bae7f
175: 04 700090e2 ba02c6e4
181: 04 f0000012 3f99c9dc
182: 04 f000000c f1d2df6b
186: 04 70000013 728c27fd
197: 04 f0000012 3f716cb1
249: 04 7000e89d 877c862f
[0..1023]: 92 entries
root@keystone-evm:~#
root@keystone-evm:~# ls -l /sys/devices/soc.0/2090000.netcp/cpsw/stats
--w-------    1 root     root          4096 Feb 28 20:32 A
--w-------    1 root     root          4096 Feb 28 20:33 B
root@keystone-evm:~#
root@keystone-evm:~#
root@keystone-evm:~# ethtool -S eth0
NIC statistics:
     CPSW_A:rx_good_frames: 133
     CPSW_A:rx_broadcast_frames: 0
     CPSW_A:rx_multicast_frames: 0
     CPSW_A:rx_pause_frames: 0
     CPSW_A:rx_crc_errors: 0
     CPSW_A:rx_align_code_errors: 0
     CPSW_A:rx_oversized_frames: 0
     CPSW_A:rx_jabber_frames: 0
     CPSW_A:rx_undersized_frames: 0
     CPSW_A:rx_fragments: 0
     CPSW_A:rx_bytes: 20878
     CPSW_A:tx_good_frames: 1469
     CPSW_A:tx_broadcast_frames: 553
     CPSW_A:tx_multicast_frames: 805
     CPSW_A:tx_pause_frames: 0
     CPSW_A:tx_deferred_frames: 0
     CPSW_A:tx_collision_frames: 0
     CPSW_A:tx_single_coll_frames: 0
     CPSW_A:tx_mult_coll_frames: 0
     CPSW_A:tx_excessive_collisions: 0
     CPSW_A:tx_late_collisions: 0
     CPSW_A:tx_underrun: 0
     CPSW_A:tx_carrier_sense_errors: 0
     CPSW_A:tx_bytes: 148064
     CPSW_A:tx_64byte_frames: 335
     CPSW_A:tx_65_to_127byte_frames: 834
     CPSW_A:tx_128_to_255byte_frames: 384
     CPSW_A:tx_256_to_511byte_frames: 45
     CPSW_A:tx_512_to_1023byte_frames: 4
     CPSW_A:tx_1024byte_frames: 0
     CPSW_A:net_bytes: 168942
     CPSW_A:rx_sof_overruns: 0
     CPSW_A:rx_mof_overruns: 0
     CPSW_A:rx_dma_overruns: 0
     CPSW_B:rx_good_frames: 923
     CPSW_B:rx_broadcast_frames: 355
     CPSW_B:rx_multicast_frames: 499
     CPSW_B:rx_pause_frames: 0
     CPSW_B:rx_crc_errors: 0
     CPSW_B:rx_align_code_errors: 0
     CPSW_B:rx_oversized_frames: 0
     CPSW_B:rx_jabber_frames: 0
     CPSW_B:rx_undersized_frames: 0
     CPSW_B:rx_fragments: 0
     CPSW_B:rx_bytes: 95208
     CPSW_B:tx_good_frames: 77
     CPSW_B:tx_broadcast_frames: 0
     CPSW_B:tx_multicast_frames: 0
     CPSW_B:tx_pause_frames: 0
     CPSW_B:tx_deferred_frames: 0
     CPSW_B:tx_collision_frames: 0
     CPSW_B:tx_single_coll_frames: 0
     CPSW_B:tx_mult_coll_frames: 0
     CPSW_B:tx_excessive_collisions: 0
     CPSW_B:tx_late_collisions: 0
     CPSW_B:tx_underrun: 0
     CPSW_B:tx_carrier_sense_errors: 0
     CPSW_B:tx_bytes: 12654
     CPSW_B:tx_64byte_frames: 201
     CPSW_B:tx_65_to_127byte_frames: 527
     CPSW_B:tx_128_to_255byte_frames: 236
     CPSW_B:tx_256_to_511byte_frames: 33
     CPSW_B:tx_512_to_1023byte_frames: 4
     CPSW_B:tx_1024byte_frames: 0
     CPSW_B:net_bytes: 107862
     CPSW_B:rx_sof_overruns: 0
     CPSW_B:rx_mof_overruns: 0
     CPSW_B:rx_dma_overruns: 0
root@keystone-evm:~# echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/A
root@keystone-evm:~#
root@keystone-evm:~# echo 0 > /sys/devices/soc.0/2090000.netcp/cpsw/stats/B
root@keystone-evm:~#
```

```
root@keystone-evm:~# ethtool -S eth0

NIC statistics:

     CPSW_A:rx_good_frames: 5

     CPSW_A:rx_broadcast_frames: 0

     CPSW_A:rx_multicast_frames: 0

     CPSW_A:rx_pause_frames: 0

     CPSW_A:rx_crc_errors: 0

     CPSW_A:rx_align_code_errors: 0

     CPSW_A:rx_oversized_frames: 0

     CPSW_A:rx_jabber_frames: 0

     CPSW_A:rx_undersized_frames: 0

     CPSW_A:rx_fragments: 0

     CPSW_A:rx_bytes: 766

     CPSW_A:tx_good_frames: 5

     CPSW_A:tx_broadcast_frames: 0

     CPSW_A:tx_multicast_frames: 0

     CPSW_A:tx_pause_frames: 0

     CPSW_A:tx_deferred_frames: 0

     CPSW_A:tx_collision_frames: 0

     CPSW_A:tx_single_coll_frames: 0

     CPSW_A:tx_mult_coll_frames: 0

     CPSW_A:tx_excessive_collisions: 0

     CPSW_A:tx_late_collisions: 0

     CPSW_A:tx_underrun: 0

     CPSW_A:tx_carrier_sense_errors: 0

     CPSW_A:tx_bytes: 814

     CPSW_A:tx_64byte_frames: 0

     CPSW_A:tx_65_to_127byte_frames: 2

     CPSW_A:tx_128_to_255byte_frames: 8

     CPSW_A:tx_256_to_511byte_frames: 0

     CPSW_A:tx_512_to_1023byte_frames: 0

     CPSW_A:tx_1024byte_frames: 0

     CPSW_A:net_bytes: 1580

     CPSW_A:rx_sof_overruns: 0

     CPSW_A:rx_mof_overruns: 0

     CPSW_A:rx_dma_overruns: 0

     CPSW_B:rx_good_frames: 4

     CPSW_B:rx_broadcast_frames: 0

     CPSW_B:rx_multicast_frames: 1

     CPSW_B:rx_pause_frames: 0

     CPSW_B:rx_crc_errors: 0

     CPSW_B:rx_align_code_errors: 0

     CPSW_B:rx_oversized_frames: 0

     CPSW_B:rx_jabber_frames: 0

     CPSW_B:rx_undersized_frames: 0

     CPSW_B:rx_fragments: 0

     CPSW_B:rx_bytes: 814
     CPSW_B:tx_good_frames: 5

     CPSW_B:tx_broadcast_frames: 0

     CPSW_B:tx_multicast_frames: 0

     CPSW_B:tx_pause_frames: 0

     CPSW_B:tx_deferred_frames: 0

     CPSW_B:tx_collision_frames: 0

     CPSW_B:tx_single_coll_frames: 0

     CPSW_B:tx_mult_coll_frames: 0

     CPSW_B:tx_excessive_collisions: 0

     CPSW_B:tx_late_collisions: 0

     CPSW_B:tx_underrun: 0

     CPSW_B:tx_carrier_sense_errors: 0

     CPSW_B:tx_bytes: 766

     CPSW_B:tx_64byte_frames: 0

     CPSW_B:tx_65_to_127byte_frames: 2

     CPSW_B:tx_128_to_255byte_frames: 8

     CPSW_B:tx_256_to_511byte_frames: 0

     CPSW_B:tx_512_to_1023byte_frames: 0

     CPSW_B:tx_1024byte_frames: 0

     CPSW_B:net_bytes: 1580

     CPSW_B:rx_sof_overruns: 0

     CPSW_B:rx_mof_overruns: 0

     CPSW_B:rx_dma_overruns: 0

root@keystone-evm:~#
```

Using Ethtool to Change PHY Settings

Use the Linux "ethtool -s" command to change PHY settings of selected parameters as shown below

```
        ethtool -s|--change DEVNAME     Change generic options
                [ speed %d ]
                [ duplex half|full ]
                [ port tp|aui|bnc|mii|fibre ]
                [ autoneg on|off ]
                [ advertise %x ]
```

Enabling MDIO

In the default K2HK DTS, currently MDIO is disabled as a software workaround for the MDIO hardware issue on K2HK EVM. To enable MDIO driver on custom board, add the following bindings in the K2HK DTS for 1G. For u-boot has_mdio env variable should be set as well to enable MDIO hardware IP.

For K2HK EVM:

```
                mdio: mdio@2090300 {
                        compatible      = "ti,davinci_mdio";
                        #address-cells = <1>;
                        #size-cells = <0>;
                        reg             = <0x2090300 0x100>;
                        bus_freq        = <1000000>;
                        clocks          = <&clkcpgmac>;
                        clock-names     = "fck";
                        phy0: phy@0 {
                                compatible = "marvell,88e1111";
                                reg = <0>;
                        };
                        phy1: phy@1 {
                                compatible = "marvell,88e1111";
                                reg = <1>;
                        };
                };
```

Also modify the slaveX nodes in the netcp node to the following:

```
        netcp {
                ...
                slaves {
                        slave0 {
                                label           = "slave0";
                                link-interface  = <1>;
                                phy-handle      = <&phy0>;
                        };
                        slave1 {
                                label           = "slave1";
                                link-interface  = <1>;
                                phy-handle      = <&phy1>;
                        };
                };
        };
```

Starting from MCSDK release 3.1.1, K2E and K2L MDIO are enabled by default. To enable MDIO on these platforms in previous MCSDK releases, see the k2e-net.dtsi or k2l-net.dtsi DTS file in MCSDK release 3.1.1.

Note: Enabling MDIO will have impact on the system bootup time due to the time spent in auto-negotiation on each ethernet port.

Common Platform Time Sync (CPTS)

The Common Platform Time Sync (CPTS) module is used to facilitate host control of time sync operations. It enables compliance with the IEEE 1588-2008 standard for a precision clock synchronization protocol.

Although CPTS timestamping co-exists with PA timestamping, CPTS timestamping is only for PTP packets and in that case, PA will not timestamp those packets.

CPTS Hardware Configurations

1. CPTS Device Tree Bindings Following are the CPTS related device tree bindings

- cpts_reg_ofs

  cpts register offset in cpsw module

- cpts_rftclk_sel

  chooses the input rftclk, default is 0

- cpts_rftclk_freq

ref clock frequency in Hz if it is an **external** clock

- cpsw_cpts_rft_clk

  ref clock name if it is an **internal** clock

- cpts_ts_comp_length

  PPS Asserted Length (in Ref Clk Cycles)

- cpts_ts_comp_polarity

  if 1, PPS is assered high; otherwise asserted low

- cpts_clock_mult, cpts_clock_shift, cpts_clock_div

  multiplier and divider for converting cpts counter value to timestamp time

```
Example:
```

```
   netcp: netcp@2090000 {
      ...
      clocks = <&papllclk>, <&clkcpgmac>, <&chipclk12>;
      clock-names = "clk_pa", "clk_cpgmac", "cpsw_cpts_rft_clk";
      ...
      cpsw: cpsw@2090000 {
      ...
         cpts_reg_ofs = <0xd00>;
         ...
         cpts_rftclk_sel=<8>;
         /*cpts_rftclk_freq = <122800000>;*/
         cpts_ts_comp_length = <3>;
         cpts_ts_comp_polarity = <1>;  /* 1 - assert high */
         /* cpts_clock_mult = <6250>; */
         /* cpts_clock_shift = <8>; */
         /* cpts_clock_div = <3>; */
         ...
      };
      ...
   };
```

2. Configurations during driver initialization

By default, cpts is configured with the following configurations at boot up:

- Tx and Rx Annex D support but only one vlan tag (ts_vlan_ltype1_en)
- Tx and Rx Annex E support but only one vlan tag (ts_vlan_ltype1_en)
- Tx and Rx Annex F support but only one vlan tag (ts_vlan_ltype1_en)
- ts_vlan_ltype1 = 0x8100 (default)
- uni-cast enabled
- ttl_nonzero enabled

3. Configurations during runtime (Sysfs)

Currently the following sysfs are available for cpts related runtime configuration

- /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/n/uni_en

  (where n is slave port number)

  - Read/Write

  - 1 (enable unicast)
  - 0 (disable unicast)
- /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/n/mcast_addr

  (where n is slave port number)

  - Read/Write
  - bit map for mcast addr .132 .131 .130 .129 .107
    - bit[4]: 224.0.1.132
    - bit[3]: 224.0.1.131
    - bit[2]: 224.0.1.130
    - bit[1]: 224.0.1.129
    - bit[0]: 224.0.0.107
- /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/n/config

  (where n is slave port number)

  - Read Only
  - shows the raw values of the cpsw port ts register configurations

```
Examples:
```

```
1. Checking whether uni-cast enabled
   $ cat /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/uni_en
   $ 0
```

```
2. Enabling uni-cast
   $ echo 1 > /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/uni_en
```

```
3. Checking which multi-cast addr is enabled (when uni_en=0)
   $ cat /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/mcast_addr
   $ 0x1f
```

```
4. Disabling 224.0.1.131 and 224.0.0.107 but enabling the rest (when uni_en=0)
   $ echo 0x16 > /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/mcast_addr
```

```
5. Showing the current port time sync config
   $ cat /sys/devices/soc.0/2090000.netcp/cpsw/port_ts/1/config
   000f06bb 001e88f7 81008100 01a088f7 00040000
```

```
   where the displayed hex values correspond to the port registers
   ts_ctl, ts_seq_ltype, ts_vlan_ltype, ts_ctl_ltype2 and ts_ctl2
```

Note 1: Although the above configurations are done through command line, they can also be done by using standard Linux open()/read()/write() file function calls.

Note 2: When uni-cast is enabled, ie. uni_en=1, mcast_addr configuration will not take effect since uni-cast will allow any uni-cast and multi-cast address.

CPTS Driver Internals Overview

1. Driver Initialization

On start up, the cpts driver

- initializes the input clock if it is an internal clock:
  - enable the input clock
  - get the clock frequency
- gets the frequency configuration of the input clock from the device tree bindings if it is an external clock
- selects/calculates (see Notes below for details) the multiplier (M), shift (S) and divisor (D) corresponding to the frequency for internal usage, ie. converting counter cycles to nsec by using the formula

  nsec = ((cycles * M) >> S) / D
- gets the cpts_rftclk_sel value and program the CPTS RFTCLK_SEL register.
- configures the cpsw Px_TS_CTL, Px_TS_SEQ_LTYPE, Px_TS_VLAN_LTYPE, Px_TS_CTL_LTYPE2 and Px_TS_CTL2 registers (see section Configurations)
- registers itself to the Linux kernel ptp layer as a clock source (doing so makes sure the Linux kernel ptp layer and standard user space API's can be used)
- mark the currnet cpts counter value to the current system time
- schedule a periodic work to catch the cpts counter overflow events and updates the driver's internal time counter and cycle counter values accordingly.

Note 1: For a rftclk freq of 400MHz, the counter overflows at about every 10.73 secs. It is the responsibility of the software (ie. the driver) to keep track of the overflows and hence the correct time passed.

Note 2: The multiplier (M) shift (S) and divisor (D) depends on the rftclk frequency (F). Ideally, "good" values of M/S/D should be chosen so that when converting counter value when it reaches the rftclk frequency value (F) to timestamp time, i.e.

((F * M) >> S) / D

gives exactly 1000000000 nsec for accuracy and D should be 1 (if possible) to avoid long division for efficiency.

For example, if F = 614400000, to find M/S/D such that

1000000000 = 614400000 * M / (2^S * D)

simplify and rewrite both sides so that

2^4 * 5^4 = 2^11 * 3 * M / (2^S * D)

or

M / (2^S * D) = 5000 / (2^10 * 3)

hence

M = 5000, S = 10, D = 3

Note 3: cpts driver keeps a table of M/S/D for some common frequencies

Freq (Hz)	M	S	D
400000000	2560	10	1
425000000	5120	7	17
500000000	2048	10	1
600000000	5120	10	3
614400000	5000	10	3
625000000	4096	9	5
675000000	5120	7	27
700000000	5120	9	7
750000000	4096	10	3

Note 4: At start up, cpts driver selects or calculates the M/S/D for the rftclk frequency according to the following

a. if M/S/D is defined in devicetree bindings, use them; otherwise

b. if the rftclk frequency matches one of the frequencies in the table above, select the corresponding M/S/D; otherwise

c. if the rftclk frequency differs from one of the frequencies in the table above by less than 1 MHz, select the M/S/D that corresponds to the frequency with the minimum difference; otherwise

d. call clocks_calc_mult_shift( ) to calculate the M & S and set D = 1

Note 5: (**WARNING**) On Keystone 2 platforms, the default rftclk select is the internal SYSCLK2. On K2L, core pll is configured (based on the programmed efuse of max speed of 1 GHz and ref clk of 122880000 Hz) to 1000594244 Hz. As such, SYSCLK2 = 1000594244 / 2 = 500297122 Hz. With such a rftclk frequency, it is unlikely that some "good" M/S/D can be found so that 1000000000 = ((500297122 * M) >> S) / D. Hence based on the algorithm in Note 4, the M/S/D corresponding to 500000000 Hz will be used and unfortunately inaccuracy will be observed in timestamping. However, this issue is not observed on K2HK and K2E since the respective core pll is configured to exactly 1200000000 Hz and 1000000000 Hz, thus the cpts rftclk frequency is 600000000 and 500000000 Hz respectively and "good" M/S/D exist for these rftclk frequencies.

Note 6: Instead of an internal rftclk, cpts can be provided with an external rftclk. Also custom M/S/D can be configured in devicetree bindings.

2. Timestamping in Tx

In the tx direction during runtime, the driver

- marks the submitted packet to be CPTS timestamped if the the packet passes the PTP filter rules
- retrieves the timestamp on the transmitted ptp packet (packets submitted to a socket with proper socket configurations, see below) from CPTS's event FIFO
- converts the counter value to nsec (recall the internal time counter and the cycle counter kept internally by the driver)
- packs the retrieved timestamp with a clone of the transmitted packet in a buffer
- returns the buffer to the app which submits the packet for transmission through the socket's error queue

3. Timestamping in Rx

In the rx direction during runtime, the driver

- examines the received packet to see if it matches the PTP filter requirements
- if it does, then it retrieves the timestamp on the received ptp packet from the CPTS's event FIFO
- coverts the counter value to nsec (recall the internal time counter and the cycle counter kept internally by the driver)

- packs the retrieved timestamp with received packet in a buffer
- pass the packet buffer onwards

Using CPTS Timestamping

CPTS user applications use standard Linux APIs to send and receive PTP packets, and to adjust CPTS clock.

1. Send/receive L4 PTP messages (Annex D and E)

User application sends and receives L4 PTP messages by calling Linux standard socket API functions

```
  Example (see Reference i):
```

```
a. open UDP socket
     b. call ioctl(sock, SIOCHWTSTAMP, ...) to set the hw timestamping
        socket config
     c. bind to PTP event port
     d. set dst address to socket
     d. setsockopt to join multicast group (if using multicast)
     f. setsockopt to set socket option SO_TIMESTAMP
     g. sendto to send PTP packets
     h. recvmsg( ... MSG_ERRQUEUE ...) to receive timestamped packets
```

2. Send/receive L2 PTP messages (Annex F)

User application sends and receives PTP messages over Ethernet by opening Linux RAW sockets.

```
  Example (see file raw.c in Reference iii):
```

```
int fd
     fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
     ...
```

In this case, PTP messages are encapsulated directly in Ethernet frames with EtherType 0x88f7.

3. Send/receive PTP messages in VLAN

When sending L2/L4 PTP messages over VLAN, **step b** in above example need to be applied to the actual interface instead of the VLAN interface.

```
  Example (see Reference i):
```

```
  Suppose a VLAN interface with vid=10 is added to the eth0 interface.
```

```
  $ vconfig add eth0 10
  $ ifconfig eth0.10 192.168.1.200
  $ ifconfig
  eth0      Link encap:Ethernet  HWaddr 00:17:EA:F4:32:3A
            inet addr:132.168.138.88  Bcast:0.0.0.0  Mask:255.255.254.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:647798 errors:0 dropped:158648 overruns:0 frame:0
            TX packets:1678 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:58765374 (56.0 MiB)  TX bytes:84321 (82.3 KiB)
```

```
  eth0.10   Link encap:Ethernet  HWaddr 00:17:EA:F4:32:3A
            inet addr:192.168.1.200  Bcast:192.168.1.255  Mask:255.255.255.0
            inet6 addr: fe80::217:eaff:fef4:323a/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:6 errors:0 dropped:0 overruns:0 frame:0
            TX packets:61 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:836 (836.0 B)  TX bytes:6270 (6.1 KiB)
```

```
  To enable hw timestamping on the eth0.10 interface, the ioctl(sock, SIOCHWTSTAMP, ...)
  function call needs to be on the actual interface eth0:
```

```
     int sock;
     struct ifreq hwtstamp;
     struct hwtstamp_config hwconfig;
```

```
     ...
```

```
     sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

```
/* enable hw timestamping for interfaces eth0 or eth0.10 */
     strncpy(hwtstamp.ifr_name, "eth0", sizeof(hwtstamp.ifr_name));
     hwtstamp.ifr_data = (void *)&hwconfig;
     memset(&hwconfig, 0, sizeof(hwconfig));
     hwconfig.tx_type = HWTSTAMP_TX_ON
     hwconfig.rx_filter = HWTSTAMP_FILTER_PTP_V1_L4_SYNC
     ioctl(sock, SIOCSHWTSTAMP, &hwtstamp);
     ...
```

4. Clock Adjustments

User application needs to inform the CPTS driver of any time or reference clock frequency adjustments, for example, as a result of running PTP protocol.

- It's the application's responsibility to modify the (physical) rftclk frequency.
- However, the frequency change needs to be sent to the cpts driver by calling the standard Linux API clock_adjtime() with a flag ADJ_FREQUENCY. This is needed so that the CPTS driver can calculate the time correctly.
- As indicated above, CPTS driver keeps a pair of numbers, the multiplier and divisor, to represent the reference clock frequency. When the frequency change API is called and passed with the ppb change, the CPTS driver updates its internal multiplier as follows:

  new_mult = init_mult + init_mult * (ppb / 1000000000)

  Note: the ppb change is always applied to the initial orginal frequency, NOT the current frequency.

```
  Example (see Reference ii):
```

```
     struct timex tx;
     ...
     fd = open("/dev/ptp0", O_RDWR);
     clkid = get_clockid(fd);
     ...
     memset(&tx, 0, sizeof(tx));
     tx.modes = ADJ_FREQUENCY;
     tx.freq = ppb_to_scaled_ppm(adjfreq);
```

```
   if (clock_adjtime(clkid, &tx)) {
         perror("clock_adjtime");
   } else {
         puts("frequency adjustment okay");
   }
```

- To set time (due to shifting +/-), call the the standard Linux API clock_adjtime() with a flag ADJ_SETOFFSET

```
Example (see Reference ii):
```

```
   memset(&tx, 0, sizeof(tx));
   tx.modes = ADJ_SETOFFSET;
   tx.time.tv_sec = adjtime;
   tx.time.tv_usec = 0;
   if (clock_adjtime(clkid, &tx) < 0) {
         perror("clock_adjtime");
   } else {
         puts("time shift okay");
   }
```

- To get time, call the the standard Linux API clock_gettime()

```
Example (see Reference ii):
```

```
   if (clock_gettime(clkid, &ts)) {
         perror("clock_gettime");
   } else {
         printf("clock time: %ld.%09ld or %s",
                ts.tv_sec, ts.tv_nsec, ctime(&ts.tv_sec));
   }
```

- To set time, call the the standard Linux API clock_settime()

```
Example (see Reference ii):
```

```
   clock_gettime(CLOCK_REALTIME, &ts);
   if (clock_settime(clkid, &ts)) {
         perror("clock_settime");
   } else {
         puts("set time okay");
   }
```

Testing CPTS/PTP

To check the ptp clock adjustment with PTP protocol, a PTP slave (client) and a PTP master (server) applications are needed to run on separate devices (EVM or PC). Open source application package linuxptp (Reference iii) can be used as slave and as well as master. Another option for PTP master is the open source project ptpd (Reference iv).

- Slave Side Examples

The following command can be used to run a ptp-over-L4 client on the evm in slave mode

```
./ptp4l -E -4 -H -i eth0 -s -l 7 -m -q -p /dev/ptp0
```

For ptp-over-L2 client, use the command

```
./ptp4l -E -2 -H -i eth0 -s -l 7 -m -q -p /dev/ptp0
```

ptp4l runtime configuartions can be applied by saving desired configurations in a configuration file and start the ptp4l with an argument "-f <config_filename>" Note: Only ptp4l supports L2 ethernet, ptpd2 does not support L2. For example, put the following two lines

```
[global]
tx_timestamp_timeout  15
```

in a file named config, and start a ptp4l-over-L2 client with command

```
./ptp4l -E -2 -H -i eth0 -s -l 7 -m -q -p /dev/ptp0 -f config
```

the tx poll timeout interval will be set to 15 msec instead of the default 1 msec.

The adjusted time can be checked by cross compiling the testptp application from the linux kernel: Documentation/ptp/testptp.c. (e.g) ./testptp -g

- Master Side Examples

ptp4l can also be run in master mode. For example, the following command starts a ptp4l-over-L2 master on an EVM using **hardware timestamping**,

```
./ptp4l -E -2 -H -i eth0 -l 7 -m -q -p /dev/ptp0 -f config
```

On a Linux PC which does not supoort hardware timestamping, the following command starts a ptp4l-over-L2 master using **software timestamping**.

```
./ptp4l -E -2 -S -i eth0 -l 7 -m -q -p -f config
```

Who Is Timestamping What?

Notice that PA timestamping and CPTS timestamping are running simultaneously. This is desirable in some use cases because, for example, NTP timestamping is also needed in some systems and CPTS timestamping is only for PTP. However, CPTS has priority over PA to timestamp PTP messages. When CPTS timestamps a PTP message, PA will not timestamp it. See the section PA Timestamping for more details about PA timestamping.

If needed, PA timestamping can be completely disabled by adding force_no_hwtstamp to the device tree.

```
  Example:
```

```
     pa: pa@2000000 {
             label = "keystone-pa";
             ...
             force_no_hwtstamp;
     };
```

CPTS timestamping can be completely disabled by removing the following line from the device tree

```
cpts_reg_ofs = <0xd00>;
```

Pulse-Per-Second (PPS)

The CPTS driver uses the timestamp compare (TS_COMP) output to support PPS.

The TS_COMP output is asserted for ts_comp_length[15:0] RCLK periods when the time_stamp value compares with the ts_comp_val[31:0] and the length value is non-zero. The TS_COMP rising edge occurs three RCLK periods after the values compare. A timestamp compare event is pushed into the event FIFO when TS_COMP is asserted. The polarity of the TS_COMP output is determined by the ts_polarity bit. The output is asserted low when the polarity bit is low.

1. CPTS Driver PPS Initialization

- The driver enables its pps support capability when it registers itself to the Linux PTP layer.
- Upon getting the pps support information from CPTS driver, the Linux PTP layer registers CPTS as a pps source with the Linux PPS layer. Doing so allows user applications to manage the PPS source by using Linux standard API.

2. CPTS Driver PPS Operation

- Upon CPTS pps being enabled by user application, the driver programs the TS_COMP_VAL for a pulse to be generated at the next (absolute) 1 second boundary. The TS_COMP_VAL to be programmed is calculated based on the reference clock frequency.
- Driver polls the CPTS event FIFO 5 times a second to retrieve the timestamp compare event of an asserted TS_COMP output signal.
- The driver reloads the TS_COMP_VAL register with a value equivalent to one second from the timestamp value of the retrieved event.
- The event is also reported to the Linux PTP layer which in turn reports to the PPS layer.

3. PPS User Application

- Enabling CPTS PPS by using standard Linux ioctl PTP_ENABLE_PPS

```
Example (Reference ii: Documentation/ptp/testptp.c):
```

```
   fd = open("/dev/ptp0", O_RDWR);
   ...
```

```
   if (ioctl(fd, PTP_ENABLE_PPS, 1))
        perror("PTP_ENABLE_PPS");
   else
        puts("pps for system time enable okay");
```

```
   if (ioctl(fd, PTP_ENABLE_PPS, 0))
        perror("PTP_ENABLE_PPS");
   else
        puts("pps for system time disable okay");
```

- Reading PPS last timstamp by using standard Linux ioctl PPS_FETCH

```
Example (Reference iii: linuxptp-1.2/phc2sys.c)
```

```
   ...
   struct pps_fdata pfd;
```

```
    pfd.timeout.sec = 10;
    pfd.timeout.nsec = 0;
    pfd.timeout.flags = ~PPS_TIME_INVALID;
    if (ioctl(fd, PPS_FETCH, &pfd)) {
          pr_err("failed to fetch PPS: %m");
          return 0;
    }
```

```
    ...
```

- Enabling PPS from sysfs
  - The Linux PTP layer provides a sysfs for enabling/disabling PPS.

```
$ cat /sys/devices/soc.0/2090000.netcp/ptp/ptp0/pps_available
1
$ echo 1 > /sys/devices/soc.0/2090000.netcp/ptp/ptp0/pps_enable
```

- Sysfs Provided by Linux PPS Layer (see Reference v for more details)
  - The Linux PPS layer implements a new class in the sysfs for supporting PPS.

```
$ ls /sys/class/pps/
pps0/
$
$ ls /sys/class/pps/pps0/
assert      clear  echo  mode  name  path  subsystem@  uevent
```

  - Inside each "assert" you can find the timestamp and a sequence number:

```
$ cat /sys/class/pps/pps0/assert
1170026870.983207967#8
```

```
where before the "#" is the timestamp in seconds; after it is the sequence number.
```

4. Effects of Clock Adjustments on PPS

The user application calls the API functions clock_adjtime() or clock_settime() to inform the CPTS driver about any clock adjustment as a result of running the PTP protocol. The PPS may also need to be adjusted by the driver accordingly.

See **Clock Adjustments** in the CPTS User section for more details on clock adjustments.

- Shifting Time

  The user application informs CPTS driver of the shifts the clock by calling clock_adjtime() with a flag ADJ_SETOFFSET.

  Shifting time may result in shifting the 1 second boundary. As such the driver recalculates the TS_COMP_VAL for the next pulse in order to align the pulse with the 1 second boundary after the shift.

```
Example 1. Positive Shift
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no shifting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose a shift of +0.25 sec occurs at cntr=1458
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5                <- adjtime(ADJ_SETOFFSET, +0.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-13.25 after
the shift), a pulse will go out at cntr=1583 (or sec-14) after the
re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75             (after +0.25 sec shift)
1483   13
1508   13.25             (realign orig pulse to cntr=1583)
1583   14      ^
1608   14.25
1683   15      ^
1708   15.25
.
.
.
```

```
Example 2. Negative Shift
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no shifting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose a shift of -3.25 sec occurs at cntr=1458
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5                <- adjtime(ADJ_SETOFFSET, -3.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-9.75
after the shift), a pulse will go out at cntr=1533 (or sec-10) after the
re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   9.25             (after -3.25 sec shift)
1508   9.75             (realign orig pulse to cntr=1533)
1533   10      ^
1558   10.25
```

```
1608   10.75
1633   11      ^
1658   11.25
1708   11.75
.
.
.
```

Remark: If a second time shift is issued before the next re-aligned pulse is asserted after the first time shift, shifting of the next pulse can be accumulated.

```
Example 3. Accumulated Pulse Shift
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no shifting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose a shift of +0.25 sec occurs at cntr=1458
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5                <- adjtime(ADJ_SETOFFSET, +0.25 sec)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-13.25 after
the shift), a pulse will go out at cntr=1583 (or sec-14) after the
re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75            (after +0.25 sec shift)
1483   13
1508   13.25            (realign orig pulse to cntr=1583)
1583   14      ^
1608   14.25
1683   15      ^
1708   15.25
.
.
.
```

```
Suppose another +0.25 sec time shift is issued at cntr=1533 before the
re-align pulse at cntr=1583 is asserted.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75
1483   13
1508   13.25
1533   13.5             <- adjtime(ADJ_SETOFFSET, +0.25 sec)
1583   14
1608   14.25
1683   15
1708   15.25
.
.
.
```

```
In this case the scheduled pulse at cntr=1583 is further shifted to cntr=1658.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.75
1483   13
1508   13.25
```

```
1533   13.75              (after +0.25 sec shift)
1583   14.25
1608   14.5
1658   15      ^          (realign the cntr-1583-pulse to cntr=1658)
1683   15.25
1708   15.5
1758   16      ^
.
.
.
```

- Setting Time

  The user application may set the internal timecounter kept by the CPTS driver by calling clock_settime().

  Setting time may result in changing the 1-second boundary. As such the driver recalculates the TS_COMP_VAL for the next pulse in order to align the pulse with the 1 second boundary after the shift. The TS_COMP_VAL recalculation is similar to shifting time.

```
Example.
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no time setting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose at cntr=1458, time is set to 100.25 sec
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5                <- settime(100.25 sec)
1508   13
1608   14
1708   15
.
```

```
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-100.75 after
the shift), a pulse will go out at cntr=1533 (or sec-101) after the
re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec      pulse
----   ---      -----
1208   10        ^
1308   11        ^
1408   12        ^
1458   100.25            (after setting time to 100.25 sec)
1508   100.75            (realign orig pulse to cntr=1533)
1533   101       ^
1608   101.75
1633   102       ^
1708   102.75
1733   103       ^
.
.
.
```

- Changing Reference Clock Frequency

  The user application informs the CPTS driver of the changes of the reference clock frequency by calling clock_adjtime() with a flag ADJ_FREQUENCY.

  In this case, the driver re-calculates the TS_COMP_VAL value for the next pulse, and the following pulses, based on the new frequency.

```
Example.
```

```
Assuming a reference clock with freq = 100 Hz and the cpts counter is 1208
at the 10-th second (sec-10).
```

```
If no time setting happens, a pulse is asserted according to the following
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1508   13      ^
1608   14      ^
1708   15      ^
.
.
.
```

```
Suppose at cntr=1458, reference clock freq is changed to 200Hz
```

```
*** Remark: The change to 200Hz is only for illustration.  The
            change should usually be parts-per-billion or ppb.
```

```
      (abs)
cntr   sec    pulse
----   ---    -----
1208   10      ^
1308   11      ^
1408   12      ^
1458   12.5              <- adjtime(ADJ_FREQUENCY, +100Hz)
1508   13
1608   14
1708   15
.
.
.
```

```
Instead of going out at cntr=1508 (which was sec-13 but is now sec-12.75 after
the freq change), a pulse will go out at cntr=1558 (or sec-13 in the new freq)
after the re-alignment at the 1-second boundary.
```

```
      (abs)
cntr   sec      pulse
----   ---      -----
1208   10        ^
1308   11        ^
1408   12        ^
1458   12.5             (after freq changed to 200Hz)
1508   12.75            (realign orig pulse to cntr=1558)
1558   13        ^
1608   13.25
1658   13.5
1708   13.75
1758   14        ^
.
.
.
```

CPTS Hardware Timestamp Push

There are eight hardware time stamp inputs (HW1/8_TS_PUSH) that can cause hardware time stamp push events to be loaded into the event FIFO. The CPTS driver supports the reporting of such timestamps by using the PTP EXTTS feature of the Linux PTP infrastructure.

User applications can request such timestamps through ioctl() and read() function calls.

```
   Example (Reference ii: Documentation/ptp/testptp.c):
```

```
       struct ptp_extts_event event;
       struct ptp_extts_request extts_request;
```

```
       /* which pin to get timestamp from, index is 0 based */
       extts_request.index = 3;
       extts_request.flags = PTP_ENABLE_FEATURE;
```

```
       fd = open("/dev/ptp0", O_RDWR);
```

```
       /* enabling */
       ioctl(fd, PTP_EXTTS_REQUEST, &extts_request);
```

```
       /* reading timestamps */
       for (i=0; i < 10; i++) {
               read(fd, &event, sizeof(event));
               printf("event index %u at %lld.%09u\n", event.index,
                       event.t.sec, event.t.nsec);
       }
```

```
       /* disabling */
       extts_request.flags = 0;
       ioctl(fd, PTP_EXTTS_REQUEST, &extts_request);
```

Testing HW_TS_PUSH on Keystone2 (K2HK) EVM

Note: On K2HK EVM, only two HW_TS_PUSH pins are brought out. These are HW3_TS_PUSH and HW4_TS_PUSH. Refer to K2HK schematic for more details.

To use the TS_COMP_OUT signal to test HW_TS_PUSH:

1. Connect jumper pins CN17-5 (TSCOMPOUT_E) and CN17-3 (TSPUSHEVt0)
2. Connect pins CN3-114 (TSPUSHEVt0) and CN3-109 (TSPUSHEVt0_E). A ZX102-QSH 060-ST card is needed.
3. Modify testptp.c to "extts_request.index = 3", ie. reading timestamp from HW4_TS_PUSH pin
4. Compile testptp
5. Bootup K2HK Linux kernel
6. Under Linux prompt, issue "echo 1 > /sys/devices/soc.0/2090000.netcp/ptp/ptp0/pps_enable" to generate TS_COMP_OUT signals.
7. Under Linux prompt, issue "./testptp -e 10" to read the HW4_TS_PUSH timestamps.

CPTS References

i. Linux Documentation Timestamping Test [11]

ii. Linux Documentation PTP Test [12]

iii. Open Source Project linuxptp [13]

iv. Open Source Project ptpd [14]

v. Linux Documentation PPS [15]

vi. Linux pps-tools [16]

Quality of Service

The linux hardware queue driver will download the Quality of Service Firmware to PDSP 1 of QMSS. PDSP 0 has accumulator firmware.

The firmware will be programmed by the linux keystone hardware queue QoS driver.

The configuration of the firmware is done with the help of device tree bindings. These bindings are documented in the kernel itself at **Documentation/devicetree/bindings/hwqueue/keystone-qos.txt**

QoS Tree Configuration

The QoS implementation allows for an abstracted tree of scheduler nodes represented in device tree form. An example is depicted below

```
+-----------+
|  inputs   |  . . .
+-----------+              +------------+       +------------+
      |       |       |    |  inputs    |       |  inputs    |
      +-------+-------+    +------------+       +------------+
              |                   |                    |
      +---------------+  +----------------+    +--------------+
      |     prio 0    |  |    prio 1      |    |   prio 2     |
      |   unordered   |  |    inputs      |    |   inputs     |
      +---------------+  +----------------+    +--------------+
              |                  |                    |
              +------------------+--------------------+
                                 |
                      +--------------------+
                      |  strict prio node  |
                      +--------------------+
                                 |
                      output to network transmit
```

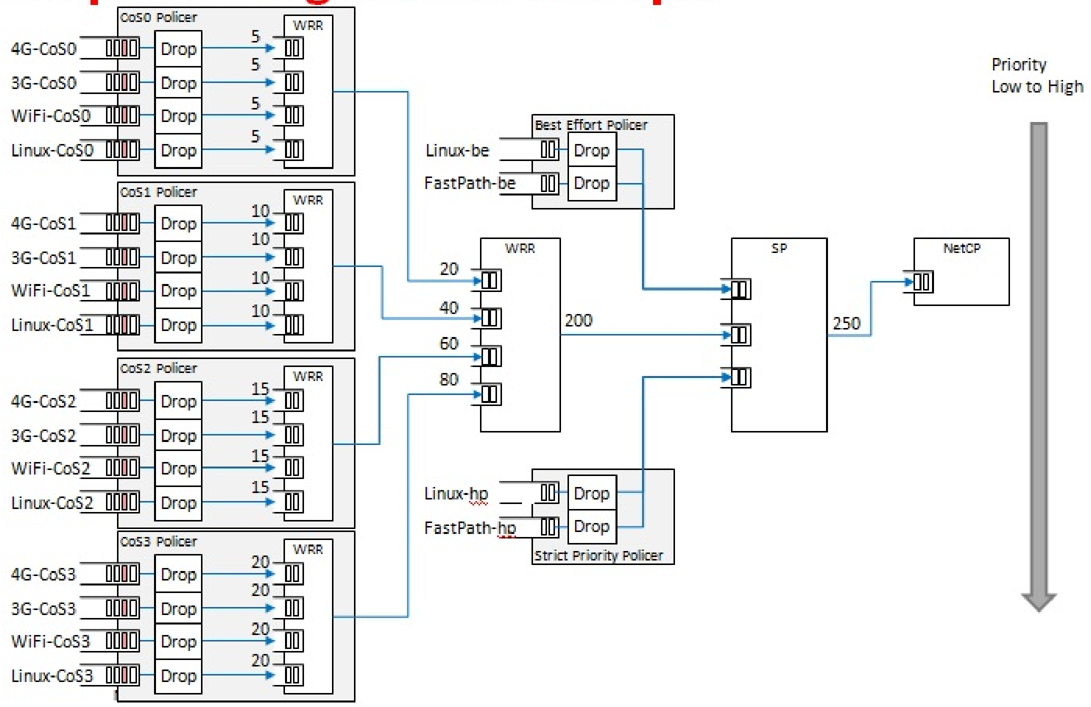
At each node, shaping and dropping parameters may be specified, within limits of the constraints outlined in this document. The following sections detail the device tree attributes applicable for this implementation.

The actual qos tree configuration can be found at linux/arch/arm/boot/dts/tci6638-evm.dts.

The device tree has attributes for configuring the QoS shaper. In the sections below we explain the various qos specific attributes which can be used to setup and configure a QoS shaper.

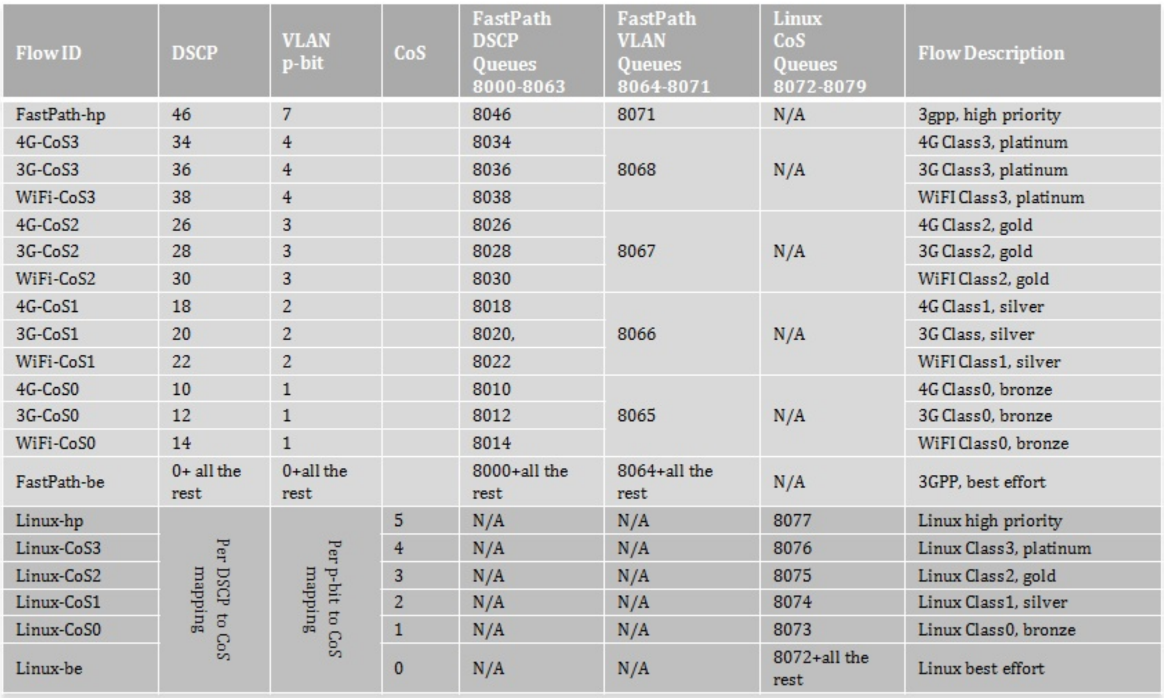
In the device tree we are setting up a shaper that is depicted below

Shaper configuration example



When egress shaper is enabled, all packets will be sent to the QoS firmware for shaping via a set of the queues starting from the Q0S base queue which is 8000 by default. DSCP value in the IP header(outer IP incase of IPSec tunnels) or VLAN pbits (if VLAN interface) are used to determine the QoS queue to which the packet is sent. E.g., if the base queue is 8000, if the DSCP value is 46, the packet will be sent to queue number 8046. i.e., base queue number + DSCP value Incase of VLAN interfaces, if the pbit is 7, the packet will be sent to queue number 8071. i.e., base queue number + skip 64 queues used for DSCP + pbit value.

Shaper Configuration example, details



Flow ID	DSCP	VLAN p-bit	CoS	FastPath DSCP Queues 8000-8063	FastPath VLAN Queues 8064-8071	Linux CoS Queues 8072-8079	Flow Description
FastPath-hp	46	7		8046	8071	N/A	3gpp, high priority
4G-CoS3	34	4		8034	8068	N/A	4G Class3, platinum
3G-CoS3	36	4		8036			3G Class3, platinum
WiFi-CoS3	38	4		8038			WiFI Class3, platinum
4G-CoS2	26	3		8026	8067	N/A	4G Class2, gold
3G-CoS2	28	3		8028			3G Class2, gold
WiFi-CoS2	30	3		8030			WiFI Class2, gold
4G-CoS1	18	2		8018	8066	N/A	4G Class1, silver
3G-CoS1	20	2		8020,			3G Class, silver
WiFi-CoS1	22	2		8022			WiFI Class1, silver
4G-CoS0	10	1		8010	8065	N/A	4G Class0, bronze
3G-CoS0	12	1		8012			3G Class0, bronze
WiFi-CoS0	14	1		8014			WiFI Class0, bronze
FastPath-be	0+ all the rest	0+all the rest		8000+all the rest	8064+all the rest	N/A	3GPP, best effort
Linux-hp	Per DSCP to CoS mapping	Per p-bit to CoS mapping	5	N/A	N/A	8077	Linux high priority
Linux-CoS3			4	N/A	N/A	8076	Linux Class3, platinum
Linux-CoS2			3	N/A	N/A	8075	Linux Class2, gold
Linux-CoS1			2	N/A	N/A	8074	Linux Class1, silver
Linux-CoS0			1	N/A	N/A	8073	Linux Class0, bronze
Linux-be			0	N/A	N/A	8072+all the rest	Linux best effort

QoS Node Attributes

The following attributes are recognized within QoS configuration nodes:

- **"strict-priority" and "weighted-round-robin"**

e.g. strict-priority;

This attribute specifies the type of scheduling performed at a node. It is an error to specify both of these attributes in a particular node. The absence of both of these attributes defaults the node type to unordered(first come first serve).

- **"weight"**

e.g. weight = <80>;

This attribute specifies the weight attached to the child node of a weighted-round-robin node. It is an error to specify this attribute on a node whose parent is not a weighted-round-robin node.

- **"priority"**

e.g. priority = <1>;

This attribute specifies the priority attached to the child node of a strict-priority node. It is an error to specify this attribute on a node whose parent is not a strict-priority node. It is also an error for child nodes of a strict-priority node to have the same priority specified.

- **"byte-units" or "packet-units"**

e.g. byte-units;

The presence of this attribute indicates that the scheduler accounts for traffic in byte or packet units. If this attribute is not specified for a given node, the accounting mode is inherited from its parent node. If this attribute is not specified for the root node, the accounting mode defaults to byte units.

- **"output-rate"**

e.g. output-rate = <31250000 25000>;

The first element of this attribute specifies the output shaped rate in bytes/second or packets/second (depending on the accounting mode for the node). If this attribute is absent, it defaults to infinity (i.e., no shaping). The second element of this attribute specifies the maximum accumulated credits in bytes or packets (depending on the accounting mode for the node). If this attribute is absent, it defaults to infinity (i.e., accumulate as many credits as possible).

- **"overhead-bytes"**

e.g. overhead-bytes = <24>;

This attribute specifies a per-packet overhead (in bytes) applied in the byte accounting mode. This can be used to account for framing overhead on the wire. This attribute is inherited from parent nodes if absent. If not defined for the root node, a default value of 24 will be used. This attribute is passed through by inheritence (but ignored) on packet accounted nodes.

- **"output-queue"**

e.g. output-queue = <645>;

This specifies the QMSS queue on which output packets are pushed. This attribute must be defined only for the root node in the qos tree. Child nodes in the tree will ignore this attribute if specified.

- **"input-queues"**

e.g. input-queues = <8010 8065>;

This specifies a set of ingress queues that feed into a QoS node. This attribute must be defined only for leaf nodes in the QoS tree. Specifying input queues on non-leaf nodes is treated as an error. The absence of input queues on a leaf node is also treated as an error.

- **"stats-class"**

e.g. stats-class = "linux-best-effort";

The stats-class attribute ties one or more input stage nodes to a set of traffic statistics (forwarded/discarded bytes, etc.). The system has a limited set of statistics blocks (up to 48), and an attempt to exceed this count is an error. This attribute is legal only for leaf nodes, and a stats-class attribute on an intermediate node will be treated as an error.

- **"drop-policy"**

e.g. drop-policy = "no-drop"

The drop-policy attribute specifies a drop policy to apply to a QoS node (tail drop, random early drop, no drop, etc.) when the traffic pattern exceeds specifies parameters. The drop-policy parameters are configured separately within device tree (see "Traffic Police Policy Attributes section below). This attribute defaults to "no drop" for applicable input stage nodes. If a node in the QoS tree specifies a drop-policy, it is an error if any of its descendent nodes (children, children of children, ...) are of weighted-round-robin or strict-priority types.

Traffic Police Policy Attributes

The following attributes are recognized within traffic drop policy nodes:

- **"byte-units" or "packet-units"**

e.g. byte-units;

The presence of this attribute indicates that the dropr accounts for traffic in byte or packet units. If this attribute is not specified, it defaults to byte units. Policies that use random early drop must be of byte unit type.

- **"limit"**

e.g. limit = <10000>;

Instantaneous queue depth limit (in bytes or packets) at which tail drop takes effect. This may be specified in combination with random early drop, which operates on average queue depth (instead of instantaneous). The absence of this attribute, or a zero value for this attribute disables tail drop behavior.

- **"random-early-drop"**

e.g. random-early-drop = <32768 65536 2 2000>;

The random-early-drop attribute specifies the following four parameters in order:

low threshold: No packets are dropped when the average queue depth is below this threshold (in bytes). This parameter must be specified.

high threshold: All packets are dropped when the average queue depth above this threshold (in bytes). This parameter is optional, and defaults to twice the low threshold.

max drop probability: the maximum drop probability

half-life: Specified in milli seconds. This is used to calculate the average queue depth. This parameter is optional and defaults to 2000.

Sysfs support

The keystone hardware queue driver has sysfs support for statistics, drop policies and the tree configuration.

```
root@keystone-evm:/sys/devices/soc.0/hwqueue.8/qos-inputs-1# ls
drop-policies  qos-tree       statistics
root@keystone-evm:/sys/devices/soc.0/hwqueue.8/qos-inputs-1#
```

The above shows the location in the kernel where sysfs entries for the keystone hardware queue can be found. There are sysfs entries for the qos trees (qos-inuputs1, qos-tree-inputs2). Within the qos directory there are separate directories for statistics, drop-policies and the qos-tree itself. Each node in the tree is a separate directory entry, starting with the root (tip) entry.

Statistics are displayed for each statistics class in the device tree. Four statistics are represented for each stats class.

- bytes forwarded
- bytes discarded
- packets forwarded
- packets discarded

An example is depicted below

```
cat /sys/devices/soc.0/hwqueue.8/qos-inputs-1/statistics/linux-be/packets_forwarded
```

Drop policy configuration is also displayed for each drop policy. In the case of a drop policy, the parameters can also be changed. This is depicted below. Please note the the parameters that can be modified for tail drop are a subset of the parameters that can be modified for random early drop.

The qos tree is reached via the **qos_tree** directory and its sub-directories. Each sub-directory entry may contain:

- directory entries to reach the subtrees feeding this node
- the input queues to this node (valid for leaf nodes only)
- the output queue from this node
- the output rate for the node. The current value can be shown by: "cat output_rate". The value can be modified by: *echo "<val>" > output_rate*
- the overhead bytes parameter for the node. The current value can be shown by: "cat overhead_bytes". The value can be modified by: *echo "<val>" > overhead_bytes*
- burst size . The current value can be shown by: "cat burst_size". The value can be modified by: *echo "<val>" > burst_size*
- drop_policy . This is the name of the drop policy to be used.
- stats_class associated with node. This is the name of stats class to be used
- the priority of the node (for strict priority nodes only). The current value can be shown by: "cat priority". The value can be modified by: *echo "<val>" > priority*
- weight : for wrr nodes. The current value can be shown by: "cat weight". The value can be modified by: *echo "<val>" > weight*

Debug Filesystem support

Debug Filesystem(debugfs) support is also being provided for QoS support. To make use of debugfs support a user might have to mount a debugfs filesystem. This can be done by issuing the command.

```
mount -t debugfs debugfs /debug
```

The appropriate path and contents are shown below

```
root@keystone-evm:/debug/qos-1# ls
config_profiles  out_profiles     queue_configs    sched_ports
```

With the debugfs support we will be able to see the actual configuration of

- QoS scheduler ports
- Drop scheduler queue configs
- Drop scheduler output profiles
- Drop scheduler config profiles

The QoS scheduler port configuration can be seen by issuing the command **cat /debug/qos/sched_ports**. This is shown below

```
root@keystone-evm:/debug/qos-1# cat sched_ports
port 14
unit flags 15 group # 1 out q 8171 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 15
unit flags 15 group # 1 out q 8170 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 16
unit flags 15 group # 1 out q 8169 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 17
unit flags 15 group # 1 out q 8168 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
```

```
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 6144000
queue 1 cong thresh 0 wrr credit 6144000
queue 2 cong thresh 0 wrr credit 6144000
queue 3 cong thresh 0 wrr credit 6144000
```

```
port 18
unit flags 15 group # 1 out q 8173 overhead bytes 24 throttle thresh 2500 cir credit 5120000 cir max
51200000
total q's 4 sp q's 0 wrr q's 4
queue 0 cong thresh 0 wrr credit 2457600
queue 1 cong thresh 0 wrr credit 4915200
queue 2 cong thresh 0 wrr credit 7372800
queue 3 cong thresh 0 wrr credit 9830400
```

```
port 19
unit flags 15 group # 1 out q 645 overhead bytes 24 throttle thresh 3125 cir credit 6400000 cir max
51200000
total q's 3 sp q's 3 wrr q's 0
queue 0 cong thresh 0 wrr credit 0
queue 1 cong thresh 0 wrr credit 0
queue 2 cong thresh 0 wrr credit 0
```

```
root@keystone-evm:/debug/qos-1#
```

cat command can be used in a similar way for displaying the Drop scheduler queue configs, output profiles and config profiles

Configuring QoS on an 1-GigE interface

To configure QoS on an interface, several definitions must be added to the device tree:

- Drop policies and a QoS tree must be defined. The outer-most QoS block must specify an output queue number; this may be the 1-GigE NETCP's PA PDSP 5 (645) or CPSW (648), one of the 10-GigE CPSW's queues (8752, 8753), or other queue as appropriate.

```
Example (k2hk-evm.dts):
```

```
droppolicies: default-drop-policies {
        no-drop {
                default;
                packet-units;
                limit = <0>;
        };
        ...
        all-drop {
                byte-units;
                limit = <0>;
        };
};
```

```
Example (k2hk-evm.dts):
```

```
qostree: qos-tree {
        strict-priority;                /* or weighted-round-robin */
        byte-units;                     /* packet-units or byte-units */
        output-rate = <31250000 25000>;
        overhead-bytes = <24>;          /* valid only if units are bytes */
        output-queue = <645>;           /* allowed only on root node */
```

```
        high-priority {
                ...
        }
        ...
        best-effort {
                ...
        };
};
```

```
qostree2: qos-tree-2 {
        strict-priority;                /* or weighted-round-robin */
        byte-units;                     /* packet-units or byte-units */
        output-rate = <31250000 25000>;
        overhead-bytes = <24>;          /* valid only if units are bytes */
        output-queue = <648>;           /* allowed only on root node */
```

```
        high-priority {
                ...
        }
        ...
        best-effort {
                ...
        };
};
```

- QoS inputs must be defined to the hwqueue subsystem. The QoS inputs block defines which group of hwqueues will be used, and links to the set of drop policies and QoS tree to be used.

```
Example (k2hk-evm.dts):
```

```
hwqueue0: hwqueue@2a40000 {
        ...
        queues {
                ...
                qos-inputs-1 {
                        values                  = <8000 192>;
                        pdsp-id                 = <3>;
                        ...
                        drop-policies           = <&droppolicies>;
                        qos-tree                = <&qostree>;
                        reserved;
                };
                qos-inputs-2 {
                        values                  = <6400 192>;
                        pdsp-id                 = <7>;
                        ...
                        drop-policies           = <&droppolicies>;
                        qos-tree                = <&qostree2>;
                        reserved;
                };
        };
}; /* hwqueue0 */
```

- A PDSP must be defined, and loaded with the QoS firmware.

```
Example (k2hk-evm.dts):
```

```
hwqueue0: hwqueue@2a40000 {
        ...
        pdsps {
                ...
                pdsp3@0x2a13000 {
                        firmware = "keystone/qmss_pdsp_qos_k2_le_2_0_1_5.fw";
                        ...
                        id = <3>;
                };
                pdsp7@0x2a17000 {
                        firmware = "keystone/qmss_pdsp_qos_k2_le_2_0_1_5.fw";
                        ...
                        id = <7>;
                };
        };
}; /* hwqueue0 */
```

- A Packet DMA channel must be defined and associated with each of the QoS input queues

```
Example (k2hk-evm.dts):
```

```
padma: pktdma@2004000 {
         ...
         channels {
                ...
                qos0 {
                        transmit;
                        label           = "qos0";
                        pool            = "pool-net";
                        submit-queue    = <8072>;
                        ...
                };
                ...
                qos5 {
                        transmit;
                        label           = "qos5";
                        pool            = "pool-net";
                        submit-queue    = <8077>;
                        complete-queue  = <8707>;
                        ...
                };
                qos6 {
                        transmit;
                        label           = "qos6";
                        pool            = "pool-net";
                        submit-queue    = <6472>;
                        ...
                };
                ...
                qos11 {
                        transmit;
                        label           = "qos11";
                        pool            = "pool-net";
                        submit-queue    = <6477>;
                        complete-queue  = <8708>;
                        ...
                };
        };
};
```

- A NETCP QoS block must be defined. For each interface and "interface-x" block is defined, which contains definitions for each of the QoS input DMA channels to be associated with that interface.

```
Example (k2hk-evm.dts):
```

```
netcp: netcp@2090000 {
        ...
        qos: qos@0 {
                label = "keystone-qos";
                multi-interface;
```

```
                interface-0 {
                        chan-0 {
                                tx-channel = "qos0";
                                tx_queue_depth = <64>;
                        };
                        ...
                        chan-5 {
                                tx-channel = "qos5";
                                tx_queue_depth = <64>;
                        };
                };
                interface-1 {
                        chan-0 {
                                tx-channel = "qos6";
                                tx_queue_depth = <64>;
                        };
                        ...
                        chan-5 {
                                tx-channel = "qos11";
                                tx_queue_depth = <64>;
                        };
                };
        };
};
```

- By default, Linux network traffic will be queued to the interface "chan-0" subqueue. To classify and route packets from Linux to specific QoS queues, the Linux traffic control utility "tc" must be used. First a class-full root queuing discipline must be established for the interface, and then filters may be used to classify packets. These filters can use the "skbedit queue_mapping" action to set the subqueue number for the packet. Here is an example:

```
# Clear any existing configuration
tc qdisc del dev eth0 root
```

```
# Add DSMARK as the root qdisc
tc qdisc add dev eth0 root handle 1 dsmark indices 8 default_index 0
```

```
# Create filters to classify packets and route to queues
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5002 0xffff \
        action skbedit queue_mapping 1
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5003 0xffff \
        action skbedit queue_mapping 2
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5004 0xffff \
        action skbedit queue_mapping 3
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5005 0xffff \
        action skbedit queue_mapping 4
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
        u32 match ip dport 5006 0xffff \
        action skbedit queue_mapping 5
```

Please refer to the Linux Advanced Routing & Traffic Control how-tos and related manpages available on the Internet for more information on "tc".

Disabling QoS on an 1-GigE interface

The released "k2hk-evm.dts" file contains definitions for two QoS trees and associates them with the first two ports on the 1-GigE interface. These default trees are configured so that traffic queued to interface subqueue 0 will bypass the QoS tree. Only traffic specifically directed to subqueues 1-6 will be processed through the hardware QoS subsystem. This may be sufficient for your needs. However, you may prefer to remove the QoS configuration entirely from the device tree.

To disable QoS on the two 1-GigE interfaces

- delete all the qos related blocks or entries shown in the examples in section Configuring QoS on an 1-GigE interface, namely
  - droppolicies: default-drop-policies {...}
  - qostree: qos-tree {...}
  - qostree2: qos-tree-2 {...}
  - qos-inputs-1 {...}
  - qos-inputs-2 {...}
  - pdsp3@0x2a13000 {...}
  - pdsp7@0x2a17000 {...}
  - qos0 {...} to qos11 {...}
  - qos: qos@0 {...}
- modify tx_data_queue_depth in the pa: pa@2000000 {...} block in k2hk-evm.dts to 256.

Configuring QoS on a 10-GigE interface

The following snippets together shows how to remove the QoS tree associated with the second port of the 1-GigE interface and associate it with the first port on the 10-GigE interface. In these snippets, we only depict and highlight the modifications made to the above 1-GigE examples. Contents not shown in the definitions should just be copy and paste from the file k2hk-evm.dts.

Note: this is only for demonstration purpose and is not part of the release.

- Modify the output-queue number of qostree2 to that of the transmit queue of the 10-GigE's first port.

```
qostree2: qos-tree-2 {
        strict-priority;                /* or weighted-round-robin */
        byte-units;                     /* packet-units or byte-units */
        output-rate = <31250000 25000>;
        overhead-bytes = <24>;          /* valid only if units are bytes */
        output-queue = <8752>;           /* allowed only on root node */
```

```
        high-priority {
                ...
        }
        ...
        best-effort {
                ...
        };
};
```

- Remove qos6 to qos11 from padma's channels block.

```
padma: pktdma@2004000 {
         ...
         channels {
                ...
                qos0 {
                        transmit;
                        label           = "qos0";
                        pool            = "pool-net";
                        submit-queue    = <8072>;
                        ...
                };
                ...
                qos5 {
                        transmit;
                        label           = "qos5";
                        pool            = "pool-net";
                        submit-queue    = <8077>;
                        complete-queue  = <8707>;
                        ...
                };
                /* qos6 - qos11 removed */
        };
};
```

- Define qos6 to qos11 in xgedma's channels block. This is a cut-and-paste of those defined in the padma block but with the pool name pool-**net** modified to pool-**xge**.

```
xgedma: pktdma@2fa1000 {
         ...
         channels {
                ...
                qos6 {
                        transmit;
                        label           = "qos6";
                        pool            = "pool-xge";
                        submit-queue    = <6472>;
                        ...
                };
                ...
                qos11 {
                        transmit;
                        label           = "qos11";
                        pool            = "pool-xge";
                        submit-queue    = <6477>;
                        complete-queue  = <8708>;
                        ...
                };
        };
};
```

- Remove interface-1 from netcp's qos block.

```
netcp: netcp@2090000 {
        ...
        qos: qos@0 {
                label = "keystone-qos";
                multi-interface;
                interface-0 {
                        chan-0 {
                                tx-channel = "qos0";
                                tx_queue_depth = <64>;
                        };
                        ...
                        chan-5 {
                                tx-channel = "qos5";
                                tx_queue_depth = <64>;
                        };
                };
                /* interface-1 removed */
        };
};
```

- Define a qos block qos**x**: qos@0 {...} in 10-GigE's netcp block. The chan-N content of this block is the same as those defined in the interface-1 of netcp's qos.

```
netcpx: netcp@2f00000 {
        ...
        qosx: qos@0 {
                label = "keystone-qos";
                multi-interface;
```

```
                interface-0 {
                        chan-0 {
                                tx-channel = "qos6";
                                tx_queue_depth = <64>;
                        };
                        ...
                        chan-5 {
                                tx-channel = "qos11";
                                tx_queue_depth = <64>;
                        };
                };
        };
};
```

Crypto Driver

Keystone SoC includes a hardware cryptographic accelerator engine also known as CP_ACE (Communication Processor Adaptive Cryptographic engine). The CP_ACE subsystem is designed to provide packet security as part of IPSec, SRTP and 3GPP industry standards. Keystone Linux kernel implements a crypto driver which offloads crypto algorithm processing to CP_ACE. Crypto driver registers algorithm implementations in the kernel's crypto algorithm management framework. Since the primary use case for this driver is IPSec ESP offload, it currently registers only AEAD algorithms. Following algorithms are supported by the driver:

1. authenc(hmac(sha1),cbc(aes))
2. authenc(hmac(sha1),cbc(des3-ede))
3. authenc(xcbc(aes),cbc(aes))

4. authenc(xcbc(aes),cbc(des3-ede))

The driver code can be found in the files drivers/crypto/keystone-sa.[ch]

USB Driver

The USB 3.0 xHC controller supports the hardware part of the xHCI standard and is implemented with the Synopsys DesignWare core. Driver support of the hardware is provided by the generic usb driver code found in directories and files drivers/usb/, drivers/usb/host/xhci*.c and drivers/usb/dwc3/. Platform specific XHCI glue code is implemented in drivers/usb/dwc3/dwc3-keystone.c and drivers/usb/phy/phy-keystone.c. Currently only usb host mode is supported.

K2E has 2 USB controllers, USB0 and USB1. This USB driver supports both USB controllers simultaneously in host mode. However, the USB connector for USB1 on K2E EVM is an USB 3.0 micro AB connector, adapter is needed for using it in host mode. Because of that, the devicetree bindings for USB1 is disabled by default. To enable the USB1 devicetree bindings, do one of the following:

- Enabling K2E USB1 at Compile Time

  Update the **status** property of usb1_phy and usb1 bindings from "disabled" to "ok" in arch/arm/boot/dts/k2e.dtsi:

```
usb1_phy: usb_phy@2620750 {
        status = "ok";
};
usb1: usb@25000000 {
        status = "ok";
};
```

- Enabling K2E USB1 at Bootup Time

  Alternately, the above updates can be done right before kernel bootup through u-boot scripting.

```
setenv run_fdt_usb1 'fdt addr ${addr_fdt}; fdt set /soc/usb_phy@2620750 status "ok"; fdt set /soc/usb@25000000 status "ok" '

setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon run_fdt_usb1 run_kern'
```

10Gig Ethernet Driver

The TI Keystone-2 platforms include a 10 Gig Ethernet Subsystem. The subsystem combines a 3-port ethernet switch sub-module and a packet DMA capable of 10Gbps and 1Gbps per ethernet port. The ethernet switch sub-module contains an EMAC module, SGMII modules, 10GBase-R module, SERDES module, MACSec module and MDIO module.

The 10Gig Ethernet Driver makes use of the multi-instance support feature of the NETCP driver which serves as a hook to the Linux Network Stack. The 10Gig Ethernet Driver provides its API to the NETCP driver by registering itself to an instance of the NETCP driver during kernel initialization.

Refer to the Network Driver section for more details about the NETCP driver model. But notice that the packet accelerator in that section is not applicable to the 10Gig Ethernet subsystem.

The 10Gig Ethernet Driver code can be found in files keystone_xgess.c, keystone_xgepcsr.c and keystone_xgemdio.c in directory drivers/net/ethernet/ti/.

For releases **before MCSDK 3.0.2** release, the 10Gig ethernet driver **was not tested due to lack of hardware**. In those releases the driver is not included in the kernel default configuration. To enable the 10Gig Ethernet Driver, add the following two lines in file arch/arm/configs/keystone2_defconfig or arch/arm/configs/keystone2_fullrt_defconfig, and do a distclean build.

```
CONFIG_TI_KEYSTONE_XGE=y
CONFIG_TI_KEYSTONE_XGE_MDIO=y
```

Starting from MCSDK 3.0.2 release, the following lines are added to arch/arm/configs/keystone2_defconfig and arch/arm/configs/keystone2_fullrt_defconfig to enable 10Gig ethernet driver by default.

```
CONFIG_TI_KEYSTONE_XGE=y
CONFIG_MDIO_BITBANG=y
CONFIG_MDIO_GPIO=y
CONFIG_GPIO_PCA953X=y
```

Enabling 10Gig Ethernet Driver Device Tree Bindings

Starting from MCSDK 3.1.0 release, the 10Gig related device tree bindings are disabled by default. Depending on the link interface type, the 10Gig device tree bindings can be enabled for K2E and K2HK platforms at compile time or bootup time as follows.

MAC-to-PHY Link Interface

For example, when a TI K2HK or K2E EVM is connected to a **RevA RTM-BOC with 10Gig PHY**.

- Enabling 10Gig at Compile Time

  Update the **status** property of mdiox and netcpx bindings from "disabled" to "ok" in arch/arm/boot/dts/k2e-net.dtsi or arch/arm/boot/dts/k2hk-net.dtsi:

  mdiox0: mdiox {

  status = "**ok**";

  };

  netcpx: netcp@2f00000 {

  status = "**ok**";

  };

  Also update the **status** property of pca@20 binding from "disabled" to "ok" in arch/arm/boot/dts/keystone.dtsi:

  pca@20 {

  status = "**ok**";

  };

- Enabling 10Gig at Bootup Time

  Alternately, the above updates can be done right before kernel bootup through u-boot scripting.

  setenv run_fdt 'fdt addr ${addr_fdt}; fdt set /soc/i2c2/pca@20 status "ok"; fdt set /soc/netcp@2f00000 status "ok"; fdt set /soc/mdiox/ status "ok" '

  setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon **run_fdt** run_kern'

MAC-to-MAC Link Interface

For example, when a TI K2HK or K2E EVM is connected to a **Rev.B/C RTM-BOC with Dual Retimer**.

- Enabling 10Gig at Compile Time

  Update the **status** property of netcpx binding from "disabled" to "ok" and set each slave's **link-interface** type to 11 in arch/arm/boot/dts/k2e-net.dtsi or arch/arm/boot/dts/k2hk-net.dtsi:

  netcpx: netcp@2f00000 {

```
                status = "ok";
                slaves {
                        slave0 {
                                link-interface = <11>;
                                phy-handle = <&phyx0>;
                        };
                        slave1 {
                                link-interface = <11>;
                                phy-handle = <&phyx1>;
                        };
                };
        };
```

Note: The phy-handle in each slave node can optionally be removed. phy-handle will not take effect when the link-interface is MAC-to-MAC.

- Enabling 10Gig at Bootup Time

  Alternately, the above updates can be done right before kernel bootup through u-boot scripting.

```
setenv run_fdt_1 'fdt addr ${addr_fdt}; fdt set /soc/netcp@2f00000 status "ok"'
setenv run_fdt_2 'fdt set /soc/netcp@2f00000/cpswx@2f00000/slaves/slave0 link-interface <11>'
setenv run_fdt_3 'fdt set /soc/netcp@2f00000/cpswx@2f00000/slaves/slave1 link-interface <11>'
setenv bootcmd 'run init_${boot} get_fdt_${boot} get_mon_${boot} get_kern_${boot} run_mon run_fdt_1 run_fdt_2 run_fdt_3 run_kern'
```

The driver has been verified on Keystone K2HK and K2E EVMs connected to RevA RTM-BOC with 10Gig PHY or RevB RTM-BOC with Dual Retimer.

Disabling Support of 10-GigE

To disable the support of 10Gig ethernet,

- Remove the 10-GigE configurations from the defconfig file arch/arm/configs/keystone2_defconfig or arch/arm/configs/keystone2_fullrt_defconfig
- Remove the following bindings from the file arch/arm/boot/dts/k2hk-evm.dts
  - xge{...} and pool-xge {...} in hwqueue0
  - xgedma: pktdma@2fa1000 {...}
  - mdiox0: mdiox {...}
  - netcpx: netcp@2f00000 {...}

PCIe Driver

The TI Keystone platforms contain a PCIe module which supports a single bi-directional link interface with a maximum of two lanes width (x2). The module supports Root Complex and End Point operation modes.

The PCIe driver implemented supports only the Root Complex (RC) operation mode on K2 platforms (K2HK, K2E). An irq-chip is implemented to support both legacy and MSI irqs. The driver's main implementation can be found in drivers/pci/host/pcie-keystone.c and platform specific code can be found at drivers/pci/host/k2-platform.c

PCIe has been verified on K2E and K2HK EVMs. K2E supports two PCI ports. Port 0 is on Domain 0 and Port 1 is on Domain 1. On K2E EVM, a Marvel SATA controller, 0x9182 is connected to port 1 that supports interfacing with Hard disk drives (HDD). Following h/w setup is used to test SATA HDD interface with K2E. Western Digital 1.0

TB SATA / 64MB Cache hard disk drive, WD10EZEX is used for the test over PCI port 1.

```
-----------     SATA 6Gbps data cable    ------------
| WD10EZEX | --------------------------> |  K2E EVM |
-----------                              ------------
       ^
       |
(External power supply)
```

Connect HDD to an external power supply. Connect the HDD SATA port to K2E EVM SATA port using a 6Gbps data cable and power on the HDD. Power On K2E EVM. The K2E rev 1.0.2.0 requires a hardware modification to get the SATA detection on the PCI bus. Please check with EVM hardware vendor for the details.

At boot up, the PCI enumeration should display the SATA EP detection.

```
[341130.650587] keystone-pcie: Starting PCI scan, nr 0...
[341130.650728] PCI host bridge to bus 0001:00
[341130.650737] pci_bus 0001:00: root bus resource [mem 0x60000000-0x6fffffff]
[341130.650744] pci_bus 0001:00: root bus resource [io  0x4000-0x7fff]
[341130.650751] pci_bus 0001:00: No busn resource found for root bus, will use [bus 00-ff]
[341130.651003] PCI: bus0: Fast back to back transfers disabled
[341130.651011] pci 0001:00:00.0: bridge configuration invalid ([bus 00-00]), reconfiguring
[341130.651519] PCI: bus1: Fast back to back transfers disabled
[341130.651548] keystone-pcie: Ending PCI scan...
[341130.651556] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651560] keystone-pcie: keystone_pcie_map_irq: legacy_irq 572
[341130.651570] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 2
[341130.651574] keystone-pcie: keystone_pcie_map_irq: legacy_irq 573
[341130.651584] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651588] keystone-pcie: keystone_pcie_map_irq: legacy_irq 572
[341130.651597] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651601] keystone-pcie: keystone_pcie_map_irq: legacy_irq 608
[341130.651609] keystone-pcie: keystone_pcie_map_irq: slot 0, pin 1
[341130.651613] keystone-pcie: keystone_pcie_map_irq: legacy_irq 608
[341130.651631] pci 0001:00:00.0: BAR 8: assigned [mem 0x60000000-0x601fffff]
[341130.651639] pci 0001:00:00.0: BAR 9: assigned [mem 0x60200000-0x603fffff pref]
[341130.651647] pci 0001:00:00.0: BAR 7: assigned [io  0x4000-0x4fff]
[341130.651657] pci 0001:01:00.0: BAR 6: assigned [mem 0x60200000-0x6020ffff pref]
[341130.651665] pci 0001:01:00.0: BAR 5: assigned [mem 0x60000000-0x600001ff]
[341130.651676] pci 0001:01:00.0: BAR 4: assigned [io  0x4000-0x400f]
[341130.651686] pci 0001:01:00.0: BAR 0: assigned [io  0x4010-0x4017]
[341130.651695] pci 0001:01:00.0: BAR 2: assigned [io  0x4018-0x401f]
[341130.651705] pci 0001:01:00.0: BAR 1: assigned [io  0x4020-0x4023]
[341130.651715] pci 0001:01:00.0: BAR 3: assigned [io  0x4024-0x4027]
[341130.651725] pci 0001:00:00.0: PCI bridge to [bus 01]
[341130.651732] pci 0001:00:00.0:   bridge window [io  0x4000-0x4fff]
[341130.651740] pci 0001:00:00.0:   bridge window [mem 0x60000000-0x601fffff]
[341130.651748] pci 0001:00:00.0:   bridge window [mem 0x60200000-0x603fffff pref]
[341130.651757] PCI: enabling device 0001:00:00.0 (0140 -> 0143)
[341130.651764] keystone-pcie: keystone_pcie_rc_init - end
```

And then look for AHCI and SCSI logs as

```
[   15.651707] ahci 0001:01:00.0: AHCI 0001.0000 32 slots 2 ports 6 Gbps 0x3 impl SATA mode
[   15.658443] ahci 0001:01:00.0: flags: 64bit ncq sntf led only pmp fbs pio slum part sxs
[   15.666086] scsi0 : ahci
[   15.668410] scsi1 : ahci
[   15.670689] ata1: SATA max UDMA/133 abar m512@0x60000000 port 0x60000100 irq 612
[   15.676869] ata2: SATA max UDMA/133 abar m512@0x60000000 port 0x60000180 irq 612
```

Very first time with no partition and file format on HDD, following logs will be displayed

```
[   15.967048] ata1: SATA link down (SStatus 0 SControl 300)
[   16.142035] ata2: SATA link up 6.0 Gbps (SStatus 133 SControl 300)
[   16.147646] ata2.00: ATA-9: WDC WD10EZEX-08M2NA0, 01.01A01, max UDMA/100
[   16.153230] ata2.00: 1953525168 sectors, multi 0: LBA48 NCQ (depth 31/32), AA
[   16.159659] ata2.00: configured for UDMA/100
[   16.163344] scsi 1:0:0:0: Direct-Access     ATA      WDC WD10EZEX-08M 01.0 PQ: 0 ANSI: 5
[   16.170463] sd 1:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/931 GiB)
[   16.176842] sd 1:0:0:0: [sda] 4096-byte physical blocks
[   16.181339] sd 1:0:0:0: [sda] Write Protect is off
[   16.185364] sd 1:0:0:0: Attached scsi generic sg0 type 0
[   16.185393] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[   16.212599]  sda: unknown partition table
[   16.216342] sd 1:0:0:0: [sda] Attached SCSI disk
```

With partition created (in this example there are 4 primary partitions, one for storing file system, one for swap and additional two partitions.

```
[ 4570.941363] ata1: SATA link down (SStatus 0 SControl 300)
[ 4571.108000] ata2: SATA link up 6.0 Gbps (SStatus 133 SControl 300)
[ 4571.113604] ata2.00: ATA-9: WDC WD10EZEX-08M2NA0, 01.01A01, max UDMA/100
[ 4571.119188] ata2.00: 1953525168 sectors, multi 0: LBA48 NCQ (depth 31/32), AA
[ 4571.125625] ata2.00: configured for UDMA/100
[ 4571.129313] scsi 1:0:0:0: Direct-Access     ATA      WDC WD10EZEX-08M 01.0 PQ: 0 ANSI: 5
[ 4571.136422] sd 1:0:0:0: [sda] 1953525168 512-byte logical blocks: (1.00 TB/931 GiB)
[ 4571.136550] sd 1:0:0:0: Attached scsi generic sg0 type 0
[ 4571.147216] sd 1:0:0:0: [sda] 4096-byte physical blocks
[ 4571.151686] sd 1:0:0:0: [sda] Write Protect is off
[ 4571.155726] sd 1:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[ 4571.179058]  sda: sda1 sda2
```

When booted with Filesystem on SATA HDD with ext4 file format.

Command line parameters for SATA rootfs.

```
[    0.000000] Kernel command line: console=ttyS0,115200n8 rootwait=1 ro root=/dev/sda1
[    0.000000] PID hash table entries: 4096 (order: 2, 16384 bytes)
```

Log showing mounting for ext4 FS on partition 1 formatted with ext4 file format

```
[  433.556931] EXT3-fs (sda1): error: couldn't mount because of unsupported optional features (240)
[  433.564563] EXT4-fs (sda1): couldn't mount as ext2 due to feature incompatibilities
[  433.578464] EXT4-fs (sda1): INFO: recovery required on readonly filesystem
[  433.584242] EXT4-fs (sda1): write access will be enabled during recovery
[  433.664817] EXT4-fs (sda1): recovery complete
[  433.672994] EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
[  433.679391] VFS: Mounted root (ext4 filesystem) readonly on device 8:1.
```

Port 0 is known to work on customer boards using SATA interface. Internally verified using microTCA AMC chassis and using intel's e1000e NIC card on port 0. The driver for e1000e is supported by intel on its website and IS NOT supported by TI.

There are few dts configuration parameters available to customize the driver.See Documentation/devicetree/bindings/pci/pcie-keystone.txt in Linux tree for DT bindings information.

Procedure to boot Linux with FS on harddisk

Boot Linux kernel on K2E EVM using NFS file system or Ramfs and using rootfs provided in the SDK. Make sure SATA HDD is connected to EVM as explained above and SATA EP is detected during boot up. This example uses a 1TB HDD and create two partition. First partition is for filesystem and is 510GB and second is for swap and is 256MB.

1. create partition with fdisk

First step is to create 2 partitions using fdisk command. At Linux console type the following commands

```
root@keystone-evm:~# fdisk /dev/sda
Welcome to fdisk (util-linux 2.21.2).
```

```
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

```
Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x9b51b66e.
```

```
The device presents a logical sector size that is smaller than
the physical sector size. Aligning to a physical sector (or optimal
I/O) size boundary is recommended, or performance may be impacted.
```

```
Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)
```

```
Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-1953525167, default 2048): 2048
Last sector, +sectors or +size{K,M,G} (2048-1953525167, default 1953525167): +510G
Partition 1 of type Linux and of size 510 GiB is set
```

```
Command (m for help): n
Partition type:
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1-4, default 2): 2
First sector (1069549568-1953525167, default 1069549568):
Using default value 1069549568
Last sector, +sectors or +size{K,M,G} (1069549568-1953525167, default 1953525167): +256M
Partition 2 of type Linux and of size 256 MiB is set
```

```
Command (m for help): p
```

```
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x9b51b66e
```

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1069549567   534773760   83  Linux
/dev/sda2      1069549568  1070073855      262144   83  Linux
```

```
Command (m for help): p
```

```
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x9b51b66e
```

```
  Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1069549567   534773760   83  Linux
/dev/sda2      1069549568  1070073855      262144   83  Linux
```

```
Command (m for help): t
Partition number (1-4): 2
```

```
Hex code (type L to list codes): L
```

```
 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden C:  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi eb  BeOS fs
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         ee  GPT
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ef  EFI (FAT-12/16/
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        f0  Linux/PA-RISC b
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f1  SpeedStor
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f4  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f2  DOS secondary
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      fb  VMware VMFS
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fc  VMware VMKCORE
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fd  Linux raid auto
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fe  LANstep
1c  Hidden W95 FAT3 75  PC/IX           be  Solaris boot    ff  BBT
1e  Hidden W95 FAT1 80  Old Minix
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap / Solaris)
```

```
Command (m for help): p
```

```
Disk /dev/sda: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders, total 1953525168 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
Disk identifier: 0x9b51b66e
```

```
  Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048  1069549567   534773760   83  Linux
/dev/sda2      1069549568  1070073855      262144   82  Linux swap / Solaris
```

2. Format partitions

```
root@keystone-evm:~# mkfs.ext4 /dev/sda1
mke2fs 1.42.1 (17-Feb-2012)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
```

```
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
33423360 inodes, 133693440 blocks
6684672 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=0
4080 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
       32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
       4096000, 7962624, 11239424, 20480000, 23887872, 71663616, 78675968,
       102400000
```

```
Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

```
root@keystone-evm:~# ls -ltr /dev/sda*
brw-rw----    1 root     disk        8,   2 Sep 21 14:37 /dev/sda2
brw-rw----    1 root     disk        8,   0 Sep 21 14:37 /dev/sda
brw-rw----    1 root     disk        8,   1 Sep 21 14:40 /dev/sda1
```

3. Copy filesystem to rootfs

This procedure assumes the cpio file for SDK filesystem is available on the NFS or ramfs.

```
>mkdir /mnt/test
>mount -t ext4 /dev/sda1 /mnt/test
>cd /mnt/test
>cpio -i -v </<rootfs>.cpio
>cd /
>umount /mnt/test
Where rootfs.cpio is the cpio file for the SDK fileystem.
```

4. Booting with FS on harddisk

Once the harddisk is formatted and has a rootfs installed, following procedure can be used to boot Linux kernel using this rootfs.

Boot EVM to u-boot prompt. Add following env variables to u-boot environment :-

```
K2E EVM # setenv boot hdd
K2E EVM # setenv get_fdt_hdd 'dhcp ${addr_fdt} ${tftp_root}/${name_fdt}'
K2E EVM # setenv get_kern_hdd 'dhcp ${addr_kern} ${tftp_root}/${name_kern}'
K2E EVM # setenv get_mon_hdd 'dhcp ${addr_mon} ${tftp_root}/${name_mon}'
K2E EVM # setenv init_hdd 'run set_fs_none  args_all  args_hdd'
K2E EVM # setenv args_hdd 'setenv bootargs ${bootargs} ro root=/dev/sda1'
K2E EVM # saveenv
```

Now type boot command and boot to Linux. The above steps can be skipped once u-boot implements these env variables by default which is expected to be supported in the future.

AER Driver

AER driver is used for PCIE error handling and recovery. AER driver is attached to RC Port. AER driver requires following kernel command line parameters in bootargs

```
pcie_ports=native pcie_aer=nomsi
```

AER driver is tested by simulating a Unsupported Request. To do so, the EP's BAR is corrupted. For the EP driver to send error responses to RC, SERR bit in PCI_COMMAND must be enabled. Also must enable AER error reporting using pci_enable_pcie_error_reporting() API. For example to enable AHCI driver to enable error reporting to RC port, following code (shows diff) was added to drivers/ata/ahci.c in ahci_init_one()

```
#define DRV_VERSION    "3.0"
+extern int pci_enable_pcie_error_reporting(struct pci_dev *dev);
```

```
....
```

```
ahci_init_one(struct pci_dev *pdev, const struct pci_device_id *ent)
.....
```

```
       int ahci_pci_bar = AHCI_PCI_BAR_STANDARD;
+      u16 pci_cmd;
```

```
....
```

```
+       pci_read_config_word(pdev, PCI_COMMAND, &pci_cmd);
+       pci_cmd |= PCI_COMMAND_SERR;
+       pci_write_config_word(pdev, PCI_COMMAND, pci_cmd);
+       pci_enable_pcie_error_reporting(pdev);
        return ata_host_activate(host, pdev->irq, ahci_interrupt,
                 IRQF_SHARED, &ahci_sht);
```

To simulate Unsupported request error, corrupt the memory BAR at EP's config space. For example, write 0 to BAR offset 0x24 as below (dump should show an address in the 0x5xxxxxxx range at 0x21802024 assuming CFG_SETUP is pointing to EP's config space) :-

```
>devmem2 0x21802024 w 0x0
```

The above assumes the CFG_SETUP register is pointing to bus 1, dev 0, function 0. If not, read the value at offset 0x8 and update it using dev2mem command. Refer the device UG for details of the register.

For AHCI case, try writing to the disk using following command after corrupting the BAR

```
dd if=/dev/zero of=/media/sda1/test-file.bin bs=1K count=1024
```

The command will time out. Once this is timedout, dmesg command will show something like below for unsupported error received from EP at the RC's port.

```
[  342.828683] pcieport 0000:00:00.0: AER: Multiple Uncorrected (Non-Fatal)
   error received: id=0100
[  342.837572] ahci 0000:01:00.0: PCIe Bus Error: severity=Uncorrected (Non-
   Fatal), type=Transaction Layer, id=0100(Requester ID)
[  342.848916] ahci 0000:01:00.0:   device [1b4b:9125] error
   status/mask=00100000/00000000
[  342.848920] ahci 0000:01:00.0:    [20] Unsupported Request    (First)
[  342.848927] ahci 0000:01:00.0:   TLP Header: 40003001 0000000f 50000134
   00000000
[  342.848945] ahci 0000:01:00.0: broadcast error_detected message
```

```
root@keystone-evm:~# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
 29:          0          0          0          0       GIC  arch_timer
 30:      51241      51511      51736      51561       GIC  arch_timer
 70:          1          0          0          0       GIC  pcie-error-irq, aerdrv
 80:      10858          0          0          0       GIC  hwqueue-8704
 81:          0          0          0          0       GIC  hwqueue-8705
 82:         16          0          0          0       GIC  hwqueue-8706
 83:          0          0          0          0       GIC  hwqueue-8707
 84:          0          0          0          0       GIC  hwqueue-8708[
```

The above shows one error interrupt received by RC port.

More information on AER driver can be seen at Documentation/PCI/pcieaer-howto.txt in Linux source tree.

DMA Coherency

Multicore Cortex-A15 supports cache coherency. So, if one core writes to the memory page and that page has a sharable attribute, corresponding cache entries of other cores will be updated. If SoC doesn’t support DMA coherency and external DMA master performs a write transaction L1 and L2 caches will not be updated. So, the software has to invalidate corresponding cache lines. Keystone 2 SoC supports DMA coherency by hardware. MSMC has a special snooping mechanism to monitor bus transactions and communicates to Tetris to update caches. That eliminates necessity to invalidate cache from software. Several conditions must be met in order to support DMA coherency:

- MMU is enabled
- LPAE is enabled
- Linux runs in DDR3A 0x800000000 address space
- US bit in MSMC MPAX Segment registers has to be cleared to enable page sharing and coherency actions. Please read Multicore Shared Memory Controller [17] for details.
- Dma-coherent property is set for appropriate node in the DTS.

The MCSDK release meets those requirements because:

- Linux always runs with MMU enabled.
- Kernel has LPAE enabled by default in this release.
- U-boot in arch_cpu_init() calls share_all_segments() functions for ARM and NetCP MPAXes, which clears US bits.
- The default mem_lpae u-boot enviroment variable is set to "1" which makes kernel to work at 0x800000000 DDR3A address space.
- The "dma-coherent" property is set for "pktdma" and "netcp" nodes of tci6638_evm.dts

RT Preempt patch

RT Preempt patch is supported in Linux. The master-rt branch in the Linux repo maintains a patched kernel. keystone2_fullrt_defconfig is used to configure the RT Preempt patched kernel. More details on the RT Preempt patch can be obtained from [18]

Capture kernel crash dump using kexec

When there is a kernel panic, and the system goes down there is a need to capture the coredump before resetting the system. This coredump could be later analyzed to debug the kernel panic and for other purposes. To achieve this with Keystone II devices, a few user space utilities were added as part of the file system. The following sections provides details on what these utilities are and how they were used to enable this feature.

Kexec is a patch to the Linux kernel that allows you to boot directly to a new kernel from the currently running one. In the boot sequence, kexec skips the entire bootloader stage (the first part) and directly jumps into the kernel that we want to boot to. There is no hardware reset, no firmware operation, and no bootloader involved.

Note: kernel crash dump is currently supported only on K2HK EVM for this release

Kexec/Kdump Terminology

Some terms that will be used later on in this section are explained here:

Main-kernel: The kernel that is being used for regular use by the system.
Crash-kernel: The kernel that will be used for bringing up the system at the time of a kernel crash.

Crash-kernel-device tree: The device tree file that will be used by the crash kernel at the time of boot up after a panic from main kernel.

Kexec/Kdump User Space tools

Kexec-tools provides the /sbin/kexec binary that facilitates a new kernel to boot using the kernel's kexec feature either on a normal or a panic reboot. This package contains the /sbin/kexec binary and ancillary utilities that together form the userspace component of the kernel's kexec feature.

We use this tool only at the time of a kernel panic, but like the description says this can be used even to load a different kernel at a particular point in time.

User space kexec tools (version used is v2.0.7) are available in gitweb at: Kexec Git web [19]

We also need to apply a [[patch][20]] on kexec tools v2.0.7 to resolve the address aliasing issue. Keystone II device doesn't have physical memory at the first 4GB address range. In order to access the memory at boot time it aliases two GB of the memory to the that range. So when kexec creates elf header for the crash kernel, it has to convert Keystone II physical addresses to the corresponding aliased ones.

Kexec Usage

To use kexec to load a different kernel at the time of panic, do this: kexec -p <path-to-zImage> --command-line=<boot command line> --dtb=<path-to-device-tree-file>

The -p option indicates that this rule will be applicable at the time of a panic only. The path to zImage is the zImage that will be loaded at the time of a panic, this is called crashkernel. The path to device tree is the device tree file that this crashkernel will use for boot up. The boot command line will provide the command line argument for the new crashkernel to come up.

More options for kexec can be seen by doing: "kexec --help" from the shell prompt.

Reserving size of crashkernel

For kexec-tools to work, a separate area of memory will need to be used to load the crashkernel. This is done by adding "crashkernel=33M@0x810000000" to the bootargs of the main kernel's bootargs. If this is not done, the "kexec -p" command descibed previously will not work and will return an error saying crashkernel size is not allocated.

/proc/vmcore

When the main kernel panics, the crashkernel is triggered and when the crashkernel comes up, the /proc/vmcore entry will show the exact state of the previous kernel's view of the system. Capturing this file is needed to analyze the previous kernel panic. But this file will be really huge (because this is in effect, the previous kernel's view of the system). So there are some techniques done to get the exact coredump file that we want which are discussed in the next sections.

makedumpfile

With kexec/kdump, the memory image of the first kernel (called "main kernel") can be taken as /proc/vmcore while the second kernel (called "crash kernel") is running. makedumpfile makes a small DUMPFILE from the /proc/vmcore by compressing dump data or by excluding unnecessary pages for analysis, or both. makedumpfile needs the first kernel's debug information, so that it can distinguish unnecessary pages by analyzing how the first kernel uses the memory.

More information on makedumpfile can be found at: makedump file [21]

There is one particular usage of makedumpfile that we will be employing to capture relevant detail:

```
   makedumpfile –E –d 31 /proc/vmcore/ coredump
   the -E option means the output file coredump will be in ELF format.
(The default is in "crash" format, which needs "crash" utility)
   -d 31 means all zero pages, cache pages, cache private, user and
free pages are removed when generating the coredump file. This is
needed otherwise the coredump file will become really huge.
```

makedumpfile option	Description for "-d"
1	Zero pages
2	Cache pages
4	Cache private
8	User pages
16	Free pages

Refer for more information: makedumpfile options [22]

Crashkernel and crash kernel device-tree

Our kernel uImage file is relocatable and automatically relocates to 0x80008000, so to be used for crashkernel the zImage is preferred and tested.

At the time of kernel panic, the idea is that none of the hardware modules are reliable - because there may be some data traffic happening, and file system may be also be corrupted. So the idea is to bring up a crashkernel that does not have any of the hw modules enabled, and crash kernel will use a new filesystem. The "k2hk_evm_recovery_defconfig" has all TI_KEYSTONE disabled, networking disabled etc. Correspondingly we need a device tree file also to be used by the crashkernel. This device tree file need to provide the bootargs for the crashkernel, and listthe "elfcorehdr=xxx" for the crashkernel to generate the /proc/vmcore entry.

This device tree source file is under <linux-repo>/arch/arm/boot/dts/k2hk-evm-recovery.dts.

The crashkernel can be built from our Linux kernel repo, by doing:
make keystone2_recovery_defconfig
make
make <device>-evm-recovery.dtb

The output zImage is in <linux-repo>/arch/arm/boot

Normally when the kernel is booted by the u-boot, the u-boot takes care to reserve free memory space in the DTB, which is required for the kernel. User has to add the free space manually using dtc complier:

dtc -I dtb -O dtb -p 2048 -o <device>-evm-recovery.dtb arch/arm/boot/dts/k2hk-evm-recovery.dtb

The output DTB is in <linux-repo>/

Note: <device> is k2hk, k2l or k2e

Init-script to run in the crash kernel's filesystem

The crash kernel will use a filesystem and the main expectation from this filesystem is to extract the vmcore information to a elf coredump. For this purpose, an init-script is written which gets executed automatically from the recovery-filesystem. It looks like below:

```
#!/bin/sh
if [ -f /proc/vmcore ] ; then
  mkdir -p /mnt/boot2
  mount -o rw,sync -t ubifs /dev/ubi0_2 /mnt/boot2
  if [ $? -eq 0 ] ; then
     if [ -f /usr/bin/makedumpfile ] ; then
       makedumpfile -E -d 31 /proc/vmcore /mnt/boot2/home/root/coredump.elf
       gzip -c /mnt/boot2/home/root/coredump.elf > /mnt/boot2/home/root/coredump.elf.gz
       rm -rf /mnt/boot2/home/root/coredump.elf
       sync
       reboot
     else
       echo "makedumpfile not found"
       exit 1
     fi
  else
    echo "mount unsuccessful"
    exit 1
  fi
fi
exit 0
```

Note: This script is part of the rootfs-recovery filesystem by default under: /etc/rc3.d/S09recoveryfs

How to enable crashdump kernel with MCSDK on K2HK-EVM

The default root-filesystem of SC-MCSDK contains the kexec and kdump user space utilities under /usr/sbin. The crashkernel and crash kernel's device tree binary file is pre-built and located under /usr/bin/crashdump. This kernel is built based on the "k2hk_evm_recovery_defconfig" located under arch/arm/configs of the linux repo. This device tree file is built based on the "k2hk-evm-recovery.dts" located under <linux-repo>/arch/arm/boot/dts/.

To verify the kexec way of loading a crashkernel and capturing main kernel panic information, use steps below:

From a UBI based NAND filesystem, do the following:

```
Boot the system with the following text added to the args_all line in u-boot:

"crashkernel=33M@0x810000000"

Boot the system.

cd /usr/bin/crashdump
```

For k2hk or k2e:

```
 kexec -p zImage --command-line="console=ttyS0,115200n8 earlyprintk
rootwait=1 maxcpus=1 rootfstype=ubifs root=ubi0:rootfs-recovery
rootflags=sync rw ubi.mtd=2,2048" --dtb=<device>-evm-recovery.dtb
```

For k2l:

```
 kexec -p zImage --command-line="console=ttyS0,115200n8 earlyprintk
rootwait=1 maxcpus=1 rootfstype=ubifs root=ubi0:rootfs-recovery
rootflags=sync rw ubi.mtd=2,4096" --dtb=<device>-evm-recovery.dtb
```

Note: <device> is k2hk, k2l or k2e.

```
At this time, the main kernel is configured to use the zImage as the crash kernel.
Now invoke a panic, by manually building a simple panic kernel module.
```

A sample panic.c which can be built as kernel module is shown below:

```
/*
 * File name: panic.c
 */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sort.h>
#include <linux/bsearch.h>
#include <linux/xfrm.h>
#include <net/net_namespace.h>
#include <linux/list.h>
#include <linux/hash.h>
MODULE_LICENSE("GPL v2");
static int __init panic_init(void)
{
      printk("calling panic()\n");
      panic("panic has been called");
```

```
       return 0;
 }
 module_init(panic_init);
```

Now do:

```
 insmod panic.ko
```

This will reboot the board and:

1. Come up to the new dump-kernel/recovery-fs (because the crash kernel device tree file uses the recovery filesystem)
2. Automatically copy the elf-core-dump (the initscripts of recovery filesystem checks for /proc/vmcore entry and generates the coredump elf)
3. Reboots back to the original kernel/fs. (Again, the initscript of recovery fs does a reboot after copying the coredump)

Now: from /home/root directory, check if there is a coredump.elf.gz. This can be transferred to the linux pc, and analyzed with gdb against a vmlinux file.

```
 arm-linux-gnueabihf-gdb <vmlinux-file> --core=coredump.elf
```

Graceful Power shutdown

This is a sample implementation to support graceful power shutdown. To have a complete implementation, it is assumed that user space daemon listen to the power button event and initiate shutdown.

Smart Reflex Class 0

Please refer to Smart Reflex Class 0 software User's Guide [23] for details.

EVM Setup

This is the sample configuration used for testing the graceful power shutdown use case. The DTS bindings may be altered for the individual GPIO pins used for the target board and also others as needed for the specific usecase.

```
       GPIO pin 1 (power button)
BMC  ------------------> SOC
```

```
       GPIO pin 2 (shutdown complete - power off)
BMC  <------------------ SOC
```

Tests

Used BMC version - 1.0.2.5 Use Linux kernel image with patches applied Boot up the EVM

test power button event reception at the user space

---

step 1. type the following command at the BMC to enable hwdbg

```
[00:00:07]  BMC>hwdbg cmd gpio show
[00:00:07]  Executing command "hwdbg"
[00:00:07]  Enabling command: gpio
```

step 2. type the following command at the Linux console to wait for the power button event.

```
root@keystone-evm:~# evtest /dev/input/event0
Input driver version is 1.0.1
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio_keys.7"
Supported events:
 Event type 0 (Sync)
 Event type 1 (Key)
 Event code 116 (Power)
Testing ... (interrupt to exit)
```

step 3. type the following command at the BMC to toggle GPIO pin 1 to high.

```
BMC>gpio xa  XXXX_XXXXXX_XX1X
[19:26:21]  Executing command "gpio"
[19:26:21]  Writing to GPIO expander A...
[19:26:21]  Inputs: 0x0000
High Outputs: 0x0002
Low Outputs: 0x0000
[19:26:21]  Finished setting GPIO expander A...
```

Following log will display at the linux console showing the Power button event when GPIO pin is toggled to high

```
Event: time 1373363555.593249, type 1 (Key), code 116 (Power), value 1
Event: time 1373363555.593249, -------------- Report Sync ------------
```

test power off event from SoC to BMC

step 1. type the following command to at the BMC to display the current GPIO pin 2 state

```
[19:37:10]  BMC>gpio xa
[19:37:10]  Executing command "gpio"
[19:37:10]  Reading from GPIO expander A...
[19:37:10]  Inputs:  0000_0000_0000_0001
[19:37:10]  Outputs: XXXX_XXXX_XXXX_XXXX
```

step 2. Initiate shutdown command from user space by typing following command at the Linux console

```
root@keystone-evm:~# shutdown -P -h "now"
```

```
INIT: Sending processes the TERM signalm (ttyS0) (Tue Jul  9 10:02:20 2013):
INIT:Stopping telnet daemon.
Stopping tiipclad daemon.
Stopping syslogd/klogd: stopped syslogd (pid 1586)
stopped klogd (pid 1589)
done
Stopping thttpd.
NOT deconfiguring network interfaces: / is an NFS mount
Sending all processes the TERM signal...
Sending all processes the KILL signal...
Unmounting remote filesystems...
Deactivating swap...
Unmounting local filesystems...
[  460.561642] Power down.
```

step 3. type the following command to at the BMC to display the current GPIO pin 2 state again

```
[19:37:12]  BMC>gpio xa
[19:37:12]  Executing command "gpio"
[19:37:12]  Reading from GPIO expander A...
[19:37:12]  Inputs:  0000_0000_0000_0101
[19:37:12]  Outputs: XXXX_XXXX_XXXX_XXXX
```

Note that pin2 state is toggled to 1 (in step.1 it was showing 0)

configuration

The feature is enabled by default in the default image shipped with release. To get this working on a board, user need to identify the GPIO pins used and add following bindings to the DTS file

```
              gpio_poweroff {
                      compatible = "gpio-poweroff";
                      /*
                       * Change this as needed in the target board to match the
                       * pin used for signaling Power down event from the SoC.
                       * It is assumed that external controller reads this pin
                       * and remove power to the SoC. On EVM, pin 4 output
                       * can be read from BMC to test this.
                       */
                      gpios = <&gpio0 4 0>;
              };
```

```
              gpio_keys {
                      compatible = "gpio-keys";
                      #address-cells = <1>;
                      #size-cells = <0>;
```

```
                      button@1 {
                              label = "Power button";
                              /* this is the key power button defined in Linux input.h */
                              linux,code = <116>;
                              /* EV_KEY */
                              linux,input-type = <1>;
                              /*
                               * On EVM use GPIO pin 3 since it is also connected to
                               * BMC to generate a shutdown event to ARM. Change this
                               * as needed in the target board.
                               */
                              gpios = <&gpio0 3 0>;
                      };
              };
```

Yocto

MCSDK uses Yocto projet to build the Linux kernel, U-boot, user space components, and root filesystem.

Prerequisite

To build yocto in an ubuntu Linux machine, first get a library for 32-bit compilation:

```
sudo apt-get -y install gcc-multilib libc6-i386
```

Next, the following tools should be installed:

```
sudo apt-get -y install diffstat texi2html texinfo subversion chrpath build-essential subversion ccache sed wget cvs coreutils unzip texinfo docbook-utils gawk help2man file g++ bison flex htmldoc chrpath libxext-dev xserver-xorg-dev doxygen socat uboot-mkimage git
```

Note: The apt-get steps only needs to be done once.

Configuration

The following steps set up a default configuration that may be customized as needed:

NOTE: The first step is changing for the MCSDK 3.0.4 POST GA release. If using release earlier releases use the old procedure under: oe-layersetup for older releases.

- Clone the oe-layersetup.git (git://arago-project.org/git/projects/oe-layersetup.git) from Arago project

  $ git clone http://arago-project.org/git/projects/oe-layersetup.git

  $ cd oe-layersetup

oe-layersetup for older releases

- Clone the oe-layersetup-mcsdk.git (http://arago-project.org/git/people/hzhang/oe-layersetup-mcsdk.git) from Arago project

  $ git clone http://arago-project.org/git/people/hzhang/oe-layersetup-mcsdk.git mcsdk

  $ cd mcsdk

- Choose the configuration file based on the release used.

  The configuration file for the mcsdk release is kept at configs/mcsdk/mcsdk-<version>-config.txt

  The latest mcsdk release configuration file is configs/mcsdk/mcsdk-03.01.03.06-config.txt

  Note: During development process a different configuration file may be used.

- Run the setup script to configure oe-core layers and builds (Note to use the right config file here)

  $ ./oe-layertool-setup.sh -f configs/mcsdk/mcsdk-03.01.03.06-config.txt

---

Note: For mcsdk 3.1.4.7: In addition to the config script, need to run the following steps

$ cd sources/oe-core

$ git cherry-pick 3fa24eee41c26fecd5e4f680082288ec772d2de9

$ cd -

Build

This section describes how the linux rootfs filesystem can be built.
This section assumes that the Linaro toolchain for ARM is installed and environment variables CROSS_COMPILE, ARCH are set up as per instructions given in section Toolchain Installation Linaro_toolchain [4].

```
export CROSS_COMPILE=arm-linux-gnueabihf-
export ARCH=arm
PATH=<path to installed toolchain>/bin:$PATH
```

To build the linux rootfs filesystem, one can either use the local snapshot of the sources in MCSDK release, or fetch the latest source packages at external URLs.

Using the snapshot of the source packages in MCSDK release

Using the snapshot of the arago source packages can avoid fetch errors during the build when external URLs become unavailable. To use the snapshot of sources distributed with MCSDK release, for example, see mcsdk-arago-source [24]. Once this package is downloaded, the content can be un-tarred under the directory $(INSTALL_DIR)/downloads. With this, the build will use local source packages in the downloads directory instead of fetching the latest from external URLs.

- un-tar the arago source tarball

```
$ cd oe-layersetup
$ ls
        build configs oe-layertool-setup.sh sample-files sources
$ tar xzvf $(DOWNLOAD_DIR)/mcsdk-3_##_##_##.arago.src.tar.gz
$ ls
        build configs downloads oe-layertool-setup.sh sample-files sources
```

- Then, follow all the steps of using the latest source packages on internel URLs below to build

Note: for MCSDK 3.0.4, the build version is 3_00_04_18; for MCSDK 3.1.0, the build version is 3_01_00_03; for MCSDK 3.1.1, the build version is 3_01_01_04; for MCSDK 3.1.2, the build version is 3_01_02_05; for MCSDK 3.1.3, the build version is 3_01_03_06.

Build procedure

- Set up the environment variables and start the build using the configuration file

```
$ cd build
$ source conf/setenv
$ MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake tisdk-server-rootfs-image
```

For releases older than MCSDK 3.0.4, ARAGO_BRAND=mcsdk is not required. And also use <soc> as "keystone".

For the releases MCSDK 3.0.4 and after use: <soc> is "k2hk", "k2l" or "k2e";

Once build is complete, the images for the kernel and file system can be located at

```
$(INSTALL_DIR)/build/arago-tmp-external-linaro-toolchain/deploy/images
```

- If needed, edit the bitbake configuration file conf/local.conf to customize your bitbake build. In particular, enabling parallel task execution substantially speeds up the Arago build process by fetching package sources in parallel with other build steps. In order to do so, please enable and edit the following parameter in your bitbake configuration BB_NUMBER_THREADS = "4"

Note: The BB_NUMBER_THREADS definition is not the number of CPUs on your SMP machine. The value of 4 appears to work quite well on a single core system, and may be adjusted upwards on SMP systems.

Note: If you are developing applications outside Yocto, you can point the your build's sysroot to the sysroot, Yocto has generated using above steps. You can add the following to your build option *--sysroot=$(INSTALL_DIR)/build/arago-tmp-external-linaro-toolchain/sysroots/keystone-evm* to pick up the sysroot, the Yocto build has created.

Note: For MCSDK 3.00.04.18 release, in the file: ./sources/meta-mcsdk/recipes-core/rhino/rhino_1.7r4.bbappend file: The SRC_URI needs to be changed from: ftp:/ / ftp. freebsd. org/ pub/ FreeBSD/ ports/ distfiles/ rhino/ rhino1_7R4.zip to http://distcache.freebsd.org/ports-distfiles/rhino/rhino1_7R4.zip

Building other components in Yocto

There are other open source components for which are recipes are available in the yocto build system. And customers may choose to build them and add it to their file system.

Any open source component for the platform, if available and compatible with the platform, can be built for the using the following bitbake command similar the build procedure for the filesystem.

MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake <name_of_recipe_without_version and file extension>

For releases older than MCSDK 3.0.4, ARAGO_BRAND=mcsdk is not required. And also use <soc> as "keystone".

For the releases MCSDK 3.0.4 and after use: <soc> is "k2hk", "k2l" or "k2e";

(e.g) MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake sudo

This depending on the recipe will create the ipks under the appropriate directory structure under

$(INSTALL_DIR)/build/arago-tmp-external-linaro-toolchain/deploy/ipk

Updating a user space component in MCSDK filesystem

The following procedure can be used to update a user space component already available in MCSDK filesystem.
Prerequisite: Follow the procedure to create the build setup for Yocto build for the base MCSDK release.

Step1: Update the component recipe

Locate the component recipe under the different YOCTO layers/git repositories cloned under oe-layersetup/sources/*.
And modify locally or patch the recipe.
(And if needed create a patch to keep it for reference, to help recreate the build)

Step 2: Build the component

(Assuming the arm tools are setup based on yocto build instructions)

```
$ cd build
$ . conf/setenv
$ MACHINE=<soc>-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake <component_recipe_name>
```

Eg.

```
MACHINE=k2hk-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake ti-pa-bin
```

(Note with this build complete, installable user space packages for the module will be created under: oe-layersetup/build/arago-tmp-external-linaro-toolchain/deploy/ipk/cortexa15hf-vfp-neon-3.8 with name <module-name>*.ipk.
e.g ti-pa-bin_03.00.00.10-r0_cortexa15hf-vfp-neon-3.8.ipk : This is the main package to be used with filesystem.
ti-pa-bin-dbg_03.00.00.10-r0_cortexa15hf-vfp-neon-3.8.ipk : This is debug package (Normally not used).
ti-pa-bin-dev_03.00.00.10-r0_cortexa15hf-vfp-neon-3.8.ipk : This is dev package to be used with devkit.
)

Step 3: Add package to filesystem

There are two ways this can be added to the filesystem.
On the host machine:
- Untar the filesystem in a host directory. (e.g)

```
mkdir rootfs
cd rootfs
sudo tar xzf  ../tisdk-rootfs-<k2e|k2h|k2l>-evm.tar.gz
```

- Install the package into the filesystem.

```
dpkg -x <dirname>/<component_recipe_name>_*.ipk <dirname>/rootfs
```

(e.g)

```
dpkg -x  ~/ti-pa-bin*.ipk ~/rootfs
```

- Repackage the file system

```
sudo tar czvf  . ../ tisdk-rootfs-<k2e|k2h|k2l>-evm-modified.tar.gz
```

(Now this filesytem can be used to create the UBI image etc).
On the target:
- Tftp the *.ipks into the filesytem. And use the following command to install it

```
opkg install <component_recipe_name>_*.ipk
```

e.g:

```
opkg install ti-pa-bin_*.ipk
```

FAQ

1. Error in coying toolchain files:

```
| DEBUG: Executing shell function do_install

| cp: cannot stat '/opt/linaro-3.1.0.3/sysroots/i686-arago-linux/usr/arm-linux-gnueabihf/libc/usr/share/*': No such file or directory
```

Make sure the default shell is bash. If not, please use the following command:
(to choose bash instead of dash)

```
sudo dpkg-reconfigure dash
```

Flattened Device Tree

The Device Tree is a data structure for describing hardware. Rather than hard coding every detail of a device into an operating system, many aspect of the hardware can be described in a data structure that is passed to the operating system at boot time. The device tree is used both by Open Firmware, and in the standalone Flattened Device Tree (FDT) form. For more details refer [[25]]

creating dtb image

Kernel tree has device tree compiler and can be build as part of the kernel build process. DTB can also be built using the dtc compiler part of the rootfs that is shipped with the kernel. Do the following to copy the dts files to rootfs (either on the Host nfs root directory or tftp to the EVM from Linux console) and build dtb

```
git clone git://git.ti.com/keystone-linux/linux.git linux-keystone
cd linux-keystone
git reset --hard <release tag>
```

The release tag is obtained from the Release notes. Copy arch/arm/boot/dts/k2hk-evm.dts and arch/arm/boot/dts/skeleton.dtsi to the rootfs (either on Host nfs root directory if using nfs rootfs or rootfs on NAND on EVM if using rootfs on NAND)

Once the Linux booted up with the released image, do the following:-

```
cd <path where the dts files are copied>
dtc -I dts -O dtb -o uImage-k2hk-evm.dtb k2hk-evm.dts
```

Copy the uImage-k2hk-evm.dtb to the tftp server or boot volume of the UBI depending on the rootfs used and reboot the EVM.

UBI/UBIFS

UBI

UBI (Latin: "where?") stands for "Unsorted Block Images". It is a volume management system for raw flash devices which manages multiple logical volumes on a single physical flash device and spreads the I/O load (i.e, wear-leveling) across whole flash chip.

In a sense, UBI may be compared to the Logical Volume Manager (LVM). Whereas LVM maps logical sectors to physical sectors, UBI maps logical eraseblocks to physical eraseblocks. But besides the mapping, UBI implements global wear-leveling and transparent I/O errors handling.

An UBI volume is a set of consecutive logical eraseblocks (LEBs). Each logical eraseblock may be mapped to any physical eraseblock (PEB). This mapping is managed by UBI, it is hidden from users and it is the base mechanism to provide global wear-leveling (along with per-physical eraseblock erase counters and the ability to transparently move data from more worn-out physical eraseblocks to less worn-out ones).

UBI volume size is specified when the volume is created and may later be changed (volumes are dynamically re-sizable). There are user-space tools which may be used to manipulate UBI volumes.

There are 2 types of UBI volumes - dynamic volumes and static volumes. Static volumes are read-only and their contents are protected by CRC-32 checksums, while dynamic volumes are read-write and the upper layers (e.g., a file-system) are responsible for ensuring data integrity.

UBI is aware of bad eraseblocks (e.g., NAND flash may have them) and frees the upper layers from any bad block handling. UBI has a pool of reserved physical eraseblocks, and when a physical eraseblock becomes bad, it transparently substitutes it with a good physical eraseblock. UBI moves good data from the newly appeared bad physical eraseblocks to good ones. The result is that users of UBI volumes do not notice I/O errors as UBI takes care of them.

NAND flashes may have bit-flips which occur on read and write operations. Bit-flips are corrected by ECC checksums, but they may accumulate over time and cause data loss. UBI handles this by moving data from physical eraseblocks which have bit-flips to other physical eraseblocks. This process is called scrubbing. Scrubbing is done transparently in background and is hidden from upper layers.

Here is a short list of the main UBI features:

- UBI provides volumes which may be dynamically created, removed, or re-sized;
- UBI implements wear-leveling across whole flash device (i.e., you may continuously write/erase only one logical eraseblock of an UBI volume, but UBI will spread this to all physical eraseblocks of the flash chip);

- UBI transparently handles bad physical eraseblocks;
- UBI minimizes chances to lose data by means of scrubbing.

For more information on UBI, refer [[26][UBI]]

UBIFS

UBIFS may be considered as the next generation of the JFFS2 file-system.

JFFS2 file system works on top of MTD devices, but UBIFS works on top of UBI volumes and cannot operate on top of MTD devices. In other words, there are 3 subsystems involved:

MTD subsystem, which provides uniform interface to access flash chips. MTD provides an notion of MTD devices (e.g., /dev/mtd0) which basically represents raw flash UBI subsystem, which is a wear-leveling and volume management system for flash devices. .UBI works on top of MTD devices and provides a notion of UBI volumes; UBI volumes are higher level entities than MTD devices and they are devoid of many unpleasant issues MTD devices have (e.g., wearing and bad blocks); For more information on MTD, refer [[27][MTD]]

For more information on UBIFS, refer [[28][UBIFS]]

UBIFS User-space tools

UBI user-space tools, as well as other MTD user-space tools, are available from the the following git repository: git://git.infradead.org/mtd-utils.git

The repository contains the following UBI tools:

- ubinfo - provides information about UBI devices and volumes found in the system
- ubiattach - attaches MTD devices (which describe raw flash) to UBI and creates corresponding UBI devices
- ubidetach - detaches MTD devices from UBI devices (the opposite to what ubiattach does)
- ubimkvol - creates UBI volumes on UBI devices
- ubirmvol - removes UBI volumes from UBI devices
- ubiupdatevol - updates UBI volumes. This tool uses the UBI volume update feature which leaves the volume in "corrupted" state if the update was interrupted; additionally, this tool may be used to wipe out UBI volumes.
- ubicrc32 - calculates CRC-32 checksum of a file with the same initial seed as UBI would use
- ubinize - generates UBI images
- ubiformat - formats empty flash, erases flash and preserves erase counters, flashes UBI images to MTD devices
- mtdinfo - reports information about MTD devices found in the system.

All UBI tools support "-h" option and print sufficient usage information.

NAND Layout

The NAND flash in the EVM contains three partitions:-

- bootloader - Contains u-boot
- params - contains env variables
- ubifs - contains following UBI volumes:-
  - boot volume - contains Kernel image (uImage), device tree blob etc,
  - rootfs volume - contains the rootfs which is the primary filesystem

Note: bootloader partition is blank, u-boot is stored on SPI NOR flash.

Compiling UBIFS Tools

The MTD and UBI user-space tools are available from the the following git repository: git://git.infradead.org/mtd-utils.git

Suggest using 1.4.8 of the mtd-utils

For instructions on compiling MTD-utils, refer [[29][MTD-Utils Compilation]]. In the instruction for building mtd-utils, please replace PREFIX with INSTALL_DIR. The makefile doesn't like the use of PREFIX variable and result in build error. This is a work around to fix the build error.

Creating UBIFS file system

For information on how to create a UBIFS image. refer [[30]]

- mkfs.ubifs

```
mtd-utils# mkfs.ubifs -r filesystem/ -o ubifs.img -F -m 2048 -e 126976 -c 936
```

Where

- -m 2KiB (or 2048). The minimum I/O size of the underlying UBI and MTD devices. In our case, we are running

  the flash with no sub-page writes, so this is a 2KiB page.
- -e 124KiB (or 126976) Erase Block Size: UBI requires 2

  minimum I/O units out of each Physical Erase Block (PEB) for overhead: 1 for maintaining erase count information, and 1

  for maintaining the Volume ID information. The PEB size for the Micron flash is 128KiB, so this leads to each Logical

  Erase Block (LEB) having 124KiB available for data.
- -c 936 The maximum size, in LEBs, of this file system. See calculation below for how this number is determined.
- -r filesystem. Use the contents of the 'filesystem/' directory to generate the initial file system image.
- -o ubifs.img Output file.
- -F parameter is used to set the "fix up free space" flag in the superblock, which forces UBIFS to "fixup" all the free space which it is going to use.

The output of the above command, ubifs.img is fed into the 'ubinize' program to wrap it into a UBI image.

Creating UBI image

The images produced by mkfs.ubifs must be further fed to the ubinize tool to create a UBI image which must be put to the raw flash to be used a UBI partition.

- Create ubinize.cfg file and write the contents into it

```
mtd-utils# vi ubinize.cfg
[ubifs_rootfs_volume] <== Section header
 mode=ubi <== Volume mode (other option is static)
 image=ubifs.img <== Source image
 vol_id=0 <== Volume ID in UBI image
 vol_size=113MiB <== Volume size
 vol_type=dynamic <== Allow for dynamic resize
 vol_name=rootfs <== Volume name
 vol_flags=autoresize <== Autoresize volume at first mount [See calculations below to determine the value
 associated with 'vol_size']
```

Note: There is a sample ubinize.cfg under mcsdk_linux_3_00_00_xx/images directory which is used to create the MCSDK ubi image.

- ubinize

```
mtd-utils# ubi-utils/ubinize -o ubifs.ubi -m 2048 -p 128KiB -s 2048 -O 2048 ubinize.cfg
```

Where:

- -o ubifs.ubi Output file
- -m 2KiB (or 2048) Minimum flash I/O size of 2KiB page
- -p 128KiB Size of the physical eraseblock of the flash this UBI image is created for
- -s 2028 Use a sub page size of 2048 -O 2048 offset if the VID header from start of the physical eraseblock

The output of the above command, 'ubifs.ubi' is the required image.

Calculations

Usable Size Calculation As documented here, UBI reserves a certain amount of space for management and bad PEB handling operations. Specifically:

- 2 PEBs are used to store the UBI volume table
- 1 PEB is reserved for wear-leveling purposes;
- 1 PEB is reserved for the atomic LEB change operation;
- a % of PEBs is reserved for handling bad EBs. The default for NAND is 1%
- UBI stores the erase counter (EC) and volume ID (VID) headers at the beginning of each PEB.
- 1 min I/O unit is required for each of these.

To calculate the full overhead, we need the following values:

Symbol	Meaning	value
SP	PEB Size	128KiB
SL	LEB Size	128KiB - 2 * 2KiB = 124 KiB
P	Total number of PEBs on the MTD device	126.5MiB / 128KiB = 1012
B	Number of PEBs reserved for bad PEB handling	1% of P = 10.12(round to 10)
O	The overhead related to storing EC and VID headers in bytes, i.e. O = SP - SL	4KiB

Assume a partition size of 126.5M (We use 1.5M for bootloader and params leaving 126.5M for ubifs partition)

UBI Overhead = (B + 4) * SP + O * (P - B - 4) = (10 + 4) * 128Kib + 4 KiB * (1012 - 9- 4) = 5784 KiB = 45.1875 PEBs (round to 45)

This leaves us with 967 PEBs or 123776 KiB available for user data.

Note that we used "-c 998 " in the above mkfs.ubifs command line to specify the maximum filesystem size, not "-c 967" The reason for this is that mkfs.ubifs operates in terms of LEB size (124 KiB), not PEB size (128Kib). 123776 KiB / 124 Kib

```
= 998.19 (round to 998).
```

Volume size = 123776 KiB (~120 MiB)

Use this caculation method to calculate the size required for each ubifs image going into each of the ubifs volumes on NAND.

A sample ubinize.cfg file is included in the images folder of the release. This was used to create the keystone-evm-ubifs.ubi image. There is also a keystone-evm-boot.ubifs that is part of the release images folder. This was created as follows: Use the mkfs.ubifs and ubinize utilities provided with the release (under bin folder)

```
cd <release_folder>/mcsdk_linux_<version>/images
mkdir boot
cp uImage-keystone-evm.bin boot/
cp uImage-k2hk-evm.dtb boot/
cp skern-keystone-evm.bin boot/
export PATH=<release_folder>/mcsdk_linux_<version>/bin:$PATH
mkfs.ubifs -r boot -F -o keystone-evm-boot.ubifs -m 2048 -e 126976 -c 41
cp keystone-evm-boot.ubifs images/
cd images/
ubinize -o keystone-evm-ubifs.ubi -m 2048 -p 128KiB -s 2048 -O 2048 ubinize.cfg
```

Using UBIFS file system

Preparing NAND partition Kindly erase the NAND partition before using it for UBI file system. The partition can be erased from either u-boot or from Linux.

Follow below steps to erase.

From U-boot,

1. Reset environment variables to default values and set the boot mode to ubi:

```
u-boot# env default -f -a
u-boot# setenv boot ubi
```

Note: On Rev 1.0 or older EVMs, the NAND part used is 128M. So following commands needed to set mtdparts

```
u-boot# setenv mtdparts 'mtdparts=davinci_nand.0:1024k(bootloader)ro,512k(params)
        ro,129536k(ubifs)'
```

2. Set serverip and tftp_root env variables:

```
u-boot# setenv serverip <TFTP server IP address>
u-boot# setenv tftp_root <directory under TFTP root>
u-boot# saveenv
```

3. Create ${tftp_root} directory under TFTP root and copy mcsdk_linux_3_00_00_xx/images/keystone-evm-ubifs.ubi to ${tftp_root} directory

4. TFTP keystone-evm-ubifs.ubi to DDR3 memory and write ubi image from DDR3 to ubifs partition on NAND:

```
u-boot# setenv addr_file 0x82000000
u-boot# nand erase.part ubifs
u-boot# dhcp ${addr_file} ${tftp_root}/keystone-evm-ubifs.ubi
u-boot# nand write ${addr_file} ubifs ${filesize}
```

or use the automated scripts

```
u-boot# run get_ubi_net
u-boot# run burn_ubi
```

Note that the above assumes name_ubi env variable is set to "keystone-evm-ubifs.ubi". If you are using rt version of the ubi image, change name_ubi env variable to match with the rt file name.

Note: the file size of the keystone-evm-ubifs.ubi is printed on console when tftp is completed.

Note: If you do not have a DHCP server on your network, you may assign a static IP using the U-Boot command "setenv ipaddr <ipadd>".

Note: if running on Rev 1.0 or older EVMs, please update dts for correct mtd partition size to account for 128M NAND available as follows:-

Also DTS file needs to be modified for the size of NAND as well as follows:-

```
   nand@2,0 {
        .....
        partition@180000 {
            label = "ubifs";
            reg = <0x180000 0x7e80000>;
        };
   };
```

Rebuild DTB and create a boot ubifs image (keystone-evm-boot.ubifs) using the new DTBas described at the end of the section creating boot ubifs. Then remove and create a new boot volume on UBI from u-boot using following commands

```
u-boot# dhcp ${addr_file} ${tftp_root}/keystone-evm-boot.ubifs
u-boot# ubi part ubifs
u-boot# ubifsmount boot
u-boot# ubi remove boot
u-boot# ubi create boot [size in hex of the ubifs image]
u-boot# ubi write $addr_file} boot [size in hex of the ubifs image]
```

From Linux. Assuming MTD partition 2 needs to be erased and used for UBI file system.

```
root@arago-armv7:~# flash_eraseall /dev/mtd2
```

Flashing UBIFS image to a NAND partition

We can Flash UBIFS image from either Linux Kernel or U-Boot.

Follow steps mentioned here to create an UBIFS image.

From Linux,

Flash the UBI file system image (ubifs.ubi) to MTD partition "X"

ubiformat /dev/mtd<X> -f ubifs.ubi -s 2028 -O 2048

Assuming 2nd mtd partition, we can use the following command to flash the ubifs ubi image to partition 2.

```
#flash_erase /dev/mtd2 0 0
#ubiformat /dev/mtd2 -f keystone-evm-ubifs.ubi -s 2048 -O 2048
```

Using UBIFS image as root file system

Set up the bootargs environment variable as below to use the UBIFS file system image present in a MTD partition:

```
setenv bootargs 'console=ttyS0,115200n8 mem=512M earlyprintk debug rootwait=1 rw ubi.mtd=X,YYYY rootfstype=ubifs
root=ubi0:rootfs rootflags=sync'
```

Where X is the MTD partition number being used for file system and YYYY is the NAND page size. make sure that an UBI file system is flashed into this partition before passing it as a boot partition for Linux.

Assuming 2nd mtd partition and page size of 2048,

```
setenv bootargs 'console=ttyS0,115200n8 mem=512M earlyprintk debug rootwait=1 rw ubi.mtd=2,2048 rootfstype=ubifs
root=ubi0:rootfs rootflags=sync'
```

Updating Boot volume images from Linux kernel

The files in the boot volume may be removed and replaced by new file and EVM may be rebooted using the new images. See below the steps to replace the file in boot volume.

```
root@keystone-evm:~# mkdir /mnt/boot
root@keystone-evm:~# mount -t ubifs ubi0_0 /mnt/boot
```

```
[ 1337.657081] UBIFS: mounted UBI device 0, volume 0, name "boot"
[ 1337.663070] UBIFS: file system size: 3936256 bytes (3844 KiB, 3 MiB,
 31 LEBs)u
[ 1337.670334] UBIFS: journal size: 1142785 bytes (1116 KiB, 1 MiB, 8
LEBs)
[ 1337.677502] UBIFS: media format: w4/r0 (latest is w4/r0)
[ 1337.683297] UBIFS: default compressor: lzo
[ 1337.687360] UBIFS: reserved for root: 0 bytes (0 KiB)
```

```
root@keystone-evm:~# root@keystone-evm:~# cd /mnt/boot
root@keystone-evm:/mnt/boot# ls
uImage-k2hk-evm.dtb  uImage-keystone-evm.bin skern-keystone-evm.bin
```

The above files can be deleted and overwritten with new file. For example to replace the dtb file, do

```
root@keystone-evm:/mnt/boot# rm uImage-k2hk-evm.dtb
```

TFTP the uImage-k2hk-evm.dtb to this folder or using the DTC compiler on target DTS file may compiled and saved in this folder. Please note that the file name should match with default files in the boot volume.

Once done umount the folder as

```
root@keystone-evm:/# umount /mnt/boot
UBIFS: un-mount UBI device 0, volume 0 root@keystone-evm:/# reboot
```

SYS/BIOS

Placeholder for future

DSP Subsystem

Overview

The Multicore Software Development Kit (MCSDK) provides the core foundational building blocks that facilitate application software development on TI's high performance and multicore SoC. The foundational DSP components include:

- SYS/BIOS which is a light-weight real-time embedded operating system for TI devices
- Chip support libraries, drivers, and basic platform utilities
- Interprocessor communication for communication across cores and devices
- Basic networking stack and protocols
- Optimized application-specific and application non-specific algorithm libraries

- Debug and instrumentation
- Bootloaders and boot utilities
- Demonstrations and examples

The purpose of this *User's Guide* is to provide more detailed information regarding the software elements and infrastructure provided with MCSDK. MCSDK pulls together all the elements into demonstrable multicore applications and examples for supported EVMs. The objective being to demonstrate device, platform, and software capabilities and functionality as well as provide the user with instructive examples. The software provided is intended to be used as a reference when starting their development.

Note: In deciding the endianness of the DSP, all the features using little endian Linux configuration are only supported with Little endian DSP configuration in this release. Although some of the modules are tested standalone with Big endian DSP.

Note: It is expected the user has gone through the *EVM Quick Start Guide (TBD)* provided with their EVM and have booted the out-of-box demonstration application flashed on the device. It is also assumed the user has installed both CCS and the MCSDK.

API and LLD User Guides

API Reference Manuals and LLD User Guides are provided with the software. You can reference them from the Eclipse Help system in CCS or you can navigate to the components *doc* directory and view them there.

Tools Overview

The following documents provide information on the various development tools available to you.

Document	Description
CCS v5 Getting Started Guide [18]	How to get up and running with CCS v5
XDS560 Emulator Information [31]	Information on XDS560 emulator
XDS100 Emulator Information [32]	Information on XDS100 emulator
TMS320C6000 Optimizing Compiler v 7.3 [33]	Everything you wanted to know about the compiler, assembler, library-build process and C++ name demangler.
TMS320C6000 Assembly Language Tools v 7.3 [34]	More in-depth information on the assembler, linker command files and other utilities.
Multi-core System Analyzer [35]	How to use and integrate the system analyzer into your code base.
Eclipse Platform Wizard [36]	How to create a platform for RTSC. The demo uses CCSv4 but the platform screens are the same in CCSv5.
Runtime Object Viewer [37]	How to use the Object Viewer for Eclipse Based Debugging.

Hardware - EVM Overview

The following documents provide information about the EVM.

Document	Description
Introducing the C66x Lite EVM Video [38]	Short video on the C66x Lite Evaluation Module, the cost-efficient development tool from Texas Instruments that enables developers to quickly get started working on designs for the C66x multicore DSPs based on the KeyStone architecture.

Hardware - Processor Overview

The following documents provide information about the processor used on the EVM.

Document	Description
TCI6636K2H Data Manual [39]	Data manual for specific TI DSP

Related Software

This section provides a collection links to additional software elements that may be of interest.

Link	Description
C6x DSP Linux Project [40]	Community site for C6x DSP Linux project
Telecom Libraries [41]	TI software folder for information and download of Telecom Libraries (Voice, Fax, etc) for TI processors.
Medical Imaging Software Tool Kits [42]	TI software folder for information and download of medical imaging software tool kits for TI processors.

Platform Development Kit (PDK)

Chip Support Library (CSL)

The Chip Support Library constitutes a set of well-defined APIs that abstract low-level details of the underlying SoC device so that a user can configure, control (start/stop, etc.) and have read/write access to peripherals without having to worry about register bit-field details. The CSL services are implemented as distinct modules that correspond with the underlying SoC device modules themselves. By design, CSL APIs follow a consistent style, uniformly across Processor Instruction Set Architecture and are independent of the OS. This helps in improving portability of code written using the CSL.

CSL is realized as twin-layer – a basic register-layer and a more abstracted functional-layer. The lower register layer comprises of a very basic set of macros and type definitions. The upper functional layer comprises of “C” functions that provide an increased degree of abstraction, but intended to provide “directed” control of underlying hardware.

It is important to note that CSL does not manage data-movement over underlying h/w devices. Such functionality is considered a prerogative of a device-driver and serious effort is made to not blur the boundary between device-driver and CSL services in this regard.

CSL does not model the device state machine. However, should there exist a mandatory (hardware dictated) sequence (possibly atomically executed) of register reads/writes to setup the device in chosen “operating modes” as per the device data sheet, then CSL does indeed support services for such operations.

The CSL services are decomposed into modules, each following the twin-layer of abstraction described above. The APIs of each such module are completely orthogonal (one module’s API does not internally call API of another module) and do not allocate memory dynamically from within. This is key to keeping CSL scalable to fit the specific usage scenarios and ease the effort to ROM a CSL based application.

The source code of the CSL is located under $(TI_PDK_INSTALL_DIR)\packages\ti\csl directory.

Note:

The CSL is build with LLD using same script, please refer the LLD build section for details

For KeyStone2 Devices CSL Package includes support for multiple devices. Software layer using CSL source would need to pass compile time define -DDEVICE_XXX. Refer ti\csl\cslr_device.h for list of devices/SOC's

Chip Support Library Summary	
Component Type	Library
Install Package	PDK
Install Directory	pdk_keystone2_<version>\packages\ti\csl
Project Type	Eclipse RTSC [43]
Endian Support	Little & Big
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\csl
Linker Sections	.vecs , .switch, .args, .cio
Section Preference	L2 Cache
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\csl
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Chip support library [44]
Downloads	Product Updates
License	BSD [45]

Low Level Drivers

The Low Level Drivers (LLDs) provide ineterfaces to the various peripherals on your SoC Device.

The source code for the LLDs is located under $(TI_PDK_INSTALL_DIR)\packages\ti\drv directory.

The following table shows PDK LLD vs. SoC Availability

Driver	Keystone2
CSL	X
QMSS	X
PKTDMA (CPPI)	X
PA	X
SA	X
SRIO	X
PCIe	X
Hyperlink	X
EDMA3	X
FFTC	X
TCP3d	X
BCP	X
RM	X
DFE	X
IQN2	X
TSIP	X
AIF2	X

Driver Library Summary	
Component Type	Library
Install Package	PDK
Install Directory	pdk_keystone2_<version>\packages\ti\drv
Project Type	Eclipse RTSC [43]
Endian Support	Little & Big
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\drv
Linker Sections	N/A
Section Preference	N/A
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\drv
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Chip support library [44]
Downloads	Product Updates
License	BSD [45]

Multicore Navigator

Multicore Navigator provides multicore-safe communication while reducing load on DSPs in order to improve overall system performance.

Packet DMA (CPPI)

The CPPI low level driver can be used to configure the CPPI block in CPDMA for the Packet Accelerator (PA). The LLD provides resource management for descriptors, receive/transmit channels and receive flows.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [46]
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\cppi\docs\ CPPI_QMSS_LLD_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\cppi\docs\cppilldDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_CPPI_LLD.pdf

Queue Manager (QMSS)

The QMSS low level driver provides the interface to Queue Manager Subsystem hardware which is part of the Multicore Navigator functional unit for a keystone device. QMSS provides hardware assisted queue system and implements fundamental operations such as en-queue and de-queue, descriptor management, accumulator functionality and configuration of infrastructure DMA mode. The LLD provides APIs to get full entitlement of supported hardware functionality.

The LLD also includes accumulation and QoS (Quality of Service) firmware. QoS enables restriction of data rates in bytes per second or packets per second, weighted round robin queue selection, and selective descriptor dropping for oversubscribed queues. Accumulation The APIs are provided through the LLD. The API documentation for both

QoS and Accumulator is available in the API Reference manual below, for all versions of MCSDK. The capabilities of the QoS firmware are documented in their design documents, which are present in MCSDK 3.1.4 and later. The capabilities of the accumulator are documented in the Hardware Peripheral User Guide.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [46]
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\CPPI_QMSS_LLD_SDS.pdf
QoS (Quality of Service) Design Document covering qos_sched_drop_sched, qos_sched, and qos_sched_wide.	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\firmware\qos_sched,qos_sched_drop_sched,qos_sched_wide.pdf
QoS (Quality of Service) Design Document covering qos (leaky bucket and SRIO TX context tracking).	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\firmware\qos.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\qmss\docs\qmsslldDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_QMSS_LLD.pdf

Packet Library (PKTLIB)

Module expands underlying CPPI hardware descriptors for optimal usage at application layer.Functionality includes: - Zero Copy operations for Packet split/merge,Cloning - Headroom/Tail room addition through merge operations - Allocation of packet buffer and descriptors during startup time

Additional documentation can be found in:

Document	Location
Module Users Guide	$(TI_TRANSPORT_NET_LIB_INSTALL_DIR)\docs\TransportNetLib_UserGuide.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\runtime\pktlib\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\runtime\pktlib\docs\ReleaseNotes_pktlib.pdf

Network Co-processor (NETCP)

NETCP provides hardware accelerator functionality for processing Ethernet packets.

Security Accelerator (SA)

The SA also known as cp_ace (Adaptive Cryptographic Engine) is designed to provide packet security for IPsec, SRTP and 3GPP industry standards. The SA LLD provides APIs to abstract configuration and control between application and the SA. Similar to the PA LLD, it does not provide a transport layer. The Multicore Navigator is used to exchange control packets between the application and the SA firmware.

Note: Due to export control restrictions the SA driver is a separate download from the rest of the MCSDK.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [47]
LLD Users Guide	$(TI_SA_LLD_<ver>_INSTALL_DIR)\sasetup\docs\UserGuide_SA_LLD.pdf
API Reference Manual	$(TI_SA_LLD_<ver>_INSTALL_DIR)\sasetup\packages\ti\drv\sa\docs\doxygen\sa_lld_docs.chm
Release Notes	$(TI_SA_LLD_<ver>_INSTALL_DIR)\sasetup\packages\ti\drv\sa\docs\ReleaseNotes_SA_LLD.pdf

Packet Accelerator (PA)

The PA LLD is used to configure the hardware PA and provides an abstraction layer between an application and the PA firmware. This does not include a transport layer. Commands and data are exchanged between the PA and an application via the Mutlicore Navigator.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [48]
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\pa\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_PA_LLD.pdf

Network Adaptation Layer (NWAL)

Module provides higher level network adaptation layer, and abstracts NETCP access to all upper software layers in TransportNetLib package. Module uses API services from PA/SA/CPPI/QMSS/PKTLIB modules for control packets towards NetCP. Additionally module provides API for packet transmit and receive from NETCP

Additional documentation can be found in:

Document	Location
Module Users Guide covered in TransportNetLib User Guide	$(TI_TRANSPORT_NET_LIB_INSTALL_DIR)\docs\TransportNetLib_UserGuide.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\nwal\docs\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\nwal\docs\\ReleaseNotes_NWAL.pdf
User Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\nwal\docs\UserGuide_NWAL.pdf

I/O and Buses

Antenna Interface (AIF2)

The AIF2 low-level driver is meant to be used by applications which make use of AIF2, QMSS and CPPI IPs. The AIF2 low-level driver aims at generalizing the configuration of AIF2 for different modes (CPRI/OBSAI/Generic packet, WCDMA/LTE/Dual mode). The first goal of the AIF2 LLD is to provide programmers with a “functional layer” or an abstraction of the AIF2 configuration complexity. That means that within a short amount of configuration parameters / API calls, the end user can configure AIF2 for a specific high level scenario.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [49]
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\aif2\docs\AIF2-c66xx_usersguide.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\aif2\docs\ AIF2-c66xx_apireferenceguide.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_AIF2_LLD.pdf

IQN2

IQN2 is intended to communicate with FFTC, RAC, TAC, PA/NetCP sub-systems without the need of constant supervision or control from application software. It is envisioned that application software would initially configure the interaction between IQN2 and these sub-systems, but that in steady state, no or very minimal CPU interventions are required for this data interchange.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [50]
LLD SDS	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\iqn2\docs\IQN2_LLD_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\iqn2\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_IQN2_LLD.pdf

Digital Radio Front End (DFE)

DFE is a high performance wideband digital IF transmit and receive signal processing peripheral for small cell base station applications. It implements advanced algorithms for RF power amplifier linearization including crest factor reduction (CFR) and digital pre-distortion (DPD), and for correcting other receiver RF impairments like IQ imbalance, DC offset and distortion.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [51]
LLD SDS	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\dfe\docs\DFE_LLD_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\dfe\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_DFE_LLD.pdf

Serial RapidIO (SRIO)

The SRIO Low Level Driver provides a well defined standard interface which allows application to send and receive messages via the SRIO peripheral.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [52]
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\srio\docs\SRIO_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\srio\docs\srioDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_SRIODriver.pdf

Peripheral Component Interconnect Express(PCIe)

The PCIe module supports dual operation mode: End Point (EP or Type0) or Root Complex (RC or Type1). This driver focuses on EP mode but it also provides access to some basic RC configuration/functionality. The PCIe subsystem has two address spaces. The first (Address Space 0) is dedicated for local application registers, local configuration accesses and remote configuration accesses. The second (Address Space 1) is dedicated for data transfer. This PCIe driver focuses on the registers for Address Space 0.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [53]
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\pcie\docs\pcieDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_PCIE_LLD.pdf

Hyperlink

The Hyperlink peripheral provides a high-speed, low-latency, and low-power point-to-point link between two Keystone (SoC) devices. The peripheral is also known as vUSR and MCM. Some chip-specific definitions in CSL and documentation may have references to the old names. The LLD provides a well defined standard interface which allows application to configure this peripheral.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [54]
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\hyplnk\docs\hyplnkDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\docs\ReleaseNotes_HYPLNK_LLD.pdf

Co-processors

Bit-rate Coprocessor (BCP)

The BCP driver is divided into 2 layers: Low Level Driver APIs and High Level APIs. The Low Level Driver APIs provide BCP MMR access by exporting register read/write APIs and also provides some useful helper APIs in putting together BCP global and sub-module headers required by the hardware. The BCP Higher Layer provides APIs useful in submitting BCP requests and retrieving their results from the BCP engine.

Turbo Coprocessor Decoder (TCP3d)

The TCP3 decoder driver provides a well defined standard interface which allows application to send code blocks for decoding and receive hard decision and status via EDMA3 transfers.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [55]
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\tcp3d\docs\TCP3D_DriverSDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\tcp3d\docs\TCP3D_DRV_APIIF.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\tcp3d\docs\ReleaseNotes_TCP3DDriver.pdf

FFT Accelerator Coprocessor(FFTC)

The FFTC driver is divided into 2 layers: Low Level Driver APIs and High Level APIs. The Low Level Driver APIs provide FFTC MMR access by exporting register read/write APIs and also provides some useful helper APIs in putting together FFTC control header, DFT size list etc. required by the hardware. The FFTC Higher Layer provides APIs useful in submitting FFT requests and retrieving their results from the FFTC engine without having to know all the details of the Multicore Navigator.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	User Guide [56]
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\fftc\docs\FFTC_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\fftc\docs\fftcDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\fftc\docs\ReleaseNotes_FFTCDriver.pdf

Instrumentation

The instrumentation directory contains components that can be used for the purposes of DSP exception handling, logging, and debug.

Fault Management (FM) Library

The FM library defines a standard interface for a DSP to inform the Linux kernel of a fault and provide crash dump information. On a fault, a DSP configured to use the fault management library can shut down all data transfer IO and then write crash dump information to an ELF Note Section accessible to the Linux kernel. The fault-originating DSP core can use a FM API to send NMI pulses to other DSP cores to initiate exceptions via the DSP core NMI generation registers. Initiating exceptions on the remote DSP cores stops their processing to gain a potentially more complete picture of the system state when the originating-exception occurred. Lastly, the FM module provides an API that the exception handling routine can use to inform the Kernel the DSP has crashed due to an exception and that a coredump has been produced. This signal is sent to the ARM core via the ARM IPC interrupt generation register.

Note: Please note that sending NMI pulses to remote DSP cores to initiate remote exceptions is optional. Please do not initiate a call to this function from the exception hook function if crashing remote DSP cores is not needed.

Additional documentation can be found in:

Document	Location
FM Users Guide	FM Users Guide [57]
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\fault_mgmt\docs\fault_mgmtlibDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\fault_mgmt\docs\ReleaseNotes_fault_mgmt.pdf

Watchdog Timer Module

The Watchdog Timer Module provides APIs for using the local DSP timers as watchdog timers. Each DSP has a local timer, typically DSP n's local timer is system timer n, that can be configured and used as a 64-bit watchdog timer for that DSP. The Watchdog Timer Module provides APIs for an application to configure and "feed" a DSPs watchdog timer as an exception failsafe. If the DSP were to crash or become corrupted the watchdog timer will timeout. The Watchdog Timer Module provides options for if the watchdog times out. The timer can be configured to perform a hard/soft reset, generate an NMI, and generate an exception via the BIOS Exception module.

If the Watchdog Timer module is configured to generate an exception through the BIOS Exception module it can be plugged into the Fault Management library to generate a core dump on exception.

Additional documentation can be found in:

Document	Location
Hardware Peripheral Users Guide	Timer64 User Guide [58]
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\wdtimer\docs\WatchdogTimerDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\wdtimer\docs\ReleaseNotes_WatchdogTimer.pdf

Trace framework (TF) Library

Trace framework is an infrastructure for sharing the information from a “single producer” to multi consumers (maximum 4) over shared memory, designed specifically for the needs of small cell.

Example use cases are:

1. UIA traces produced by a single producer, and consumed by other consumers. (Small cell uses this for instrumentation purposes).
2. Application produced proprietary information by one producer, consumed by multiple consumers in the system (Small cell uses this feature for OAM functionalities)

Additional documentation can be found in:

Document	Location
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\docs\doxygen\html\index.html
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\docs\ReleaseNotes_traceframework.pdf
User Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\docs\traceframework_ug.pdf

CSL MODIFICATIONS COMPARED TO KEYSTONE-I

- Multiple CSL macro define modifications corresponding to the register fields
- Several module specific MMR register fields are expanded which were arrays in overlay structures. For example in the modules COR, DNT, INT, etc.
- Data Structures changed - Bcp_DntHdrCfg, Bcp_SslHdr_WiMaxCfg, Bcp_IntHdrCfg.
- Following table captures register field changes at high level. Please refer the files for actual changes

Module	Removed	Added
SSL – HDR	tdd_output_pack field in WORD1 register	
TM – MMR		New STARVE registers (CDMA_DESC_STARVE_STATUS, CDMA_DESC_STARVE_CLEAR, CDMA_DESC_STARVE_SET, CDMA_DESC_STARVE_INTR_SEL)
TM – MMR		rx_teardown, tx_teardown, clkgate_disable fields in BCP_SOFT_RESET register
TM – MMR	rx_cdmahp_wr_arb_hpriority field in TM_CONTROL register	frc_payload_endian in TM_CONTROL register

LLD API MODIFICATIONS COMPARED TO KEYSTONE-1

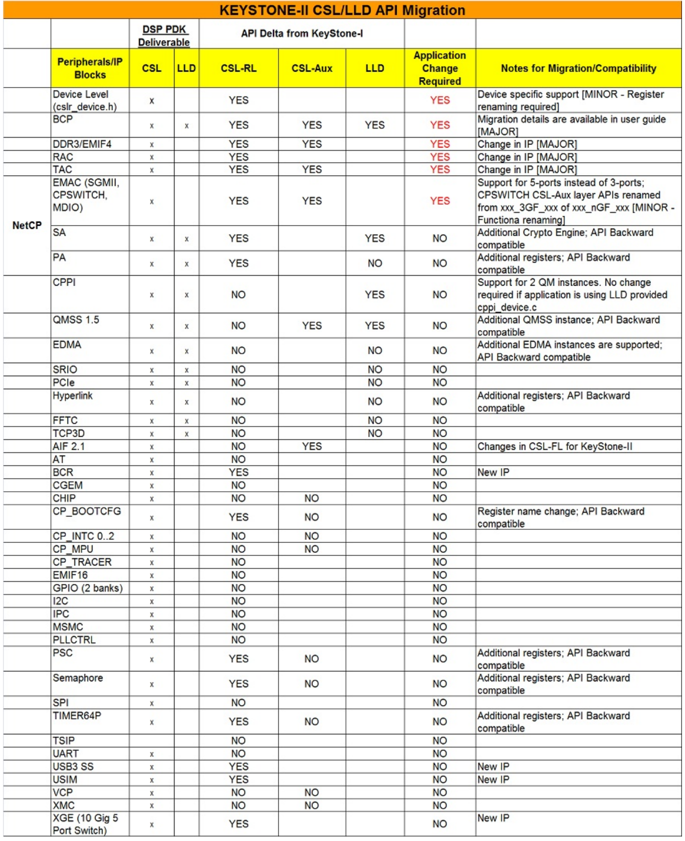
Removed	Bcp_enableRxCdmaHpWriteArbPrio(), Bcp_disableRxCdmaHpWriteArbPrio(), Bcp_isRxCdmaHpWriteArbPrioEnabled()
Modified	Bcp_setTMControlReg(), Bcp_getTMControlReg()
Added	Bcp_doSoftResetRxTeardown(), Bcp_doSoftResetTxTeardown(), Bcp_doSoftResetClockGateDisable(), Bcp_enableForcePayloadEndian(), Bcp_disableForcePayloadEndian(), Bcp_isForcePayloadEndianEnabled(), Bcp_getCdmaDescStarveStatus(), Bcp_setCdmaDescStarveClear(), Bcp_setCdmaDescStarveSet(), Bcp_setCdmaDescStarveInterruptSelect()

Additional documentation can be found in:

Document	Location
LLD Users Guide	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\bcp\docs\BCP_SDS.pdf
API Reference Manual	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\bcp\docs\bcpDocs.chm
Release Notes	$(TI_PDK_INSTALL_DIR)\packages\ti\drv\bcp\docs\ReleaseNotes_BCPDriver.pdf

Keystone-II CSL/LLD API Migration

The following table shows API migration information for CSL/LLD:



KEYSTONE-II CSL/LLD API Migration								
		DSP PDK Deliverable		API Delta from KeyStone-I				
	Peripherals/IP Blocks	CSL	LLD	CSL-RL	CSL-Aux	LLD	Application Change Required	Notes for Migration/Compatibility
	Device Level (cslr_device.h)	x		YES			YES	Device specific support [MINOR - Register renaming required]
	BCP	x	x	YES	YES	YES	YES	Migration details are available in user guide [MAJOR]
	DDR3/EMIF4	x		YES	YES		YES	Change in IP [MAJOR]
	RAC	x		YES			YES	Change in IP [MAJOR]
	TAC	x		YES	YES		YES	Change in IP [MAJOR]
NetCP	EMAC (SGMII, CPSWITCH, MDIO)	x		YES	YES		YES	Support for 5-ports instead of 3-ports; CPSWITCH CSL-Aux layer APIs renamed from xxx_3GF_xxx of xxx_nGF_xxx [MINOR - Functiona renaming]
	SA	x	x	YES		YES	NO	Additional Crypto Engine; API Backward compatible
	PA	x	x	YES		NO	NO	Additional registers; API Backward compatible
	CPPI	x	x	NO		YES	NO	Support for 2 QM instances. No change required if application is using LLD provided cppi_device.c
	QMSS 1.5	x	x	NO	YES	YES	NO	Additional QMSS instance; API Backward compatible
	EDMA	x	x	NO		NO	NO	Additional EDMA instances are supported; API Backward compatible
	SRIO	x	x	NO		NO	NO	
	PCIe	x	x	NO		NO	NO	
	Hyperlink	x	x	NO		NO	NO	Additional registers; API Backward compatible
	FFTC	x	x	NO		NO	NO	
	TCP3D	x	x	NO		NO	NO	
	AIF 2.1	x		NO	YES		NO	Changes in CSL-FL for KeyStone-II
	AT	x		NO			NO	
	BCR	x		YES			NO	New IP
	CGEM	x		NO			NO	
	CHIP	x		NO	NO		NO	
	CP_BOOTCFG	x		YES	NO		NO	Register name change; API Backward compatible
	CP_INTC 0..2	x		NO	NO		NO	
	CP_MPU	x		NO	NO		NO	
	CP_TRACER	x		NO			NO	
	EMIF16	x		NO			NO	
	GPIO (2 banks)	x		NO			NO	
	I2C	x		NO			NO	
	IPC	x		NO			NO	
	MSMC	x		NO			NO	
	PLLCTRL	x		NO			NO	
	PSC	x		YES	NO		NO	Additional registers; API Backward compatible
	Semaphore	x		YES	NO		NO	Additional registers; API Backward compatible
	SPI	x		NO			NO	
	TIMER64P	x		YES	NO		NO	Additional registers; API Backward compatible
	TSIP			NO			NO	
	UART	x		NO			NO	
	USB3 SS	x		YES			NO	New IP
	USIM	x		YES			NO	New IP
	VCP	x		NO	NO		NO	
	XMC	x		NO	NO		NO	
	XGE (10 Gig 5 Port Switch)	x		YES			NO	New IP

Platform Library

The platform library defines a standard interface for platform utilities and functionality and provides sample implementations for the EVM platform. These include things such as reading and writing to EEPROM, FLASH, UART, etc. Platform library supports three libraries

1. debug library (ti.platform.evm###.ae66) - located under \platform_lib\lib\debug, needed only when a debug is needed on the platform library since the source is compiled with full source debugging.
2. release library (ti.platform.evm###.ae66) - located under \platform_lib\lib\release, should be used normally for the best performance of the cycles since the code is compiled with the full optimization.

Platform Library Summary	
Component Type	Library
Install Package	PDK for Keystone II
Install Directory	pdk_keystone2_<version>\packages\ti\platform\evm###\platform_lib
Project Type	CCS [18]
Endian Support	Little
Library Name	Select for the TCI66##### EVM ti.platform.evm###.ae66 (little)
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\platform\evm###\platform_lib\lib\debug - for debug version $(TI_PDK_INSTALL_DIR)\packages\ti\platform\evm###\platform_lib\lib\release - for release version
Linker Sections	platform_lib
Section Preference	none

Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\platform platform.h defines the interface
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Texas Instruments Embedded Processors Wiki [59]
Downloads	Product Updates
License	BSD [45]

Transport

The Transports are intermediate drivers that sit between either the NDK and interface them to the appropriate EVM peripherals. The Transport supported by MCSDK are:

- NDK transport - Network Interface Management Unit (NIMU) Driver

More information on these can be found in the NDK or IPC sections of this guide.

EDMA3 Low Level Driver

EDMA3 Low Level Driver is targeted to users (device drivers and applications) for submitting and synchronizing EDMA3-based DMA transfers.

EDMA3 is a peripheral that supports data transfers between two memory mapped devices. It supports EDMA as well as QDMA channels for data transfer. This peripheral IP is re-used in different SoCs with only a few configuration changes like number of DMA and QDMA channels supported, number of PARAM sets available, number of event queues and transfer controllers etc. The EDMA3 peripheral is used by other peripherals for their DMA needs thus the EDMA3 Driver needs to cater to the requirements of device drivers of these peripherals as well as other application software that may need to use DMA services.

The EDMA3 LLD consists of an EDMA3 Driver and EDMA3 Resource Manager. The **EDMA3 Driver** provides functionality that allows device drivers and applications for submitting and synchronizing with EDMA3 based DMA transfers. In order to simplify the usage, this component internally uses the services of the **EDMA3 Resource Manager** and provides one consistent interface for applications or device drivers.

EDMA3 Driver Summary	
Component Type	Library
Install Package	EDMA3 Low level drivers
Install Directory	<root_install_dir>/edma3_lld_xx_xx_xx_xx
Project Type	N/A
Endian Support	Little and Big
Library Name	edma3_lld_drv.ae66 (little endian) and edma3_lld_drv.ae66e (big endian)
Linker Path	N/A
Linker Sections	N/A
Section Preference	N/A
Include Paths	N/A
Reference Guides	See docs under install directory
Support	Technical Support

Additional Resources	Programming the EDMA3 using the Low-Level Driver (LLD) [60]
Downloads	Product Updates
License	BSD [45]

SYS/BIOS

SYS/BIOS is a scalable real-time kernel. It is designed to be used by applications that require real-time scheduling and synchronization or realtime instrumentation. SYS/BIOS provides preemptive multi-threading, hardware abstraction, real-time analysis, and configuration tools. SYS/BIOS is designed to minimize memory and CPU requirements on the target.

SYS/BIOS Summary	
Component Type	Libraries
Install Package	SYS/BIOS
Install Directory	bios_6_<version>\
Project Type	Eclipse RTSC [43]
Endian Support	Little and Big
Library Name	The appropriate libraries are selected for your device and platform as set in the RTSC build properties for your project and based on the use module statements in your configuration.
Linker Path	The appropriate path is selected to the libraries for your device and platform as set in the RTSC build properties for your project.
Linker Sections	N/A
Section Preference	N/A
Include Paths	BIOS_CG_ROOT is set automatically by CCS based on the version of BIOS you have checked to build with. ${BIOS_CG_ROOT}\packages\ti\bios\include
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	SYS/BIOS Online Training [61] SYS/BIOS 1.5-DAY Workshop [62] Eclipse RTSC Home [43]
Downloads	SYS/BIOS Downloads [63]
License	BSD [45]

Inter-Processor Communication (IPC)

Inter-Processor Communication (IPC) provides communication between processors in a multi-processor environment.

IPC provides messaging APIs for Linux-to-BIOS and BIOS-to-BIOS communication.

IPC Summary	
Component Type	Libraries
Install Package	IPC
Install Directory	ipc_<version>
Project Type	Eclipse RTSC [43]
Endian Support	Little and Big
Library Name	The appropriate libraries are selected based on your application's configuration.
Linker Path	The appropriate paths are added in the config-generated linker command script, based on your application's configuration.
Linker Sections	N/A
Section Preference	N/A
Include Paths	IPC_INSTALL_DIR/packages
Reference Guides	IPC Users Guide API Reference [64] Config Reference [65]
Support	Technical Support
Additional Resources	Eclipse RTSC Home [43]
Downloads	IPC Downloads [66]
License	BSD [45]

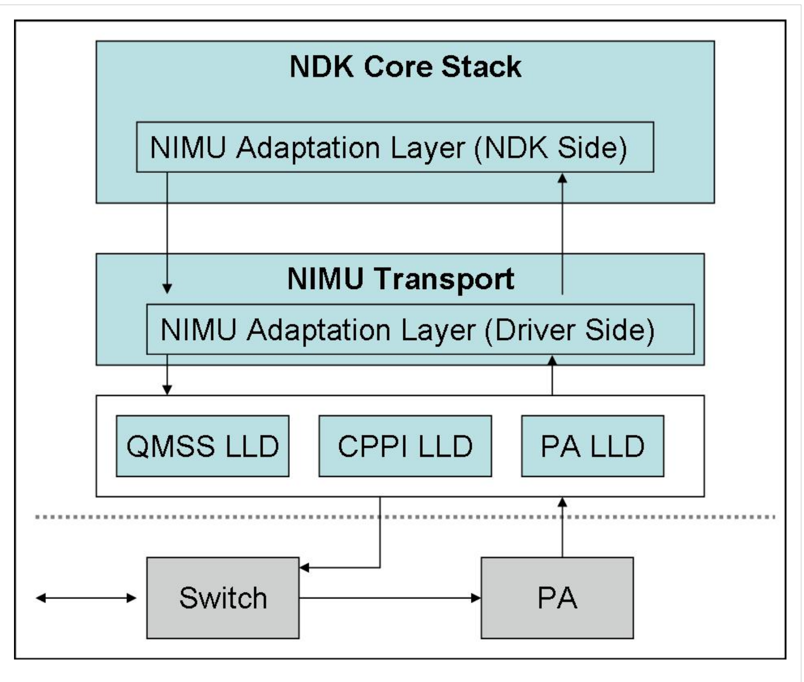
Network Development Kit (NDK)

The NDK is a platform for development and demonstration of network enabled applications on DSP devices and includes demonstration software showcasing DSP capabilities across a range of network enabled applications. The NDK serves as a rapid prototype platform for the development of network and packet processing applications, or to add network connectivity to existing DSP applications for communications, configuration, and control. Using the components provided in the NDK, developers can quickly move from development concepts to working implementations attached to the network.

The NDK provides an IPv6 and IPv4 compliant TCP/IP stack working with the SYS/BIOS real-time operating system. Its primary focus is on providing the core Layer 3 and Layer 4 stack services along with additional higher-level network applications such as HTTP server and DHCP.

The NDK itself does not include any platform or device specific software. The NDK interfaces through well-defined interfaces to the PDK and platform software elements needed for operation.

The functional architecure for NDK is shown below.


NDK Core Stack
NIMU Adaptation Layer (NDK Side)
NIMU Transport
NIMU Adaptation Layer (Driver Side)
QMSS LLD
CPPI LLD
PA LLD
Switch
PA

Network Development Kit Summary	
Component Type	Libraries
Install Package	NDK
Install Directory	ndk_<version>\
Project Type	Eclipse RTSC [43]
Endian Support	Little and Big
Library Name	binsrc.lib or binsrce.lib and cgi.lib or cgie.lib and console.lib or consolee.lib and hdlc.lib or hdlce.lib and miniPrintf.lib or miniPrintfe.lib and netctrl.lib or netctrle.lib and nettool.lib or nettoole.lib and os.lib or ose.lib and servers.lib or serverse.lib and stack.lib or stacke.lib
Linker Path	$(NDK_INSTALL_DIR)\packages\ti\ndk\lib\<arch>
Linker Sections	.far:NDK_OBJMEM, .far:NDK_PACKETMEM
Section Preference	L2 Cache

Include Paths	NDK_INSTALL_DIR is set automatically by CCS based on the version of NDK you have checked to build with. ${NDK_INSTALL_DIR}\packages\ti\ndk\inc ${NDK_INSTALL_DIR}\packages\ti\ndk\inc\tools
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	The NDK unit test examples are available in $(TI_MCSDK_INSTALL_DIR)\packages\ti\platform\nimu\test\evm####
Extended Support	Eclipse RTSC Home [43] NDK User's Guide [67] NDK Programmer's Reference Guide [68] NDK Support Package Ethernet Driver Design Guide [69] NDK_FAQ [70] Rebuilding NDK Core [71]
Downloads	NDK Downloads [72]
License	BSD [45]

Network Interface Management Unit (NIMU) Driver

NIMU sits between NDK common software and the TCI66##### platform library and provides a common interface for NDK to communicate with. This package have NDK unit test examples for all supported platforms.

Note: This module is only intended to be used with NDK. As such, users should not tie up to its API directly.

The functional architecture for NIMU (taking TCI66##### platform as an example) is shown below.

NIMU Summary	
Component Type	Library
Install Package	PDK_INSTALL_DIR
Install Directory	mcsdk_<version>\packages\ti\transport\ndk\nimu
Project Type	Eclipse RTSC [43]
Endian Support	Little
Library Name	ti.transport.ndk.nimu.ae66 (little)
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\transport\ndk\nimu\lib\debug for debug version $(TI_PDK_INSTALL_DIR)\packages\ti\transport\ndk\nimu\lib\release for release version
Linker Sections	nimu_eth_ll2
Section Preference	L2SRAM
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\transport\ndk\nimu\include
Reference Guides	None
Support	Technical Support
Additional Resources	The NDK unit test examples are available in $(TI_MCSDK_INSTALL_DIR)\examples\ndk\evm###
Downloads	[73]
License	BSD [45]

Please note that all contributions to MediaWiki may be edited, altered, or removed by other contributors. If you do not want your writing to be edited mercilessly, then do not submit it here. You are also promising us that you wrote this yourself, or copied it from a public domain or similar free resource (see MediaWiki:Copyrights for details). Do not submit copyrighted work without permission!

Algorithm Libraries

TI provides several algorithm libraries, each specific to a particular arena. Each library provides a collection of C-callable low-level functions (kernels), each tailored for optimal performance on a specific TI processing device (or devices). The libraries are typically used in computationally intensive real-time applications where execution speed is a critical factor. Their use generally accelerates execution speeds well beyond that achieved by equivalent code written in standard ANSI C. Additionally, use of these libraries can significantly reduce application development time. Source code is provided in all cases to facilitate kernel modification when needed.

See c6x Software Library mediawiki [74] for a comprehensive overview of the various software libraries available for TI's c6x family of processors.

DSP Library (DSPLIB)

DSPLIB is an optimized DSP Function Library and includes many C-callable, optimized, general-purpose signal-processing routines including:

- Adaptive Filtering
- Correlation
- Fast Fourier Transform
- Filtering and convolution
- Matrix

DSPLIB Summary	
Component Type	Library
Install Package	DSPLIB
Install Directory	dsplib_c66x_<version>\
Project Type	CCS [18]
Endian Support	Big and Little
Library Name	dsplib.a66 (COFF, little-endian) dsplib.a66e (COFF, big-endian) dsplib.ae66 (ELF, little-endian) dsplib.ae66e (ELF, big-endian)
Linker Path	<root_install_dir>\lib\
Linker Sections	N/A
Section Preference	N/A
Include Paths	<root_install_dir>\inc\ <root_install_dir>\packages\
Reference Guides	See docs under Install Directory
Support	BIOS E2e Forum [75]
Additional Resources	c6x Software Library mediawiki [74]
Downloads	DSPLIB Downloads [76]
License	BSD [45]

Image Processing Library (IMGLIB)

IMGLIB is an optimized image/video processing library with kernels in the following functional categories:

- Compression & Decompression
- Image Analysis
- Image Filtering and Conversion

IMGLIB Summary	
Component Type	Library
Install Package	IMGLIB
Install Directory	imglib_c66x_<version>\
Project Type	CCS [18]
Endian Support	Little
Library Name	imglib.ae66 (ELF, little-endian)
Linker Path	<root_install_dir>\lib\
Linker Sections	N/A
Section Preference	N/A
Include Paths	<root_install_dir>\inc\ <root_install_dir>\packages\
Reference Guides	See docs under Install Directory
Support	BIOS E2e Forum [75]
Additional Resources	c6x Software Library mediawiki [74]
Downloads	IMGLIB Downloads [77]
License	BSD [45]

Floating Point Math Library (MATHLIB)

MATHLIB contains optimized versions of most commonly used floating point math routines contained in the RTS library. Kernels are offered in two variations:

- Double-precision floating point
- Single-precision floating point

MATHLIB Summary	
Component Type	Library
Install Package	MATHLIB
Install Directory	mathlib_c66x_<version>\
Project Type	CCS [18]
Endian Support	Big and Little
Library Name	mathlib.a66 (COFF, little-endian) mathlib.a66e (COFF, big-endian) mathlib.ae66 (ELF, little-endian) mathlib.ae66e (ELF, big-endian)
Linker Path	<root_install_dir>\lib\

Linker Sections	N/A
Section Preference	N/A
Include Paths	<root_install_dir>\inc\ <root_install_dir>\packages\
Reference Guides	See docs under Install Directory
Support	BIOS E2e Forum [75]
Additional Resources	c6x Software Library mediawiki [74]
Downloads	MATHLIB Downloads [78]
License	BSD [45]

Resource Manager

The Resource Manager is detailed here: RM User Guide [79]

DSP Management

Boot Utilities

Overview

MCSDK includes a Tools Package which provides boot utilities for use with the TI EVMs and are intended to serve as example/reference for customers.

The MCSDK tools package is located in the C:\ti\mcsdk_bios_#_##_##_##\tools directory and includes:

- **Writer Utilities**: Utilities to program an application image to flash or EEPROM.
- **Program EVM**: Utilities to program default application images to flash.
- **Other Utilities**: Utilities to do file format conversion that are required by the boot examples.

Flash and Flash Utilities

The following boot utilities for loading code into the EEPROM, NOR and NAND are provided as part of the Tools Package with the MCSDK. All source code is provided along with documentation so that customers can port to other environments as necessary or to make modifications and enhancements.

- **EEPROM Writer**: Utility for writing to the EEPROM. This utility executes on the EVM using CCS and JTAG and it is located under C:\ti\mcsdk_bios_#_##_##_##\tools\writer\eeprom\evm###\bin directory.
- **NOR Writer**: Utility for writing to the NOR flash. This utility executes on the EVM using CCS and JTAG and it is located under C:\ti\mcsdk_bios_#_##_##_##\tools\writer\nor\evm###\bin directory.
- **NAND Writer**: Utility for writing to the NAND flash. This utility executes on the EVM using CCS and JTAG and it is located under C:\ti\mcsdk_bios_#_##_##_##\tools\writer\nand\evm###\bin directory.

Useful Tip

The program_evm utility provides the ability to format the NAND (i.e., permanently erase the entire NAND device). Please refer to program_evm_userguide.pdf (located in the mcsdk_bios_#_##_##_##\tools\program_evm\ directory) for more information.

c6x DSP Linux Community

(point to community page) ß Not required since not applicable to KeyStone II as of yet, but just a test of scalability of the document.

Tools

Multicore System Analyzer (MCSA)

Multicore System Analyzer (MCSA) is a suite of tools that provide real-time visibility into the performance and behavior of your code, and allow you to analyze information that is collected from software and hardware instrumentation in a number of different ways.

Advanced Tooling Features:

- Real-time event monitoring
- Multicore event correlation
- Correlation of software events, hardware events and CPU trace
- Real-time profiling and benchmarking
- Real-time debugging

Two key components of System Analyzer are:

- DVT: Various features of Data Analysis and Visualization Technology (DVT) provide the user interface for System Analyzer within Code Composer Studio (CCS)
- UIA: The Unified Instrumentation Architecture (UIA) target package defines APIs and transports that allow embedded software to log instrumentation data for use within CCS

MCSA Summary	
Component Type	Libraries
Install Package	UIA + DVT
Install Directory	ccsv5/uia_<version>, ccsv5/eclipse, ccsv5/ccs_base_5.0.0.*/dvt\
Project Type	Eclipse RTSC [43]
Endian Support	Little
Library Name	The appropriate libraries are selected for your device and platform as set in the RTSC build properties for your project and based on the use module statements in your configuration.
Linker Path	The appropriate path is selected to the libraries for your device and platform as set in the RTSC build properties for your project.
Linker Sections	N/A
Section Preference	N/A
Include Paths	N/A
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Multicore System Analyzer [80]
Downloads	Installed as a part of MCSDK installation
UIA License	BSD [45]

DVT License	TI Technology and Software Publicly Available (TSPA). See DVT Manifest in the install directory.

Eclipse RTSC Tools (XDC)

RTSC is a C-based programming model for developing, delivering, and deploying Real-Time Software Components targeted for embedded platforms. The XDCtools product includes tooling and runtime elements for component-based programming using RTSC.

XDC Summary	
Component Type	Tools
Install Package	XDC
Install Directory	xdctools_<version>\
Project Type	Eclipse RTSC [43]
Endian Support	Little and Big
Library Name	The appropriate libraries are selected for your device and platform as set in the RTSC build properties for your project and based on the use module statements in your configuration.
Linker Path	The appropriate path is selected to the libraries for your device and platform as set in the RTSC build properties for your project.
Linker Sections	systemHeap
Section Preference	none
Include Paths	N/A
Reference Guides	See docs under Install Directory
Support	Technical Support
Additional Resources	Eclipse RTSC Home [43] Users Guide and Reference Manual [81]
Downloads	N/A
License	See XDC Manifest in the install directory

Demonstrations

The MCSDK consist of demonstration software to illustrate device and software capabilities, benchmarks, and usage.

See the Getting Started Guide's demonstration section [82] for instructions to run the pre-built demonstrations in the filesystem.

Utility Application

http://processors.wiki.ti.com/index.php/MCSDK_Utility_App_Demonstration_Guide

Image Processing

http://processors.wiki.ti.com/index.php/MCSDK_Image_Processing_Demonstration_Guide

Inter-Processor Communication

http://processors.wiki.ti.com/index.php/MCSDK_Inter_Processor_Communication_Demonstration_Guide

How To

How to run Linux kernel from a different physical address than default (0x80000000) ?

A sample patch to change DDR base address to 0xA0000000 instead of the default address 0x80000000 is provided below. User is responsible for maintaining this patch at their code base and the same will not be included in the MCSDK.

Linux code patch

```
--- ref/keystone-0430a//linux-keystone/arch/arm/mach-keystone/Makefile.boot      2013-04-30 11:19:07.000000000 -0700
+++ linux-keystone/arch/arm/mach-keystone/Makefile.boot      2013-05-09 13:47:29.829257106 -0700
@@ -1 +1 @@
-zreladdr-y      := 0x80008000
+zreladdr-y      := 0xA0008000
--- ref/keystone-0430a//linux-keystone/arch/arm/mach-keystone/include/mach/memory.h      2013-04-30 11:19:07.000000000 -0700
+++ linux-keystone/arch/arm/mach-keystone/include/mach/memory.h      2013-05-09 12:05:10.561756528 -0700
@@ -16,6 +16,7 @@
 #ifndef __ASM_MACH_MEMORY_H
 #define __ASM_MACH_MEMORY_H

+
 #define MAX_PHYSMEM_BITS      36
 #define SECTION_SIZE_BITS      34

@@ -32,10 +33,17 @@

 #ifndef __ASSEMBLY__

+static inline phys_addr_t __virt_to_phys(unsigned long x);
+
 static inline phys_addr_t __virt_to_idmap(unsigned long x)
 {
+#ifdef      ORIGINAL_PRE_TBESEMER
       return (phys_addr_t)(x) - CONFIG_PAGE_OFFSET +
             KEYSTONE_LOW_PHYS_START;
+#else
+      return __virt_to_phys(x) - KEYSTONE_HIGH_PHYS_START +
+            KEYSTONE_LOW_PHYS_START;
+#endif
 }

 #define virt_to_idmap(x)      __virt_to_idmap((unsigned long)(x))
```

Device tree source patch

```
--- ref/keystone-0430a//linux-keystone/arch/arm/boot/dts/k2hk-evm.dts      2013-04-30 11:19:07.000000000 -0700
+++ linux-keystone/arch/arm/boot/dts/k2hk-evm.dts      2013-05-10 09:56:40.418347675 -0700
@@ -17,7 +17,7 @@
       };

       memory {
-            reg = <0x00000000 0x80000000 0x00000000 0x20000000>;
+            reg = <0x00000000 0xA0000000 0x00000000 0x20000000>;
       };

       droppolicies: default-drop-policies {
```

u-boot code patch

```
--- ref/keystone-0430a//u-boot-keystone/include/configs/tci6638_evm.h      2013-04-30 10:29:03.000000000 -0700
+++ u-boot-keystone/include/configs/tci6638_evm.h      2013-05-09 13:46:25.277274692 -0700
@@ -47,8 +47,8 @@

 /* Memory Configuration */
 #define CONFIG_NR_DRAM_BANKS            1
-#define CONFIG_SYS_SDRAM_BASE            0x80000000
-#define CONFIG_MAX_RAM_BANK_SIZE      (2 << 30)      /* 2GB */
+#define CONFIG_SYS_SDRAM_BASE            0xA0000000
+#define CONFIG_MAX_RAM_BANK_SIZE      ((1024 * 1024) * 1024)
 #define CONFIG_STACKSIZE            (512 << 10)      /* 512 KiB */
 #define CONFIG_SYS_MALLOC_LEN            (512 << 10)      /* 512 KiB */
 #define CONFIG_SYS_MEMTEST_START      CONFIG_SYS_SDRAM_BASE
@@ -194,7 +194,7 @@
 #define CONFIG_CMD_SF

 /* U-Boot general configuration */
-#define CONFIG_SYS_PROMPT            "TCI6638 EVM # "
+#define CONFIG_SYS_PROMPT            "TCI6638 EVM (teb) # "
 #define CONFIG_SYS_CBSIZE            1024
 #define CONFIG_SYS_PBSIZE            2048
 #define CONFIG_SYS_MAXARGS            16
@@ -265,6 +265,6 @@
 #define CONFIG_SETUP_MEMORY_TAGS
 #define CONFIG_SYS_BARGSIZE            1024
 #define CONFIG_SYS_LOAD_ADDR            (CONFIG_SYS_SDRAM_BASE + 0x08000000)
-#define LINUX_BOOT_PARAM_ADDR            (CONFIG_SYS_SDRAM_BASE + 0x100)
+#define LINUX_BOOT_PARAM_ADDR            (CONFIG_SYS_SDRAM_BASE + 0x00000100)

 #endif /* __CONFIG_H */
```

Also when using the patch, make sure to update the following env variables in u-boot

```
setenv addr_fdt 0xA7000000
setenv addr_kern 0xA8000000
```

How to boot with a combined kernel and initramfs image?

This section is borrowed from [[83]]

To use initramfs as part of a combined kernel/initramfs image, a cpio archive is embedded directly into the kernel. I.e. you don't create an additional ramfs image. Instead, the initial file system is directly incorporated into the kernel. With this, the kernel size increases by the file system size. It's like you embed above ramfs directly into the kernel. You simply have to fill a directory on your host with the target filesystem you like and then pass the path to this directory to the kernel build process.

Create target file system

```
host > mkdir target_fs

host > ... copy stuff you want to have in initramfs to target_fs... do following command using arago-console-image.tar.gz provided in the release

host > cd target_fs

host > sudo tar -xvzf <path of arago-console-image.tar.gz>

host > cd ..

host > sudo chmod a+rwx target_fs -R
```

Now cd to linux kernel root directory.

Configure kernel to embed initramfs

Kernel options

Now give the kernel the path to the target file system you like to embed. Edit the defconfig for the platform and update the CONFIG_INITRAMFS_SOURCE variable and save:-

```
#
# General setup
#
...
CONFIG_BLK_DEV_INITRD=y
CONFIG_INITRAMFS_SOURCE="<path_to>/target_fs>"
...
```

```
#
# UBI - Unsorted block images
#
...
CONFIG_BLK_DEV_RAM=y
CONFIG_BLK_DEV_RAM_COUNT=1
CONFIG_BLK_DEV_RAM_SIZE=8192
...
```

Then, if you compile the kernel, e.g. by make uImage, the cpio archive is generated and embedded into the kernel:

```
...
CHK     include/linux/compile.h
GEN     usr/initramfs_data.cpio.gz
```

```
AS      usr/initramfs_data.o
LD      usr/built-in.o
...
```

Setup u-boot environment and boot combined image

At the u-boot console, do the following:-

For boot mode = ramfs do the following:-

Copy the kernel image (build using procedure above), uImage-combined, to the tftp server root directory.

```
env default -f -a
setenv serverip <tftp server IP>
setenv args_ramfs 'setenv bootargs ${bootargs} earlyprintk rdinit=/sbin/init'
setenv name_kern uImage-combined
setenv tftp_root <tftp server root directory>
setenv init_ramfs 'run args_all args_ramfs'
saveenv
```

At this time, issue the boot command and Linux boots up with embedded initramfs image.

How to change Tetris and Core PLL speed?

The prerequisite is to follow hardware setup guide to update EVM with the latest BMC and UCD firmware version.

1. The PLL clock frequency is now programmed based on EFUSE register settings for K2HK Rev. 3.x EVM and K2EL Rev. 1.0.2.x. The getclk command in u-boot can be used to know the actual frequency used. The default PLL speeds of the Core(DSP) and Tetris(ARM) are

Platform	Core	Tetris
K2H	800MHz	1200MHz
K2K	800MHz	1200MHz
K2E	1250MHz	1250MHz
K2L	1250MHz	1250MHz

2. To change the Tetris frequency, the EVM's reference clock is needed for the calculation. Each platform may have different main reference clock frequency, please refer to the sys_clk defined in board_k2x.c for the refclkmain frequency.

If Tetris PLL speed needs to be changed on K2HK, at Uboot prompt, pllset command can be used. E.g.:

```
  pllset arm 19 1 2
```

Here the calculation example assuming the refclkmain is 125MHz please use the refclkmain value mentioned above when changing on EVM):

```
  125000000*19/(1*2) = 1187500000 Hz
```

At Uboot prompt, getclk command can be used to verify the Tetris frequency setting. The argument for Tetris component is based on the enum clk_e definition in clock_k2x.h. For K2H, the Tetris speed can be read by the following command

```
  >getclk 2
```

For K2E, the Tetris speed is the same as core PLL and can not be changed. The core_pll_clk defined in enum clk_e of clock_k2e.h is 0. So, for K2E, the Tetris speed can be read by the following command

```
>getclk 0
```

3. To change the Core PLL clock, you need to update the core_pll_config[] in board/ti/ks2_evm/board_k2x.c by replacing CORE_PLL_1228 with appropriate clock define.

For example, if current core speed of K2H is at 800MHz, modify the core_pll_config setting in board_k2hk.c

```
from  { CORE_PLL,     13,     1,      2 }, /* 799 */
to    { CORE_PLL,     625,     32,      2 }, /* 1200 */
```

How to boot Linux using uinitrd?

To boot Linux using a uImage formatted initrd, follow the procedure given here.

1. First generate the uinitrd.bin image using the following command on a Linux Host:

```
mkimage  -A arm -O linux -T ramdisk -C gzip -a 0 -e 0 -n initramfs -d arago-console-image.cpio.gz uinitrd.bin
```

Where arago-console-image.cpio.gz is available as part of the release. Copy uinitrd.bin to the tftp server root directory

2. Make sure the u-boot image is built using latest u-boot-keystone source at git.ti.com. Following commits are required at the minimum:-

```
keystone: dt fix up for uinitrd
keystone2: automate boot up using uinitrd fs
```

3. Upgrade the u-boot image to the EVM.

4. Do the following command after u-boot boots up

```
env default -f -a
setenv serverip <tftp server IP>
setenv tftp_root <Tftp server root directory>
setenv boot uinitrd
saveenv
```

4a. If using LPAE on Linux version below 3.13, and have the patch "keystone2: add env option to do unitrd dt fixup", set the following flag:

```
setenv uinitrd_fixup 1
saveenv
```

5. Issue boot command to boot to Linux. A sample initial part of u-boot part of the bootlog is shown below.

```
## Booting kernel from Legacy Image at 88000000 ...
   Image Name:   Linux-3.8.4-rt2-01184-gf78749b
   Created:      2013-07-05  19:56:56 UTC
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3648776 Bytes = 3.5 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 82000000 ...
```

```
   Image Name:   initramfs
   Created:      2013-07-01  18:38:53 UTC
   Image Type:   ARM Linux RAMDisk Image (gzip compressed)
   Data Size:    7652966 Bytes = 7.3 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
   Booting using the fdt blob at 0x87000000
   Loading Kernel Image ... OK
OK
   Using Device Tree in place at 87000000, end 8700dae3
```

How to use fixed IP address instead of DHCP?

1. U-boot IP address

```
  To statically configure an IP address for u-boot, use the setenv command:
   setenv ipaddr <ip_address>
```

2. If static IP address is configured in u-boot, then it is not applicable to use dhcp to download files. It is necessary to change all downloading commands in printenv from 'dhcp <file_name>' to 'tftp <file_name>'. and the command examples are as following:

```
  setenv get_fdt_net 'tftp ${addr_fdt} ${tftp_root}/${name_fdt}'
  setenv get_fdt_ramfs 'tftp ${addr_fdt} ${tftp_root}/${name_fdt}'
  setenv get_fs_ramfs 'tftp ${addr_fs} ${tftp_root}/${name_fs}'
  setenv get_kern_net 'tftp ${addr_kern} ${tftp_root}/${name_kern}'
  setenv get_kern_ramfs 'tftp ${addr_kern} ${tftp_root}/${name_kern}'
  setenv get_mon_net 'tftp ${addr_mon} ${tftp_root}/${name_mon}'
  setenv get_mon_ramfs 'tftp ${addr_mon} ${tftp_root}/${name_mon}'
  setenv get_uboot_net 'tftp ${addr_uboot} ${tftp_root}/${name_uboot}'
  setenv get_uboot_ramfs 'tftp ${addr_uboot} ${tftp_root}/${name_uboot}'
```

To revert the commands back to default settings if not using static IP address:

```
  setenv get_fdt_net 'dhcp ${addr_fdt} ${tftp_root}/${name_fdt}'
  setenv get_fdt_ramfs 'dhcp ${addr_fdt} ${tftp_root}/${name_fdt}'
  setenv get_fs_ramfs 'dhcp ${addr_fs} ${tftp_root}/${name_fs}'
  setenv get_kern_net 'dhcp ${addr_kern} ${tftp_root}/${name_kern}'
  setenv get_kern_ramfs 'dhcp ${addr_kern} ${tftp_root}/${name_kern}'
  setenv get_mon_net 'dhcp ${addr_mon} ${tftp_root}/${name_mon}'
  setenv get_mon_ramfs 'dhcp ${addr_mon} ${tftp_root}/${name_mon}'
  setenv get_uboot_net 'dhcp ${addr_uboot} ${tftp_root}/${name_uboot}'
  setenv get_uboot_ramfs 'dhcp ${addr_uboot} ${tftp_root}/${name_uboot}'
```

3. Linux Kernel IP address

```
  To statically configure IP address for Linux Kernel, The args_all
need to be updated/modified for all boot modes:ubi, net, ramfs etc.
with the following commands using boot=net as the example and assuming
```

```
the static IP address is the same as u-boot environment variable,
ipaddr:
```

```
  1) Create uboot environment variable for netmask.

     # setenv netmask 255.255.255.0

  2) Check environment variable for ipaddr is set.  Set if necessary.

     # printenv ipaddr

  3) Check environment variable for gatewayip is set. Set if necessary.

     # printenv gatewayip

  4) Modify args_all

     # setenv args_all 'setenv bootargs console=ttyS0,115200n8 rootwait=1 ip=${ipaddr}::${gatewayip}:${netmask}::eth0:off'

  5) Remove “ip=dhcp” from args_net

     a)printenv args_net

     b)copy everything except ip=dhcp

     c)setenv args_net ‘<copy args_net from step above without ip=dhcp>’ (quotes are important).

       The command should look like:

       #setenv args_net 'setenv bootargs ${bootargs} rootfstype=nfs root=/dev/nfs rw nfsroot=${serverip}:${nfs_root},${nfs_options}'

  6) Save the uboot environment

     #saveenv

  7) Boot the EVM

     #boot
```

If using multiple interfaces, DHCP and NFS filesystem

To make sure the interface your NFS server is connected to is the first interface to boot in the bootargs explained in the section above, you may assign "ip=:::::eth0:dhcp". This will ensure eth0 is used for NFS mount.

How to boot from USB Flash drive?

On Ubuntu Host

I - Install gparted

```
$sudo apt-get install gparted
```

Used GParted 0.11.0 for this test. So exact steps may vary if version is different

II - Partition the device

Use partition 1 for boot images and Partition 2 for rootfs

insert the usb flash drive to usb slot

```
$sudo gparted
```

The GUI will show up now. Choose correct device (/dev/sdx) at the top right corner (shows the correct size of the usb drive). This example uses /dev/sdc1 for partion 1 and /dev/sdc2 for partition 2

Warning!! Before you proceed, make sure you selected the correct device, otherwise it may format the active harddisk partition of your PC

Got to top level "Partition" pulldown menu and select Unmount to unmount the device. If there is existing partition, delete the same (Partition -> delete)

Now Partition and filesystem status shows as unallocated

1. Create fat32 partion for boot (contains boot images)

Partition -> new

New size = 32MiB

File system = fat32

label = boot

Keep rest of the fields default

click add to add partition

select the partiton #1 just created and format it (Partition -> format to).

select fat32 format

2. Create ext4 format for rootfs (contains root filesystem)

select the unallocate partition

Partition -> new

File system = ext4

label = rootfs

Keep rest of the fields default

click add to add partition

select the partiton #2 just created and format it (Partition -> format to).

select ext4 format

3. Apply the changes and Quit

Edit -> Apply All Operations

GParted -> Quit

III - Copy images and rootfs files to partitions

Unmount if the devices are automounted

```
$sudo umount  /dev/sdc1
$sudo umount  /dev/sdc2
```

1. Copy images to partition #1 (boot)

```
$sudo mount -t vfat /dev/sdc1 /mnt/test
```

Assume all images are available at the current directory

```
$sudo cp skern-keystone-evm.bin /mnt/test/
$sudo cp uImage-k2hk-evm.dtb /mnt/test/
$sudo cp uImage-keystone-evm.bin /mnt/test/
$ls /mnt/test
skern-keystone-evm.bin  uImage-k2hk-evm.dtb  uImage-keystone-evm.bin
$sudo umount /dev/sdc1
```

2. Copy rootfs files to partition #2 (rootfs)

```
$mdir test
$cd test
$gunzip <path of arago-console-image.cpio.gz>
$sudo cpio -i -F ../arago-console-image.cpio
```

```
$sudo mount -t ext4 /dev/sdc2 /mnt/test
$cd ..
$sudo cp -r test/* /mnt/test
$ls /mnt/test/
bin  boot  dev  etc  home  init  lib  lost+found  media  mnt  proc  sbin  srv  sys  tmp  usr  var
$sudo umount /dev/sdc2
```

On EVM

Requires Linux Kernel image with EXT4 support

Setup u-boot env variables

Insert USB flash drive to usb slot on EVM and Power ON EVM Type the following commands to setup the env for usb boot

```
>setenv boot usb
>setenv args_usb 'setenv bootargs ${bootargs} rootfstype=ext4 root=/dev/sda2 rw'
>setenv init_usb 'usb start; run set_fs_none args_all args_usb'
>setenv get_fdt_usb 'fatload usb 0:1 ${addr_fdt} ${name_fdt}'
>setenv get_kern_usb 'fatload usb 0:1 ${addr_kern} ${name_kern}'
>setenv get_mon_usb 'fatload usb 0:1 ${addr_mon} ${name_mon}'
>saveenv
>boot
IMPORTANT: Make sure "rootwait" (NOT "rootwait=1") appears in args_all
```

Here is the log using a USB 3.0 device. Boot takes about 1 minute

```
U-Boot 2013.01-00009-gd274e3f-dirty (Aug 29 2013 - 12:10:33)
```

```
I2C:   ready
DRAM:  1 GiB
NAND:  512 MiB
Net:   TCI6638_EMAC, TCI6638_EMAC1
Hit any key to stop autoboot:  0
(Re)start USB...
USB0:   No power optimization available
Register 2001040 NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus 0 for devices...
2 USB Device(s) found
      scanning usb for storage devices... 1 Storage Device(s) found
reading uImage-k2hk-evm.dtb
44562 bytes read in 1067 ms (40 KiB/s)
reading skern-keystone-evm.bin
45056 bytes read in 949 ms (45.9 KiB/s)
reading uImage-keystone-evm.bin
3989320 bytes read in 48025 ms (81.1 KiB/s)
## installed monitor, freq [194560000], status 0
## Booting kernel from Legacy Image at 88000000 ...
```

```
  Image Name:   Linux-3.8.4-rt2-01225-g62655f8
  Created:      2013-08-30  14:33:02 UTC
  Image Type:   ARM Linux Kernel Image (uncompressed)
  Data Size:    3989256 Bytes = 3.8 MiB
  Load Address: 80008000
  Entry Point:  80008000
  Verifying Checksum ... OK
## Flattened Device Tree blob at 87000000
  Booting using the fdt blob at 0x87000000
  Loading Kernel Image ... OK
OK
  Using Device Tree in place at 87000000, end 8700de11
```

```
Starting kernel ...
```

```
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 3.8.4-rt2-01225-g62655f8 (a0868495@ares-ubuntu.am.dhc
p.ti.com) (gcc version 4.7.3 20130226 (prerelease) (crosstool-NG linaro-1.13.1-4.7
-2013.03-20130313 - Linaro GCC 2013.03) ) #3 SMP PREEMPT RT Fri Aug 30 10:32:50 ED
T 2013
[    0.000000] CPU: ARMv7 Processor [412fc0f4] revision 4 (ARMv7), cr=30c7387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] Machine: KeyStone2, model: Texas Instruments Keystone 2 SoC
[    0.000000] switching to high address space at 0x800000000
[    0.000000] cma: CMA: reserved 16 MiB at 1f000000
[    0.000000] Memory policy: ECC disabled, Data cache writealloc
[    0.000000] PERCPU: Embedded 9 pages/cpu @c0bd2000 s12864 r8192 d15808 u36864
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pages
: 130048
[    0.000000] Kernel command line: console=ttyS0,115200n8 rootwait=1 rootwait=1 r
ootfstype=ext4 rootwait root=/dev/sda2 rw ip=dhcp
[    0.000000] PID hash table entries: 2048 (order: 1, 8192 bytes)
[    0.000000] Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
[    0.000000] Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
[    0.000000] __ex_table already sorted, skipping sort
[    0.000000] Memory: 512MB = 512MB total
[    0.000000] Memory: 495164k/495164k available, 29124k reserved, 0K highmem
[    0.000000] Virtual kernel memory layout:
[    0.000000]     vector  : 0xffff0000 - 0xffff1000   (   4 kB)
[    0.000000]     fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
[    0.000000]     vmalloc : 0xe0800000 - 0xff000000   ( 488 MB)
[    0.000000]     lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
[    0.000000]     pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
[    0.000000]     modules : 0xbf000000 - 0xbfe00000   (  14 MB)
[    0.000000]       .text : 0xc0008000 - 0xc070243c   (7146 kB)
[    0.000000]       .init : 0xc0703000 - 0xc0742240   ( 253 kB)
[    0.000000]       .data : 0xc0744000 - 0xc078b8e0   ( 287 kB)
[    0.000000]        .bss : 0xc078b8e0 - 0xc07b63ac   ( 171 kB)
[    0.000000] SLUB: Genslabs=11, HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nod
es=1
[    0.000000] Preemptible hierarchical RCU implementation.
[    0.000000] NR_IRQS:16 nr_irqs:16 16
[    0.000000] ipc irq: irqchip registered, range 512-539
[    0.000000] main_pll_clk rate is 1167360000, postdiv = 2, mult = 18,prediv = 0
[    0.000000] pll_clk parent_rate(122880000 Hz), rate(327680000 Hz),postdiv = 6,
mult = 15, prediv = 0
[    0.000000] tci6614-timer: no matching node
[    0.000000] Architected local timer running at 194.56MHz (phys).
[    0.000000] Switching to timer-based delay loop
[    0.000000] sched_clock: 32 bits at 194MHz, resolution 5ns, wraps every 22075ms
[    0.000000] Console: colour dummy device 80x30
[    0.000145] Calibrating delay loop (skipped), value calculated using timer freq
uency.. 389.12 BogoMIPS (lpj=1945600)
[    0.000148] pid_max: default: 4096 minimum: 301
[    0.000300] Mount-cache hash table entries: 512
[    0.011993] CPU: Testing write buffer coherency: ok
[    0.012169] CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
[    0.012204] Setting up static identity map for 0x80517268 - 0x8051729c
[    0.182030] CPU1: Booted secondary processor
[    0.182044] CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
[    0.261490] CPU2: Booted secondary processor
[    0.261504] CPU2: thread -1, cpu 2, socket 0, mpidr 80000002
[    0.341575] CPU3: Booted secondary processor
[    0.341590] CPU3: thread -1, cpu 3, socket 0, mpidr 80000003
[    0.341652] Brought up 4 CPUs
[    0.341674] SMP: Total of 4 processors activated (1556.48 BogoMIPS).
[    0.360841] NET: Registered protocol family 16
[    0.384047] DMA: preallocated 256 KiB pool for atomic coherent allocations
[    0.395928] keystone2_pcie_serdes_setup
[    0.397997] keystone2_pcie_serdes_setup done
[    0.398015] hw-breakpoint: found 5 (+1 reserved) breakpoint and 4 watchpoint re
gisters.
[    0.398021] hw-breakpoint: maximum watchpoint size is 8 bytes.
[    0.411553] bio: create slab <bio-0> at 0
[    0.411792] keystone-pcie: keystone_pcie_rc_init - start
[    0.411941] MEM 0x0000000050000000..0x000000005fffffff -> 0x0000000050000000
[    0.411952] IO 0x0000000024000000..0x0000000024003fff -> 0x0000000000000000
[    0.411993] pcie - number of legacy irqs = 4
[    0.412054] pcie - number of MSI host irqs = 8, msi_irqs = 32
[    0.522253] keystone-pcie: Doing PCI Setup...Done
[    0.522260] keystone-pcie: Starting PCI scan...
[    0.522425] PCI host bridge to bus 0000:00
[    0.522438] pci_bus 0000:00: root bus resource [mem 0x50000000-0x5fffffff]
[    0.522446] pci_bus 0000:00: root bus resource [io  0x0000-0x3fff]
[    0.522454] pci_bus 0000:00: No busn resource found for root bus, will use [bus
```

```
 00-ff]
[    0.522497] PCI: bus0: Fast back to back transfers enabled
[    0.522515] keystone-pcie: Ending PCI scan...
[    0.522524] keystone-pcie: keystone_pcie_rc_init - end
[    0.522757] vgaarb: loaded
[    0.523196] SCSI subsystem initialized
[    0.523743] usbcore: registered new interface driver usbfs
[    0.523871] usbcore: registered new interface driver hub
[    0.524010] usbcore: registered new device driver usb
[    0.525636] pps_core: LinuxPPS API ver. 1 registered
[    0.525643] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giomett
i <giometti@linux.it>
[    0.525729] PTP clock support registered
[    0.525906] keystone-hwqueue hwqueue.4: qmgr start queue 0, number of queues 81
92
[    0.526024] keystone-hwqueue hwqueue.4: added qmgr start queue 0, num of queues
8192, reg_peek e0840000, reg_status e0804000, reg_config e0806000, reg_region e08
08000, reg_push e0880000, reg_pop e08c0000
[    0.526034] keystone-hwqueue hwqueue.4: qmgr start queue 8192, number of queues
8192
[    0.526144] keystone-hwqueue hwqueue.4: added qmgr start queue 8192, num of que
ues 8192, reg_peek e0900000, reg_status e080a400, reg_config e080c000, reg_region
e080e000, reg_push e0940000, reg_pop e0980000
[    0.527132] keystone-hwqueue hwqueue.4: qos: sched port @8096, drop sched @8000
[    0.528509] keystone-hwqueue hwqueue.4: qos: sched port @6496, drop sched @6400
[    0.529871] keystone-hwqueue hwqueue.4: added pool pool-net: 2048 descriptors o
f size 128
[    0.529882] keystone-hwqueue hwqueue.4: added pool pool-rio: 128 descriptors of
size 256
[    0.529892] keystone-hwqueue hwqueue.4: added pool pool-udma: 1636 descriptors
of size 256
[    0.529902] keystone-hwqueue hwqueue.4: added pool pool-xge: 2048 descriptors o
f size 128
[    0.529912] keystone-hwqueue hwqueue.4: added pool pool-crypto: 512 descriptors
 of size 128
[    0.533290] keystone-hwqueue hwqueue.4: registered queues 0-16383
[    0.533672] keystone-hwqueue hwqueue.4: qos version 0x2000105, magic valid
[    0.534173] keystone-hwqueue hwqueue.4: qos version 0x2000105, magic valid
[    0.541329] keystone-pktdma 2004000.pktdma: registered 24 logical channels, flo
ws 32, tx chans: 9, rx chans: 24
[    0.545326] keystone-pktdma 2a08000.pktdma: registered 24 logical channels, flo
ws 32, tx chans: 32, rx chans: 32, loopback
[    0.546110] keystone-pktdma 2fa1000.pktdma: registered 4 logical channels, flow
s 32, tx chans: 16, rx chans: 16
[    0.546301] Switching to clocksource arch_sys_counter
[    0.577408] NET: Registered protocol family 2
[    0.577857] TCP established hash table entries: 4096 (order: 3, 32768 bytes)
[    0.577936] TCP bind hash table entries: 4096 (order: 4, 114688 bytes)
[    0.578066] TCP: Hash tables configured (established 4096 bind 4096)
[    0.578105] TCP: reno registered
[    0.578117] UDP hash table entries: 256 (order: 2, 16384 bytes)
[    0.578143] UDP-Lite hash table entries: 256 (order: 2, 16384 bytes)
[    0.578361] NET: Registered protocol family 1
[    0.578574] RPC: Registered named UNIX socket transport module.
[    0.578581] RPC: Registered udp transport module.
[    0.578587] RPC: Registered tcp transport module.
[    0.578592] RPC: Registered tcp NFSv4.1 backchannel transport module.
[    0.579044] hw perfevents: enabled with ARMv7 Cortex-A15 PMU driver, 7 counters
available
[    0.667192] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[    0.667560] NTFS driver 2.1.30 [Flags: R/O].
[    0.667944] jffs2: version 2.2. (NAND) Â© 2001-2006 Red Hat, Inc.
[    0.668478] msgmni has been set to 999
[    0.669499] NET: Registered protocol family 38
[    0.669789] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 252
)
[    0.669798] io scheduler noop registered
[    0.669804] io scheduler deadline registered
[    0.670000] io scheduler cfq registered (default)
[    0.671576] keystone-udma udma0.5: registered udma device udma0
[    0.734545] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
[    0.736476] 2530c00.serial: ttyS0 at MMIO 0x2530c00 (irq = 309) is a 16550A
[    1.523400] console [ttyS0] enabled
[    1.527763] 2531000.serial: ttyS1 at MMIO 0x2531000 (irq = 312) is a 16550A
[    1.538861] loop: module loaded
[    1.542184] at24 0-0050: 131072 byte 24c1024 EEPROM, writable, 1 bytes/write
[    1.550398] Generic platform RAM MTD, (c) 2004 Simtec Electronics
[    1.564415] ONFI param page 0 valid
[    1.567913] ONFI flash detected
[    1.571064] NAND device: Manufacturer ID: 0x2c, Chip ID: 0xac (Micron MT29F4G08
ABBDAHC), 512MiB, page size: 2048, OOB size: 64
[    1.582770] Bad block table found at page 262080, version 0x02
[    1.589032] Bad block table found at page 262016, version 0x02
[    1.595089] nand_read_bbt: bad block at 0x000001a20000
[    1.600266] nand_read_bbt: bad block at 0x00001ce60000
[    1.605484] 3 ofpart partitions found on MTD device 30000000.nand
[    1.611596] Creating 3 MTD partitions on "30000000.nand":
[    1.617014] 0x000000000000-0x000000100000 : "u-boot"
[    1.622827] 0x000000100000-0x000000180000 : "params"
[    1.628599] 0x000000180000-0x000008000000 : "ubifs"
[    1.634328] davinci_nand 30000000.nand: controller rev. 2.5
[    1.640645] spi_davinci 21000400.spi: master is unqueued, this is deprecated
[    1.648106] m25p80 spi32766.0: found n25q128a11, expected n25q128
[    1.654221] m25p80 spi32766.0: n25q128a11 (16384 Kbytes)
[    1.659558] 2 ofpart partitions found on MTD device spi32766.0
[    1.665408] Creating 2 MTD partitions on "spi32766.0":
[    1.670562] 0x000000000000-0x000000080000 : "u-boot-spl"
[    1.676874] 0x000000080000-0x000001000000 : "test"
[    1.682874] spi_davinci 21000400.spi: Controller at 0xe0878400
[    1.689789] tun: Universal TUN/TAP device driver, 1.6
[    1.694854] tun: (C) 1999-2004 Max Krasnyansky <maxk@qualcomm.com>
[    1.701654] keystone-netcp 2f00000.netcp: No streaming regs defined
[    1.708067] keystone-netcp 2090000.netcp: cpts rftclk freq not defined
[    1.715696] keystone-netcp 2090000.netcp: Created interface "eth0"
[    1.721904] keystone-netcp 2090000.netcp: dma_chan_name nettx0
[    1.728761] keystone-netcp 2090000.netcp: Created interface "eth1"
[    1.734967] keystone-netcp 2090000.netcp: dma_chan_name nettx1
[    1.742144] keystone-dwc3 2690000.dwc: usbss revision 47914300
[    1.748031] keystone-dwc3 2690000.dwc: mapped irq 425 to virq 608
[    1.956808] xhci-hcd xhci-hcd: xHCI Host Controller
[    1.961718] xhci-hcd xhci-hcd: new USB bus registered, assigned bus number 1
[    1.969739] xhci-hcd xhci-hcd: irq 608, io mem 0x02690000
[    1.975240] usb usb1: New USB device found, idVendor=1d6b, idProduct=0002
[    1.982052] usb usb1: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    1.989300] usb usb1: Product: xHCI Host Controller
[    1.994191] usb usb1: Manufacturer: Linux 3.8.4-rt2-01225-g62655f8 xhci-hcd
[    2.001176] usb usb1: SerialNumber: xhci-hcd
[    2.005922] hub 1-0:1.0: USB hub found
[    2.009692] hub 1-0:1.0: 1 port detected
[    2.013864] xhci-hcd xhci-hcd: xHCI Host Controller
[    2.018765] xhci-hcd xhci-hcd: new USB bus registered, assigned bus number 2
[    2.025955] usb usb2: New USB device found, idVendor=1d6b, idProduct=0003
[    2.032766] usb usb2: New USB device strings: Mfr=3, Product=2, SerialNumber=1
[    2.040012] usb usb2: Product: xHCI Host Controller
[    2.044904] usb usb2: Manufacturer: Linux 3.8.4-rt2-01225-g62655f8 xhci-hcd
[    2.051888] usb usb2: SerialNumber: xhci-hcd
[    2.056645] hub 2-0:1.0: USB hub found
[    2.060414] hub 2-0:1.0: 1 port detected
[    2.064649] Initializing USB Mass Storage driver...
[    2.069687] usbcore: registered new interface driver usb-storage
[    2.075711] USB Mass Storage support registered.
[    2.080647] mousedev: PS/2 mouse device common for all mice
[    2.086548] i2c /dev entries driver
[    2.090806] watchdog 22f0080.wdt: heartbeat 60 sec
[    2.097888] keystone-crypto 20c0000.crypto: crypto accelerator enabled
[    2.104879] usbcore: registered new interface driver usbhid
[    2.110470] usbhid: USB HID core driver
[    2.114750]  remoteproc0: 2620040.dsp0 is available
[    2.119643]  remoteproc0: Note: remoteproc is still under development and consi
dered experimental.
[    2.128633]  remoteproc0: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
```

```
compatibility isn't yet guaranteed.
[    2.139060]  remoteproc0: no firmware found
[    2.143397] rproc-user 2620040.dsp0: registered misc device dsp0
[    2.149740]  remoteproc1: 2620044.dsp1 is available
[    2.154631]  remoteproc1: Note: remoteproc is still under development and consi
dered experimental.
[    2.163623]  remoteproc1: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.174048]  remoteproc1: no firmware found
[    2.178379] rproc-user 2620044.dsp1: registered misc device dsp1
[    2.184719]  remoteproc2: 2620048.dsp2 is available
[    2.189612]  remoteproc2: Note: remoteproc is still under development and consi
dered experimental.
[    2.198600]  remoteproc2: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.209064]  remoteproc2: no firmware found
[    2.213487] rproc-user 2620048.dsp2: registered misc device dsp2
[    2.219843]  remoteproc3: 262004c.dsp3 is available
[    2.224734]  remoteproc3: Note: remoteproc is still under development and consi
dered experimental.
[    2.233723]  remoteproc3: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.244149]  remoteproc3: no firmware found
[    2.248479] rproc-user 262004c.dsp3: registered misc device dsp3
[    2.254819]  remoteproc4: 2620050.dsp4 is available
[    2.259712]  remoteproc4: Note: remoteproc is still under development and consi
dered experimental.
[    2.266440] hub 1-0:1.0: unable to enumerate USB device on port 1
[    2.274811]  remoteproc4: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.285242]  remoteproc4: no firmware found
[    2.289572] rproc-user 2620050.dsp4: registered misc device dsp4
[    2.295905]  remoteproc5: 2620054.dsp5 is available
[    2.300797]  remoteproc5: Note: remoteproc is still under development and consi
dered experimental.
[    2.309787]  remoteproc5: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.320212]  remoteproc5: no firmware found
[    2.324547] rproc-user 2620054.dsp5: registered misc device dsp5
[    2.330885]  remoteproc6: 2620058.dsp6 is available
[    2.335776]  remoteproc6: Note: remoteproc is still under development and consi
dered experimental.
[    2.344765]  remoteproc6: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.355192]  remoteproc6: no firmware found
[    2.359519] rproc-user 2620058.dsp6: registered misc device dsp6
[    2.365870]  remoteproc7: 262005c.dsp7 is available
[    2.370762]  remoteproc7: Note: remoteproc is still under development and consi
dered experimental.
[    2.379750]  remoteproc7: THE BINARY FORMAT IS NOT YET FINALIZED, and backward
compatibility isn't yet guaranteed.
[    2.390175]  remoteproc7: no firmware found
[    2.394505] rproc-user 262005c.dsp7: registered misc device dsp7
[    2.400557] rproc-user dspmem.3: kick gpio
[    2.404798] rproc-user dspmem.3: registered misc device dspmem
[    2.410817] GACT probability on
[    2.413967] Mirror/redirect action on
[    2.417641] Simple TC action Loaded
[    2.421669] netem: version 1.3
[    2.424734] u32 classifier
[    2.427448]     Performance counters on
[    2.431290]     input device check on
[    2.434958]     Actions configured
[    2.438379] Netfilter messages via NETLINK v0.30.
[    2.443107] nf_conntrack version 0.5.0 (7992 buckets, 31968 max)
[    2.449469] ctnetlink v0.93: registering with nfnetlink.
[    2.455114] IPv4 over IPv4 tunneling driver
[    2.459827] gre: GRE over IPv4 demultiplexor driver
[    2.464719] ip_gre: GRE over IPv4 tunneling driver
[    2.470185] ip_tables: (C) 2000-2006 Netfilter Core Team
[    2.475612] ipt_CLUSTERIP: ClusterIP Version 0.8 loaded successfully
[    2.482023] arp_tables: (C) 2002 David S. Miller
[    2.486697] TCP: cubic registered
[    2.490018] Initializing XFRM netlink socket
[    2.495227] NET: Registered protocol family 10
[    2.500552] NET: Registered protocol family 17
[    2.505028] NET: Registered protocol family 15
[    2.509539] Bridge firewalling registered
[    2.513564] Ebtables v2.0 registered
[    2.517249] 8021q: 802.1Q VLAN Support v1.8
[    2.521741] sctp: Hash tables configured (established 4096 bind 4681)
[    2.528729] NET: Registered protocol family 40
[    2.533347] VFP support v0.3: implementor 41 architecture 4 part 30 variant f r
ev 0
[    2.541048] Registering SWP/SWPB emulation handler
[    2.546470] usb 2-1: new SuperSpeed USB device number 2 using xhci-hcd
[    2.547633] input: gpio_keys.7 as /devices/soc.0/gpio_keys.7/input/input0
[    2.554138] keystone-netcp 2090000.netcp: initializing cpsw version 1.3 (1) SGM
II identification value 0x4ed1
[    2.554301] keystone-netcp 2090000.netcp: Created a cpsw ale engine
[    2.554307] keystone-netcp 2090000.netcp: initialized cpsw ale revision 1.3
[    2.557389] pps pps0: new PPS source ptp0
[    2.557412] cpts rftclk rate(583680000 HZ),mult(1839607018),shift(30)
[    2.597538] keystone-netcp 2090000.netcp: Using Packet Accelerator Firmware ver
```

```
sion 0x01030008
[    2.606188] keystone-netcp 2090000.netcp: pa_clk_rate(163840000 HZ),mult(25000)
,shift(12)
[    2.606609] usb 2-1: Parent hub missing LPM exit latency info.  Power managemen
t will be impacted.
[    2.606962] usb 2-1: New USB device found, idVendor=05dc, idProduct=a203
[    2.606966] usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[    2.606970] usb 2-1: Product: JumpDrive
[    2.606973] usb 2-1: Manufacturer: Lexar
[    2.606977] usb 2-1: SerialNumber: 09020000003802162764
[    2.636458] scsi0 : usb-storage 2-1:1.0
[    2.658553] keystone-netcp 2090000.netcp: netcp device eth0 opened
[    2.667093] 8021q: adding VLAN 0 to HW filter on device eth0
[    2.672772] keystone-netcp 2090000.netcp: adding rx vlan id: 0
[    2.679143] keystone-netcp 2090000.netcp: initializing cpsw version 1.3 (1) SGM
II identification value 0x4ed1
[    2.695045] keystone-netcp 2090000.netcp: netcp device eth1 opened
[    2.701867] IPv6: ADDRCONF(NETDEV_UP): eth1: link is not ready
[    2.707719] 8021q: adding VLAN 0 to HW filter on device eth1
[    2.713399] keystone-netcp 2090000.netcp: adding rx vlan id: 0
[    2.736385] Sending DHCP requests ., OK
[    2.786410] IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
[    2.806392] IP-Config: Got DHCP answer from 158.218.104.3, my address is 158.21
8.104.111
[    3.636778] scsi 0:0:0:0: Direct-Access     Lexar    JumpDrive        1.00 PQ:
0 ANSI: 5
[    3.645783] sd 0:0:0:0: [sda] 31293440 512-byte logical blocks: (16.0 GB/14.9 G
iB)
[    3.645884] sd 0:0:0:0: Attached scsi generic sg0 type 0
[    3.658873] sd 0:0:0:0: [sda] Write Protect is off
[    3.663853] sd 0:0:0:0: [sda] Write cache: disabled, read cache: disabled, does
n't support DPO or FUA
[    3.674832]  sda: sda1 sda2
[    3.678982] sd 0:0:0:0: [sda] Attached SCSI removable disk
[    4.256594] keystone-netcp 2090000.netcp: removing rx vlan id: 0
[    4.263472] IP-Config: Complete:
[    4.266713]      device=eth0, hwaddr=00:17:ea:f4:46:0b, ipaddr=158.218.104.111,
mask=255.255.255.0, gw=158.218.104.1
[    4.277271]      host=158.218.104.111, domain=am.dhcp.ti.com, nis-domain=(none)
[    4.284603]      bootserver=0.0.0.0, rootserver=0.0.0.0, rootpath=
[    4.290629]      nameserver0=158.218.108.21, nameserver1=157.170.32.67
[    4.305325] EXT4-fs (sda2): mounted filesystem with ordered data mode. Opts: (n
ull)
[    4.313039] VFS: Mounted root (ext4 filesystem) on device 8:2.
[    4.319097] Freeing init memory: 252K
INIT: version 2.88 booting
Starting udev
```

```
e or directory
Starting Bootlog daemon: bootlogd.
[    5.869741] EXT4-fs (sda2): re-mounted. Opts: data=ordered
Configuring network interfaces... udhcpc (v1.20.2) started
Sending discover...
Sending select for 158.218.104.111...
Lease of 158.218.104.111 obtained, lease time 3600
/etc/udhcpc.d/50default: Adding DNS 158.218.108.21
/etc/udhcpc.d/50default: Adding DNS 157.170.32.67
done.
Wed May 15 06:38:00 UTC 2013
Starting telnet daemon.
Starting syslogd/klogd: done
Starting thttpd.
Stopping Bootlog daemon: bootlogd.
```

```
_____                    _____           _         _
|  _  |___ ___ ___ ___   |  _  |___ ___  |_|___ ___| |_
|     |  _| .'| . | . |  |   __|  _| . | | | -_|  _|  _|
|__|__|_| |__,|_  |___|  |__|  |_| |___|_| |___|___|_|
              |___|                    |___|
```

```
Arago Project http://arago-project.org keystone-evm ttyS0
```

```
Arago 2013.04 keystone-evm ttyS0
```

```
keystone-evm login: root
```

Why can't I connect to DSP cores from CCS in latest u-boot?

In the latest u-boot, we have DSP cores powered OFF by default. Need to set debug_options=1 U-Boot env variable and reset the board to get this working

```
>setenv debug_options 1
>saveenv
>reset
```

How to turn OFF ARM core 0?

U-Boot support turning OFF core 0. By default Core 1-3 are OFF. The following command can be used to turn OFF core 0

```
>setenv boot net
>run get_mon_net
>run run_mon
>killme
```

How to debug kernel through CCS?

Boot up EVM to u-boot. Startup CCS and Launch Target configuration. connect to ARM core using "connect to target from right click menu" or cntrl-alt-c. To do source level debugging, first enable source look up (right click on *.ccxml on the CCS window and choose "Edit source lookup" option. In the window, Right click on ARM core and choose "add" option. Choose "Path Mapping" in the next window. Assuming w:/linux-keystone on the windows machine is mapped to linux /home/<user-id>/linux-keystone, following entries are required

```
Compilation path \home\<user-id>\linux-keystone
Local file system path w:\linux-keystone
```

Click ok and complete the source lookup settings.

Boot to Linux. As soon as the Linux start booting, click suspend button on CCS window.

Now Load the symbol file, vmlinux that is available in the root directory of the kernel tree where the kernel build is done. Choose "Run" from the top level menu and then "load" -> "Load Symbol" and choose vmlinux.

Once loading completes, the source code will be shown with the PC pointing to current location of code execution. May put break point to stop kernel at specific point in Linux kernel boot up sequence.

How to workaround the UBI boot "bad image sequence number" problem ?

Sometimes the UBI image is not properly rebooted, the last PEB may get corrupted and cause the "bad image sequence number" error. This mostly happen when users logically configure a ubifs partition with the size smaller than the available NAND space size. E.g. the actual NAND size is 2GB on K2L EVM, but users only use 512MB for bootloader/env params/ubifs partitions.

When the problem happens, there is a workaround to scrub the entire NAND device by using the following U-boot command:

```
nand scrub.chip
```

How to boot a custom images on Keystone2 devices?

Examples for creation and flashing custom boot images on devices in Keystone2 family are described in the wiki article KeystoneII_Boot_Examples [84]

The article provides single stage and multi-stage boot examples from different boot modes. It also demonstrates formating uboot images for different boot media using the boot utilities provided in the software package.

Trouble shooting and FAQ

For K2e EVM, not able to do ">mon_install 0x0c5f0000" and stuck there ?

Probably, the MSMC RAM address might be not right. The MSMC RAM address for loading the *.dtb, uImage, skern is given below.

Please do "> print " to display the image loadable addresses and all the values of the env variables before loading the images to the EVM.

For K2E:-

>tftpboot 0x87000000 k2e-evm.dtb

>tftpboot 0x88000000 uImage-keystone-evm.bin

>tftpboot 0x0c140000 skern-k2e.bin

```
>mon_install 0x0c140000
>bootm 0x88000000 - 0x87000000
```

For K2H:-

```
>tftpboot 0x87000000 uImage-k2hk-evm.dtb
>tftpboot 0x88000000 uImage-keystone-evm.bin
>tftpboot 0x0c5f0000 skern-k2hk.bin
>mon_install 0x0c5f0000
>bootm 0x88000000 - 0x87000000
```

In CCS version: 5.5.0.00077, how to do target configuration in *.ccxml file as I do not see K2E option under "Board or device" column?

User has to install the emulation packages, "ti_emupack_keystone2_setup_1.1.0.0.bin" for linux or "ti_emupack_keystone2_setup_1.1.0.0.exe" for Windows; After which it will list the "66AK2E05" in Board or device column.

you can download those packages from this link:http://software-dl.ti.com/sdoemb/sdoemb_public_sw/mcsdk/latest/index_FDS.html

After loading the loadlin-evm-uboot.js file, not able to connect to target.

In loadlin-evm-uboot.js file, please check that the session name matches correctly.If not edit it.

For example, change like below:

```
//var sessionName = "Texas Instruments XDS2xx USB Emulator_0/CortexA15_1";
var sessionName = "Texas Instruments XDS2xx USB Emulator_0/arm_A15_0";
```

MCSDK Utility App Demonstration Guide

Multicore Software Development Kit
Utility Application
Demonstration Guide

Last updated: 09/20/2015

Overview

Utility applicaton demo is a Matrix GUI v2 based demo that can be remotely launched by the Matrix App Launcher. Please refer to http://processors.wiki.ti.com/index.php/Matrix_Users_Guide [1] for details.

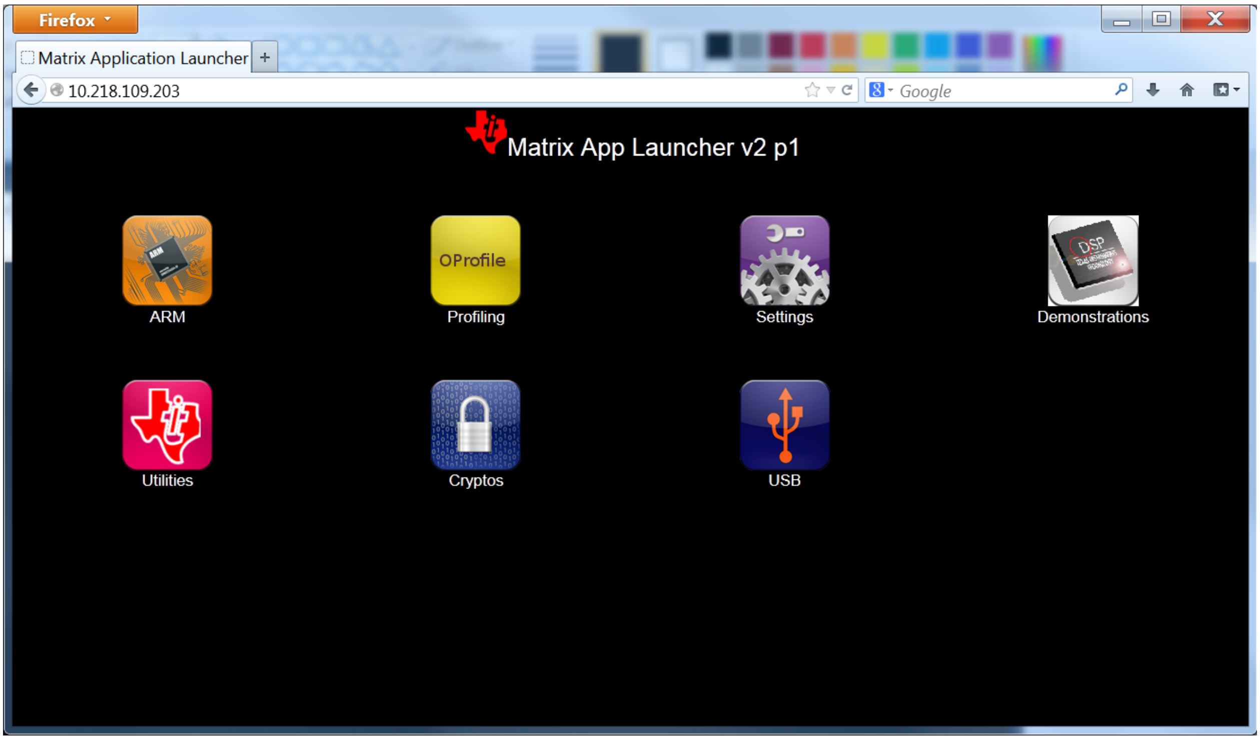
Software Design

The demo applications are text based shell scripts with cgi support. It includes the following applications:

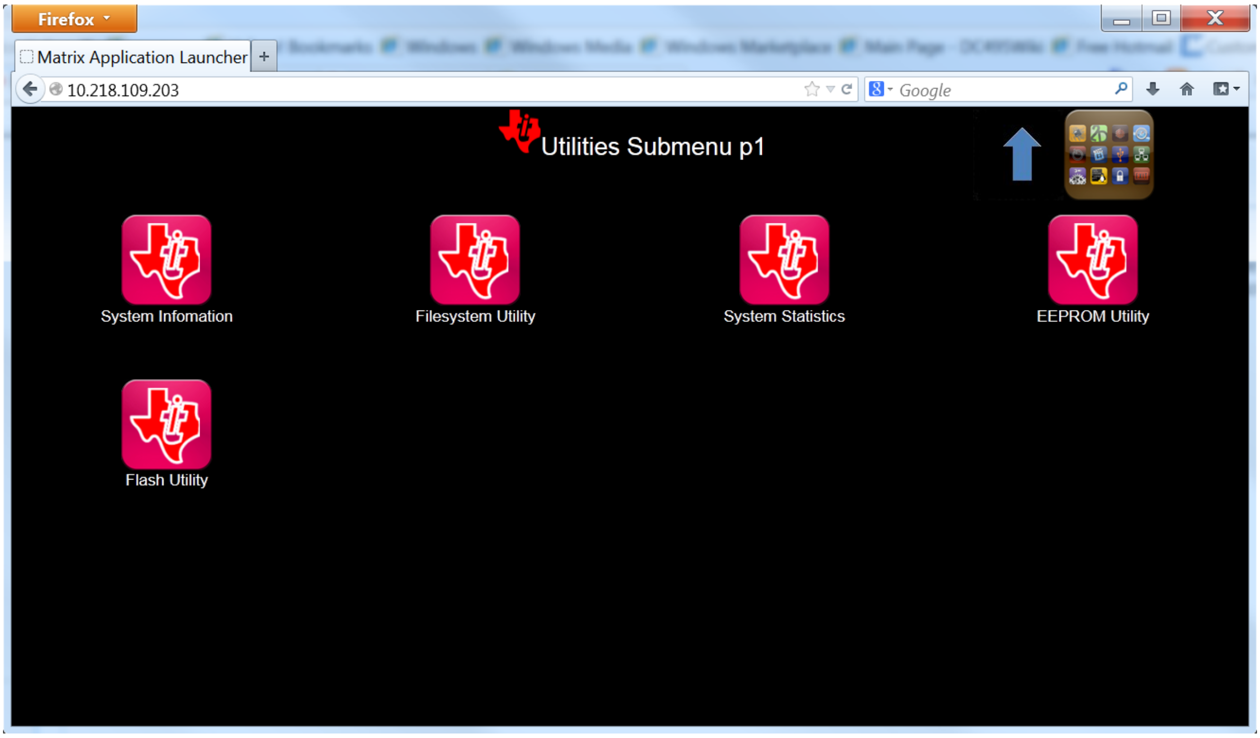
- Utility to show system Infomation
- Utility to show system statistics
- Utilities to read/write eeprom/flash devices
- Utility to write a file to the file system

Run Instructions

The Matrix application launcher will start automatically when the system boots up. Once the target EVM gets an IP address assigned by the DHCP server, type the IP address to an Internet browser running on a remote computer, which is connected to the same network as the target EVM does.



Click the Utilities icon and follow the instructions to run the demos.



MCSDK Image Processing Demonstration Guide

Multicore Software Development Kit

Image Processing

Demonstration Guide

Last updated: 09/20/2015

Overview

The Image Processing Demonstration illustrates the integration of key components in the Multicore Software Development Kit (MCSDK) on Texas Instruments (TI) multicore DSPs and System-on-Chips. The purpose of the demonstration is to provide a multicore software development framework on an evaluation module (EVM).

This document covers various aspects of the demonstration application, including a discussion on the requirements, software design, instructions to build and run the application, and troubleshooting steps. Currently, only SYS/BIOS is supported as the embedded OS.

This application shows implementation of an image processing system using a simple multicore framework. This application will run TI image processing kernels (a.k.a, *imagelib*) on multiple cores to do image processing (eg: edge detection, etc) on an input image.

There are three different versions of this demonstration that are included in the MCSDK. However, not all three versions are available for all platforms.

- *Serial Code*: This version uses file i/o to read and write image file. It can run on the simulator or an EVM target platform. The primary objective of this version of the demo is to run Prism and other software tools on the code to analyze the basic image processing algorithm.
- *IPC Based*: The IPC based demo uses SYS/BIOS IPC to communicate between cores to perform an image processing task parallel. See below for details.

- *OpenMP Based*: (Not available for C6657) This version of the demo uses OpenMP to run the image processing algorithm on multiple cores.

Note: The current implementation of this demonstration is not optimized. It should be viewed as the initial implementation of the BIOS MCSDK software eco-system for creating an image processing functionality. Further analysis and optimization of the demonstration are under progress.

Note: There are three versions of the demo provided in the release. The IPC based version runs on multiple cores and shows explicit IPC programming framework. The serial version of the demo runs on the simulator. The OpenMP version uses OpenMP to communicate between cores to process the input images. Unless explicitly specified, the IPC based version is assumed in this document.

Note: Before running the demo, please follow the software Getting Started Guide to configure your EVM properly.

Requirements

The following materials are required to run this demonstration:

- TMS320C6x low cost EVMs [Check Image Processing release notes for supported platforms]
- Power cable
- Ethernet cable
- Windows PC with CCSv5

Software Design

The following block diagram shows the framework used to implement the image processing application:

The following diagram shows the software pipeline for this application: .

More about processing algorithms

The application will use imagelib APIs for its core image processing needs.

Following steps are performed for edge detection

- Split input image into multiple overlapping slices
- If it is a RGB image, separate out the Luma component (Y) for processing (See YCbCr [1] for further details)
- Run Sobel operator [2] (IMG_sobel_3x3_8) to get the gradient image of each slices
- Run the thresholding operation (IMG_thr_le2min_8) on the slices to get the edges
- Combine the slices to get the final output

Framework for multicore

The current framework for multicore is either IPC Message Queue based framework or OpenMP. Following are the overall steps (the master and threads will be run on 1 or more cores)

- The master thread will preprocess the input image (described in User interface section) to make a gray scale or luma image
- The master thread signal each slave thread to start processing and wait for processing complete signal from all slave threads
- The slave threads run edge detection function (described above) to generate output edge image of the slice
- Then the slave threads signal master thread indicating the processing completed

- Once master thread receives completion signal from all threads it proceeds with further user interface processing (described in User interface section)

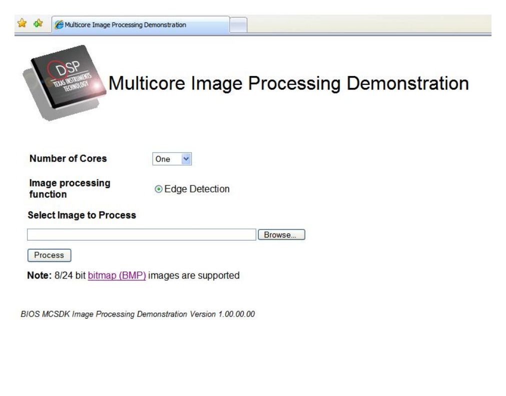
Profiling of the algorithm

- The profiling information live processing time will be presented at the end of the processing cycle
- Core image processing algorithms is instrumented using UIA for analysis and visualization using MCSA (Multicore System Analyzer)

User interface

The user input image will be a BMP image. The image will be transferred to external memory using NDK (http). Following are the states describing application user interface and their interaction

- At the time of bootup the board will bring configure IP stack with static/dynamic IP address and start a HTTP server
- The board will print the IP address in CCS console
- The user will use the IP address to open the index/input page (see link Sample Input Page)
- The application will support BMP image format
- The master thread will extract the RGB values from BMP image
- Then the master thread will initiate the image processing (as discussed above) and wait for its completion
- Once the processing completes, it will create output BMP image
- The master thread will put input/output images in the output page (see link Sample Output Page)


Multicore Image Processing Demonstration
DSP TEXAS INSTRUMENTS TECHNOLOGY
Browse...
Process

Multicore Image Processing Demonstration

Number of Cores: One

Image processing function: ● Edge Detection

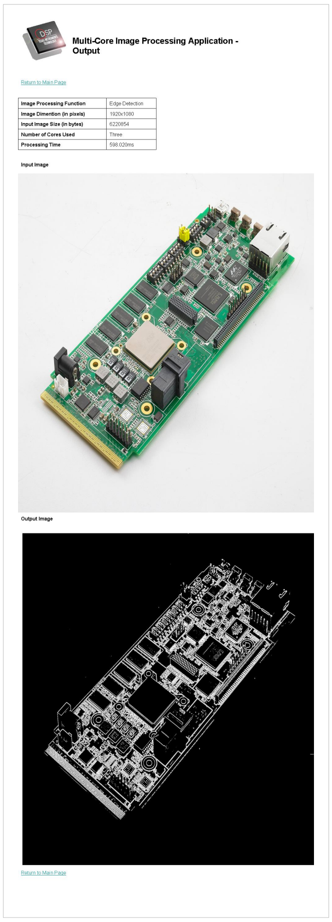
Select Image to Process

Note: 8/24 bit bitmap (BMP) images are supported

BIOS MCSDK Image Processing Demonstration Version 1.00.00.00

Software outline of the OpenMP demo

- The main task is called by OpenMP in core 0, spawns a task to initialize NDK, then gets/sets IP address and starts a web service to transfer user inputs and images. The main task then creates a mailbox and waits on a message post to the mailbox.
- The NDK calls a callback function to the application to retrieve the image data from user. The function reads the image and posts a message with the image information to the main task. Then it waits on a mailbox for a message post from main task.
- After receiving the message, the main task extracts RGB, splits the image into slices and processes each slices in different cores. A code snippet of this processing is provided below.



Multi-Core Image Processing Application - Output

Return to Main Page

Image Processing Function	Edge Detection
Image Dimention (in pixels)	1920x1080
Input Image Size (in bytes)	6220854
Number of Cores Used	Three
Processing Time	598.020ms

Input Image

Output Image

Return to Main Page

```
#pragma omp parallel for shared(p_slice, number_of_slices, ret_val)
private(i)
for (i = 0; i < number_of_slices; i++ ) {
  DEBUG_PRINT(printf("Processing slice # %d\n", i);)
    /* Process a slice */
    process_rgb (&p_slice[i]);
  if (p_slice[i].flag != 0) {
    printf("mc_process_bmp: Error in processing slice %d\n", i);
#pragma omp atomic
    ret_val = -1;
  }
  DEBUG_PRINT(printf("Processed slice # %d\n", i);)
}

if (ret_val == -1) {
  goto close_n_exit;
}
```

- After processing is complete, the main task creates the output image and sends the image information to the callback task using a message post. Then it waits on the mailbox again.
- The NDK callback task wakes up with the message post and sends the result image to the user.

Software outline of the Keystone II demo

The Image Processing Demo for Keystone II differs in the following ways from the original Image Processing Demo:

- The main process runs on the ARM cluster and is launched using the Matrix Application launcher.
- NDK is removed, as the web server is now run via Matrix.
- The DSP images are loaded using MPM.
- DDR3 memory is allocated by the ARM through requests to the DSP core0. The physical address returned by core0 is then mapped to user space.
- A diagram of the Keystone II demo framework is shown below:

Different Versions of Demo

Software Directory Structure Overview

The Image Processing Demonstration is present at <MCSDK INSTALL DIR>\demos\image_processing

- <MCSDK INSTALL DIR>\demos\image_processing\ipc\common directory has common slave thread functions which runs on all cores for the IPC based demo; The image processing function runs in this slave thread context
- <MCSDK INSTALL DIR>\demos\image_processing\ipc\master directory has main thread, which uses NDK to transfer images and IPC to communicate to other cores to process the images
- <MCSDK INSTALL DIR>\demos\image_processing\ipc\slave directory has the initialization function for all slave cores
- <MCSDK INSTALL DIR>\demos\image_processing\openmp\src directory has the main thread, which uses NDK to transfer images and OpenMP to communicate between cores to process the image
- <MCSDK INSTALL DIR>\demos\image_processing\ipc\evmc66##l\[master|slave] directories have the master and slave CCS project files for the IPC based demo
- <MCSDK INSTALL DIR>\demos\image_processing\openmp\evm66##l directory has the CCS project files for the OpenMP based demo
- <MCSDK INSTALL DIR>\demos\image_processing\######\evmc66##l\platform directory has the target configuration for the project
- <MCSDK INSTALL DIR>\demos\image_processing\serial directory has the serial version of the implementation
- <MCSDK INSTALL DIR>\demos\image_processing\utils directory has utilities used on the demo, like MAD config files
- <MCSDK INSTALL DIR>\demos\image_processing\images directory has sample BMP images

Serial Code

Run Instructions for Serial based demo application

The pre-compiled libraries are provided as a part of MCSDK release.

Please follow the procedures below to load images using CCS and run the demo.

Please refer the hardware setup guide for further the setup details.

- Connect the board to a Ethernet hub or PC using Ethernet cable.
- The demo runs in Static IP mode if User Switch 1 (SW9, position 2) is OFF else if it is ON then it runs DHCP mode. See the TMDXEVM6678L EVM Hardware Setup for the location of User Switch 1.
- If it is configured in static IP mode, the board will come up with IP address 192.168.2.100, GW IP address 192.168.2.101 and subnet mask 255.255.254.0

- If it is configured in DHCP mode, it would send out DHCP request to get the IP address from a DHCP server in the network.
- There is one image to be loaded on core 0. The image name is <MCSDK INSTALL DIR>\demos\image_processing\serial\Debug\image_processing_serial_simc6678.out.
- Connect the debugger and power on the board.
- It should open debug perspective and open debug window.
- Connect to only core 0, if the board is in no-boot mode make sure gel file is run to initialize ddr.
- Load image_processing_seria_simc6678.out to core 0.
- Run the core 0, in the CIO window, the board should pint IP address information (eg: Network Added: If-1:192.168.2.100)
- Open a web browser in the PC connected to the HUB or the board.
- Enter the IP address of the board, it should open up the image processing demo web page.
- Please follow the instructions in the web page to run the demo.
- Note that sample BMP images are provided in <MCSDK INSTALL DIR>\demos\image_processing\images

Build Instructions for Serial based demo application

Please follow the steps below to re-compile the Serial based demo image (These steps assume you have installed the MCSDK and all dependent packages).

- Open CCS->Import Existing... tab and import project from <MCSDK INSTALL DIR>\demos\image_processing\serial.
- It should import image_processing_serial_simc6678 project.
- The project should build fine for Release and Debug profile.

IPC-Based

Run Instructions for IPC based demo application

The pre-compiled libraries are provided as a part of MCSDK release.

Please follow the procedures below to load images using CCS and run the demo.

Please refer the hardware setup guide for further the setup details.

- Connect the board to a Ethernet hub or PC using Ethernet cable.
- The demo runs in Static IP mode if User Switch 1 (SW9, position 2) is OFF else if it is ON then it runs in DHCP mode. See the TMDXEVM6678L EVM Hardware Setup for the location of User Switch 1.
- If it is configured in static IP mode, the board will come up with IP address 192.168.2.100, GW IP address 192.168.2.101 and subnet mask 255.255.254.0
- If it is configures in DHCP mode, it would send out DHCP request to get the IP address from a DHCP server in the network.
- There are two images to be loaded to master (core 0) and other cores. The core 0 to be loaded with <MCSDK INSTALL DIR>\demos\image_processing\ipc\evmc66##l\master\Debug\image_processing_evmc66##l_master.out image and other cores (referred as slave cores) to be loaded with <MCSDK INSTALL DIR>\demos\image_processing\ipc\evmc66##l\slave\Debug\image_processing_evmc66##l_slave.out image.
- Connect the debugger and power on the board.
- In CCS window, launch the target configuration file for the board.
- It should open debug perspective and open debug window with all the cores.
- Connect to all the cores and load image_processing_evmc66##l_master.out to core 0 and image_processing_evmc66##l_slave.out to all other cores.

- Run all the cores, in the CIO console window, the board should print IP address information (for eg: Network Added: If-1:192.168.2.100)
- Open a web browser in the PC connected to the HUB or the board.
- Enter the IP address of the board, it should open up the image processing demo web page.
- Please follow the instructions in the web page to run the demo.
- Note that, sample BMP images are provided in <MCSDK INSTALL DIR>\demos\image_processing\images

Note: If you want to run the demo in static IP address mode, make sure the host PC is in same subnet or can reach the gateway. A sample setup configuration is shown below.

In Windows environment

> Set up TCP/IP configuration of ‘Wired Network Connection’ as shown in Wired Network Connection in Windows.

In Linux environment

> Run following command to set the static IP address for the current login session on a typical Linux setup.

```
sudo ifconfig eth0 192.168.2.101 netmask 255.255.254.0
```

Build Instructions for IPC based demo application

Please follow the steps below to re-compile the IPC based demo image (These steps assume you have installed the MCSDK and all the dependent packages).

- Open CCS->Import Existing... tab and import project from <MCSDK INSTALL DIR>\demos\image_processing\ipc.
- It should import two projects image_processing_evmc66##l_master and image_processing_evmc66##l_slave.
- Right click on each project->Properties to open up the properties window.
- Goto CCS Build->RTSC and check if in other repository have link to <MCSDK INSTALL DIR> (the actual directory).
- If IMGLIB C66x is unchecked, please select 3.0.1.0 to check it.
- The RTSC platform should have demos.image_processing.evmc66##l.platform.
- The project should build fine.

CToolsLib IPC-Based

This demonstrates Common Platform Tracer (CPT) capabilities using image processing demo as an application. Please see below for details on the demo and steps to be followed to run it on the C6670/C6678/C6657 EVM. The instrumentation examples are provided on the IPC version of the demo. The following are the supported use cases:

1. System Bandwidth Profile
    1. Captures bandwidth of data paths from all system masters to the slave (or) slaves associated with one or more CP Tracers
    2. Demo: For the Image demo, instrument the external memory (MSMC & DDR3) bandwidth used by all system masters
2. System Latency Profile
    1. Captures Latency of data paths from all system masters to the slave (or) slaves associated with one or more CP Tracers
    2. Demo: For the Image demo, instrument the external memory (MSMC & DDR3) Latency values from all system masters
3. The CP tracer messages (system trace) can be exported out for analysis in 3 ways:

    1. System ETB drain using CPU (capture only 32KB (SYS ETB size in keystone devices) of system trace data)
    2. System ETB drain using EDMA (ETB extension to capture more than 32KB (SYS ETB size in keystone devices))
    3. External Emulator like XDS560 PRO or XDS560V2
4. Total Bandwidth Profile
    1. The bandwidth of data paths from a group of masters to a slave is compared with the total bandwidth from all masters to the same slave.
        1. The following statistics are captured:
            1. Percentage of total slave activity utilized by a selected group of masters
            2. Slave Bus Bandwidth (bytes per second) utilized by the selected group of masters
            3. Average Access Size of slave transactions (for all system masters)
            4. Bus Utilization (transactions per second) (for all system masters)
            5. Bus Contention Percentage (for all system masters)
            6. Minimum Average Latency (for all system masters)
        2. Demo: For Image Demo, compare DDR3 accesses by Core0 (master core) with the DDR3 accesses by all other masters (which includes Core1, Core2…)
5. Master Bandwidth Profile
    1. The bandwidth of data paths from two different group of masters to a slave is measured and compared.
        1. The following statistics are captured:
            1. Slave Bus Bandwidth (bytes per second) utilized by master group 0
            2. Slave Bus Bandwidth (bytes per second) utilized by master group 1
            3. Average Access Size of slave transactions (for both the master groups)
            4. Bus Utilization (transactions per second) (for both the master groups)
            5. Bus Contention Percentage (for both the master groups)
            6. Minimum Average Latency (for both the master groups)
        2. Demo: For Image Demo, compare DDR3 accesses by Core0 and Core1 with the DDR3 accesses by Core2 and Core3
6. Event Profile
    1. Capture new request transactions accepted by the slave. Filtering based on Master ID or address range is supported.
        1. The following statistics are captured for every new request event: Master ID which initiated this particular transaction
            1. Bus Transaction ID
            2. Read/Write Transaction
            3. 10 bits of the address (Address export mask selects, which particular 10 bits to export)
        2. Demo: For Image Demo, implement a Global SoC level watch point on a variable in DDR3 memory. Only Core0 is intended to read/write to this DDR3 variable. Unintended accesses by other masters (or Cores) should be captured and flagged.
7. Statistical profiling provides a light weight(based on the number of Trace samples needed to be captured) and coarse profiling of the entire application.
    1. PC trace is captured at periodic sampling intervals. By analyzing these PC trace samples, a histogram of all functions called in the application, with the following information:
        1. percent of total execution time
        2. number of times encountered

2. Demo: In the Image Demo, Statistically profile the process_rgb() function, which processes a single slice of the Image.

Run Instructions for CToolsLib IPC based demo applications

The instructions to run each CToolsLib demo are the same as the non-CToolsLib demo; however, before pressing 'run' in CCS, you must perform the following steps:

CCS v5.4

If using CCS v5.4 or later, use the following instructions. Please see the next section for earlier versions of CCS.

For the event_profiler, master_bandwidth, system_bandwidth, system_latency, and total_bandwidth demos:

- While in the CCS Debug perspective, load all cores as you normally would with the IPC Demo.
- Go to **Tools->Hardware Trace Analyzer->Custom System Trace.**
- Select the appropriate **Transport Type** (depending on how the board is connected, either **560 V2 Trace** or **Pro Trace**, **ETB** is also acceptable).
- If using **560 V2 Trace** or **Pro Trace**, select the following options:
    - Buffer Type: **Stop-on-full**
    - Buffer Size: **64 MB**
    - Number of Pins: **4 pin**
    - Check **Synchronize trace collection with target run and halt**.
- Click **Start**.
- Run the demo.
- In the **Trace Viewer** tab, click **Stop**.
- For master_bandwidth, system_bandwidth, system_latency and total_bandwidth; a graphical representation of the results can be generated by clicking **Analyze->Memory Throughput** in the **Trace Viewer** tab.
- See the section below for details on the output of projects.

For the mem_aetint_watchpoint and mem_pct_watchpoint demos:

- While in the CCS Debug perspective, load all cores as you normally would with the IPC Demo.
- With the first core selected, go to **Tools->Hardware Trace Analyzer->Custom PC Trace.**
- Select the **Transport Type** as **ETB**, make sure **Synchronize trace collection with target run and halt** is selected and click **Start**.
- Run the demo.
- After the demo is complete, click **Stop** in the **Trace Viewer** tab.
- See the section below for details on the output of projects.

For the statistical_profiling demo:

- While in the CCS Debug perspective, load all cores as you normally would with the IPC Demo.
- Create a folder named 'temp' located at C:\temp.
- Run the program. (Note: The application will take considerably longer than normal).
- The output of the PC trace will be present at C:\temp\StatProf_etbdata.bin.

Earlier CCS Versions

If using a versions of CCS earlier than 5.4, use the following instructions.

For the event_profiler, master_bandwidth, system_bandwidth, system_latency, and total_bandwidth demos:

- While in the CCS Debug perspective; right click in the "Debug" window and click "show all cores."
- Under the "Non Debuggable Devices" section that appears, right click on CSSTM_0 and click "connect."
- Go to Tools->Trace Control.
- Click Receiver and choose 560 V2 Trace.
- In the CSSTM_0 tab, in the Trace Settings section, choose 4 pin for the Port width, 8 MB as the buffer size, Stop-on-full and check Synchronize with target.
- Click Apply and/or OK.
- Go to Tools->Trace Analyzer->Open Trace Connection in New View and choose the option that ends in CCSTM_0.
- Run the program and perform the demo as you normally would.
- After the demo has been run, pause the application to view the results.
- For master_bandwidth, system_bandwidth, system_latency and total_bandwidth; a graphical representation of the results can be generated by going to Tools->Trace Analyzer->Analysis->STM Analysis->STM Analysis Graph.
- See the section below for details on the output of projects.

For the mem_aetint_watchpoint and mem_pct_watchpoint demos:

- Go to Tools->Trace Control.
- Select the C66x_0 tab, click receiver and choose ETB.
- Leave the values as default, and click Apply and/or OK.
- The Trace Display window should automatically open. Run the program.
- See the section below for details on expected output.

For the statistical_profiling demo:

- Create a folder named 'temp' located at C:\temp.
- Run the program. (Note: The application will take considerably longer than normal).
- The output of the ctools portion will be present at C:\temp\StatPof_etbdata.bin.

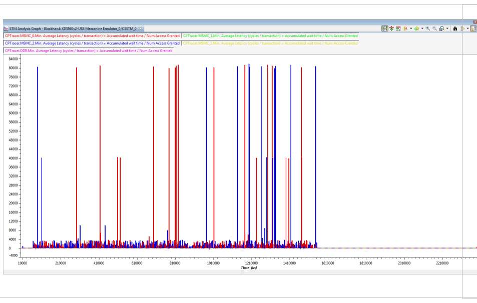
Expected Output of the CToolsLib Examples

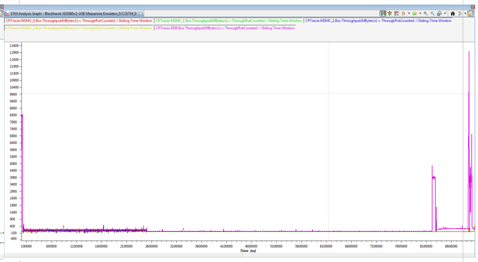
Total Bandwidth

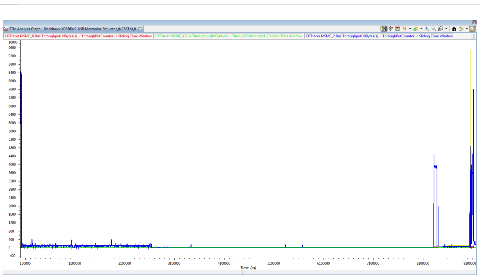
Below are three images showing DDR3 bandwidth by core0 (master) and the system, as well as the DDR3 latency.

System Latency, System Bandwidth and Master Bandwidth

Below are the expected results for each example.

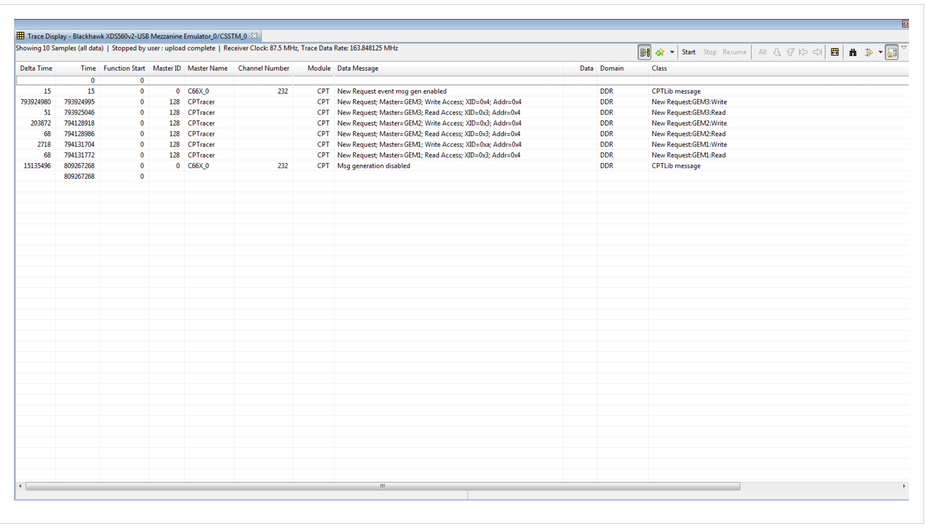

STM Analysis Graph - Blackhawk XDS560v2-USB Mezzanine Emulator_0/CSSTM_0
CPTracer.MSMC_0.Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted
CPTracer.MSMC_1.Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted
CPTracer.MSMC_2.Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted
CPTracer.MSMC_3.Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted
CPTracer.DDR.Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted
Time (ns)


STM Analysis Graph - Blackhawk XDS560v2-USB Mezzanine Emulator_0/CSSTM_0
CPTracer.MSMC_0.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
CPTracer.MSMC_1.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
CPTracer.MSMC_2.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
CPTracer.MSMC_3.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
CPTracer.DDR.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
Time (ns)


STM Analysis Graph - Blackhawk XDS560v2-USB Mezzanine Emulator_0/CSSTM_0
CPTracer.MSMC_0.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
CPTracer.MSMC_1.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
CPTracer.MSMC_2.Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window
Time (ns)

Event Profiler

Below is the expected result for the event_profiler example project. The project shows illegal accesses to a specific address by the slave cores (GEM1-GEM3).



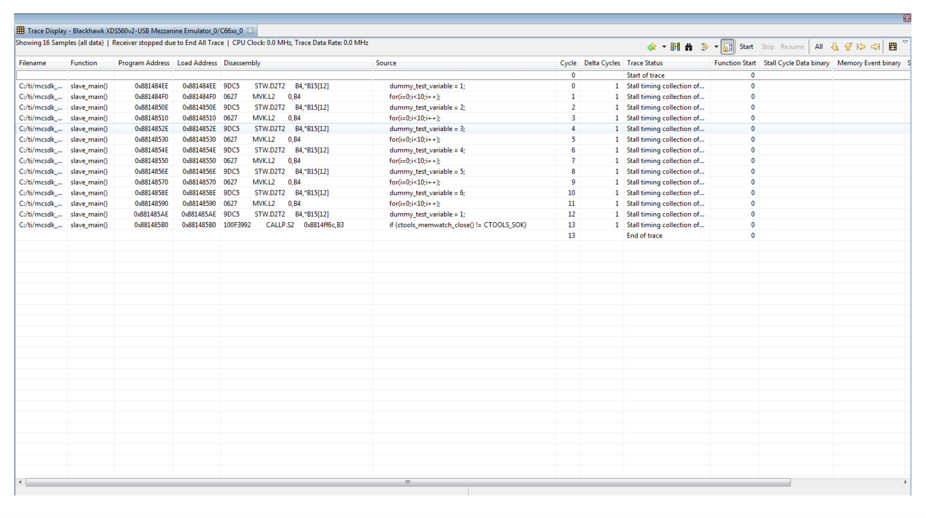
Trace Display - Blackhawk XDS560v2-USB Mezzanine Emulator_0/CSSTM_0

Showing 10 Samples (all data) | Stopped by user : upload complete | Receiver Clock: 87.5 MHz, Trace Data Rate: 163.848125 MHz

Delta Time	Time	Function Start	Master ID	Master Name	Channel Number	Module	Data Message	Data	Domain	Class
	0	0								
15	15	0	0	C66X_0	232	CPT	New Request event msg gen enabled		DDR	CPTLib message
793924980	793924995	0	128	CPTracer		CPT	New Request; Master=GEM3; Write Access; XID=0x4; Addr=0x4		DDR	New Request:GEM3:Write
51	793925046	0	128	CPTracer		CPT	New Request; Master=GEM3; Read Access; XID=0x3; Addr=0x4		DDR	New Request:GEM3:Read
203872	794128918	0	128	CPTracer		CPT	New Request; Master=GEM2; Write Access; XID=0x3; Addr=0x4		DDR	New Request:GEM2:Write
68	794128986	0	128	CPTracer		CPT	New Request; Master=GEM2; Read Access; XID=0x3; Addr=0x4		DDR	New Request:GEM2:Read
2718	794131704	0	128	CPTracer		CPT	New Request; Master=GEM1; Write Access; XID=0xa; Addr=0x4		DDR	New Request:GEM1:Write
68	794131772	0	128	CPTracer		CPT	New Request; Master=GEM1; Read Access; XID=0x3; Addr=0x4		DDR	New Request:GEM1:Read
15135496	809267268	0	0	C66X_0	232	CPT	Msg generation disabled		DDR	CPTLib message
	809267268	0								

Mem_PCT_WatchPoint

Below is the expected result for the mem_pct_watchpoint example project, which keeps watch on a dummy variable and counts the accesses to the variable.



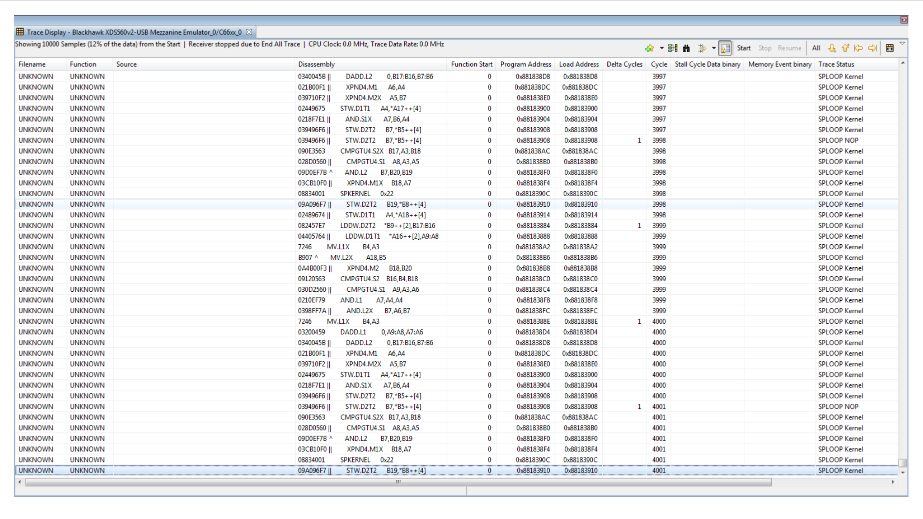
Trace Display - Blackhawk XDS560v2-USB Mezzanine Emulator_0/C66xx_0

Showing 16 Samples (all data) | Receiver stopped due to End All Trace | CPU Clock: 0.0 MHz, Trace Data Rate: 0.0 MHz

Filename	Function	Program Address	Load Address	Disassembly	Source	Cycle	Delta Cycles	Trace Status	Function Start	Stall Cycle Data binary	Memory Event binary	S
						0		Start of trace	0			
C:/ti/mcsdk_...	slave_main()	0x881484EE	0x881484EE	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 1;	0	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x881484F0	0x881484F0	0627 MVK.L2 0,B4	for(i=0;i<10;i++);	1	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x8814850E	0x8814850E	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 2;	2	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x88148510	0x88148510	0627 MVK.L2 0,B4	for(i=0;i<10;i++);	3	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x8814852E	0x8814852E	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 3;	4	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x88148530	0x88148530	0627 MVK.L2 0,B4	for(i=0;i<10;i++);	5	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x8814854E	0x8814854E	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 4;	6	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x88148550	0x88148550	0627 MVK.L2 0,B4	for(i=0;i<10;i++);	7	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x8814856E	0x8814856E	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 5;	8	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x88148570	0x88148570	0627 MVK.L2 0,B4	for(i=0;i<10;i++);	9	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x8814858E	0x8814858E	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 6;	10	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x88148590	0x88148590	0627 MVK.L2 0,B4	for(i=0;i<10;i++);	11	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x881485AE	0x881485AE	9DC5 STW.D2T2 B4,*B15[12]	dummy_test_variable = 1;	12	1	Stall timing collection of...	0			
C:/ti/mcsdk_...	slave_main()	0x881485B0	0x881485B0	100F3992 CALLP.S2 0x8814ff6c,B3	if (ctools_memwatch_close() != CTOOLS_SOK)	13	1	Stall timing collection of...	0			
						13		End of trace	0			

Mem_AETINT_WatchPoint

Below is the expected result for the mem_aetint_watchpoint example project, which throws an exception and halts the master (core0) when a particular variable is accessed. The point in the code where the first unintended access happened can be viewed in the results.



Trace Display - Blackhawk XDS560v2-USB Mezzanine Emulator_0/C66xx_0

Showing 10000 Samples (12% of the data) from the Start | Receiver stopped due to End All Trace | CPU Clock: 0.0 MHz, Trace Data Rate: 0.0 MHz

Filename	Function	Source	Disassembly	Function Start	Program Address	Load Address	Delta Cycles	Cycle	Stall Cycle Data binary	Memory Event binary	Trace Status
UNKNOWN	UNKNOWN		0340045B || DADD.L2 0,B17:B16,B7:B6	0	0x881838D8	0x881838D8		3997			SPLOOP Kernel
UNKNOWN	UNKNOWN		021B00F1 || XPND4.M1 A6,A4	0	0x881838DC	0x881838DC		3997			SPLOOP Kernel
UNKNOWN	UNKNOWN		039710F2 || XPND4.M2X A5,B7	0	0x881838E0	0x881838E0		3997			SPLOOP Kernel
UNKNOWN	UNKNOWN		02449675 STW.D1T1 A4,*A17++[4]	0	0x88183900	0x88183900		3997			SPLOOP Kernel
UNKNOWN	UNKNOWN		0218F7E1 || AND.S1X A7,B6,A4	0	0x88183904	0x88183904		3997			SPLOOP Kernel
UNKNOWN	UNKNOWN		039496F6 || STW.D2T2 B7,*B5++[4]	0	0x88183908	0x88183908		3997			SPLOOP Kernel
UNKNOWN	UNKNOWN		039496F6 || STW.D2T2 B7,*B5++[4]	0	0x88183908	0x88183908	1	3998			SPLOOP NOP
UNKNOWN	UNKNOWN		090E3563 CMPGTU4.S2X B17,A3,B18	0	0x881838AC	0x881838AC		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		028D0560 || CMPGTU4.S1 A8,A3,A5	0	0x881838B0	0x881838B0		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		09D0EF7B ^ AND.L2 B7,B20,B19	0	0x881838F0	0x881838F0		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		03CB10F0 || XPND4.M1X B18,A7	0	0x881838F4	0x881838F4		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		08834001 SPKERNEL 0x22	0	0x8818390C	0x8818390C		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		09A096F7 || STW.D2T2 B19,*B8++[4]	0	0x88183910	0x88183910		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		02489674 || STW.D1T1 A4,*A18++[4]	0	0x88183914	0x88183914		3998			SPLOOP Kernel
UNKNOWN	UNKNOWN		082457E7 LDDW.D2T2 *B9++[2],B17:B16	0	0x88183884	0x88183884	1	3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		04405764 || LDDW.D1T1 *A16++[2],A9:A8	0	0x88183888	0x88183888		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		7246 MV.L1X B4,A3	0	0x881838A2	0x881838A2		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		B907 ^ MV.L2X A18,B5	0	0x881838B6	0x881838B6		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		0A4B00F3 || XPND4.M2 B18,B20	0	0x881838B8	0x881838B8		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		09120563 CMPGTU4.S2 B16,B4,B18	0	0x881838C0	0x881838C0		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		030D2560 || CMPGTU4.S1 A9,A3,A6	0	0x881838C4	0x881838C4		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		0210EF79 AND.L1 A7,A4,A4	0	0x881838F8	0x881838F8		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		0398FF7A || AND.L2X B7,A6,B7	0	0x881838FC	0x881838FC		3999			SPLOOP Kernel
UNKNOWN	UNKNOWN		7246 MV.L1X B4,A3	0	0x8818388E	0x8818388E	1	4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		03200459 DADD.L1 0,A9:A8,A7:A6	0	0x881838D4	0x881838D4		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		0340045B || DADD.L2 0,B17:B16,B7:B6	0	0x881838D8	0x881838D8		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		021B00F1 || XPND4.M1 A6,A4	0	0x881838DC	0x881838DC		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		039710F2 || XPND4.M2X A5,B7	0	0x881838E0	0x881838E0		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		02449675 STW.D1T1 A4,*A17++[4]	0	0x88183900	0x88183900		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		0218F7E1 || AND.S1X A7,B6,A4	0	0x88183904	0x88183904		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		039496F6 || STW.D2T2 B7,*B5++[4]	0	0x88183908	0x88183908		4000			SPLOOP Kernel
UNKNOWN	UNKNOWN		039496F6 || STW.D2T2 B7,*B5++[4]	0	0x88183908	0x88183908	1	4001			SPLOOP NOP
UNKNOWN	UNKNOWN		090E3563 CMPGTU4.S2X B17,A3,B18	0	0x881838AC	0x881838AC		4001			SPLOOP Kernel
UNKNOWN	UNKNOWN		028D0560 || CMPGTU4.S1 A8,A3,A5	0	0x881838B0	0x881838B0		4001			SPLOOP Kernel
UNKNOWN	UNKNOWN		09D0EF7B ^ AND.L2 B7,B20,B19	0	0x881838F0	0x881838F0		4001			SPLOOP Kernel
UNKNOWN	UNKNOWN		03CB10F0 || XPND4.M1X B18,A7	0	0x881838F4	0x881838F4		4001			SPLOOP Kernel
UNKNOWN	UNKNOWN		08834001 SPKERNEL 0x22	0	0x8818390C	0x8818390C		4001			SPLOOP Kernel
UNKNOWN	UNKNOWN		09A096F7 || STW.D2T2 B19,*B8++[4]	0	0x88183910	0x88183910		4001			SPLOOP Kernel

Statistical Profiling

Please follow the steps mentioned on the Statistical Profiling wiki page (http://processors.wiki.ti.com/index.php/Statistical_Profiling) for decoding the StatPof_etbdata.bin file.

For more information, please refer to the CTools UCLib documentation, available under [<CTOOLSLIB INSTALL>\packages\ti\Ctools_UCLib\doc\Ctools_UCLib_html\index.html].

OpenMP-Based

Run Instructions for OpenMP based demo application

The pre-compiled libraries are provided as a part of MCSDK release.

Please follow the procedures below to load images using CCS and run the demo.

Please refer the hardware setup guide for further the setup details.

- Connect the board to a Ethernet hub or PC using Ethernet cable.
- The demo runs in Static IP mode if User Switch 1 (SW9, position 2) is OFF else if it is ON then it runs in DHCP mode. See the TMDXEVM6678L EVM Hardware Setup for the location of User Switch 1.
- If it is configured in static IP mode, the board will come up with IP address 192.168.2.100, GW IP address 192.168.2.101 and subnet mask 255.255.254.0
- If it is configures in DHCP mode, it would send out DHCP request to get the IP address from a DHCP server in the network.
- There ONE image to be loaded to core 0. The image name is <MCSDK INSTALL DIR>\demos\image_processing\openmp\evmc66##l\Release\image_processing_openmp_evmc66##l.out.
- Connect the debugger and power on the board.
- In CCS window, launch the target configuration file for the board.
- It should open debug perspective and open debug window.
- Connect to only core 0, if the board is in no-boot mode make sure gel file is run to initialize ddr.
- Load image_processing_openmp_evmc66##l.out to core 0.

- Run the core 0, in the CIO console window, the board should print IP address information (for eg: Network Added: If-1:192.168.2.100)
- Open a web browser in the PC connected to the HUB or the board.
- Enter the IP address of the board, it should open up the image processing demo web page.
- Please follow the instructions in the web page to run the demo.
- Note that, sample BMP images are provided in <MCSDK INSTALL DIR>\demos\image_processing\images

Note: If you want to run the demo in static IP address mode, make sure the host PC is in same subnet or can reach the gateway. A sample setup configuration is shown below.

In Windows environment

> Set up TCP/IP configuration of 'Wired Network Connection' as shown in Wired Network Connection in Windows.

In Linux environment

> Run following command to set the static IP address for the current login session on a typical Linux setup.

```
sudo ifconfig eth0 192.168.2.101 netmask 255.255.254.0
```

Build Instructions for OpenMP based demo application

Please follow the steps below to re-compile the OpenMP based demo image (These steps assume you have installed the MCSDK and all the dependent packages).

- Open CCS->Import Existing... tab and import project from <MCSDK INSTALL DIR>\demos\image_processing\openmp\evmc66##l.
- It should import image_processing_openmp_evmc66##l project.
- The project should build fine for Release and Debug profile.

Multicore System Analyzer integration and usage

The System Analyzer provides correlated realtime analysis and visibility into application running on single or multicore. Analysis and visibility includes Execution Graph, Duration Analysis, Context Aware Profile, Load Analysis and Statistics Analysis. Basic instrumentation using Unified Instrumentation Architecture (UIA) collects data in realtime and transport via Ethernet or JTAG to host where it it decoded, correlated, analyzed and visualized.

System Analyzer is automatically added to CCS5.0 by the MCSDK installer. CCS5.1 is shipped with the System Analyzer included.

The Image Processing Demo has been instrumented for duration/benchmark and CPU load analysis. Detailed information on running the demo with System Analyzer is provided in System Analyzer and the MCSDK Demo [3] page.

Image Processing Demo Analysis with Prism

This section we will use Prism software [4] to analyze the serial version of image processing demo. The steps below would assume Prism with C66x support is installed in the system and user completed the *Prism Getting Started - Tutorials* provided in the help menu.

Bring up the demo with Prism software

- Bring up the CCS as specified in the Prism documentation
- Edit *macros.ini* file from *<MCSDK INSTALL DIR>\demos\image_processing\serial* directory and change *../../../../imglib_c66x_#_#_#_#* to static path to IMGLIB package
- Open *Import Existing CCS Eclipse Project* and select search directory *<MCSDK INSTALL DIR>\demos\image_processing\serial*
- Check *Copy projects into workspace* and click *Finish*. This will copy the project to the workspace for Prism.
- Clean and re-compile the project
- Open the C6678 simulator, load the image to core0
- Open *Tools->GEL files*, select *Load GEL* and load/open *tisim_traces.gel* from *<CCSv5 INSTALL DIR>\ccs_base_####\simulation_csp_ny\env\ccs\import* directory
- Then select *CPU Register Trace->StartRegTrace* to start the trace, then run the program, wait till it finishes, then select *CPU Register Trace->StopRegTrace* to stop the trace
- Open *Prism->Show Prism Perspective*. It will start Prism Perspective
- Right click on the project and select *Create New PAD File*, it would open the New Prism Analysis Definition window, hit next
- It will open *Architecture Selection* window, select C6671 (single core) template. Then select Finish to open PAD file
- Select *Run->Debug Configurations->prismtrace*, this will convert the simulator generated traces to the traces required by Prism
- The PAD window should have filled in with default trace (PGSI, PGT) file names generated in above step
- Select *Read Trace* to read the trace
- After it read the trace, then select the complete trace from overview window and hit *Load Slice*
- The *Functions* tab will show the functions and their cycle information during the execution
- Observe the *Core 0* scheduling in the schedule window, you can place a marker for this run

What If analysis

The Prism tool allows user to analyze *What If* scenarios for the code

- *What If* the code is run on multiple cores
  - In the function window, right click on *process_rgb* function, select *Force Task* and hit *Apply*. This would make *process_rgb* function simulated as a separate task
  - In the *Architecture* tab, select *C6678 (8 Core)* template, hit *Apply*
  - Observe in *Schedule* tab, the change in execution when selected the *process_rgb* is simulated to run on 8 cores
  - A marker can be placed to compare the improvement
- *What If* the dependencies are removed
  - The *Dependencies* window helps to see and analyze the dependencies (which are preventing the task to be executed in multiple cores simultaneously)
  - Un-check *Serialized* check-boxes against dependency rows and hit *Apply*
  - Add the comparison marker in the *Schedule* tab and check the improvement

The Prism supports more functionality then described in this section. Please see Prism documentation for more information.

Multicore booting using MAD utilities

The detailed information on the Multicore Application Deployment a.k.a MAD utility is provided in MAD user guide [5] page.

This section will provide you the detail instructions on how the tool and boot the demo from flash/ethernet.

Linking and creating bootable application image using MAD utilities

The BIOS MCSDK installation provides MAD tool in *<MCSDK INSTALL DIR>\tools\boot_loader\mad-utils*. This package contains necessary tools to link the application to a single bootable image.

The image processing demo has following updates to create MAD image:

- The master and slave images are linked with *--dynamic* and *--relocatable* options.
- The MAD config files used to link the master and slave programs are provided in *<MCSDK INSTALL DIR>\demos\image_processing\utils\mad\evmc66##l\config-files*. Following are few items to note on the config file.
  - *maptoolCfg_evmc#####.json* has the directory and file name information for the tools
  - *deployment_template_evmc#####.json* has the deployment configuration (it has device name, partition and application information). Following are some more notes on the configuration file.
    - For C66x devices, the physical address is 36 bits and virtual address is 32 bits for external devices, this includes MSMC SRAM and DDR3 memory subsystem.
    - The *secNamePat* element string is a regular expression string.
    - The sections *bss*, *neardata*, *rodata* must be placed in one partition and in the order it is shown here
- The build script *<MCSDK INSTALL DIR>\demos\image_processing\utils\mad\evmc66##l\build_mad_image.bat* can be used to re-create the image

Note: The compilation will split out lots of warning like *Incompatible permissions for partition ...*, it can be ignored for now. This is due to mis-match in partition permissions wrt. the sections placed in the partition

- The bootable image is placed in *<MCSDK INSTALL DIR>\demos\image_processing\utils\mad\evmc66##l\images*

Pre-link bypass MAD image

Please see MAD user guide for more information on pre-link bypassed MAD image. The build script *build_mad_image_prelink_bypass.bat* can be used to build images with this mode.

Booting the application image using IBL

This image can be booted using IBL bootloader.

Following things to be noted on booting the image

- The image type/format is *ibl_BOOT_FORMAT_BBLOB*, so the IBL needs to be configured to boot this format
- The branch address (Branch address after loading) of the image [it is set to *0x9e001040* (or *0x80001040* if you are using BIOS MCSDK v 2.0.4 or prior) in MAL application], is different from default IBL boot address, so the IBL configuration needs to be updated to jump to this address

The following sections will outline the steps to boot the image from Ethernet and NOR using IBL. Please see IBL documentation on the detail information on booting.

Booting from Ethernet (TFTP boot)

- Change IBL configuration: The IBL configuration parameters are provided in a GEL file <*MCSDK INSTALL DIR*>*\tools\boot_loader\ibl\src\make\bin\i2cConfig.gel*. All the changes needs to be done in the function *setConfig_c66##_main()* of the gel file.
  - The IBL configuration file sets PC IP address 192.168.2.101, mask 255.255.255.0 and board IP address as 192.168.2.100 by default. If these address needs to be changed, open the GEL file, change *ethBoot.ethInfo* parameters in function *setConfig_c66##_main()*
  - Make sure the *ethBoot.bootFormat* is set to *ibl_BOOT_FORMAT_BBLOB*
  - Set the *ethBoot.blob.branchAddress* to *0x9e001040* (or *0x80001040* if you are using BIOS MCSDK v 2.0.4 or prior).
  - Note that the application name defaults to *app.out*

```
menuitem "EVM c66## IBL";

hotmenu setConfig_c66##_main()
{
 ibl.iblMagic = ibl_MAGIC_VALUE;
 ibl.iblEvmType = ibl_EVM_C66##L;

 ...

 ibl.bootModes[2].u.ethBoot.doBootp = FALSE;
 ibl.bootModes[2].u.ethBoot.useBootpServerIp = TRUE;
 ibl.bootModes[2].u.ethBoot.useBootpFileName = TRUE;
 ibl.bootModes[2].u.ethBoot.bootFormat = ibl_BOOT_FORMAT_BBLOB;


 SETIP(ibl.bootModes[2].u.ethBoot.ethInfo.ipAddr, 192,168,2,100);
 SETIP(ibl.bootModes[2].u.ethBoot.ethInfo.serverIp, 192,168,2,101);
 SETIP(ibl.bootModes[2].u.ethBoot.ethInfo.gatewayIp, 192,168,2,1);
 SETIP(ibl.bootModes[2].u.ethBoot.ethInfo.netmask, 255,255,255,0);

 ...

 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[0] = 'a';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[1] = 'p';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[2] = 'p';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[3] = '.';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[4] = 'o';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[5] = 'u';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[6] = 't';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[7] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[8] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[9] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[10] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[11] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[12] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[13] = '\0';
 ibl.bootModes[2].u.ethBoot.ethInfo.fileName[14] = '\0';

 ibl.bootModes[2].u.ethBoot.blob.startAddress = 0x9e000000 /*0x80000000
 for BIOS MCSDK v2.0.4 or prior*/; /* Load start address */
 ibl.bootModes[2].u.ethBoot.blob.sizeBytes = 0x20000000;
 ibl.bootModes[2].u.ethBoot.blob.branchAddress = 0x9e001040
/*0x80001040 for BIOS MCSDK v2.0.4 or prior*/; /* Branch address after
loading */

 ibl.chkSum = 0;
}
```

- Write IBL configuration:
  - Connect the board using JTAG, power on the board, open CCS, load the target and connect to core 0. Select *Tools->GEL Files* and in the GEL Files window right click and load GEL. Then select and load *<MCSDK INSTALL DIR>\tools\boot_loader\ibl\src\make\bin\i2cConfig.gel*.
  - Load I2C writer *<MCSDK INSTALL DIR>\tools\boot_loader\ibl\src\make\bin\i2cparam_0x51_c66##_le_0x500.out* to Core 0 and run. It will ask to run the GEL in console window. Run the GEL script from *Scripts->EVM c66##->setConfig_c66##_main*.
  - Open the CCS console window and hit enter to complete the I2C write.
- Booting the image:
  - Disconnect the CCS from board, power off the board.
  - Connect ethernet from board to switch/hub/PC and UART cables from board to PC.
  - Make sure your PC have the IP address specified above.
  - Set the board dip switches to boot from ethernet (TFTP boot) as specified in the hardware setup table (TMDXEVM6678L [6] TMDXEVM6670L [7])
  - Copy the demo image *<MCSDK INSTALL DIR>\demos\image_processing\utils\mad\evmc66##l\images\mcip-c66##-le.bin* to tftp directory and change its name to *app.out*
  - Start a tftp server and point it to the tftp directory
  - Power on the board. The image will be downloaded using TFTP to the board and the serial port console should print messages from the demo. This will also print the configured IP address of the board
  - Use the IP address to open the demo page in a browser and run the demo

Booting from NOR

- Change IBL configuration: The IBL configuration parameters are provided in a GEL file *<MCSDK INSTALL DIR>\tools\boot_loader\ibl\src\make\bin\i2cConfig.gel*. All the changes needs to be done in the function *setConfig_c66##_main()* of the gel file.
  - Make sure the *norBoot.bootFormat* is set to *ibl_BOOT_FORMAT_BBLOB*
  - Set the *norBoot.blob[0][0].branchAddress* to *0x9e001040* (or *0x80001040* if you are using BIOS MCSDK v 2.0.4 or prior)

```
menuitem "EVM c66## IBL";

hotmenu setConfig_c66##_main()
{
 ibl.iblMagic = ibl_MAGIC_VALUE;
 ibl.iblEvmType = ibl_EVM_C66##L;

 ...

 ibl.bootModes[0].bootMode = ibl_BOOT_MODE_NOR;
 ibl.bootModes[0].priority = ibl_HIGHEST_PRIORITY;
 ibl.bootModes[0].port = 0;

 ibl.bootModes[0].u.norBoot.bootFormat = ibl_BOOT_FORMAT_BBLOB;
 ibl.bootModes[0].u.norBoot.bootAddress[0][0] = 0; /* Image 0 NOR
offset byte address in LE mode */
 ibl.bootModes[0].u.norBoot.bootAddress[0][1] = 0xA00000; /* Image 1
NOR offset byte address in LE mode */
 ibl.bootModes[0].u.norBoot.bootAddress[1][0] = 0; /* Image 0 NOR
offset byte address in BE mode */
 ibl.bootModes[0].u.norBoot.bootAddress[1][1] = 0xA00000; /* Image 1
NOR offset byte address in BE mode */
 ibl.bootModes[0].u.norBoot.interface = ibl_PMEM_IF_SPI;
 ibl.bootModes[0].u.norBoot.blob[0][0].startAddress = 0x9e000000
/*0x80000000 for BIOS MCSDK v2.0.4 or prior*/; /* Image 0 load start
address in LE mode */
 ibl.bootModes[0].u.norBoot.blob[0][0].sizeBytes = 0xA00000; /* Image 0
 size (10 MB) in LE mode */
 ibl.bootModes[0].u.norBoot.blob[0][0].branchAddress = 0x9e001040
/*0x80001040 for BIOS MCSDK v2.0.4 or prior*/; /* Image 0 branch
address after loading in LE mode */

 ...

 ibl.chkSum = 0;
}
```

- Write IBL configuration:
  - Connect the board using JTAG, power on the board, open CCS, load the target and connect to core 0. Select *Tools->GEL Files* and in the GEL Files window right click and load GEL. Then select and load *<MCSDK INSTALL DIR>\tools\boot_loader\ibl\src\make\bin\i2cConfig.gel*.
  - Load I2C writer *<MCSDK INSTALL DIR>\tools\boot_loader\ibl\src\make\bin\i2cparam_0x51_c66##_le_0x500.out* to Core 0 and run. It will ask to run the GEL in console window. Run the GEL script from *Scripts->EVM c66##->setConfig_c66##_main*
  - Open the CCS console window and hit enter to complete the I2C write
- Write NOR image:
  - Copy application image (*<MCSDK INSTALL DIR>\demos\image_processing\utils\mad\evmc66##l\images\mcip-c66##-le.bin*) to *<MCSDK INSTALL DIR>\tools\writer\nor\evmc66##l\bin\app.bin*
  - Connect the board using JTAG, power on the board, open CCS, load the target and connect to core 0. Make sure the PLL and DDR registers are initialized from the platform GEL (if it is not done automatically, run

Global_Default_Setup function from the GEL file). Load image *<MCSDK INSTALL DIR>\tools\writer\nor\evmc66##l\bin\norwriter_evm66##l.out*

- Open memory window and load the application image (*<MCSDK INSTALL DIR>\demos\image_processing\utils\mad\evmc66##l\images\mcip-c66##-le.bin*) to address *0x80000000*
- Be sure of your **Type-size** choice 32 bits
- Hit run for NOR writer to write the image
- The CCS console will show the write complete message

- Boot from NOR:
  - Disconnect CCS and power off the board
  - Set the board dip switchs to boot from NOR (NOR boot on image 0) as specified in the hardware setup table (TMDXEVM6678L [6] TMDXEVM6670L [7])
  - Connect ethernet cable from board to switch/hub
  - Connect serial cable from board to PC and open a serial port console to view the output
  - Power on the board and the image should be booted from NOR and the console should show bootup messages
  - The demo application will print the IP address in the console
  - Use the IP address to open the demo page in a browser and run the demo

Performance numbers of the demo

The following table compares the performance between OpenMP and explicit IPC based image processing demo applications.

Note: The numbers are based on a **non-DMA** based implementation with **non-optimized RGB to Y** kernel. The L2 cache is set to 256KB.

Note: The results shown were taken during the BIOS-MCSDK 2.1.0 beta, results taken with current component versions may vary.

	Processing time for a ~16MB BMP image (in msec)	
Number of Cores	**OpenMP based demo**	**Explicit IPC based demo**
1	290.906	290.378
2	150.278	149.586
3	101.697	100.75
4	77.147	77.485
5	63.154	63.318
6	54.709	54.663
7	49.144	47.659
8	42.692	42.461

The performance numbers seem to be similar in both cases. This might be due to the nature of the demo application. It spends most of its processing time on actual image processing and sends, at most, 16 IPC messages between cores. So the contribution of the communication delays (IPC vs. OMP/IPC) are very minimal compared to any significant difference in processing times.

MCSDK Inter Processor Communication Demonstration Guide

Multicore Software Development Kit
Inter-Processor Communication
Demonstration Guide

Last updated: 09/20/2015

Overview

The Inter-Processor Communications (IPC) demo provides an example that can be run on the EVM.

Requirements

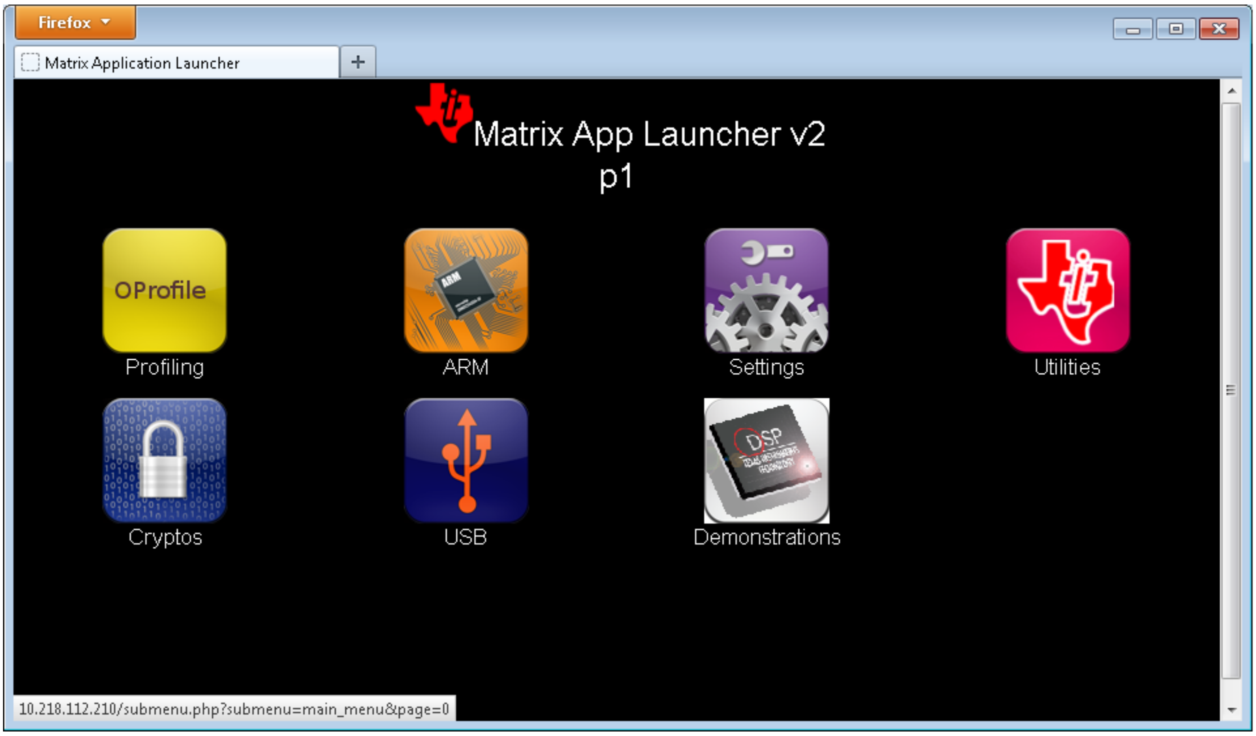
The aim of this demo is to show Inter processor control path communication between A15 Host and C66x DSP.

Software Design

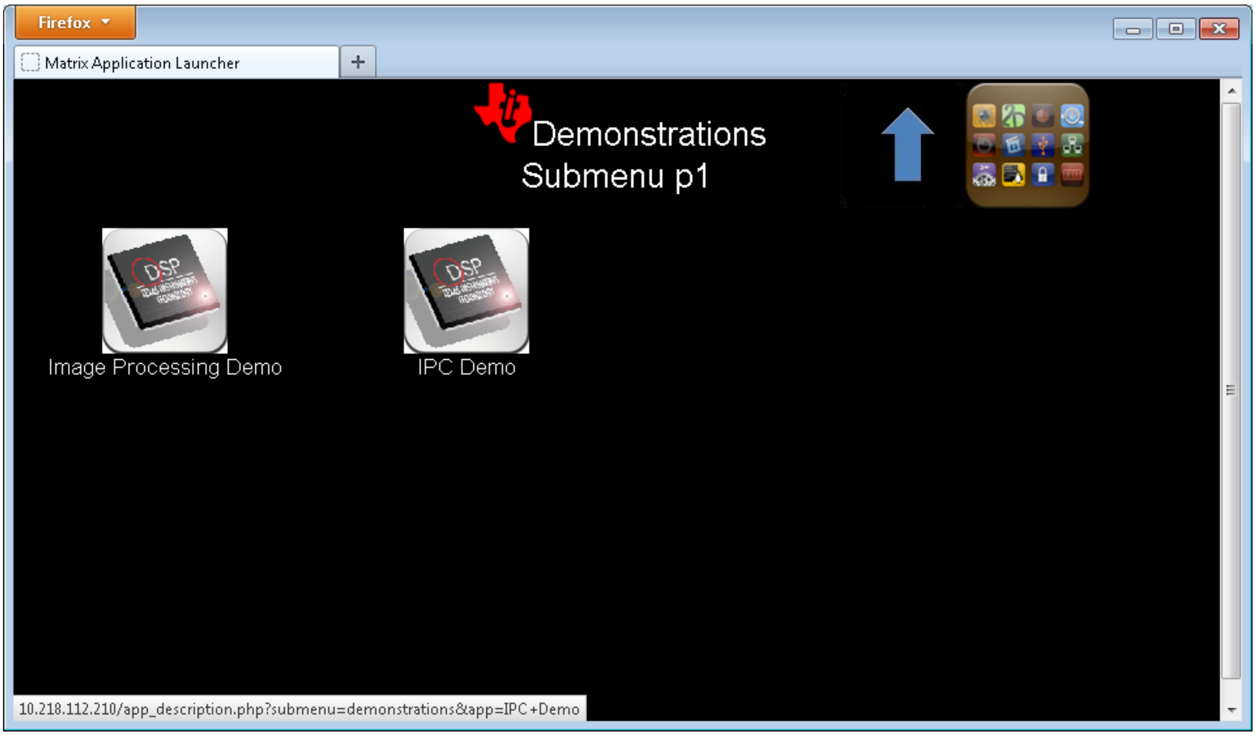
The IPC in this context is MessageQ based IPC between ARM to DSP . This transport is based on rpmsg for communication between ARM and DSPs, and a proprietary shared memory protocol for communication between DSPs. MessageQ is a reliable control path for inter-processor communication. See reference to details at Control_path_IPC [1]. The demo application uses a IPC benchmark application MessageQBench. Please see details at IPC_BenchMarking [2]

Run Instructions

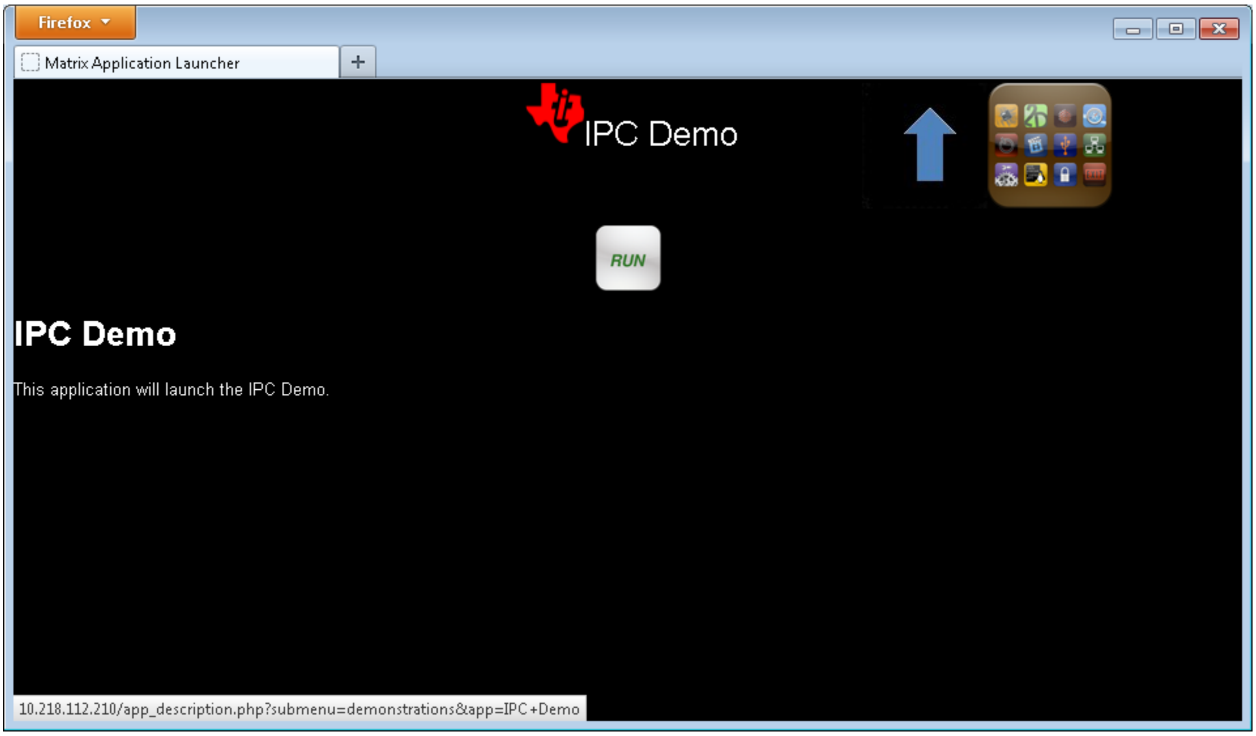
The Matrix application launcher will start automatically when the system boots up. Once the target EVM gets an IP address assigned by the DHCP server, type the IP address to an Internet browser running on a remote computer (Target EVM should be accessible through network connection by the remote computer).



Click on "Demonstrations" icon.


Firefox
Matrix Application Launcher
+
Demonstrations
Submenu p1
Image Processing Demo
IPC Demo
10.218.112.210/app_description.php?submenu=demonstrations&app=IPC+Demo

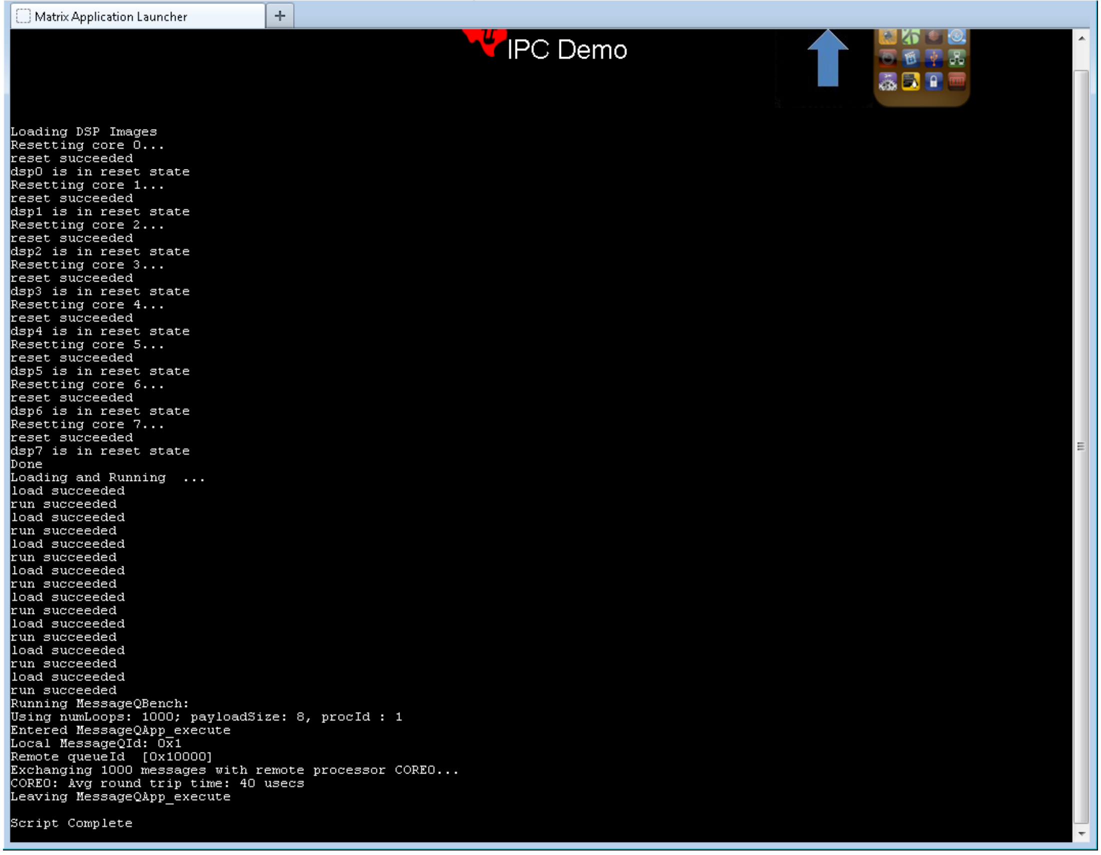
Click on "IPC Demo" icon


Firefox
Matrix Application Launcher
IPC Demo
RUN
10.218.112.210/app_description.php?submenu=demonstrations&app=IPC+Demo

IPC Demo

This application will launch the IPC Demo.

Click on "Run" Button and see the following results.



```
Loading DSP Images
Resetting core 0...
reset succeeded
dsp0 is in reset state
Resetting core 1...
reset succeeded
dsp1 is in reset state
Resetting core 2...
reset succeeded
dsp2 is in reset state
Resetting core 3...
reset succeeded
dsp3 is in reset state
Resetting core 4...
reset succeeded
dsp4 is in reset state
Resetting core 5...
reset succeeded
dsp5 is in reset state
Resetting core 6...
reset succeeded
dsp6 is in reset state
Resetting core 7...
reset succeeded
dsp7 is in reset state
Done
Loading and Running  ...
load succeeded
run succeeded
load succeeded
run succeeded
load succeeded
run succeeded
load succeeded
run succeeded
load succeeded
run succeeded
load succeeded
run succeeded
load succeeded
run succeeded
load succeeded
run succeeded
Running MessageQBench:
Using numLoops: 1000; payloadSize: 8, procId : 1
Entered MessageQApp_execute
Local MessageQId: 0x1
Remote queueId  [0x10000]
Exchanging 1000 messages with remote processor CORE0...
CORE0: Avg round trip time: 40 usecs
Leaving MessageQApp_execute

Script Complete
```

The results show the IPC round trip delay measured.

MCSDK UG Chapter Developing

MCSDK User Guide: Developing with the MCSDK

Last updated: **09/20/2015**

Topics

Migration Guide

This chapter provides information for migrating software from major MCSDK releases.

Your First Application

Write your first application by stepping through a series of small example programs designed to increase your proficiency with the tools, development environments and software that is part of the MCSDK.

Platform Development Kit and Peripherals

The Platform Development Kit (PDK) is a package that provides the foundational drivers and software to enable the device. It contains device specific software consisting of a Chip Support Library (CSL) and Low Level Drivers (LLD) for various peripherals; both the CSLs and LLDs include example projects and examples within the relevant directories.

Debug and Trace

This chapter provides information on tools for debug and trace.

Transports

Learn about the various Transports that are included in the MCSDK and how they move data between the ARM and DSP subsystems or between different software layers.

System Management

Learn how to manage resources on the device/EVM using the Resource Manager, download and manage DSP from ARM along with topics that have a global impact on your system.

Fault Management

Learn about troubleshooting, monitoring, and error recovery with the MCSDK.

Security

Learn about security features offered in the SDK.

TransportNetLib User Guide

Learn about the TransportNet Library and the software APIs it provides to utilize Network Coprocessor(NETCP) hardware accelerator functionality offered by Keystone devices. Purpose of this document is to introduce system level details and is intended for software application developers using the TransportNetLib software package

MCSDK UG Chapter Developing Migration

Developing with MCSDK: Migration Guide

Last updated: **09/20/2015**

Overview

This chapter provides information for migrating software from major MCSDK releases.

MCSDK 3.1.3 to MCSDK 3.1.4

The intent of this section is to provide details for migrating software from MCSDK 3.1.3 to MCSDK 3.1.4 for all supported devices.

RM LLD

The RM Server socket location in the Linux file system has changed. In previous releases a header called sockrmmsg.h was copied into any test or example project that wished to connect to the Linux RM Server. This header placed the RM Server socket in /tmp/var/run/rm/rm_server. A new RM API has been created in order to unify the RM Server socket location. Any application wishing to connect to the Linux RM Server should

```
#include <pdk_keystone2_3_01_04_07\packages\ti\drv\rm\rm_server_if.h>
```

This file places the RM Server socket in /var/run/rm/rm_server

MCSDK 3.1.1 to MCSDK 3.1.3

The intent of this section is to provide details for migrating software from MCSDK 3.1.1 to MCSDK 3.1.3 for all supported devices.

U-Boot

I2C driver supports multiple busses. You cannot assume the default bus is always '0'. Use 'i2c dev <bus number>' command to switch to desired bus.

ARM IPC MessageQ

With the move to IPC 3.35.01.07, any ARMv7 Linux Host application using IPC must include an additional function call prior to Ipc_start(). There was a circular dependency between the IPC and the newly introduced transport libraries. To break this circle, the transport create method was moved outside of IPC core logic. IPC invokes this through a function table configured with a new Ipc_transportConfig() method which is called by the application.

User-space applications that start IPC must be modified:

```
/* Old IPC startup */
status = Ipc_start()
if (status < 0) {
    /* Failure case */
}
```

```
/* New IPC startup */
#include <ti/ipc/transports/TransportRpmsg.h>

Ipc_transportConfig(&TransportRpmsg_Factory);
status = Ipc_start();
if (status < 0) {
    /* Failure case */
}
```

Additionally, the linking process of the user-space application must now include libtitransportrpmsg as follows (order of library inclusion matters):

```
# Old included IPC libraries in link of app
-ltiipc -ltiipcutils
```

```
# New included IPC libraries in link of app
-ltitransportrpmsg -ltiipc -ltiipcutils
```

For more information on the changes made in IPC 3.35.01.07 please see the latest release notes [1]

PA LLD

Below changes are required to be done in both DSP and user mode applications when using PA LLD 3.0.1.4, integrated in MCSDK 3.1.3 release. For full details please visit PA LLD release notes.

- PA instance memory increased to 288 bytes from 256 bytes.
- PA Number of buffer requirement increased to 8 buffers from 7 buffers

MCSDK 3.1.0 to MCSDK 3.1.1

The intent of this section is to provide details for migrating software from MCSDK 3.1.0 to MCSDK 3.1.1 for all supported devices.

U-Boot

There are two u-boot changes that may be interest, for full details, see the Linux/u-boot portion of the user guide.

- The PLL clock frequency is now programmed based on EFUSE register settings. The getclk command in u-boot can be used to know the actual frequency used. Note that the clock may be rounded to the nearest possible value based on settings for the PLL clock.
- For K2L, The NAND part used in the EVM is 2GB in size. Previously, the u-boot environment variable was set to use only 512MB of the NAND capacity. In this release, the u-boot environment variable is now set up for 2GB. This results in increased bootup time caused by UBI mount of larger partition size.

Linux

Prior to MCSDK 3.1.1, MDIO was disabled for K2L and K2E EVM. In this release, MDIO is enabled for K2L and K2E EVMs. Note that boot up time increases slightly by about 2-3 seconds per port due to auto negotiation. On K2H/K, the MDIO continues to be disabled to keep backward compatibility with older revision of the hardware.

This maintenance update does not have any API changes.

MCSDK 3.0.4 to MCSDK 3.1.0

The intent of this section is to provide details for migrating software from MCSDK 3.0.x to MCSDK 3.1.0 for **K2K and K2H** devices.

Installer

The installers for Windows, Linux host, and Linux target have changed in this release.

Windows

The windows installer with the MCSDK 3.1.0 release will not have the mcsdk_linux_<version> package. The full contents of the installer, including the mcsdk_linux_<version> directory, can be obtained from the mcsdk_<version>.tar.gz on the release page.

Linux host

The Linux host installer mcsdk_<version>_setuplinux32.bin is replaced by a new command line installer named mcsdk_<version>_setuplinux.bin (non gui version). This installer can be used to install the MCSDK package on the target as well as on any linux host. This installer includes the XDC package usable on an ARM target.

Linux target

The target mcsdk installer previously called mcsdk_<version>_native_setuplinux.bin is now renamed to mcsdk_<version>_setuplinux.bin.

Boot Monitor

In MCSDK 3.1, boot monitor supports K2HK, K2L and K2E EVMs. When user builds from a git cloned source tree, the *make* command builds all three images: skern-k2hk.bin, skern-k2l.bin and skern-k2e.bin. The pre-built images in the SDK are renamed to skern-k2hk-evm.bin, skern-k2l-evm.bin and skern-k2e-evm.bin respectively.

Uboot

Target Images

Two new U-boot target blob images (u-boot-k2l-evm.bin for K2L EVM and u-boot-k2e-evm.bin for K2E EVM) are supported in MCSDK 3.1, the old target blob image name is changed from u-boot-keystone-evm.bin (K2HK EVM) to u-boot-k2hk-evm.bin.

Two new U-boot target SPI GPH images (u-boot-spi-k2l-evm.gph for K2L EVM and u-boot-spi-k2e-evm.gph for K2E EVM) are supported in MCSDK 3.1, the old SPI GPH image name is changed from u-boot-spi-keystone-evm.gph (K2HK EVM) to u-boot-spi-k2hk-evm.gph.

New NAND boot GPH images are supported in MCSDK 3.1, the names are u-boot-nand-k2hk-evm.gph (K2HK EVM), u-boot-nand-k2l-evm.gph (K2L EVM) and u-boot-nand-k2e-evm.gph (K2E EVM).

Target Configurations

The command to make K2HK EVM configuration has changed:

```
> make tci6638_evm_config ==> make k2hk_evm_config
```

The command to make K2L EVM configuration is:

```
> make k2l_evm_config
```

The command to make K2E EVM configuration is:

```
> make k2e_evm_config
```

Environment Variables

The "has_mdio" environment variable is removed, depending on sgmiiN_link_type (N is a port number starting from 0), keytone2_eth_open() sets the local sys_has_mdio for the currently opening port.

A new environment variable, "uinitrd_fixup", is added do device tree fixup for older versions of kernels (3.08/3.10).

System Clock frequency

U-boot program the Core and Tetris clock rate based on the Efuse register settings. So it will change from board to board based on the Efuse settings.

MDIO frequency

MDIO clock frequency is increased from 1MHz to 2.5 MHz.

Linux kernel

Ethernet drivers

A new 1GB Ethernet driver plugin driver is added to support the new NSS (Network Sub-System) module for K2E and K2L devices. The new driver supports a Ethernet switch with up to 8 EMAC (slave) ports and 1 host port.

A new Packet Accelerator plugin driver is added to support the new NSS (Network Sub-System) for K2E and K2L devices. The new driver supports 9 clusters with up to 15 PDSP processors (8 Ingress, 2 post and 5 Egress PDSPs).

A new SGMII SERDES driver is added to support the TI Keystone II SERDES IP. The driver has generic API's that are also extendable to other Keystone II peripherals, such as PCSR SERDES, PCIe SERDES, etc.)

The Address Lookup Engine driver is updated to support both revisions (v1.3 and v1.4) ALE IP, v1.4 has up to 8 slave ports and 1 host port.

10G Ethernet driver is disabled through DT bindings by default. See the user guide on how to enable it on a board that has 10G ethernet hardware support available.

Yocto

The Yocto machine name for K2H/K2K has changed in MCSDK 3.1.0. For MCSDK 3.0.x, the machine name was "keystone-evm". For MCSDK 3.1.0 and onward, it is "k2hk-evm". Please refer to Yocto Build Instructions [2] for details.

DDR3 ECC

DDR3 error detection and correction feature is enabled on K2H/K2K PG 2.0 devices and K2L/K2E devices. Please refer to [3] for details.

Device Tree

The device tree files are re-organized in MCSDK 3.1.0, the common device tree bindings for all the Keystone II devices are extracted to the following files:

- keystone.dtsi (common SoC related device tree)
- keystone-clocks.dtsi (common clocks related device tree)
- keystone-qostree.dtsi (common network QoS related device tree)

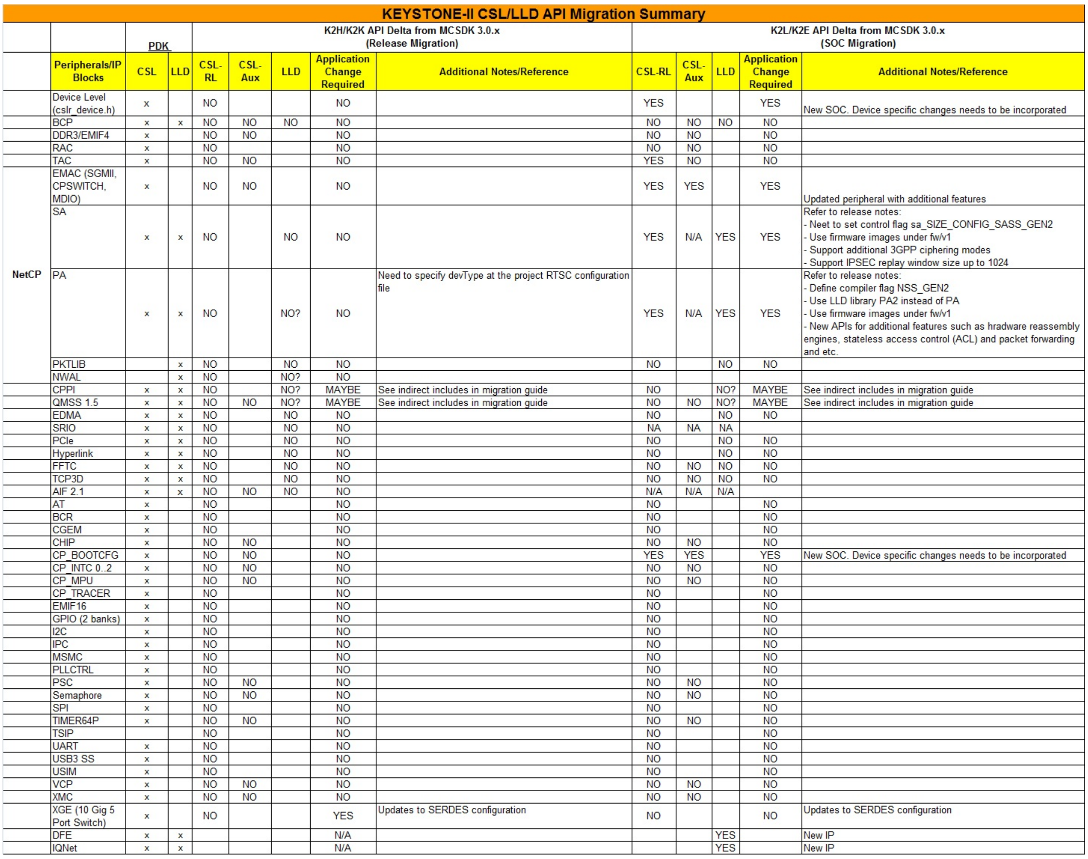
The SoC specific device tree bindings are extracted to the following files:

- k2hk.dtsi (K2HK SoC specific device tree)
- k2hk-clocks.dtsi (K2HK specific clocks device tree)
- k2hk-net.dtsi (K2HK specific network device tree)
- k2l.dtsi (K2L SoC specific device tree)
- k2l-clocks.dtsi (K2L specific clocks device tree)
- k2l-net.dtsi (K2L specific network device tree)
- k2e.dtsi (K2E SoC specific device tree)
- k2e-clocks.K2E (K2E specific clocks device tree)
- k2e-net.dtsi (K2E specific network device tree)

The EVM specific device tree bindings are extracted to the following files:

- k2hk-evm.dts (K2HK EVM specific device tree)
- k2l-evm.dts (K2L EVM specific device tree)
- k2e-evm.dts (K2E EVM specific device tree)

PDK



KEYSTONE-II CSL/LLD API Migration Summary													
		PDK		K2H/K2K API Delta from MCSDK 3.0.x (Release Migration)					K2L/K2E API Delta from MCSDK 3.0.x (SOC Migration)				
	Peripherals/IP Blocks	CSL	LLD	CSL-RL	CSL-Aux	LLD	Application Change Required	Additional Notes/Reference	CSL-RL	CSL-Aux	LLD	Application Change Required	Additional Notes/Reference
	Device Level (cslr_device.h)	x		NO			NO		YES			YES	New SOC. Device specific changes needs to be incorporated
	BCP	x	x	NO	NO	NO	NO		NO	NO	NO	NO	
	DDR3/EMIF4	x		NO	NO		NO		NO	NO		NO	
	RAC	x		NO			NO		NO	NO		NO	
	TAC	x		NO	NO		NO		YES	NO		NO	
NetCP	EMAC (SGMII, CPSWITCH, MDIO)	x		NO	NO		NO		YES	YES		YES	Updated peripheral with additional features
	SA	x	x	NO		NO	NO		YES	N/A	YES	YES	Refer to release notes: - Neet to set control flag sa_SIZE_CONFIG_SASS_GEN2 - Use firmware images under fw/v1 - Support additional 3GPP ciphering modes - Support IPSEC replay window size up to 1024
	PA	x	x	NO		NO?	NO	Need to specify devType at the project RTSC configuration file	YES	N/A	YES	YES	Refer to release notes: - Define compiler flag NSS_GEN2 - Use LLD library PA2 instead of PA - Use firmware images under fw/v1 - New APIs for additional features such as hradware reassembly engines, stateless access control (ACL) and packet forwarding and etc.
	PKTLIB		x	NO		NO	NO		NO		NO	NO	
	NWAL		x	NO		NO?	NO						
	CPPI	x	x	NO		NO?	MAYBE	See indirect includes in migration guide	NO		NO?	MAYBE	See indirect includes in migration guide
	QMSS 1.5	x	x	NO	NO	NO?	MAYBE	See indirect includes in migration guide	NO	NO	NO?	MAYBE	See indirect includes in migration guide
	EDMA	x	x	NO		NO	NO		NO		NO	NO	
	SRIO	x	x	NO		NO	NO		NA	NA	NA		
	PCIe	x	x	NO		NO	NO		NO		NO	NO	
	Hyperlink	x	x	NO		NO	NO		NO		NO	NO	
	FFTC	x	x	NO		NO	NO		NO	NO	NO	NO	
	TCP3D	x	x	NO		NO	NO		NO	NO	NO	NO	
	AIF 2.1	x	x	NO	NO	NO	NO		N/A	N/A	N/A		
	AT	x		NO			NO		NO			NO	
	BCR	x		NO			NO		NO			NO	
	CGEM	x		NO			NO		NO			NO	
	CHIP	x		NO	NO		NO		NO	NO		NO	
	CP_BOOTCFG	x		NO	NO		NO		YES	YES		YES	New SOC. Device specific changes needs to be incorporated
	CP_INTC 0..2	x		NO	NO		NO		NO	NO		NO	
	CP_MPU	x		NO	NO		NO		NO	NO		NO	
	CP_TRACER	x		NO			NO		NO			NO	
	EMIF16	x		NO			NO		NO			NO	
	GPIO (2 banks)	x		NO			NO		NO			NO	
	I2C	x		NO			NO		NO			NO	
	IPC	x		NO			NO		NO			NO	
	MSMC	x		NO			NO		NO			NO	
	PLLCTRL	x		NO			NO		NO			NO	
	PSC	x		NO	NO		NO		NO	NO		NO	
	Semaphore	x		NO	NO		NO		NO	NO		NO	
	SPI	x		NO			NO		NO			NO	
	TIMER64P	x		NO	NO		NO		NO	NO		NO	
	TSIP			NO			NO		NO			NO	
	UART	x		NO			NO		NO			NO	
	USB3 SS	x		NO			NO		NO			NO	
	USIM	x		NO			NO		NO			NO	
	VCP	x		NO	NO		NO		NO	NO		NO	
	XMC	x		NO	NO		NO		NO	NO		NO	
	XGE (10 Gig 5 Port Switch)	x		NO			YES	Updates to SERDES configuration	NO			NO	Updates to SERDES configuration
	DFE	x	x				N/A				YES		New IP
	IQNet	x	x				N/A				YES		New IP

SA LLD

SA LLD is available as a PDK component starting in MCSDK 3.1.0 release. This is a change compared to standalone SA LLD release from ti.com download page. This component is also available at https://git.ti.com/keystone-rtos/sa-lld.

QMSS LLD

Device-specific includes (cslr_device.h, csl_qm_queue.h) were removed from QMLLD's public header files. Application will need to add these includes together with appropriate device (DEVICE_K2H/DEVICE_K2L/DEVICE_K2E/DEVICE_K2K) defines if it uses symbols from these files.

CPPI LLD

Device-specific includes (cslr_device.h, csl_cppi.h) were removed from CPPILLD's public header files. Application will need to add these includes together with appropriate device (DEVICE_K2H/DEVICE_K2L/DEVICE_K2E/DEVICE_K2K) defines if it uses symbols from these files.

NWAL

Device-specific libraries are being provided as part of MCSDK 3.1. The naming conventions of the libraries for Linux use case are the following: libnwal_$DEVICE.a (for shared libraries, libnwal_$DEVICE.so) where $DEVICE can be k2h/k2k/k2e/k2l.

For Linux use case, applications using the nwal library will need to update their makefiles to link in the desired device specific library based on the device the application is being compiled for.

For DSP use case, the naming convention of the provide libraries remains the same but libraries are located in device specific directories under ti/drv/nwal/lib/$DEVICE where $DEVICE can be k2h/k2k/k2e/k2l.

MCSDK UG Chapter Developing KS II First App

Developing with MCSDK: Your First Application

Last updated: **09/20/2015**

Overview

The following sequence is recommended to get familiar with Keystone II EVM and MCSDK software.

Out-of-box Demo

Please refer to Getting Started Guide [1] to execute every step from “Hardware Setup” till “Program EVM”. This is to verify that all versions of hardware modules and software are up-to-date, and update them if not. Once these are done, users can proceed to run out-of-box demo.

Get the first hello world to run on A15

Refer to Tools Chapter of Keystone-II User's Guide [2] to download and setup Linario tool chain. To compile the hello.c file and run it on ARM, please follow the steps below:

a) Building the executable

```
arm-linux-gnueabi-gcc -o hello hello.c
```

b) Transfer the executable to the EVM First, copy the executable to the tftpboot directory of the TFTP server. Then from the linux prompt of the EVM, type:

tftp -r hello -g <tftp server IP address>

Or if NFS is used, please the hello binary under NFS server directory.

[Quick note on how to set up NFS filesystem [3]. For more detailed info, please google for Linux NFS]

c) Run the application on EVM after it boots up

```
chmod +x hello
./hello
```

Installation of MCSDK on target

Starting from MCSDK 3.0.4, the release includes an MCSDK native linux package that can be installed on the target platform.
The MCSDK release package can be installed in the filesytem if the required space is available. Here are the steps to install the MCSDK on the target platform. Note the XDC tools which can be used on the target for C66x DSP compilation are installed under the directory.

```
wget <http-url>mcsdk_3_00_04_18_native_setuplinux.bin
chmod +x mcsdk_3_00_04_18_native_setuplinux.bin
./mcsdk_3_00_04_18_native_setuplinux.bin
```

Installing Linux-devkit on the target platform follow the following steps.

```
cd <target_directory for devkit installation>
sed '1,/^MARKER:/d' mcsdk_linux_<version>/linux-devkit/arago-2013.12-cortexa15-linux-gnueabi-mcsdk-sdk-i686.sh > devkit.bz2
tar xjf devkit.bz2
```

Compilation of ARM Image on A15 Target

Starting from MCSDK 3.0.4, the released root filesystem includes tools for target compilation for ARM A15 on the target platform.

To compile the hello.c file and run, please follow the steps below a) Build the executable on the target using

```
gcc -o hello hello.c
```

b) Run the application on EVM using

```
./hello
```

Compilation of DSP Image on A15 Target

Starting from MCSDK 3.0.4, the release target filesystem includes additional tools to enable building a C66x DSP image on the ARM A15 target.
Note that the code gen tools used for compilation of the C66x DSP image is not included in the MCSDK release. Contact TI technical support for early adopter versions of this tool.

Get the first DSP application to run with MPM (Multiple Processor Manager)

In Keystone-II, Multiple Processor Manager (MPM) must be used to load and run DSP applications.

A pre-build DSP image is in this folder:

```
mcsdk_bios_x_xx_xx_xx/examples/mpm/mpmsrv_keystone2_example/Debug/mpmsrv_keystone2_example.out
```

[Note: The MCSDK package should have been installed in step 1 by now. If it has not been installed, please visit the section of installing the MCSDK [4] in Getting Started Guide to install it.

Follow the following steps to run the demo.

a) Copy the pre-build DSP image to tftpboot directory of the TFTP server b) Tftp the DSP image to EVM using the tftp command shown in previous holloworld example, and rename the image file to mpmdemo.out c) In Linux prompt of the EVM, type the following

```
      mpmcl ping dsp0                                (shows the DSP0 is alive)
      mpmcl status dsp0                             (shows the DSP0 status)
      mpmcl load dsp0 mpmdemo.out   (load the prebuild image to DSP0)
      mpmcl run dsp0                                  (run the executable on DSP0)
```

The output of mpmdemo.out will be in /debug/remoteproc/remoteproc0/trace0 of the EVM file system.

```
Replace dsp0 to dsp# in above commands to run demo in different dsp cores (where # = 0, 1, ..., 7)
```

The result for each core will be in /debug/remoteproc/remoteproc#/trace0

For more details about MPM, please refer to System Management Section in Exploring Chapter [5].

User space example on ARM

For user space example on ARM, please refer to the example code in mcsdk_linux_x_xx_xx_xx\example-applications\smp_test directory.

ARM and DSP Communication

MCSDK provides high level abstraction component to provide communication between different processors. The MessageQ based IPC is used between ARM to DSP and beween DSP to DSP. For more info, please refer to Transport Section in Exploring Chapter of the Keystone-II User's Guide.

The image processing demo has ARM side master code and DSP slave code. To see how the IPC communication between ARM and DSP is coded, please refer to the source code in mcsdk_bios_x_xx_xx_xx/demos/image_processing/ipc/master/src and slave/src directories

Explore for your application needs

Follow this wiki page [6] to build uboot, kernel and file system and build your applications.

MCSDK UG Chapter Developing PDK

Developing with MCSDK: Platform Development Kit and Peripherals

Last updated: **09/20/2015**

Acronyms

The following acronyms are used throughout this chapter.

Acronym	Meaning
CSL	Texas Instruments Chip Support Library
DSP	Digital Signal Processor
EDMA	Enhanced Direct Memory Access
MCSDK	Texas Instruments Multi-Core Software Development Kit
PDK	Texas Instruments Programmers Development Kit
SRIO	Serial Rapid IO
TI	Texas Instruments

CSL/LLD User Guide

Overview

This document provides guideline for compiling and executing Low Level Device Driver (LLD) with examples included in the release package. As a pre-requisite refer release note for PDK in order to find recommended version of compiler and dependent tools. LLD would need to be installed by running the installer part of PDK. For rest of the document keyword

- <*PDK_INSTALL_PATH*> refers to the location where PDK is installed.
- <*lld*> refers to individual LLDs in PDK package

Build guidelines for Full PDK package for C66x Target

1. Release package comes with a pre-built set of libraries for individual LLD’s and CSL modules. Below are the steps in case if an LLD would need to be rebuilt. Note that makefile support is now added so that LLDs and CSL libraries can be built using makefiles.

Building PDK using gmake in Windows environment

1. Setting Environment Variables by editing the <PDK_INSTALL_PATH>\packages>pdksetupenv.bat to reflect your installation (E.g.):

```
set C6X_GEN_INSTALL_PATH="C:/ti_5_5/ccsv5/tools/compiler/c6000_7.4.7"
set EDMA3LLD_BIOS6_INSTALLDIR="C:/ti/edma3_lld_02_11_13_17"
set PDK_INSTALL_PATH=C:/ti/pdk_keystone2_3_01_00_03/packages
set BIOS_INSTALL_PATH=C:/ti/bios_6_37_03_30
set XDC_INSTALL_PATH=C:/ti/xdctools_3_25_06_96
set CG_XML_BIN_INSTALL_PATH=C:/ti/cg_xml/bin
```

2. Run *pdksetupenv.bat* to setup the build enviornment and to check in case if any modifications are required for the target environment.

```
pdksetupenv.bat
```

Note: The pdksetupenv.bat requires *path2dos.exe* utility available from XDC tools. If *path2dos.exe* is not available in the build environment then short names for paths **SHOULD** be provided to environment variables e.g. *C:\PROGRA~1\TEXASI~1\PDK_TC~4\packages*. In Windows environment, "dir /x" and "dir /-n" displays the short names generated for non-8dot3 file and directory names.

3. To build all the PDK components, execute pdkbuilder.bat from the top level PDK packages directory <*PDK_INSTALL_PATH*>*packages* directory

```
pdkbuilder.bat
```

4. The batch file cleans and rebuilds all components which are part of the PDK release.

Building PDK using gmake in MSYS environment

1. Setting Environment Variables
   - C6X_GEN_INSTALL_PATH
     - Install location for Code Gen Tool
     - Examples for default location of Code Gen Tool 7.2.2 GA

```
set C6X_GEN_INSTALL_PATH="C:/Program Files/Texas Instruments/C6000 Code Generation Tools 7.2.2"
```

   - PDK_INSTALL_PATH
     - Install location for PDK package
     - Change the shell prompt directory to PDK installation directory. Example for a default PDK installation at *C:/Program Files/Texas Instruments/pdk_TCI6608_1_0_0_10/packages*, the commands would be

```
$cd "C:/Program Files/Texas Instruments/pdk_TCI6608_1_0_0_10/packages"
$export PDK_INSTALL_PATH=$PWD
```

   - Additional environment variables are optional and not required on this environment. Refer pdksetupenv.bat or pdksetupenv.sh for the details.
2. To build all the PDK components, run pdkbuilder.sh from the top level PDK packages directory *<PDK_INSTALL_PATH>/packages* directory.

```
$./pdkbuilder.sh
```

   The shell script cleans and rebuilds all components which are part of the PDK release.

Build guidelines for ARM User Mode LLDs

1. ARM User Mode support is available for following PDK components:
   - CPPI
   - RM
   - QMSS
   - PA
   - SA
   - NWAL
   - PKLIB

4. **Note:**
5. In Linux user mode environment all components are supports multi thread in one process environment. CPPI/QMSS LLDs supports multi process use case with resources being allocated through RM LLD.

Building user mode LLD examples and libraries with Linux devkit

This step is applicable to LLDs supported for ARM target. Check the PDK release notes for the list of supported LLDs.

- Pre-requisites:

- Linaro ARM tool chain version as mentioned in MCSDK release notes is available in Linux host.
- Linux Devkit installed from MCSDK releases into a directory with write permission. Please note that the document would refer linux devkit location as *<LINUX_DEVKIT_INSTALL_PATH>*

- Change directory to *<PDK_INSTALL_PATH>/packages*
- Modify *armv7setupenv.sh* script based on host build environment. See below environment variables

```
# Update the below environment variable to the Linux Devkit Path
export LINUX_DEVKIT_INSTALL_PATH=$HOME/linux-devkit/sysroots/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi
# Export CORE type
export CORE=armv7
# Export Device
export DEVICE=k2h
# ARM cross tool executable path to MCSDK published tool chain (Below is an example).
export CROSS_TOOL_INSTALL_PATH=/opt/linaro/gcc-linaro-arm-linux-gnueabihf-4.7-2013.03-20130313_linux/bin
echo "CROSS_TOOL_INSTALL_PATH set to $CROSS_TOOL_INSTALL_PATH"
# ARM cross tool prefix
export CROSS_TOOL_PRFX=arm-none-linux-gnueabihf-

# Default Output directory for generated binaries and objects
export ARMV7LIBDIR=$PWD/../lib/armv7
#Modify to Linux Dev Kit Lib Path as below
if [ ! -z "$LINUX_DEVKIT_INSTALL_PATH" ]; then
#Modify the location here if the intent is not to get the libraries under linux devkit, otherwise no modications needed
export ARMV7LIBDIR=$LINUX_DEVKIT_INSTALL_PATH/usr/lib
fi
```

- Execute the script in target shell environment by executing
  source armv7setupenv.sh
- Run **make all** to build all executables and libraries
- In the case of building subset of targets take a look at the makefile being provided to go through targets supported. Following are the options

Building only libraries

make lib
Output libraries will be available at $ARMV7LIBDIR location

Building only tests

make tests
Output test binaries will be available at $ARMV7BINDIR location

Building only examples

make examples
Output example binaries will be available at $ARMV7BINDIR location and *Temporary object files will be available at $ARMV7OBJDIR location*

Note: Please note that the user would need write permissions to the locations as specified by *$ARMV7LIBDIR*.

Building user mode LLD examples and libraries with *NO* Linux devkit (Not a standard practice and not supported starting with MCSDK 3.0.4)

This step is applicable to LLDs supported for ARM target. Check the PDK release notes for the list of supported LLDs.

- Pre-requisite:
  - Linaro ARM tool chain version as mentioned in MCSDK release notes is available in Linux host.
  - IPC libraries are rebuilt with same ARM cross tool version as MCSDK provided one.
  - HPLIB and SA library to be built with the same Linaro ARM tool chain as MCSDK provided one

Note: Please refer to *<ipc_install>/docs/IPC_Install_Guide* for details on rebuilding the IPC libraries.

- Change directory to *<PDK_INSTALL_PATH>/packages*
- Modify *armv7setupenv.sh* script based on host build environment. See below environment variables

```
# Make sure below environment is unset for building without LINUX DEVKIT
export LINUX_DEVKIT_INSTALL_PATH=
# Provide the HPLIB INSTALL PATH
export HPLIB_INSTALL_PATH=$HOME/ti/transport_net_lib_1_0_0_6/packages
# Provide the SA LLD install PATH
export SA_INSTALL_PATH=$HOME/ti/mcsdk_3_00_00_11/salld_keystone2_02_00_00_06/packages

#SA LLD Library path
# Modify to match the actual armv7 sa lld library build path
export ARMV7SALIBDIR=$SA_INSTALL_PATH/ti/drv/sa/lib/armv7

# HPLIB library path
# Modify to match the actual armv7 sa lld library build path
export ARMV7HPLIBDIR=$HPLIB_INSTALL_PATH/ti/runtime/hplib/lib/armv7

# Update the IPC library paths - please note that this is the path after IPC libraries and include files are installed in one common folder
export IPC_DEVKIT_INSTALL_PATH=$HOME/ipc_install_path


# Default Output directory for generated binaries and objects, modify for any other locations
export ARMV7LIBDIR=$PWD/../lib/armv7
#Modify to Linux Dev Kit Lib Path
if [ ! -z "$LINUX_DEVKIT_INSTALL_PATH" ]; then
export ARMV7LIBDIR=$LINUX_DEVKIT_INSTALL_PATH/usr/lib
fi
```

- Execute the script in target shell environment by executing
  source armv7setupenv.sh
- Run **make all** to build all executables and libraries
- In the case of building subset of targets take a look at the makefile being provided to go through targets supported. Following are the options

Building only libraries

make lib
Output libraries will be available at $ARMV7LIBDIR location

Building only tests

make tests
Output test binaries will be available at $ARMV7BINDIR location

Building only examples

make examples
Output example binaries will be available at $ARMV7BINDIR location and *Temporary object files will be available at $ARMV7OBJDIR location*

Note: Please note that the user would need write permissions to the locations as specified by *$ARMV7LIBDIR*.

Steps to run example and/or unit test projects on C66x Target

1. The
2. *example*
3. directory in individual LLD contains several examples which demonstrate usage of API's. The "test" directory in some LLD contains unit test module.
4. **Check Prerequisites**
   Please ensure that all dependent/pre-requisite packages are installed before proceeding with the examples and/or unit test.
5. **Configure CCS Environment**
   The CCS environment configuration step needs to be done only once for a workspace as these settings are saved in the workspace preferences. These settings only need to be modified if:
   - New workspace is selected
   - Newer version of the component is being used. In that case modify the paths of the upgraded component to the newer directory.

   The procedure mentioned in this section is provided using <Managed Build Macro> option in CCS. Following are the steps:

   1. Create a macro file if not available from the PDK release. For the PDK release file: *<PDK_INSTALL_PATH>\packages\macros_ini* can be used

   Following environment would need to be available in the *macros.ini* file

```
PDK_INSTALL_PATH = <PDK_INSTALL_PATH>\packages
CSL_INSTALL_PATH = <PDK_INSTALL_PATH>\packages
CPPI_INSTALL_PATH = <PDK_INSTALL_PATH>\packages
QMSS_INSTALL_PATH = <PDK_INSTALL_PATH>\packages
SRIO_INSTALL_PATH = <PDK_INSTALL_PATH>\packages
TCP3D_INSTALL_PATH = <PDK_INSTALL_PATH>\packages
```

   1. Open CCS and select an appropriate workspace
   2. Load Managed Build Macros by

   i. Click on File -> Import
   ii. Click on Managed Build Macros
   iii. Select the file macros.ini
   iv. Click on "Overwrite existing values"
   v. Click Finish
6. **Create CCS Project**
   PDK package includes a batch file *pdkProjectCreate.bat* under *<PDK_INSTALL_PATH>\packages*. The batch file allows creation of projects based on the different dependent tool version for all examples and unit tests included in PDK. The batch file also allows additional executable types eg: *COFF/ELF* and Endianess. Batch file is supported for both CCSv4.2 and CCSv5.x environment. Alternatively, projects can be created using the CCS

wizard and importing required files from test and example directories. Additional details on using *pdkProjectCreate.bat*:

- Prerequisite: All dependent components need to be installed. After the components are installed, start CCS once and wait for eclipse plugin to get into effect before running the batch file.
- Modify environment variables in pdkProjectCreate.bat under "<PDK_INSTALL_PATH>\packages" directory to reflect project options. This would also include
  - IS_SIMULATOR_SUPPORT_NEEDED: For running projects in simulator environment
  - ENDIAN: To select "little" or "big" endian
  - Refer additional environment settings in the batch file

Run the batch file: pdkProjectCreate.bat

- The command line above will create projects for all PDK examples. In order to create projects for a single example, for instance the PA LLD example, the command line should be modified as follows:

pdkProjectCreate.bat *<PDK_INSTALL_PATH>\packages\ti\drv\pa\example\simpleExample*. The CCS projects will be located under the directory selected for environment variable *<MY_WORKSPACE>* which by default points to *C:\MyPDKWorkspace<PART_NUMBER>*

7. **Import Project**

Below are the steps for importing project assumes that CCS project is already available.

a. Select C/C++ Development perspective
b. Click on File -> Import
c. On the Import Dialog Box select CCS Existing CCS/CCE Eclipse Project
d. Click on Next
e. This will pop up a new dialog box; ensure that 'Select Root Directory' option is selected
f. Click on Browse and select the top level directory where the project is present. For example

```
    C:\MyPDKWorkspace<PART_NUMBER>
```

<PART_NUMBER> reflects the device part number for the PDK being installed.

g. Under the projects section you should see the project. For example

```
   qmInfraExampleProject
```

h. Click Finish

8. **Build Project**

To build the project; ensure that the project you want to build, i.e., **qmInfraExampleProject** is set as the active project. Click on Project -> Build Active Project.

9. **Run Project**

- Launch the Debugger and switch to the Debug Perspective.
- To execute the project ensure the following is done:
  - Click on Target -> Reset CPU
  - Click on Target -> Load Program
  - Select the executable file to be loaded. Example: *C:\MyPDKWorkspace<PART_NUMBER>\qmInfraExampleProject\Debug\ qmInfraExampleProject.out*
  - Click on OK.
  - Once the project is loaded; click on Target -> Run to execute it.

Steps to run ARM user mode LLD example and/or unit test on Target

- Execution of user mode examples and tests included in MCSDK by default requires root previlige. This is primarily dependency of devmem for mapping the device memory which by default is restricted to super user. The default mechanism is also to reduce exposure of device memory to any user not having admin privilege
- Pre-requite for this step is to have built binaries as mentioned in previous sections.
- Transfer the example/test binaries available under *$ARMV7BINDIR* directory to the file system using TFTP or NFS
- Change the permission execute by running command chmod
- Execute the binary

MCSDK UG Chapter Developing Debug Trace

Developing with MCSDK: Debug and Trace

Last updated: **09/20/2015**

Overview

This chapter provides information on tools for debug and trace.

Acronyms

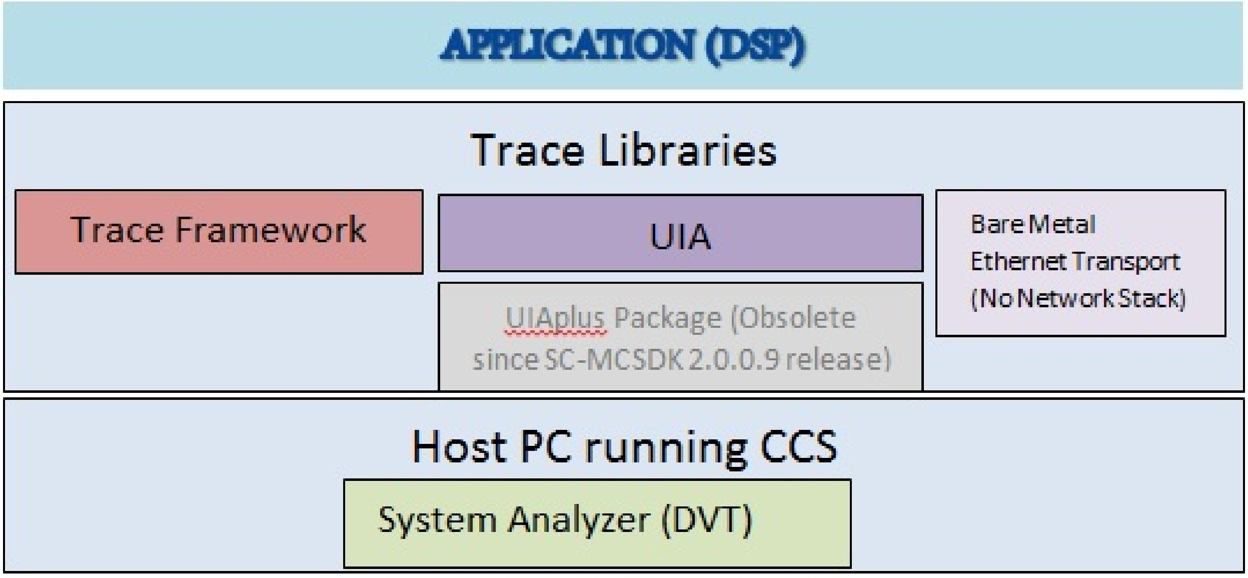
The following acronyms are used throughout this chapter.

Acronym	Meaning
CCS	Texas Instruments Code Composer Studio
DSP	Digital Signal Processor
DVT	Texas Instruments Data Analysis and Visualization Technology
IP	Internet Protocol
JTAG	Joint Test Action Group
MCSA	Texas Instruments Multi-Core System Analyzer
MCSDK	Texas Instruments Multi-Core Software Development Kit
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter
UIA	Texas Instruments Unified Instrumentation Architecture

Trace Framework

The trace framework in PDK supports the ring producer and consumer libraries.
Please refer to pdk_keystone2_1_00_00_##\packages\ti\instrumentation\traceframework\docs\html\index.html
for traceframework library API information
Please refer to \pdk_keystone2_1_00_00_##\packages\ti\instrumentation\traceframework\docs\trace_frmwrk_ug.pdf
for the traceframework concept and example applications.

The system level representation for the trace framework library is as below.


APPLICATION (DSP)
Trace Libraries
Trace Framework
UIA
UIAplus Package (Obsolete since SC-MCSDK 2.0.0.9 release)
Bare Metal Ethernet Transport (No Network Stack)
Host PC running CCS
System Analyzer (DVT)

Note:
System Analyzer's Tutorial 5B is now available at System_Analyzer_Tutorial_5B [1] link - it covers using LoggerStreamer and the PDK Trace Framework ‘producer/consumer’ model.
Trace Framework on DSP depends on UIA, UIA is a DSP component that is included in the MCSDK installer. The documentation for the UIA packages are in the docs folder, that contains the system analyzer's user guide. UIA’s API documentation can be located from CCS’s help window (Help->Help Contents).

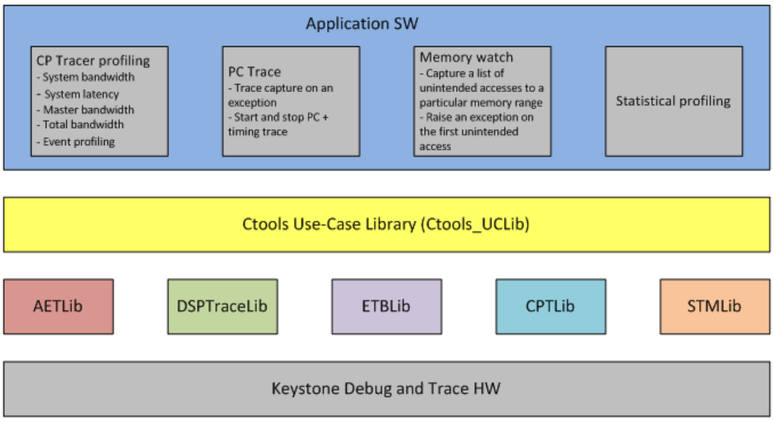
Trace Framework Library Summary	
Component Type	Library
Install Package	PDK for keystone2
Install Directory	pdk_keystone_<version>\packages\ti\instrumentation\traceframework
Project Type	CCS [18]
Endian Support	Little & Big
Library Name	Select for the TCI6638 EVM ti.instrumentation.traceframework.ae66 (little) ti.instrumentation.traceframework.ae66e (big)
Linker Path	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\lib - for release version
Linker Sections	.tf_genLogBuffersNull
Section Preference	none
Include Paths	$(TI_PDK_INSTALL_DIR)\packages\ti\instrumentation\traceframework\traceframework.h
Reference Guides	See docs under Install Directory
Support	Technical Support

Additional Resources	Texas Instruments Embedded Processors Wiki [59]
Downloads	Product Updates
License	BSD [45]

cToolsLibrary

The cTools library provides APIs for using the individual instrumentation libraries for advanced users and also few use case based libraries that are built on top of the individual instrumentation libraries.

As shown in the figure below, the *ctoolslib_sdk* module provided in the package is intended to provide the glue between the use case based libraries, individual instrumentation libraries and application:


Application SW
CP Tracer profiling
- System bandwidth
- System latency
- Master bandwidth
- Total bandwidth
- Event profiling
PC Trace
- Trace capture on an exception
- Start and stop PC + timing trace
Memory watch
- Capture a list of unintended accesses to a particular memory range
- Raise an exception on the first unintended access
Statistical profiling
Ctools Use-Case Library (Ctools_UCLib)
AETLib
DSPTraceLib
ETBLib
CPTLib
STMLib
Keystone Debug and Trace HW

cToolsLibrary SW architecture

The *ctoolslib_sdk* is a collection of libraries for Advanced Event Triggering(AET), Common Platform Tracers(CPT), Ctools use case library(Ctools_UC), DSPTrace, Embedded Trace Buffer(ETB) and System Trace Module(STM) - located under \aet\lib, \CPTLib\lib, \Ctools_UCLib\lib, \DSPTraceLib\lib, \ETBLib\lib and \STMLib\lib

Note: To run examples, import the included CCS project. If KeyStone II folder does not exist, set "C6678Debug" as the active build configuration for KeyStone II.

Ctools Library Package Summary	
Component Type	Library
Install Package	CtoolsLibrary for keystone2
Install Directory	ctoolslib_<version>
Project Type	CCS [18]
Endian Support	Little & Big
Library Name	Select for the KeyStone2 EVM Please see *GettingStarted.htm* for details
Linker Path	$(TI_CTOOLSLIB_INSTALL_DIR)\packages\ti\LIBRARY_NAME\lib - for release/debug version, where LIBRARY_NAME = aet, CPTLib, Ctools_UCLib, DSPTraceLib, ETBLib and STMLib.
Linker Sections	none
Section Preference	none
Include Paths	$(TI_CTOOLSLIB_INSTALL_DIR)\packages\ti\LIBRARY_NAME\include\*.h and ti\ctoolslib_sdk\evmk2h\package.xdc define the interface for keystone2 EVM use case library support

Reference Guides	See doc*html*\index.html file under respective libraries for details and CtoolsLib [2]
Support	Technical Support
Additional Resources	Texas Instruments Embedded Processors Wiki [59]
Downloads	Product Updates
License	BSD [45]

ctools Image processing Demo example

MCSDK demonstrates Common Platform Tracer (CPT) capabilities using image processing demo as an application. Please refer to Image processing demo guide [3] for details on the demo and steps to be followed to run it on the KeyStone2 EVM. The instrumentation examples are provided on the IPC version of the demo. Please refer to Run instructions and Expected output [4] for details on steps to be followed and expected output for use case based MCSDK instrumentation examples. The following are the supported use cases:

1. System Bandwidth Profile
    1. Captures bandwidth of data paths from all system masters to the slave (or) slaves associated with one or more CP Tracers
    2. Demo: For the Image demo, instrument the external memory (MSMC & DDR3) bandwidth used by all system masters
2. System Latency Profile
    1. Captures Latency of data paths from all system masters to the slave (or) slaves associated with one or more CP Tracers
    2. Demo: For the Image demo, instrument the external memory (MSMC & DDR3) Latency values from all system masters
3. The CP tracer messages (system trace) can be exported out for analysis in 3 ways:
    1. System ETB drain using CPU (capture only 32KB (SYS ETB size in keystone devices) of system trace data)
    2. System ETB drain using EDMA (ETB extension to capture more than 32KB (SYS ETB size in keystone devices))
    3. External Emulator like XDS560 PRO or XDS560V2
4. Total Bandwidth Profile
    1. The bandwidth of data paths from a group of masters to a slave is compared with the total bandwidth from all masters to the same slave.
        1. The following statistics are captured:
            1. Percentage of total slave activity utilized by a selected group of masters
            2. Slave Bus Bandwidth (bytes per second) utilized by the selected group of masters
            3. Average Access Size of slave transactions (for all system masters)
            4. Bus Utilization (transactions per second) (for all system masters)
            5. Bus Contention Percentage (for all system masters)
            6. Minimum Average Latency (for all system masters)
        2. Demo: For Image Demo, compare DDR3 accesses by Core0 (master core) with the DDR3 accesses by all other masters (which includes Core1, Core2…)
5. Master Bandwidth Profile
    1. The bandwidth of data paths from two different group of masters to a slave is measured and compared.
        1. The following statistics are captured:

         1. Slave Bus Bandwidth (bytes per second) utilized by master group 0
         2. Slave Bus Bandwidth (bytes per second) utilized by master group 1
         3. Average Access Size of slave transactions (for both the master groups)
         4. Bus Utilization (transactions per second) (for both the master groups)
         5. Bus Contention Percentage (for both the master groups)
         6. Minimum Average Latency (for both the master groups)
      2. Demo: For Image Demo, compare DDR3 accesses by Core0 and Core1 with the DDR3 accesses by Core2 and Core3

6. Event Profile
   1. Capture new request transactions accepted by the slave. Filtering based on Master ID or address range is supported.
      1. The following statistics are captured for every new request event: Master ID which initiated this particular transaction
         1. Bus Transaction ID
         2. Read/Write Transaction
         3. 10 bits of the address (Address export mask selects, which particular 10 bits to export)
      2. Demo: For Image Demo, implement a Global SoC level watch point on a variable in DDR3 memory. Only Core0 is intended to read/write to this DDR3 variable. Unintended accesses by other masters (or Cores) should be captured and flagged.
7. Statistical profiling provides a light weight(based on the number of Trace samples needed to be captured) and coarse profiling of the entire application.
   1. PC trace is captured at periodic sampling intervals. By analyzing these PC trace samples, a histogram of all functions called in the application, with the following information:
      1. percent of total execution time
      2. number of times encountered
   2. Demo: In the Image Demo, Statistically profile the process_rgb() function, which processes a single slice of the Image.

Ctools PC trace example

There is a hardsupport for logging the PC trace in KeyStone2 in the ETB (Extended Trace Buffer), which is 4Kb in size. The Program Counter Trace along with Timing trace examples are located under **\mcsdk_bios_#_##_##_##\examples\ctools\evmtk2h** folder. The examples include the following.

PC Trace where ETB is drained by CPU

This demonstrates the PC trace where ETB is drained by CPU. The PC Trace can be logged in either circular mode or stop mode to get 4Kb encoded PC trace information. Please follow below steps.

1. Power Cycle the EVM, make sure the EVM is brought up with Linux and you get to root prompt on the console.
2. Open CCS
3. Connect to DSP Core 0
4. Load the DSP ELF executable provided in the Example's Debug folder on DSP Core 0.
5. After the program is executed, the trace bin file gets generated in the same location which has the DSP ELF executable.
6. CCS provides a trace decoder tool, which consumes the .bin file and provides the PC trace decoded information. Please follow the steps outlined in the below section *Using_the_Trace_Decoder_Utility*.

PC Trace where ETB is drained by EDMA

This demonstrates the PC trace where ETB is drained by EDMA. This mode provides the capability to extend the 4Kb ETB buffer size using EDMA. The PC Trace can be logged in either circular mode or stop mode to get more than 4Kb encoded PC trace information. Please follow below steps to execute the program.

1. Power Cycle the EVM, make sure the EVM is brought up with Linux and you get to root prompt on the console.
2. Open CCS
3. Connect to DSP Core 0
4. Load the DSP ELF executable provided in the Example's Debug folder on DSP Core 0.
5. After the program is executed, the trace bin file gets generated in the same location which has the DSP ELF executable.
6. CCS provides a trace decoder tool, which consumes the .bin file and provides the PC trace decoded information. Please follow the steps outlined the below section *Using_the_Trace_Decoder_Utility*.

PC Trace where ETB is drained by CPU targetted for Exception debug

This demonstrates the PC trace when the application is hitting an exception. The ETB is drained by CPU. The test example code demontrates the APIs provided in the ctools Use case library. The PC Trace is always logged in circular mode. Please follow below steps to execute the program.

1. Power Cycle the EVM, make sure the EVM is brought up with Linux and you get to root prompt on the console.
2. Open CCS
3. Connect to DSP Core 0
4. Load the DSP ELF executable provided in the Example's Debug folder on DSP Core 0.
5. After the program is executed, the trace bin file gets generated in the same location which has the DSP ELF executable.
6. CCS provides a trace decoder tool, which consumes the .bin file and provides the PC trace decoded information. Please follow the steps outlined the below section *Using_the_Trace_Decoder_Utility*.

Using the Trace Decoder Utility

Trace Decoder utility is available from CCS. Please refer to TraceDecoder [5] page for details. The below sample usage should provide the PC trace dump to the console.

```
''<Dir Containing the Bin file>''\''<CCS Install
DIR>''\ccs_base\emulation\analysis\bin\td -procid 66x -bin
<binfilename>.bin -app <DSP ELF Executable Name>.out -rcvr
ETB
```

Other Ctools Library examples

Please refer to CCSv5 CtoolsLib Examples [6] for more information and to download other supported Ctools library examples. The downloaded Examples.zip should be extracted into [<CTOOLSLIB INSTALL>\packages\ti\] location. All the examples are CCSv5 compatible.

MCSDK UG Chapter Developing Transports

Developing with MCSDK: Transports

Last updated: **09/20/2015**

Overview

Learn about the various Transports that are included in the MCSDK and how they move data between the ARM and DSP subsystems or between different software layers.

MCSDK package provides multiple high level software abstractions component to facilitate applications to communicate between different processors. There are also low level drivers which talks to CPPI/QMSS/PA and can be used to communicate between separate entities. This section will focus on high level software abstractions for transport communication.

Acronyms

The following acronyms are used throughout this chapter.

Acronym	Meaning
API	Application Programming Interface
ARM	Advanced RISC Machine
CCS	Code Composer Studio
CPPI	Communications Port Programming Interface (Multicore Navigator)
DSP	Digital Signal Processor
EVM	Evaluation Module, hardware platform containing the Texas Instruments DSP
IPC	Texas Instruments Inter-Processor Communication
LLD	Low Level Driver
MCSDK	Texas Instruments Multi-Core Software Development Kit
MSMC	Multicore Shared Memory
PDK	Texas Instruments Programmers Development Kit
QMSS	Queue Manager Sub-System
RM	Resource Manager
RTSC	Eclipse Real-Time Software Components
SRIO	Serial RapidIO
TI	Texas Instruments
TID	MessageQ Network Transport ID

Transport Network Library

Please see chapter TransportNetLib User Guide [1] for details.

IPC Transports

IPC transports are the IPC MessageQ API's underlying configurable data paths over shared memory and hardware resources. Transports are registered with MessageQ providing a common IPC interface between processors within a system that contains a single or multiple KeyStone II devices. The transports supplied with the IPC component are shared memory based. Additional transports, utilizing the QMSS and SRIO LLDs, are supplied via Yocto/bitbake for ARMv7 Linux IPC and MCSDK BIOS PDK for SYS/BIOS DSP IPC.

MessageQ can support up to nine simultaneous transports over two transport interfaces. The first interface is the standard MessageQ, priority based, interface that has always existed. A shared memory transport is always registered as the normal priority transport when MessageQ is initialized at IPC start. An additional transport can be registered with MessageQ as a high priority transport. The second interface is the Network transport interface. The LLD transports can be registered with MessageQ as a Network transport. Network transports are registered with MessageQ using a Transport ID, or TID. There are seven possible TID values ranging from 1 through 7. A MessageQ message is routed over a transport registered with a certain priority, or TID, based on the settings in the message's MessageQ header. Editing a message's MessageQ header to contain the desired transport priority, or TID, is accomplished by calling the proper MessageQ header modification macro.

The below table gives an overview of the transport offerings, their location, and the communication path they enable.

Transport	MessageQ Interface Type	Location	Communication Route	Enabled Communication Path	Special Considerations
TransportShm...	MessageQ (priority based)	IPC component - SYS/BIOS subdirectories	Shared memory	SYS/BIOS DSP to DSP	There are multiple implementations of TransportShm delivered within the IPC component. Please see the IPC documentation provided with the component for more information on these shared memory transport implementations.
TransportRpmsg	MessageQ (priority based)	• IPC component - ARMv7 Linux subdirectories • IPC component - SYS/BIOS subdirectories • Yocto/bitbake ti-ipc recipe	Shared memory	• ARMv7 Linux to/from SYS/BIOS DSP	MessageQ messages sent over TransportRpmsg traveling from/to ARMv7 user space go through the Linux kernel before reaching the DSP. This provides *clean* partitioning between user memory and DSP memory. However, TransportRpmsg is considered a *slow path* since the user space MessageQ messages must be copied from/to DSP memory by kernel and DSP.

SYS/BIOS DSP TransportSrio	Network	MCSDK BIOS PDK	SRIO LLD	• SYS/BIOS DSP to/from SYS/BIOS DSP (intra- and inter-device) • SYS/BIOS DSP to/from ARMv7 Linux (intra- and inter-device)	• TransportSrio can send MessageQ messages to ARMv7 and DSP processors on remote devices in a multiple device system. IPC MultiProc must be configured to be aware of all processors existing on all devices and all devices must be connected over a SRIO interconnect. • The main purpose of TransportSrio is for multi-device communication over MessageQ. The transmission latency is greater for this transport due to the latter capability. Therefore, it is recommended a shared memory or other LLD-based transport is used for intra-device communication due to their lower latency costs.
ARMv7 Linux TransportSrio	Network	Yocto/bitbake ti-transport-srio recipe	SRIO LLD	• ARMv7 Linux to/from ARMv7 Linux (intra- and inter-device) • SYS/BIOS DSP to/from ARMv7 Linux (intra- and inter-device)	• TransportSrio can send MessageQ messages to ARMv7 and DSP processors on remote devices in a multiple device system. IPC MultiProc must be configured to be aware of all processors existing on all devices and all devices must be connected over a SRIO interconnect. • The main purpose of TransportSrio is for multi-device communication over MessageQ. The transmission latency is greater for this transport due to the latter capability. Therefore, it is recommended a shared memory or other LLD-based transport is used for intra-device communication due to their lower latency costs.
ARMv7 Linux TransportQmss	Network	Yocto/bitbake ti-transport-qmss recipe	QMSS LLD	• ARMv7 Linux process to process • ARMv7 Linux to/from SYS/BIOS DSP	
SYS/BIOS DSP TransportQmss	Network		QMSS LLD	• SYS/BIOS DSP to/from SYS/BIOS DSP • SYS/BIOS DSP to/from ARMv7 Linux	

The IPC component (ARMv7 and SYS/BIOS) is available in MCSDK BIOS and MCSDK Linux installations. It will be installed in <MCSDK BIOS/Linux install root>/ipc_3_##_##_##<version>. Additionally, the IPC component's ARMv7 source is packaged in a Yocto/bitbake recipe. A user can develop ARMv7 Linux user-space applications with IPC on KeyStone II devices by building the ti-ipc package in Yocto.

The IPC component is also used on OMAP-based Android devices. The component is compatible with legacy SYS/BIOS IPC MessageQ APIs available in other TI SDKs. A rich set of documentation can be browsed at IPC_3.x [2].

The following sections will give architecture, build, and configuration details for the LLD-based transports delivered via MCSDK BIOS PDK and Yocto/bitbake. More information on the shared memory transports delivered with the

IPC component can be found in the IPC_Users_Guide PDF found in the docs directory of the IPC component.

KeyStone II IPC Details

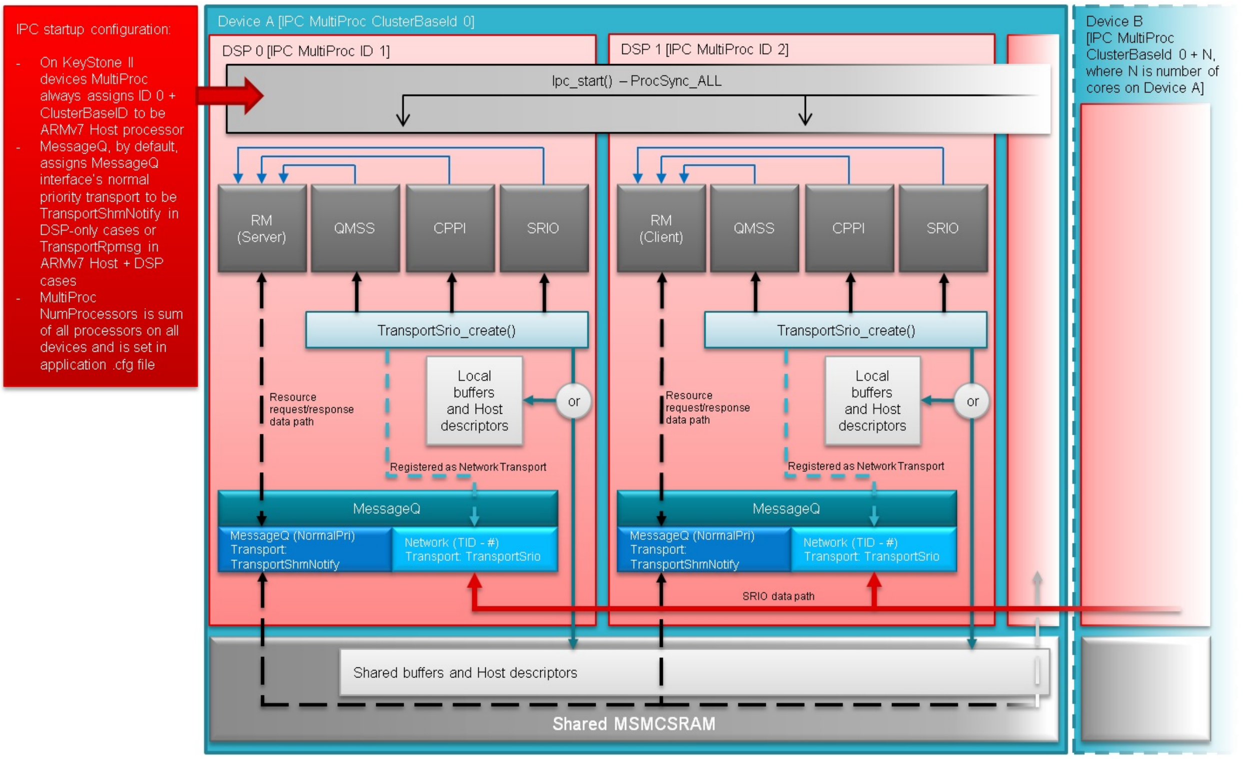
1. KeyStone II platforms must use MPM [3] to load and run DSP applications linked with IPC 3.x that wish to perform ARM to DSP, and vice versa, over MessageQ. MPM reads the DSP image on download and sets the kernel appropriate parameters for IPC.
2. CCS can be used to debug DSP applications after MPM has loaded and run the app using the load symbols facility.
3. Headers and libraries for the ARMv7 IPC component and IPC transports are provided as a part of *linux-devkit*.
4. The ARMv7 IPC's NameServer runs as a Linux daemon from the filesystem provided in the MCSDK LINUX component.

SYS/BIOS DSP TransportSrio

The SYS/BIOS DSP TransportSrio is a MessageQ Network interface transport that can be used on a KeyStone II DSP running SYS/BIOS IPC to send and receive MessageQ messages between any ARMv7 and DSP processor on the same or different device. Communication between any processor comes with the caveat that all processor's must have a unique ID assigned by the IPC MultiProc module. Additionally, the ID mappings maintained by the MultiProc modules on each device must be in sync.

Architecture

The SYS/BIOS DSP TransportSrio is a MessageQ Network interface transport that utilizes the SRIO LLD to send and receive MessageQ messages between SRIO endpoints. The SRIO endpoints can be on the same device or on another device entirely. All SRIO endpoints mapped through TransportSrio must have MessageQ as the upper level messaging layer.



IPC startup configuration:

- On KeyStone II devices MultiProc always assigns ID 0 + ClusterBaseID to be ARMv7 Host processor
- MessageQ, by default, assigns MessageQ interface's normal priority transport to be TransportShmNotify in DSP-only cases or TransportRpmsg in ARMv7 Host + DSP cases
- MultiProc NumProcessors is sum of all processors on all devices and is set in application .cfg file

Device A [IPC MultiProc ClusterBaseId 0]
DSP 0 [IPC MultiProc ID 1]
DSP 1 [IPC MultiProc ID 2]
Ipc_start() – ProcSync_ALL
RM (Server)
RM (Client)
QMSS
CPPI
SRIO
TransportSrio_create()
Local buffers and Host descriptors
or
Resource request/response data path
Registered as Network Transport
MessageQ
MessageQ (NormalPri) Transport: TransportShmNotify
Network (TID - #) Transport: TransportSrio
SRIO data path
Shared buffers and Host descriptors
Shared MSMCSRAM
Device B [IPC MultiProc ClusterBaseId 0 + N, where N is number of cores on Device A]

TransportSrio is restricted to being a MessageQ Network interface transport. Network interface transports are registered with MessageQ with a Transport ID value, or TID, which can be any integer from 1 through seven. The transport must be created and added to MessageQ's Network transport routing table after IPC has started, IPC has synced with all cores, and MessageQ has enabled a default intra-device, core to core transport. The default priority-based, intra-device, MessageQ interface transport is TransportShmNotify in DSP-only cases or TransportRpmsg in ARMv7 + DSP cases. MessageQ messages can be routed over the different transports by setting the desired transport priority, or TID value, in the MessageQ header's flags field. A message will be sent over a registered Network transport if a valid TID and a priority are set.

```
MessageQ_Msg msg;

msg = MessageQ_alloc(MY_HEAP_ID, sizeof(msg));

/* Route over MessageQ's MessageQ interface normal priority transport.
 * Should never need to explicitly set since MessageQ_alloc()
 * will set normal priority by default */
MessageQ_setMsgPri(msg, MessageQ_NORMALPRI);
MessageQ_put(queueId, msg);

/* ...or... */

/* Route over MessageQ's MessageQ interface high priority transport */
MessageQ_setMsgPri(msg, MessageQ_HIGHPRI);
MessageQ_put(queueId, msg);

/* ...or... */

/* Route over MessageQ's Network interface
 * TID value has to be a value between 1 and 7 */
MessageQ_setTransportId(msg, transport_tid);
MessageQ_put(queueId, msg);
```

The TransportSrio must be created and registered with MessageQ after IPC has started so that all initialization requirements for the SRIO transport can be satisfied. First, and most importantly, TransportSrio initialization will be making resource requests from the CPPI, QMSS, and SRIO LLDs. As a result, the Resource Manager (RM) LLD must be fully initialized and a transport path from RM Clients to the RM Server must be available. Typically, the TransportShmNotify is used to enable RM message passing between the Clients and Server. Second, forcing TransportSrio initialization after IPC start and sync allows any CPPI Host descriptors and attached buffers to be placed in any type of device memory.

TransportSrio relies on IPC MultiProc's cluster functionality in order to communicate with remote device DSP cores running TransportSrio. The application must define the entire processor topology for the MultiProc module. The number of processors across all devices must be defined for MultiProc. The application's RTSC .cfg file must also define the local device's cluster base ID for MultiProc. An example for an application spanning three devices:

Device A

```
var MultiProc = xdc.useModule('ti.sdo.utils.MultiProc');
/* Cluster definitions - Example has three clusters, one for each
device.  Each cluster
 * has two DSPs within it.
 * Device A [Cluster Base ID: 0] - 1 Host + 2 DSPs (Procs)
 * Device B [Cluster Base ID: 3] - 1 Host + 2 DSPs (Procs)
 * Device C [Cluster Base ID: 3] - 1 Host + 2 DSPs (Procs)
 * Total of 3 Hosts + 6 DSPs (Procs) */
MultiProc.numProcessors = 9;
/* baseIdOfCluster and numProcessors must be set BEFORE setConfig is
run */
MultiProc.numProcsInCluster = 3;
MultiProc.baseIdOfCluster = 0;
var procNameList = ["HOST", "CORE0", "CORE1"];
MultiProc.setConfig(null, procNameList);
```

Device B

```
var MultiProc = xdc.useModule('ti.sdo.utils.MultiProc');
/* Cluster definitions - Example has three clusters, one for each
device.  Each cluster
 * has two DSPs within it.
 * Device A [Cluster Base ID: 0] - 1 Host + 2 DSPs (Procs)
 * Device B [Cluster Base ID: 3] - 1 Host + 2 DSPs (Procs)
 * Device C [Cluster Base ID: 3] - 1 Host + 2 DSPs (Procs)
 * Total of 3 Hosts + 6 DSPs (Procs) */
MultiProc.numProcessors = 9;
/* baseIdOfCluster and numProcessors must be set BEFORE setConfig is
run */
MultiProc.numProcsInCluster = 3;
MultiProc.baseIdOfCluster = 3;
var procNameList = ["HOST", "CORE0", "CORE1"];
MultiProc.setConfig(null, procNameList);
```

Device C

```
var MultiProc = xdc.useModule('ti.sdo.utils.MultiProc');
/* Cluster definitions - Example has three clusters, one for each
device.  Each cluster
 * has two DSPs within it.
 * Device A [Cluster Base ID: 0] - 1 Host + 2 DSPs (Procs)
 * Device B [Cluster Base ID: 3] - 1 Host + 2 DSPs (Procs)
 * Device C [Cluster Base ID: 3] - 1 Host + 2 DSPs (Procs)
 * Total of 3 Hosts + 6 DSPs (Procs) */
MultiProc.numProcessors = 9;
/* baseIdOfCluster and numProcessors must be set BEFORE setConfig is
run */
MultiProc.numProcsInCluster = 3;
MultiProc.baseIdOfCluster = 6;
var procNameList = ["HOST", "CORE0", "CORE1"];
MultiProc.setConfig(null, procNameList);
```

SYS/BIOS DSP TransportSrio Source Delivery and Recompilation

The SYS/BIOS DSP TransportSrio source code and examples are delivered within the MCSDK BIOS PDK component. DSP TransportSrio can be rebuilt using the environment setup scripts provided with the PDK package. DSP TransportSrio example applications are created as part of the pdkProjectCreate scripts. They can be imported and built the same as PDK LLD example and test CCS projects.

Recompiling on Windows

1. Open a Windows command terminal and navigate to <pdk_install_dir>/packages.
2. Set the component install paths. The following commands can be used assuming installation of MCSDK 3.1.3.6 (Presuming CCS and MCSDK 3.1.3 is installed in C:\ti\)
```
set C6X_GEN_INSTALL_PATH="C:\ti\ccsv5\tools\compiler\c6000_7.4.8"
set XDC_INSTALL_PATH=C:\ti\xdctools_3_30_05_60
set EDMA3LLD_BIOS6_INSTALLDIR="C:\ti\edma3_lld_02_11_13_17"
set CG_XML_BIN_INSTALL_PATH=C:\ti\cg_xml\bin
set BIOS_INSTALL_PATH=C:\ti\bios_6_41_00_26\packages
set IPC_INSTALL_PATH=C:\ti\ipc_3_35_01_07\packages
set PDK_INSTALL_PATH=C:\ti\pdk_keystone2_3_01_03_06
```
3. Run pdksetupenv.bat
```
>pdksetupenv.bat
```
4. Navigate to <pdk_install_path>/packages/ti/transport/ipc/c66/srio/
5. Build the IPC SRIO Transport library
```
>xdc
```

Issue the following commands if the SRIO transport ever needs to be rebuilt:

```
>xdc clean
>xdc
```

Recompiling on Linux

1. Open a Linux bash terminal and navigate to <pdk_install_dir>/packages.
2. Export the component install paths. The following commands can be used assuming installation of MCSDK 3.1.3.6 (Presuming CCS and MCSDK 3.1.3 is installed in /opt/ti)
```
export C6X_GEN_INSTALL_PATH=/opt/ti/ccsv5/tools/compiler/c6000_7.4.8
export XDC_INSTALL_PATH=/opt/ti/xdctools_3_30_05_60
export EDMA3LLD_BIOS6_INSTALLDIR=/opt/ti/edma3_lld_02_11_13_17
export CG_XML_BIN_INSTALL_PATH=/opt/ti/cg_xml/bin
export BIOS_INSTALL_PATH=/opt/ti/bios_6_41_00_26/packages
export IPC_INSTALL_PATH=/opt/ti/ipc_3_35_01_07/packages
export PDK_INSTALL_PATH=/opt/ti/pdk_keystone2_3_01_03_06
```
3. Run pdksetupenv.sh
```
$ source pdksetupenv.sh
```
4. Navigate to <pdk_install_path>/packages/ti/transport/ipc/c66/srio/
5. Build the IPC SRIO Transport library
```
$ xdc
```

Issue the following commands if the SRIO transport ever needs to be rebuilt:

```
$ xdc clean
$ xdc
```

SYS/BIOS DSP TransportSrio Configuration Parameters

Following are the configuration parameters for DSP TransportSrio instance creation. Descriptions, default values, and programming considerations are provided for each configuration parameter. Each parameter is an element of the TransportSrio_Params structure. A structure of this type must be created, populated, and passed to the TransportSrio_create() function via pointer. The structure should be initialized to its default values using the TransportSrio_Params_init() function prior to population with user specific parameters.

```
    TransportSrio_Params  transSrioParams;

    ...
    TransportSrio_Params_init(&transSrioParams);
    transSrioParams.deviceCfgParams   = ...;
    transSrioParams.txMemRegion       = ...;
    ...
    srioTransHandle = TransportSrio_create(&transSrioParams,
&errorBlock);
```

Configuration Parameter Element	Description	Initial Value	Special Considerations
`TransportSrio_DeviceConfigParams *deviceCfgParams;`	Pointer to the device specific TransportSrio configuration parameter structure.	NULL	TransportSrio configuration structure is defined in the TransportSrio_device.c source file for supported devices.
`Int txMemRegion;`	QMSS memory region from which to allocate transmit side Host descriptors.	-1	• Descriptors inserted into this region must be of Host type. • The descriptors can be located in L2, MSMC, or DDR3 memory. The base address of the descriptors must be cache line aligned if located within a shared memory (MSMC or DDR3) and caching is enabled.

`UInt32 txNumDesc;`	Number of Host descriptors to pre-allocate for SRIO transmit operations. MessageQ data buffers are attached to the buffers and sent out via the SRIO LLD. Descriptors are recycled onto a completion queue. Descriptors, and their attached buffers, are recycled in future TransportSrio_put operations.	2	Minimum of two descriptors needed for a ping-pong-like operation. While one descriptor+buffer pair is sent, the other is recycled.
`UInt32 txDescSize;`	Size of the transmit descriptors in bytes.	0	Cache coherence operations may be performed on the descriptors based on their memory location. As a result, the descriptor size should be a multiple a cache line.
`UInt32 rxQType;`	The QMSS queue type that will be opened and used with the receive accumulator. The queue type does not matter. Accumulator GEM events are mapped to the accumulator channels, not the queue type+value.	0	

`Int rxMemRegion;`	QMSS memory region from which to allocate receive side Host descriptors.	-1	• This memory region can be the same as that provided for txMemRegion. • Descriptors inserted into this region must be of Host type. • The descriptors can be located in L2, MSMC, or DDR3 memory. The base address of the descriptors must be cache line aligned if located within a shared memory (MSMC or DDR3) and caching is enabled.
`UInt32 rxNumDesc;`	Number of Host descriptors to pre-allocate for SRIO receive operations. MessageQ data buffers are pre-allocated and attached to the descriptors at TransportSrio_create() time. Data received by the SRIO LLD is copied directly into the MessageQ buffer attached the receive descriptor.	1	Minimum of one descriptor needed to receive a packet. The descriptors are reused for receive operations. New MessageQ buffers are allocated and attached to descriptors prior to their reuse.
`UInt32 rxDescSize;`	Size of the receive descriptors in bytes.	0	Cache coherence operations may be performed on the descriptors based on their memory location. As a result, the descriptor size should be a multiple a cache line.

UInt16 rxMsgQHeapId;	Rx-side MessageQ Heap ID. MessageQ buffers are pre-allocated out of this heap and attached to descriptors for packets received by the SRIO interface.	~1	• The heap must have AT LEAST rxNumDesc number of buffers. • The heap can reside can reside in L2, MSMC, or DDR3. TransportSrio will perform cache coherence operations on heap buffers residing in shared memory areas. • Buffers should be sized to be a multiple of a cache line if the heap is located in a shared memory. This prevents corruptions when cache coherence operations are performed.
UInt32 maxMTU;	Maximum transmittable unit in bytes that will be handled by TransportSrio. This is also the size of the buffers within the heap mapped to the rxMsgQHeapId.	256	maxMTU should be sized to be a multiple of a cache line. This prevents corruptions when cache coherence operations are performed on rxMsgQHeapId buffers located in a shared memory.
UInt8 accumCh;	The accumulator channel used for SRIO packet reception. The GEM event for the accumulator interrupt will be derived from the provided accumulator channel and the DSP core number.	0	Please refer to the device's Multicore Navigator specification for the proper accumulator channels for each DSP core.

`UInt32 accumTimerCount;`	Number of global timer ticks to delay the periodic accumulator interrupt. A value of zero will cause an interrupt immediately upon a descriptor being placed in the accumulator ping/pong buffer.	0	
`Void *rmServiceHandle;`	RM service handle that will be given to Srio_start. The RM service handle only needs to be provided if the intent is for RM to manage SRIO resources	NULL	
`UInt rxIntVectorId;`	Interrupt vector ID to tie to the receive side accumulator operation.	~1	
`srioSockParams *sockParams;`	Pointer to socket parameters used by the SRIO transport to bind the socket and to route messages to the proper endpoints.	NULL	Socket parameters include the socket type (Type 11 or Type 9), the number of endpoints in the endpoint list, and a pointer to the Type 11 or Type 9 endpoint parameter list. The endpoint parameter list contains the address information for each processor endpoint in the system. The parameter list must be indexed by the IPC MultiProc ID so that all processors can be mapped to a unique SRIO address.
`UInt transNetworkId;`	The transport instance will be registered with MessageQ's network transport interface using the supplied network transport ID. MessageQ messages with a matching transport ID in their MessageQ header will be sent over the transport instance.	0	The MessageQ network interface transport ID must have a value between 1 and 7.

Adding the SYS/BIOS DSP TransportSrio to a DSP Application

TransportSrio requires some special considerations when adding it to an application since it is not a standard, shared memory, IPC transport. As described earlier, TransportSrio is an IPC MessageQ Network interface transport. It relies on IPC and MessageQ being initialized with a, priority-based, shared memory transport prior to creating any TransportSrio instances. The latter occurs by default in the RTSC configuration and Ipc_start(). Creating TransportSrio instances after IPC and MessageQ have initialized allows the transport to be initialized without any hardcoded assumptions about the locations of heaps buffers and QMSS descriptors in memory. It also allows the LLDs used by TransportSrio to request their resources from RM since RM will use the MessageQ shared memory transport as the resource request/response path.

Additions to the Application RTSC .cfg

- TransportSrio requires the CPPI, QMSS, and SRIO LLDs in order to operate. The RM LLD is a requirement for Keystone II devices

```
var Cppi = xdc.loadPackage('ti.drv.cppi');
var Qmss = xdc.loadPackage('ti.drv.qmss');
var Srio = xdc.loadPackage('ti.drv.srio');
var Rm   = xdc.loadPackage('ti.drv.rm');

Program.sectMap[".qmss"] = new Program.SectionSpec();
Program.sectMap[".qmss"] = "MSMCSRAM";

Program.sectMap[".cppi"] = new Program.SectionSpec();
Program.sectMap[".cppi"] = "MSMCSRAM";

Program.sectMap[".sharedGRL"] = new Program.SectionSpec();
Program.sectMap[".sharedGRL"] = "L2SRAM";

Program.sectMap[".sharedPolicy"] = new Program.SectionSpec();
Program.sectMap[".sharedPolicy"] = "L2SRAM";

Program.sectMap[".srioSharedMem"] = new Program.SectionSpec();
Program.sectMap[".srioSharedMem"] = "MSMCSRAM";
```

- The TransportSrio module must be included to pull in the library. MessageQ must be configured with reserve queue which will be used by any MessageQ Network interface transports that are registered. The NameServer module does not work with the MessageQ Network interface transports since these transports can potentially send MessageQ messages off-device. NameServer cannot query outside of the device.

```
var Ipc = xdc.useModule('ti.sdo.ipc.Ipc');
/* Ipc_start() will synchronize all local DSP processors */
Ipc.procSync = Ipc.ProcSync_ALL;
var MessageQ = xdc.useModule('ti.sdo.ipc.MessageQ');
/* Reserve a block of MessageQ queues for use by the MessageQ network
interface
 * transports since they don't use the NameServer module */
MessageQ.numReservedEntries = 4;
```

```
var TransportSrio =
xdc.useModule('ti.transport.ipc.c66.srio.TransportSrio');
```

- The device-specific low-level IPC modules must be included so that the interrupt logic properly associates the device MultiProc IDs to destination interrupt generation. The MultiProc module and the low-level IPC module must both be aware that an ARMv7 processor exists as MultiProc ID 0 on KeyStone II devices.

```
/* Use the correct version of the low-level IPC modules so that the
ARMv7
 * processor is correctly factored into the notification logic */
var NotifyDriverCirc =
xdc.useModule('ti.sdo.ipc.notifyDrivers.NotifyDriverCirc');
var Interrupt = xdc.useModule('ti.ipc.family.tci6638.Interrupt');
NotifyDriverCirc.InterruptProxy = Interrupt;
var VirtQueue = xdc.useModule('ti.ipc.family.tci6638.VirtQueue');

/*  Notify brings in the ti.sdo.ipc.family.Settings module, which does
 *  lots of config magic which will need to be UNDONE later, or setup
 *  earlier, to get the necessary overrides to various IPC module
proxies!
 */
var Notify = xdc.module('ti.sdo.ipc.Notify');
var Ipc = xdc.useModule('ti.sdo.ipc.Ipc');

/* Note: Must call this to override what's done in Settings.xs ! */
Notify.SetupProxy =
xdc.module('ti.ipc.family.tci6638.NotifyCircSetup');
```

- As shown earlier in the Architecture [4] section, the MultiProc parameters must be configured correctly for each device.

Source Code Additions

- A TransportSrio instance can be created between two endpoints after the following:
  - IPC has started and all local DSPs have attached
  - RM instances have been created. Communication between the RM Clients and the RM Server will take place over the default MessageQ interface, priority-based, IPC transport.
  - QMSS and CPPI have been initialized and started
  - The SRIO IP block has been turned ON via the PSC and the SRIO device initialization routine has been executed
  - The heaps that will provide buffers for MessageQ send/receive have been created. The same heap can be used for both transmit and receive. **Note:** The heap used by the TransportSrio receive logic must have a gate that is able to operate within an interrupt context. In the transport examples a GateMP configured for GateMP_LocalProtect_INTERRUPT is used.

```
/* Create the heap that will be used to allocate messages. */
GateMP_Params_init(&gateMpParams);
gateMpParams.localProtect = GateMP_LocalProtect_INTERRUPT;
gateMpHandle = GateMP_create(&gateMpParams);
```

```
HeapBufMP_Params_init(&heapBufParams);
heapBufParams.regionId  = 0;
...
heapBufParams.gate      = gateMpHandle;
heapHandle = HeapBufMP_create(&heapBufParams);
```

- SYS/BIOS DSP TransportSrio instances can be created between end points once all previous requirements are satisfied. A TransportSrio instance of each SRIO type, 11 and 9, can exist on a DSP simultaneously. The following example code comes from the MultiBoard example:

```
    /* Create SRIO type 11 & 9 transport instances.  They will be a
network
     * transport so won't interfere with default MessageQ transport,
shared
     * memory notify transport */

    /* Type 11 configuration */
    TransportSrio_Params_init(&transSrioParamsT11);
    /* Configure common parameters */
    transSrioParamsT11.deviceCfgParams   = &srioTransCfgParams;
    transSrioParamsT11.txMemRegion       = HOST_DESC_MEM_REGION;
    /* Descriptor pool divided between all cores.  Account type 9+11
for send/receive (divide by 4) */
    transSrioParamsT11.txNumDesc        = (HOST_DESC_NUM / 4) /
ipcNumLocalDspCores;
    transSrioParamsT11.txDescSize       = HOST_DESC_SIZE_BYTES;
    transSrioParamsT11.rxQType          =
Qmss_QueueType_HIGH_PRIORITY_QUEUE;
    transSrioParamsT11.rxMemRegion       = HOST_DESC_MEM_REGION;
    /* Descriptor pool divided between all cores.  Account type 9+11
for send/receive (divide by 4) */
    transSrioParamsT11.rxNumDesc        = (HOST_DESC_NUM / 4) /
ipcNumLocalDspCores;
    transSrioParamsT11.rxDescSize       = HOST_DESC_SIZE_BYTES;
    transSrioParamsT11.rxMsgQHeapId      = SRIO_MSGQ_HEAP_ID;
    transSrioParamsT11.maxMTU           = SRIO_MTU_SIZE_BYTES;
    transSrioParamsT11.rmServiceHandle   = rmServiceHandle;
    /* Must map to a valid channel for each DSP core.  Follow
sprugr9f.pdf Table 5-9 */
    transSrioParamsT11.accumCh           = DNUM;
    transSrioParamsT11.accumTimerCount   = 0;
    transSrioParamsT11.transNetworkId    = SRIO_T11_TRANS_NET_ID;
    transSrioParamsT11.rxIntVectorId     = 8;

    memset(&t11EpParams, 0, sizeof(t11EpParams));
    /* Linux Host (Producer) MultiProc ID - 0 */
    t11EpParams[0].tt       = 0;
    t11EpParams[0].deviceId = DEVICE_ID1_8BIT;
    t11EpParams[0].mailbox  = 0;
    t11EpParams[0].letter   = 0;
    t11EpParams[0].segMap   = (sizeof(TstMsg) > 256 ? 1 :0);
    /* Core 0 (Producer) MultiProc ID - 1 */
    t11EpParams[1].tt       = 0;
    t11EpParams[1].deviceId = DEVICE_ID1_8BIT;
    t11EpParams[1].mailbox  = 0;
    t11EpParams[1].letter   = 1;
    t11EpParams[1].segMap   = (sizeof(TstMsg) > 256 ? 1 :0);
    /* Core 1 (Producer) MultiProc ID - 2 */
    t11EpParams[2].tt       = 0;
    t11EpParams[2].deviceId = DEVICE_ID1_8BIT;
    t11EpParams[2].mailbox  = 0;
    t11EpParams[2].letter   = 2;
    t11EpParams[2].segMap   = (sizeof(TstMsg) > 256 ? 1 :0);
    /* Linux Host (Consumer) MultiProc ID - 3 */
    t11EpParams[3].tt       = 0;
    t11EpParams[3].deviceId = DEVICE_ID2_8BIT;
    t11EpParams[3].mailbox  = 0;
    t11EpParams[3].letter   = 0;
    t11EpParams[3].segMap   = (sizeof(TstMsg) > 256 ? 1 :0);
    /* Core 0 (Consumer) MultiProc ID - 4 */
    t11EpParams[4].tt       = 0;
    t11EpParams[4].deviceId = DEVICE_ID2_8BIT;
    t11EpParams[4].mailbox  = 0;
    t11EpParams[4].letter   = 1;
    t11EpParams[4].segMap   = (sizeof(TstMsg) > 256 ? 1 :0);
    /* Core 1 (Consumer) MultiProc ID - 5 */
    t11EpParams[5].tt       = 0;
    t11EpParams[5].deviceId = DEVICE_ID2_8BIT;
    t11EpParams[5].mailbox  = 0;
    t11EpParams[5].letter   = 2;
    t11EpParams[5].segMap   = (sizeof(TstMsg) > 256 ? 1 :0);

    memset(&t11socketParams, 0, sizeof(t11socketParams));
    t11socketParams.epListSize = NUM_TOTAL_CORES;
    t11socketParams.sockType = TransportSrio_srioSockType_TYPE_11;
    t11socketParams.u.pT11Eps = &t11EpParams[0];

    transSrioParamsT11.sockParams = &t11socketParams;

    Error_init(&errorBlock);
    System_printf("IPC Core %d : "
                  "Creating SRIO Transport instance with Type 11
socket\n",
                  ipcCoreId);
    srioT11TransHandle = TransportSrio_create(&transSrioParamsT11,
&errorBlock);
    if (srioT11TransHandle == NULL) {
        System_printf("Error IPC Core %d : "
                      "TransportSrio_create failed with id %d\n",
ipcCoreId,
                      errorBlock.id);
        return;
    }

    /* Type 9 configuration */
    TransportSrio_Params_init(&transSrioParamsT9);
    /* Configure common parameters */
    transSrioParamsT9.deviceCfgParams   = &srioTransCfgParams;
    transSrioParamsT9.txMemRegion       = HOST_DESC_MEM_REGION;
    /* Descriptor pool divided between all cores.
     * Account type 9+11 for send/receive (divide by 4) */
    transSrioParamsT9.txNumDesc         = (HOST_DESC_NUM / 4) /
ipcNumLocalDspCores;
    transSrioParamsT9.txDescSize        = HOST_DESC_SIZE_BYTES;
    transSrioParamsT9.rxQType           =
Qmss_QueueType_HIGH_PRIORITY_QUEUE;
    transSrioParamsT9.rxMemRegion       = HOST_DESC_MEM_REGION;
    /* Descriptor pool divided between all cores.
     * Account type 9+11 for send/receive (divide by 4) */
    transSrioParamsT9.rxNumDesc         = (HOST_DESC_NUM / 4) /
ipcNumLocalDspCores;
    transSrioParamsT9.rxDescSize        = HOST_DESC_SIZE_BYTES;
    transSrioParamsT9.rxMsgQHeapId      = SRIO_MSGQ_HEAP_ID;
    transSrioParamsT9.maxMTU            = SRIO_MTU_SIZE_BYTES;
    transSrioParamsT9.rmServiceHandle   = rmServiceHandle;
    /* Type 9 instance specific parameters */
    /* Must map to a valid channel for each DSP core.
     * Follow sprugr9g.pdf Table 5-9 */
    transSrioParamsT9.accumCh            = DNUM + 8; /* Next valid
accum ch is
                                                      * number of
device DSP
                                                      * cores away
                                                      * (8 for K2HK) */
    transSrioParamsT9.accumTimerCount    = 0;
    transSrioParamsT9.transNetworkId     = SRIO_T9_TRANS_NET_ID;
    transSrioParamsT9.rxIntVectorId      = 9;

    /* Type 9 specific */
    memset(&t9EpParams, 0, sizeof(t9EpParams));
    /* Linux Host (Producer) MultiProc ID - 0 */
    t9EpParams[0].tt       = 0;
    t9EpParams[0].deviceId = DEVICE_ID1_8BIT;
    t9EpParams[0].cos      = 0;
    t9EpParams[0].streamId = 0;
    /* Core 0 (Producer) MultiProc ID - 1 */
    t9EpParams[1].tt       = 0;
    t9EpParams[1].deviceId = DEVICE_ID1_8BIT;
    t9EpParams[1].cos      = 0;
    t9EpParams[1].streamId = 1;
    /* Core 1 (Producer) MultiProc ID - 2 */
    t9EpParams[2].tt       = 0;
    t9EpParams[2].deviceId = DEVICE_ID1_8BIT;
    t9EpParams[2].cos      = 0;
    t9EpParams[2].streamId = 2;
    /* Linux Host (Consumer) MultiProc ID - 3 */
    t9EpParams[3].tt       = 0;
    t9EpParams[3].deviceId = DEVICE_ID2_8BIT;
    t9EpParams[3].cos      = 0;
    t9EpParams[3].streamId = 0;
    /* Core 0 (Consumer) MultiProc ID - 4 */
    t9EpParams[4].tt       = 0;
    t9EpParams[4].deviceId = DEVICE_ID2_8BIT;
    t9EpParams[4].cos      = 0;
    t9EpParams[4].streamId = 1;
    /* Core 1 (Consumer) MultiProc ID - 5 */
    t9EpParams[5].tt       = 0;
    t9EpParams[5].deviceId = DEVICE_ID2_8BIT;
    t9EpParams[5].cos      = 0;
    t9EpParams[5].streamId = 2;

    memset(&t9socketParams, 0, sizeof(t9socketParams));
    t9socketParams.epListSize = NUM_TOTAL_CORES;
    t9socketParams.sockType = TransportSrio_srioSockType_TYPE_9;
    t9socketParams.u.pT9Eps = &t9EpParams[0];

    transSrioParamsT9.sockParams = &t9socketParams;

    Error_init(&errorBlock);
    System_printf("IPC Core %d : Creating SRIO Transport instance with
Type 9 socket\n", ipcCoreId);
    srioT9TransHandle = TransportSrio_create(&transSrioParamsT9,
&errorBlock);
    if (srioT9TransHandle == NULL) {
        System_printf("Error IPC Core %d : TransportSrio_create failed
with id %d\n", ipcCoreId,
                      errorBlock.id);
        return;
    }
```

- MessageQ_create() and MessageQ_open() must utilize the reserve queues set-aside in MessageQ since the network interface transports, like TransportSrio, do not use the NameServer. To utilize the reserved queues:
  - To create a local reserved queue MessageQ_create() will take NULL instead of a string containing a remote queue name
  - To open a remote reserved queue MessageQ_open() is replaced by the MessageQ_openQueueId() API

```
    System_printf("IPC Core %d : Creating reserved MessageQ %d\n",
ipcCoreId, MESSAGEQ_RESERVED_RCV_Q);
    MessageQ_Params_init(&msgQParams);
    msgQParams.queueIndex = MESSAGEQ_RESERVED_RCV_Q;
    /* Create reserved message queue. */
    localMessageQ = MessageQ_create(NULL, &msgQParams);
    if (localMessageQ == NULL) {
        System_printf("Error IPC Core %d : MessageQ_create failed\n",
ipcCoreId);
        return;
    }

    System_printf("IPC Core %d : Opening reserved MessageQ %d on IPC
core %d\n", ipcCoreId,
                  MESSAGEQ_RESERVED_RCV_Q, remoteProcId);
    remoteQueueId = MessageQ_openQueueId(MESSAGEQ_RESERVED_RCV_Q,
remoteProcId);
```

- MessageQ_put() and MessageQ_get() can be used normally to send and receive messages between end points using TransportSrio. The only caveat is that messages that are to use TransportSrio must be sent with the transport ID, TID, that was used to register the TransportSrio instance with the MessageQ Network interface.

```
typedef struct {
    MessageQ_MsgHeader header; /* 32 bytes */
    int32_t            src;
    int32_t            flags;
    int32_t            numMsgs;
    int32_t            seqNum;
    uint32_t           data[TEST_MSG_DATA_LEN_WORDS];
    uint8_t            pad[16]; /* Pad to cache line size of 128 bytes
*/
} TstMsg;

...

TstMsg   *txMsg;

txMsg = (TstMsg *) MessageQ_alloc(SRIO_MSGQ_HEAP_ID, sizeof(TstMsg));
/* Set the transport ID to route message through Type 11 SRIO Transport
 instance */
MessageQ_setTransportId(txMsg, SRIO_T11_TRANS_NET_ID);

/* OR */
/* Set the transport ID to route message through Type 9 SRIO Transport
instance */
MessageQ_setTransportId(txMsg, SRIO_T9_TRANS_NET_ID);

/* Send message */
MessageQ_put(remoteQueueId, (MessageQ_Msg) txMsg);
```

Note: Please pay extra attention to the alignments and padding of CPPI descriptors and heap buffers used by the SRIO Transport. Cache coherence operations will be performed on descriptors and buffers that are detected to be from a shared memory, such as MSMC or DDR3. All descriptors and buffers should be cache line aligned and padded to avoid data corruptions when the coherence operations are performed.

SYS/BIOS DSP TransportSrio Examples

Two examples are included with the SYS/BIOS DSP TransportSrio component. Both examples are fully integrated with RM so they can be run while ARM Linux is up:

- transportIpcSrioBenchmarkK2XExampleProject - An example that performs latency, throughput, and data integrity tests over DSP TransportSrio. In addition, it tests the MessageQ routing capabilities via the NORMALPRI and HIGHPRI flags.
- transportIpcSrioMultiBoardK2XExampleProject - An example that sends packets from two DSPs on one EVM to two DSPs on another EVM. A data integrity check is performed on the data transferred between the two devices. Two different executables are required for this example.
  - transportIpcSrioMultiBoardProducerK2XExampleProject - The producer side of the multi-board test. The producer opens two MessageQ's on the consumer device and uses them to send MessageQ messages to the consumer device.
  - transportIpcSrioMultiBoardConsumerK2XExampleProject - The consumer side of the multi-board test. The consumer creates two MessageQ's and waits for the producer to open them and start sending data. Data integrity checks are performed on the data received from the producer device.

The projects are built as part of the generic pdkProjectCreate process. They can be imported, built, and run through CCS just like any other LLD example and test project.

Note: The multi-board test requires two Keystone II devices connected via SRIO lanes by way of a breakout card or chassis in order to operate successfully.

ARMv7 Linux TransportSrio

ARMv7 Linux TransportSrio is the ARMv7 Linux MessageQ Network interface transport counterpart to the SYS/BIOS DSP version of TransportSrio. The ARMv7 Linux version of TransportSrio can be registered with Network transport interface of ARMv7 Linux MessageQ to allow SRIO-based communication with other ARMv7 and DSP processors running TransportSrio on the local, or a remote, device. Communication between local and remote device processors is achieved by each processor having a unique MultiProc ID assigned to it. The MultiProc ID mapping is maintained within the LAD daemon in the ARMv7 Linux version of IPC. Each Linux Host within a system must be running a LAD daemon that has been configured to know about the maximum number of processors in the system. The LAD daemon must also be configured with a unique cluster base ID within the processor ID space. The MultiProc ID mappings maintained by the LAD daemon on each Linux Host must be in sync with ID configurations on all other processors, ARMv7 and DSP, in the system.

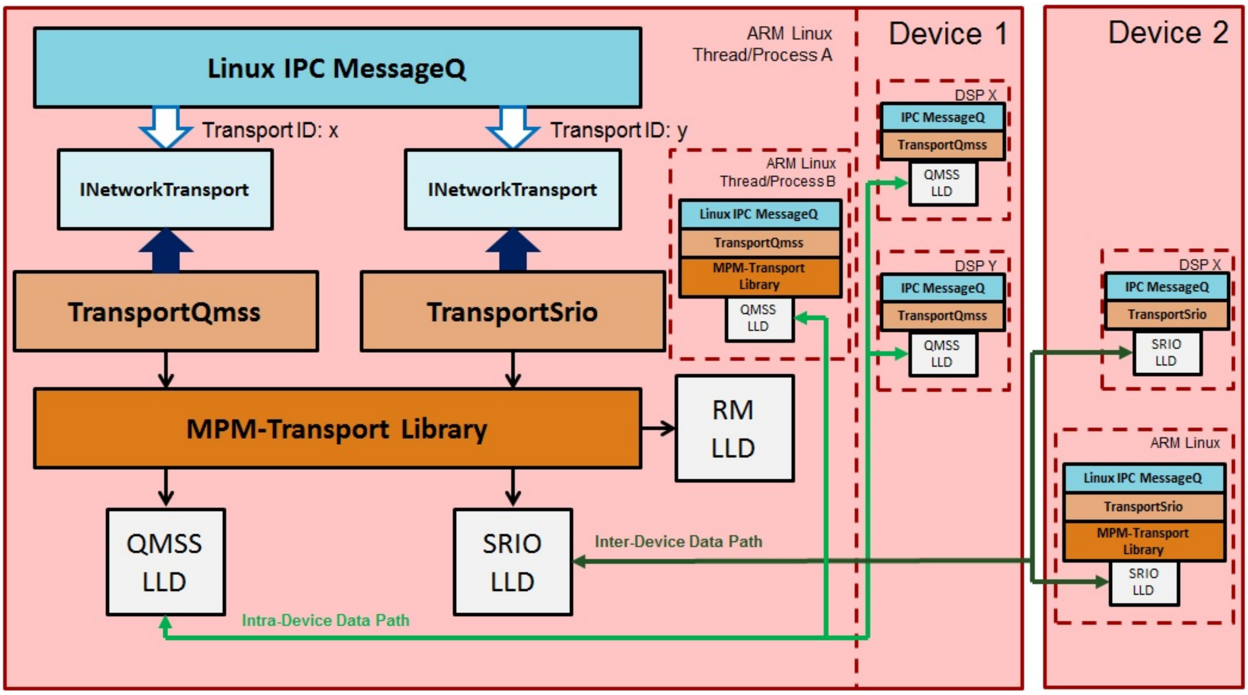
Architecture

TransportSrio does not directly interact with SRIO. Instead the transport interfaces with a MPM-Transport instance that supports SRIO.

Only one TransportSrio instance of each Type (11 and 9) can exist across all Linux processes. This was a design decision made so that SRIO addresses could be assigned based on MultiProc ID rather than a combination of MultiProc ID + MessageQ queue ID. The latter scheme would overly complicate SRIO address assignments to processors in a multi-device system. A reroute functionality exists within the TransportSrio reception logic to compensate for the lack of direct process reception for more than one Linux process. The reroute logic checks the destination MessageQ queue in the received MessageQ message header. The message is sent over a registered process to process MessageQ transport if the destination queue does not exist within the receiving Linux process. The transport over which the reroute occurs is mapped to a TID provided at TransportSrio instance creation.

An ARMv7 Linux processor running a TransportSrio instance can communicate with any other processor on any device as long as:

- The processors are running an ARMv7 Linux TransportSrio or SYS/BIOS DSP TransportSrio instance
- All processors have a synchronized mapping of MultiProc ID to SRIO addresses
- All processors are physically connected over SRIO lanes.



ARMv7 Linux TransportSrio Source Delivery and Recompilation

The ARMv7 Linux TransportSrio source code can be downloaded and built two ways. The transport source code is delivered and built as part of Yocto/bitbake. The source code can also be downloaded and built directly from the GIT repository.

Recompiling Through Yocto/bitbake

1. Follow the instructions in the Exploring section of the user guide to configure the Yocto build environment [2]. The tisdk-server-rootfs-image does not need to be built. Instead look at the section for building other components [5]
2. Build the TransportSrio libraries, ipc-transport-srio recipe, and user-space tests, ipc-transport-srio-test recipe:

```
$ MACHINE=k2hk-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake ipc-transport-srio
$ MACHINE=k2hk-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake
```

ipc-transport-srio-test

Note: The initial build may take quite some time since the kernel is built as a dependency

Note: Building with just the ipc-transport-srio-test recipe will also build the ipc-transport-srio recipe since the test recipe depends on the library recipe.

3. The built TransportSrio static library will be located in <base_path>/oe-layersetup/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi/ipc-transp
   The built TransportSrio shared library will be located in <base_path>/oe-layersetup/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi/ipc-transp
4. The ipc-transport-srio-test recipe will build test static and shared library executables for all supported devices.
   The executables will be located in base_path>/oe-layersetup/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi/ipc-transpo

Recompiling Through GIT Repository

Recompiling through the ARMv7 Linux TransportSrio GIT repository requires that the latest MCSDK Linux installation. The MCSDK Linux PDK component and the Linux devkit must be installed. The Linux devkit installation script can be found in <MCSDK Linux install root>/mcsdk_linux_3_XX_YY_ZZ/linux-devkit/

1. Clone the keystone-linux/ipc-transport repository from git.ti.com
   $ git clone git://git.ti.com/keystone-linux/ipc-transport.git
2. Navigate to the MCSDK Linux installation of pdk_3_XX_YY_ZZ/packages and source armv7setupenv.sh.
   Note: The armv7setupenv.sh script must be modified to point to the linaro toolchain and installed devkit path
   $ source armv7setupenv.sh
3. Navigate back to the SRIO transport directory in the ipc-transport GIT repository
   $ cd <repo_root_path>/ipc-transport/linus/srio
4. Build the TransportSrio library and user-space test executables:
   $ make lib
   $ make tests
5. The TransportSrio static and shared libraries will be copied directly into the Linux devkit's /usr/lib folder as long as the devkit install path was setup correctly prior to running the armv7setupenv.sh script
6. The test executables will be generated in the <base_repo_path>/ipc-transport/bin/<k2 device>/test/ folder. Only the device specified in the armv7setupenv.sh will be built.
   Note: Setting the USEDYNAMIC_LIB environment variable to "yes" will generate the shared library test executables
   $ export USEDYNAMIC_LIB=yes

ARMv7 Linux TransportSrio Configuration Parameters

Following are the configuration parameters for TransportSrio instance creation. Descriptions, default values, and programming considerations are provided for each configuration parameter. Each parameter is an element of the TransportSrio_Params structure. A structure of this type must be created, populated, and passed to the TransportSrio_create() function via pointer. The structure should be initialized to its default values using the TransportSrio_Params_init() function prior to population with application specific parameters.

```
TransportSrio_Params  srio_trans_params;

...
TransportSrio_Params_init(&srio_trans_params);
snprintf(mpm_inst_name, MPM_INST_NAME_LEN, "arm-srio-generic");
srio_trans_params.mpm_trans_inst_name = mpm_inst_name;
```

```
    srio_trans_params.rm_service_h          = ...;
    ...
    srio_handle = TransportSrio_create(&srio_trans_params);
```

Configuration Parameter Element	Description	Initial Value	Special Considerations
`Char *mpm_trans_inst_name;`	MPM-Transport instance name. This string must match a "slaves" name string defined within the mpm_config.json file used in the Linux filesystem	NULL	This string name must match the generic SRIO instance name in the filesystems /etc/mpm/mpm_config.json. The MPM JSON file's generic SRIO string is "arm-srio-generic".
`Void *rm_service_h;`	RM instance service handle needed by MPM-Transport to request hardware resources	NULL	
`TransportSrio_SocketParams *sock_params;`	Pointer to socket parameters used by the SRIO transport to bind the socket and to route messages to the proper endpoints.	NULL	• Socket parameters include the socket type (Type 11 or Type 9), the number of endpoints in the endpoint list, and a pointer to the Type 11 or Type 9 endpoint parameter list. The endpoint parameter list contains the address information for each processor endpoint in the system. The parameter list must be indexed by the IPC MultiProc ID so that all processors can be mapped to a unique SRIO address. • This structure will be replicated within the TransportSrio instance. So structure passed by pointer can be allocated from temporary memory.

`Int rx_msg_size_bytes;`	Maximum size in bytes of messages that will be received by this transport. Used to allocated MessageQ messages for reception	0	The value specified must be in sync with the "sizebuf" values specified in the "qmss-queue-map" transmit and receive free queue nodes of the mpm_config.json configuration file.
`Int reroute_tid;`	MessageQ Network interface Transport ID of ARMv7 Linux TransportQmss instance. Will be used to reroute received MessageQ messages destined for a MessageQ queue that was opened from another Linux process. Messages received for a MessageQ queue opened from another Linux process will be dropped if this value is left as 0.	0	A valid TID value is between 1 and 7.

`Int (*srio_device_init) (Void *init_cfg,`
`                         Void *srio_base_addr,`
`                         UInt32 serdes_addr);`
	Pointer to the SRIO device initialization routine. This routine initializes the SRIO hardware with routing and address information. MPM-Transport will call the provided function pointer during the SRIO initialization process.	NULL	• Routine should be run once per device.
			• Function pointer should be NULL if another process or device core was the first to initialize the usage of SRIO.
			• The init_cfg input parameter can be a pointer to an application specific input parameter to the srio_device_init function

`Void *init_cfg;`	Application specific input parameter to the srio_device_init implementation	NULL	

`Int (*srio_device_deinit) (Void *deinit_cfg,`
`Void *srio_base_addr);`
	Pointer to the SRIO device de-initialization routine. Will be invoked by MPM-Transport when de-initializing the SRIO hardware within deletion routines.	NULL	• Routine should be run once per device
			• Function pointer should be NULL if another process or device core will be the last to delete its usage of SRIO
			• The deinit_cfg input parameter can be a pointer to an application specific input parameter to the srio_device_deinit function.

`Void *deinit_cfg;`	Application specific input parameter to the srio_device_deinit implementation	NULL	
`Int mpm_trans_init_qmss;`	Controls whether MPM-Transport will initialize the QMSS hardware 0 - MPM-Transport will not initialize QMSS. Set if another entity within the system initialized the QMSS hardware. 1 - MPM-Transport will initialize QMSS. Set if this is the first system entity being created that used QMSS and QMSS has not been initialized	0	

MPM Transport Configuration Effects on ARMv7 Linux TransportSrio

TransportSrio assumes MPM Transport will manage configuration of the QMSS, CPPI, and SRIO LLDs. As a result, descriptor and descriptor buffer management is pushed to MPM Transport in the ARMv7 Linux version of TransportSrio. The MPM Transport JSON configuration file should be modified in order to change QMSS descriptor and buffer related parameters.

The MPM Transport JSON configuration file is located in the Linux file system at /etc/mpm/mpm_config.json

Adding the ARMv7 Linux TransportSrio to a User-Space Application

A TransportSrio instance can be created after the following:

- IPC has started
- A RM client instance has been created. Communication between the RM Client and the RM Server will take place over a Linux socket.
- [**Optional**] A TransportQmss instance has been created so that TransportSrio can reroute MessageQ messages received over SRIO that are destined for a Linux process other than the process in which the TransportSrio instance will be created.
- ARMv7 Linux TransportSrio instances can be created once all previous requirements are satisfied. A TransportSrio instance of each SRIO type, 11 and 9, can exist simultaneously in Linux user space. The following example code comes from the Producer-side of the Producer/Consumer example:

```
/* Type 11 instance */

TransportSrio_Params_init(&srio_trans_params);
snprintf(mpm_inst_name, MPM_INST_NAME_LEN, "arm-srio-generic");
srio_trans_params.mpm_trans_inst_name = mpm_inst_name;
srio_trans_params.rm_service_h        = rm_service_h;
srio_trans_params.rx_msg_size_bytes   = MAX_PACKET_SIZE;
srio_trans_params.reroute_tid         = TRANS_QMSS_NET_ID;
srio_trans_params.srio_device_init    = &mySrioDevice_init;
srio_trans_params.init_cfg            = (void *)&path_mode;
srio_trans_params.srio_device_deinit  = &mySrioDevice_deinit;
srio_trans_params.deinit_cfg          = NULL;
/* TransportQmss informs mpm-transport to init QMSS */
srio_trans_params.mpm_trans_init_qmss = 0;

/* Configure producer's static socket parameters. Structures can
 * be local since TransportSrio will make a copy */
memset(&t11_params, 0, sizeof(t11_params));
sock_params.num_eps   = MAX_SYSTEM_PROCESSORS;
sock_params.sock_type = sock_TYPE_11;

/* Linux Host (Producer) */
t11_params[0].tt        = 0;
t11_params[0].device_id = DEVICE_ID1_8BIT;
t11_params[0].letter    = 0;
t11_params[0].mailbox   = 0;
t11_params[0].seg_map   = (MAX_PACKET_SIZE > 256) ? 1 : 0;

/* Linux Host (Consumer) */
t11_params[9].tt        = 0;
t11_params[9].device_id = DEVICE_ID2_8BIT;
t11_params[9].letter    = 0;
t11_params[9].mailbox   = 0;
t11_params[9].seg_map   = (MAX_PACKET_SIZE > 256) ? 1 : 0;
sock_params.u.t11_eps = &t11_params[0];
```

```
    srio_trans_params.sock_params = &sock_params;
    printf("Process %d : Creating TransportSrio Type 11 instance\n",
           local_process);
    srio_trans_t11_h = TransportSrio_create(&srio_trans_params);
    if (!srio_trans_t11_h) {
        printf("ERROR Process %d : "
               "Failed to create TransportSrio Type 11 handle\n",
               local_process);
        status = -1;
        goto err_exit;
    }

    /* Register transport with MessageQ as network transport */
    net_trans_h = TransportSrio_upCast(srio_trans_t11_h);
    base_trans_h = INetworkTransport_upCast(net_trans_h);
    if (MessageQ_registerTransportId(TRANS_SRIO_T11_NET_ID,
base_trans_h) < 0) {
        printf("ERROR Process %d : "
               "Failed to register TransportSrio Type 11 as network "
               "transport with TID\n", local_process,
TRANS_SRIO_T11_NET_ID);
        status = -1;
        goto err_exit;
    }

    /* Type 9 instance */

    TransportSrio_Params_init(&srio_trans_params);
    snprintf(mpm_inst_name, MPM_INST_NAME_LEN, "arm-srio-generic");
    srio_trans_params.mpm_trans_inst_name = mpm_inst_name;
    srio_trans_params.rm_service_h        = rm_service_h;
    srio_trans_params.rx_msg_size_bytes   = MAX_PACKET_SIZE;
    srio_trans_params.reroute_tid         = TRANS_QMSS_NET_ID;
    /* Don't run device init/deinit for second transport */
    srio_trans_params.srio_device_init    = NULL;
    srio_trans_params.init_cfg            = NULL;
    srio_trans_params.srio_device_deinit  = NULL;
    srio_trans_params.deinit_cfg          = NULL;
    /* TransportQmss informs mpm-transport to init QMSS */
    srio_trans_params.mpm_trans_init_qmss = 0;

    /* Configure producer's static socket parameters. Structures can
     * be local since TransportSrio will make a copy */
    memset(&t9_params, 0, sizeof(t9_params));
    sock_params.num_eps    = MAX_SYSTEM_PROCESSORS;
    sock_params.sock_type = sock_TYPE_9;
    /* Linux Host (Producer) */
    t9_params[0].tt        = 0;
    t9_params[0].device_id = DEVICE_ID1_8BIT;
    t9_params[0].cos       = 0;
    t9_params[0].stream_id = 0;

    /* Linux Host (Consumer) */
    t9_params[9].tt        = 0;
    t9_params[9].device_id = DEVICE_ID2_8BIT;
    t9_params[9].cos       = 0;
    t9_params[9].stream_id = 0;
    sock_params.u.t9_eps = &t9_params[0];
    srio_trans_params.sock_params = &sock_params;
    printf("Process %d : Creating TransportSrio Type 9 instance\n",
           local_process);
    srio_trans_t9_h = TransportSrio_create(&srio_trans_params);
    if (!srio_trans_t9_h) {
        printf("ERROR Process %d : "
               "Failed to create TransportSrio Type 9 handle\n",
               local_process);
        status = -1;
        goto err_exit;
    }

    /* Register transport with MessageQ as network transport */
    net_trans_h = TransportSrio_upCast(srio_trans_t9_h);
    base_trans_h = INetworkTransport_upCast(net_trans_h);
    if (MessageQ_registerTransportId(TRANS_SRIO_T9_NET_ID,
base_trans_h) < 0) {
        printf("ERROR Process %d : "
               "Failed to register TransportSrio Type 9 as network "
               "transport with TID %d\n", local_process,
TRANS_SRIO_T9_NET_ID);
        status = -1;
        goto err_exit;
    }
```

- MessageQ_create() and MessageQ_open() must utilize the reserve queues set-aside in MessageQ since the network interface transports do not use the NameServer. On the Linux the IPC LAD daemon must have its configuration changed to reserve queues. To utilize the reserved queues:
  - To create a local reserved queue MessageQ_create() will take NULL instead of a string containing a remote queue name
  - To open a remote reserved queue MessageQ_open() is replaced by the MessageQ_openQueueId() API

```
        /* Create queue used to receive messages from remote devices
and open
         * remote device queues - one per process */
        MessageQ_Params_init(&msg_params);
        msg_params.queueIndex = MESSAGEQ_RESERVED_RCV_Q;
        srio_msg_q_h = MessageQ_create(NULL, &msg_params);
        if (srio_msg_q_h == NULL) {
            printf("ERROR Process %d : Failed to create MessageQ\n",
                   local_process);
            status = -1;
            goto err_exit;
        }
        printf("Process %d : "
               "Created remote device reception MessageQ with QId:
0x%x\n",
               local_process, MessageQ_getQueueId(srio_msg_q_h));

        /* Open Consumer Linux Host MessageQ queues on each process of
the
         * remote Linux Host */
        for (i = 0; i < CONSUMER_PROCESSES; i++) {
            rem_srio_q_id[i] =
MessageQ_openQueueId(MESSAGEQ_RESERVED_RCV_Q + i,

CONSUMER_HOST_PROC_ID);
            printf("Process %d : Opened remote device QId: 0x%x\n",
                   local_process, rem_srio_q_id[i]);
        }
```

- MessageQ_put() and MessageQ_get() can be used normally to send and receive messages between end points using TransportSrio. The only caveat is that messages that are to use TransportSrio must be sent with the transport ID, TID, that was used to register the TransportSrio instance with the MessageQ Network interface.

```
/* TransportSrio Type 11 MessageQ Network Interface TID */
#define SRIO_T11_TRANS_NET_ID 1
/* TransportSrio Type 9 MessageQ Network Interface TID */
#define SRIO_T9_TRANS_NET_ID 2
typedef struct {
    MessageQ_MsgHeader header; /* 32 bytes */
    int32_t            src;
    int32_t            flags;
    int32_t            numMsgs;
    int32_t            seqNum;
    uint32_t           data[TEST_MSG_DATA_LEN_WORDS];
    uint8_t            pad[16]; /* Pad to cache line size of 128 bytes
*/
} test_msg_t;

...

test_msg_t *msg = NULL;
```

```
msg = (test_msg_t *) MessageQ_alloc(0, sizeof(*msg));
if (msg == NULL) {
    printf("ERROR Process %d : MessageQ_alloc failed\n",
           local_process);
    goto err_exit;
}

...

/* Set the transport ID to route message through Type 11 SRIO Transport
 instance */
MessageQ_setTransportId(msg, SRIO_T11_TRANS_NET_ID);

/* OR */

/* Set the transport ID to route message through Type 9 SRIO Transport
instance */
MessageQ_setTransportId(msg, SRIO_T9_TRANS_NET_ID);

/* Send message */
status = MessageQ_put(rem_srio_q_id, (MessageQ_Msg)msg);
if (status < 0) {
    printf("ERROR Process %d : MessageQ_put failed\n",
           local_process);
    goto err_exit;
}
```

ARMv7 Linux TransportSrio Tests

ARMv7 Linux TransportSrio includes a single test that uses TransportSrio to send MessageQ messages between Linux Hosts on two different KeyStone II devices. The test consists of two user-space executables, producer.out and consumer.out. The producer and consumer applications will create Type 11 and Type 9 TransportSrio instances. A TransportQmss instance will also be created in the consumer application and provided to TransportSrio for multi-process routing. The applications will synchronize over TransportSrio and then a bulk MessageQ message transfer will take place from producer to consumer. The bulk transfer will be run over both the Type 11 and Type 9 instance. The producer application will send messages to two different MessageQ queues on the consumer device. One queue will be located in the process where the TransportSrio instance exists. The other queue will be in a process where only a TransportQmss instance exists. Only one TransportSrio instance of each type can exist on the Linux host at any given time. Messages received by TransportSrio that are destined for the queue located on the other process will be rerouted to that process using the TransportQmss instance.

Building the Producer/Consumer Test

The producer and consumer user-space applications can be built through Yocto/bitbake or through downloading the keystone-linux/ipc-transport GIT repository. For instructions on how to do either please see UG section on ARMv7 Linux TransportSrio Source Delivery [6]

Running the Producer/Consumer Test

Setup is needed on both the producer and consumer devices prior to running the test. The LAD daemon provided with the release file system has been patched to assume up to 65535 processors exist in the system and to reserve 8 MessageQ queues. It has also been patched to take the MultiProc cluster base ID as a command line input.

Producer Setup:
Note: Replace "K2X" and "k2x" with the proper device

1. Load the CMEM module
   $ insmod /lib/modules/3.10.61/extra/cmemk.ko
2. Download and run the IPC DSP minimal startup applcation
   $ mpmcl load dsp0 transportIpcStartupK2XUtilProject.out
   $ mpmcl run dsp0
3. Start the RM Server
   $ rmServer.out /usr/bin/device/k2x/global-resource-list.dtb /usr/bin/device/k2x/policy_dsp_arm.dtb

Consumer Setup:
Note: Replace "K2X" and "k2x" with the proper device

1. Kill and restart LAD with a cluster base ID of 9. This makes the consumer device unique from the producer device from an IPC MultiProc point of view
   $ pkill lad_tci6638
   $ lad_tci6638 -l log.txt -b 9
2. Load the CMEM module
   $ insmod /lib/modules/3.10.61/extra/cmemk.ko
3. Download and run the IPC DSP minimal startup applcation
   $ mpmcl load dsp0 transportIpcStartupK2XUtilProject.out
   $ mpmcl run dsp0
4. Start the RM Server
   $ rmServer.out /usr/bin/device/k2x/global-resource-list.dtb /usr/bin/device/k2x/policy_dsp_arm.dtb

The produce and consumer executables can be run once setup is complete on both devices. Order of execution shouldn't matter but start the consumer.out prior to the producer.out to be safe.

ARMv7 Linux TransportQmss

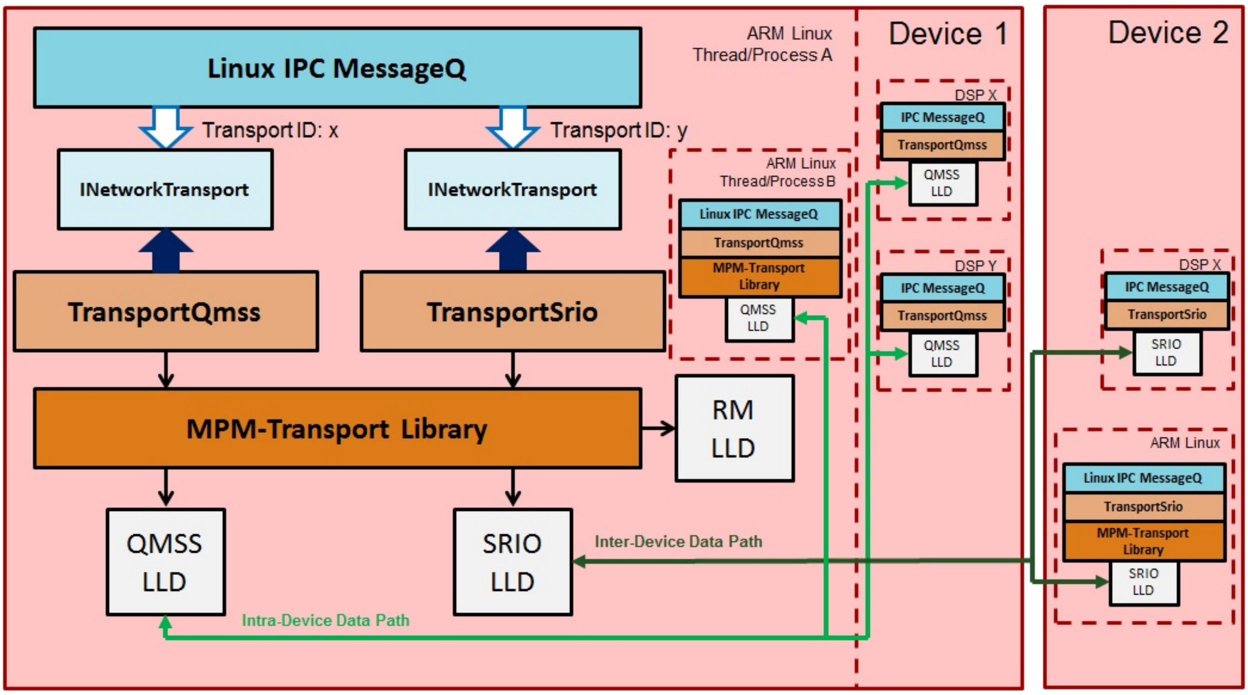
ARMv7 Linux TransportQmss is the ARMv7 Linux MessageQ Network interface transport counterpart to the SYS/BIOS DSP version of TransportQmss (**In Development**). The ARMv7 Linux version of TransportQmss can be registered with Network transport interface of ARMv7 Linux MessageQ to allow QMSS-based communication with other Linux user-space processes and DSP processors running TransportQmss.

Architecture

TransportQmss does not directly interact with QMSS. Instead the transport interfaces with a MPM-Transport instance that supports QMSS.

Each Linux user-space process can create its own TransportQmss instance. MessageQ APIs can be used to send messages between user-space processes as long as each process registers a TransportQmss instance with MessageQ.

Communication with DSPs on the same device will be possible once development of SYS/BIOS DSP TransportQmss is complete.



ARMv7 Linux TransportQmss Source Delivery and Recompilation

The ARMv7 Linux TransportQmss source code can be downloaded and built two ways. The transport source code is delivered and built as part of Yocto/bitbake. The source code can also be downloaded and built directly from the GIT repository.

Recompiling Through Yocto/bitbake

1. Follow the instructions in the Exploring section of the user guide to configure the Yocto build environment [2]. The tisdk-server-rootfs-image does not need to be built. Instead look at the section for building other components [5]
2. Build the TransportQmss libraries, ipc-transport-qmss recipe, and user-space tests, ipc-transport-qmss-test recipe:
   $ MACHINE=k2hk-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake ipc-transport-qmss
   $ MACHINE=k2hk-evm TOOLCHAIN_BRAND=linaro ARAGO_BRAND=mcsdk bitbake ipc-transport-qmss-test
   Note: The initial build may take quite some time since the kernel is built as a dependency
   Note: Building with just the ipc-transport-qmss-test recipe will also build the ipc-transport-qmss recipe since the test recipe depends on the library recipe.
3. The built TransportQmss static library will be located in
   <base_path>/oe-layersetup/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi/ipc-transp
   The built TransportQmss shared library will be located in
   <base_path>/oe-layersetup/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi/ipc-transp
4. The ipc-transport-qmss-test recipe will build test static and shared library executables for all supported devices.
   The executables will be located in
   base_path>/oe-layersetup/build/arago-tmp-external-linaro-toolchain/work/cortexa15hf-vfp-neon-3.8-oe-linux-gnueabi/ipc-transpo

Recompiling Through GIT Repository

Recompiling through the ARMv7 Linux TransportQmss GIT repository requires that the latest MCSDK Linux installation. The MCSDK Linux PDK component and the Linux devkit must be installed. The Linux devkit installation script can be found in <MCSDK Linux install root>/mcsdk_linux_3_XX_YY_ZZ/linux-devkit/

1. Clone the keystone-linux/ipc-transport repository from git.ti.com
   $ git clone git://git.ti.com/keystone-linux/ipc-transport.git
2. Navigate to the MCSDK Linux installation of pdk_3_XX_YY_ZZ/packages and source armv7setupenv.sh.
   Note: The armv7setupenv.sh script must be modified to point to the linaro toolchain and installed devkit path
   $ source armv7setupenv.sh
3. Navigate back to the QMSS transport directory in the ipc-transport GIT repository
   $ cd <repo_root_path>/ipc-transport/linus/qmss
4. Build the TransportQmss library and user-space test executables:
   $ make lib
   $ make tests
5. The TransportQmss static and shared libraries will be copied directly into the Linux devkit's /usr/lib folder as long as the devkit install path was setup correctly prior to running the armv7setupenv.sh script
6. The test executables will be generated in the <base_repo_path>/ipc-transport/bin/<k2 device>/test/ folder. Only the device specified in the armv7setupenv.sh will be built.
   Note: Setting the USEDYNAMIC_LIB environment variable to "yes" will generate the shared library test executables
   $ export USEDYNAMIC_LIB=yes

ARMv7 Linux TransportQnss Configuration Parameters

Following are the configuration parameters for TransportQnss instance creation. Descriptions, default values, and programming considerations are provided for each configuration parameter. Each parameter is an element of the TransportQmss_Params structure. A structure of this type must be created, populated, and passed to the TransportQmss_create() function via pointer. The structure should be initialized to its default values using the TransportQmss_Params_init() function prior to population with application specific parameters.

```
TransportQmss_Params  qmss_trans_params;

...
TransportQmss_Params_init(&qmss_trans_params);
snprintf(mpm_inst_name, MPM_INST_NAME_LEN, "arm-qmss-generic");
qmss_trans_params.mpm_trans_inst_name = mpm_inst_name;
qmss_trans_params.rm_service_h        = ...;
...
qmss_handle = TransportQmss_create(&qmss_trans_params);
```

Configuration Parameter Element	Description	Initial Value	Special Considerations
`Char *mpm_trans_inst_name;`	MPM-Transport instance name. This string must match a "slaves" name string defined within the mpm_config.json file used in the Linux filesystem	NULL	This string name must match the generic QMSS instance name in the filesystem's /etc/mpm/mpm_config.json. The MPM JSON file's generic QMSS string is "arm=qmss-generic".
`Void *rm_service_h;`	RM instance service handle needed by MPM-Transport to request hardware resources	NULL	
`Int rx_msg_size_bytes;`	Maximum size in bytes of messages that will be received by this transport. Used to allocated MessageQ messages for reception	0	The value specified must be in sync with the "sizebuf" values specified in the "qmss-queue-map" transmit and receive free queue nodes of the mpm_config.json configuration file.
`Int mpm_trans_init_qmss;`	Controls whether MPM-Transport will initialize the QMSS hardware 0 - MPM-Transport will not initialize QMSS. Set if another entity within the system initialized the QMSS hardware. 1 - MPM-Transport will initialize QMSS. Set if this is the first system entity being created that used QMSS and QMSS has not been initialized	0	

MPM Transport Configuration Effects on ARMv7 Linux TransportQmss

TransportQmss assumes MPM Transport will manage configuration of the QMSS and CPPI LLDs. As a result, descriptor and descriptor buffer management is pushed to MPM Transport in the ARMv7 Linux version of TransportQmss. The MPM Transport JSON configuration file should be modified in order to change QMSS descriptor and buffer related parameters.

The MPM Transport JSON configuration file is located in the Linux file system at /etc/mpm/mpm_config.json

Adding the ARMv7 Linux TransportQmss to a User-Space Application

A TransportQmss instance can be created after the following:

- IPC has started
- A RM client instance has been created. Communication between the RM Client and the RM Server will take place over a Linux socket.
- ARMv7 Linux TransportQmss instances can be created once all previous requirements are satisfied. A TransportQmss instance can be created per user-space process. The number of TransportQmss instances that can exist simultaneously is limited to the number of QMSS queue pend queues available for use outside the kernel. The following example code comes from the TransportQmss multi-process test:

```
    TransportQmss_Params_init(&qm_trans_params);
    snprintf(mpm_inst_name, MPM_INST_NAME_LEN, "arm-qmss-generic");
    qm_trans_params.mpm_trans_inst_name = mpm_inst_name;
    qm_trans_params.rm_service_h         = rm_client_service_handle;
    qm_trans_params.rx_msg_size_bytes    = MAX_PACKET_SIZE;
    qm_trans_params.mpm_trans_init_qmss = 1;
    printf("Process %d : Creating TransportQmss instance\n",
local_process);
    qmss_trans_h = TransportQmss_create(&qm_trans_params);
    if (!qmss_trans_h) {
        printf("ERROR Process %d : Failed to create TransportQmss
handle\n",
               local_process);
        status = -1;
        goto err_exit;
    }

    /* Register transport with MessageQ as network transport */
    net_trans_h = TransportQmss_upCast(qmss_trans_h);
    base_trans_h = INetworkTransport_upCast(net_trans_h);
    if (MessageQ_registerTransportId(TRANS_QMSS_NET_ID, base_trans_h) < 0) {
        printf("ERROR Process %d : "
               "Failed to register TransportQmss as network
transport\n",
               local_process);
        status = -1;
        goto err_exit;
    }
```

- Use the standard MessageQ_create() and MessageQ_open() APIs to create a local process MessageQ and open a remote process MessageQ, respectively. The LAD daemon's NameServer will store created queues and respond to queue open queries.

```
    /* Create a MessageQ */
    snprintf(msg_q_name, MSGQ_Q_NAME_LEN, "Process_%d_MsgQ",
local_process);
    MessageQ_Params_init(&msg_params);
    msg_q_h = MessageQ_create(msg_q_name, &msg_params);
    if (msg_q_h == NULL) {
        printf("ERROR Process %d : Failed to create MessageQ\n",
local_process);
        status = -1;
        goto err_exit;
    }
    printf("Process %d : Local MessageQ: %s, QId: 0x%x\n",
local_process,
           msg_q_name, MessageQ_getQueueId(msg_q_h));

    /* Open next process's MessageQ */
    snprintf(remote_q_name, MSGQ_Q_NAME_LEN, "Process_%d_MsgQ",
next_process);
    printf ("Process %d : Attempting to open remote queue: %s\n",
            local_process, remote_q_name);
    do {
        status = MessageQ_open(remote_q_name, &remote_q_id);
        sleep(1);
    } while ((status == MessageQ_E_NOTFOUND) || (status ==
MessageQ_E_TIMEOUT));
    if (status < 0) {
        printf("ERROR Process %d : Error %d when opening next process
MsgQ\n",
               local_process, status);
        status = -1;
        goto err_exit;
    } else {
        printf("Process %d : Opened Remote queue: %s, QId: 0x%x\n",
               local_process, remote_q_name, remote_q_id);
    }
```

- MessageQ_put() and MessageQ_get() can be used normally to send and receive messages between processes using TransportQmss. The only caveat is that messages that are to use TransportQmss must be sent with the transport ID, TID, that was used to register the TransportQmss instance with the MessageQ Network interface.

```
/* TransportQmss MessageQ Network Interface TID */
#define TRANS_QMSS_NET_ID 4

typedef struct {
    MessageQ_MsgHeader header; /* 32 bytes */
    int32_t                src;
    int32_t                flags;
    int32_t                numMsgs;
    int32_t                seqNum;
    uint32_t               data[TEST_MSG_DATA_LEN_WORDS];
    uint8_t                pad[16]; /* Pad to cache line size of 128 bytes
*/
} test_msg_t;

...

test_msg_t *msg = NULL;

msg = (test_msg_t *) MessageQ_alloc(0, sizeof(*msg));
if (msg == NULL) {
    printf("ERROR Process %d : MessageQ_alloc failed\n",
           local_process);
    goto err_exit;
}

...

/* Set the transport ID to route message through TransportQmss instance
```

```
  */
MessageQ_setTransportId(msg, TRANS_QMSS_NET_ID);
status = MessageQ_put(remote_q, (MessageQ_Msg)msg);
if (status < 0) {
    printf("ERROR Process %d : MessageQ_put failed\n",
           local_process);
    goto err_exit;
}
```

ARMv7 Linux TransportQmss Tests

ARMv7 Linux TransportQmss includes a single test that uses TransportQmss to send MessageQ messages between Linux user-space processes. The MessageQ interface will be used to send a message, round-robin, between a configurable number of Linux processes. Each process will be running an exclusive TransportQmss instance. Data integrity of the message will be checked after each process to process transfer. The default number of processes and round-robin iterations is 4, and 100, respectively.

Building the Multi-Process Test

The Multi-Process user-space application can be built through Yocto/bitbake or through downloading the keystone-linux/ipc-transport GIT repository. For instructions on how to do either please see UG section on ARMv7 Linux TransportQmss Source Delivery [7]

IPC currently has the restriction that a DSP image that starts IPC on the DSP must be downloaded in order to run IPC on the Linux Host. The Multi-Process test doesn't interact with the DSP. A minimal IPC startup DSP application is provided in the MCSDK BIOS PDK for use with any user-space example or test application that runs IPC but doesn't interact with the DSPs.

The minimal IPC startup DSP application is called transportIpcStartupK2XUtilProject. The source code is located in <install_base>/pdk_keystone2_3_01_03_06/packages/ti/transport/ipc/c66/example/util/. The project can be imported into CCS from <install_base>/pdk_keystone2_3_01_03_06/packages/exampleProjects folder. Build the project and copy the generated .out to the Linux file system.

Running the Multi-Process Test

Setup is needed prior to running the test.

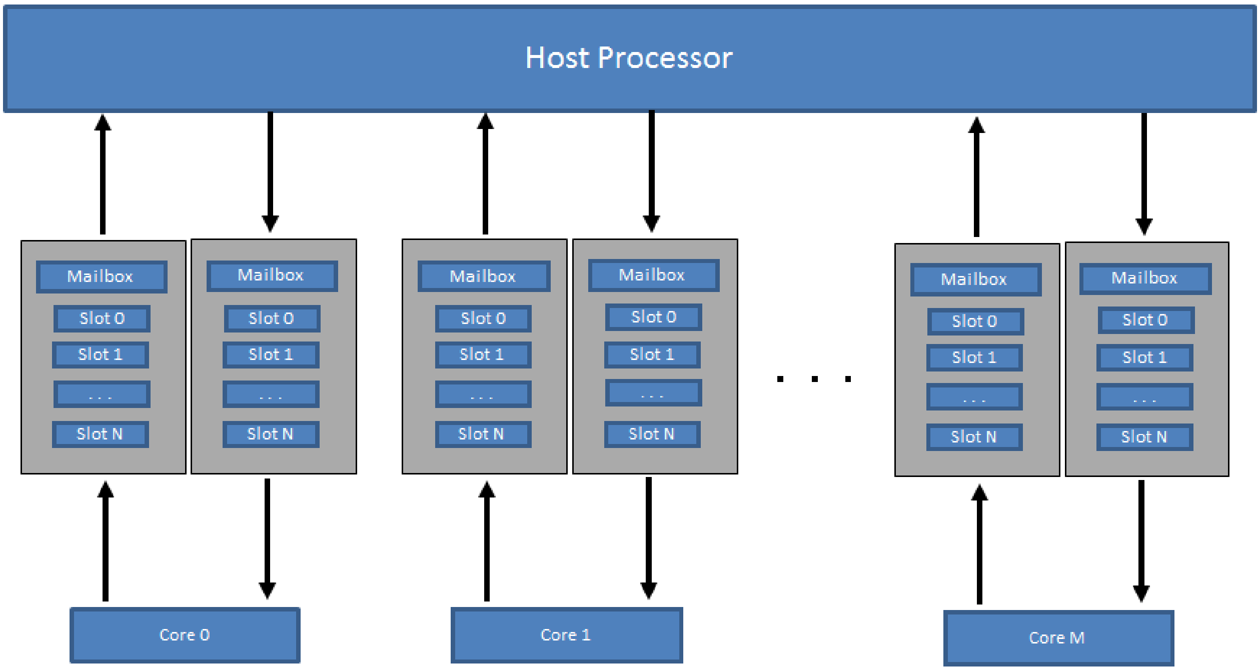
Note: Replace "K2X" and "k2x" with the proper device

1. Load the CMEM module
   $ insmod /lib/modules/3.10.61/extra/cmemk.ko
2. Download and run the IPC DSP minimal startup applcation
   $ mpmcl load dsp0 transportIpcStartupK2XUtilProject.out
   $ mpmcl run dsp0
3. Start the RM Server
   $ rmServer.out /usr/bin/device/k2x/global-resource-list.dtb /usr/bin/device/k2x/policy_dsp_arm.dtb

The Multi-Process test executable can be run once setup is complete.

MPM Mailbox

Mailbox is used for exchanging control messages between the host and individual DSP cores. As shown in the picture below, a mailbox is uni-directional, either host->DSP or DSP->host. Mailboxes are identified by a unique integer value which is returned after a mailbox is created on the host and opened on the DSP core. There exists a maximum of 2 mailboxes per DSP core (1 mailbox for Host -> DSP messages and 1 mailbox for DSP -> Host messages). Each mailbox has configurable amount of memory space to allocate slots which store a pending message.



An empty Mailbox slot must be allocated prior to sending a message. Receiving a message frees a slot and marks it as being empty. Mailboxes can be queried to obtain the number of unread messages within the mailbox.

Here is the high-level API for this module:

- **mpm_mailbox_create** is called by the host and dsp to create a mailbox at the location specified
- **mpm_mailbox_open** is called by both Host and DSP core to open the mailbox
- **mpm_mailbox_write** is a non-blocking call, writes a message to the mailbox. Returns with error message indicating full, if all the mailbox slots are occupied
- **mpm_mailbox_read** is a non-blocking call. A message is picked up and returned to the application when available. Returns error message indicating empty, if there are no messages to be processed.
- **mpm_mailbox_query** obtains the number of unread messages in the mailbox

MPM Sync

Sync Module implements support for Multicore Barriers and Locks.

Barrier

Step-1) Number of participants in a barrier need to be defined up front, and using the below API, the memory size required for that many participants can be obtained.

```
int32_t mpm_sync_barr_get_sizes(int32_t num_users)
```

Step-2) One of the participants has to invoke a barrier init. It is not harmful to invoke this by some or all the participants. If a barrier is already initialized, subsequent calls (by other cores) simply returns.

```
mpm_sync_barr_init(void *barr, int32_t num_users)
```

Step-3) Participants call mpm_sync_barr_wait function to busy wait until all the participants arrive at that location.

```
mpm_sync_barr_wait(void *barr)
```

Lock

Lock module implements Lamport’s bakery algorithm using shared memory. It is a multi-process safe lock.

Maximum number of participants in a lock need to be determined up front for memory allocation reasons. The required memory can be queried using mpm_sync_lock_get_sizes() API.

Here is the high-level API for this module:

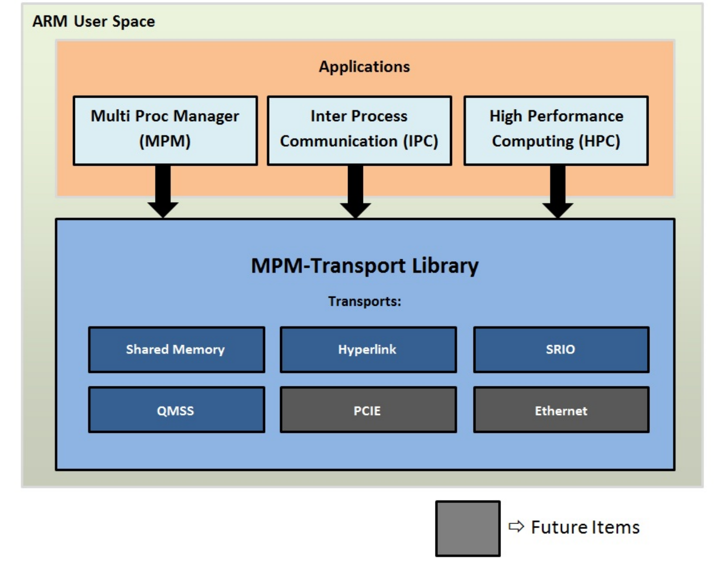
- **mpm_sync_lock_init**() needs to be called once to initialize the lock.
- **mpm_sync_lock_check**() is a non-blocking call to check if lock is in use.
- **mpm_sync_lock_acquire**() is a blocking call to acquire the lock and
- **mpm_sync_lock_release**() releases the lock.

MPM Transport

The MPM transport is designed to provide access to memory associated with remote cores/nodes. The current supported transport modes are: *shared memory transport* and *Hyperlink transport*.

Following are some implementation details:

- The API's to access transport and its static library are provided in *linux-devkit*.
- The APIs can be reviewed from mpm_transport.h [8].
- The parameters of transport can be configured from the JSON config file mpm_config.json [9]. Currently the MPM downloader shares same config file, it is likely to change in future.
- The static library to be linked for the transport is *libmpmtransport.a*, which is in linux-devkit provided in the release. The link option should be *-lmpmtransport*.
- An example application of the transport component is provided in test [10].



MPM Transport

Release History

Refer to following table for release history:

MCSDK Version	MPM-Transport Version	Major Updates / Features Release
3.0.1	1.0.0	Initial MPM-Transport release. Shared memory functionality.
3.0.2	1.0.0	No major changes
3.0.3	1.0.1	Hyperlink support for transfers. JSON to add new slaves and additional array segment for hyperlink peripheral parameters. Leverages hyperlink user mode driver.
3.0.4	1.0.4	Added support for EDMA3 and 36-bit addresses. Intersect with MCSDK-HPC developments.
3.1.0, 3.1.1	1.0.5	Multiple optimizations for hyperlink, such as linked transfers, parsing DTB for params, K2E support
3.1.2	1.0.6	Added hyperlink interrupt support
3.1.3	1.0.7	Added QMSS and SRIO support

Migration from MCSDK 3.1.1 to MCSDK 3.1.3

The default examples using hyperlink are changed to serdes_init = 0, which means that you need to enable the serdes externally before running the examples. You can do this by using mpmcl:

```
user@k2hk-evm> mpmcl transport <slave name> open
```

Additionally, hyperlink and SRIO LLD symbols are now declared as weak to accommodate devices without these peripherals. If you are linking with the static MPM Transport library (libmpmtransport.a), you will have to link with --whole-archive so that the user space LLDs will override the weak symbols. No changes if you use the shared library (libmpmtransport.so).

Building the Library

1) Clone the git repository for the source code:

```
git clone git://git.ti.com/keystone-linux/mpm-transport.git
cd mpm-transport
```

2) Set up your environment

```
export BUILD_LOCAL=true
export PATH="$PATH:<your_linaro_toolchain>/bin"
```

3) Source the general MCSDK environment setup file (used also for building other LLDs)

```
cd <your_mcsdk_install>/pdk_keystone2_3_xx_xx_xx/packages
source armv7setupenv.sh
cd -
```

Note: armv7setupenv.sh is located in the pdk_keystone2_3_xx_xx_xx folder that came with MCSDK installation. Your setup should have linux-devkit installed and use the resource in there for development. **Modify** armv7setupenv.sh according to your system as necessary

4) Run make:

```
make clean
make
```

Modifying and Rebuilding the Library for Profiling

MPM-Transport provides some internal time profiling for its provided API. Prior to running make (to rebuild the library as the steps state above), modify the file mpm_transport_time_profile.h to enable the areas you want to profile. Edit the mpm_transport_time_profile.h file:

vi <mpm_transport>/src/utils/time_profile/mpm_transport_time_profile.h

You should see

```
#define TIME_PROFILING 0
#if TIME_PROFILING
#define TIME_PROFILE_HYPLNK_PUT_INITIATE                  0
#define TIME_PROFILE_HYPLNK_GET_INITIATE                  0
#define TIME_PROFILE_HYPLNK_PUT_INITIATE_LINKED             0
#define TIME_PROFILE_HYPLNK_GET_INITIATE_LINKED             0
...
...
```

- To enable TIME_PROFILING, change "#define TIME_PROFILING 0" to "#define TIME_PROFILING 1"
- Change the define to 1 for the function you want to profile, or vice versa, to 0 for regular usage
- The variables are named according to the functions it targets. For example, "TIME_PROFILE_HYPLNK_PUT_INITIATE" refers to the mpm_transport_put_initiate() function that is using Hyperlink transport
- Please account for nested timestamps. For example, "TIME_PROFILE_HYPLNK_GET_WINDOW" profiles the portion where it maps hyperlink to a remote address, which is within any of the read/write operations. Enabling this will also skew the results for those read/write APIs (such as "TIME_PROFILE_HYPLNK_PUT_INITIATE"

Building Examples with SerDes Bypass

In the transport modes that uses SerDes, examples may need to explicitly skip the SerDes initialization portion if it is already handled by another process. Motivation to do so would be to minimize risk of breaking the SerDes link or to use the given setup as-is. To do so, set the serdes_init variable in the open structure to 0 before calling mpm_transport_open()

```
   ...
   mpm_transport_open_t ocfg = {
       .open_mode      = (O_SYNC|O_RDWR),
       .msec_timeout = 5000,
       .serdes_init = 0,
   };
   ...
   h = mpm_transport_open("arm-remote-hyplnk-0", &ocfg);
   ...
```

In this case, "arm-remote-hyplnk-0" is the slave name to be opened and the slave peripheral specification can be found in the JSON file. Since serdes_init is 0, a separate process should manage opening and closing the transport associated with the slave. MPM has the capability to do so by: "mpmcl transport arm-remote-hyplnk-0 open" and "mpmcl transport arm-remote-hyplnk-0 close", respectively.

Setting Transport Parameters from JSON file

The JSON configuration file for MPM-Transport is mpm_config.json [9]. There are four major arrays in this file that you should be aware of:

1) segments - The target destination address has be within the range specified in a memory segment.

name - the name of the segment

localaddr - address from a local point of view. This is *NOT* needed for transport modes outside of sharedmem because peripherals see the globaladdr instead.

globaladdr - address from the soc point of view

length - how long the segment is

devicename - /dev/* to open. This only applies to sharedmem. All other transport modes will have their respective transport device.

2) slaves - The top-level profile that includes all the information about the transport you want to use

name - name of the slave

transport - points to a profile in the "transports" array. Exception is "sharedmem", which does not need a profile

dma - points to a profile in the "dma-params" array.

memorymap - points to profiles in the "segments" array. This entry is an array of segment names.

crashcallback - points to the script for crash callback. This is optional for mpm-transport, but is used by mpmcl.

3) transports - Individual transport configurations for the transport mode being used

name - name of the transport profile

transporttype - the transport mode, eg. hyperlink, pcie, srio

Note: peripheral and transport parameters will vary based on the transport used. See peripheral-related section for mpm-transport

4) dma-params - Configuration for using EDMA3

name - name of the DMA profile

edma3inst - the EDMA3 instance number to use (0-4, total of 5)

shadowregion - the shadowregion of the EDMA3 instance to use (0-7, total of 8)

5) mpax-params - Configuration for using keystonemmap library

name - name of the profile

base - base address to use for creating logical addresses (32-bit addresses aliased to specific 36-bit addresses)

size - length of the space allow, from the base, to do the logical mapping

index - entry number in the MPAX table

6) qmss-mem-regions - parameters to figure a QMSS memory region

name - name of the memory region

regiondescnum - number of descriptors this region will have

regiondescsize - size per descriptor

regiondescflag - to be used for manageDescFlag

regionnum - the region number. Use -1 for next available

regionstartidx - the index number to offset from

7) qmss-queue-map - parameters to setup a queue

name - the name of the queue

queue - the queue number. Use -1 for next available

qtype - to be used for the queue type to request

numdesc - number of descriptors to init for this queue

sizebuf - size of the buffer for each descriptor. Will be allocated with CMEM.

Transport over Hyperlink

Hyperlink can be configured point-to-point or loopback. There are two Hyperlink ports in Keystone 2. Please refer to the Hyperlink user guide under your SOC's product page for more detailed explanation of the Hyperlink settings and how they are used.

Pre-requisites

- MCSDK 3.0.3 or higher. The Hyperlink user mode driver and associated hyplnk_device.c file will be available in the linux-devkit.
- libhyplnk*.so* - you will need the shared libraries at runtime. Please ensure that your filesystem has the Hyperlink libs under /usr/lib

JSON Transport Profile

Fields needed for a Hyperlink transport profile in the JSON file:

- name - name of the profile
- transporttype - transport mode. valid value = hyperlink (for using Hyperlink)
- direction - valid values = "loopback" for loopback mode, "remote" for sending outside of SOC
- hyplnkinterface - Hyperlink instance to use. Valid values = "hyplnk0" to use port 0, "hyplnk1" to use port 1
- txprivid - transmit privid overlay field
- rxprivid - receiving side's privid select field
- rxsegsel - receiving side's segment select field
- rxlenval - Length of the Hyperlink segment
- lanerate - Valid values = full, half, quarter
- numlanes - Valid values = 4, 1, 0

Example:

```
{
  "name": "hyplnk0-loopback",
  "transporttype": "hyperlink",
  "direction": "loopback",
  "hyplnkinterface": "hyplnk0",
  "txprivid": 0,
  "rxprivid": 0,
  "rxsegsel": 6,
  "rxlenval": 21,
  "numlanes": "4"
},
```

Compiling and Linking with Hyperlink Libraries

- If you are compiling with the dynamic MPM Transport library, your application will link in the needed user space LLDs at runtime. Please make sure that libhyplnk_device.so.1 exists in your linker path and points to the appropriate device-specific library in your filesystem.
- If you are compiling with static libraries, you will need to link in the Hyperlink library for your application. Please use --whole-archive, -lhyplnk_<device>, and -L${DEVKIT_USR_LIB} (DEVKIT_USR_LIB is your devkit's user library directory containing the needed libs).

Transport over QMSS

QMSS transport is set up to push and pop descriptors from queues. This is a cornerstone for many of the packet-based transports that is used in the system.

Pre-requisites

- MCSDK 3.1.3 or higher. The Hyperlink user mode driver and associated hyplnk_device.c file will be available in the linux-devkit.
- libqmss*.so* and libcppi*.so* - you will need these shared libraries at runtime. Please ensure that your filesystem has them under /usr/lib.
- TI CMEM - the cmem module will be needed to allocate continuous memory.

JSON Transport Profile

Fields needed:

- name - name of the profile
- transporttype - needs to be "qmss" to signify QMSS transport
- qmssmaxdesc - max number of descriptors to initialize QMSS with. This is done once per process
- initregion - looks for the corresponding qmss-mem-region for params to setup a memory region
- txfreeq - looks for the corresponding qmss-queue-map to setup a TX queue
- rxfreeq - looks for the corresponding qmss-queue-map to setup a RX queue
- writefifodepth - FIFO depth to setup CPPI with
- cpdmatimeout - timeout value for cpdma to setup CPPI with

Initializing QMSS

- Setup Resource Manager - applications will need to initialize the RM driver and pass in the RM client handle for QMSS and CPPI LLDs to use:

```
mpm_transport_open_t ocfg;
ocfg.rm_info.rm_client_handle        = rmClientServiceHandle;
```

QMSS Send and Receive

For receiving: int mpm_transport_packet_recv(mpm_transport_h h, char **buf, int *len, mpm_transport_recv_t *cfg);

- buf - a pointer to a buffer of space to copy content to
- len - length to copy
- cfg - receive options, will be used to specified memcpy or direct linking of buffer

When you open a QMSS handle, you will set up your MPM Transport handle to listen in on a specific RX flow. You can get this flow ID from the mpm_transport_open_t structure that you opened with.

```
mpm_transport_open_t ocfg;
mpm_transport_open("arm-qmss-generic", &ocfg);
flow_id = ocfg.transport_info.qmss.flowId;
```

For sending: int mpm_transport_packet_send(mpm_transport_h h, char **buf, int *len, mpm_transport_packet_addr_t *addr_info, mpm_transport_send_t *cfg);

- buf - a pointer to a buffer of space to copy content from
- len - length to copy
- addr_info - the address to send to
- cfg - send options, will be used to specified memcpy or direct linking of buffer

To send a packet in QMSS, addr_info needs to be specified with packet_addr_type_QMSS and the flow ID to send to.

```
mpm_transport_packet_addr_t send_addr;
send_addr.addr_type = packet_addr_type_QMSS;
send_addr.addr.qmss.flowId = 16;
```

Compiling and Linking with QMSS Libraries

- If you are compiling with the dynamic MPM Transport library, your application will link in the needed user space LLDs at runtime. Please make sure that libqmss_device.so.1 and libcppi_device.so.1 exists in your linker path and points to the appropriate device-specific library in your filesystem.
- If you are compiling with static libraries, you will need to link in the needed peripheral LLDs for your application. Please use --whole-archive, -lqmss_<device>, -lcppi_<device> and -L${DEVKIT_USR_LIB} (DEVKIT_USR_LIB is your devkit's user library directory containing the needed libs).

Transport over SRIO

SRIO is a high speed transport that can transfer data between multiple SOCs or loopback to the same device. This builds on top of the QMSS packet-based transport and abstracts the SRIO user space driver for ease of use.

Pre-requisites

- MCSDK 3.1.3 or higher. The Hyperlink user mode driver and associated hyplnk_device.c file will be available in the linux-devkit.
- libqmss*.so*, libcppi*.so*, and libsrio*.so* - you will need these shared libraries at runtime. Please ensure that your filesystem has them under /usr/lib.
- TI CMEM - the cmem module will be needed to allocate continuous memory.

JSON Transport Profile

Fields needed:

- name - name of the profile
- transporttype - needs to be "srio" to signify SRIO transport
- qmssmaxdesc - max number of descriptors to initialize QMSS with. This is done once per process
- initregion - looks for the corresponding qmss-mem-region for params to setup a memory region
- txfreeq - looks for the corresponding qmss-queue-map to setup a TX queue
- rxfreeq - looks for the corresponding qmss-queue-map to setup a RX queue

Initializing SRIO

- Setup Resource Manager - applications will need to initialize the RM driver and pass in the RM client handle for QMSS and CPPI LLDs to use:

```
mpm_transport_open_t ocfg;
ocfg.rm_info.rm_client_handle         = rmClientServiceHandle;
```

- SRIO specific initialization - applications will need to tell MPM Transport the handle's SRIO address information. Example for type 11:

```
ocfg.transport_info.srio.type              =
packet_addr_type_SRIO_TYPE11;
ocfg.transport_info.srio.type11.tt         = 1;
ocfg.transport_info.srio.type11.id         = coreDeviceID[coreNum];
ocfg.transport_info.srio.type11.letter     = 2;
ocfg.transport_info.srio.type11.mailbox    = 3;
ocfg.transport_info.srio.type11.segmap     = 0x0;
```

- SRIO device initialization - applications will to provide MPM Transport a function to initialize/deinitialize the SRIO device:

```
ocfg.transport_info.srio.deviceInit   = &mySrioDevice_init;
ocfg.transport_info.srio.initCfg      = NULL; //param for device init
function if needed
ocfg.transport_info.srio.deviceDeinit = &mySrioDevice_deinit;
ocfg.transport_info.srio.deinitCfg    = NULL; //param for device deinit
 function if needed
```

SRIO Send and Receive

For receiving: int mpm_transport_packet_recv(mpm_transport_h h, char **buf, int *len, mpm_transport_recv_t *cfg);

- buf - a pointer to a buffer of space to copy content to
- len - length to copy
- cfg - receive options, will be used to specified memcpy or direct linking of buffer

When you open a SRIO handle, it will internally open a SRIO socket handle to listen in on.

For sending: int mpm_transport_packet_send(mpm_transport_h h, char **buf, int *len, mpm_transport_packet_addr_t *addr_info, mpm_transport_send_t *cfg);

- buf - a pointer to a buffer of space to copy content from
- len - length to copy
- addr_info - the address to send to
- cfg - send options, will be used to specified memcpy or direct linking of buffer

To send a packet in SRIO, addr_info needs to be specified with packet_addr_type_SRIO_TYPE9 or packet_addr_type_SRIO_TYPE11, and the SRIO information associated with that type. For example, in type 11:

```
mpm_transport_packet_addr_t send_addr;
send_addr.addr_type = packet_addr_type_SRIO_TYPE11;
send_addr.addr.srio.type11.tt = 1;
send_addr.addr.srio.type11.id = 0xABCD;
send_addr.addr.srio.type11.letter = 2;
send_addr.addr.srio.type11.mailbox = 3;
```

Compiling and Linking with SRIO Libraries

- If you are compiling with the dynamic MPM Transport library, your application will link in the needed user space LLDs at runtime. Please make sure that libqmss_device.so.1, libcppi_device.so.1, and libsrio.so.1 exists in your linker path and points to the appropriate device-specific library in your filesystem.
- If you are compiling with static libraries, you will need to link in the needed peripheral LLDs for your application. Please use --whole-archive, -lqmss_<device>, -lcppi_<device>, -lsrio_<device> and -L${DEVKIT_USR_LIB} (DEVKIT_USR_LIB is your devkit's user library directory containing the needed libs).

Using EDMA3

Using the Enhanced Direct Memory Access controller will provide performance increase in transporting large chunks of data over using memcpy.

- currently only supports hyperlink transport mode

Pre-requisites

- MCSDK 3.0.4 or higher. The EDMA3 user mode driver and associated evmTCI6636K2HSample.c file will be available in the linux-devkit.
- libedma3.so* and libedma3rm.so* - you will need the shared libraries at runtime. Please ensure that your filesystem has the EDMA3 libs under /usr/lib

MPM-Transport Functions/API to use EDMA3

The tranport functions are named get/put to signify direction of transfer, and "initiate" to signify it is a call to the EDMA3 controller to start a transfer. All transfer calls has a parameter input, *boolean is_blocking*, to determine whether EDMA3 should wait for completion before continuing. If this parameter is false, mpm_transport_transfer_check() must be called to check for transfer completion and clear EDMA3 flags. All transfer API returns a transfer handle, mpm_transport_trans_h, for mpm_transport_transfer_check() to check.

Note that all from_addr and to_addr are physical addresses.

- **mpm_transport_get_initiate()** - Get length data from remote from_addr and store it in local to_addr
- **mpm_transport_put_initiate()** - Put length data from local from_addr to remote destination to_addr
- **mpm_transport_get_initiate_linked()** - Same as mpm_transport_get_initiate(), except that this API accepts arrays of to_addr, from_addr, and length and complete all transfer with a single call. Parameter num_links must specify number of linked transfer and should equal the size of the three aforementioned arrays.
- **mpm_transport_put_initiate_linked()** - Same as mpm_transport_put_initiate(), except that this API accepts arrays of to_addr, from_addr, and length and complete all transfer with a single call. Parameter num_links must specify number of linked transfer and should equal the size of the three aforementioned arrays.

Compiling and Linking with EDMA3 Libraries

- You will need to compile and link in your device's evm[DEVICE].c file. This is located in `linux-devkit/sysroots/armv7ahf-vfp-neon-oe-linux-gnueabi/usr/include/ti/sdo/edma3/drv/s`
- You will need to link in the EDMA3 drv and rm libraries for your application. Please use -ledma3 and -ledma3rm with -L${DEVKIT_USR_LIB}, where DEVKIT_USR_LIB is your devkit's user library directory containing the needed libs.

Using libkeystonemmap

Using the Keystone MMAP driver will allow you to access the 36-bit physical memory space.

- currently only supports hyperlink transport mode

Pre-requisites

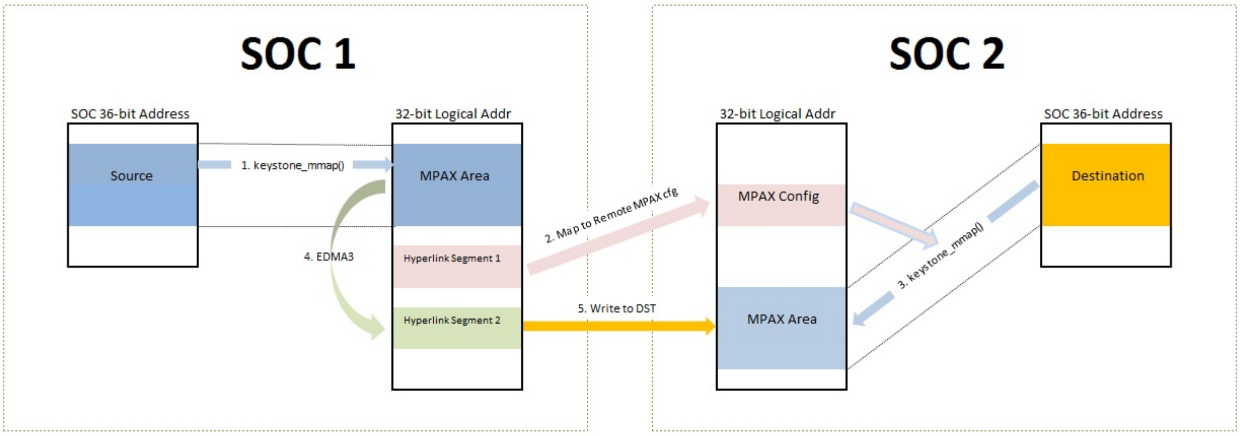
- MCSDK 3.0.4 or higher. The libkeystonemmap user mode driver and headers will be available in the linux-devkit
- libkeystonemmap.so* - you will need the shared libraries at runtime. Please ensure that your filesystem has the libkeystonemmap libs under /usr/lib
- All EDMA3 pre-requisites

MPM-Transport Functions/API to use libkeystonemmap

The 36-bit address space is accessed by modifying an entry in the MPAX table. Access to the newly mapped 32-bit address space will be done by the EDMA3 peripheral. Thus, the EDMA3 driver and requirements are needed, and the API provided are forms of "put/get_initiate()"

- **mpm_transport_get_initiate64()** - Same as mpm_transport_get_initiate(), except from_addr and to_addr are 64-bit values (uses MPAX to access 36-bit SOC addresses)
- **mpm_transport_put_initiate64()** - Same as mpm_transport_put_initiate(), except from_addr and to_addr are 64-bit values (uses MPAX to access 36-bit SOC addresses)

How the mpm_transport_put_initiate64() API Works for Hyperlink



1. keystone_mmap() is used to translate the local 36-bit source address to a local 32-bit location. The 32-bit location that is used is specified by the MPAX entry in the JSON file.
2. A hyperlink segment will be created from SOC1 to SOC2's MPAX registers
3. SOC1 calls keystone_mmap() again, but with Hyperlink Segement 1 as the base address. This allows SOC1 to update SOC2's MPAX entries remotely, to translate the 36-bit destination address to a 32-bit logical address in SOC2.

4. A second hyperlink segment will be created to map to the 32-bit destination address in SOC2. EDMA3 can then take the source data from the 32-bit local address (via step 1) and put it into Hyperlink Segment 2
5. When data hits Hyperlink Segment 2, it will be written to the remote 32-bit logical address (via step 3). This has the same effect as writing to the remote 36-bit destination address.

Compiling and Linking with libkeystonemmap Libraries

You will need to link in the libkeystonemmap library for your application. Please use -lkeystonemmap with -L${DEVKIT_USR_LIB}, where DEVKIT_USR_LIB is your devkit's user library directory containing the needed libs.

MCSDK UG Chapter Developing System Mgmt

WARNING: Article could not be rendered - ouputting plain text.
Potential causes of the problem are: (a) a bug in the pdf-writer software (b) problematic Mediawiki markup (c) table is too wide

MCSDK_UG_Chapter_DevelopingDeveloping with MCSDK: System ManagementLast updated: 09/20/2015Overview Learn how to manage resources on the device/EVM using the Resource Manager, download and manage DSP from ARM along with topics that have a global impact on your system. Acronyms The following acronyms are used throughout this chapter.AcronymMeaning API Application Programming Interface CD Resource Manager Client Delegate Instance DSP Digital Signal Processor DTB Device Tree Blob DTC Device Tree Compiler DTS Device Tree Source EVM Evaluation Module, hardware platform containing the Texas Instruments DSP FDT Flattened Device Tree GRL Resource Manager Global Resource List IPC Texas Instruments Inter-Processor Communication Development Kit LLD Low Level Driver MCSDK Texas Instruments Multi-Core Software Development Kit MPM Multiple Processor Manager OSAL Operating System Abstraction Layer PDK Texas Instruments Programmers Development Kit RM Resource Manager RTSC Eclipse Real-Time Software Components TI Texas Instruments Multiple Processor Manager The Multiple Processor Manager (MPM) module is used to load and run DSP images from ARM. Structure of MPM MPM slave node state transitions The MPM has two following major components MPM server (mpmsrv): It runs as a daemon and in default filesystem supplied in MCSDK it runs comes up automatically. It parses mpm config file from /etc/mpm/mpm_config.json, then waits on an UNIX domain socket. The mpm server runs/maintains a state machine for each slave core.MPM commandline/client utility (mpmcl): It is installed in the filesystem provides commandline access to the server. Following are the different methods, MPM can be used to load/run slave images Using mpmcl utility From config file, to load at bootup Writing an application to use mpmclient header file and library The mpm server/daemon logs go to "/var/log/daemon.log" or "/var/log/mpmsrv.log" based on "outputif" configuration in the JSON config file. The load command writes the slave image segments to memory using UIO interface. The run command runs the slave images. An example DSP image is provided in the MCSDK package at mcsdk_bios_#_##_##_##/examples/mpm directory. All events from the state transition diagram are available as options of mpmcl command, except for the crash event. The reset state powers down the slave nodes. Methods to load and run ELF images using MPM Using mpmcl utility to manage slave processors Use mpmcl --help to the details of the commands supported. Following is the output of the mpmcl help Usage: mpmcl <command> [slave name] [options] Multiproc manager CLI to manage slave processors <command> Commands for the slave processor Supported commands: ping, load, run, reset, status, coredump, transport load_withpreload, run_withpreload [slave name] Name of the slave processor as specified in MPM config file [options] In case of load, the option field need to have image file name Following is sample set of commands to manage slave processors. - Ping daemon if it is alivempmcl ping - Check status of dsp core 0mpmcl status dsp0 - Load dsp core 0 with an imagempmcl load dsp0 dsp-core0.out - Run dsp core 0mpmcl run dsp0 - Reset dsp core 0mpmcl reset dsp0 - Load dsp core 0 image with a preload imagempmcl load_withpreload dsp0 preload_image.out dsp-core0.out - Run dsp core 0 with preloadmpmcl run_withpreload dsp0NoteNote: In case of error, the mpm server takes the slave to error state. You need to run reset command to change back to idle state so that the slave can be loaded and run again.NoteNote: The status of slave core is idle means the slave core is not loaded as far as MPM is concerned. It does NOT mean the slave core is running idle instructions.Loading and running slave images at bootup The config file can load a command script to load and run slave cores at bootup. The path of the script is to be added in "cmdfile": "/etc/mpm/slave_cmds.txt" in the config file. Following is a sample command to load and run DSP images. dsp0 load ./dsp-core0.out dsp1 load ./dsp-core0.out dsp0 run dsp1 run Managing slave processors from application program An application can include mpmclient.h from MPM package and link to libmpmclient.a to load/run/reset slave cores. The mpmcl essentially is a wrapper around this library to provide commandline access for the functions from mpmclient.h.DSP Image Requirements For MPM to properly load and manage a DSP image, the following are required: The DSP image should be in ELF format. The MPM ELF loader loads those segments to DSP memory, whose PT_LOAD field is set. In order to skip loading of a particular section, set the type to NOLOAD in the command/cfg file. /* Section not to be loaded by remoteproc loader */ Program.sectMap[".noload_section"].type = "NOLOAD"; The default allowed memory ranges for DSP segments are as follows Start Address Length L2 Local 0x00800000 1MB L2 Global 0x[1-4]0800000 1MB MSMC 0x0C000000 6MB DDR3 0xA0000000 (512MB) The segment mapping can be changed using the mpm_config.json and Linux kernel device tree. Getting DSP prints(trace) output from ARM/Linux using MPM The DSP image needs to have an uncompressed section with name .resource_table. The MPM looks for this table in the DSP ELF image before loading it. The .resource_table can be used to provide SYS/BIOS information on trace buffer location and size.The .resource_table for must match to remoteproc resource table format used in IPC.Following steps shows how to create .resource_table section in the DSP image with trace buffer information. These code snippet is taken from MCSDK release package <mcsdk_bios_#_##_##_##>/examples/mpmIn the Configuro Script File of the application add following commands to create a section /* * The SysMin used here vs StdMin, as trace buffer address is required for * Linux trace debug driver, plus provides better performance. */ Program.global.sysMinBufSize = 0x8000; var System = xdc.useModule('xdc.runtime.System'); var SysMin = xdc.useModule('xdc.runtime.SysMin'); System.SupportProxy = SysMin; SysMin.bufSize = Program.global.sysMinBufSize; /* Configure resource table for trace only. Note that, it traceOnly parameter should not be set if application is using MessageQ based IPC to communicate between cores. */ var Resource = xdc.useModule('ti.ipc.remoteproc.Resource'); Resource.loadSegment = Program.platform.dataMemory; Resource.traceOnly = true; DSP trace/print messages from Linux The DSP log messages can be read from following debugfs locations DSP log entry for core #: /sys/kernel/debug/remoteproc/remoteproc#/trace0 Where # is the core id starting from 0. root@keystone-evm:~# cat /sys/kernel/debug/remoteproc/remoteproc0/trace0 Main started on core 1 .... root@keystone-evm:~# MPM configuration file The MPM configuration file is a JSON format configuration file. A sample config file can be reviewed from mpm_config.json. Usually it is installed in default root file system release as a part of MCSDK. The location is /etc/mpm. The MPM parser ignores any JSON elements which it does not recognize. This can be used to put comment in the config file. The tag cmdfile (which is commented as _cmdfile by default) loads and runs MPM commands at bootup. The tag outputif can be syslog, stderr or filename if it does not matches to any predefined string. By default the config file allows loading of DSP images to L2, MSMC and DDR. It can be changes to add more restrictions on loading or load to L1 sections. In current from MPM does not do MPAX mapping for local to global addresses. The MPM needs 1KB of scratch memory in either DDR or MSMC for its processing (running trampoline for get 10bit alignment and workaround for DSP reset issue). This is reserved at the end of DDR in the config file with keyword "scratchaddr" and "scratchlength". The DSP images must avoid to load to the scratch memory location. NoteNote: : In future, MPM is going to get scratch memory from kernel and the scratch memory

requirement will be removed.Crash event notification, coredump generation and processing The MPM can monitors crash events from DSPs.The DSP image needs to be instrumented to indicate MPM that it is crashed. Instrumenting DSP application The DSP application should install an exception hook for the DSP fault management APIs. Please follow the steps below to add the crash indication and add necessary segments for coredump. In the .cfg (Configuro Script) file of the application add following commands to create a section var devType = "k2?"; /* Replace k2? with the k2 device in use k2e, k2h, k2k, or k2l */ /* Load and use the Fault Management package */ var Fault_mgmt = xdc.useModule('ti.instrumentation.fault_mgmt.Settings') Fault_mgmt.deviceType = devType; /* Load the Exception and register a exception hook */ var Exception = xdc.useModule('ti.sysbios.family.c64p.Exception'); Exception.exceptionHook = '&myExceptionHook'; Exception.enablePrint = true; /* Add note section for coredump */ Program.sectMap[".note"] = new Program.SectionSpec(); Program.sectMap[".note"] = Program.platform.dataMemory; Program.sectMap[".note"].loadAlign = 128; In a source/header file, create a resource array as follows /* Fault Management Include File */ #include <ti/instrumentation/fault_mgmt/fault_mgmt.h> Void myExceptionHook(Void) { uint32_t i; Fm_HaltCfg haltCfg; uint32_t efr_val; /* Copy register status into fault management data region for Host */ Fault_Mgmt_getLastRegStatus(); memset(&haltCfg, 0, sizeof(haltCfg)); efr_val = CSL_chipReadEFR(); /* If triggered exception originates from another core through * NMI exception don't need to halt processing and notify other cores * since the parent core where the exception originally triggered via * event would notify them. This eliminates recursive exceptions */ if (!(efr_val & 0x80000000)) { /* Halt all processing - Only need to be done on one core */ haltCfg.haltAif = 1; haltCfg.haltCpdma = 1; #if EXCLUDE_LINUX_RESOURCES_FROM_HALT haltCfg.haltSGMII = 0; /* EDMA used by kernel to copy data to/from NAND in UBIFS */ haltCfg.haltEdma3 = 0; haltCfg.excludedResources = &linuxResources[0]; #else haltCfg.haltSGMII = 1; haltCfg.haltEdma3 = 1; haltCfg.excludedResources = NULL; #endif Fault_Mgmt_haltIoProcessing(&fmGblCfgParams, &haltCfg); for (i = 0; i < fmGblCfgParams.maxNumCores; i++) { /* Notify remote DSP cores of exception - WARNING: This will generate NMI * pulse to the remote DSP cores */ if (i != CSL_chipReadDNUM()) { Fault_Mgmt_notify_remote_core(i); } } } /* Notify Host of crash */ Fault_Mgmt_notify(); } A sample test application is provided in pdk_keystone2_#_##_##_##\packages\ti\instrumentation\fault_mgmt\testDetecting crash event in MPM In case of a DSP exception, the MPM calls the script provided in JSON config file. The MCSDK Linux filesystem has a sample script /etc/mpm/crash_callback.sh that sends message to syslog indicating which core crashed. This script can be customized to suit notification needs.Generating DSP coredump The DSP exceptions can be following Software-generated exceptions Internal/external exceptions Watchdog timer expiration The MPM creates an ELF formatted core dump. root@keystone-evm:~# mpmcl coredump dsp0 coredump.out The above command will generate coredump file with name coredump.out for the DSP core 0.NoteNote: The coredump can be captured from a running system which is not crashed, in this case the register information won't be available in the coredump.Converting and loading core dump image to CCS The current version of CCS (5.4) does not support ELF core dump. The dspcoreparse utility provided in mcsdk_linux_#_##_##_##/host-tools, is used to parse the ELF format coredump and generate the crash dump file that can be uploaded to CCS for further analysis. Copy the coredump file generated above to mcsdk_linux_#_##_##_##/host-tools/dspcoreparse and run the following command to get crash dump file for CCS.NoteNote: In CCS, a core dump is referred to as a "crash dump". user@dspcoreparse $ ./dspcoreparse -o coredump.txt coredump.out This utility can also be used to display the parsed content. However, CCS provides a graphical view of the information. Analyzing DSP crash in CCS The crash dump file can be loaded to CCS with the symbols of the DSP executable to see register content, stack, memory and other information. Please see Crash_Dump_Analysis for more information on loading and analyzing a CCS crashdump file. DSP_crash_stack.jpgFurther analysis of the crash can be done in CCS by opening ROV. Please see Runtime Object Viewer for more details on ROV. NoteNote: The scripting console of CCS sometimes does load the crashdump file with path other than base directory. As an workaround please following command to load the coredump file from CCS scripting console. activeDS.expression.evaluate('GEL_SystemRestoreState("/home/ubuntu/coredump.txt")') NoteNote: The coredump schemes will be changing in future releases.MPM error codes and debugging Following are some pointers for MPM debugging and error codes If MPM server crashed/exited/not running in the system, mpmcl ping will return failure If there is any load/run/reset failure MPM client provides error codes. The major error codes are given below error code error type -100 error_ssm_unexpected_event -101 error_ssm_invalid_event -102 error_invalid_name_length -103 error_file_open -104 error_image_load -105 error_uio -106 error_image_invalid_entry_address -107 error_resource_table_setting -108 error_error_no_entry_point -109 error_invalid_command The MPM daemon logs goes to /var/log/mpmsrv.log by default. This file can provide more information on the errors.Loading DSP images from CCS (without using MPM) By default, the DSP cores are powered down by u-boot at the time of EVM boot. After kernel is running, MPM can be used to load and run DSP images from Linux command-line/utility. Rather than using MPM, if you want to use CCS to load and run DSP images, then set the following setting in u-boot prompt: setenv debug_options 1 saveenv reset This will not power down DSPs at startup and CCS/JTAG can connect to the DSP for loading and debugging. This option is useful if you want to boot Linux on ARM and then use JTAG to manually load and run the DSPs. Otherwise you may see "held in reset" errors in CCS. NoteNote: The above step is not needed if you want to load DSP cores using MPM and subsequently use CCS to connect to DSP.Frequently Asked Questions Q: MPM does not load and run the DSP image A: There can be several scenarios, following are few of them The MPM server may not be running. The command mpmcl ping will timeout in this case. The mpm server is expected to be running in background to service the requests from mpm client. The log file /var/log/mpmsrv.log can provide more information. An issue can be the devices relevant to MPM /dev/dsp0, ... , /dev/dsp7, /dev/dspmem are not created. You need to check if these devices are present. If they are not present then check if the kernel and device tree have right patches for these devices. The log can print error codes provided in MPM error codes section. Another way to debug loading issues is, to run mpm server in non-daemon mode from one shell using command mpmsrv -n, before this you need to kill the server if it is running. (The command to kill is mpmsrv -k or you can choose to kill the process). Then from other shell run the client operations. Q: MPM fails to load the segments A: The MPM fundamentally copies segments from DSP image to memory using a custom UIO mmap interface. Each local or remote nodes (DSPs) are allocated some amount of resources using config file. The segments in the config file needs to be subset of memory resources present in kernel dts file. The system integrator can choose to add or change memory configurations as needed by application. In order to change the default behavior user need to changed in JSON config file and kernel device tree. In JSON configuration file, the segments section need to be updated. You need to make sure it does not overlap the scratch memory section. You might have to move the scratch section if the allocated DDR size is increased. And, in the kernel device tree the mem sections of dsp0, .. , dsp7, dspmem need to be updated. Sometimes few segments used by DSP may not accessible by ARM at the time of loading. These segment can cause load failure. So it is useful to understand the memory layout of your own application and if there are any such sections, you can skip loading those segments to memory using NOLOAD method described above. The MPM does not have MPAX support yet. So the MPAX support needs to be handled by application. If the linker adds a hole in the resource table section right before the actual resource_table due to the alignment restriction, then MPM as of now won't be able to skip the hole and might get stuck. In this case if you hex-dump resource table (method given below) size will be quite large (normally for a non-IPC case it is around 0xac). The workaround is to align the .resource_table section to 0x1000 using linker command file or some other method so that linker does not add any hole in the resource_table section. In future, MPM will take care of this offset. Q: MPM fails to run the image A: MPM takes DSP out of reset to run the image. So, the fails to run normally attributed to DSP is crashing before main or some other issue in the image. But, to debug such issue, after mpmcl run, use CCS to connect to the target and then do load symbols of the images. Then the DSP can be debugged using CCS. Another way to debug the run issue, is to aff a infinite while loop in the reset function so that the DSP stops at the very beginning. Then load and run the DSP using MPM and connect thru CCS, do load symbols and come out of while loop and debug. Q: I don't see DSP prints from debugfs A: Make sure you followed the procedure described above to include the resource table in the image. Care should be taken for the resource table not being compiled out by linker. To check if the resource table present in the image using command readelf --hex-dump=.resource_table

<image name>. It should have some non-zero data. Another point is, if you are loading same image in multiple cores and if the resource table and trace buffer segments overlap with each other in memory, then there can be undesirable effect. Q: I see the DSP state in /sys/kernel/debug/remoteproc/remoteproc0/state as crashed A: The file /sys/kernel/debug/remoteproc/remoteproc#/state does not indicate state of DSP when MPM is used for loading. The state of the DSP can be seen using MPM client. See the description of the command in #Methods_to_load_and_run_ELF_images_using_MPMMethods to load and run ELF images using MPM sections.Resource Manager The Resource Manager (RM) is delivered as part of PDK as a means for managing system resource contentions. RM provides the ability to allocate system resources to "entities" within a software architecture based on sets of allocation rules. The term "entities" can refer to anything from a device core, to an OS task or process, to a specific software module. The resources managed and the "entities" for which the resources are managed by RM are completely defined by the RM configuration parameters. The RM configuration parameters are device specific. Whereas, the RM source code is completely device independent. What follows is a description of the RM architecture and in-depth instruction on how to integrate, configure, and use RM. Architecture Resource Manager is an instance based architecture. Integrating RM into a system consists of creating a set of RM instances, connecting these instances via transport mechanisms, and then using the instances to request resources from different device cores, processes, tasks, modules, etc. Resource permissions are derived from RM instance names so it is imperative that if two system entities require different permissions they issue their service requests through different RM instances. There are no restrictions on where a RM instance can be created as long as the means exist to connect the instance to other RM instances within the system. There are three primary RM instance types: Server - Manages all defined system resources. Handles resource requests received from connected Client Delegates and Clients. Client Delegate (CD) - Manages resource provided by the RM Server. Handles resource requests received from connected Clients. Client - Connects and forwards resource requests to Server or CD for servicing Example multicore RM instance topology (not based on any specific device or system architecture): RM_inst_multi_core.jpg Example multi-task/process RM instance topology (not based on any specific device or system architecture): RM_inst_multi_task.jpg Example multi-DSP with multi-task/process RM instance topology (not based on any specific device or system architecture): RM_inst_multi_dsp.jpgGeneral Instance Interfaces All the RM instance types share a common set of APIs allowing them to receive resource requests and communicate on any device. Service API Transport API OS Abstraction Layer (OSAL) API RM_general_interfaces.jpgService API The RM service API defines what resource services RM provides to the system. All RM instances can be issued resource service requests. While all instances can receive requests, not all instances can process requests. Most service requests will be processed and validated on the RM Server instance. Due to the blocking nature of most available transports that may be used to connect two RM instances the service API provides facilities to the system that allow the system to decide how RM's service API issues completed service requests. When a resource service is requested by the system RM can be told to block until the service request has been completed or RM can be told to return the service request result at a later time via a callback function provided by the system. The following services are supported by RM: Service TypePurposeRm_service_RESOURCE_ALLOCATE_INIT Allocates a resource to the requesting system entity, checking initialization privileges of the entity prior to allocation Rm_service_RESOURCE_ALLOCATE_USE Allocates a resource to the requesting system entity, checking usage privileges of the entity prior to allocation Rm_service_RESOURCE_FREE Frees the specified resource from control of the requesting system entity Rm_service_RESOURCE_STATUS Returns the allocation reference count of a specified resource to the requesting system entity Rm_service_RESOURCE_MAP_TO_NAME Maps a specified resource to a specified string and stores the mapping in the RM NameServer Rm_service_RESOURCE_GET_BY_NAME Returns a set of resource values to the requesting system entity based on a specified, existing NameServer name string. The resource is not allocated to the requesting entity. Just the resource values are returned. Rm_service_RESOURCE_UNMAP_NAME Unmaps the specified, existing NameServer name string from a resource and removes the mapping from the RM NameServer Transport API Messages exchanged between RM instances in order to complete service requests all flow through the instance

transport API. RM does not implement any transport mechanisms internally in order to stay device and OS agnostic. A system which integrates RM must supply the transport between any two RM instances. The transport mechanism used to connect two RM instances is completely up to the system. The RM transport API requires two RM instances be registered with one another if they are to communicate. The registration process involves the system providing the RM instances the following application implemented transport functions: Rm_Packet *(*rmAllocPkt)(Rm_AppTransportHandle appTransport, uint32_t pktSize, Rm_PacketHandle *pktHandle);This function will be discussed in depth later but it essentially returns a transport buffer to RM. RM will place the RM specific message within the transport buffer int32_t (*rmSendPkt)(Rm_AppTransportHandle appTransport, Rm_PacketHandle pktHandle);This function will be discussed in depth later but it takes a RM populated transport buffer and sends it on the application transport using the appTransport handle When the application receives a packet/message on a transport designated for RM it must extract the RM packet and provide it to RM via RM's transport receive API: int32_t Rm_receivePacket(Rm_TransportHandle transportHandle, const Rm_Packet *pkt);The RM receive API will not free the RM packet provided to it. It assumes the application transport code will free the RM packet once the RM receive API returns OSAL API The OS Abstraction Layer API allows RM's memory, cache, and blocking mechanism management functions to be defined within the context of the device and/or OS that it will be operating. OSAL APIPurposeSpecial Considerationsextern void *Osal_rmMalloc (uint32_t num_bytes); Allocates a block of memory of specified size to RM Location (local or shared memory), alignment and cache considerations do not matter when allocations originate from the standard Server, Client Delegate, and Client instances. Memory allocated for Shared Server/Client instances must originate from shared memory and be aligned and padded to a cache line. extern void Osal_rmFree (void *ptr, uint32_t size); Frees a block of memory of specified size that was allocated to RM extern void *Osal_rmCsEnter (void); Enters a critical section protecting against access from multiple cores, threads, tasks, and/or processes Critical section protection is not required for the standard Server, Client Delegate, and Client instances since they all operate using independent, non-shared data structures Critical section protection is required for Shared Server/Client instances since the resource management data structures are contained in shared memory extern void Osal_rmCsExit (void *CsHandle); Exits a critical section protecting against access from multiple cores, threads, tasks, and/or processes Critical section protection is not required for the standard Server, Client Delegate, and Client instances since they all operate using independent, non-shared data structures Critical section protection is required for Shared Server/Client instances since the resource management data structures are contained in shared memory extern void Osal_rmBeginMemAccess (void *ptr, uint32_t size); Indicates a block of memory is about to be accessed. If the memory is cached a cache invalidate will occur to ensure the cache is updated with the memory block data residing in actual memory Cache invalidate operations are only required if RM instance data structures are allocated from a cached memory region extern void Osal_rmEndMemAccess (void *ptr, uint32_t size); Indicates a block of memory has finished being accessed. If the memory is cached a cache writeback will occur to ensure the actual memory is updated with contents of the cache Cache writeback operations are only required if RM instance data structures are allocated from a cached memory region extern void *Osal_rmTaskBlockCreate (void); Creates an instance of a task blocking mechanism allowing a RM instance to block in order to wait for a service request to complete RM task blocking is only required if application service requests specifically request RM not return until the service request is satisfied extern void Osal_rmTaskBlock (void *handle); Blocks a RM instance waiting for a service request to complete using the provided task blocking mechanism handle RM task blocking is only required if application service requests specifically request RM not return until the service request is satisfied extern void Osal_rmTaskUnblock (void *handle); Unblocks a RM instance when when a service request has completed. RM task blocking is only required if application service requests specifically request RM not return until the service request is satisfied extern void Osal_rmTaskBlockDelete (void *handle); Deletes an instance of a task blocking mechanism RM task blocking is only required if application service requests specifically request RM not return until the service request is satisfied extern void Osal_rmLog (char *fmt, ...); Allows RM to log various messages This OSAL API is used by RM to print resource status and RM instance status logs Server The RM Server manages all resource data structures and tracks resource ownership. The Server also maintains the NameServer. A majority of resource requests will be forwarded by other instances to the Server for completion. A system integrating RM should contain no more than one RM Server since it maintains the view of all resource's managed by RM. It is possible to have more than one RM Server but the resources managed by each Server must be mutually exclusive. If there is any overlap in resources there may be fatal resource conflicts. There is no limit to the number of Client Delegates and Clients that can connect to the Server. The RM Server is the root of the RM instance connection tree. Client The RM Client is mainly used as an interface to request services. No resource management is done locally on Client instances. Therefore, all requests issued via a Client will be forwarded to a RM Server or CD based on the Client's instance connections. There is no limit to the number of Clients that can exist. However, Clients cannot connect to another Client and can connect to at most one Server or CD. A Client cannot connect to both a Server and CD. Client Delegate (CD) The CD is middle ground between a Server and a Client. The RM CD can manage small subsets of resources, provided by the Server, locally. The CD can handle service requests from connected Clients as long as the resource specified in the resource has been provided to the CD by the Server for local management. Otherwise, the service will be forwarded to the Server for processing. All NameServer requests received by the CD will be forwarded to the Server. The CD is of use in architectures where the transport path between the Server and other RM instances is slow. A CD with a faster transport path between itself and Clients can be established so that not all service requests need to flow over a slow transport path to the Server for completion. If the CD can handle N requests based on the resources provided to it by the Server only every N+1th transaction will be slower since it must be sent over the slow transport path to the Server. There is no limit to the number of CDs that can exist and the number of Clients that can connect to a single CD. However, no two CDs can connect to each other and a CD can be connected to only one Server. Shared Instances Special shared memory versions of the RM Server and Client can be created for systems that have strict cycle requirements and little to no tolerance for blocking operations that make take place within RM instances. When configured the Shared Server and any Shared Clients connected with the Server will complete service requests immediately by accessing resource data structures existing within shared memory. The major requirement for the shared instances is that some form of shared memory is available for access between the Shared Server and Clients. Shared Servers and Shared Clients cannot connect to any RM instances via the transport API. Only Shared Clients can connect to Shared Servers and the connection is made at Shared Client initialization via the shared memory area containing the RM Server resource data structures. Shared Clients are essentially piggybacking on the Shared Server instance located in shared memory. Shared Server A Shared Server instance is no different than standard Server instance. The only difference is the RM OSAL APIs provided by the system must allocate and free from a shared memory area accessible to the portions of the system that will be running the Shared Server and any Shared Clients. Since shared memory will be accessed by the Shared Server the CsEnter/Exit and Begin/EndMemAccess OSAL APIs must account for shared memory accesses and any caching that make take place. As previously mentioned the Shared Server will not accept any connections via the transport API. Only Shared Clients can connect to the Shared Server and that will be at Shared Client initialization time. Shared Client Shared Client instances are no different from standard Client instances from a data structure, resource request standpoint. The major difference is at instance initialization the Shared Client will be provided the location of the Shared Server in shared memory. When service requests are issued via a Shared Client it will map the Shared Server instance and directly access its resource data structures. Therefore, no blocking operations, besides any cache writeback/invalidate operations, will take place. As previously mentioned Shared Clients cannot connect to any instances via the transport API. Shared Clients will connect to a Shared Server by storing the Shared Server instance pointer. If this pointer is not allocated from a shared memory the Shared Server-Client connection will fail to operate and system integrity cannot be guaranteed. How Resources Are Managed RM makes no upfront assumptions about what resources are managed and how they are managed. These decisions are left up to the system integrator. In essence, RM is a glorified number allocator. Allocations and frees are based on strings which are assumed to map to system resources. Who is allowed to use which resource is defined by the system integrator based on the RM instance names. All resource service requests originate from RM instances. Separate portions of a system can be assigned different resources based on a RM instance name. The separate portions of the system are provided a RM instance created with the respective instance name. Allocate/free resource requests originating from the system will only be provided resources assigned to the RM instance it uses to perform the request. This is essentially how different areas within a system can be assigned different resources. The key takeaway is resource names and RM instance names must be synchronized across the different areas of RM in order for proper resource management to take place. The name synchronization architecture allows RM to be completely device agnostic from the perspective of managed resources. Which resources are managed and how they're managed can change from application to application and device to device with no RM source code changes. The key RM features that must be synchronized are the following: Global Resource List (GRL) - Defines all resources that will be managed by the RM Server and the CDs/Clients connected to it. Resources are defined within a resource node containing a resource name and its range of values (base + length format). Policies - Defines how resources are partitioned amongst the RM instances based on the names used to initialize the RM instances RM Instances - Instance names much match the names in the policies used to divide up resources. Service Requests - Resource requests through the service API must match a resource name defined in the GRL. RM_name_synchronization.jpgMost of RM's resource management takes place on the Server instance. The GRL and Policy are provided to the Server as part of the Server's initialization. The GRL is used to allocate and initialization all the resource allocator instances. A resource allocator will be created for each resource node specified within the GRL. The GRL is not used past initialization so it can be placed in a memory region local to the Server instance. The Policy will be validated (i.e. checked for formatting errors) and stored for reference against service requests received from all instances. Policy using the policy provided to the Server will only be made by the Server, through service requests made with the Server instance or forwarded from other instances, so, like the GRL, the policy can be placed in a memory region local to the Server instance. The latter still applies the Shared Server instance. The policy will be copied, wholesale, to a shared memory region provided by the OSAL memory alloc API. Some resource management can be offloaded to the CD instance. The CD can be configured to receive resources from the Server and allocate/free those resources to Clients in lieu of the Server. This offloads some of the management duties from the Server and can save time and cycles if the Server connects over a high latency transport while the CD connects to Clients over a low latency transport. The CD is not provided a GRL at initialization but will request resource blocks from the Server when it receives a request from its service API or a Client that it classifies as something it can handle in lieu of the Server. Typical requests classified in this manner by the CD are non-specific requests or resource requests with an unspecified base value. The At initialization, the CD is provided an exact copy of the policy provided to the Server. This way service request policy checks on the CD will be in sync with the policy check that would have taken place on the Server if the request was forwarded to the Server instead of being handled by the CD. No resource management takes place on the Client. Service requests received on Client instance are always forwarded to either a connected CD or Server. The NameServer is managed solely by the Server instance. The RM NameServer is a very simple NameServer that allows a resource range to be mapped to a string name. Any service request received by any RM instance involving the NameServer will always be forwarded to the Server instance for completion. Allocator Algorithm The allocators are implemented to save cycles when parsing allocated/freed nodes and to save memory as allocations and frees take place over the lifetime of system execution. An open source balanced binary search tree algorithm was used to implement the allocators. Each allocator will maintain the status of a resource's values by creating a node in the tree for each resource value range with different attributes. Where attributes include whether the resource range is allocated or freed, how many RM instances own the resource range, and which RM instances own the resource range. Each time a resource range is modified via service request checks are performed on adjacent resource nodes. The allocator algorithm will attempt to maintain a minimal number of resource nodes in the allocator tree by combining nodes that result in the same attributes as service requests are completed. This will save system memory and minimize the cycles needed to search the allocator trees. Some tree algorithm APIs were added to perform cache writeback/invalidate operations while walking the tree. The additional tree functions were added to support the Shared Server/Client model where the Shared Server's allocators are stored in shared memory. The unmodified search tree algorithm is open source under the BSD license and can be downloaded from OpenBSD.org. The modified search tree algorithm can be found in the pdk_<device>_w_xx_yy_zz/packages/ti/drv/rm/util directory. The modified algorithm is open source, maintaining the BSD license. Static Policies Static policies can be provided to CD and Client instances as a means to "pre-allocate" resources. There may be some cases where a RM instance must be able to allocate some resources prior to the full system being up. In this initialization environment it is unlikely that all RM instances will be created and even more unlikely that they'll be connected via transports. For cases such as the latter, the static policy can used by CDs and Clients to pre-approve resources requested from their service API prior to their transport interface being connected to the Server. The static policy must either be an exact replica of the global policy provided to the Server at initialization or a subset of of the global policy provided to the Server. Service requests from a non-Server instance utilizing a static policy will immediately return an approved resource based on the static policy. Any static requests are stored by the CDs and Clients and forwarded to the Server for validation as soon as the transport to the Server is established. If a static policy allocated resource fails validation via the Server policy the instance that allocated the resource will go into a "locked" state. Service requests cannot be process by locked instances. Recovering a RM instance from the locked state at runtime is not possible. The static policy must be modified and the application restarted. This is intended since the failed validation of an already allocated resource can result in unknown system operation. Only the following services are available with static policies prior to the transport to the Server being established: Rm_service_RESOURCE_ALLOCATE_INITRm_service_RESOURCE_ALLOCATE_USEGRL & Policy Format The GRL and Policy formats follow the ePAPR 1.1 Device Tree Specification. The device tree specification is used to define a simple flattened device tree (FDT) format for expressing resources and their permissions in the GRL and policy device tree source (DTS) files. The GRL and Policy DTS files are converted to device tree blob (DTB) binary files for consumption by the RM Server and CD instances during runtime init. The Linux kernel device tree compiler (DTC) v1.3.0 is used to perform the conversion from DTS to DTB file. Packaged with the DTC is a BSD-licensed, open source, flattened device tree library called LIBFDT. This library has been integrated with RM, packaged in the pdk_<device>_w_xx_yy_zz/packages/ti/drv/rm/util directory, facilitating RM's ability to read the GRL and policy files at runtime. The following graphic portrays the process used to integrate a GRL or policy DTS file into an application using RM. RM_grl_policy_format.jpgFor more information on Flattened Device Trees please see: http://elinux.org/Device_TreesLinux DTB Support In some cases it is desirable for RM to partially automate the extraction of resources consumed by a Linux kernel. The system resources used by Linux are defined in the kernel DTB file, which originates from a, ePAPR 1.1-based, DTS file. RM can easily extract resources identified as used by Linux from the kernel DTB given RM integrates the same DTC utility and LIBFDT library. As will be explained later, the GRL provides a facility for a defined resource to specify a Linux DTB extraction path. At initialization, RM will use this extraction path to parse the Linux DTB file for the resource's value range that has been specified as used by Linux. The RM Server must be provided a pointer to the Linux DTB in the file system at initialization. Otherwise, the automatic extraction will not occur. Configuring & Using RM In the sections to follow the complete process to integrate and configure RM will be covered. A generic RM example will be built over the following sections to supplement the explanation of how to integrate RM. The example will integrate RM with a generic software system that needs three shared resources managed, apples, oranges, and bananas. The use of fruits in the example is in part to be generic but also to convey just how flexible RM is. RM can manage any resource that can be mapped to a string and a value range. Defining the GRL & Policy The GRL defines all system resources, and their value ranges, that will be managed by the RM instances at runtime. If a resource, or a resource's value, is not specified in the GRL it will be nonexistent from the point of view of RM. The policies define how resources defined in the GRL are split amongst the RM instances. If a resource, or a resource's value, is not specified in the policy it is assumed to be not available to any RM instance. The GRL and policies start out as easily editable device tree source files. They are converted to DTBs where they can be fed to the RM Server (or CD and Client in the case of static policies) instance at initialization. Offloading the resource definitions and their access permissions to files provided to RM at initialization allows a system integrator to easily modify which resources are managed by RM, and how they are managed, without having to make RM source code changes. GRL/Policy Definition & Conversion Tools Device Tree Compiler The GRL and policies are based on Device Tree Compiler v1.3.0 available from http://jdl.com/software/dtc-v1.3.0.tgz. DTC v1.3.0 is only supported in a Linux environment. One can attempt to bring the tool up in a Windows environment via Msys or Cygwin but proper operation cannot be guaranteed. DTC Installation Steps Download dtc-v1.3.0.tgz and copy it to a Linux environment Unzip the tar: $ tar xzf dtc-v1.3.0.tgz CD to the created dtc directory: $ cd dtc-v1.3.0 Build the DTC utilities: $ make all An error due to an unused set variable will cause the make to fail. Edit dtc-v1.3.0/Makefile to remove -Wall from the WARNINGS macro Rebuild the DTC utilities: $ make all The 'dtc' and 'ftdump' executables will now exist in the dtc-v1.3.0 directory RM_dtc_dir.jpgDTB to C-Style Array Conversion Utility The cify.sh Shell script used to convert a DTB binary to a C-style array is provided with RM under the pdk_<device>_w_xx_yy_zz/packages/ti/drv/rm/util directory. No installation is needed. Just copy the script to the directory in the Linux environment where DTBs will be located. How to use this utility will be covered in a later section. General DTS Text Format Please read dtc-v1.3.0/Documentation/dts-format.txt prior to creating and editing any DTS files. It contains basic knowledge required to create a valid DTS file. In its basic form the DTS file is a list of nodes containing optional properties. Nodes are defined with C-style curly braces, ending with a semi-colon. Properties can be defined in different ways, with a single integer, a list of integers, a single string, a list of strings, or a mix of all the latter. A property definition ends with a semi-colon. C-style comments are allowed in DTS files. The basic DTS file: /* All RM DTS files must start with the Version 1 DTS file layout identifier */ /dts-v1/; /* Root node - Must ALWAYS be defined */ / { /* Optional root node properties */ root-property = ...; /* Begin child node definitions */ child-node-1 { /* child node optional properties */ cn-1-property = ...; cn-1-other-property = ...; /* Begin child-node-1 sub nodes */ sub-node-1 { property = ...; }; }; child-node-2 { cn-2-property = ...; }; }; The generic property types for DTS files were used to create specific sets of properties for both the GRL and Policy DTS files. GRL Nodes & Properties Any node with a resource range property in the GRL will be identified as a resource for management by RM. The DTS root node should not be assigned a resource range.GRL resource node format: /* Single resource node */ resource-name { resource-range = ...; /* Optional */ ns-assignment = ...; /* Optional */ linux-dtb-alias = ...; }; /* Resource nodes can be grouped as long as the grouping * node does not have a resource-range property */ resource-group { /* Optional */ ns-assignment = ...; resource-name1 { resource-range = ...; /* Optional */ ns-assignment = ...; /* Optional */ linux-dtb-alias = ...; }; resource-name2 { resource-range = ...; /* Optional */ ns-assignment = ...; /* Optional */ linux-dtb-alias = ...; }; resource-name3 { resource-range = ...; /* Optional */ ns-assignment = ...; /* Optional */ linux-dtb-alias = ...; }; }; GRL PropertyFormatPurpose resource-group Single string with no spaces String identifying a resource group. The resource group will not be stored by RM it's main purpose is to allow readability in the GRL file. NameServer assignments can be made from the group for any mappings that don't apply to any specific resources. Linux DTB alias paths are not valid from resource groups. resource-name Single string with no spaces String identifying the resource that RM will manage via an allocator. To apply permissions to this resource the policy must contain a node with the same resource-name. System requests referencing this resource must provide a string matching resource-name as part of the request. resource-range <base-value length-value> Defines the absolute range of values for the resource in base+length format. Bases and lengths can be specified in decimal or hex format. A comma-separated list of multiple base+length pairs can be specified but the base+length pair values cannot overlap. Allocator initialization will fail if any of the pairs overlap. ns-assignment "String_To_Associate", <base-value length-value> Defines a string to resource value to be stored in the RM NameServer. A comma-separated list of multiple NameServer associations can be specified. linux-dtb-alias "Space separated Linux DTB node path", <num-vals base-offset len-offset> Defines the path to an associated resource in the Linux DTB for automatic Linux kernel resource reservation. RM cannot rely on the Linux DTB and the GRL defining a resource in the same format. The linux-dtb-alias property is a way to associate a Linux DTB resource with a resource defined by the GRL and automatically mark the resource as allocated to the Linux kernel. "Space separated string" - A space separated string specifying the node-path to an alias resource in the Linux DTB. The string words match the node names in the path to the alias resource. The last word in the string is the property name that corresponds to the alias resource's value parameters. The complete path need not be specified but the node-path must be exact starting with the first node specified. num-vals - Can have a value of 1 or 2. If 1, the alias resource has just a base value. If 2, the alias resource has a base and a length value. base-offset - Specifies the valid offset to the alias resource's base value. The alias resource's property may contain multiple values. len-offset - [Only applicable when num-vals = 2] Specifies the valid offset to the alias resource's length value, if applicable. The alias resource's property may contain multiple values.GRL PropertyExample resource-group /* Group the system's foo resources */ foos { resource-foo { resource-range = ...; }; resource-bar { resource-range = ...; }; resource-foobar { resource-range = ...; }; }; resource-name /* Define the system's "resource-foo" resource */ resource-foo { resource-range = ...; ... }; resource-range resource-foo { resource-range = <0 25>; }; resource-foo { resource-range = <5 100>, <200 500>, <1000 2000>; }; ns-assignment resource-foo { resource-range = ...; ns-assignment = "Important_Resource", <5 1>; }; resource-foo { resource-range = ...; ns-assignment = "Foo's_Resource", <6 1>, "Bar's_Resource", <7 1>, "Foobar's_Resource", <8 1>; }; linux-dtb-alias Example Linux DTB: /dts-v1/; / { model = "Texas Instruments EVM"; compatible = "ti,evm"; #address-cells = <1>; #size-cells = <1>; memory { device_type = "memory"; reg = <0x80000000 0x8000000>; }; soc6614@2000000 { ... hwqueue0: hwqueue@2a00000 { compatible = "ti,keystone-hwqueue"; ... qmgrs { ... }; queues { general { values = <4000 64>; }; infradma { values = <800 12>; reserved; }; accumulator-low-0 { values = <0 32>; // pdsp-id, channel, entries, pacing mode, latency accumulator = <0 32 8 2 0>; irq-base = <363>; multi-queue; reserved; }; accumulator-low-1 { values = <32 32>; // pdsp-id, channel, entries, pacing mode, latency accumulator = <0 33 8 2 0>; irq-base = <364>; multi-queue; }; accumulator-high { values = <728 8>; // pdsp-id, channel, entries, pacing mode, latency accumulator = <0 20 8 2 0>; irq-base = <150>; reserved; }; ... }; regions { ... }; }; ... }; }; Example Mapping to an alias resource in the Linux DTB: RM will create "accumulator-ch" resource with values 0-47. Alias resources found at the linux-dtb-alias path will be automatically resourced for the Linux Kernel within the "accumulator-ch" allocator. accumulator-ch { resource-range = <0 48>; /* Extract the accumulator channels which * just have a base value in the Linux DTB */ linux-dtb-alias = "hwqueue@2a00000 queues accumulator-low-0 accumulator", <1 1>, "hwqueue@2a00000 queues accumulator-low-1 accumulator", <1 1>, "hwqueue@2a00000 queues accumulator-high accumulator", <1 1>; }; RM will create "infra-queue" resource with values 800-831. Alias resources found at the linux-dtb-alias path will be automatically resourced for the Linux Kernel within the "infra-queue" allocator. infra-queue { resource-range = <800 32>; /* Extract the infrastructure DMA channels which * have a base+length value in the Linux DTB */ linux-dtb-alias = "hwqueue@2a00000 queues infradma values", <2 0 1>; }; Example: fruit-GRL.dts /dts-v1/; / { /* Device Resource Definitions */ fruits { apples { /* 10 apples in system */ resource-range = <0 10>; }; oranges { /* 25 oranges in system */ resource-range = <0 25>; }; bananas { /* 15 bananas in system */ resource-range = <0 15>; ns-assignment = "Banana_for_lunch", <10 1>; }; }; }; Policy Nodes & Properties Policy valid instance node and resource node format: /* Define the RM instances that can assigned resources */ valid-instances = ...; /* Single resource node */ resource-name { assignments = ...; /* Optional */ allocation-alignment = ...; }; /* Resource policy nodes can be grouped */ resource-group { resource-name1 { assignments = ...; /* Optional */ allocation-alignment = ...; }; resource-name2 { assignments = ...; /* Optional */ allocation-alignment = ...; }; resource-name3 { assignments = ...; /* Optional */ allocation-alignment = ...; }; }; Policy PropertyFormatPurpose valid-instances "RM-Instance-Name" List of RM instance name strings that are identified as valid for the allocation specifications made within the policy DTS file. The instance names must match exactly the names assigned to RM instances at their initialization. RM requests will be denied for any RM instance with a name that does not match any of the names in the valid instance list. The policy will be declared invalid if an instance name is specified within the assignment property that does not match any of the instance names in the valid instance list. resource-group Single string with no spaces String identifying a resource group. resource-name Single string with no spaces String identifying the resource that the assignment specifications apply to. RM will return an error specifying the policy is invalid if a resource-name does not match any of the resource allocators i.e. the resource-name does not correspond to a resource-name node defined in the GRL. assignments <base-value length-value>, "permission assignment string" Defines the allocation permissions for a resource range provided in base+length format. Bases and lengths can be specified in decimal or hex format. A comma-separated list of multiple resource assignments can be specified but the base+length values cannot overlap. The "permissions assignment string" consists of the following: "Permission_bits = (RM_instances) & Permission_bits = (RM_instances) & ..." Permission_bits - list of characters that represent the permissions assigned to the RM_instances for the resource range specified by the base+length values. Permission bits can be specified in any order and can be space separated. Possible permissions include: 'i' - Assigns allocation for initialization permission to the instances in the RM_instances list 'u' - Assigns allocation for usage permission to the instances in the (RM_instances) list 'x' - Assigns exclusive allocation permissions to the instances in the (RM_instances) list. Exclusive permissions for an instance entail a resource allocated to an instance that has exclusive permissions for that resource cannot be allocated to another instance. Put another way, the resource cannot be shared amongst multiple instances if an instance with exclusive permissions has been allocated the resource. 's' - Allows shared Linux permissions to the instances in the (RM_instances) list. Resources that are automatically reserved via the Linux DTB for the kernel are by default not allowed to be shared. If an instance(s) are assigned Linux shared permissions they can be allocated a resource from the base+length range that has already been allocated to the Linux kernel. '=' Operator - The equivalence operator signifies the completion of the permission_bits specification and ties the permission bits to a list of (RM_instances). The equivalence operator can be on the left or the right side of the (RM_instances) list. However, no more than one equivalence can be made per (RM_instances). RM will declare the policy as invalid if more than one equivalence operator is used per (RM_instances) list. Empty equivalence - If the equivalence operator equates (RM_instances) to no permission character the base+length range will not be allocatable to any RM instance. (RM_instances) - A space separated list of RM instance names for which the permission bits should be applied for the base+length resource range. Any RM instance name specified must match a name specified in the valid-instances list at the top of the policy. RM will declare the policy invalid if any of the RM Instance names differ. (*) - The '*' operator can be used to specify the permissions bits are valid for ALL instances in the valid-instance list '&' Operator - The and operator allows multiple permission assignments per resource base+length range. The instances within the (RM_instances) lists must be mutually exclusive. A RM instance cannot appear more than once if multiple permission assignments are made for resource base+length allocation-alignment <alignment-value> Defines an alignment value in decimal or hexadecimal to be used by RM for allocation requests that have an unspecified base and unspecified alignment. Only one alignment per resource node can be specified. Policy PropertyExample valid-instances /* Define the RM instances that are valid for the policy */ valid-instances = "RM_foo_Server", "RM_foo_Client", "RM_bar_Client"; }; resource-group /* Group the system's foo resources */ foos { resource-foo { assignments = ...; }; resource-bar { assignments = ...; }; resource-foobar { assignments = ...; }; }; resource-name /* Define "resource-foo"'s allocation specifications */ resource-foo { assignments = ...; ... }; assignments valid-instances = "Server", "Client0", "Client1"; resource-foo { /* All instances get init and use permissions */ assignments = <0 25>, "iu = (*)", <25 25>, "(*) = u i"; }; resource-bar { /* 0 - 4 exclusive for Server, can be shared by clients * if not allocated to Server * 5 - 9 cannot be allocated to anyone * 10 - 14 shared between Client0 and Linux * 15 - 19 can be allocated for initialization by anyone and * shared with any instance */ assignments = <0 5>, "u i x = (Server) & u i = (Client0 Client1)", <5 5>, "(*)", <10 5>, "(Client0) = ius", <15 5>, "i = (Server Client0 Client1)"; }; /* Invalid permission strings * 0-4 more than one equivalence operator * 5 more than one equivalence operator * 6 Instance name not in valid-instances list * 7 invalid permission character */ resource-foobar { assignments = <0 5>, "u = i = (Server)", <5 1>, "iu = (Server) = x", <6 1>, "iux = (My_Server)", <7 1>, "iut = (Server)"; }; allocation-alignment resource-foo { assignments = ...; allocation-alignment = <16>; }; Example: fruit-policy.dts /dts-v1/; / { /* Define the valid instances */ valid-instances = "Server", "Client0", "Client1"; /* Specify who receive the fruit */ fruits { apples { /* Everyone shares the apples */ assignments = <0 10>, "iu = (*)"; }; oranges { /* First 10 oranges can only be shared between the clients * Last 15 oranges can be taken by all instances but can only be shared between the clients */ assignments = <0 10>, "iu = (Client0 Client1)", <10 15>, "iux = (Server) & iu = (Client0 Client1)"; /* Give out every other orange if an instance doesn't know which one it wants */ allocation-alignment = <2>; }; bananas { /* Each instance gets 5 bananas. Also, in this odd world a Linux kernel has taken bananas * 5 and 6 but is willing to share them with the Server. */ assignments = <0 5>, "(Client0) = xiu", <5 2>, "(Server) = siu", <7 3>, "(Server) = xiu", <10 5>, "(Client1) = xiu"; }; }; }; Example: fruit-policy-static.dts /dts-v1/; / { /* Define the valid instances */ valid-instances = "Client1"; /* Statically assign Client1 some bananas before the RM system * is brought up */ bananas { /* Statically allocated resources must align with global policy */ assignments = <10 2>, "xui = (Client1)"; }; }; GRL/Policy Conversion for Input to RM Before any DTS file can be included in an application and passed to a RM instance initialization the DTS file must be converted to a DTB binary using the installed DTC utility. Please see the "MCSDK_UG_Chapter_Developing_System_Mgmt#Defining_the_GRL_.26_PolicyDefining the GRL & Policy" section for instructions on where to find and how to install DTC. DTS to DTB Conversion Instructions Make sure DTC has been installed in a Unix environment Copy the .dts files to a directory accessible to the built dtc and ftdump utilities. Convert the .dts files to .dtb binary files: $ dtc -I dts -O dtb -o <output file name> <input file name> If a syntax error occurs during conversion the dtc tool will provide a file row and column address where the syntax error likely occurred. The address will be in the form row.startCol-endCol as depicted:RM_conversion_syntax_error.jpg The output of the .dtb binary can be checked using the ftdump utility if the conversion succeeds. Strings will be displayed as hex bytes: $ ./ftdump <.dtb file>After conversion there are two possible means to include the DTBs in an application: File Pointer C const Byte Array Inclusion via File Pointer1. Place the DTB files in a file system accessible to the application during runtime 2. mmap or fopen the DTB file. In the case of fopen the data in the DTB file will need to be copied to local data buffer prior to passing the DTB data to a RM instance. A method for doing this is shown here: /* Open the GRL and policy DTB files */ grlFp = fopen("...\\grl.dtb", "rb"); policyFp = fopen("...\\policy.dtb", "rb"); /* Get the size of the GRL and policy */ fseek(grlFp, 0, SEEK_END); grlFileSize = ftell(grlFp); rewind(grlFp); fseek(policyFp, 0, SEEK_END); policyFileSize = ftell(policyFp); rewind(policyFp); /* Allocate buffers to hold the GRL and policy */ grl = Osal_rmMalloc(grlFileSize); policy = Osal_rmMalloc(policyFileSize); /* Read the file data into the allocated buffers */ readSize = fread(grl, 1, grlFileSize, grlFp); System_printf("Read Size compared to file size: %d : %d\n", readSize, grlFileSize); readSize = fread(policy, 1, policyFileSize, policyFp); System_printf("Read Size compared to file size: %d : %d\n", readSize, policyFileSize); /* Create the RM Server instance */ rmInitCfg.instName = "Server"; rmInitCfg.instType = Rm_instType_SERVER; rmInitCfg.instCfg.serverCfg.globalResourceList = (void *)grl; rmInitCfg.instCfg.serverCfg.globalPolicy = (void *)policy; /* Get the RM Server handle */ rmServerHandle = Rm_init(&rmInitCfg); Inclusion via C const Byte ArrayIn this case, the cify shell script provided with RM is used to convert the DTB binary to a C source file containing a const C byte array. The cify script will generate the code to align and pad the byte array to a specified byte size Copy the cify.sh shell script to the Unix environment directory in which the generated .dtb files are located. Convert the shell script to Unix format: $ dos2unix cify.sh Convert any .dtb binaries to C const byte arrays. The usage menu can be printed running the cify script without any input parameters: $ ./cify.sh <input .dtb file> <output .c file> <byte array name> <linker data section name> <byte alignment> extern the byte arrays into the application source and compile the generated C source files into the application Example: Converting fruit GRL and Policies Convert fruit-GRL.dts to .dtb $ dtc -I dts -O dtb -o fruit-GRL.dtb fruit-GRL.dts Convert fruit-policy.dts to .dtb $ dtc -I dts -O dtb -o fruit-policy.dtb fruit-policy.dts Convert fruit-policy-static.dts to .dtb $ dtc -I dts -O dtb -o fruit-policy-static.dtb fruit-policy-static.dts Dump fruit-GRL.dtb (Not required - just showing the output) $ ./ftdump fruit-GRL.dtbRM_dump_fruit_GRL.jpg Convert fruit-GRL.dtb to fruit-GRL.c aligned to 128 bytes $ ./cify.sh fruit-GRL.dtb fruit-GRL.c grl grlSect 128RM_fruit_GRL.jpg Convert fruit-policy.dtb to fruit-policy.c $ ./cify.sh fruit-policy.dtb fruit-policy.c policy policySect 128RM_fruit_policy.jpg Convert fruit-policy-static.dtb to fruit-policy-static.c $ ./cify.sh fruit-policy-static.dtb fruit-policy-static.c policy-static policySect 128RM_fruit_policy_static.jpgRM Instance Initialization & Configuration Source code to add RM instances to the system application can be added now that the resource list has been defined. The first step in this process is identifying how many RM instances are needed and in what context they need to execute from. At the least, RM needs a Server instance to operate. If the system application is one that executes from a single thread on a single core only a Server is needed. However, an application in need of resource management is more than likely multi-threaded and multi-core. For systems it's best to place the RM Server on what can be considered the centralized thread or core.

For example, on TI Keystone devices it's best to place the RM Server on the ARM, running from Linux user-space, or, if the application is DSP-only, core 0, since most applications consider these core contexts the source of most control. Once the RM Server location has been established, identification of where RM Clients are needed should be identified. Clients will be required in any system context that will require different allocation permissions than the Server and any other Client. This can boil down to one Client for each core, one Client per process/thread, or Clients for each software module that may be running within the same thread. The number of RM instances needed is based on the system topology and resource management granularity needed. There are five major steps to initializing the RM instances: Populate the RM OSAL functions based on the placement of the RM instances. The answers to the following questions and more should be considered when populating the RM OSAL APIs Will RM instance memory be allocated from a shared memory? Will RM instance memory be allocated from a cached memory? Will a single instance be shared across multiple tasks/threads/processes? Is internal blocking required within a RM instance because the application can't support receiving service responses at a later time via callback function? (Typical for LLDs) Set each instances initialization parameters and create the RM instances Open the instance service handle In some cases the RM instance will need allocate resources prior to the RM infrastructure being connected via transports. For example, a shared resource needs to be allocated in order to connect two RM instances (semi-chicken and egg problem). To handle these cases the RM instance will be provided a static policy (RM Servers will just reference the regular policy and use the allocators since it has access to everything) during initialization in order to allocate a minimal set of resources to get the system up and running prior to the RM infrastructure being connected. Bringup the application transport code required to connect each RM instance Register the transports between RM instances with each instance via the transport API Instance Initialization RM instance initialization is fairly straightforward process. The key aspect of instance initialization is in regards to RM's name synchronization requirements. Please keep in mind that each RM instance must be assigned a name independent from other instance names at initialization. Also, the instance names assigned to the RM instances must be present in the valid-instances list and the permission strings in the policy files. If the latter requirements are not fulfilled RM instances will fail initialization or not be able to allocate resources. Standard instance initialization parameters are well documented in the RM API Documentation found in pdk_<device>_w_xx_yy_zz/packages/ti/drv/rm/docs/rmDocs.chm or, if the .chm doesn't exist, pdk_<device>_w_xx_yy_zz/packages/ti/drv/rm/docs/doxygen/html/index.html Server Initialization One Server should be initialized per system. All resource management data structures including the NameServer are tracked and modified by the Server instance. Because of this, all service requests will be processed by the Server. Clients will always forward any request to the Server instance for processing. If multiple Servers are desired the resources managed by each server MUST be mutually exclusive. Otherwise, proper system resource management cannot be guaranteed.Standard Server initialization: /* Server instance name (must match with policy valid-instances list */ char rmServerName[RM_NAME_MAX_CHARS] = "Server"; ... /* Initialize the Server */ rmInitCfg.instName = rmServerName; rmInitCfg.instType = Rm_instType_SERVER; rmInitCfg.instCfg.serverCfg.globalResourceList = (void *)rmGRL; rmInitCfg.instCfg.serverCfg.linuxDtb = (void *)rmLinuxDtb; rmInitCfg.instCfg.serverCfg.globalPolicy = (void *)rmGlobalPolicy; rmServerHandle = Rm_init(&rmInitCfg, &result); Configuration breakdown: rmInitCfg.instName = rmServerName; - Pointer to Server's instance name. Must be in policy valid-instances list. rmInitCfg.instType = Rm_instType_SERVER; - Declare Server instance type. rmInitCfg.instCfg.serverCfg.globalResourceList = (void *)rmGRL; - Pointer to GRL dtb. The GRL can be a linked C const byte array or read from a file. The Server will process the GRL and create an allocator for each resource specified within. If the Linux DTB is provided it will be parsed for resources to automatically reserve for the Linux kernel. rmInitCfg.instCfg.serverCfg.linuxDtb = (void *)rmLinuxDtb; - Pointer to the Linux dtb. This is an optional parameter and only useful if Linux is part of the system. In the latter case, the Linux DTB can be provided to the Server at initialization to automatically reserve resource for the kernel. rmInitCfg.instCfg.serverCfg.globalPolicy = (void *)rmGlobalPolicy; - Pointer to the RM infrastructure-wide policy. The policy can be a linked C const byte array or read from a file. The policy provided to the Server must contain the resource allocation specifications for all RM instances present in the system. Client Delegate Initialization Feature not Complete - At time of writing this user guide the CD features are not complete. This section will be updated once full CD functionality is available. Client Initialization There is no limit to the number of Clients that can be defined. The only caveat is no two Clients have the same instance name. Clients contain no resource management data structures. Their main purpose is a permission end point within the RM infrastructure. Any service request received on a Client instance will be forwarded to the Server. The Server will return the response which the Client must provide back to the entity that made the request. Clients can perform static allocations if a static policy is provided at initialization. Static allocation will be fulfilled from the Client until the Client's transport API is registered with another Server or CD instance. Once transport registration occurs static allocations will cease and any requests will be forwarded to the Server. Any static allocations that occurred will be forwarded to the Server upon the first service request made post-transport configuration. The Server will validate any static allocations and provide the responses back to the Client. If any of the static allocations failed validation against the Server's system-wide policy the Client instance will enter a locked state. The Client cannot service requests when in the locked state. The locked state cannot be exited once entered. The system must be restarted with a new, valid static policy that is a subset of the system-wide policy. Standard Client initialization: /* Server instance name (must match with policy valid-instances list */ char rmClientName[RM_NAME_MAX_CHARS] = "Client"; ... /* Initialize a Client */ rmInitCfg.instName = rmClientName; rmInitCfg.instType = Rm_instType_CLIENT; rmInitCfg.instCfg.clientCfg.staticPolicy = (void *)rmStaticPolicy; rmClientHandle = Rm_init(&rmInitCfg, &result); Configuration breakdown: rmInitCfg.instName = rmClientName; - Pointer to Client's instance name. Must be in policy (Server policy and, if applicable, static policy provided to this Client) valid-instances list. rmInitCfg.instType = Rm_instType_CLIENT; - Declare Client instance type rmInitCfg.instCfg.clientCfg.staticPolicy = (void *)rmStaticPolicy; - Pointer to the Client's static policy. This is an optional parameter used if the Client must allocate some specific resources to a system entity prior the system being able to connect the Client to the Server. The static policy can be a linked C const byte array or read from a file. The static policy must be a subset of the system-wide policy given to the Server. Shared Server/Client Initialization A Shared Server and Shared Clients is a special type of RM infrastructure setup that does not require the RM instances be connected via the Transport API. The "transport" between the Shared Clients and Shared Server is essentially the Shared Clients have direct access to the Shared Server's resource management data structures. This requires that all RM Shared Server data structures be allocated from shared memory via the OSAL API. The Shared Server/Client architecture is useful in cases where the application cannot tolerate the blocking operations required by RM instances to send and receive service requests to the Server for completion. The downside to the Shared Server/Client architecture the Shared instances cannot be connected to any standard RM instance via the transport API. All "communication" between the Shared Server and Shared Clients is assumed to be through resource management data structures being located in shared memory. The rmK2H/KC66BiosSharedTestProject delivered with PDK provides an example of how to initialize and use the Shared Server/Client RM infrastructure. The following graphic give a high-level view of how the Shared Server/Client architecture operates: RM_shared_inst.jpgStandard Shared Server initialization: /* Server instance name (must match with policy valid-instances list */ char rmServerName[RM_NAME_MAX_CHARS] = "Server"; ... /* Initialize the Server */ rmInitCfg.instName = rmServerName; rmInitCfg.instType = Rm_instType_SHARED_SERVER; rmInitCfg.instCfg.serverCfg.globalResourceList = (void *)rmGRL; rmInitCfg.instCfg.serverCfg.linuxDtb = (void *)rmLinuxDtb; rmInitCfg.instCfg.serverCfg.globalPolicy = (void *)rmGlobalPolicy; /* RM Shared Server handle returned will be from shared memory */ rmSharedServerHandle = Rm_init(&rmInitCfg, &result); /* Writeback Shared Server handle for Shared Clients - Application must make sure writeback * is aligned and padded to cache line. */ Osal_rmEndMemAccess((void *)&rmSharedServerHandle, sizeof(Rm_Handle)); Configuration breakdown: rmInitCfg.instName = rmServerName; - Pointer to Server's instance name. Must be in policy valid-instances list. rmInitCfg.instType = Rm_instType_SHARED_SERVER; - Declare Shared Server instance type. rmInitCfg.instCfg.serverCfg.globalResourceList = (void *)rmGRL; - Pointer to GRL dtb. The GRL can be a linked C const byte array or read from a file. The GRL can be located in a local or shared memory area since it will only be accessed once during Shared Server initialization. The Server will process the GRL and create an allocator for each resource specified within. If the Linux DTB is provided it will be parsed for resources to automatically reserve for the Linux kernel. rmInitCfg.instCfg.serverCfg.linuxDtb = (void *)rmLinuxDtb; - Pointer to the Linux dtb. This is an optional parameter and only useful if Linux is part of the system. In the latter case, the Linux DTB can be provided to the Server at initialization to automatically reserve resource for the kernel. rmInitCfg.instCfg.serverCfg.globalPolicy = (void *)rmGlobalPolicy; - Pointer to the RM infrastructure-wide policy. The policy can be a linked C const byte array or read from a file. The policy can be located in a local or shared memory area since it will only be accessed once during Shared Server initialization. The Shared Server will malloc a memory block from shared memory and copy the policy into malloc'd memory. The policy provided to the Server must contain the resource allocation specifications for all RM instances present in the system. Osal_rmEndMemAccess((void *)&rmSharedServerHandle, sizeof(Rm_Handle)); - The Shared Server handle will be located in shared memory. It must be written back to memory if cache is enabled so that the Shared Clients can access the Shared Server and it's resource data structures. Standard Shared Client initialization: /* Server instance name (must match with policy valid-instances list */ char rmClientName[RM_NAME_MAX_CHARS] = "Client"; ... /* Wait for Shared Server handle to be valid */ do { Osal_rmBeginMemAccess((void *)&rmSharedServerHandle, sizeof(Rm_Handle)); } while (!sharedServerHandle); /* Initialize a Client */ rmInitCfg.instName = rmClientName; rmInitCfg.instType = Rm_instType_SHARED_CLIENT; rmInitCfg.instCfg.sharedClientCfg.sharedServerHandle = (void *)rmSharedServerHandle; rmClientHandle = Rm_init(&rmInitCfg, &result); Configuration breakdown: Osal_rmBeginMemAccess((void *)&rmSharedServerHandle, sizeof(Rm_Handle)); - Need to invalidate the Shared Server handle from local memory if caching is enabled. Once the Shared Server handle is non-null it has been created and writtenback. rmInitCfg.instName = rmClientName; - Pointer to Client's instance name. Must be in policy (Server policy and, if applicable, static policy provided to this Client) valid-instances list. rmInitCfg.instType = Rm_instType_SHARED_CLIENT; - Declare Shared Client instance type rmInitCfg.instCfg.sharedClientCfg.sharedServerHandle = (void *)rmSharedServerHandle; - Provide the Shared Server handle to the Shared Client instance. When a service request is received on the Shared Client it will remap itself to the Shared Server's instance data so that it can access the resource management data structures located in shared memory. Service requests made through the Shared Client will be validated against the policy using the Shared Client's instance name. RM Transport Configuration The system application must connect communicating RM instances with application created transport paths. The transport paths between instances must be registered with the RM instances using the Transport API. The type of transport path used and the means by which it is created is completely up to the system application. RM Instance Transport Requirements RM will return an error if the following rules are not followed when registering transports with instance: Server Instance Cannot register a transport whose remote instance is another Server Can register an unlimited amount of transports whose remote instances are CDs or Clients CD Instance Cannot register a transport whose remote instance is another CD Cannot register more than one transport whose remote instance is a Server Can register an unlimited amount of transports whose remote instances are Clients Client Instance Cannot register a transport whose remote instance is another Client Cannot register more than one transport whose remote instance is a Server or CD If transport to CD registered cannot register transport to Server If transport to Server registered cannot register transport to CD Transport Registration The system application uses the Rm_transportRegister API to register application created transports to remote instances. The Rm_TransportCfg structure provides specific details about the transport to the RM instance. The details are used by the RM instances to route internal RM messages properly. If a transport is successfully registered a transport handle will be returned to the application. The application can use this handle to tie specific transport pipes to specific RM instances. Following are some configuration details for each Rm_TransportCfg parameter: typedef struct { Rm_Packet *(*rmAllocPkt)(Rm_AppTransportHandle appTransport, uint32_t pktSize, Rm_PacketHandle *pktHandle); int32_t (*rmSendPkt)(Rm_AppTransportHandle appTransport, Rm_PacketHandle pktHandle); } Rm_TransportCallouts; typedef struct { Rm_Handle rmHandle; Rm_AppTransportHandle appTransportHandle; Rm_InstType remoteInstType; const char *remoteInstName; Rm_TransportCallouts transportCallouts; } Rm_TransportCfg; Rm_Handle rmHandle; Instance handle for the instance the transport should be registered with Rm_AppTransportHandle appTransportHandle; Handle to the transport "pipe" object that will transmit a RM packet to specified remote RM instance. The handle provided can be anything from a function pointer, to a pointer to a data structure. In the multi-core RM examples this value is a IPC MessageQ receive queue that is tied to the remote RM instance. This value will be provided to the application as part of the rmSendPkt callout from RM. The application should then be able to use this value to send the Rm_Packet to the remote RM instance. Rm_InstType remoteInstType; Instance type at the remote end of the transport being registered. RM instances that forward service requests will use this field to find the transport that connects to the RM Server or CD const char *remoteInstName; Instance name at the remote end of the transport being registered. Used internal for proper Rm_Packet routing and service request handling Rm_TransportCallouts transportCallouts; Each transport registered can have a different set of alloc and send functions registered with the RM instance. This would be needed if not all transports between RM instances are the same. For example, a CD instance on a DSP core connected to a Server on ARM and Clients on other DSP cores. The application would register a DSP to ARM transport with the CD for the connection to the Server. While for the Client connections the application would register a DSP to DSP transport. The same DSP to DSP transport could be registered for each Client as long as the appliation transport send function can route the Rm_Packets based on the Rm_AppTransportHandle provided by the instance. Rm_Packet *(*rmAllocPkt)(Rm_AppTransportHandle appTransport, uint32_t pktSize, Rm_PacketHandle *pktHandle); RM instance callout function used to get a transport-specific buffer for the Rm_packet to be sent. The pointer to the data buffer where RM should place the Rm_Packet is returned as a Rm_Packet pointer return value. If the data buffer is part of a larger transport packet data structure the pointer to the start of the transport packet data structure is returned via the Rm_PacketHandle pointer argument. The RM instance will not modify anything using the Rm_PacketHandle pointer. The instance will populate the internal RM packet data into the data buffer pointed to by the returned Rm_Packet pointer. The following image depicts the two use cases that must be handled by the application supplied transport allocation function based on the transport's packet type:RM_packet_diff.jpgint32_t (*rmSendPkt)(Rm_AppTransportHandle appTransport, Rm_PacketHandle pktHandle); RM instance callout function used to send the transport packet containing the RM instance data to the remote RM instance. The Rm_AppTransportHandle should be used by the application to route the packet to the correct transport "pipe". The Rm_PacketHandle is a pointer to the application transport packet. Providing Received Packets to RM The system application must receive packets over the transports, extract the buffer containing the Rm_Packet pointer and pass the pointer to the RM instance through the RM transport receive API: int32_t Rm_receivePacket(Rm_TransportHandle transportHandle, const Rm_Packet *pkt); When the application receives a packet it will provide the packet to RM along with a Rm_TransportHandle. The Rm_TransportHandle should map to registered transport whose remote instance is the instance from which the packet was received. RM instances assume all packet free operations will be done by the application. Therefore, after processing a received packet RM will return without attempting to free the packet. The application must free the data buffer containing the RM data and the packet which carried the data buffer. The following image provides a visual depiction of how RM transport routing can be handled by a system application: RM_packet_routing.jpgA good example of how RM instance transports can be handled over IPC for DSP to DSP applications is provided in pdk_<device>_w_xx_yy_zz/packages/ti/drv/rm/test/rm_transport_setup.c Service Request/Response Mechanisms System applications can request resource services from RM via the instance service handles. After each instance has been initialized a service handle can be opened from the instance via the Rm_serviceOpenHandle API: Rm_ServiceHandle *Rm_serviceOpenHandle(Rm_Handle rmHandle, int32_t *result);The service handle returned to the system application consists of two items The RM instance handle from which the service handle was opened A function pointer to RM's service handler function typedef struct { void *rmHandle; void (*Rm_serviceHandler)(void *rmHandle, const Rm_ServiceReqInfo *serviceRequest, Rm_ServiceRespInfo *serviceResponse); } Rm_ServiceHandle; Service requests can be made using the service handle by calling the Rm_serviceHandler function, passing the service handle's rmHandle as a parameter, as well as pointers to RM service request and service response data structures created in the application context (stack or global variable). Standard Service Request

Example void func (void) { Rm_ServiceReqInfo requestInfo; Rm_ServiceRespInfo responseInfo; ... /* Open service handle */ serviceHandle = Rm_serviceOpenHandle(rmHandle, &result); memset((void *)&requestInfo, 0, sizeof(requestInfo)); memset((void *)&responseInfo, 0, sizeof(responseInfo)); /* Configure request */ requestInfo.type = ...; ... serviceHandle->Rm_serviceHandler(serviceHandle->rmHandle, &requestInfo, &responseInfo); /* Retrieve response from responseInfo */ /* Next request */ memset((void *)&requestInfo, 0, sizeof(requestInfo)); memset((void *)&responseInfo, 0, sizeof(responseInfo)); /* Configure request */ requestInfo.type = ...; ... serviceHandle->Rm_serviceHandler(serviceHandle->rmHandle, &requestInfo, &responseInfo); /* Retrieve response from responseInfo */ ... } Service Request Parameters Service requests can be configured use the Rm_ServiceReqInfo structure. Configurations include the ability to: Allocate and free specific resource values Allocate a resource with an unspecified value or, in other words, allocator a specific type of resource but the system doesn't care what value it is. Allocate and free resource blocks of different sizes and alignments Map and unmap resources to names using the RM NameServer Retrieve resources by name via the NameServer Request RM block and not return until the service request has been completed Request RM immediately and provide the completed service request via a application supplied callback function Service Request Structure typedef struct { Rm_ServiceType type; const char *resourceName; #define RM_RESOURCE_BASE_UNSPECIFIED (-1) int32_t resourceBase; uint32_t resourceLength; #define RM_RESOURCE_ALIGNMENT_UNSPECIFIED (-1) int32_t resourceAlignment; const char *resourceNsName; Rm_ServiceCallback callback; } Rm_ServiceReqInfo; Rm_ServiceType type; - The type of RM service requested. See the MCSDK_UG_Chapter_Developing_System_Mgmt#Service_APIService API section for more details. const char *resourceName; - Resource name affected by the request. According to the name synchronization rules the resource name is required to match a resource node name in the GRL and policy. Otherwise, RM will return an error. int32_t resourceBase; - Base value of the resource affected by the request #define RM_RESOURCE_BASE_UNSPECIFIED (-1) - Can be supplied in place of a non-negative resourceBase value. Will tell RM to find the next available resource. An unspecified base value is only valid for allocation requests. uint32_t resourceLength; - Length value of the resource affected by the request. Together the base and length will cause a resource's values from base to (base+length-1) to be affected by the request. int32_t resourceAlignment; - Alignment value of the resource affected by the request. Only valid for allocation requests when the resourceBase is UNSPECIFIED. #define RM_RESOURCE_ALIGNMENT_UNSPECIFIED (-1) - Can be supplied in place of a non-negative resourceAlignment value. RM will default to an alignment of 0 if the alignment is left UNSPECIFIED. An unspecified alignment value is only valid for allocation requests when the resourceBase is UNSPECIFIED. const char *resourceNsName; For NameServer mapping requests specifies the NameServer name to map the specified resources to For NameServer unmap requests specifies the NameServer name to unmap/remove from the NameServer For NameServer get by name requests specifies the NameServer name to retrieve mapped resources from in the NameServer. RM will return an error if a get by name request has both a resource specified via the base and length values and a NameServer name. Rm_ServiceCallback callback; - Application's callback function for RM to use to return the completed service request to the application. The RM instance will by default block until the completed service request can be returned via the call stack if the callback function pointer is left NULL. RM instance blocking is handled by the block/unblock OSAL functions. RM will return a serviceId in the service response via the call stack if a callback function is provided and the service request resulted in the RM instance needing to forward the service request to another RM instance for completion. The completed request will be returned at a later time via the provided callback function. the completed request will contain a serviceId that matches the serviceId returned by the instance via the call stack. Service Response Parameters Service Response Structure typedef struct { Rm_Handle rmHandle; int32_t serviceState; uint32_t serviceId; char resourceName[RM_NAME_MAX_CHARS]; int32_t resourceBase; uint32_t resourceLength; #define RM_RESOURCE_NUM_OWNERS_INVALID (-1) int32_t resourceNumOwners; } Rm_ServiceRespInfo; Rm_Handle rmHandle; - The service response will contain the RM instance handle from which the service request was originally issues. This is for the application to sort responses received via callback function in system contexts where more than one RM instance runs int32_t serviceState; - Server state contains the service completion status. SERVICE_APPROVED - Service request completed successfully SERVICE_APPROVED_STATIC - Service request completed successfully via static policy. The service request will be validated on the Server once transports are configured SERVICE_PROCESSING - Service request was forwarded to a CD or Server for completion. The response will be returned through the provided application callback function Any other value > 0 - Service request denied (See rm.h) Any other value < 0 - Error encountered processing the service request (See rm.h) uint32_t serviceId; - ID assigned to service request by RM The ID returned by RM will never zero. IDs assigned to processing requests can be used to match responses received at later time via the application callback function char resourceName[RM_NAME_MAX_CHARS]; - Contains the resource name affected by the completed request int32_t resourceBase; - Contains the resource base value affected by the completed request uint32_t resourceLength; - Contains the resource length affected by the completed request int32_t resourceNumOwners; - If valid, returns the number of current owners for the resource specified in the service request Will contain the value RM_RESOURCE_NUM_OWNERS_INVALID if the service request has nothing to do with owner reference counts. Such as NameServer services. Blocking & Non-Blocking Requests Please see the MCSDK_UG_Chapter_Developing_System_Mgmt#Service Request Configuration OptionsService Request Config Options section describing the service request's callback function pointer parameter for an accurate description of how RM blockin/non-blocking operations can be configured. RM InstanceBlocking Criteria Server Will never block for service requests received through its service API Client Delegate Will not block for service requests received through its service API that can be completed using resources the CD currently owns Will need to block for service requests that must be forwarded to a Server located on a different core/process/thread/task because the locally owned resources cannot be used to complete the request Client Will need to block for all service requests since no resource management data structures on managed by Clients. Blocking will not occur if the Client is local to the Server i.e. the Client and Server can be connected over the transport API via function calls. Shared Server & Client Will never block since the Shared Server and Shared Clients can all access the resource management data structures in shared memory. Direct & Indirect Resource Management Service requests allow resources to be managed through direct and indirect means. Direct management refers to RM being integrated with a software component under the hood. Beyond providing a RM instance handle to the module, resource management interactions with the module's resources are abstracted from the system. An example of this would be QMSS, CPPI, and PA LLDs which have integrated RM to differing degrees. Indirect management refers to RM being used to provide a resource value (or set of values) then passing those values to a module API to actually allocate the resource with the provided values. For example, RM is not currently integrated into any module that manages semaphores. Therefore, RM resource management of semaphores would need to occur indirectly with the following steps Add a semaphore resource to the GRL Add semaphore allocation permissions to the policy (and static policy if applicable) System creates a RM instance System issues a service request to the instance requesting a semaphore resource RM returns semaphore value based on request and policy System uses returned value in call to module to manages semaphores (BIOS, CSL, etc.) In the latter case, RM is managing which system contexts are allowed to allocate and share which resources. It's just that the resource value management is not abstracted from the system. RM_direct_indirect.jpgRM Supported LLDs At this point in time the following LLDs have had RM support added: QMSS CPPI PA The breadth of resources supported for each of these LLDs varies and will be covered in each LLDs section. Each LLD remains backwards compatible with expected operation prior to RM integration. A RM instance service handle can be registered with each existing LLD context. The LLD will use RM to allocate resources tagged as resource to be managed by RM if a RM service handle is registered. The LLD will allocate resources in the same way it allocated resources prior to RM integration if a RM service handle is registered. QMSS LLD The RM service handle can be registered with the QMSS LLD in the following manner: /* Define the system initialization core */ #define SYSINIT 0 void foo (void) { Qmss_StartCfg qmssStartCfg; ... /* Open RM service handle */ rmServiceHandle = Rm_serviceOpenHandle(rmHandle, &rmResult); ... if (core == SYSINIT) { /* Register RM Service handle with QMSS */ qmssGblCfgParams.qmRmServiceHandle = rmServiceHandle; /* Initialize Queue Manager SubSystem */

result = Qmss_init (&qmssInitConfig, &qmssGblCfgParams); } else { /* Register RM service handle with QMSS */ qmssStartCfg.rmServiceHandle = rmServiceHandle; result = Qmss_startCfg (&qmssStartCfg); } } Please refer to the LLD for resources managed by RM if a RM instance's service handle is registered. QMSS test/example projects that register RM on the DSP qmInfraMCK2HC66BiosExampleProject qmInfraMCK2KC66BiosExampleProject qmQosSchedDropSchedK2HC66BiosTestProject qmQosSchedDropSchedK2KC66BiosTestProject QMSS test/example programs that use RM on the ARM qmInfraDmaMC_k2[hk].out qmInfraDmaSC_k2[hk].out qmQAllocTest_k2[hk].out In order to use the ARM programs, the RM server must be explicitly started separately from the qmss program. Using the files global-resource-list.dtb and policy_dsp_arm.dtb from pdk_keystone2_1_00_00_11/packages/ti/drv/rm/device/k2[hk], invoke: ./rmServer.out global-resource-list.dtb policy_dsp_arm.dtbThe qmInfraDmaMC.out and qmQAllocTest.out are compiled using packages/ti/drv/qmss/makefile_armv7 after running configuring and running packages/armv7setup.sh for tools locations in your linux system. qmInfraDmaSC and qmQAllocTest do not require arguments. The qmInfraDmaMC also doesn't require an argument, but can take one argument which is task/core number. Without arguments it forks 4 tasks; with arguments it runs one task with the specified ID number. This facilitates gdb on indvidual tasks. CPPI LLD The RM service handle can be registered with the CPPI LLD in the following manner: void foo (void) { Cppi_StartCfg cppiStartCfg; ... /* Open RM service handle */ rmServiceHandle = Rm_serviceOpenHandle(rmHandle, &rmResult); ... /* Register RM with CPPI for each core */ cppiStartCfg.rmServiceHandle = rmServiceHandle; Cppi_startCfg(&cppiStartCfg); } Please refer to the LLD for resources managed by RM if a RM instance's service handle is registered. CPPI test/example projects that register RM: cppiK2HC66BiosExampleProject cppiK2KC66BiosExampleProject cppiK2HC66BiosTestProject cppiK2KC66BiosTestProject PA LLD The RM service handle can be registered with the PA LLD in the following manner: void foo (void) { paStartCfg_t paStartCfg; ... /* Open RM service handle */ rmServiceHandle = Rm_serviceOpenHandle(rmHandle, &rmResult); ... /* Register RM with PA for each core */ paStartCfg.rmServiceHandle = rmServiceHandle; Pa_startCfg(paInst, &paStartCfg); } Please refer to the LLD for resources managed by RM if a RM instance's service handle is registered. PA test/example projects that register RM: PA_simpleExample_K2HC66BiosExampleProject PA_simpleExample_K2KC66BiosExampleProject Running RM Test Projects This section describes how to run the RM test projects ARM Linux RM Server Daemon The RM test code is delivered with a RM Server daemon application based on a socket transport interface. The RM Server daemon can be used to run all RM test projects and user-space LLD examples and projects. The RM Server daemon can even be used with end-user application infrastructures. However, please keep in mind that the RM Server daemon is meant more of a guide for implementing a Linux-based RM server than an end-all, be-all Linux RM Server. The RM Server communicates with other RM Clients running in other processes via sockets. The daemon's transport code will remember established socket connections to the RM Server.The RM Server daemon logs all incoming service requests and prints the resource status upon sending out each resource request response. The RM server daemon logs can be found in /var/log/rmServer.log. The RM Server daemon does not have a run-time user interface but does provide configuration options when started. The configuration options are displayed with the '-h' or '--help' commands. Here is an example from, the version you are using may be different:rmserverhelp.jpgNoteNote: Please take special care to provide the proper GRL and policy files to the RM Server at startup when attempting to run different tests. For example, the RM test applications require the GRL, policy and linux DTB files in the rm/test/dts_files directory. Whereas, all user-space LLD tests require the device GRL and policy files within the rm/device/k2*/ directories.NoteNote: There's currently a bug in the RM Server daemon -kill logic that sometimes causes the kill to timeout. When this happens the RM Server can be killed with following command, <RM server PID> is retrieved from "ps -aux": root@keystone-evm:~# kill -9 <RM server PID> ARM Linux Multi-Process Test Project The ARM Linux Multi-Process Test has a RM Server and RM Client running in separate user-space processes. The Client and Server exchange resource requests and responses over a socket. Test application components: Linux User-Space RM Server - rmServer.out (located in /usr/bin) Linux User-Space RM Client - rmLinuxClientTest.out (located in /usr/bin) Running the RM ServerCopy the global-resources.dtb, server-policy.dtb, and linux-evm.dtb from rm/test/dts_files/ to the directory containing rmServer.out Run the RM Server: $ ./rmServer.out global-resources.dtb server-policy.dtb -l linux-evm.dtb The Server will wait for Client socket connections and service any requests received via those sockets. Running the RM Client Linux User-Space TestCopy the static-policy.dtb from rm/test/dts_files/ to the directory containing rmLinuxClientTest.out Run the RM Client: $ ./rmLinuxClientTest.out static-policy.dtb The Client test will establish a socket connection with the Server, request resources and then free all resources requested. DSP + ARM Linux Test Project The DSP+ARM Linux Test has a RM Client running on a DSP who sends resource requests to the RM Server running within a Linux user-space process. The RM Client sends messages to another Linux user-space process over IPC 3.x MessageQ. The process extracts the RM messages received on MessageQ interface and sends them through a socket to the RM Server instance. From RM Server to Client the same process extracts RM messages received over the socket from the Server and sends them to the Client over the IPC MessageQ interface. The DSP application must be downloaded to the DSP core using MPM. Otherwise, IPC MessageQ will fail to work. Test application components: Linux User-Space RM Server - rmServer.out (located in /usr/bin) Linux User-Space IPC to MessageQ Exchange process - rmDspClientTest.out (located in /usr/bin) C66 DSP RM Client - rmK2xArmv7LinuxDspClientTestProject.out Build the DSP RM ClientImport the rmK2xArmv7LinuxDspClientTestProject into CCS and build the project Copy the projects .out file from the project /Debug directory to the EVM filesystem Running the RM ServerCopy the global-resources.dtb, server-policy.dtb, and linux-evm.dtb from rm/test/dts_files/ to the directory containing rmServer.out Run the RM Server: $ ./rmServer.out global-resources.dtb server-policy.dtb -l linux-evm.dtb Downloading the DSP ApplicationThe DSP application must be downloaded to all DSP cores via MPM prior to running the exchange process. Otherwise, IPC will fail to start Download the following shell scripts and place them on the EVM. The shell scripts are used to download, reset, and dump traces from DSPs over MPM - Mpm_scripts load_all.sh - Downloads a DSP image to all DSPs stop_all.sh - Resets and halts all DSPs dump_trace.sh - Dumps trace information, like printf output, from the DSPs Make sure LAD is running: # ps aux|grep lad - the output should show an instance of /usr/bin/lad_tci6638 lad.txt running Create a /debug directory for the MPM trace information: # mkdir /debug Mount debugfs in /debug: # mount -t debugfs none /debug Download the DSP Client: # ./load_all.sh rmK2xArmv7LinuxDspClientTestProject.out - the output should show 8 DSPs cores loaded and run successfully root@keystone-evm:~# ./load_all.sh rmK2HArmv7LinuxDspClientTestProject.out Loading and Running rmK2HArmv7LinuxDspClientTestProject.out ... load succeded run succeded load succeded run succeded load succeded run succeded load succeded run succeded load succeded run succeded load succeded run succeded load succeded run succeded load succeded run succeded Running the MessageQ - Socket Exchange ApplicationRun the exchange application: # ./rmDspClientTest.out - The output should show messages received from the DSP Client root@keystone-evm:~/pdk_keystone2_1_00_00_10/bin/k2h/armv7/rm/test# ./rmDspClientTest.out Using procId : 1 Entered rmServerExchange_execute Local MessageQId: 0x1 Remote queueId [0x10000] Setting up socket connection with RM Server Waiting for RM messages from DSP Client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Received RM pkt of size 264 from DSP client Check the Test ResultDump the trace information from the DSPs: # ./dump_trace.sh - DSP Core 0 should have printed an Example Completion with no errors. Cores 1 - 7 will not execute the full test root@keystone-evm:~# ./dump_trace.sh Core 0 Trace... 3 Resource entries at 0x800000 *** ************ RM DSP+ARM DSP Client Testing ************** *** RM Version : 0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Initialized RM_Client Core 0 : --- Core 0 : ---------------- Static Init Allocation ----------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: qos-cluster - Core 0 : - Start: 0 - Core 0 : - End: 0 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : ---------------- Static Init Allocation ----------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: qos-cluster - Core 0 : - Start: 2 - Core 0 : - End: 2 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : ---------------- Static Init Allocation ----------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: qos-cluster - Core 0 : - Start: 1 - Core 0 : - End: 1 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Denial: 79 - Core 0 : --- Core 0 : --- Core 0 : ---------------- Static Init Allocation ----------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-queue - Core 0 : - Start: 525 - Core 0 : - End: 525 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : ---------------- Static Init Allocation ----------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-queue - Core 0 : - Start: 525 - Core 0 : - End: 525 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : Creating RM startup task... Core 0 : Starting BIOS... registering rpmsg-proto service on 61 with HOST Awaiting sync message from host... [t=0x00000012:2c9060f4] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53 [t=0x00000012:2d3612fb] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53 Core 0 : Creating RM receive task... Core 0 : Creating RM client task... Core 0 : --- Core 0 : --------------- Create NameServer Object ---------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 1002 - Core 0 : - End: 9161389 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : ------- Retrieve Resource Via NameServer Object --------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 1002 - Core 0 : - End: 1002 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : -------- Init Allocate Using Retrieved Resource --------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 1002 - Core 0 : - End: 1002 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : ---- Retrieve Resource Status Via NameServer Object ----- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 1002 - Core 0 : - End: 1002 - Core 0 : - Expected Owner Count: 1 - Core 0 : - Returned Owner Count: 1 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : --- Free of Retrieved Resource Using NameServer Name ---- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: My_Favorite_Queue - Core 0 : - Start: 0 - Core 0 : - End: 0 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : --- Core 0 : --- Core 0 : -------------- Delete NameServer Object ---------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: My_Favorite_Queue - Core 0 : - Start: 0 - Core 0 : -

End: 0 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : - Resource Node Expand/Contract Testing (Use Allocate) -- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-rx-ch - Core 0 : - Start: 0 - Core 0 : - End: 5 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : - Resource Node Expand/Contract Testing (Init Allocate) - Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-rx-ch - Core 0 : - Start: 50 - Core 0 : - End: 56 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : ---------- Use Allocation w/ UNSPECIFIED Base ----------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: accumulator-ch - Core 0 : - Start: UNSPECIFIED - Core 0 : - Length: 5 - Core 0 : - Alignment: 4 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : ---------- Use Allocation w/ UNSPECIFIED Base ----------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: accumulator-ch - Core 0 : - Start: UNSPECIFIED - Core 0 : - Length: 2 - Core 0 : - Alignment: 1 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : ---- Use Allocation w/ UNSPECIFIED Base & Alignment ----- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: accumulator-ch - Core 0 : - Start: UNSPECIFIED - Core 0 : - Length: 2 - Core 0 : - Alignment: UNSPECIFIED - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : -- Init Allocation of Shared Linux and Client Resource -- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: infra-queue - Core 0 : - Start: 800 - Core 0 : - End: 800 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : - Init Allocation (RM Blocked Until Resource Returned) -- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7000 - Core 0 : - End: 7000 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : -- Use Allocation (RM Blocked Until Resource Returned) -- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7005 - Core 0 : - End: 7029 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : -- Use Allocation (RM Blocked Until Resource Returned) -- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7010 - Core 0 : - End: 7014 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : ----- Use Allocation of Owned Resource (RM Blocked) ----- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7011 - Core 0 : - End: 7011 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : -- Status Check of Resources from Client (Non-Blocking) - Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7012 - Core 0 : - End: 7013 - Core 0 : - Expected Owner Count: 1 - Core 0 : - Returned Owner Count: 1 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : ---- Status Check of Resources from Client (Blocking) --- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 4025 - Core 0 : - End: 4044 - Core 0 : - Expected Owner Count: 1 - Core 0 : - Returned Owner Count: 1 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : -- Core 0 : ------------- Static Allocation Validation ------------- Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: qos-cluster - Core 0 : - Start: 0 - Core 0 : - End: 0 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -- Core 0 : --

```
Core 0 : ------------ Static Allocation Validation ------------- Core 0 : - Instance Name: RM_Client - Core 0 : -
Resource Name: qos-cluster - Core 0 : - Start: 2 - Core 0 : - End: 2 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : -
PASSED - Core 0 : ----------------------------------------------------- Core 0 :
----------------------------------------------------- Core 0 : ------------ Static Allocation Validation ------------- Core 0 :
- Instance Name: RM_Client - Core 0 : - Resource Name: aif-queue - Core 0 : - Start: 525 - Core 0 : - End: 525 -
Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : ------------ Static Allocation Validation -------------
Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-queue - Core 0 : - Start: 525 - Core 0 : - End:
525 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 :
----------------------------------------------------- Core 0 : Creating RM cleanup task... Core 0 :
----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------ Core 0 :
- Instance Name: RM_Client - Core 0 : - Resource Name: accumulator-ch - Core 0 : - Start: 0 - Core 0 : - End: 6 -
Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------
Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: accumulator-ch - Core 0 : - Start: 40 - Core 0 : -
End: 41 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 :
----------------------------------------------------- Core 0 : ----------------------------------------------------- Core 0 :
------------------- Resource Cleanup ------------------ Core 0 : - Instance Name: RM_Client - Core 0 : - Resource
Name: qos-cluster - Core 0 : - Start: 0 - Core 0 : - End: 0 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED -
Core 0 : ----------------------------------------------------- Core 0 : -----------------------------------------------------
Core 0 : ------------------- Resource Cleanup ------------------ Core 0 : - Instance Name: RM_Client - Core 0 : -
Resource Name: qos-cluster - Core 0 : - Start: 2 - Core 0 : - End: 2 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : -
PASSED - Core 0 : ----------------------------------------------------- Core 0 :
----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------ Core 0 :
- Instance Name: RM_Client - Core 0 : - Resource Name: aif-queue - Core 0 : - Start: 525 - Core 0 : - End: 525 -
Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------
Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-queue - Core 0 : - Start: 525 - Core 0 : - End:
525 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 :
----------------------------------------------------- Core 0 : ----------------------------------------------------- Core 0 :
------------------- Resource Cleanup ------------------ Core 0 : - Instance Name: RM_Client - Core 0 : - Resource
Name: infra-queue - Core 0 : - Start: 800 - Core 0 : - End: 800 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : -
PASSED - Core 0 : ----------------------------------------------------- Core 0 :
----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------ Core 0 :
- Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7000 - Core 0 : - End: 7000 -
Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------
Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7011 - Core 0 : - End:
7011 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 :
----------------------------------------------------- Core 0 : ----------------------------------------------------- Core 0 :
------------------- Resource Cleanup ------------------ Core 0 : - Instance Name: RM_Client - Core 0 : - Resource
Name: gp-queue - Core 0 : - Start: 7010 - Core 0 : - End: 7014 - Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : -
PASSED - Core 0 : ----------------------------------------------------- Core 0 :
----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------ Core 0 :
- Instance Name: RM_Client - Core 0 : - Resource Name: gp-queue - Core 0 : - Start: 7005 - Core 0 : - End: 7029 -
Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : ------------------- Resource Cleanup ------------------
Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-rx-ch - Core 0 : - Start: 0 - Core 0 : - End: 5 -
Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : -------------------- Resource Cleanup -------------------
Core 0 : - Instance Name: RM_Client - Core 0 : - Resource Name: aif-rx-ch - Core 0 : - Start: 50 - Core 0 : - End: 56
- Core 0 : - Alignment: 0 - Core 0 : - - Core 0 : - PASSED - Core 0 : -----------------------------------------------------
Core 0 : ----------------------------------------------------- Core 0 : ------------------ Memory Leak Check
-------------------- Core 0 : - : malloc count | free count - Core 0 : - Example Completion : 101 | 101 - Core 0 : -
PASSED - Core 0 : ----------------------------------------------------- Core 0 :
----------------------------------------------------- Core 0 : ------------------ Example Completion ------------------- Core
0 : ----------------------------------------------------- ---------------------------------------- Core 1 Trace... 3 Resource
entries at 0x800000 *********************************************************** ************ RM
DSP+ARM DSP Client Testing **************
*********************************************************** RM Version : 0x02000004 Version String:
RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 1 : RM DSP+ARM Linux test not executing on this core
Core 1 : Creating RM startup task... Core 1 : Starting BIOS... registering rpmsg-proto service on 61 with HOST
[t=0x00000012:2572e5da] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53
[t=0x00000012:2617dddd] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53
---------------------------------------- Core 2 Trace... 3 Resource entries at 0x800000
*********************************************************** ************ RM DSP+ARM DSP Client
Testing ************** *********************************************************** RM Version :
0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 2 : RM DSP+ARM Linux test
not executing on this core Core 2 : Creating RM startup task... Core 2 : Starting BIOS... registering rpmsg-proto
service on 61 with HOST [t=0x00000012:13e85d6e] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for
endpoint: 53 [t=0x00000012:148d0415] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53
---------------------------------------- Core 3 Trace... 3 Resource entries at 0x800000
*********************************************************** ************ RM DSP+ARM DSP Client
Testing ************** *********************************************************** RM Version :
0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 3 : RM DSP+ARM Linux test
not executing on this core Core 3 : Creating RM startup task... Core 3 : Starting BIOS... registering rpmsg-proto
service on 61 with HOST [t=0x00000012:0da8b05a] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for
endpoint: 53 [t=0x00000012:0e4c8b53] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53
---------------------------------------- Core 4 Trace... 3 Resource entries at 0x800000
*********************************************************** ************ RM DSP+ARM DSP Client
Testing ************** *********************************************************** RM Version :
0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 4 : RM DSP+ARM Linux test
not executing on this core Core 4 : Creating RM startup task... Core 4 : Starting BIOS... registering rpmsg-proto
service on 61 with HOST [t=0x00000012:073f981e] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for
endpoint: 53 [t=0x00000012:07e2a25d] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53
---------------------------------------- Core 5 Trace... 3 Resource entries at 0x800000
*********************************************************** ************ RM DSP+ARM DSP Client
Testing ************** *********************************************************** RM Version :
0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 5 : RM DSP+ARM Linux test
not executing on this core Core 5 : Creating RM startup task... Core 5 : Starting BIOS... registering rpmsg-proto
service on 61 with HOST [t=0x00000012:012ce260] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for
endpoint: 53 [t=0x00000012:01cfa139] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53
---------------------------------------- Core 6 Trace... 3 Resource entries at 0x800000
*********************************************************** ************ RM DSP+ARM DSP Client
```

Testing ************** *** RM Version : 0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 6 : RM DSP+ARM Linux test not executing on this core Core 6 : Creating RM startup task... Core 6 : Starting BIOS... registering rpmsg-proto service on 61 with HOST [t=0x00000011:f8bb09ae] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53 [t=0x00000011:f95d7d57] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53 -- Core 7 Trace... 3 Resource entries at 0x800000 *** ************ RM DSP+ARM DSP Client Testing ************** *** RM Version : 0x02000004 Version String: RM Revision: 02.00.00.04:May 14 2013:16:19:59 Core 7 : RM DSP+ARM Linux test not executing on this core Core 7 : Creating RM startup task... Core 7 : Starting BIOS... registering rpmsg-proto service on 61 with HOST [t=0x00000011:f1ed4c0c] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53 [t=0x00000011:f28ed241] ti.ipc.rpmsg.RPMessage: RPMessage_send: no object for endpoint: 53 root@keystone-evm:~# User Space DMA The user space DMA framework provides the zero copy access from user-space to packet dma channels. A new keystone specific udma kernel driver is added to the baseline which interfaces with the packet dma driver for this copy from user space to kernel (and vice versa). Apart from this, there is a user space library provided (libudma.a) which can be used by an application to create a udma channel, submit buffers to the channel, and receive callbacks for any buffers that the channel gets on the receive side. The user space udma can be accessed using meta-arago/meta-arago-extras/recipes-bsp/ti-udma/ti-udma_1.0.bb. To interface with the user dma library, the APIs are provided under the "include" directory of the package.UDMA Files in the filesystem The UDMA user space library is present in /usr/bin/libudma.a. This is the library that an application should include if they want to use the user space udma application. There is a small udma_test application which opens a single udma tx and a single udma rx channel and performs a loopback of 2M packets.This udma_test application can be found in /usr/bin/udma_test.Default udma channels setup in the kernel As part of the integration of the UDMA into the baseline - the device tree was modified to setup 12 tx channels (udmatx0 - udmatx11) and 12 rx channels (udmarx0 - udmarx11) by default. Each of the TX channel is configured with a hardware queue number to use. Please refer to the device tree in the Linux kernel tree (./arch/arm/boot/dts/k2hk-evm.dts)Contiguous memory allocator Many applications need access to contiguous memory from user space. The MCSDK integrates a Contiguous memory allocator module which can be configured to use different pools of memory. The CMEM module supports allocating from the global CMA pool or from pre-configured pools. Linux configuration Linux configuration is done when installing the cmemk.ko driver, typically done using the insmod command. The cmemk.ko driver accepts command line parameters for configuring the physical memory to reserve and how to carve it up. The cmemk.ko is located at /lib/modules/<kernel_version>/extra/cmemk.ko NoteNote: The kernel module cmemk.ko included in MCSDK filesystem is compiled for the specific version of the kernel. If the kernel version used is different from the MCSDK release kernel version, the cmemk kernel module needs to be recompiled with the linux-headers package for the specific kernel version.The following is an example of installing the cmem kernel module: /sbin/insmod /lib/modules/3.8.4-g42865b7/extra/cmemk.ko pools=4x30000,2x500000 phys_start=0x0 phys_end=0x3000000 phys_start and phys_end must be specified in hexadecimal format pools must be specified using decimal format (for both number and size), since using hexadecimal format would visually clutter the specification due to the use of "x" as a token separator This particular command creates 32 pools. The first pool is created with 4 buffers of size 30000 bytes and the second pool is created with 2 buffers of size 500000 bytes. The CMEM pool buffers start at 0x0 and end at 0x3000000 (max). Pool buffers are aligned on a module-dependent boundary, and their sizes are rounded up to this same boundary. This applies to each buffer within a pool. The total space used by an individual pool will therefore be greater than (or equal to) the exact amount requested in the installation of the module. The poolid used in the driver calls would be 0 for the first pool and 1 for the second pool. Pool allocations can be requested explicitly by pool number, or more generally by just a size. For size-based allocations, the pool which best fits the requested size is automatically chosen. User space Access The user space application can be compiled with the linux development kit to access the user space APIs. The API details can be found at ./sysroots/armv7ahf-vfp-neon-oe-linux-gnueabi/usr/include/ti/cmem.h in the linux development kit. NoteNote: The user space application need to be compiled with "–D_FILE_OFFSET_BITS=64" to allow physical addresses > 32 bits.The CMEM example code can be obtained from git repository: git://git.ti.com/ipc/ludev.git See src/cmem/tests for api usage test.

MCSDK UG Chapter Developing Fault Mgmt

Developing with MCSDK: Fault Management

Last updated: **09/20/2015**

Overview

Learn about troubleshooting, monitoring, error recovery with the MCSDK.

DSP Fault Management (FM)

This section describes the Fault Management APIs and provides information on how to instrument a DSP image for exception, capture the DSP core dump, and analyze the fault using the core dump in CCS.

FM DSP Exception APIs & Usage

A DSP application can create and install an exception hook configured with the following features using the APIs provided by FM:

- Retrieve DSP register status for a core dump
- Halt system IO, such as DMAs
- Inform remote DSPs an exception has occurred
- Inform Host a DSP exception has occurred

Retrieving Last Register Status

An exception hook function that intends to create a core dump must call the following API:

```
void Fault_Mgmt_getLastRegStatus(void);
```

The API stores relevant DSP core register values into the memory region mapped to the fault_mgmt_data data array for the core dump that is provided to the Host.

Halting System IO

When a DSP exception is detected, invoking the FM instrumented exception hook function, system IO that offload data transfers can be stopped using the following API:

```
Fault_Mgmt_haltIoProcessing(Fm_GlobalConfigParams *fmGblCfgParams,
Fm_HaltCfg *haltCfg)
```

The API provides the ability to disable all or subsets of the following system IO:

- AIF2 PE (Tx) and PD (Rx) channels
- EDMA3 DMA, QDMA, and INT channels
- CPDMA Tx/Rx channels
- SGMII switch

The exception hook can invoke this function in order to stop DSP enabled IO from continuing data transfers that may wipe out data important to deciphering the source of the DSP exception. However, care must be taken to not disable resources in use by the Host. If this occurs the Host (ARM) may experience unstable behavior preventing it from properly receiving and handling the DSP core dump.

IO Halt Configuration Parameters

The Fm_HaltCfg structure is used to define which system IO will be disabled upon invocation of the IO halt API. A user can specify that an entire IO be disabled or a subset of an IO be disabled. The Fm_HaltCfg structure has the following parameters:

```
int32_t haltAif
```

If set to a non-zero value, will halt all AIF2 PE and PD channels except those specified within the Fm_ExcludedResource's list. All AIF2 PE and PD channels will be disabled if the Fm_ExcludedResource's list is NULL.

```
int32_t haltSGMII
```

If set to a non-zero value, will halt the SGMII ethernet switch. **Note:** This will prevent ARM Linux ethernet from working properly.

```
int32_t haltEdma3
```

If set to a non-zero value, will halt all EMDA3 DMA, QDMA, and INT channels except those specified within the Fm_ExcludedResource's list. All EDMA3 DMA, QDMA, and INT channels will be disabled if the Fm_ExcludedResource's list is NULL. **Note:** Some EDMA3 channels are used by ARM Linux to access NAND in UBIFS. UBIFS will not work correctly if these channels are halted.

```
int32_t haltCpdma
```

If set to a non-zero value, will halt all CPDMA channels except those specified within the Fm_ExcludedResource's list. All CPDMA channels will be disabled if the Fm_ExcludedResource's list is NULL. **Note:** Some CPDMA channels are used by ARM Linux which will exhibit unknown behavior if the relevant CPDMA channels are halted.

```
Fm_ExcludedResource *excludedResources;
```

List of specific system IO values that should not be disabled.

A customized version of the Fm_ExcludedResource's list can be created. However, a version has been created and supplied in *pdk_keystone2_#_##_##_##\packages\ti\instrumentation\fault_mgmt\device\k2?\src\fm_device.c* which already accounts for all resources used the ARM Linux kernel delivered with the latest MCSDK 3.x release.

Note: Following the IO halt configuration defined in the FM test source code will diable all IO except that used by ARM Linux UBIFS and NFS.

Informing Remote DSP Core's of Exception

Remote DSP cores can be informed of a DSP exception and told to halt functioning using the following API:

```
void Fault_Mgmt_notify_remote_core(uint32_t core_id);
```

The DSP local to the exception will interrupt remote cores via their NMI causing them to enter their own exception handling routine. This allows all DSPs to be brought down when a single DSP exception occurs. In multi-core application this may help preserve information relevant to deciphering the root cause of the original DSP exception.

Informing Host of DSP Exception

A DSP core can inform the ARM Host an exception has occurred via the following API:

```
void Fault_Mgmt_notify(void);
```

This function will notify the ARM Host an exception has occurred via Remoteproc and should be the last FM API called within the instrumented exception hook function.

Instrumenting a DSP Application with FM Exception APIs

Create and install an exception hook in the DSP application that utilizes the DSP FM APIs.

- In the .cfg (Configuro Script) file of the application add following commands to create a section

```
var devType = "k2?";  /* Replace k2? with the k2 device in use k2e,
k2h, k2k, or k2l */
/* Load and use the Fault Management package */
var Fault_mgmt =
xdc.useModule('ti.instrumentation.fault_mgmt.Settings')
Fault_mgmt.deviceType = devType;

/*
 * The SysMin used here vs StdMin, as trace buffer address is required
for
 * Linux trace debug driver, plus provides better performance.
 */
Program.global.sysMinBufSize = 0x8000;
var System = xdc.useModule('xdc.runtime.System');
var SysMin = xdc.useModule('xdc.runtime.SysMin');
System.SupportProxy = SysMin;
SysMin.bufSize = Program.global.sysMinBufSize;

/* Configure resource table for trace only.
   Note that traceOnly parameter should not
   be set if application is using MessageQ based IPC
   to communicate between cores.
 */
var Resource = xdc.useModule('ti.ipc.remoteproc.Resource');
Resource.loadSegment = Program.platform.dataMemory;
Resource.traceOnly = true;

/* Load the Exception and register a exception hook */
```

```
var Exception = xdc.useModule('ti.sysbios.family.c64p.Exception');
Exception.exceptionHook = '&myExceptionHook';
Exception.enablePrint = true;

/* Add note section for coredump */
Program.sectMap[".note"] = new Program.SectionSpec();
Program.sectMap[".note"] = Program.platform.dataMemory;
Program.sectMap[".note"].loadAlign = 128;
```

- In a source/header file, create a exception hook function as follows

```
/* Fault Management Include File */
#include <ti/instrumentation/fault_mgmt/fault_mgmt.h>

Void myExceptionHook(Void)
{
    uint32_t    i;
    Fm_HaltCfg haltCfg;
    uint32_t    efr_val;

    /* Copy register status into fault management data region for Host
*/
    Fault_Mgmt_getLastRegStatus();

    memset(&haltCfg, 0, sizeof(haltCfg));
    efr_val = CSL_chipReadEFR();

    /* If triggered exception originates from another core through
     * NMI exception don't need to halt processing and notify other
cores
     * since the parent core where the exception originally triggered
via
     * event would notify them.  This eliminates recursive exceptions
*/
    if (!(efr_val & 0x80000000)) {
        /* Halt all processing - Only need to be done on one core */
        haltCfg.haltAif = 1;
        haltCfg.haltCpdma = 1;
#if EXCLUDE_LINUX_RESOURCES_FROM_HALT
        haltCfg.haltSGMII = 0;
        /* EDMA used by kernel to copy data to/from NAND in UBIFS */
        haltCfg.haltEdma3 = 0;
        haltCfg.excludedResources = &linuxResources[0];
#else
        haltCfg.haltSGMII = 1;
        haltCfg.haltEdma3 = 1;
        haltCfg.excludedResources = NULL;
#endif
```

```
        Fault_Mgmt_haltIoProcessing(&fmGblCfgParams, &haltCfg);

        for (i = 0; i < fmGblCfgParams.maxNumCores; i++) {
            /* Notify remote DSP cores of exception - WARNING: This
will generate NMI
             * pulse to the remote DSP cores */
            if (i != CSL_chipReadDNUM()) {
                Fault_Mgmt_notify_remote_core(i);
            }
        }
    }

    /* Notify Host of crash */
    Fault_Mgmt_notify();
}
```

A sample test application is provided in *pdk_keystone2_#_##_##_##\packages\ti\instrumentation\fault_mgmt\test\k2?*. The test application uses the default resource exclusion list provided with FM in *pdk_keystone2_#_##_##_##\packages\ti\instrumentation\fault_mgmt\device\k2?\src*. The default list has been configured to exclude all Linux owned IO from the halting on exception. This allows the Linux kernel to remain operational after DSP exception so that the core dump can be processed.

Note: It is recommended that the IO halt configuration defined within #if EXCLUDE_LINUX_RESOURCES_FROM_HALT be used in addition to halting AIF and CPDMA if Linux must remain active after a DSP exception occurs. This IO halt configuration has been tested with both UBIFS and NFS. The documented configuration shuts down all IO except those needed by Linux to operate, such as EDMA3 (for access to NAND), the SGMII (for Ethernet), and Linux owned CPPI DMAs.

FM Global Configuration Parameters

The Fm_GlobalConfigParams structure informs the IO halt and cleanup features of the system peripheral resource ranges that could not be pulled from CSL. The user should not create their own version of this structure. Instead, the version of the structure provided within *pdk_tci6614_#_##_##_##\packages\ti\instrumentation\fault_mgmt\device\tci6614\src\fm_device.c* should be used. This structure has been statically created based on the TCI6614 device peripheral parameters.

MCSDK UG Chapter Developing Security

Developing with MCSDK: Security

Last updated: **09/20/2015**

Overview

Learn about security features offered in the SDK.

Secure storage for storing long-term secrets

Overview

Secure store is a software representation of a hardware token, which provides facility to securely store and retrieve longterm device secrets in NAND.

This have following features

- The secure store filesystem is encrypted and stored in NAND with a AES128 bit key called *securestore encryption key*
- The *securestore encryption key* is stored in NAND with a separate UBI volume
  - The *securestore encryption key' is stored as a key blob in NAND with a header and checksum to check for integrity*
  - In non-secure device the key blob is stored in plain text
  - In secure device the key is encrypted and signed by boot keys and stored in NAND
- The encrypted and signed secure store file is stored in NAND on two separate UBI volumes to increase reliability
- Secure store services are provided by a daemon process called *softhsm-daemon*
- Application access to secure store is through PKCS#11 APIs provided by the secure store library(*libsecstore.so*)
  - This library serializes PKCS11 API requests and sends to *softhsm-daemon*.
  - The communication between application & daemon is using Unix domain socket IPC.
  - *softhsm-daemon* processes the PKCS11 requests and provides response to the application
  - Applications in this context will be *strongswan*, *libp11* etc.
  - OpenSSL applications like *TLS client/server* & *CMPv2 client* will be accessing secure store services via the OpenSSL engine interface.

Following is the startup sequence wrt. secure store

1. U-Boot reads the *securestore encryption key* blob from NAND UBI volume
   1. In secure device: Authenticated/Decrypted key blob is placed in internal memory
   2. In non-secure device: The key blob is in plain text, and placed in internal memory
2. During Linux kernel boot, the UBIFS secure store volumes are mounted in filesystem using fstab
3. The Linux init script starts softhsm-daemon
4. The softhsm-daemon reads/parses softhsm config file from Linux filesystem for the following
   1. location and size of internal memory
   2. location of *securestore encryption key*
   3. securestore file paths
   4. Unix domain socket names to be used for IPC
5. The softhsm-daemon waits on a Unix domain socket, the socket name is provided in the softhsm config file

Internal memory for secure store

Secure store uses internal memory (MSMC SRAM) to keep the working copy of secure file system. 64KBytes of MSMC SRAM starting from address *0x0c000000* is used by secure store and this memory range must not be used by any other DSP or ARM application. If configured, Secure store can also use internal memory to maintain heap for dynamic memory allocations of crypto library(OpenSSL), in which case it will consume 38KB of memory from the reserved 64KB.

Secure store ubifs volumes

- Following UBIFS volumes used for secure store, the volumes names are as follows
  - *securedbv0*, *securedbv1*: Two volumes to store encrypted and integrity protected filesystem blob
  - Securestore encryption key file blob is present in boot volume
- Sample UBIFS config file is provided in <release_folder>/sc_mcsdk_linux_<version>/images/ubinize.cfg

Creating securestore encryption key blob

A raw text or binary key file can be converted to the format required by secure store using *securestore-formatkey.py* script provided in <release_folder>/sc_mcsdk_linux_<version>/bin file. Note that the key has to be a 128bit AES key.

```
<release_folder>/sc_mcsdk_linux_<version>/bin/securestore-formatkey.py --input
./aeskey.txt --output securedb.key.bin
```

- Note: The u-boot will use "securedb.key.bin" file name to access load the key from volume "boot"
- Please see the script help *<release_folder>/sc_mcsdk_linux_<version>/bin/securestore-formatkey.py --help* for other usage details
- In case of secure board, this file has to be processed further, please see MCSDK-SECDEV user guide for further details

Creating securestore filesystem ubifs volume image

The default UBI volumes (securedbv[0-1].ubifs.img) are provided in the release package. The following sequence of commands will create securestore filesystem image ubifs partition `export PATH=<release_folder>/sc_mcsdk_linux_<version>/bin:$PATH`

rm -rf ./securedbv0 ./securedbv1 mkdir ./securedbv0 mv <path to filesystem blob> ./securedbv0/securedb mkfs.ubifs -r ./securedbv0 -F -o securedbv0.ubifs.img -m 2048 -e 126976 -c 22

mv ./securedbv0 ./securedbv1 mkfs.ubifs -r ./securedbv1 -F -o securedbv1.ubifs.img -m 2048 -e 126976 -c 22

Secure store token initialization

An empty, encrypted & initialized token is present in the default filesystem. In case it is needed to re-initialize the token, it can be done on the target using *softhsm-util* command. The default PINs are *1234*. NOTE: Secure store has a restriction that the SO PIN & User PIN be set the same. `softhsm-util --init-token --slot 0 --label token-0 --module /usr/lib/softhsm/libsecstore.so.1`

softhsm-util --show-slots --module /usr/lib/softhsm/libsecstore.so.1 For more information see help for the tool *softhsm-util --help* NOTE: A reboot of the system is needed after initializing the token.

Using Strongswan with secure storage

Storing credentials

- RSA Key pair
  - RSA key pair to be used by Strongswan during IKE negotiation has to be generated and stored on the secure store.
  - Key generation/storage can be done through OpenSSL command line or via engine API programmatically.
- X.509 Certificate
  - End entity certificate is issued by a certification authority, and needs to be brought into the secure store
  - In a production system this would be done by CMPv2 client.
  - For lab setup
    - Certificate issuance can be done using command line tools provided by Strongswan or OpenSSL.
    - This process will require access to the RSA public key of the end entity. Secure store solution provides an OpenSSL engine command to retrieve the Public key from store.
    - The issued certificate can then be written to secure store using OpenSSL engine commands.
- CA certificate also needs to be in the secure store and can be written using OpenSSL Engine commands.

PKCS#11 plugin configurations

To use the strongswan pkcs#11 plugin, PKCS#11 module has to be configured in */etc/strongswan.conf*.

Configuration Keys specific to pkcs#11 plugin are explained in the PKCS11Plugin wiki [1].

Default configuration provided will be: `libstrongswan { plugins {`

```
   pkcs11 {
     modules {
       secstore {
         path = /usr/lib/softhsm/libsecstore.so.1
       }
     }
   }
 }
```

}

PIN configurations

Soft token will have a static PIN. Following wiki explains the format for configuring a static PIN in the file */etc/ipsec.secrets* PIN Secret [2].

Private Key configuration

The RSA private key to be used from the smartcard is specified as part of the PIN configuration with the format: *%smartcard[<slotnr>[@<module>]]:<keyid>* e.g.

```
: PIN %smartcard0@secstore:04 1234
```

Certificate configuration

Certificates stored on smart cards will get loaded automatically when the IKEv2 daemon is started. You don't have to specify *leftcert=%smartcard* in ipsec.conf (it actually will fail if you do so). Instead the first certificate matching the “leftid” parameter is used.

CA certificates are also automatically available as trust anchors without the need to copy them into the */etc/ipsec.d/cacerts/* directory.

```
conn alice
```

```
     left=158.218.103.42
     leftprotoport=udp/5000
     #leftcert=bobCert.der
     leftid="C=US, O=TestInc, CN=bob"
     right=158.218.103.108
     rightid="C=US, O=TestInc, CN=alice"
     rightprotoport=udp/5000
     keyexchange=ikev2
     type=tunnel
```

```
lifetime=24h auto=start
```

PKCS11 engine commands for OpenSSL

Following commands are provided by the OpenSSL engine (engine_pkcs11) to interface with secure storage

- SO_PATH: Specifies the path to the 'pkcs11-engine' shared library
- MODULE_PATH: Specifies the path to the pkcs11 module shared library
- PIN: Specifies the pin code
- VERBOSE: Print additional details
- QUIET: Remove additional details
- INIT_ARGS: Specifies initialization arguments to the pkcs11 module
- LIST_OBJS: List the objects from token
- STORE_CERT: Store X.509 certificate to token (DER format)
- GEN_KEY: Generate & store RSA key pair to token
- DEL_OBJ: Delete objects from token
- GET_PUBKEY: Get Public Key from token (PEM format)

Following is the listing of these commands

- Command to load the engine dynamically and set the PIN

  > *engine -vvvv dynamic -pre SO_PATH:/usr/lib/engines/engine_pkcs11.so -pre ID:pkcs11 -pre LIST_ADD:1 -pre LOAD -pre MODULE_PATH:/usr/lib/softhsm/libsecstore.so.1 -pre "VERBOSE" -pre "PIN:1234"*

- Command to enable verbose

  > *engine pkcs11 -pre "VERBOSE"*

- Command to set the PIN

  > *engine pkcs11 -pre "PIN:1234"*

- Command to list the objects on slot 0

  > *engine pkcs11 -pre "LIST_OBJS:0"*

- Command to store certificate to token

engine pkcs11 -pre "STORE_CERT:slot_<slot>:id_<id>:label_<label>:cert_<filename>"

e.g.

engine pkcs11 -pre "STORE_CERT:slot_0:id_03:label_tcert:cert_caCert.der"

- Command to generate a RSA key pair

  engine pkcs11 -pre "GEN_KEY:slot_<slot>:size_<size>:id_<id>:label_<label>"

  where <size> is the RSA key size in bits

  e.g.

  engine pkcs11 -pre "GEN_KEY:slot_0:size_2048:id_04:label_tkey"

- Command to get the Public key from token

  engine pkcsc11 -pre "GET_PUBKEY:slot_<slot>:id_<id>:label_<label>:key_<filename>"

  e.g.

  engine pkcs11 -pre "GET_PUBKEY:slot_0:id_04:label_tkey:key_tkey.pem"

- Command to delete an object from the token

  engine pkcs11 -pre "DEL_OBJ:slot_<slot>:type_<type>:id_<id>:label_<label>"

  where <type> is the object type (privkey/pubkey/cert)

  e.g. to delete certificate

  engine pkcs11 -pre "DEL_OBJ:slot_0:type_cert:id_03:label_tcert"

  e.g. to delete a public key

  engine pkcs11 -pre "DEL_OBJ:slot_0:type_pubkey:id_04:label_tkey"

  e.g. to delete a private key

  engine pkcs11 -pre "DEL_OBJ:slot_0:type_privkey:id_04:label_tkey"

MCSDK UG Chapter Downloads

MCSDK User Guide: Downloads and Product Add-Ons

Last updated: **09/20/2015**

Overview

This chapter provides information to get to the latest version of various software. For installation information, see the Getting Started [1] chapter of the User Guide.

Product Downloads

Useful Tip

To get to the download page for the latest KeyStone II MCSDK release, click here [1].

To get the latest release, please go to the **MCSDK** product folder at MCSDK Product Page [2]. For KeyStone II software, click on the "Get Software" tab for the "BIOSLINUXMCSDK-K2" row. The download page contains installers and links to dependencies, such as Code Composer Studio and Emulator software. Please review the release notes and software manifest before downloading and/or installing the software.

Note: The EVM comes with media (DVD/SD card) containing the MCSDK, CCS, and Emupack software. You can start with these or go to the MCSDK software download site listed above to check for the latest updates and version.

Related Software

This section provides a collection links to additional software elements that may be of interest.

Link	Description
Security Accelerator LLD [3]	Download page for Security Accelerator (SA) low level driver for MCDSK 3.0.X releases
Security Accelerator 3GPP Enabler LLD [4]	Download page for SA 3GPP Enabler designed to be used along with SA LLD 3.x package, delivered as part of MCSDK 3.1.0 (or higher) to enable 3GPP functionalities
C6x DSP Linux Project [40]	Community site for C6x DSP Linux project
Telecom Libraries [41]	TI software folder for information and download of Telecom Libraries (Voice, Fax, etc) for TI processors.
c66x Speech and Video Codecs [5]	TI software folder for information and download of Speech and Video codecs for c66x.
Medical Imaging Software Tool Kits [42]	TI software folder for information and download of medical imaging software tool kits for TI processors.
c6x Software Libraries [74]	Mediawiki providing an overview of available software libraries for TI's c6x family of DSP processors.
Multicore Video Infrastructure Demonstration Application [6]	TI software folder for information and download of multicore video infrastructure demonstration application using the BIOS-MCSDK.
Keystone 2 Emupack [7]	TI software folder for KeyStone II emupack installers (Gel files and XML files) for CCS.

How To

How do I speed up downloading the installer?

The size of the installer is large since we want to provide one bundle for all the components. The bad side of this is that if you are manually downloading the MCSDK (or CCS) installer, you may run into issues such as download stall or slow download. One simple solution is to run a download manager/accelerator. One open source solution is http://www.freedownloadmanager.org/.

MCSDK UG Chapter Support

MCSDK User Guide: Support and Other Resources

Last updated: **09/20/2015**

Overview

This Chapter has links for technical support, current bug listings, online training, white papers, additional documents and other similar resources.

Technical Support

For technical discussions and issues, please visit

- **KeyStone Multicore forum**: http://e2e.ti.com/support/dsp/c6000_multi-core_dsps/f/639.aspx

For local support in China, please visit

- **China Support forum**: http://www.deyisupport.com

Note: When asking for help in the forum you should tag your posts in the Subject with “MCSDK”, the part number (e.g. “66AK2H12”) and additionally the component (e.g. “MPM”).

Training

This section provides a collection links to training resources.

Link	Description
MCSDK Overview Online [1]	This video training module provides an overview of the multicore SoC software for KeyStone devices. This module introduces the optimized software components that enable the rapid development of multicore applications and accelerate time to market using foundational software in the MCSDK. The MCSDK also enables developers to evaluate the hardware and software capabilities using the KeyStone evaluation module. The **Mandarin** version of this training can be found here [2].
KeyStone II Architecture Wiki [3]	KeyStone II Architecture Overview mediawiki page.
KeyStone II Architecture Online [4]	KeyStone II SOC Online Training. Topics include: • KeyStone II SoC Overview provides a high-level view of the device architecture, including the C66x and ARM Cortex-A15 processors, memory and transport topologies, networking and interface enhancements, as well as power saving and debug features for the KeyStone II family of C66x multicore devices. • KeyStone II Software Overview describes the software development ecosystem for the KeyStone II family of C66x multicore devices from both a Linux and SYS/BIOS perspective. • KeyStone II ARM Cortex-A15 Corepac Overview introduces the implementation of the ARM Cortex-A15 MPCore processor in the KeyStone II family of C66x multicore devices.
SYS/BIOS Online [61]	SYS/BIOS online training.
SYS/BIOS 1.5 Day [62]	SYS/BIOS 1.5-day workshop.
MCSA Online [5]	Multicore System Analyzer online tutorial.

EVM documentation and support

- EVMK2H: http://www.advantech.com.tw/Support/TI-EVM/EVMK2HX.aspx
- EVMK2E: https://www.einfochips.com/index.php/partnerships/texas-instruments/k2e-evm.html
- EVMK2L: https://www.einfochips.com/index.php/partnerships/texas-instruments/k2l-evm.html

Article Sources and Contributors

MCSDK User Guide for KeyStone II *Source*: http://processors.wiki.ti.com/index.php?oldid=200277 *Contributors*: Alexander.stohr, Dharik, Frankfruth, JackM, RajSivarajan, Samsnelson

MCSDK UG Chapter Getting Started *Source*: http://processors.wiki.ti.com/index.php?oldid=201928 *Contributors*: A0216693, A0271519, A0850409, A0850461, A0870196, Cesar, Csmith, Cwagner, Frankfruth, Gurnani, Hao, JackM, Mkaricheri, Mscherban, RajSivarajan, Samsnelson, Sandeeppaulraj, Vitalya

EVMK2H Hardware Setup *Source*: http://processors.wiki.ti.com/index.php?oldid=165602 *Contributors*: A0271519, A0870196, Bigtexan1, Cesar, Csmith, Cwagner, Dharik, Hao, Mkaricheri, RajSivarajan, Randyp, Samsnelson

EVMK2E Hardware Setup *Source*: http://processors.wiki.ti.com/index.php?oldid=184411 *Contributors*: RajSivarajan

TCIEVMK2L Hardware Setup *Source*: http://processors.wiki.ti.com/index.php?oldid=206633 *Contributors*: Cwagner, Mscherban, RajSivarajan

Program EVM UG *Source*: http://processors.wiki.ti.com/index.php?oldid=188354 *Contributors*: A0850409, A0850461, Csmith, Fahrizaqeel, Hao, RajSivarajan, Samsnelson

MCSDK UG Chapter Tools *Source*: http://processors.wiki.ti.com/index.php?oldid=181166 *Contributors*: A0271519, Dharik, Hao, JackM, RajSivarajan, Zhoujt

MCSDK UG Chapter Exploring *Source*: http://processors.wiki.ti.com/index.php?oldid=206618 *Contributors*: A0216521, A0216552, A0271519, A0272049, A0850461, A0870196, Alyytinen, AravindBatni, Bazbell, Cesar, ChrisRing, Csmith, Fahrizaqeel, Hao, JackM, Justin32, M-karicheri2, Marcshaffer, Merdahl, Mkaricheri, Mscherban, Raghunambiath, RajSivarajan, Sajeshsaran, Samsnelson, Sandeepnair, Sandeeppaulraj, Shankari G, Tmannan, Vitalya, Wingmankwok, Wmat, X0183204

MCSDK Utility App Demonstration Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=167887 *Contributors*: Hao, RajSivarajan

MCSDK Image Processing Demonstration Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=206560 *Contributors*: A0792105, Catheyri, ChrisRing, Csmith, DanRinkes, Eric, Jbtheou, Mdamato, RajSivarajan, Sajeshsaran, Wmat

MCSDK Inter Processor Communication Demonstration Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=179030 *Contributors*: RajSivarajan, Samsnelson

MCSDK UG Chapter Developing *Source*: http://processors.wiki.ti.com/index.php?oldid=195269 *Contributors*: AravindBatni, JackM, RajSivarajan, SajeshSaran, Tmannan

MCSDK UG Chapter Developing Migration *Source*: http://processors.wiki.ti.com/index.php?oldid=206146 *Contributors*: A0850461, AravindBatni, Hao, Ipang, Justin32, Mkaricheri, RajSivarajan, Samsnelson, Vitalya

MCSDK UG Chapter Developing KS II First App *Source*: http://processors.wiki.ti.com/index.php?oldid=183423 *Contributors*: A0850461, JackM, RajSivarajan, Samsnelson

MCSDK UG Chapter Developing PDK *Source*: http://processors.wiki.ti.com/index.php?oldid=193506 *Contributors*: A0271519, AravindBatni, Eric Ding, JackM, Justin32, Raghunambiath, RajSivarajan, Sajeshsaran, Samsnelson

MCSDK UG Chapter Developing Debug Trace *Source*: http://processors.wiki.ti.com/index.php?oldid=170348 *Contributors*: A0850461, AravindBatni, JackM, RajSivarajan, Zhoujt

MCSDK UG Chapter Developing Transports *Source*: http://processors.wiki.ti.com/index.php?oldid=206561 *Contributors*: ChrisRing, Ipang, JackM, Jstiffler, Justin32, RajSivarajan, SajeshSaran, Sajeshsaran

MCSDK UG Chapter Developing System Mgmt *Source*: http://processors.wiki.ti.com/index.php?oldid=204015 *Contributors*: A0850461, Dharik, JackM, JohnDowdal, Justin32, RajSivarajan, SajeshSaran, Sajeshsaran, Samsnelson

MCSDK UG Chapter Developing Fault Mgmt *Source*: http://processors.wiki.ti.com/index.php?oldid=183395 *Contributors*: JackM, Justin32, RajSivarajan

MCSDK UG Chapter Developing Security *Source*: http://processors.wiki.ti.com/index.php?oldid=195266 *Contributors*: Alyytinen, JackM, RajSivarajan

MCSDK UG Chapter Downloads *Source*: http://processors.wiki.ti.com/index.php?oldid=206566 *Contributors*: A0850461, AravindBatni, Csmith, JackM, Lalindra, RajSivarajan, Shanmuga

MCSDK UG Chapter Support *Source*: http://processors.wiki.ti.com/index.php?oldid=190185 *Contributors*: Csmith, JackM, RajSivarajan, Shanmuga

Image Sources, Licenses and Contributors

Image:KeyStone Multicore DSP ARM.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:KeyStone_Multicore_DSP_ARM.jpg *License*: unknown *Contributors*: JackM

Image:McsdkUGGS NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGGS_NT.jpg *License*: unknown *Contributors*: JackM

Image:McsdkUGTOOLS NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGTOOLS_NT.jpg *License*: unknown *Contributors*: JackM

Image:McsdkUGSUPPORT NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGSUPPORT_NT.jpg *License*: unknown *Contributors*: JackM

Image:MCSDKUGDEVELOPING NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:MCSDKUGDEVELOPING_NT.jpg *License*: unknown *Contributors*: JackM

Image:McsdkUGEXPLORING NT .jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGEXPLORING_NT_.jpg *License*: unknown *Contributors*: JackM

Image:McsdkUGDOWNLOAD NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGDOWNLOAD_NT.jpg *License*: unknown *Contributors*: JackM

Image:ti_hz_2c_pos_rgb_jpg.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Ti_hz_2c_pos_rgb_jpg.jpg *License*: unknown *Contributors*: RajSivarajan

File:McsdkUGGS_NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGGS_NT.jpg *License*: unknown *Contributors*: JackM

File:Helpful_tips_image.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Helpful_tips_image.jpg *License*: unknown *Contributors*: DanRinkes, PagePusher

Image:ccs-install-processor.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Ccs-install-processor.jpg *License*: unknown *Contributors*: RajSivarajan

Image:ccs-install-components.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Ccs-install-components.jpg *License*: unknown *Contributors*: RajSivarajan

Image:matrix_apps1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Matrix_apps1.png *License*: unknown *Contributors*: Hao

Image:matrix_apps2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Matrix_apps2.png *License*: unknown *Contributors*: Hao

File:ENET0.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:ENET0.jpg *License*: unknown *Contributors*: A0870196

Image:Bmc ver screenshot.JPG *Source*: http://processors.wiki.ti.com/index.php?title=File:Bmc_ver_screenshot.JPG *License*: unknown *Contributors*: Bigtexan1

Image:K2EVM.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:K2EVM.jpg *License*: unknown *Contributors*: Bigtexan1

Image:evmk2e-image001.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2e-image001.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2e-image002.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2e-image002.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2e-image003.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2e-image003.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2e-image005.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2e-image005.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2e-image006.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2e-image006.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2e-image007.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2e-image007.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2l-image001.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2l-image001.png *License*: unknown *Contributors*: RajSivarajan

Image:evmk2l-image002.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2l-image002.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2l-image004.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2l-image004.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2l-image005.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2l-image005.jpg *License*: unknown *Contributors*: RajSivarajan

Image:evmk2l-image006.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Evmk2l-image006.jpg *License*: unknown *Contributors*: RajSivarajan

File:Tera_Term_Connect.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Tera_Term_Connect.jpg *License*: unknown *Contributors*: Csmith

File:Spl-boot.j.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Spl-boot.j.jpg *License*: unknown *Contributors*: Mkaricheri

File:Kernel_Boot.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Kernel_Boot.jpg *License*: unknown *Contributors*: Csmith

File:McsdkUGTOOLS_NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGTOOLS_NT.jpg *License*: unknown *Contributors*: JackM

Image:tci6638_ccxml_1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Tci6638_ccxml_1.png *License*: unknown *Contributors*: Vitalya

Image:uboot_tci6638_2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Uboot_tci6638_2.png *License*: unknown *Contributors*: Vitalya

Image:bmc-console.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Bmc-console.jpg *License*: unknown *Contributors*: Mkaricheri

Image:spl-boot.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Spl-boot.jpg *License*: unknown *Contributors*: Mkaricheri

Image:tci6638_evm2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Tci6638_evm2.png *License*: unknown *Contributors*: Hao

Image:Ti hz 2c pos rgb jpg.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Ti_hz_2c_pos_rgb_jpg.jpg *License*: unknown *Contributors*: RajSivarajan

Image:800px-MCSDK-KSII-picture.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:800px-MCSDK-KSII-picture.jpg *License*: unknown *Contributors*: JackM

Image:Tci6638 ccxml 1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Tci6638_ccxml_1.png *License*: unknown *Contributors*: Vitalya

Image:Uboot tci6638 1.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Uboot_tci6638_1.png *License*: unknown *Contributors*: Vitalya

Image:Uboot tci6638 2.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Uboot_tci6638_2.png *License*: unknown *Contributors*: Vitalya

Image:Loadlin-evm.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Loadlin-evm.jpg *License*: unknown *Contributors*: Vitalya

Image:Boot-kernel-arch.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Boot-kernel-arch.jpg *License*: unknown *Contributors*: Mkaricheri

Image:Netstack.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Netstack.png *License*: unknown *Contributors*: Sandeeppaulraj

Image:Sgmii.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Sgmii.png *License*: unknown *Contributors*: Sandeeppaulraj

Image:Qos-tree.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Qos-tree.jpg *License*: unknown *Contributors*: Sandeeppaulraj

Image:Qos-new-shaper.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Qos-new-shaper.jpg *License*: unknown *Contributors*: Sandeeppaulraj

Image:Shaper-config-details.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Shaper-config-details.jpg *License*: unknown *Contributors*: Sandeeppaulraj

Image:Keystone2 csl lld migration.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Keystone2_csl_lld_migration.jpg *License*: unknown *Contributors*: RajSivarajan

Image:Ndkarch.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ndkarch.png *License*: unknown *Contributors*: DanRinkes, Sajeshsaran

Image:matrix_apps3.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Matrix_apps3.png *License*: unknown *Contributors*: Hao

Image:Inputpage.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Inputpage.jpg *License*: unknown *Contributors*: Sajeshsaran

Image:Outputpage.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Outputpage.jpg *License*: unknown *Contributors*: Sajeshsaran

Image:Ctoolslib system latency.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_system_latency.png *License*: unknown *Contributors*: Csmith

Image:Ctoolslib system bandwidth.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_system_bandwidth.png *License*: unknown *Contributors*: Csmith

Image:Ctoolslib master bandwidth.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_master_bandwidth.png *License*: unknown *Contributors*: Csmith

Image:Ctoolslib event profiler.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_event_profiler.png *License*: unknown *Contributors*: Csmith

Image:Ctoolslib mem pct watchpoint.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_mem_pct_watchpoint.png *License*: unknown *Contributors*: Csmith

Image:Ctoolslib mem aetint watchpoint.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_mem_aetint_watchpoint.png *License*: unknown *Contributors*: Csmith

Image:Matrix_demo_mcsdk.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Matrix_demo_mcsdk.png *License*: unknown *Contributors*: Samsnelson

Image:Demo_submenu.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Demo_submenu.png *License*: unknown *Contributors*: Samsnelson

Image:IPC Demo.png *Source*: http://processors.wiki.ti.com/index.php?title=File:IPC_Demo.png *License*: unknown *Contributors*: Samsnelson

Image:IPC run results.png *Source*: http://processors.wiki.ti.com/index.php?title=File:IPC_run_results.png *License*: unknown *Contributors*: Samsnelson

File:MCSDKUGDEVELOPING_NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:MCSDKUGDEVELOPING_NT.jpg *License*: unknown *Contributors*: JackM

File:KeyStoneII_CSL_LLD_Migration_MCSDK_3_0_To_3_1.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:KeyStoneII_CSL_LLD_Migration_MCSDK_3_0_To_3_1.jpg *License*: unknown *Contributors*: RajSivarajan

Image:TraceFramework DSP.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:TraceFramework_DSP.jpg *License*: unknown *Contributors*: Zhoujt

File:Ctoolslib_uclib_bitmap.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Ctoolslib_uclib_bitmap.png *License*: unknown *Contributors*: A0272312

Image:SRIO_IPC_Trans_Arch.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:SRIO_IPC_Trans_Arch.jpg *License*: unknown *Contributors*: Justin32
Image:Linux_TransSrio_Arch.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Linux_TransSrio_Arch.jpg *License*: unknown *Contributors*: Justin32
Image:DLSDK Mailbox_A.png *Source*: http://processors.wiki.ti.com/index.php?title=File:DLSDK_Mailbox_A.png *License*: unknown *Contributors*: Samsnelson
File:mpm-transport.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Mpm-transport.jpg *License*: unknown *Contributors*: Ipang, SajeshSaran
File:mpm-transport-put-initiate64.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Mpm-transport-put-initiate64.jpg *License*: unknown *Contributors*: Ipang
File:dspstate.png *Source*: http://processors.wiki.ti.com/index.php?title=File:Dspstate.png *License*: unknown *Contributors*: Sajeshsaran
Image:DSP_crash_stack.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:DSP_crash_stack.jpg *License*: unknown *Contributors*: SajeshSaran
Image:RM_inst_multi_core.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_inst_multi_core.jpg *License*: unknown *Contributors*: Justin32
Image:RM_inst_multi_task.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_inst_multi_task.jpg *License*: unknown *Contributors*: Justin32
Image:RM_inst_multi_dsp.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_inst_multi_dsp.jpg *License*: unknown *Contributors*: Justin32
Image:RM_general_interfaces.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_general_interfaces.jpg *License*: unknown *Contributors*: Justin32
Image:RM_name_synchronization.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_name_synchronization.jpg *License*: unknown *Contributors*: Justin32
Image:RM_grl_policy_format.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_grl_policy_format.jpg *License*: unknown *Contributors*: Justin32
Image:RM_dtc_dir.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_dtc_dir.jpg *License*: unknown *Contributors*: Justin32
Image:RM_conversion_syntax_error.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_conversion_syntax_error.jpg *License*: unknown *Contributors*: Justin32
Image:RM_dump_fruit_GRL.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_dump_fruit_GRL.jpg *License*: unknown *Contributors*: Justin32
Image:RM_fruit_GRL.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_fruit_GRL.jpg *License*: unknown *Contributors*: Justin32
Image:RM_fruit_policy.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_fruit_policy.jpg *License*: unknown *Contributors*: Justin32
Image:RM_fruit_policy_static.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_fruit_policy_static.jpg *License*: unknown *Contributors*: Justin32
Image:RM_shared_inst.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_shared_inst.jpg *License*: unknown *Contributors*: Justin32
Image:RM_packet_diff.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_packet_diff.jpg *License*: unknown *Contributors*: Justin32
Image:RM_packet_routing.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_packet_routing.jpg *License*: unknown *Contributors*: Justin32
Image:RM_direct_indirect.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:RM_direct_indirect.jpg *License*: unknown *Contributors*: Justin32
Image:rmserverhelp.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:Rmserverhelp.jpg *License*: unknown *Contributors*: Justin32
File:McsdkUGDOWNLOAD_NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGDOWNLOAD_NT.jpg *License*: unknown *Contributors*: JackM
File:McsdkUGSUPPORT_NT.jpg *Source*: http://processors.wiki.ti.com/index.php?title=File:McsdkUGSUPPORT_NT.jpg *License*: unknown *Contributors*: JackM