NXP Semiconductors
User's Guide

Document Number: LS1028ABSPUG
Rev. 0.3, 04/2019

Layerscape LS1028A BSP User Guide

Supports BSP v0.3

Contents

Chapter 1 Introduction

About QorIQ LS1028A BSP

LS1028A BSP is a complete Linux kit for NXP LS1028A SoC and its reference and evaluation boards.

It is a *hybrid form* of a Linux distribution because it combines the following major components to form a complete Linux system.

- NXP boot loader: U-Boot, based on denx.de plus patches
- NXP Linux kernel, based on kernel.org upstream plus patches.
- NXP added user space components.
- Ubuntu standard user space file set (user land), including compilers and cross compiler.

The use of Ubuntu user land is what makes LS1028A BSP a hybrid. It is not entirely an Ubuntu distribution because it uses an NXP kernel, but it still uses Ubuntu user space files. This hybrid is possible because NXP ARM SoC's are standards-based so programs like bash and thousands of others run without being recompiled.

The benefit of using Ubuntu user land is the easy availability of thousands of standard Linux user space packages. The experience of using the LS1028A BSP is similar to using Ubuntu, but the kernel, firmware, and some special NXP packages are managed separately.

Accessing LS1028A BSP

LS1028A BSP is distributed via nxp.com.

There are two ways to use the LS1028A BSP, as an integration and as a source of individual components.

LS1028A BSP as an integration

Using the link above, notice the `flexbuild` component. You can clone it and run a script to create and install LS1028A BSP onto a mass storage device as an integration, ready for use on an NXP reference or evaluation board. You can build NXP components from source using a script called flex-builder or install from binaries of NXP components using flex-installer.

LS1028A BSP as components

The same link shows git repositories for individual components, for example the LS1028A BSP Linux kernel. If you clone and examine this git, you will see a conventional kernel source tree. You can compile the kernel using `make` in the normal way, like a kernel.org kernel. However, notice the configuration fragment in a`rch/arm64/configs`. See Linux kernel on page 37.

Having git access to components is helpful if you assemble your own Linux distribution or wish to form a hybrid with a user land other than Ubuntu's.

LS1028A BSP git tags

LS1028A BSP git repositories use git tags to indicate component revisions that have been release tested together. Use the `git tag` command to examine them and choose a tag to check out.

LS1028A BSP Relies on Mass Storage Devices

Ubuntu user land is very convenient for evaluation because it is possible to use the command `apt-get install` on the standard Ubuntu components you need. It also provides native development tools.

But this richness means that the user space file is large, too large for RAM disks.

Therefore, LS1028A BSP requires installation to and use of a mass storage device such as

- SD card
- USB flash drive
- USB hard drive

- SATA drive, spinning, or SSD (for boards with a SATA port)
- eMMC flash (when available on board)

LS1028A BSP provides scripts that populate a mass storage device with the needed files. These scripts can run on a Linux PC. It is especially simple to use an SD card or USB flash drive because they are the easiest to move between a Linux PC and the NXP board.

1.1 Reference documentation

The table below lists and explains the additional documents and resources that you can refer to for more information on the LS1028A SoC and LS1028A board (RDB and QDS).

Some of the documents listed below may be available only under a non-disclosure agreement (NDA). To request access to these documents, contact your local field applications engineer or sales representative.

Table 1. Reference documentation

Document	Description	Link/How to access
QorIQ LS1028A Reference Manual	Provides a detailed description about the LS1028A QorIQ multicore processor and its features, such as memory map, serial interfaces, power supply, chip features, and clock information.	Contact FAE / sales representative
QorIQ LS1028A Reference Design Board Getting Started Guide	Explains the LS1028ARDB settings and physical connections needed to boot the board.	QorIQ LS1028A Reference Design Board Getting Started Guide
QorIQ LS1028A Reference Design Board Reference Manual	Provides detailed information about LS1028ARDB interfaces, power supplies, clocks, DIP switches, LEDs, and CPLD system controller.	QorIQ LS1028A Reference Design Board Reference Manual
QorIQ LS1028A Development System Getting Started Guide	Explains the LS1028AQDS settings and physical connections needed to boot the board.	Contact FAE / sales representative
QorIQ LS1028A Development Sytem Reference Manual	Provides detailed information about LS1028AQDS interfaces, power supplies, clocks, DIP switches, LEDs, and CPLD system controller.	Contact FAE / sales representative

Chapter 2
Release notes

2.1 What's New

What's new in LS1028A BSP v0.3

- Boot flow changed from PPA to Arm Trusted Firmware (TF-A)
- L2 switch support in U-Boot
- PF level MDIO support in ENETC
- SerDes related updates
- Performance enhancements for ENETC and IPSec
- New switch SW architecture
- IEEE 1588 support in ENETC and Felix
- LPUART support in U-Boot
- Support of 4-bit mode for SD and eMMC
- Flextimer support in Linux
- Performance monitor support in Linux
- Gstreamer video player support in user space
- Run-time detection of display monitor’s resolution

2.2 Summary of overall features

Highlights

- Processor support
  - — LS1028A processor
- Board support:
  - — LS1028ARDB board
  - — LS1028AQDS board
- Frequency support:
  - — Core: 1300 MHz, DDR: 1600 MT/s, Platform: 400 MHz [default]
  - — Core: 800 MHz, DDR: 1300 MT/s, Platform: 400 MHz
- SerDes protocol support:
  - — SerDes1: 0x85bb [default]
  - — SerDes2: 0x85be
- Multimedia and audio support:
  - — Resolution supported: 480p,720p,1080p, and 4K
  - — Support audio on serial interfaces with frame synchronization

- Time Sensitive Network (TSN) support:
  - — TSN configuration tool (tsntool)
  - — ENETC 1588 two steps timestamping support
  - — ENETC TSN driver: Qbv, Qbu, Qci, Qav
  - — SWITCH TSN driver: Qbv, Qci, Qbu, Qav, 802.1CB support
- **U-Boot: 2018.09**
  - — U-Boot image includes the device tree
  - — Non-secure
  - — Boot from FlexSPI NOR flash, SD
  - — A72 core, Timer, UART
  - — Clock, FPGA/CPLD, UART, DDR4
  - — eSDHC, eMMC, GIC, I2C, OCRAM
  - — PCIe-gen1-rootcomplex, USB, SATA
  - — FlexSPI access to NOR flash
  - — Networking support using ENETC
  - — MDIO PHY support
  - — eDP/DP firmware loading

Linux: 4.14.47

- — A72 core, Timer
- — SMP-boot
- — Clock, UART, DDR4
- — DSPI [QDS board only]
- — eSDHC, eMMC, GIC, I2C, OCRAM
- — USB, SATA
- — FlexSPI access to NOR flash
- — PCIe-gen3-rootcomplex, USB, SATA
- — Networking interfaces: ENETC, L2Switch
- — ENETC PCIe PF and VF full featured Linux ethernet drivers and TSN driver
- — L2Switch features: Switchdev support and TSN driver
- — SEC, QDMA
- — System Memory Management Unit (SMMU)
- — MDIO PHY support
- — Mutimedia IPs - GPU, LCD, eDP/DP
- — Integrated Interchip Sound (I2S) / Synchronous Audio Interface (SAI)
- — eDMA, CAN

Userspace components

- — TSNtool for configuring ENETC-TSN and L2Switch
- — OpenCL and OpenGL testcases and applications

— Flexbuild and Toolchain

Flexbuild

— Ubuntu userland 18.04 update

— gcc: Ubuntu/Linaro 7.3.0-16ubuntu3~18.04, glibc-2.27, binutils-2.30-0, gdb-8.1

2.3 Component Location

The below table lists the component location.

Table 2. Component location

Component	CAF/github location	commit/tag
linux	https://source.codeaurora.org/external/qoriq/qoriq-components/linux/	ls1028a-early-access-bsp0.3
U-Boot	https://source.codeaurora.org/external/qoriq/qoriq-components/u-boot/	ls1028a-early-access-bsp0.3
rcw	https://source.codeaurora.org/external/qoriq/qoriq-components/rcw/	ls1028a-early-access-bsp0.3
cst	https://source.codeaurora.org/external/qoriq/qoriq-components/cst/	ls1028a-early-access-bsp0.3
atf	https://source.codeaurora.org/external/qoriq/qoriq-components/atf/	ls1028a-early-access-bsp0.3
dp-firmware	https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_01/ls1028a-dp-fw.bin	NA
tsntool	https://source.codeaurora.org/external/qoriq/qoriq-components/tsntool/	ls1028a-early-access-bsp0.3
libdrm	https://source.codeaurora.org/external/imx/libdrm-imx/	rel_imx_4.9.123_2.3.0_8mm_ga
wayland	https://github.com/wayland-project/wayland	1.16.0
wayland-protocol	https://source.codeaurora.org/external/imx/wayland-protocols-imx/	rel_imx_4.9.123_2.3.0_8mm_ga
gpu-viv6	https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_01/ls1028a-gpu-viv-6.2.4.p2-aarch64.bin	NA
weston	https://source.codeaurora.org/external/imx/weston-imx/	rel_imx_4.9.123_2.3.0_8mm_ga
flexbuild	Click here	NA

2.4 Feature Support Matrix

The following tables show the features that are supported in this release.

Feature	Description
FlexSPI	• Read/Write/Erase • Read in AHB mode with Octal command support • Write in IP mode • Erase size: 128KB (u-boot) • Supported Flash device: MT35XU02G
UART	• UART1, UART2 verified. • Default frequency: 115.2 kbps
DDR	• Fixed Settings (RDB) • DDR SPD support (QDS) • Default frequency: 1600 MT/s
ENETC	• Support for PF and VF ENETC PCIe Integrated Endpoints • Core transmit and receive side packet processing, PF and VF • Fast Rx buffer management with paged allocation and recycling • Multiple Rx and Tx queues supported, separate MSI-X per CPU • Scather/Gather and Jumbo frame (up to 9600B) support, Rx & Tx • MAC filtering, VLAN insertion/ extraction and VLAN filtering • Checksum offload: CHECKSUM_COMPLETE on Rx (INET CSUM) • Rx Frame Distribution (RSS) and Frame Classification (RFS) • PCIe SRIOV support to configure ENETC Virtual Functions (VFs)
L2Switch	• Data transfer through switch ports • Switchdev support • TSN Qci, Qbv, Qbu, Qav, 802.1CB support
ENETC-TSN	• 1588 two steps time stamping support • Qbv, Qbu, Qci, Qav protocol support

Table continues on the next page...

Table continued from the previous page...

Multimedia	• LCD controller: — Supports resolutions of: 720x480p60, 1280x720p60, 1920x1080p60, and 3840x2160p60 • Display Transmitter Controller – DisplayPort — Supports up to four lanes — Supports hot plug — Supports full link training • Graphics Processing Unit (GPU): — Built-in kernel driver for GPU — Supports the following graphics APIs in Linux user space: ◦ OpenGL ES 1.1, 2.0, 3.0, 3.1 ◦ EGL 1.4 ◦ OpenGL 2.x ◦ OpenVG 1.1 ◦ OpenCL 1.1, 1.2 • Integrated Interchip Sound (I2S) / Synchronous Audio Interface (SAI): — Supports serial interfaces with frame synchronization, such as I2S and codec interfaces
LPUART	Read/write support in U-Boot
Flextimer	Support in Linux
Performance monitor	Perf events support in Linux
CAN	Tx/Rx support in Linux

2.5 Open, Fixed and Known Issues

This section contains Fixed and Open issue tables:

- Table2: Fixed/closed issues: Contains Issues which have a software fix that has been integrated into this Release or the issues are root cause and fix is outside the scope of this release.
- Table3: Open/known issues: Contains Issues which have do not currently have a resolution. Workaround suggestions are provided wherever possible.

Table 3. Fixed issues

ID	Description	Status
QLINUX-10979	LS1028ARDB - flextimer: no 'ftm_alarm' under /sys/devices/platform/soc/2800000.ftm0/	Fixed

Table continues on the next page...

Table 3. Fixed issues (continued)

QLINUX-10857	LS1028ARDB GPU: few demos such as OpenGLES (for example, tutorial4_es20) and OpenVG (for example, tiger) hang	Fixed
QLINUX-11094	LS1028A: There is IOMMU call trace when running 802.1AS with linuxptp	Fixed
QUBOOT-4813	LS1028AQDS - DSPI does not work under U-Boot	Fixed
QLINUX-11091	LS1028ARDB: Linux IPv4 Forward Performance with switch port is much lower than PRL	Fixed
QLINUX-11152	LS1028ARDB: Kernel does not reboot after kill watchdog process	Fixed
QLINUX-11391	LS1028ARDB: System hangs when add switch port to bridge when connect switch port to TFTP server	Fixed
QUBOOT-4990	LS1028ARDB: SATA does not work	Fixed

Table 4. Open/Known issues

ID	Description
QLINUX-11337	LS1028A: Time gate accuracy does not reach PRL
QLINUX-11394	LS1028ARDB: video playback using GStreamer hang quickly on weston
QLINUX-11423	LS1028AQDS: Kernel booting failed using LPUART U-Boot

NOTE

PPA is replaced by ‘TF-A’, that is, Trusted Firmware for ARM in this release.

Chapter 3
LS1028A BSP user guide

This section provides LS1028ARDB-specific information on switch setting configurations, U-Boot environment variable settings as well as supported binaries. It also provides a description of the virtual banks and flash and memory map layouts.

For more information on the LS1028ARDB refer to the QorIQ LS1028A Reference Design Board Reference Manual and the QorIQ LS1028A Reference Design Board Getting Started Guide.

3.1 LS1028A BSP Quick Start

3.1.1 Introduction

The following sections describe the procedure to download and assemble BSP images and then to deploy these images on to LS1028ARDB. For more information on the different components of the board and on how to configure and boot the board, see QorIQ LS1028A Reference Design Board Getting Started Guide.

3.1.2 Host system requirements

- Ubuntu 18.04 should be installed on the host machine.
- If this requirement is not fulfilled, see “Emulate Ubuntu 18.04 environment using Docker container” topic below.
- For root users, there is no limitation for the build. For non-root users, obtain sudo permission by running the command `sudoedit /etc/sudoers` and adding a line `<user-account-name> ALL=(ALL:ALL) NOPASSWD: ALL in /etc/sudoers`.
- To build the target Ubuntu userland for arm64/armhf arch, the user's network environment must have access to the remote Ubuntu official server.

Emulate Ubuntu 18.04 environment using Docker container (optional)

If a Linux distribution other than Ubuntu 18.04 is installed on the host machine, perform the following steps to create an Ubuntu 18.04 Docker container to emulate the environment.

- Install Docker on the host machine. See https://docs.docker.com/engine/installation/ for information on how to install Docker on the host machine.
- To build the Ubuntu userland, the user must have *sudo* permission for Docker commands or the user must be added to a group called “ docker” as specified below.

  Change current group to “ docker”

  ```
  $ sudo newgrp - docker
  ```

  NOTE

  User can run the command `cut -d: -f1 /etc/group | sort` to check if *docker* group is included in the list of all the available groups.

  Add your account to “ docker” group

  ```
  $ sudo usermod -aG docker <accountname>
  $ sudo gpasswd -a <accountname> docker
  ```

Restart service

```
$ sudo service docker restart
```

- Logout from current terminal session, then login again to ensure user can run `docker ps -a`.
- The user's network environment must have access to the remote Ubuntu official server.
- Run flex-builder command for docker as given in the next section.

3.1.3 Download and assemble LS1028A BSP images

In this BSP Quick Start Guide, the build framework Flexbuild is used. Flexbuild contains three scripts: flex-builder, flex-installer, and flex-mkdistrorfs. In this section, “Download and assemble LS1028A BSP images” flex-builder is used to generate an Ubuntu userland.

In the next section, “Deploy LS1028A BSP images on board,” the script flex-installer is used to deploy Ubuntu userland to a storage device (SD, USB, or SATA). In addition, flex-installer is used to deploy a boot partition and firmware images to the same storage device.

The boot partition contains a Linux kernel image and device tree blob. Firmware images include RCW (Reset Configuration Word), U-Boot and a FIT image. For a complete list of the boot partition components and the firmware components see LS1028A BSP Memory Layout on page 34.

See Build tools on page 35 for more information on flexbuild.

To download and assemble LS1028A BSP images, perform the steps below:

1. Download the flexbuild source tarball and set up flexbuild environment.
    - Go to www.nxp.com and click the “Download” button to download flexbuild source tarball named “flexbuild_<version>.tgz”. It is required to login and sign a license agreement before downloading the tarball.
    - Set up flexbuild environment.

    ```
    $ tar xvzf flexbuild_<version>.tgz
    $ cd flexbuild
    $ source setup.env
    ```

    Run the following two commands in addition to the above three commands only when building BSP in Docker container.

    ```
    $ flex-builder docker (This command emulates Ubuntu 18.04 environment on the host machine.)
    $ source setup.env (Again, setup the flexbuild environment after entering Docker container.)
    ```

2. Download prebuilt images for boot partition and NXP-specific components tarball.
    - Download prebuilt app components (e.g., tsntool, weston, openssl).

    ```
    wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/app_components_LS_arm64.tgz
    ```

    - Download prebuilt images for boot partition and arm modules.

    ```
    wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/bootpartition_LS_arm64_lts_4.14.tgz
    wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/lib_modules_LS_arm64_4.14.47.tgz
    ```

3. Generate LS1028A BSP Ubuntu userland, untar the prebuilt components tarball, and merge them into target userland.
    - Generate Ubuntu arm64 userland.

    ```
    $ flex-builder -i mkrfs -a arm64
    ```

- Extract app components.

```
$ tar xvzf app_components_LS_arm64.tgz -C build/apps
```

- Extract kernel modules (e.g., cryptodev).

```
$ sudo tar xvzf  lib_modules_LS_arm64_<kernel_version>.tgz -C build/rfs/
rootfs_ubuntu_bionic_LS_arm64/lib/modules
```

- Merge all components packages and kernel modules into target userland and compress ubuntu arm64 rootfs as .tgz tarball.

```

$ flex-builder -i merge-component -a arm64
$ flex-builder -i compressrfs -a arm64
```

- Exit, if using Docker.

```
$ exit (optional, this command exits from docker when building in docker)
```

NOTE

- If the Linux host machine is in a subnet that needs HTTP proxy to access external Internet, set environment variable http_proxy and https_proxy as follows.

```
Add the following proxy settings in ~/.bashrc
# No authentication:
export http_proxy=http://<domain>:<port>
export https_proxy=http://<domain>:<port>

# With authentication:
export http_proxy=http://<account>:<password>@<domain>:<port>
export https_proxy=http://<account>:<password>@<domain>:<port>

# Set no_proxy variable to bypass proxy for some local servers:
export no_proxy="localhost,<local-server1>,<local-server2>"
```

NOTE

Only Ubuntu userland is the default file system with full system test (various firmwares and images are verified based on Ubuntu userland) in formal LS1028A BSP release. Although other non-official userland (Debian, CentOS, buildroot-based tiny distro, etc) can be generated and composed with the common LS1028A BSP boot partition (containing Linux kernel, DTBs, distro boot scripts, etc) by flexbuild, that is optional part besides the formal Ubuntu userland and there is no guarantee for other userlands as of now.

3.1.4 Deploy LS1028A BSP images on board

This section assumes that the LS1028A BSP images have been assembled as given in “Download and assemble LS1028A BSP images”. There are two options for deploying LS1028A BSP images on the reference board:

1. Use a removable storage device such as an SD card, connect it to a local Linux host machine and deploy LS1028A BSP images on to the removable storage device. Then, connect this removable storage device to a reference board. This option is typically useful when the user has local access to a reference board and has a local Linux host machine.
2. If the user does not have access to a local host machine and/or a local reference board, LS1028A BSP images can be directly deployed to the storage device that is plugged into the board. This option is typically useful when a remote Linux host server is used, and/or the user is accessing the reference board remotely. For this option, a network connection to the reference board is required.

The deployment covers how to program LS1028A BSP composite firmware for "FlexSPI NOR flash boot", "SD boot", and "eMMC boot". It also covers how to deploy boot partition images and Ubuntu userland on different storage media (SD/USB/SATA).

NOTE

SD/USB/SATA capacity must be at least 8 GB.

Table 5. LS1028A BSP storage location

Boot source	Composite firmware	Kernel image and DTB	Ubuntu rootfs
FlexSPI NOR flash	FlexSPI NOR flash	SD/USB/SATA Partition 2 "Boot Partition"	SD/USB/SATA Partition 3 "Rootfs Partition"
SD	SD "raw partition"		
eMMC	eMMC		

3.1.4.1 LS1028ARDB reference information

This section provides general information about LS1028ARDB which may come in handy as a reference while completing steps for deploying BSP that follow.

System memory map

Table 6. System memory map

Start address	End address	Size	Allocation	Comment
0x0000_0000_0000	0x0000_000F_FFFF	1 MB	CCSR - Boot ROM	64 KB
0x0000_0010_0000	0x0000_00FF_FFFF	15 MB	Reserved	
0x0000_0100_0000	0x0000_0FFF_FFFF	240 MB	CCSR	
0x0000_1000_0000	0x0000_10FF_FFFF	16 MB	Reserved	
0x0000_1100_0000	0x0000_11FF_FFFF	16 MB	Reserved	
0x0000_1200_0000	0x0000_13FF_FFFF	32 MB	Reserved	
0x0000_1400_0000	0x0000_17FF_FFFF	64 MB	Reserved	
0x0000_1800_0000	0x0000_181F_FFFF	2 MB	OCRAM	128 KB
0x0000_1820_0000	0x0000_182F_FFFF	1 MB	Reserved	
0x0000_1830_0000	0x0000_18FF_FFFF	13 MB	Reserved	
0x0000_1900_0000	0x0000_19FF_FFFF	16 MB	CoreSight STM	16 MB
0x0000_1A00_0000	0x0000_1BFF_FFFF	32 MB	Reserved	
0x0000_1C00_0000	0x0000_1CFF_FFFF	16 MB	Reserved	
0x0000_1D00_0000	0x0000_1FFF_FFFF	48 MB	Reserved	
0x0000_2000_0000	0x0000_2FFF_FFFF	256 MB	FlexSPI Region #1	More FlexSPI space below
0x0000_3000_0000	0x0000_3FFF_FFFF	256 MB	Reserved	
0x0000_4000_0000	0x0000_5FFF_FFFF	512 MB	Reserved	
0x0000_6000_0000	0x0000_7FFF_FFFF	512 MB	Reserved	

Table continues on the next page...

Table 6. System memory map (continued)

Start address	End address	Size	Allocation	Comment
0x0000_8000_0000	0x0000_9FFF_FFFF	512 MB	GPP DRAM Region #1(0-2 GB)	
0x0000_A000_0000	0x0000_BFFF_FFFF	512 MB		
0x0000_C000_0000	0x0000_DFFF_FFFF	512 MB		
0x0000_E000_0000	0x0000_FFFF_FFFF	512 MB		
0x0001_0000_0000	0x0001_EFFF_FFFF	3.75 GB	Reserved	
0x0001_F000_0000	0x0001_F07F_FFFF	8 MB	ECAM config space	Embedded RC +EPECAM (256 MB)
0x0001_F080_0000	0x0001_F09F_FFFF	2 MB	Register block space	
0x0001_F0A0_0000	0x0001_F7FF_FFFF	118 MB	Reserved	
0x0001_F800_0000	0x0001_F83F_FFFF	4 MB	Reserved	
0x0001_F840_0000	0x0001_FBFF_FFFF	60 MB	Reserved	
0x0001_FC00_0000	0x0001_FC3F_FFFF	4 MB	Reserved	
0x0001_FC40_0000	0x0001_FFFF_FFFF	60 MB	Reserved	
0x0002_0000_0000	0x0003_FFFF_FFFF	8 GB	Reserved	
0x0004_0000_0000	0x0004_0FFF_FFFF	256 MB	SPI Hole	
0x0004_1000_0000	0x0004_FFFF_FFFF	3.75 GB	FlexSPI Region #2 (256 MB - 4 GB)	3.75 GB
0x0005_0000_0000	0x0005_FFFF_FFFF	4 GB	Reserved	
0x0006_0000_0000	0x0006_FFFF_FFFF	4 GB	Reserved	
0x0007_0000_0000	0x0007_3FFF_FFFF	1 GB	DCSR	
0x0007_4000_0000	0x0007_FFFF_FFFF	3 GB	Reserved	
0x0008_0000_0000	0x0008_1FFF_FFFF	512 MB	Reserved	
0x0008_2000_0000	0x000B_FFFF_FFFF	15.5 GB	Reserved	
0x000C_0000_0000	0x000F_FFFF_FFFF	16 GB	Reserved	
0x0010_0000_0000	0x001F_FFFF_FFFF	64 GB	Reserved	
0x0020_0000_0000	0x0020_7FFF_FFFF	2 GB	Reserved	
0x0020_8000_0000	0x003F_FFFF_FFFF	126 GB	GPP DRAM Region #2	
0x0040_0000_0000	0x005F_FFFF_FFFF	128 GB	Reserved	
0x0060_0000_0000	0x007F_FFFF_FFFF	128 GB	GPP DRAM Region #3	
0x0080_0000_0000	0x0087_FFFF_FFFF	32 GB	PCI Express 1	High-speed I/O (0x0080_0000_0000 -0x00FF_FFFF_FFFF)
0x0088_0000_0000	0x008F_FFFF_FFFF	32 GB	PCI Express 2	

Supported boot options

LS1028ARDB supports the following boot options:

- FlexSPI NOR flash (referred to as "FSPI" or "FSPI flash" in the following sections). CS refers to Chip Select.
- eMMC
- SD card (SDHC1)

On-board switch options

The RDBs have user selectable switches for evaluating different boot options for the LS1028A device as given in the table below ('0' is OFF, '1' is ON).

Boot source	SW2[1:8]	SW3[1:8]	SW5[1:8]
FSPI NOR (default)	1111_1000	1111_0000	0011_1001
SD Card (SDHC1)	1000_1000	1111_0000	0011_1001
eMMC	1001_1000	1111_0000	0011_1001

In addition to the above switch settings, make sure the following jumper settings are correct.

Table 7. LS1028ARDB jumper settings

Jumper	Type	Name/function	Description
J6	1x2-pin connector	TA_BB_EN enable	Open: TA_BB_TMP_DETECT_B pin is High (default value) Shorted: TA_BB_TMP_DETECT_B pin is Low
J7	1x2-pin connector	VBAT_EN	Open: Disable battery backup for TA_BB_VDD (default value) Shorted: Enable battery backup for TA_BB_VDD
J27	1x2-pin connector	PROG_MTR voltage control (for NXP use only)	Open: PROG_MTR pin is powered off (default value) Shorted: PROG_MTR pin is powered by OVDD (1.8 V)
J28	1x2-pin connector	TA_PROG_SFP voltage control (for NXP use only)	Open: TA_PROG_SFP pin is powered off (default value) Shorted: TA_PROG_SFP pin is powered by OVDD (1.8 V)

FlexSPI NOR Flash Chip-select

FlexSPI NOR flash is a simple and convenient destination for deploying images so it is frequently used.

The benefit of this feature is that it allows more than one set of images to be independently deployed to the one NOR flash. This is very helpful during development because you can use the U-Boot image in one chip-select to program an image set into a different chip-select. If the new images are flawed, the old images are still functional to let you deploy corrected images.

The logic on the board usually allows the NOR flash to be accessed from different CS (chip select) option. Each CS is connected to dedicated flash devices. U-Boot prints which CS is loaded from. The output looks like following.

```
NOTICE:  Fixed DDR on board

NOTICE:  4 GB DDR4, 32-bit, CL=11, ECC on
NOTICE:  BL2: v1.5(release):LSDK-18.12-8-gaf6a777e
NOTICE:  BL2: Built : 03:34:46, Feb 18 2019
NOTICE:  BL31: v1.5(release):LSDK-18.12-8-gaf6a777e
NOTICE:  BL31: Built : 03:35:28, Feb 18 2019
NOTICE:  Welcome to LS1028 BL31 Phase


U-Boot 2018.09-g30fb661336 (Feb 18 2019 - 03:33:37 +0800)

SoC:  LS1028AE Rev1.0 (0x870b0010)
Clock Configuration:
       CPU0(A72):1300 MHz  CPU1(A72):1300 MHz
       Bus:      400  MHz  DDR:      1600 MT/s
Reset Configuration Word (RCW):
       00000000: 34004010 00000030 00000000 00000000
       00000010: 00000000 008f0000 0030c000 00000000
       00000020: 020031a0 00002580 00000000 00003296
       00000030: 00000000 00000010 00000000 00000000
       00000040: 00000000 00000000 00000000 00000000
       00000050: 00000000 00000000 00000000 00000000
       00000060: 00000000 00000000 200e705a 00000000
       00000070: bb580000 00000000
Model: NXP Layerscape 1028a RDB Board
Board: LS1028AE Rev1.0-RDB, Version: C, boot from NOR
FPGA: v5 (RELEASED: RDB_2018_1204_1540)
SERDES1 Reference : Clock1 = 100.00MHz Clock2 = 100.00MHz
I2C:   ready
DRAM:  3.9 GiB
DDR    3.9 GiB (DDR4, 32-bit, CL=11, ECC on)
Using SERDES1 Protocol: 47960 (0xbb58)
PCIe0: pcie@3400000 Root Complex: no link
PCIe1: pcie@3500000 Root Complex: no link
Now running in RAM - U-Boot at: fbd3f000
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from SPI Flash... SF: Detected mt35xu02g with page size 256 Bytes, erase size 128
KiB, total 256 MiB
OK
EEPROM: NXID v1
In:    serial
Out:   serial
Err:   serial
Net:   eth0: enetc#0 [PRIME], eth1: enetc#1, eth2: enetc#2, eth3: netc_mdio, eth4: enetc#3
MDIO MUX: no MDIO MUX node
Hit any key to stop autoboot:  0
```

Boot option switching can be performed in U-Boot using the following statements.

- Switch to FlexSPI NOR flash (default):

  ```
  =>qixis_reset
  ```

- Switch to SD:

```
=>qixis_reset sd
```

- Switch to eMMC:

```
=>qixis_reset emmc
```

3.1.4.2 Option 1 - Deploy LS1028A BSP images using removable storage device

Given below are the steps to deploy LS1028A BSP images on to the reference board using a removable storage device (SD/USB/SATA). Option 1 can be used when the user has access to a local Linux host machine and to a local reference board. For this option, the storage device is plugged into the host machine and programmed. Once the programming is complete, the storage device is removed from the host machine and then inserted into the reference board.

1. Connect the removable storage device to the local Linux host machine.
2. The boot partition and rootfs images that were assembled during “Download and assemble LS1028A BSP images” will be available under “flexbuild_<version>” directory on the host machine. The images have following naming format:
   - Boot partition:
     —
     ```
     bootpartition_LS_arm64_<version> or bootpartition_LS_arm64_<version>.tgz (64-bit)
     ```
   - rootfs:
     —
     ```
     rootfs_ubuntu_bionic_LS_arm64 or rootfs_ubuntu_bionic_LS_arm64.tgz (64-bit)
     ```
3. Setup the environment for flex-installer to run:
   ```
   $ cd flexbuild
   $ source setup.env
   ```
4. Execute flex-installer with appropriate arguments to deploy LS1028A BSP images to the storage device.

   For example:
   ```
   $ flex-installer -b bootpartition_arm64_lts_<version>.tgz -r build/images/
   rootfs_ubuntu_bionic_arm64_<timestamp>.tgz -d /dev/sdX
   ```

   WARNING

   - The SD/USB/SATA storage drive in the Linux PC is detected as /dev/sdX, where X is a letter such as a, b, c. Make sure to choose the correct device name, because data on this device will be replaced.
   - Use the command cat /proc/partitions to see a list of devices and their sizes to make sure that the correct device names have been chosen.
   - If the Linux host machine supports read/write SD card directly without an extra SD card reader device, the device name of SD card is typically mmcblk0.
5. After a successful installation, the message “installation finished successfully” appears. Execute the following command to unmount all mounted partitions of the target device.

   Example:
   ```
   $ sudo umount /run/media/sdX
   ```
6. Unplug removable storage device from the Linux host and plugin into the reference board.

7. Make sure the DIP switch settings on the board enable booting from FlexSPI NOR (or eMMC / SD card). (Refer to "On-board switch options" in the preceding section for switch settings.)
8. Power-on the board and stop autoboot to enter the U-Boot prompt.
9. Program/update the composite firmware to FlexSPI NOR flash (or eMMC / SD card) as described below.

   Note: <filename> in below steps refers to `firmware_ls1028ardb_uboot_xspiboot.img` for FlexSPI NOR flash boot source, `firmware_ls1028ardb_uboot_sdboot.img` for SD card, and `firmware_ls1028ardb_uboot_emmcboot.img` for eMMC as boot source.

   - Load firmware image from storage media.

Storage media	Command in U-Boot

USB	`=> usb start`
	`=> load usb 0:2 a0000000 <filename>`

SATA	`=> load scsi 0:2 a0000000 <filename>`
SD	`=> load mmc 0:2 a0000000 <filename>`
eMMC	`=> load mmc 1:2 a0000000 <filename>`

   - If FlexSPI NOR flash is to be used as boot source execute the commands below to program the FlexSPI NOR flash.

```
=> sf probe 0:0; sf erase 0 +$filesize; sf write a0000000 0 $filesize
=> qixis_reset
```

   - If SD card is to be used as boot source execute the commands below to program the SD card.

```
=> mmc rescan; mmc dev 0; mmc write a0000000 8 0x25000
=> qixis_reset sd
```

   - If eMMC is to be used as boot source execute the commands below to program the onboard eMMC.

```
=> mmc dev 1; mmc write a0000000 8 0x25000
```

     where, 0x25000 is number of blocks (max) on eMMC that are required to write the eMMC composite firmware. Each block is 512 bytes size.

     Reboot with eMMC as boot source.

```
=> qixis_reset emmc
```

10. The system will automatically boot up BSP Ubuntu distro available from the removable storage device.

3.1.4.3 Option 2 - Deploy BSP images directly to the storage device on a board

If the user does not have a local Linux host machine that can connect to a removable media, and/or the user does not have local access to a reference board, the user can directly deploy BSP images to the storage device on a reference board. The reference board may be accessed by the user remotely. For this option, a network connection to the reference board is required.

1. Reboot the reference board from FlexSPI NOR CS0 (or SD card / eMMC) and let it boot automatically.
   - If the DIP switch settings are configured to be default, the board will boot from FlexSPI NOR flash CS0 (or SD card / eMMC) using the composite firmware which is present on the reference board by default.
2. After the reference board boots automatically, check whether the reference board boots TinyDistro or whether it boots Ubuntu distribution. TinyDistro is a non-customizable prebuilt ramdisk rootfs deployed in flash media on the reference board. This rootfs fits into the firmware image on flash and is therefore called tiny.
   - If the reference board boots TinyDistro, proceed to step #4.
   - If the reference board boots Ubuntu distribution, it means that an older Ubuntu distribution may already be present on the storage device that is plugged into the reference board. In this case, go to step #3 first, to force the board to boot TinyDistro.
3. Force the reference board to boot TinyDistro.
   - Reboot the board and stop autoboot to enter U-Boot prompt in FlexSPI NOR flash (or SD card / eMMC).
   - If FlexSPI NOR is the boot source, run the following command to force the reference board to boot TinyDistro present on FlexSPI NOR flash.

     ```
     $ run qspi_bootcmd
     ```

   - If SD card is the boot source, run the following commands to force the reference board to boot TinyDistro present on SD card.

     ```
     $ run sd_bootcmd
     ```

   - If eMMC is the boot source, run the following commands to force the reference board to boot TinyDistro present on eMMC.

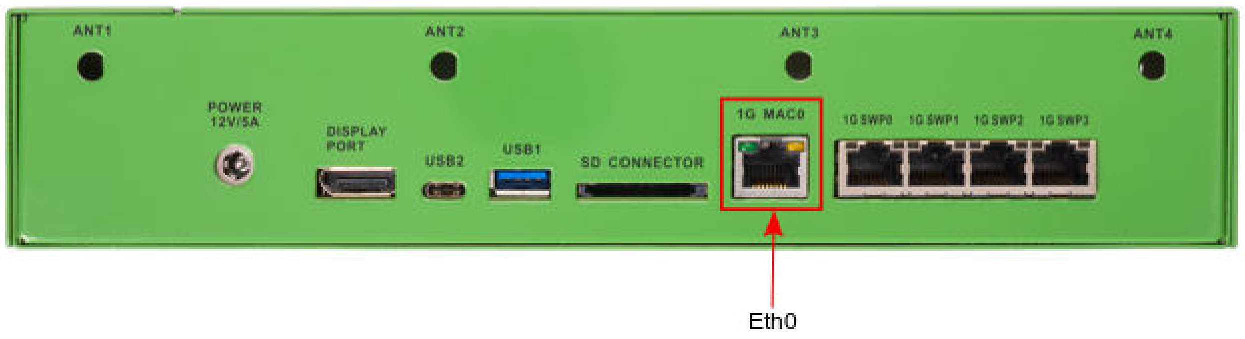
     ```
     $ run emmc_bootcmd
     ```

4. Login to TinyDistro as “root” and bring up a network interface.

   ```
   $ udhcpc -i <port name in TinyDistro>
   ```

   For LS1028ARDB, in TinyDistro as well as in Ubuntu distribution, by default, only one Ethernet port is enabled as a standard Kernel Ethernet Interface.

   For LS1028ARDB, this interface is named as Eth0 in TinyDistro and is labeled as 1G MAC0 on chassis front panel as shown in diagram below.



5. Use flex-installer to create and format the partitions for storage device (USB/SATA/SD).

Storage Device	Commands in Linux
USB	`$ flex-installer -i pf -d /dev/sdX`
SATA	`$ flex-installer -i pf -d /dev/sdX`
SD	`$ flex-installer -i pf -d /dev/mmcblk0`
eMMC	`$ flex-installer -i pf -d /dev/mmcblk1`

WARNING

- The USB/SATA storage drive in the Linux PC is detected as /dev/sdX, where X is a letter such as a, b, c. Make sure to choose the correct device name, because data on this device will be replaced.
- Use the command cat /proc/partitions to see a list of devices and their sizes to make sure that the correct device names have been chosen.

6. Download and deploy two tarballs (boot partition and Ubuntu userland) to USB/SATA/SD storage device. These tarballs were assembled during the “Download and Assemble BSP Images” step.

Storage Device	Commands in Linux
USB	a. `$ cd /run/media/sdX3` b. Download `bootpartition_<arch>_<version>.tgz and rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz` using the `wget` or `scp` command. c. `$ flex-installer -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/sdX`
SATA	a. `$ cd /run/media/sdX3` b. Download `bootpartition_<arch>_<version>.tgz and ubuntu_<codename>_<arch>_rootfs_<timestamp>.tgz` using the `wget` or `scp` command. c. `$ flex-installer -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/sdX`
SD	a. `$ cd /run/media/mmcblk0p3` b. Download `bootpartition_<arch>_<version>.tgz and ubuntu_<codename>_<arch>_rootfs_<timestamp>.tgz` using the `wget` or `scp` command. c. `$ flex-installer -i install -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/mmcblk0`
eMMC	a. `$ cd /run/media/mmcblk1p3` b. Download `bootpartition_<arch>_<version>.tgz and ubuntu_<codename>_<arch>_rootfs_<timestamp>.tgz` using the `wget` or `scp` command. c. `$ flex-installer -i install -b bootpartition_<arch>_lts_<version>.tgz -r rootfs_ubuntu_<codename>_<arch>_<timestamp>.tgz -d /dev/mmcblk1`

7. Reboot the board with BSP images. The system will automatically boot Ubuntu userland. Login using root/root or user/user.
8. After the above steps are completed, the reference board is ready to boot the latest Ubuntu userland. Optionally, to make sure that the reference board boots using the latest firmware, user may choose to update firmware using steps below:
   - FlexSPI NOR firmware
     a. Reboot the board from FlexSPI NOR flash CS0 and stop autoboot to enter U-Boot prompt.
     b. Download the BSP NOR composite firmware for LS1028ARDB using the command

     ```
     $ wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/
     firmware_ls1028ardb_uboot_xspiboot.img
     ```

     c. Under U-Boot, download the firmware via TFTP to the reference board.

     ```

     => tftp a0000000 firmware_ls1028ardb_uboot_xspiboot.img
     => sf probe 0:0; sf erase 0 +$filesize; sf write a0000000 0 $filesize
     => qixis_reset
     ```

     In the steps above, the contents of the FlexSPI NOR flash are overwritten with the new FlexSPI NOR firmware. The board is rebooted using the FlexSPI NOR flash.
   - SD firmware
     a. Reboot the board from FlexSPI NOR flash CS0 (or SD card / eMMC) and stop autoboot to enter U-Boot prompt.
     b. Download the BSP SD composite firmware for LS1028ARDB using the command

     ```
     $ wget http://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/
     firmware_ls1028ardb_uboot_sdboot.img
     ```

     c. Under U-Boot, download the firmware via TFTP to the reference board.

     ```
     => tftp a0000000 firmware_ls1028ardb_uboot_sdboot.img
     => mmc rescan; mmc dev 0; mmc write a0000000 8 0x25000
     => qixis_reset sd
     ```

     In the steps above, the contents of the SD card are overwritten with the new SD firmware. The board is then rebooted using the SD card.
   - eMMC firmware
     a. Reboot the board from FlexSPI NOR flash CS0 (or SD card / eMMC) and stop autoboot to enter U-Boot prompt.
     b. Download the BSP eMMC composite firmware for LS1028ARDB using the following command.

     ```
     $ wget http://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/
     firmware_ls1028ardb_uboot_emmcboot.img
     ```

     c. Under U-Boot, download the firmware via TFTP to the reference board.

     ```
     => tftp a0000000 firmware_ls1028ardb_uboot_emmcboot.img
     => mmc dev 1
     => mmc write a0000000 8 0x25000
     ```

     where, 0x25000 is number of blocks (max) on eMMC that are required to write the eMMC composite firmware. Each block is 512 bytes size.

d. Reboot with eMMC as boot source.

```
=> qixis_reset emmc
```

In the steps above, the contents of the eMMC are overwritten with the new eMMC firmware. The board is then rebooted using the onboard eMMC.

3.2 How to build LS1028A BSP with Flexbuild

Flexbuild provides cmdline for various build scenarios. The LS1028A BSP Quick Start on page 12 section introduces how to build the LS1028A BSP distro userland with prebuilt boot partition and component tarballs for quick deployment on the target board. This section introduces detailed steps to build LS1028A BSP with Flexbuild.

Click here to download flexbuild source tarball in the name format flexbuild_<version>.tgz

```
$  tar xvzf flexbuild_<version>.tgz
$  cd flexbuild
$  source setup.env
$  flex-builder -h
```

Build TF-A with RCW and U-Boot in Flexbuild

Layerscape platforms support TF-A (Trusted Firmware-A) which provides a reference implementation of secure world software for Armv7-A and Armv8-A.

flex-builder can automatically build the dependent RCW, U-Boot, OPTEE and CST when building ATF to generate bl2 and fip images for Layerscape platforms.

Use the commands below to build uboot-based ATF image in Flexbuild.

```
Usage:
$ flex-builder -c atf  -m <machine>  -b <boottype> [-s]
Example:
$ flex-builder -c atf -m ls1028ardb -b xspi            # build uboot-based ATF image for Flexspi-NOR
boot on ls1028ardb
$ flex-builder -c atf -m ls1028ardb -b sd              # build uboot-based ATF image for SD boot on
ls1028ardb
$ flex-builder -c atf -m ls1028ardb -b xspi -s         # build uboot-based ATF image for Flexspi-NOR
secure boot on ls1028ardb
```

NOTE

In case users modify RCW source code (in packages/firmware/rcw) or specify different RCW binary rather than the default one, run 'flex-builder -c rcw' to regenerate RCW binary, then reconfigure rcw_<boottype> variable in configs/board/<machine>/manifest.

NOTE

If RCW or U-Boot source code is updated since last build, make sure clean the obsolete image by commands 'rm -rf build/firmware/u-boot/<machine>' and/or 'rm -rf build/firmware/rcw/<machine>', then re-build ATF by command 'flex-builder -c atf -m <machine> -b <boottype>'

NOTE

The '-s' option is used for secure boot, OPTEE and FUSE_PROVISIONING are not enabled by default, change CONFIG_BUILD_OPTEE=n to y and/or change CONFIG_FUSE_PROVISIONING=n to y in configs/build_lsdk.cfg to enable it if necessary.

Build Linux kernel with Flexbuild

Besides building LS1028A BSP kernel in stand-alone way (see Configuring and building on page 38), it is easy to automatically build LS1028A BSP kernel with flex-builder.

To build kernel using the default tree/branch/tag configurations specified in configs/build_lsdk.cfg:

```
$ flex-builder -c linux                        # for 64-bit mode of all armv8 Layerscape platforms by
default
$ flex-builder -c linux -a arm64               # for 64-bit mode of all armv8 Layerscape platforms by
default
```

To build kernel with specified tree/branch/tag and additional fragment config:

```
Usage:
$ flex-builder -c linux:<kernel-repo>:<branch|tag> -a arm64 -B fragment:<custom>.config
Example:
$ flex-builder -c linux:linux:linux-4.14-ls1028a-early-access -a arm64 -B fragment:lttng.config
$ flex-builder -c linux:linux:ls1028a-early-access-bsp0.2 -a arm64
```

User can put a custom kernel fragment config file (given named test.config) in flexbuild/packages/linux/<kernel-repo>/arch/arm64/configs directory, then run the command below to compile kernel as per the default defconfig, lsdk.config and the additional test.config.

```
$ flex-builder -c linux -a arm64 -B fragment:test.config
```

Platform	Command for building Linux

LS1028AQDS 64bit	$ flex-builder -c linux:custom (optional, customize kernel config in interactive menu)
	or
	$ flex-builder -c linux:<kernel-repo>:<branch|tag> -B fragment:<custom>.config (optional, add additional fragment config)
	$ flex-builder -c linux
	(if -a <arch> is not specified, arm64 arch is used by default)
	$ flex-builder -c linux:custom -a arm64 (optional, customize kernel config in interactive menu)
	$ flex-builder -c linux -a arm64

LS1028ARDB 64bit	$ flex-builder -c linux:custom (optional, customize kernel config in interactive menu)
	or
	$ flex-builder -c linux:<kernel-repo>:<branch|tag> -B fragment:<custom>.config (optional, add additional fragment config)
	$ flex-builder -c linux
	(if -a <arch> is not specified, arm64 arch is used by default)
	$ flex-builder -c linux:custom -a arm64 (optional, customize kernel config in interactive menu)
	$ flex-builder -c linux -a arm64

Build LS1028A BSP composite firmware and boot partition

LS1028A BSP composite firmware consists of atf bl2, atf bl3 fip frimware, bootloader environment, secure headers, Ethernet MAC/PHY firmware, dtb, kernel and tiny ramdisk rfs, etc. This composite firmware can be programmed at offset 0x0 in flash device or at offset block# 8 in SD/eMMC card.

To generate LS1028A BSP composite firmware for Layerscape platform, directly run the following command:

```
Usage:
$ flex-builder -i mkfw -m <machine> -b <boottype> [-B <bootloader>] [-s]

Examples:
$ flex-builder -i mkfw -m ls1028ardb -b xspi
  firmware_ls1028ardb_uboot_xspiboot.img will be generated.

$ flex-builder -i mkfw -m ls1028ardb -b xspi -s
  firmware_ls1028ardb_uboot_xspiboot_secure.img will be generated.

$ flex-builder -i mkfw -m ls1028ardb -b sd
  firmware_ls1028ardb_uboot_sdboot.img will be generated.

$ flex-builder -i mkfw -m ls1028ardb -b sd -s
  firmware_ls1028ardb_uboot_sdboot_secure.img will be generated.
```

Similarly, the following composite firmware can be generated with the same usage of flex-builder command:

```
firmware_ls1028aqds_uboot_sdboot.img
firmware_ls1028aqds_uboot_sdboot_secure.img
firmware_ls1028aqds_uboot_xspiboot.img
firmware_ls1028aqds_uboot_xspiboot_secure.img
firmware_ls1028ardb_uboot_emmcboot.img
```

To generate LS1028A BSP boot partition tarball, run the command as below:

```
$ flex-builder -i mkbootpartition -a arm64
or
$ flex-builder -i mkbootpartition -a arm64 -s (for secure boot)
```

The command above will generate all needed images including kernel image, dtb, distro boot script, flex_linux_<arch>.itb, small ramdiskrfs, etc. Flex-builder automatically builds the dependent images if they are not present.

How to build application components in Flexbuild

The following commands are some examples of building application components.

```
Usage:
$ flex-builder -c <component> -a <arch>  # build single application component for specified <arch>
Example:
$ flex-builder -c apps          # build all apps components for arm64 arch
$ flex-builder -c tsntool       # build tsntool component for arm64 arch
$ flex-builder -c wayland       # build wayland component for arm64 arch
$ flex-builder -c weston        # build weston component for arm64 arch
$ flex-builder -c cst           # build cst component, needed for secure boot
(arm64 is the default arch if -a <arch> is not specified)
```

To add new application component in Flexbuild, follow the steps below:

1. Add new <component> to apps_repo_list and set CONFIG_BUILD_<component>=y in configs/build_xx.cfg.

2. Configure url/branch/tag/commit info for new <component_name>in configs/build_xx.cfg, default remote. Component git repository is specified by GIT_REPOSITORY_URL by default if <component>_url is not specified, user also can directly create the new component git repository in packages/apps directory.
3. Add build support of new component in packages/apps/Makefile..
4. Run flex-builder -c <component-name> -a <arch>' to build the new component.
5. Run flex-builder -i merge-component -a <arch> to merge the new component package into target distro userland.

How to update existing Linux kernel with new custom kernel for Ubuntu on target board in case of non secure boot

User can quickly install custom kernel and lib modules after Ubuntu had been deployed in SD card on target board in case of non secure boot, follow the steps below.

```
After modify Linux kernel source code in $FBDIR/packages/linux/<kernel-repo> on demand, rebuild kernel
as below
$ flex-builder -c linux:custom (optional, to customize kernel config in interactive menu)
$ flex-builder -c linux
$ flex-builder -i mkbootpartition -a arm64
The new kernel image tarball $FBDIR/build/images/linux_4.14_LS_arm64_<timestamp>.tgz will be generated.

Then login Ubuntu system on target board, update kernel image (take LS1028ARDB for example) as below:
root@localhost:/# dhclient -i eno0
root@localhost:~# cd /
root@localhost:/# wget <path>/linux_4.14_LS_arm64_<timestamp>.tgz (or by scp command)
root@localhost:/# tar xf linux_4.14_LS_arm64_<timestamp>.tgz
root@localhost:/# reboot
System will reboot and automatically boot to Ubuntu with new custom kernel.
```

Rebuild images after modifying the source code of NXP user space components locally

Flexbuild supports building specific components after the source code is changed. Flexbuild then deploys the changes to the target board.

1. Modify source code of user space components in the directory packages/apps/<apps-component> on demand.
2. Clean old build footprint under user space component.

```
$ make clean -C $FBDIR/packages/apps/<app-component>
$ rm -rf $FBDIR/build/apps/components_LS_arm64
```

3. Build the user space component and generate the compressed component tarball.

```
$ flex-builder -c <apps-component> -a arm64
$ flex-builder -i compressapps -a arm64
```

4. Upload and deploy the newly generated app_components_LS_arm64.tgz to target board on which Ubuntu distro had been installed in SD card.

```
=> run sd_bootcmd  (or run xspi_bootcmd to enter Tiny distro Linux environment)
Download app_components_LS_arm64.tgz via wget or scp command
root@TinyDistro:~# tar xf app_components_LS_arm64.tgz
root@TinyDistro:~# cp -a components_LS_arm64/*  /run/media/mmcblk0p3
root@TinyDistro:~# reboot
```

How to generate a custom Ubuntu root filesystem with custom additional package list on x86 host machine

In Flexbulid, there are two default additional package lists for Ubuntu/Debian: additional_packages_list_moderate, and additional_packages_list_tiny.

```
$ flex-builder -i mkrfs -a arm64   (as per additional_packages_list_moderate with more packages for
Ubuntu rootfs by default)
$ flex-builder -i mkrfs -r ubuntu:tiny -a <arch>   (as per additional_packages_list_tiny with less
packages for Ubuntu rootfs)
$ flex-builder -i mkrfs -r ubuntu -a <arch> -B <custom_packages_list>
```

Optionally, if you do not want to use default Ubuntu userland in certain use cases, you can generate Debian, CentOS or buildroot-based tiny userland by following instruction by Flexbuild, for examples:

```
$ flex-builder -i mkrfs -r debian:tiny:stretch -a arm64 (generate arm64 tiny Debian stretch rootfs as
per configs/ubuntu/additional_packages_list_tiny)
$ flex-builder -i mkrfs -r centos -a arm64             (generate arm64 LE CentOS rootfs as per configs/
centos/distro.cfg)
$ flex-builder -i mkrfs -r buildroot:tiny -a arm64      (generate arm64 LE buildroot userland as per
qoriq_arm64_tiny_defconfig)
$ flex-builder -i mkrfs -r buildroot:moderate -a arm64  (generate arm64 LE buildroot userland as per
qoriq_arm64_moderate_defconfig)
$ flex-builder -i mkrfs -r buildroot:custom -a arm64    (generate arm64 LE buildroot userland as per
custom qoriq_arm64_moderate_defconfig)
$ flex-builder -i mkrfs -r buildroot:custom -a arm64:be (generate arm64 big-endian buildroot userland
with custom qoriq_arm64_moderate_defconfig)
$ flex-builder -i mkrfs -r buildroot:imaevm -a arm64    (generate arm64 LE initramfs userland with
qoriq_arm64_imaevm_defconfig used for secure boot with IMA EVM)
```

To install a new package to build/rfs/rootfs_ubuntu_bionic_LS_arm64 filesystem, run the following commands:

```
$ sudo chroot build/rfs/rootfs_ubunutu_bionic_LS_arm64
$ apt-get install <new_package_name>
# exit
```

How to add a new custom machine based on LS1028A BSP release in flexbuild

Assumption: Adding a new custom machine named LS1043AXX based on LS1043A SoC

1. Run flex-builder -i repo-fetch to fetch all git repositories of LS1028A BSP components for the first time
2. Add new machine support in U-Boot and ATF, example:

   ```
   $ cd packages/firmware/u-boot
   $ git checkout LSDK-18.12 -b LSDK-18.12-LS1043AXX
   ```

   Add necessary U-Boot patches in U-boot tree and commit the patches in this branch for new machine. Add necessary ATF patches in ATF tree and commit the patches in this branch for new machine. Assume that you have added packages/firmware/u-boot/configs/ls1043axx_tfa_defconfig and added CONFIG_MACHINE_LS1043AXX=y in configs/build_lsdk.cfg, then run commands below to build TF-A image based on U-Boot for SD boot on LS1043AXX

   ```
   $ cd packages/firmware/atf
   $ git checkout LSDK-18.12 -b LSDK-18.12-LS1043AXX
   $ flex-builder -c atf -m ls1043axx -b sd
   ```

3. Add new machine support in Linux kernel, take LSDK-18.12 for example:

   ```
   $ cd packages/linux/linux
   $ git checkout LSDK-18.12-V4.14 -b LSDK-18.12-V4.14-LS1043AXX
   ```

   Now, you can add kernel patches and submit the patches in this branch for the new machine, then make a tag as below.

   ```
   $ git tag LSDK-18.12-V4.14-LS1043AXX
   ```

   Assume that you have added a new ls1043axx.dts in packages/linux/linux/arch/arm64/boot/dts directory, run as below to build kernel image for new LS1043AXX

   ```
   $ flex-builder -c linux:linux:LSDK-18.12-V4.14-LS1043AXX
   ```

4. Add configs in flexbuild for new machine.
   a. Add ls1043axx node in configs/linux/linux_arm64.its.
   b. Add CONFIG_MACHINE_LS1043AXX=y in configs/build_lsdk.cfg.
   c. Set linux_repo_tag to LSDK-18.12-V4.14-LS1043AXX and set u_boot_repo_tag to LSDK-18.12-LS1043AXX in configs/build_lsdk.cfg.
   d. Set BUILD_DUAL_KERNEL to n in configs/build_lsdk.cfg if user doesn't need the second version of linux.
   e. Optionally, user can use different memory layout from default settings by modifying them in configs/board/common/memorylayout.cfg.
   f. Add manifest for new machine as below

      ```
      $ mkdir configs/board/ls1043axx
      $ cp configs/board/ls1043ardb/manifest configs/board/ls1043axx.
      ```

   g. Update the settings in configs/board/ls1043axx/manifest on demand.
   h. Generally, user can reuse the settings of rcw/fman/qe/eth-phy used for existing ls1043ardb if those components are same for new ls1043axx, otherwise user needs to add new support in packages/firmware/rcw.
   i. Run flex-builder -i mklinux -a arm64 to generate lsdk_linux_arm64_tiny.itb image.
   j. Run flex-builder -i mkfw -m ls1043ardb -b sd to generate the shared ppa/rcw/fman/qe/eth-phy images for new ls1043axx.
   k. Run flex-builder -i mkfw -m ls1043axx -b sd to generate firmware_ls1043axx_uboot_sdboot.img.
   l. User can boot the new lsdk_linux_arm64_tiny.itb from U-Boot prompt duringdebugging stage on LS1043AXX machine.

      ```
      => tftp a0000000 lsdk_linux_arm64_tiny.itb
      => bootm a0000000#ls1043axx
      ```

5. Build all other images for new custom machine with Ubuntu userland if required as below:

   ```
   $ flex-builder -i mkrfs -a arm64
   $ flex-builder -c apps -a arm64
   $ flex-builder -i mkbootpartition -a arm64
   $ flex-builder -i merge-component -a arm64
   $ flex-builder -i compressrfs -a arm64
   ```

Finally, install bootpartition_arm64_lts_<version>.tgz and rootfs_ubuntu_<codename>_<arch>.tgz to SD/USB/SATA storage drive on LS1043AXX machine by flex-installer as below:

```
$ flex-installer -b bootpartition_LS_arm64_<version>.tgz -r build/rfs/
rootfs_ubuntu_bionic_LS_arm64 -d /dev/sdX
```

How to install distro to SD/USB/SATA storage drive

Use the LS1028A BSP flex-installer to install all the release binaries and distro userland onto a storage media (e.g. SD/eMMC card, USB/SATA disk) on the Linux host machine or on the target board.

Follow the instructions below:

Step 1: Plug SD/USB/SATA storage device to Linux Host machine or target board

Step 2: Install LS1028A BSP distro

- If the prebuilt distro tarball generated by Flexbuild is available on Linux host machine, run the following command:

```
$ flex-installer -b bootpartition_LS_arm64_lts_4.14.tgz -r rootfs_ubuntu_bionic_LS_arm64.tgz -m
ls1028ardb -d /dev/sdx
```

NOTE

`sdx` should be the actual device name on the host machine, for example: `sdb, sdc, mmcblk0`, etc.

- If the user wants to modify source code and build a custom LS1028A BSP distro with flexbuild, use the commands below on the Linux host machine:

```
$ flex-builder -c linux:custom -a <arch>  # customize kernel config in interactive menu
$ flex-builder -c linux -a <arch>
$ flex-builder -i mkfw -m <machine> -b <boottype>
$ flex-builder -i mkrfs -a <arch>
$ flex-builder -c apps -a <arch>
$ flex-builder -i merge-component -a <arch>
$ flex-builder -i mkbootpartition -a <arch>
$ flex-installer -b build/images/bootpartition_LS_arm64_lts_4.14 -r build/rfs/
rootfs_ubuntu_bionic_LS_arm64 -d /dev/sdx
```

- If the user wants to install disrto rootfs directly onto SD/USB/SATA disk on QorIQ board on which Linux is unavailable, download the prebuilt lsdk_linux_<arch>_LS_tiny.itb image using the command:

```
$ wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/lsdk_linux_arm64_LS_tiny.itb
```

  Optionally, locally build it using the command below:

```
$ flex-builder -i mklinux -a <arch> to generate lsdk_linux_arm64_LS_tiny.itb
```

  Put lsdk_linux_arm64_LS_tiny.itb to a TFTP service directory, then download it to the target board under U-Boot prompt as below:

```
=> tftp a0000000 lsdk_linux_arm64_LS_tiny.itb
=> bootm a0000000#<board-name>
```

  The <board-name> is: ls1028aqds or ls1028ardb.

- After booting and logging in to Linux on the target board, download the prebuilt distro tarballs generated by Flexbuild and install using the commands below:

```
$ flex-installer -i pf -d /dev/sdx
$ cd /run/media/{mmcblk0p3 or sdx3}, then download distro images to SD/USB/SATA storage disk via
```

```
wget or scp command
$ flex-installer -b bootpartition_LS_arm64_lts_<version>.tgz -r rootfs_ubuntu_bionic_LS_arm64.tgz -
d /dev/sdX
```

Step 3: Power on or reboot the target board after finishing the distro installation, the system will enter boot loader (U-Boot) and automatically scan boot configuration script from the attached SD/USB/SATA disk and boot the target LS1028A BSP distro if found, otherwise it falls back to boot from Flexspi-NOR flash with tiny ramdisk distro.

How to program firmware to SD/Flexspi-NOR flash media

- **For SD card (on all platforms):**
    1. Download the prebuilt image (take LS1028ARDB for example):
        — **Option 1:** Load the prebuilt image from SD card in U-Boot:

        ```
        => load mmc 0:2 a0000000 firmware_ls1028ardb_uboot_sdboot.img
        ```

        — **Option 2:** Download the prebuilt image using the `wget` command:

        ```
        $ wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/
        firmware_ls1028ardb_uboot_sdboot.img
        ```

        — **Option 3:** To locally generate firmware_ls1028ardb_uboot_sdboot.img, run the command as below:

        ```
        $ flex-builder -i mkfw -m ls1028ardb -b sd
        ```

    2. Program `firmware_<machine>_uboot_sdboot.img` to SD card:
        — **Under U-Boot:**

        ```
        => load mmc 0:2 a0000000 firmware_ls1028ardb_uboot_sdboot.img
        => mmc write a0000000 8 1fff8 (same on all platforms)
        => qixis_reset sd
        ```

        — **Under Linux:**

        ```
        $ flex-installer -f firmware_ls1028ardb_uboot_sdboot.img -d /dev/mmcblk0
        ```

- **For Flexspi-NOR flash:**
    — On LS1028ARDB:
    1. Download the image using the following options:
        ◦ **Option 1:** Load prebuilt image from SD card.

        ```
        => load mmc 0:2 a0000000 firmware_ls1028ardb_uboot_xspiboot.img
        ```

        ◦ **Option 2:** Download the prebuilt image using the `wget` command.

        ```
        $ wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/
        firmware_ls1028ardb_uboot_xspiboot.img
        ```

        ◦ **Option 3:** To locally generate firmware_ls1028ardb_uboot_xspiboot.img, run the command as below:

        ```
        $ flex-builder -i mkfw -m ls1028ardb -b xspi
        ```

2. Program `firmware_ls1028ardb_uboot_xspiboot.img` to Flexspi-NOR flash:

   ```
   => sf probe 0:0
   => sf erase 0 +$filesize && sf write 0xa0000000 0 $filesize
   ```

— On LS1028AQDS:

1. Download the image using the following options:
   - **Option 1**: Load prebuilt image from SD card.

     ```
     => load mmc 0:2 a0000000 firmware_ls1028aqds_uboot_xspiboot.img
     ```

   - **Option 2**: Download the prebuilt image using the wget command.

     ```
     $ wget https://www.nxp.com/lgfiles/sdk/ls1028a_bsp_03/
     firmware_ls1028aqds_uboot_xspiboot.img
     ```

   - **Option 3**: To generate `firmware_ls1028aqds_uboot_xspiboot.img` locally, run the command as below:

     ```
     $ flex-builder -i mkfw -m ls1028aqds -b xspi
     ```

2. Program `firmware_ls1028aqds_uboot_xspiboot.img` to Flexspi-NOR flash:

   ```
   => sf probe 0:0
   => sf erase 0 +$filesize && sf write 0xa0000000 0 $filesize
   ```

3.3 Procedure to run secure boot

The following steps describe how to run secure boot on the LS1028A RDB/QDS board, after building the images.

1. Blow One Time Programmable Master Key (OTPMK).
   a. Check initial SNVS state.

   ```
   ccs::display_mem <sap chain position>  0x1e90014 4 0 4
   88000900
   ```

   The second nibble ‘8’ indicates that OTPMK is not blown.

   b. Boot the board to U-Boot prompt.

   c. Check the OTPMK value that should not to be blown.

   ```
   md 0x1e90014
   ```

   This shows the value as 8**8**000900.

   d. Write OTPMK fuse values on shadow registers.

   ```

   md 1e90014
       80000900
   md 1e80024
       00000000
   ```

   You will see ‘0’ in the second nibble. No parity errors, that is, bits marked in read are all 0’s.

e. Blow the OTPMK to fuses if no parity error is found.

```
mw 1e80020 0x02
```

f. Reset and check that SNVS is in check state.

```

ccs::display_mem <sap chain position>  0x1e90014 4 0 4
    80000900
```

2. Generate secure boot images.
   a. Fetch the CST repository and checkout to ls1028-early-access branch.
   b. Change directory as `cd cst` and run `make` command.
   c. Generate RSA public and private keys with the following command: `./gen_keys <key_size>, key_size = 1024, 2048 or 4096`.
   d. Copy all the binaries to **./cst** directory.
      i. Rcw.bin
      ii. U-boot.bin
      iii. ppa.itb
      iv. kernel.itb
   e. Use below commands to sign images:
      i. `$./uni_pbi input_files/uni_pbi/ls1028/input_pbi_flexspi_nor_secure # pbi header`
      ii. `$./uni_sign input_files/uni_sign/ls1028/nor/input_uboot_secure #uboot header`
      iii. `$./uni_sign input_files/uni_sign/ls1028/nor/input_ppa_secure #ppa header`
      iv. `$./uni_sign input_files/uni_sign/ls1028/input_kernel_secure #kernel header`
3. Flash address for headers and respective images.

   Place all the images on the NOR flash at the specified offsets:
   - rcw_sec.bin >> 0x000000
   - hdr_ppa.out >> 0x600000
   - hdr_uboot.out >> 0x6c0000
   - hdr_kernel.out >> 0x800000
   - u-boot-dtb.bin >> 0x100000
   - ppa.itb >> 0x400000
   - kernel.itb >> 0x1000000
4. Put the board into RSP (reset pause).

```
config_chain {ls1028a dap};
display ccs::get_config_chain;

# To enable RSP:
ccs::config_chain testcore;
jtag::lock;
jtag::state_move test_logic_reset;

jtag::scan_out ir 4 3;
jtag::scan_out dr 6 1;
```

```
jtag::scan_io ir 8 0x93;
jtag::scan_io dr 64 0x0;

jtag::scan_io ir 8 0x92;
jtag::scan_io dr 64 0x0;

jtag::set_pin 0 0;
after 100;
puts [jtag::scan_io ir 8 0x93];
puts [jtag::scan_io dr 64 0x0000010071FF001F];
jtag::set_pin 0 1;
jtag::unlock;

### Wait here for 2 -3 secs to allow the board to reset as done by above steps ####

config_chain {ls1028a dap};
display ccs::get_config_chain;

ccs::write_mem 2 0x7 0x001000D0 4 0 0x00080000;
ccs::stop_core 1;
ccs::write_mem 1 0x1E80254 4 0 <SRKH1>;
ccs::write_mem 1 0x1E80258 4 0 <SRKH2>;
ccs::write_mem 1 0x1E8025c 4 0 <SRKH3>;
ccs::write_mem 1 0x1E80260 4 0 <SRKH4>;
ccs::write_mem 1 0x1E80264 4 0 <SRKH5>;
ccs::write_mem 1 0x1E80268 4 0 <SRKH6>;
ccs::write_mem 1 0x1E8026c 4 0 <SRKH7>;
ccs::write_mem 1 0x1E80270 4 0 <SRKH8>;

ccs::display_mem 1 0x1e80254 4 0 8;
ccs::run_core 1;
ccs::write_mem 2 0x7 0x001000D0 4 0 0x00040000;
```

After implementing all the above steps, the board boots up, and Linux prompt appear after successful validation of all the images.

3.4 LS1028A BSP Memory Layout

Flash layout

The following table shows the memory layout of various firmware stored in NOR/NAND/QSPI flash device or SD card on the LS1028ARDB board.

Table 8. Flash layout

Definition		Max Size	NOR/QSPI/NAND Flash Offset	SD Card Start Block No.
RCW+PBI+ BL2 (bl2.pbl)		1MB	0x00000000	0x00008
ATF FIP Image (fip.bin) BL31 + BL32 + BL33		4MB	0x00100000	0x00800
Boot firmware Environment		1MB	0x00500000	0x02800
DP firmware		256KB	0x00900000	0x04800
Kernel	lsdk_linux_<arch>.itb	16MB	0x01000000	0x08000

Table continues on the next page...

Table 8. Flash layout (continued)

Ramdisk RFS		32MB	0x02000000	0x10000

Storage layout on SD/USB/SATA for LS1028A BSP images deployment

With LS1028A BSP flex-installer, the LS1028A BSP distro can be installed into an SD/USB/SATA storage disk which should have at least 8GB of memory space.

Table 9. Storage Layout on SD/USB/SATA for LS1028A BSP Image Deployment

Region 1 (4KB)	Region 2 (RAW) 64MB Firmware	Region 3 (Partition-1 FAT32) 20MB EFI	Region 4 (Partition-2 EXT4) 1GB Boot partition	Region 5 (Partition-3 EXT4) Remaining space RootFS partition
MBR/GPT	RCW U-Boot or UEFI TF-A firmware Secure boot headers FMan/DP firmware QE/uQE firmware Eth PHY firmware MC firmware DPC firmware DPL firmware DTB lsdk_linux_<arch>.itb	BOOTAA64.EFI grub.cfg	kernel image DTB lsdk_linux_<arch>.itb distro boot scripts secure headers other	Ubuntu or Ubuntu-Core or CentOS or Debian

3.5 Build tools

Flexbuild is a component-oriented build framework and integrated platform with capabilities of flexible, easy-to-use, scalable system build and distro installation. With flex-builder CLI tool, users can build various components (Linux, U-Boot, RCW, TF-A and miscellaneous custom userspace applications) and distro userland to generate composite firmware, hybrid rootfs with customable userland. The following are Flexbuild's main features:

- Automatically build Linux, U-Boot, TF-A, RCW and miscellaneous user space applications.
- Generate machine-specific composite firmware for various boot types: SD/QSPI/NOR/NAND boot, secure boot, U-Boot.
- Support integrated management with repo-fetch, repo-branch, repo-commit, repo-tag, repo-update for git repositories of all components.
- Support cross build on x86 Ubuntu 18.04 host machine for aarch64/armhf arch target.
- Support native build on aarch64/armhf machine for ARM arch target.
- Support creating an Ubuntu docker container and building LSDK inside it when the host machine is using CentOS, RHEL, Fedora, SUSE, Debian, non-18.04 Ubuntu, etc.

- Scalability of integrating various components of both system firmware and user space applications.
- Capability of generating custom aarch64/armhf Ubuntu userland integrated configurable packages and proprietary components.

Flexbuild can separately build each component or automatically build all components, it generates the boot firmware (RCW, U-Boot, PHY firmware, kernel image, and ramdiskrfs), lsdk_linux_<arch>_tiny.itb, and the Ubuntu userland containing the specified packages and application components.

NOTE

Since LSDK-18.06, upgrading of toolchain is required for U-Boot v2018.03 or later, if your host machine is not a Ubutnu 18.04 system, there are two ways to use Ubuntu 18.04 toolchain as below:

- Run sudo do-release-upgrade command to upgrade existing Ubuntu 16.04 to Ubuntu 18.04
- Run flex-builder docker command on the existing non Ubuntu 18.04 host to create a ubuntu 18.04 docker container in which GCC 7.3.0 is available, then build LSDK in docker.

Chapter 4
Linux kernel

Introduction

The Linux kernel is a monolithic Unix-like computer operating system kernel. It is the central part of Linux operating systems that are extensively used on PCs, servers, handheld devices and various embedded devices such as routers, switches, wireless access points, set-top boxes, smart TVs, DVRs, and NAS appliances. It manages tasks/applications running on the system and manages system hardware. A typical Linux system looks like this:

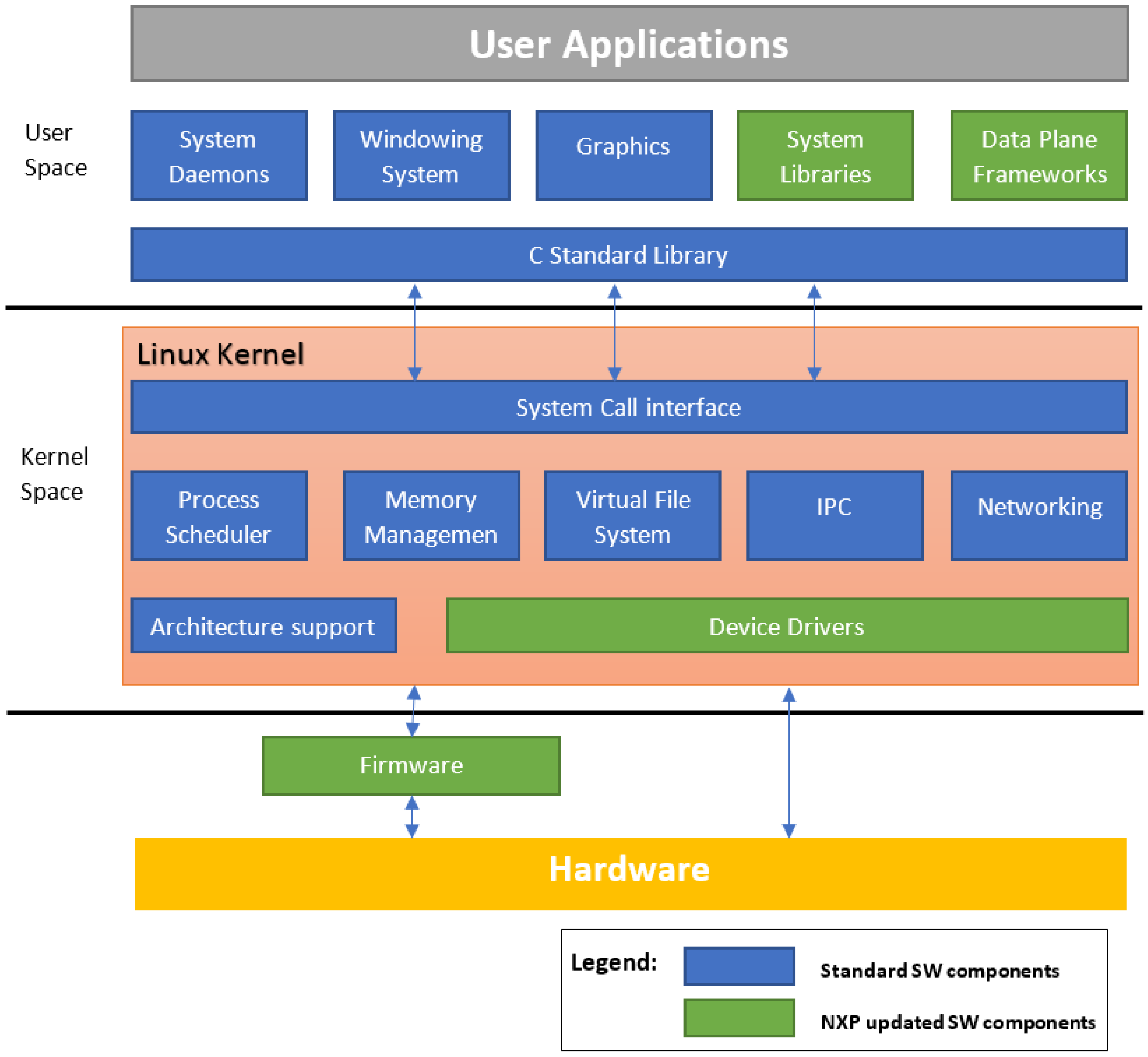


Figure 2. Typical Linux System

The Linux kernel was created in 1991 by Linus Torvalds and released as an open source project under GNU General Public License(GPL) version 2. It rapidly attracted developers around the world. In 2015 the Linux kernel has received contributions from nearly 12,000 programmers from more than 1,200 companies. The software is officially released on http://www.kernel.org website through downloadable packages and GIT repositories. A general Linux kernel introduction from kernel.org can also be found at https://www.kernel.org/doc/html/latest/admin-guide/README.html.

Kernel Releases and relationship with Layerscape SDK

There are different Linux kernel releases coming from different sources. Below we listed the ones that are related to the LSDK kernel.

Kernel.org official kernel releases

- **Mainline**

  Mainline tree is maintained by Linus Torvalds. It's the tree where all new features are introduced and where all the exciting new development happens. New mainline kernels are released every 2-3 months.

- **Longterm (LTS)**

  There are usually several "longterm maintenance" kernel releases provided for the purposes of backporting bugfixes for older kernel trees. Only important bugfixes are applied to such kernels and they don't usually see very frequent releases, especially for older trees.

  Refer to https://www.kernel.org/category/releases.html for the current maintained Longterm releases.

Linaro LSK kernel release

Linaro is an open organization focused on improving Linux on ARM. They are also providing a Linux kernel release called Linaro Stable Kernel (LSK). It is based on kernel.org Longterm kernel releases and included ARM related features developed by Linaro. Normally these features are generic kernel features for the ARM architecture. Please refer to https://wiki.linaro.org/LSK for more information about the LSK releases.

NXP Layerscape SDK kernel

NXP’s SDK kernel often contains patches that are not upstream yet so essentially the LSDK kernel is an enhanced Linaro LSK which is in turn an enhanced kernel.org LTS. In order to fully utilize the ARM open source eco-system. The kernel versions provided in NXP LSDK will be chosen from the kernel.org Longterm releases to include the important bugfixes backported. It will also include generic ARM kernel features provided by the Linaro LSK release which could be important for some users.

Getting the LSDK kernel source code

With Layerscape SDK, NXP owned/updated software components are published on github. You can use git commands to get the latest kernel source code.

- Install git command if not there already. For example, on Ubuntu:

  ```
  $ sudo apt-get install git
  ```

- Clone the Linux kernel source code with git.

  ```
  $ git clone https://source.codeaurora.org/external/qoriq/qoriq-components/linux
  ```

4.1 Configuring and building

Configuring and building the Linux kernel is controlled by the Kbuild sub-system. You can find documents describing the internal of Kbuild sub-system under the Documentation/kbuild/ folder in the Linux source code tree if you are adding new files or new configure options to the kernel. Otherwise as a user of Linux kernel, you probably only want to know how to fine tune the kernel configuration base on your system requirements and build new kernel image with updated configuration. These are done through *make* commands, below we will talk about `make` commands you probably need to know as a kernel user.

Environment setting for cross-compiling

This following settings are applicable when you are configuring and building kernel on a different architecture from the target. For example, compiling an ARMv8 kernel on an X86 computer. If you are compiling the kernel natively on a machine of the same architecture as the target, you should skip this chapter.

- Install the cross compiler of your distribution
- Specify the target architecture in ARCH environment variable
- Specify the prefix (and path) of a cross compiler in `CROSS_COMPILE` environment variable

  ```
  $ export CROSS_COMPILE=/path/to/dir/tool-chain-prefix-
  ```

  Or just the prefix if the cross-compiler commands are already in the execution `PATH`.

  ```
  $ export CROSS_COMPILE=tool-chain-prefix-
  ```

For example, the commands needed on Ubuntu Linux will be like:

- 64-bit ARM:

  ```
  $ sudo apt-get install gcc-aarch64-linux-gnu
  $ export CROSS_COMPILE=aarch64-linux-gnu-
  $ export ARCH=arm64
  ```

- 32-bit ARM (ARMv7 / 32-bit mode of ARMv8):

  ```
  $ sudo apt-get install gcc-arm-linux-gnueabihf
  $ export CROSS_COMPILE=arm-linux-gnueabihf-
  $ export ARCH=arm
  ```

For the shell environment variables exported above, you can also include them directly in each make command you use. E.g. `$ ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- make {targets}`. Exporting them will save effort if you are using make in kernel frequently.

Configuring kernel

The current kernel configuration for a kernel source tree will be kept in a hidden file named .config at the top level of the kernel source code after you changed the configuration with any of the make config command variants. You can copy it directly from one kernel source tree to another with the same kernel version to duplicate the configuration exactly. Also, you can edit it with a text editor, in which you can see a list of `CONFIG_*` symbols corresponding to each of the kernel configure option.

The following targets from the Linux kernel Kbuild framework are used to load the default kernel configuration for LS1028A BSP:

- `defconfig/${PLATFORM}_defconfig`

  Create the `.config` file by using the default config options of the architecture or platform defined in the `arch/$ARCH/configs/` directory. This normally includes all the device drivers needed for the architecture or platform.

- `${FRAGMENT}.config`

  Merge a configuration fragment that enables certain features into the .config file.

Specific command to load the default configuration of different platforms for LS1028A BSP will be:

- For Layerscape ARMv8 platforms in 64bit mode:

  ```
  $ make defconfig lsdk.config
  ```

- For Layerscape ARMv7 platforms:

  ```
  $ make multi_v7_defconfig multi_v7_lpae.config lsdk.config
  ```

- For Layerscape ARMv8 platforms in 32bit mode:

```
$ make multi_v7_defconfig multi_v7_lpae.config multi_v8.config lsdk.config
```

To further fine tune the configuration base on your system need you can use the following make commands.

- `$ make menuconfig`

  Choose configure options in text based color menus, radiolists & dialogs. It is a good way to navigate through all the selectable kernel configure options in a well-organized human-readable hierarchy and you can get a description of every option when it is highlighted by selecting the `<Help>` button. In the device driver part of this User's Manual we also provided the path to the configure options needed for a feature to work in the menuconfig.

- `$ make ${FRAGMENT}.config`

  You can also utilize this capability to enable options for a specific feature in your custom kernel configuration quickly without selecting each one of them in the menuconfig. In the device driver part of this User's Manual, we listed the `CONFIG_*` symbols needed by a specific feature/driver. Put these symbols with "=y" or "=m" depending on if you want these features/drivers to be built-in or built as loadable kernel module into a `${FEATURE}.config` file under `arch/$ARCH/configs/` directory. Run `$ make ${FEATURE}.config` command, it will enable all these listed kernel configure options together.

Building kernel

Building the kernel is simple.

- To build kernel images and device tree images.

```
make
```

- To build loadable kernel modules:

```
make modules
```

You can supply `-j <NUM>` option to the above make commands to spin `NUM` concurrent threads to reduce build time on multicore systems.

After a successful build:

- Compiled kernel images are in arch/${ARCH}/boot/ folder.
- Compiled device trees (dtb files) are in arch/${ARCH}/boot/dts folder.
- Compiled kernel modules are spread out in driver folders. You can extract them to a specific folder (e.g. /folder/to/install) by using command:

```
$ make modules_install INSTALL_MOD_PATH=/folder/to/install
```

Install new kernel and modules

The path or naming convention of kernel images and modules are different for different Linux distributions. The following instructions are based on the convention of LS1028A BSP.

Using the flex-build scripts

- Copy kernel image, dtb and kernel modules from your kernel tree to the staging folder of the flexbuild script (Skip if you are using the *flexbuild -c linux* to build the kernel directly).
  — **For 64-bit ARM:**

```
$ cp arch/arm64/boot/Image.gz ${path-to-flexbuild}/build/linux/kernel/arm64/
$ cp arch/arm64/boot/dts/freescale/*.dtb ${path-to-flexbuild}/build/linux/kernel/arm64/
$ make modules_install INSTALL_MOD_PATH=${path-to-flexbuild}/build/linux/kernel/arm64/
```

— **For 32-bit ARM:**

```
$ cp arch/arm/boot/Image.gz ${path-to-flexbuild}/build/linux/kernel/arm/
$ cp arch/arm/boot/dts/ls*.dtb ${path-to-flexbuild}/build/linux/kernel/arm/
$ make modules_install INSTALL_MOD_PATH=${path-to-flexbuild}/build/linux/kernel/arm/
```

- Regenerate the boot partition and rootfs (for commands below: `${ARCH} = arm32 | arm64`)

```
$ flex-builder -i mkbootpartition -a ${ARCH}
$ flex-builder -i merge-component -a ${ARCH}
$ flex-builder -i compressrfs -a ${ARCH}
```

- Use flex-installer to deploy the updated boot partition and rootfs to the device following the LS1028A BSP Quick Start on page 12.

Update the target filesystem directly

This can be more convenient if you are compiling the kernel on the target device locally or you can easily update the filesystem of target device remotely (e.g. using scp, tftp, or etc.).

- Copy your Image file to `/boot` folder on the target using `cp` if compiled locally; Use any available remote update approach if compiled remotely.
- Copy dtb files to `/boot` folder on the target using `cp` if compiled locally; Use any available remote update approach to do the same if compiled remotely.
- Update kernel modules. (Note: kernel modules are required to be updated when you updated the kernel image).
  — If you compiled the kernel on the target device locally. Use the command below:

```
$ make modules_install
```

  — If you compiled the kernel remotely. Do the following:
  - Install the modules into a temporary folder (e.g. `/tmp/lsdk/`).

```
$ make modules_install INSTALL_MOD_PATH=/tmp/lsdk/
```

  - Transfer the `lib/` directory from the temporary location above to the target device using any file transfer approach and put it in the / path of the filesystem.

4.2 Device Drivers

4.2.1 CAAM Direct Memory Access (DMA)

Description

The CAAM DMA module implements a DMA driver that uses the CAAM DMA controller to provide both SG and MEMCPY DMA capability to be used by the platform. It is based on the CAAM JR interface that must be enabled in the *kernel config* as a prerequisite for the CAAM DMA driver.

The driver is based on the DMA engine framework and it is located under the DMA Engine support category in the kernel config menu.

Kernel Configure Options

Tree Overview

To enable the CAAM DMA module, set the following options for `make menuconfig`:

```
-*- Cryptographic API --->
    [*] Hardware crypto devices --->
            <*> Freescale CAAM-Multicore driver backend
            <*>     Freescale CAAM Job Ring driver backend
Device Drivers --->
            <*> DMA Engine support --->
            <*>     CAAM DMA engine support
```

NOTE

Be aware that the CAAM DMA driver depends on the CAAM and CAAM JR drivers, which also have to be enabled.

Identifier

The following configure identifier is used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_CRYPTO_DEV_FSL_CAAM_DMA	y/m/n	n	CAAM DMA engine support

Device Tree Node

Below is an example device tree node required by this feature.

```
caam_dma {
    compatible = "fsl,sec-v5.4-dma";
};
```

Source Files

The following source file is related to this feature in the Linux kernel.

Source File	Description
drivers/dma/caam_dma.c	The CAAM DMA driver

Verification in Linux

On a successful probing, the driver will print the following message in `dmesg`:

```
 [    1.443940] caam-dma 1700000.crypto:caam_dma: caam dma support with 4 job rings
```

Additionally, you can also run the following commands:

```
ls -l /sys/class/dma/
total 0
lrwxrwxrwx 1 root root 0 Jan  1  1970 dma0chan0 -> ../../devices/platform/soc/1700000.crypto/
1700000.crypto:caam_dma/dma/dma0chan0
lrwxrwxrwx 1 root root 0 Jan  1  1970 dma0chan1 -> ../../devices/platform/soc/1700000.crypto/
1700000.crypto:caam_dma/dma/dma0chan1
lrwxrwxrwx 1 root root 0 Jan  1  1970 dma0chan2 -> ../../devices/platform/soc/1700000.crypto/
1700000.crypto:caam_dma/dma/dma0chan2
```

```
lrwxrwxrwx 1 root root 0 Jan  1  1970 dma0chan3 -> ../../devices/platform/soc/1700000.crypto/
1700000.crypto:caam_dma/dma/dma0chan3
```

Component Testing

To test both the SG and memcpy capability of the CAAM DMA driver use the dmatest module provided by the kernel.

Build dmatest

Build the dmatest utility as a module by running the command:

```
$ make menuconfig
```

Then select from the kernel menuconfig to build the dmatest.ko as a module:

```
Device Drivers --->
    <*> DMA Engine support --->
    <M>    DMA Test client
```

Configure dmatest

Before testing insert the module:

```
$ insmod dmatest.ko
```

The configure the dmatest. There is a general configuration that applies for both the sg and memcpy functionality:

```
$ echo 1 > /sys/module/dmatest/parameters/max_channels
$ echo 2000 > /sys/module/dmatest/parameters/timeout
$ echo 0 > /sys/module/dmatest/parameters/noverify
$ echo 4 > /sys/module/dmatest/parameters/threads_per_chan
$ echo 0 > /sys/module/dmatest/parameters/dmatest
$ echo 1 > /sys/module/dmatest/parameters/iterations
$ echo 2000 > /sys/module/dmatest/parameters/test_buf_size
```

The above configuration is self explanatory except a few:

If you set the 'noverify' parameter to 0 it will not perform check of the copied buffer at the end of each testing round. This should be used for performance testing. Set the 'noverify' parameter to 1 for functional testing.

Set the 'dmatest' parameter to 0 to test the memcpy functionality and to 1 to test the sg functionality.

Perform the test

To perform the test simply run the command:

```
$ echo 1 > /sys/module/dmatest/parameters/run
```

Depending on the type of test performed (sg/memcpy) the output may vary. Here is an example of output obtained with the above parameters:

```
[   72.113769] dmatest: Started 4 threads using dma0chan0
[   72.105334] dmatest: dma0chan0-copy0: summary 1 tests, 0 failures 9009 iops 9009 KB/s (0)
[   72.113649] dmatest: dma0chan0-copy1: summary 1 tests, 0 failures 119 iops 119 KB/s (0)
[   72.114927] dmatest: dma0chan0-copy2: summary 1 tests, 0 failures 24390 iops 0 KB/s (0)
[   72.115098] dmatest: dma0chan0-copy3: summary 1 tests, 0 failures 37037 iops 0 KB/s (0)
```

4.2.2 Enhanced Secured Digital Host Controller (eSDHC)

Description

The enhanced secured host controller (eSDHC) provides an interface between the host system and the SD/SDIO cards and eMMC devices.

The eSDHC device driver supports either kernel built-in or module.

Kernel Configure Options

Tree View

Kernel Configure Options Tree View	Description

```
Device Drivers --->
<*>     MMC/SD/SDIO card support --->
<*>       MMC block device driver
(8)          Number of minors per block device
[*]          Use bounce buffer for simple hosts
```
	Enables SD/MMC block device driver support

```
*** MMC/SD/SDIO Host Controller Drivers ***

<*>   Secure Digital Host Controller Interface
support
<*>   SDHCI platform and OF driver helper
[*]     SDHCI OF support for the NXP eSDHC
controller
```
	Enables NXP eSDHC driver support

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_MMC	y/n	y	Enable SD/MMC bus protocol
CONFIG_MMC_BLOCK	y/n	y	Enable SD/MMC block device driver support
CONFIG_MMC_BLOCK_MINORS	integer	8	Number of minors per block device
CONFIG_MMC_BLOCK_BOUNCE	y/n	y	Enable continuous physical memory for transmit
CONFIG_MMC_SDHCI	y/n	y	Enable generic sdhc interface
CONFIG_MMC_SDHCI_PLTFM	y/n	y	Enable common helper function support for sdhci platform and OF drivers
CONFIG_MMC_SDHCI_OF_ESDHC	y/n	y	Enable NXP eSDHC support

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/mmc/host/sdhci.c	Linux SDHCI driver support
drivers/mmc/host/sdhci-pltfm.c	Linux SDHCI platform devices support driver
drivers/mmc/host/sdhci-of-esdhc.c	Linux eSDHC driver

Device Tree Binding

Property	Type	Status	Description
compatible	String	Required	Should be 'fsl,esdhc'
reg	integer	Required	Register map

example:

```
esdhc: esdhc@1560000 {
    compatible = "fsl,ls1046a-esdhc", "fsl,esdhc";
    reg = <0x0 0x1560000 0x0 0x10000>;
    interrupts = <GIC_SPI 62 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&clockgen 2 1>;
    voltage-ranges = <1800 1800 3300 3300>;
    sdhci,auto-cmd12;
    big-endian;
    bus-width = <4>;
};
```

Verification in U-Boot

The U-Boot log

```
=> mmcinfo
Device: FSL_SDHC
Manufacturer ID: 74
OEM: 4a45
Name: SDC
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.5 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
=> mw.l 81000000 11111111 100
=> mw.l 82000000 22222222 100
=> cmp.l 81000000 82000000 100
word at 0x0000000081000000 (0x11111111) != word at 0x0000000082000000 (0x22222222)
Total of 0 word(s) were the same
=> mmc write 81000000 0 2

MMC write: dev # 0, block # 0, count 2 ... 2 blocks written: OK
=> mmc read 82000000 0 2

MMC read: dev # 0, block # 0, count 2 ... 2 blocks read: OK
```

```
=> cmp.l 81000000 82000000 100
Total of 256 word(s) were the same
=>
```

Verification in Linux

Set U-Boot environment

```
=> setenv hwconfig sdhc
```

The linux booting log

```
......
[    3.913163] sdhci: Secure Digital Host Controller Interface driver
[    3.919339] sdhci: Copyright(c) Pierre Ossman
[    3.931467] sdhci-pltfm: SDHCI platform and OF driver helper
[    3.938900] sdhci-esdhc 1560000.esdhc: No vmmc regulator found
[    3.944728] sdhci-esdhc 1560000.esdhc: No vqmmc regulator found
[    3.978676] mmc0: SDHCI controller on 1560000.esdhc [1560000.esdhc] using ADMA 64-bit
[    4.197784] mmc0: new high speed SDHC card at address b368
[    4.203502] mmcblk0: mmc0:b368 SDC   7.45 GiB
```

Partition the card with fdisk

```
~# fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.26.2).
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.
Created a new DOS disklabel with disk identifier 0x5a5f34b3.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p):

Using default response p.
Partition number (1-4, default 1):
First sector (2048-15628287, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-15628287, default 15628287):

Created a new partition 1 of type 'Linux' and of size 7.5 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() [  410.501876]  mmcblk0: p1
to re-read partition table.
Syncing disks.

~#
~# fdisk -l /dev/mmcblk0
Disk /dev/mmcblk0: 7.5 GiB, 8001683456 bytes, 15628288 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disklabel type: dos
Disk identifier: 0x5a5f34b3

Device         Boot Start      End  Sectors  Size Id Type
/dev/mmcblk0p1       2048 15628287 15626240  7.5G 83 Linux
```

Format the card with mkfs

```
~# mkfs.ext2 /dev/mmcblk0p1
mke2fs 1.42.9 (28-Dec-2013)
Discarding device blocks: [   37.611042] random: nonblocking pool is initialized
done
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
488640 inodes, 1953280 blocks
97664 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2000683008
60 block groups
32768 blocks per group, 32768 fragments per group
8144 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

~#
```

Mount, read, and write

```
~# mount /dev/mmcblk0p1 /mnt/
~# ls /mnt/
lost+found
~# cp -r /lib /mnt/
~# sync
~# ls /mnt/
lib  lost+found
~# umount /dev/mmcblk0p1
~# ls /mnt/
~#
```

Known Bugs, Limitations, or Technical Issues

1. Call trace of more than 120 seconds task blocking when running iozone to test card performance. This is not issue and use below command to disable the warning.

   ```
   echo 0 > /proc/sys/kernel/hung_task_timeout_secs
   ```

2. Layerscape boards could not provide a power cycle to SD card but according to SD specification, only a power cycle could reset the SD card working on UHS-I speed mode. When the card is on UHS-I speed mode, this hardware problem may cause unexpected result after board reset. The workaround is using power off/on instead of reset when using SD UHS-I card.

3. Transcend 8G class 10 SDHC card has some compatibility issue. It is observed it could not work on 50 MHz high-speed mode on LS2 boards, but other brand SD cards (Sandisk, Kingston, Sony ...) worked fine. Reducing SD clock frequency could also resolve the issue. The workaround is using other kind SD cards instead.
4. After sleep of LS1046ARDB, the card will get below interrupt timeout issue. This is hardware issue. CMD18 (multiple blocks read) has hardware interrupt timeout issue.

```
mmc0: Timeout waiting for hardware interrupt.
```

5. Linux MMC stack does not have SD UHS-II support currently. It could not handle SD UHS-II card well. If UHS-I support is enabled in eSDHC dts node, the driver may make SD UHS-II card enter 1.8v mode. Only a power cycle could reset the card, so use power off/on instead of reset for SD UHS-II card if UHS-I support is enabled in eSDHC dts node.
6. For LS1012ARDB RevD and later versions, I2C reading for DIP switch setting is not reliable so U-Boot could not enable/disable SDHC2 automatically. If SDHC2 is used, "esdhc1" should be set in U-Boot hwconfig environment to enable it manually.

4.2.3 IEEE 1588

Description

From IEEE-1588 perspective, the components required are:

1. IEEE-1588 extensions to the ENETC driver and Felix switch driver.
2. A stack application for IEEE-1588 protocol.

IEEE 1588 device driver supports either kernel built-in or module.

Kernel Configure Options

Tree View

1. ENETC

Kernel Configure Tree View Options	Description

`Device Drivers  --->`
`[*] Network device support   --->`
`[*]   Ethernet driver support   --->`
`[*]   Freescale devices`
`<*>     ENETC PF driver`
`-*-     ENETC PTP clock driver`
`[*]     ENETC hardware timestamping support`
	Enable 1588 driver for ENETC

2. Felix switch

Kernel Configure Tree View Options	Description

`Device Drivers  --->`
`[*] Network device support   --->`
`[*]   Ethernet driver support   --->`
`[*]   Microsemi devices`
	Enable 1588 driver for Felix switch

<table>
<tr><th>Kernel Configure Tree View Options</th><th>Description</th></tr>
<tr><td><pre>
<*>   FELIX switch driver
[*]   FELIX switch PTP clock support
</pre></td><td></td></tr>
</table>

Compile-time Configuration Options

1. ENETC

Option	Values	Default Value	Description
CONFIG_FSL_ENETC	y/n/m	y	Enable ENETC driver support
CONFIG_FSL_ENETC_PTP_CLOCK	y/n/m	y	Enables PTP driver support
FSL_ENETC_HW_TIMESTAMPING	y/n	y	Enable timestamping support

2. Felix switch

Option	Values	Default Value	Description
CONFIG_MSCC_FELIX_SWITCH	y/n/m	y	Enable Felix switch driver support
CONFIG_MSCC_FELIX_SWITCH_PTP_CLOCK	y/n/m	n	Enables PTP driver support

Source Files

The driver source is maintained in the Linux kernel source tree.

1. ENETC

Source File	Description
drivers/net/ethernet/freescale/enetc/enetc_pf.c	ENETC driver
drivers/net/ethernet/freescale/enetc/enetc_ptp.c	ENETC PTP driver

2. Felix switch

Source File	Description
drivers/net/ethernet/mscc/felix_board.c	Felix switch driver
drivers/net/ethernet/mscc/felix_ptp.c	Felix PTP driver

Device Tree Binding

1. ENETC

Property	Type	Status	Description
reg	integer	Required	Register map

Example:

```
pci@0,4 {
reg = <0x000400 0 0 0 0>;
clocks = <&clockgen 4 0>;
little-endian;
};
```

2. Felix switch

   NA

Verification in Linux

Connect Ethernet interfaces of two boards with back-to-back method (for example, eth0 to eth0).

One board runs as master and the other one runs as slave.

- **Check PTP clock and timestamping capability**

```
# ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
  hardware-transmit (SOF_TIMESTAMPING_TX_HARDWARE)
  hardware-receive (SOF_TIMESTAMPING_RX_HARDWARE)
  hardware-raw-clock (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
  off (HWTSTAMP_TX_OFF)
  on (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
  none (HWTSTAMP_FILTER_NONE)
  all (HWTSTAMP_FILTER_ALL)
```

- **On the master side**

```
# ifconfig eth0 up
# ifconfig eth0 192.168.1.100
# ptp4l -i eth0 -p /dev/ptp1 -m
```

- **On the slave side**

```
# ifconfig eth0 up
# ifconfig eth0 192.168.1.200
# ptp4l -i eth0 -p /dev/ptp1 -s -m
```

  The slave side would print synchronization messages.

- **Note:**

  For Felix switch, ptp4l works on both, CPU port with bridge case and CPU port without bridge case. However, ptp4l should only work on L2 Ethernet protocol with -2 option for CPU port with bridge case.

Known Bugs, Limitations, or Technical Issues

- The Felix switch interrupt had not been implemented in driver. Polling method in work queue was used instead temporary for timestamping support.

- The Felix switch extraction is working on all packets. Therefore, the L2 forwarding in CPU port with bridge case is actually software forwarding, not hardware forwarding.The frame identifying should be supported so that extraction works only on PTP packets.

4.2.4 Low Power Universal Asynchronous Receiver/Transmitter (LPUART)

Description

Low Power Universal asynchronous receiver/transmitter (LPUART) is a high speed and low-power UART. Refer to below table for the NXP SoCs that can support LPUART.

SoC	Num of LPUART module
LS1021A	6
LS1043A	6

U-Boot ConfigurationCompile time options

Below are major U-Boot configuration options related to this feature defined in platform specific config files under include/configs/ directory.

Option Identifier	Description
CONFIG_LPUART	Enable LPUART support
CONFIG_FSL_LPUART	Enable NXP LPUART support
CONFIG_LPUART_32B_REG	Select 32-bit LPUART register mode

Choosing predefined U-Boot board configs:

Please make the defconfig include 'lpuart', such as: ls1021atwr_nor_lpuart_defconfig. This will support LPUART.

Runtime options

Env Variable	Env Description	Sub option	Option Description
bootargs	Kernel command line argument passed to kernel	console=ttyLP0,1152000	select LPUART0 as the system console

Kernel Configure Options

Tree View

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel

Kernel Configure Tree View Options	Description

```
Device Drivers --->

    Character devices --->
```
	LPUART driver and enable console support

Kernel Configure Tree View Options	Description

`Serial drivers --->`
`<*> Freescale lpuart serial port support`
`[*] Console on Freescale lpuart serial port`	

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_SERIAL_FSL_LPUART	y/m/n	n	LPUART Driver

Device Tree Binding

Below is an example device tree node required by this feature. Note that it may have differences among platforms.

```
lpuart0: serial@2950000 {
                        compatible = "fsl,vf610-lpuart";
                        reg = <0x0 0x2950000 0x0 0x1000>;
                        interrupts = <GIC_SPI 80 IRQ_TYPE_LEVEL_HIGH>;
                        clocks = <&sysclk>;
                        clock-names = "ipg";
                        fsl,lpuart32;
                        status = "okay";
```

Source Files

The following source file are related to this feature in U-Boot.

Source File	Description
drivers/serial/serial_lpuart.c	The LPUART driver file

The following source file are related to this feature in Linux kernel.

Source File	Description
drivers/tty/serial/fsl_lpuart.c	The LPUART driver file

Verification in U-Boot

1. Boot up U-Boot from bank0, and update rcw and U-Boot for LPUART support to bank4, first copy the rcw and U-Boot binary to the TFTP directory.
2. Please refer to the platform deploy document to update the rcw and U-Boot.

3. After all is updated, run U-Boot command to switch to alt bank, then will bring up the new U-Boot to the LPUART console.

```
CPU:   Freescale LayerScape LS1020E, Version: 1.0, (0x87081010)
Clock Configuration:
       CPU0(ARMV7):1000 MHz,
       Bus:300  MHz, DDR:800  MHz (1600 MT/s data rate),
Reset Configuration Word (RCW):
       00000000: 0608000a 00000000 00000000 00000000
       00000010: 60000000 00407900 e0025a00 21046000
       00000020: 00000000 00000000 00000000 08038000
       00000030: 00000000 001b7200 00000000 00000000
I2C:   ready
Board: LS1021ATWR
CPLD:  V2.0
PCBA:  V1.0
VBank: 0
DRAM:  1 GiB
Using SERDES1 Protocol: 48 (0x30)
Flash: 0 Bytes
MMC:   FSL_SDHC: 0
EEPROM: NXID v16777216
PCIe1: Root Complex no link, regs @ 0x3400000
PCIe2: disabled
In:    serial
Out:   serial
Err:   serial
SATA link 0 timeout.
AHCI 0001.0300 1 slots 1 ports ? Gbps 0x1 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc
Found 0 device(s).
SCSI:  Net:   eTSEC1 is in sgmii mode.
eTSEC2 is in sgmii mode.
eTSEC1, eTSEC2 [PRIME], eTSEC3
=>
```

Verification in Linux

1. After uboot startup, set the command line parameter to pass to the linux kernel including console=ttyLP0,115200 in boootargs. For deploy the ramdisk as rootfs, the bootargs can be set as: "set bootargs root=/dev/ram0 rw console=ttyLP0,115200"

```
=> set bootargs root=/dev/ram0 rw console=ttyLP0,115200

=> dhcp 81000000 <tftpboot dir>/zImage.ls1021a;tftp 88000000 <tftpboot dir>/
initrd.ls1.uboot;tftp 8f000000 <tftpboot dir>/ls1021atwr.dtb;bootz 81000000 88000000 8f000000

[...]

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0xf00
Linux version 3.12.0+ (xxx@rock) (gcc version 4.8.3 20131202 (prerelease) (crosstool-NG
linaro-1.13.1-4.8-2013.12 - LinaroGCC 2013.11) ) #664 SMP Tue Jun 24 15:30:45 CST 2014
CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=30c73c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: Freescale Layerscape LS1021A, model: LS1021A TWR Board
```

```
Memory policy: ECC disabled, Data cache writealloc
PERCPU: Embedded 7 pages/cpu @8901c000 s7936 r8192 d12544 u32768
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 520720
Kernel command line: root=/dev/ram rw console=ttyLP0,115200
PID hash table entries: 4096 (order: 2, 16384 bytes)

[...]

ls1021atwr login: root
root@ls1021atwr:~#
```

2. After the kernel boot up to the console, you can type any shell command in the LPUART TERMINAL.

4.2.5 Flex Serial Peripheral Interface (FlexSPI)

U-Boot Configuration

Make sure your boot mode support FlexSPI.

Use FlexSPI boot mode to boot on board, please check the board user manual and boot from FlexSPI. (or some other boot mode decide by your board.)

Following Config options needs to be enabled for FlexSPI.

- CONFIG_NXP_FSPI=y
- CONFIG_FSPI_AHB_EN_4BYTE=y
- CONFIG_SYS_FSPI_AHB_INIT=y

Kernel Configure Tree View Options

```

Device Drivers --->
     Memory Technology Device (MTD) support
     RAM/ROM/Flash chip drivers --->
           < > Detect flash chips by Common Flash Interface (CFI) probe
           < > Detect non-CFI AMD/JEDEC-compatible flash chips
           < > Support for RAM chips in bus mapping
           < > Support for ROM chips in bus mapping
           < > Support for absent chips in bus mapping
     Self-contained MTD device drivers --->
           <*> Support most SPI Flash chips (AT26DF, M25P, W25X, ...)
     < >   NAND Device Support  ---
     <*>     SPI-NOR device support --->
             the framework for SPI-NOR support
     <*>     NXP Flex SPI controller
```

```
CONFIG_SPI_NXP_FLEXSPI:
This enabled support for the FlexSPI controller in master mode.


Symbol: SPI_NXP_FLEXSPI [=y]
Type  : tristate
Prompt: NXP Flex SPI controller
Location:
    -> Device Drivers
      -> Memory Technology Device (MTD) support (MTD [=y])
```

```
        -> SPI-NOR device support (MTD_SPI_NOR [=y])
  Depends on: MTD [=y] && MTD_SPI_NOR [=y]
```

Compile-time Configuration Options

Config	Values	Defualt Value	Description
`CONFIG_SPI_NXP_FLEXSPI`	`y/n`	`y`	Enable FlexSPI module
`CONFIG_MTD_SPI_NOR_BASE`	`y/n`	`y`	Enables the framework for SPI-NOR

Verification in U-Boot

```
=> sf probe 0:0
SF: Detected mt35xu512g with page size 256 Bytes, erase size 128 KiB, total 64 MiB
=> sf erase 0 100000
SF: 1048576 bytes @ 0x0 Erased: OK
=> sf write 82000000 0 1000
SF: 4096 bytes @ 0x0 Written: OK
=> sf read 81100000 0 1000
SF: 4096 bytes @ 0x0 Read: OK
=> cm.b 81100000 82000000 1000
Total of 4096 byte(s) were the same
```

Verification in Linux:

```
The booting log

......
nxp-fspi 20c0000.flexspi: mt35xu512aba (65536 Kbytes)
nxp-fspi 20c0000.flexspi: mt35xu512aba (65536 Kbytes)
......

Erase the FlexSPI flash

~ # mtd_debug erase /dev/mtd0 0x00000000 1048576
Erased 1048576 bytes from address 0x00000000 in flash


Write the FlexSPI flash

~ # dd if=/bin/ls.coreutils of=tp bs=4096 count=1
~ # mtd_debug write /dev/mtd0 0 4096 tp
Copied 4096 bytes from tp to address 0x00000000 in flash


Read the FlexSPI flash
```

```
~ # mtd_debug read /dev/mtd0 0 4096 dump_file

Copied 4096 bytes from address 0x00000000 in flash to dump_file

Check Read and Write

Use compare tools(yacto has tools named diff).
~ # diff tp dump_file
~ #
If diff command has no print log, the FlexSPI verification is passed.
```

4.2.6 Real Time Clock (RTC)

Linux SDK for QorIQ Processors

Description
Provides the RTC function.

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description

`Device Drivers->`
`      Real Time Clock-->`
`          [*] Set system time from RTC on startup and resume (new)`
`          (rtc0) RTC used to set the system time (new)`
`          <[*] /sys/class/rtc/rtcN (sysfs)`
`          <[*] /proc/driver/rtc (procfs for rtc0)`
`          <[*] /dev/rtcN (character devices)`
	Enable RTC driver

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_RTC_LIB	y/m/n	y	Enable RTC lib
CONFIG_RTC_CLASS	y/m/n	y	Enable generic RTC class support
CONFIG_RTC_HCTOSYS	y/n	y	Set the system time from RTC when startup and resume
CONFIG_RTC_HCTOSYS_DEVICE		"rtc0"	RTC used to set the system time
CONFIG_RTC_INTF_SYSFS	y/m/n	y	Enable RTC to use sysfs

Table continues on the next page...

Table continued from the previous page...

Option	Values	Default Value	Description
CONFIG_RTC_INTF_PROC	y/m/n	y	Use RTC through the proc interface
CONFIG_RTC_INTF_DEV	y/m/n	y	Enable RTC to use /dev interface

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/rtc/	Linux RTC driver

Device Tree Binding

Preferred node name: rtc

Property	Type	Status	Description
compatible	string	Required	Should be "dallas,ds3232"

Default node:

```
    i2c@3000 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "fsl-i2c";
                reg = <0x3000 0x100>;
                interrupts = <43 2>;
                interrupt-parent = <&mpic>;
                dfsrr;
                rtc@68 {
                        compatible = "dallas,ds3232";
                        reg = <0x68>;
                };
              };
```

Verification in Linux

Here is the rtc booting log

```
      ...
      rtc-ds3232 1-0068: rtc core: registered ds3232 as rtc0
      MC object device driver dpaa2_rtc registered
      rtc-ds3232 0-0068: setting system clock to 2000-01-01 00:00:51 UTC (946684851)
      ...

NOTE: Please refer to the related DTS file to enable the RTC driver before building.
```

```
      For example, LS2080AQDS board, should enable the below option:
      <*> Dallas/Maxim DS3232
```

Change the RTC time in Linux Kernel

```

        ~ # ls /dev/rtc -l
        lrwxrwxrwx    1 root     root            4 Jan 11 17:55 /dev/rtc -> rtc0
        ~ # date
        Sat Jan  1 00:01:38 UTC 2000
        ~ # hwclock
        Sat Jan  1 00:01:41 2000  0.000000 seconds
        ~ # date 011115502011
        Tue Jan 11 15:50:00 UTC 2011
        ~ # hwclock -w
        ~ # hwclock
        Tue Jan 11 15:50:36 2011  0.000000 seconds
        ~ # date 011115502010
        Mon Jan 11 15:50:00 UTC 2010
        ~ # hwclock -s
        ~ # date
        Tue Jan 11 15:50:49 UTC 2011
        ~ #

        NOTE: Before using the rtc driver, make sure the /dev/rtc node in your file system is
correct. Otherwise, you need to make correct node for /dev/rtc
```

4.2.7 Queue Direct Memory Access Controller (qDMA)

The qDMA controller transfers blocks of data between one source and one destination. The blocks of data transferred can be represented in memory as contiguous or noncontiguous using scatter/gather table(s). Channel virtualization is supported through enqueuing of DMA jobs to, or dequeuing DMA jobs from, different work queues.

QDMA can support Layerscape platform with DPAA1 or DPAA2.

QDMA for platform with DPAA1

Kernel Configure Options

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel.

Kernel Configure Tree View Options	Description

`Device Drivers --->`
`         [*] DMA Engine support  ---> --->`
`            <*> Freescale qDMA engine support`
	Support the Freescale qDMA engine with command queue and legacy mode.
	
	Channel virtualization is supported through enqueuing of DMA jobs to,
	
	or dequeuing DMA jobs from, different work queues.
	
	This module can be found on Freescale LS SoCs.

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_FSL_QDMA	y/m/n	n	qDMA driver

Device Tree Binding

Device Tree Node

Below is an example device tree node required by this feature. Note that it may has differences among platforms.

```
                qdma: qdma@8380000 {
                        compatible = "fsl,ls1046a-qdma", "fsl,ls1021a-qdma";
                        reg = <0x0 0x8380000 0x0 0x1000>, /* Controller regs */
                              <0x0 0x8390000 0x0 0x10000>, /* Status regs */
                              <0x0 0x83a0000 0x0 0x40000>; /* Block regs */
                        interrupts = <0 153 0x4>,
                                     <0 39 0x4>;
                        interrupt-names = "qdma-error", "qdma-queue";
                        channels = <8>;
                        queues = <2>;
                        status-sizes = <64>;
                        queue-sizes = <64 64>;
                        big-endian;
                };
```

Source File

The following source files are related the feature in Linux kernel.

Source File	Description
drivers/dma/fsl-qdma.c	The qDMA driver file

Verification in Linux

```
root@ls1043ardb:~# echo 1024 > /sys/module/dmatest/parameters/test_buf_size;
root@ls1043ardb:~# echo 4 > /sys/module/dmatest/parameters/threads_per_chan;
root@ls1043ardb:~# echo 2 > /sys/module/dmatest/parameters/max_channels;
root@ls1043ardb:~# echo 100 > /sys/module/dmatest/parameters/iterations;
root@ls1043ardb:~# echo 1 > /sys/module/dmatest/parameters/run

[   32.498138] dmatest: Started 4 threads using dma0chan0
[   32.503430] dmatest: Started 4 threads using dma0chan1
[   32.508939] dmatest: Started 4 threads using dma0chan2
[   32.520073] dmatest: dma0chan0-copy0: summary 100 tests, 0 failures 4904 iops 2452 KB/s (0)
[   32.520076] dmatest: dma0chan0-copy2: summary 100 tests, 0 failures 4923 iops 2461 KB/s (0)
[   32.520079] dmatest: dma0chan0-copy3: summary 100 tests, 0 failures 4928 iops 2661 KB/s (0)
[   32.520176] dmatest: dma0chan0-copy1: summary 100 tests, 0 failures 4892 iops 2446 KB/s (0)
[   32.526438] dmatest: dma0chan1-copy0: summary 100 tests, 0 failures 4666 iops 2240 KB/s (0)
[   32.526441] dmatest: dma0chan1-copy2: summary 100 tests, 0 failures 4675 iops 2291 KB/s (0)
[   32.526469] dmatest: dma0chan1-copy3: summary 100 tests, 0 failures 4674 iops 2197 KB/s (0)
[   32.529610] dmatest: dma0chan2-copy1: summary 100 tests, 0 failures 5168 iops 2791 KB/s (0)
[   32.529613] dmatest: dma0chan2-copy0: summary 100 tests, 0 failures 5164 iops 2478 KB/s (0)
[   32.529754] dmatest: dma0chan2-copy3: summary 100 tests, 0 failures 5215 iops 2555 KB/s (0)
[   32.529756] dmatest: dma0chan2-copy2: summary 100 tests, 0 failures 5211 iops 2709 KB/s (0)
[   32.537881] dmatest: dma0chan1-copy1: summary 100 tests, 0 failures 3044 iops 1461 KB/s (0) (0)
dmatest: dma0chan0-copy3: summary 1000 tests, 0 failures 4078 iops 33474 KB/s (0)
```

```
dmatest: dma0chan0-copy0: summary 1000 tests, 0 failures 3024 iops 24486 KB/s (0)
dmatest: dma0chan0-copy2: summary 1000 tests, 0 failures 2881 iops 23588 KB/s (0)
```

QDMA for platform with DPAA1

Kernel Configure Options

Below are the configure options need to be set/unset while doing "make menuconfig" for kernel.

Kernel Configure Tree View Options	Description

`Device Drivers --->`
`    [*] DMA Engine support  ---> --->`
`        <*>    NXP DPAA2 QDMA`
	NXP Data Path Acceleration
	
	Architecture 2 QDMA driver, using the NXP MC bus driver.

Identifier

Below are the configure identifiers which are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_FSL_DPAA2_QDMA	y/m/n	n	qDMA driver

Source Files

The following source files are related the feature in Linux kernel.

Source File	Description
drivers/dma/dpaa2-qdma/*	The qDMA driver file

Verification in Linux

```
Create DPDMAI object using restool:

restool dpdmai create --priorities=2,5
restool dprc assign dprc.1 --object=dpdmai.0 --plugged=1

Configure parameters for dmatest and run it:

echo 8 > /sys/module/dmatest/parameters/test_flag
echo 100 > /sys/module/dmatest/parameters/sg_size
echo 10000 > /sys/module/dmatest/parameters/iterations
echo 1 > /sys/module/dmatest/parameters/threads_per_chan
echo 8 > /sys/module/dmatest/parameters/max_channels
echo 64 > /sys/module/dmatest/parameters/test_buf_size
echo 1 > /sys/module/dmatest/parameters/run

Example log:

root@ls2085ardb:~# echo 8 > /sys/module/dmatest/parameters/test_flag
root@ls2085ardb:~# echo 10 > /sys/module/dmatest/parameters/iterations
root@ls2085ardb:~# echo 2 > /sys/module/dmatest/parameters/threads_per_chan
root@ls2085ardb:~# echo 32384 > /sys/module/dmatest/parameters/test_buf_size
root@ls2085ardb:~# echo 4 > /sys/module/dmatest/parameters/max_channels
```

```
root@ls2085ardb:~# echo 1 > /sys/module/dmatest/parameters/run
[   68.460353] dmatest: Started 2 threads using dma0chan0
[   68.465549] dmatest: Started 2 threads using dma0chan1
[   68.465755] dmatest: dma0chan0-sg0: summary 10 tests, 0 failures 1847 iops 422686 KB/s (0)
[   68.465963] dmatest: dma0chan0-sg1: summary 10 tests, 0 failures 1786 iops 367095 KB/s (0)
[   68.470694] dmatest: dma0chan1-sg0: summary 10 tests, 0 failures 1938 iops 608838 KB/s (0)
[   68.470987] dmatest: dma0chan1-sg1: summary 10 tests, 0 failures 1843 iops 517419 KB/s (0)
[   68.503858] dmatest: Started 2 threads using dma0chan2
[   68.509042] dmatest: Started 2 threads using dma0chan3
[   68.509255] dmatest: dma0chan2-sg0: summary 10 tests, 0 failures 1849 iops 549944 KB/s (0)
[   68.509454] dmatest: dma0chan2-sg1: summary 10 tests, 0 failures 1789 iops 473514 KB/s (0)
[   68.514518] dmatest: dma0chan3-sg1: summary 10 tests, 0 failures 1830 iops 414714 KB/s (0)
[   68.515016] dmatest: dma0chan3-sg0: summary 10 tests, 0 failures 1670 iops 512859 KB/s (0)
```

4.2.8 Serial Advanced Technology Attachment (SATA)

Description

The driver supports NXP native SATA controller.

Module Loading

SATA driver supports either kernel built-in or module.

Kernel Configure Tree View Options	Description

`Device Drivers--->`
` <*> Serial ATA and Parallel ATA drivers  --->`
`--- Serial ATA and Parallel ATA drivers`
`<*>   AHCI SATA support`
`<*>   Freescale QorIQ AHCI SATA`
`support`
	Enables SATA controller support on ARM-based SoCs

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_SATA_AHCI=y	y/m/n	y	Enables SATA controller
CONFIG_SATA_AHCI_QORIQ=y	y/m/n	y	Enables SATA controller

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/ata/ahci_qoriq.c	Platform AHCI SATA support

Test Procedure

```
Please follow the following steps to use USB in Simics
(1) Boot up the kernel
...
fsl-sata ffe18000.sata: Sata FSL Platform/CSB Driver init
scsi0 : sata_fsl
ata1: SATA max UDMA/133 irq 74
fsl-sata ffe19000.sata: Sata FSL Platform/CSB Driver init
scsi1 : sata_fsl
ata2: SATA max UDMA/133 irq 41
...
(2) The disk will be found by kernel.
...
ata1: Signature Update detected @ 504 msecs
ata2: No Device OR PHYRDY change,Hstatus = 0xa0000000
ata2: SATA link down (SStatus 0 SControl 300)
ata1: SATA link up 1.5 Gbps (SStatus 113 SControl 300)
ata1.00: ATA-8: WDC WD1600AAJS-22WAA0, 58.01D58, max UDMA/133
ata1.00: 312581808 sectors, multi 0: LBA48 NCQ (depth 16/32)
ata1.00: configured for UDMA/133
scsi 0:0:0:0: Direct-Access     ATA      WDC WD1600AAJS-2 58.0 PQ: 0 ANSI: 5
sd 0:0:0:0: [sda] 312581808 512-byte logical blocks: (160 GB/149 GiB)
sd 0:0:0:0: Attached scsi generic sg0 type 0
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO
 or FUA
 sda: sda1 sda2 sda3 sda4 < sda5 sda6 >
sd 0:0:0:0: [sda] Attached SCSI disk

(3)play with the disk according to the following log.
[root@ls1046 root]# fdisk -l /dev/sda
Disk /dev/sda: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

    Device Boot      Start         End      Blocks  Id System
/dev/sda1               1         237     1903671  83 Linux
/dev/sda2             238         480     1951897+ 82 Linux swap
/dev/sda3             481        9852    75280590  83 Linux
/dev/sda4            9853       19457    77152162+  f Win95 Ext'd (LBA)
/dev/sda5            9853       14655    38580066  83 Linux
/dev/sda6           14656       19457    38572033+ 83 Linux
[root@ls1046 root]#
[root@ls1046 root]# mke2fs /dev/sda1
mke2fs 1.41.4 (27-Jan-2009)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
65280 inodes, 261048 blocks
13052 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8160 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376
```

```
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 22 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@ls1046 root]#
[root@ls1046 root]# mkdir sata
[root@ls1046 root]# mount /dev/sda1 sata
[root@ls1046 root]# ls sata/
lost+found
[root@ls1046 root]# cp /bin/busybox sata/
[root@ls1046 root]# umount sata/
[root@ls1046 root]# mount /dev/sda1 sata/
[root@ls1046 root]# ls sata/
busybox     lost+found
[root@ls1046 root]# umount sata/
[root@ls1046 root]# mount /dev/sda3 /mnt
[root@ls1046 root]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
rootfs               852019676 794801552  13937948  99% /
/dev/root            852019676 794801552  13937948  99% /
tmpfs                  1036480        52   1036428   1% /dev
shm                    1036480         0   1036480   0% /dev/shm
/dev/sda3             74098076   4033092  66300956   6% /mnt
```

Known Bugs, Limitations, or Technical Issues

- CDROM is not supported due to the silicon limitation

4.2.9 Security Engine (SEC)

SEC Device Drivers

Introduction and Terminology

The Linux kernel contains a Scatterlist Crypto API driver for the NXP SEC v4.x, v5.x security hardware blocks.

It integrates seamlessly with in-kernel crypto users, such as IPsec, in a way that any IPsec suite that configures IPsec tunnels with the kernel will automatically use the hardware to do the crypto.

SEC v5.x is backward compatible with SEC v4.x hardware, so one can assume that subsequent SEC v4.x references include SEC v5.x hardware, unless explicitly mentioned otherwise.

SEC v4.x hardware is known in Linux kernel as 'caam', after its internal block name: Cryptographic Accelerator and Assurance Module.

There are several HW interfaces ("backends") that can be used to communicate (i.e. submit requests) with the engine, their availability depends on the SoC:

- Register Interface (RI) - available on all SoCs (though access from kernel is restricted on DPAA2 SoCs)

  Its main purpose is debugging (for e.g. single-stepping through descriptor commands), though it is used also for RNG initialization.

- Job Ring Interface (JRI) - legacy interface, available on all SoCs; on most SoCs there are 4 rings

  Note: there are cases when fewer rings are accessible / visible in the kernel - for e.g. when firmware like Primary Protected Application (PPA) reserves one of the rings.

- Queue Interface (QI) - available on SoCs implementing DPAA v1.x (Data Path Acceleration Architecture)

Requests are submitted indirectly via Queue Manager (QMan) HW block that is part of DPAA1.

- Data Path SEC Interface (DPSECI) - available on SoCs implementing DPAA v2.x

  Similar to QI, requests are submitted via Queue Manager (QMan) HW block; however, the architecture is different - instead of using the platform bus, the Management Complex (MC) bus is used, MC firmware performing needed configuration to link DP* objects - see DPAA2 Linux Software chapter for more details.

NXP provides device drivers for all these interfaces. Current chapter is focused on JRI, though some general / common topics are also covered. For QI and DPSECI backends and compatible frontends, please refer to the dedicated chapters: for DPAA1, Security Engine for DPAA2.

On top of these backends, there are the "frontends" - drivers that sit between the Linux Crypto API and backend drivers. Their main tasks are to:

- register supported crypto algorithms
- process crypto requests coming from users (via the Linux Crypto API) and translate them into the proper format understood by the backend being used
- forward the CAAM engine responses from the backend being used to the users

Note: It is obvious that QI and DPSECI backends cannot co-exist (they can be compiled in the same "multi-platform" kernel image, however run-time detection will make sure only the proper one is active). However, JRI + QI and JRI + DPSECI are valid combinations, and both backends will be active if enabled; if a crypto algorithm is supported by both corresponding frontends (for e.g. both *caamalg* and *caamalg_qi* register cbc(aes)), a user requesting cbc(aes) will be bound to the implementation having the highest "crypto algorithm priority". If the user wants to use a specific implementation:

- it is possible to ask for it explicitly by using the specifc (unique) "driver name" instead of the generic "algorithm name" - please see official Linux kernel Crypto API documentation (section Crypto API Cipher References And Priority); currently default priorities are: 3000 for JRI frontend and 2000 for QI and DPSECI frontends
- crypto algorithm priority could be changed dynamically using the "Crypto use configuration API" (provided that CONFIG_CRYPTO_USER is enabled); one of the tools available that is capable to do this is "Linux crypto layer configuration tool" and an example of increasing the priority of QI frontend based implementation of echainiv(authenc(hmac(sha1),cbc(aes))) algorithm is:

```
$ ./crconf update driver "echainiv-authenc-hmac-sha1-cbc-aes-caam-qi" type 3 priority 5000
```

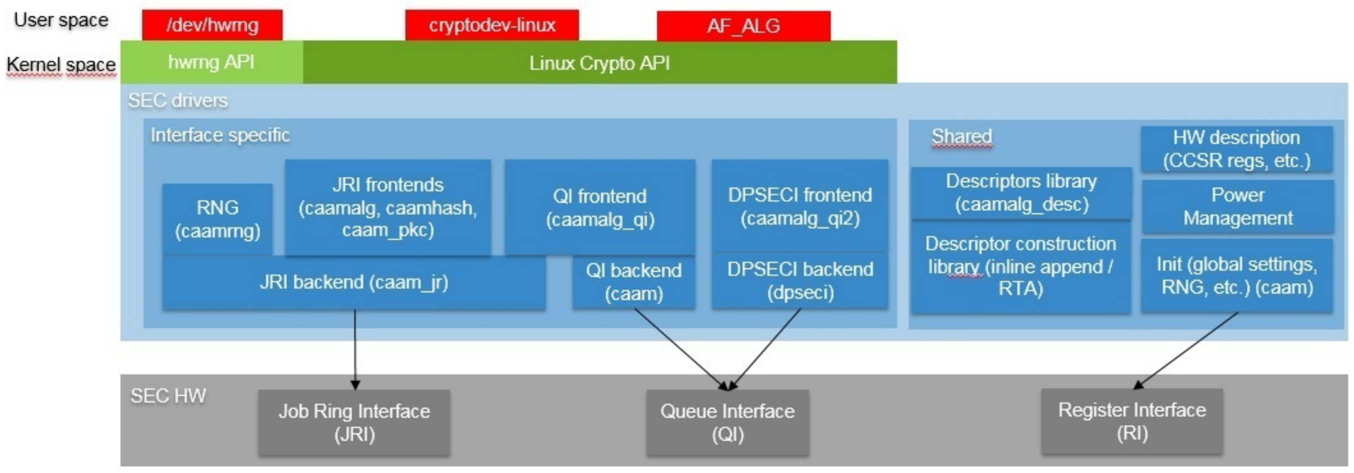


Figure 3. Linux kernel - SEC device drivers overview

Source Files

The drivers source code is maintained in the Linux kernel source tree, under *drivers/crypto/caam*. Below is a non-exhaustive list of files, mapping to Security Engine (SEC)(some files have been omitted since their existence is justified only by driver logic / design):

Source File(s)	Description	Module name
ctrl.[c,h]	Init (global settings, RNG, power management etc.)	caam
desc.h	HW description (CCSR registers etc.)	N/A
desc_constr.h	Inline append - descriptor construction library	N/A
caamalg_desc.[c,h]	(Shared) Descriptors library (symmetric encryption, AEAD)	caamalg_desc
caamrng.c	RNG (runtime)	caamrng
jr.[c,h]	JRI backend	caam_jr
qi.[c,h]	QI backend	caam
dpseci.[c,h], dpseci_cmd.h	DPSECI backend	N/A (built-in)
caamalg.c	JRI frontend (symmetric encryption, AEAD)	caamalg
caamhash.c	JRI frontend (hashing)	caamhash
caampkc.c, pkc_desc.c	JRI frontend (public key cryptography)	caam_pkc
caamalg_qi.c	QI frontend (symmetric encryption, AEAD)	caamalg_qi
caamalg_qi2.[c,h]	DPSECI frontend (symmetric encryption, AEAD)	caamalg_qi2

Module loading

CAAM device drivers can be compiled either built-in or as modules (with the exception of DPSECI backend, which is always built-in). See section Source Files on page 65 for the list of module names and section Kernel Configuration on page 65 for how kernel configuration looks like and a mapping between menu entries and modules and / or functionalities enabled.

Kernel Configuration

CAAM device drivers are located in the "Cryptographic API" -> "Hardware crypto devices" sub-menu in the kernel configuration. Depending on the target platform and / or configuration file(s) used, the output will be different; below is an example taken from NXP Layerscape SDK for ARMv8 platforms with default options:

Kernel Configure Tree View Options	Description

```
Cryptographic API  --->
     [*]   Hardware crypto devices  --->
     <*>   Freescale CAAM-Multicore platform
driver backend (SEC)
     [ ]     Enable debug output in CAAM
```
	Enable CAAM device drivers, options:
	• basic platform driver: *Freescale CAAM-Multicore platform driver backend (SEC)*; all non-DPAA2 sub-options depend on it

Table continues on the next page...

Table continued from the previous page...

Kernel Configure Tree View Options	Description

driver
     <*>   Freescale CAAM Job Ring driver
backend (SEC)
     (9)   Job Ring size
     [ ]   Job Ring interrupt coalescing
     <*>   Register algorithm
implementations with the Crypto API
     <*>   Queue Interface as Crypto API
backend
     <*>   Register hash algorithm
implementations with Crypto API
     <*>   Register public key
cryptography implementations with Crypto API
     <*>   Register caam device for
hwrng API
     <M>   QorIQ DPAA2 CAAM (DPSECI) driver
	• backends / interfaces:
	— *Freescale CAAM Job Ring driver backend (SEC)* - JRI; this also enables QI (QI depends on JRI)
	— *QorIQ DPAA2 CAAM (DPSECI) driver* - DPSECI
	• frontends / crypto algorithms:
	— symmetric encryption, AEAD, "stitched" AEAD, TLS; *Register algorithm implementations with the Crypto API* - via JRI (*caamalg* driver) or *Queue Interface as Crypto API backend* - via QI (*caamalg_qi* drive)
	— *Register hash algorithm implementations with Crypto API* - hashing (only via JRI - *caamhash* driver)
	— *Register public key cryptography implementations with Crypto API* - asymmetric / public key (only via JRI - *caam_pkc* driver)
	— *Register caam device for hwrng API* - HW RNG (only via JRI - *caamrng* driver)
	— *QorIQ DPAA2 CAAM (DPSECI) driver* - DPSECI
	• options: debugging, JRI ring size, JRI interrupt coalescing

Networking support --->
  Network option --->
    <*> TCP/IP networking
    <*>   IP: AH transformation
    <*>   IP: ESP transformation
    <*>   IP: IPsec transport mode
    <*>   IP: IPsec tunnel mode
	For IPsec support the TCP/IP networking option and corresponding sub-options should be enabled.

Device Tree binding

Property	Type	Status	Description
compatible	String	Required	fsl,sec-vX.Y (preferred) OR fsl,secX.Y

Sample Device Tree crypto node

```
    crypto@30000 {
            compatible = "fsl,sec-v4.0";
            fsl,sec-era = <2>;
            #address-cells = <1>;
            #size-cells = <1>;
            reg = <0x300000 0x10000>;
            ranges = <0 0x300000 0x10000>;
            interrupt-parent = <&mpic>;
            interrupts = <92 2>;
            clocks = <&clks IMX6QDL_CLK_CAAM_MEM>,
                     <&clks IMX6QDL_CLK_CAAM_ACLK>,
                     <&clks IMX6QDL_CLK_CAAM_IPG>,
                     <&clks IMX6QDL_CLK_EIM_SLOW>;
            clock-names = "mem", "aclk", "ipg", "emi_slow";
    };
```

NOTE

See linux/Documentation/devicetree/bindings/crypto/fsl-sec4.txt file in the Linux kernel tree for more info.

How to test the drivers

To test the drivers, under the `"Cryptographic API -> Cryptographic algorithm manager"` kernel configuration sub-menu, ensure that run-time self tests are not disabled, i.e. the "Disable run-time self tests" entry is not set (*CONFIG_CRYPTO_MANAGER_DISABLE_TESTS=n*). This will run standard test vectors against the drivers after they register supported algorithms with the kernel crypto API, usually at boot time. Then run test on the target system. Below is a snippet extracted from the boot log of ARMv8-based LS1046A platform, with JRI and QI enabled:

```
[...]
platform caam_qi: Linux CAAM Queue I/F driver initialised
caam 1700000.crypto: Instantiated RNG4 SH1
caam 1700000.crypto: device ID = 0x0a11030100000000 (Era 8)
caam 1700000.crypto: job rings = 4, qi = 1, dpaa2 = no
alg: No test for authenc(hmac(sha224),ecb(cipher_null)) (authenc-hmac-sha224-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha256),ecb(cipher_null)) (authenc-hmac-sha256-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha384),ecb(cipher_null)) (authenc-hmac-sha384-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha512),ecb(cipher_null)) (authenc-hmac-sha512-ecb-cipher_null-caam)
alg: No test for authenc(hmac(md5),cbc(aes)) (authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(aes))) (echainiv-authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(aes))) (echainiv-authenc-hmac-sha1-cbc-aes-caam)
alg: No test for authenc(hmac(sha224),cbc(aes)) (authenc-hmac-sha224-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha224),cbc(aes))) (echainiv-authenc-hmac-sha224-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha256),cbc(aes))) (echainiv-authenc-hmac-sha256-cbc-aes-caam)
alg: No test for authenc(hmac(sha384),cbc(aes)) (authenc-hmac-sha384-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha384),cbc(aes))) (echainiv-authenc-hmac-sha384-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha512),cbc(aes))) (echainiv-authenc-hmac-sha512-cbc-aes-caam)
alg: No test for authenc(hmac(md5),cbc(des3_ede)) (authenc-hmac-md5-cbc-des3_ede-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(des3_ede))) (echainiv-authenc-hmac-md5-cbc-des3_ede-
caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des3_ede))) (echainiv-authenc-hmac-sha1-cbc-des3_ede-
caam)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des3_ede))) (echainiv-authenc-hmac-sha224-cbc-
des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des3_ede))) (echainiv-authenc-hmac-sha256-cbc-
des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des3_ede))) (echainiv-authenc-hmac-sha384-cbc-
des3_ede-caam)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des3_ede))) (echainiv-authenc-hmac-sha512-cbc-
des3_ede-caam)
alg: No test for authenc(hmac(md5),cbc(des)) (authenc-hmac-md5-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(des))) (echainiv-authenc-hmac-md5-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des))) (echainiv-authenc-hmac-sha1-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des))) (echainiv-authenc-hmac-sha224-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des))) (echainiv-authenc-hmac-sha256-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des))) (echainiv-authenc-hmac-sha384-cbc-des-caam)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des))) (echainiv-authenc-hmac-sha512-cbc-des-caam)
alg: No test for authenc(hmac(md5),rfc3686(ctr(aes))) (authenc-hmac-md5-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(md5),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-md5-rfc3686-ctr-aes-
caam)
```

```
alg: No test for authenc(hmac(sha1),rfc3686(ctr(aes))) (authenc-hmac-sha1-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha1),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha1-rfc3686-ctr-
aes-caam)
alg: No test for authenc(hmac(sha224),rfc3686(ctr(aes))) (authenc-hmac-sha224-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha224),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha224-rfc3686-
ctr-aes-caam)
alg: No test for authenc(hmac(sha256),rfc3686(ctr(aes))) (authenc-hmac-sha256-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha256),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha256-rfc3686-
ctr-aes-caam)
alg: No test for authenc(hmac(sha384),rfc3686(ctr(aes))) (authenc-hmac-sha384-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha384),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha384-rfc3686-
ctr-aes-caam)
alg: No test for authenc(hmac(sha512),rfc3686(ctr(aes))) (authenc-hmac-sha512-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha512),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha512-rfc3686-
ctr-aes-caam)
caam algorithms registered in /proc/crypto
alg: No test for authenc(hmac(md5),cbc(aes)) (authenc-hmac-md5-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(md5),cbc(aes))) (echainiv-authenc-hmac-md5-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha1),cbc(aes))) (echainiv-authenc-hmac-sha1-cbc-aes-caam-qi)
alg: No test for authenc(hmac(sha224),cbc(aes)) (authenc-hmac-sha224-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha224),cbc(aes))) (echainiv-authenc-hmac-sha224-cbc-aes-caam-
qi)
alg: No test for echainiv(authenc(hmac(sha256),cbc(aes))) (echainiv-authenc-hmac-sha256-cbc-aes-caam-
qi)
alg: No test for authenc(hmac(sha384),cbc(aes)) (authenc-hmac-sha384-cbc-aes-caam-qi)
alg: No test for echainiv(authenc(hmac(sha384),cbc(aes))) (echainiv-authenc-hmac-sha384-cbc-aes-caam-
qi)
alg: No test for echainiv(authenc(hmac(sha512),cbc(aes))) (echainiv-authenc-hmac-sha512-cbc-aes-caam-
qi)
alg: No test for authenc(hmac(md5),cbc(des3_ede)) (authenc-hmac-md5-cbc-des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(md5),cbc(des3_ede))) (echainiv-authenc-hmac-md5-cbc-des3_ede-
caam-qi)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des3_ede))) (echainiv-authenc-hmac-sha1-cbc-des3_ede-
caam-qi)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des3_ede))) (echainiv-authenc-hmac-sha224-cbc-
des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des3_ede))) (echainiv-authenc-hmac-sha256-cbc-
des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des3_ede))) (echainiv-authenc-hmac-sha384-cbc-
des3_ede-caam-qi)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des3_ede))) (echainiv-authenc-hmac-sha512-cbc-
des3_ede-caam-qi)
alg: No test for authenc(hmac(md5),cbc(des)) (authenc-hmac-md5-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(md5),cbc(des))) (echainiv-authenc-hmac-md5-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha1),cbc(des))) (echainiv-authenc-hmac-sha1-cbc-des-caam-qi)
alg: No test for echainiv(authenc(hmac(sha224),cbc(des))) (echainiv-authenc-hmac-sha224-cbc-des-caam-
qi)
alg: No test for echainiv(authenc(hmac(sha256),cbc(des))) (echainiv-authenc-hmac-sha256-cbc-desi-caam-
qi)
alg: No test for echainiv(authenc(hmac(sha384),cbc(des))) (echainiv-authenc-hmac-sha384-cbc-des-caam-
qi)
alg: No test for echainiv(authenc(hmac(sha512),cbc(des))) (echainiv-authenc-hmac-sha512-cbc-des-caam-
qi)
platform caam_qi: algorithms registered in /proc/crypto
caam_jr 1710000.jr: registering rng-caam
caam 1700000.crypto: caam pkc algorithms registered in /proc/crypto
[...]
```

Crypto algorithms support

Algorithms Supported in the linux kernel scatterlist Crypto API

The Linux kernel contains various users of the Scatterlist Crypto API, including its IPsec implementation, sometimes referred to as the NETKEY stack. The driver, after registering supported algorithms with the Crypto API, is therefore used to process per-packet symmetric crypto requests and forward them to the SEC hardware.

Since SEC hardware processes requests asynchronously, the driver registers asynchronous algorithm implementations with the crypto API: ahash, ablkcipher, and aead with CRYPTO_ALG_ASYNC set in .cra_flags.

Different combinations of hardware and driver software version support different sets of algorithms, so searching for the driver name in /proc/crypto on the desired target system will ensure the correct report of what algorithms are supported.

Authenticated Encryption with Associated Data (AEAD) algorithms

These algorithms are used in applications where the data to be encrypted overlaps, or partially overlaps, the data to be authenticated, as is the case with IPsec and TLS protocols. These algorithms are implemented in the driver such that the hardware makes a single pass over the input data, and both encryption and authentication data are written out simultaneously. The AEAD algorithms are mainly for use with IPsec ESP (however there is also support for TLS 1.0 record layer encryption).

CAAM drivers currently supports offloading the following AEAD algorithms:

- "stitched" AEAD: all combinations of { NULL, CBC-AES, CBC-DES, CBC-3DES-EDE, RFC3686-CTR-AES } x HMAC-{MD-5, SHA-1,-224,-256,-384,-512}
- "true" AEAD: generic GCM-AES, GCM-AES used in IPsec: RFC4543-GCM-AES and RFC4106-GCM-AES
- TLS 1.0 record layer encryption using the "stitched" AEAD cipher suite CBC-AES-HMAC-SHA1

Encryption algorithms

The CAAM driver currently supports offloading the following encryption algorithms.

Authentication algorithms

The CAAM driver's ahash support includes keyed (hmac) and unkeyed hashing algorithms.

Asymmetric (public key) algorithms

Currently, RSA is the only public key algorithm supported.

Random Number Generation

caamrng frontend driver supports random number generation services via the kernel's built-in hwrng interface when implemented in hardware. To enable:

1. verify that the hardware random device file, e.g., /dev/hwrng or /dev/hwrandom exists. If it doesn't exist, make it with:

   ```
   $ mknod /dev/hwrng c 10 183
   ```

2. verify /dev/hwrng doesn't block indefinitely and produces random data:

   ```
   $ rngtest -C 1000 < /dev/hwrng
   ```

3. verify the kernel gets entropy:

   ```
   $ rngtest -C 1000 < /dev/random
   ```

If it blocks, a kernel entropy supplier daemon, such as rngd, may need to be run. See linux/Documentation/hw_random.txt for more info.

Table 10. Algorithms supported by each interface / backend

Algorithm name / Backend	Job Ring Interface	Queue Interface	DPSEC Interface
rsa	Yes	No	No
tls10(hmac(sha1),cbc(aes))	No	Yes	Yes
authenc(hmac(md5),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha1),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha224),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha256),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha384),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha512),cbc(aes))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(md5),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha1),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha224),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha256),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha384),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha512),cbc(des3_ede))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(md5),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha1),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha224),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha256),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha384),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)
authenc(hmac(sha512),cbc(des))	Yes (also echainiv)	Yes (also echainiv)	Yes (also echainiv)

Table continues on the next page...

Table 10. Algorithms supported by each interface / backend (continued)

Algorithm name / Backend	Job Ring Interface	Queue Interface	DPSEC Interface
authenc(hmac(md5),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha1),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha224),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha256),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha384),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(sha512),rfc3686(ctr(aes)))	Yes (also seqiv)	Yes (also seqiv)	Yes (also seqiv)
authenc(hmac(md5),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha1),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha224),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha256),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha384),ecb(cipher_null))	Yes	No	No
authenc(hmac(sha512),ecb(cipher_null))	Yes	No	No
gcm(aes)	Yes	Yes	Yes
rfc4543(gcm(aes))	Yes	Yes	Yes
rfc4106(gcm(aes))	Yes	Yes	Yes
cbc(aes)	Yes	Yes	Yes
cbc(des3_ede)	Yes	Yes	Yes
cbc(des)	Yes	Yes	Yes
ctr(aes)	Yes	Yes	Yes
rfc3686(ctr(aes))	Yes	Yes	Yes
xts(aes)	Yes	Yes	Yes
hmac(md5)	Yes	No	Yes
hmac(sha1)	Yes	No	Yes
hmac(sha224)	Yes	No	Yes
hmac(sha256)	Yes	No	Yes

Table continues on the next page...

Table 10. Algorithms supported by each interface / backend (continued)

Algorithm name / Backend	Job Ring Interface	Queue Interface	DPSEC Interface
hmac(sha384)	Yes	No	Yes
hmac(sha512)	Yes	No	Yes
md5	Yes	No	Yes
sha1	Yes	No	Yes
sha224	Yes	No	Yes
sha256	Yes	No	Yes
sha384	Yes	No	Yes
sha512	Yes	No	Yes

CAAM Job Ring backend driver specifics

CAAM Job Ring backend driver (*caam_jr*) implements and utilizes the job ring interface (JRI) for submitting crypto API service requests from the frontend drivers (*caamalg*, *caamhash*, *caam_pkc*, *caamrng*) to CAAM engine.

CAAM drivers have a few options, most notably hardware job ring size and interrupt coalescing. They can be used to fine-tune performance for a particular use case.

The option *Freescale CAAM-Multicore platform driver backend* enables the basic platform driver (*caam*). All (non-DPAA2) sub-options depend on this.

The option *Freescale CAAM Job Ring driver backend (SEC)* enables the Job Ring backend (*caam_jr*).

The sub-option *Job Ring Size* allows the user to select the size of the hardware job rings; if requests arrive at the driver enqueue entry point in a bursty nature, the bursts' maximum length can be approximated etc. One can set the greatest burst length to save performance and memory consumption.

The sub-option *Job Ring interrupt coalescing* allows the user to select the use of the hardware's interrupt coalescing feature. Note that the driver already performs IRQ coalescing in software, and zero-loss benchmarks have in fact produced better results with this option turned off. If selected, two additional options become effective:

- *Job Ring interrupt coalescing count threshold* (CRYPTO_DEV_FSL_CAAM_INTC_THLD)

  Selects the value of the descriptor completion threshold, in the range 1-256. A selection of 1 effectively defeats the coalescing feature, and any selection equal or greater than the selected ring size will force timeouts for each interrupt.

- *Job Ring interrupt coalescing timer threshold* (CRYPTO_DEV_FSL_CAAM_INTC_TIME_THLD)

  Selects the value of the completion timeout threshold in multiples of 64 SEC interface clocks, to which, if no new descriptor completions occur within this window (and at least one completed job is pending), then an interrupt will occur. This is selectable in the range 1-65535.

The options to register to Crypto API, hwrng API respectively, allow the frontend drivers to register their algorithm capabilities with the corresponding APIs. They should be deselected only when the purpose is to perform Crypto API requests in software (on the GPPs) instead of offloading them on SEC engine.

caamhash frontend (hash algorithms) may be individually turned off, since the nature of the application may be such that it prefers software (core) crypto latency due to many small-sized requests.

caam_pkc frontend (public key / asymmetric algorithms) can be turned off too, if needed.

caamrng frontend (Random Number Generation) may be turned off in case there is an alternate source of entropy available to the kernel.

Verifying driver operation and correctness

Other than noting the performance advantages due to the crypto offload, one can also ensure the hardware is doing the crypto by looking for driver messages in dmesg.

The driver emits console messages at initialization time:

```
caam algorithms registered in /proc/crypto
caam_jr 1710000.jr: registering rng-caam
caam 1700000.crypto: caam pkc algorithms registered in /proc/crypto
```

If the messages are not present in the logs, either the driver is not configured in the kernel, or no SEC compatible device tree node is present in the device tree.

Incrementing IRQs in /proc/interrupts

Given a time period when crypto requests are being made, the SEC hardware will fire completion notification interrupts on the corresponding Job Ring:

```
$ cat /proc/interrupts | grep jr
           CPU0       CPU1       CPU2       CPU3
[...]
 78:       1007          0          0          0     GICv2 103 Level     1710000.jr
 79:          7          0          0          0     GICv2 104 Level     1720000.jr
 80:          0          0          0          0     GICv2 105 Level     1730000.jr
 81:          0          0          0          0     GICv2 106 Level     1740000.jr
```

If the number of interrupts fired increment, then the hardware is being used to do the crypto.

If the numbers do not increment, then first check the algorithm being exercised is supported by the driver. If the algorithm is supported, there is a possibility that the driver is in polling mode (NAPI mechanism) and the hardware statistics in debugfs (inbound / outbound bytes encrypted / protected - see below) should be monitored.

Verifying the 'self test' fields say 'passed' in /proc/crypto

An entry such as the one below means the driver has successfully registered support for the algorithm with the kernel crypto API:

```
name         : cbc(aes)
driver       : cbc-aes-caam
module       : kernel
priority     : 3000
refcnt       : 1
selftest     : passed
internal     : no
type         : givcipher
async        : yes
blocksize    : 16
min keysize  : 16
max keysize  : 32
ivsize       : 16
geniv        : <built-in>
```

Note that although a test vector may not exist for a particular algorithm supported by the driver, the kernel will emit messages saying which algorithms weren't tested, and mark them as 'passed' anyway:

```
[...]
alg: No test for authenc(hmac(sha224),ecb(cipher_null)) (authenc-hmac-sha224-ecb-cipher_null-caam)
alg: No test for authenc(hmac(sha256),ecb(cipher_null)) (authenc-hmac-sha256-ecb-cipher_null-caam)
[...]
alg: No test for authenc(hmac(md5),cbc(aes)) (authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(md5),cbc(aes))) (echainiv-authenc-hmac-md5-cbc-aes-caam)
alg: No test for echainiv(authenc(hmac(sha1),cbc(aes))) (echainiv-authenc-hmac-sha1-cbc-aes-caam)
[...]
```

```
alg: No test for authenc(hmac(sha512),rfc3686(ctr(aes))) (authenc-hmac-sha512-rfc3686-ctr-aes-caam)
alg: No test for seqiv(authenc(hmac(sha512),rfc3686(ctr(aes)))) (seqiv-authenc-hmac-sha512-rfc3686-ctr-
aes-caam)
[...]
```

Examining the hardware statistics registers in debugfs

When using the JRI or QI backend, performance monitor registers can be checked, provided CONFIG_DEBUG_FS is enabled in the kernel's configuration. If debugfs is not automatically mounted at boot time, then a manual mount must be performed in order to view these registers. This normally can be done with a superuser shell command:

```
$ mount -t debugfs none /sys/kernel/debug
```

Once done, the user can read controller registers in /sys/kernel/debug/1700000.crypto/ctl. It should be noted that debugfs will provide a decimal integer view of most accessible registers provided, with the exception of the KEK/TDSK/TKEK registers; those registers are long binary arrays, and should be filtered through a binary dump utility such as hexdump.

Specifically, the CAAM hardware statistics registers available are:

fault_addr, or FAR (Fault Address Register): - holds the value of the physical address where a read or write error occurred.

fault_detail, or FADR (Fault Address Detail Register): - holds details regarding the bus transaction where the error occurred.

fault_status, or CSTA (CAAM Status Register): - holds status information relevant to the entire CAAM block.

ib_bytes_decrypted: - holds contents of PC_IB_DECRYPT (Performance Counter Inbound Bytes Decrypted Register)

ib_bytes_validated: - holds contents of PC_IB_VALIDATED (Performance Counter Inbound Bytes Validated Register)

ib_rq_decrypted: - holds contents of PC_IB_DEC_REQ (Performance Counter Inbound Decrypt Requests Register)

kek: - holds contents of JDKEKR (Job Descriptor Key Encryption Key Register)

ob_bytes_encrypted: - holds contents of PC_OB_ENCRYPT (Performance Counter Outbound Bytes Encrypted Register)

ob_bytes_protected: - holds contents of PC_OB_PROTECT (Performance Counter Outbound Bytes Protected Register)

ob_rq_encrypted: - holds contents of PC_OB_ENC_REQ (Performance Counter Outbound Encrypt Requests Register)

rq_dequeued: - holds contents of PC_REQ_DEQ (Performance Counter Requests Dequeued Register)

tdsk: - holds contents of TDKEKR (Trusted Descriptor Key Encryption Key Register)

tkek: - holds contents of TDSKR (Trusted Descriptor Signing Key Register)

For more information see section "Performance Counter, Fault and Version ID Registers" in the Security (SEC) Reference Manual (SECRM) of each SoC (available on company's website).

Note: for QI backend there is also *qi_congested*: SW-based counter that shows how many times queues going to / from CAAM to QMan hit the congestion threshold.

Kernel configuration to support CAAM device drivers

Using the driver

Once enabled, the driver will forward kernel crypto API requests to the SEC hardware for processing.

Running IPsec

The IPsec stack built-in to the kernel (usually called NETKEY) will automatically use crypto drivers to offload crypto operations to the SEC hardware. Documentation regarding how to set up an IPsec tunnel can be found in corresponding open source IPsec suite packages, e.g. strongswan.org, openswan, setkey, etc. DPAA2-specific section contains a generic helper script to configure IPsec tunnels.

Running OpenSSL

Please see Hardware Offloading with OpenSSL for more details on how to offload OpenSSL cryptographic operations in the SEC crypto engine (via cryptodev).

Executing custom descriptors

SEC drivers have public descriptor submission interfaces corresponding to the following backends:

- JRI: drivers/crypto/caam/jr.c:caam_jr_enqueue()
- QI: drivers/crypto/caam/qi.c:caam_qi_enqueue()
- DPSECI: drivers/crypto/caam/caamalg_qi2.c:dpaa2_caam_enqueue()

caam_jr_enqueue()

Name

caam_jr_enqueue — Enqueue a job descriptor head. Returns 0 if OK, -EBUSY if the ring is full, -EIO if it cannot map the caller's descriptor.

Synopsis

```
int caam_jr_enqueue (struct device *dev, u32 *desc,
 void (*cbk) (struct device *dev, u32 *desc, u32 status, void *areq),
 void *areq);
```

Arguments

dev: contains the job ring device that is to process this request.

desc: descriptor that initiated the request, same as “desc” being argued to caam_jr_enqueue.

cbk: pointer to a callback function to be invoked upon completion of this request. This has the form: callback(struct device *dev, u32 *desc, u32 stat, void *arg)

areq: optional pointer to a user argument for use at callback time.

caam_qi_enqueue()

Name

caam_qi_enqueue — Enqueue a frame descriptor (FD) into a QMan frame queue. Returns 0 if OK, -EIO if it cannot map the caller's S/G array, -EBUSY if QMan driver fails to enqueue the FD for some reason.

Synopsis

```
int caam_qi_enqueue(struct device *qidev, struct caam_drv_req *req);
```

Arguments

qidev: contains the queue interface device that is to process this request.

req: pointer to the request structure the driver application should fill while submitting a job to driver, containing a callback function and its parameter, Queue Manager S/Gs for input and output, a per-context structure containing the CAAM shared descriptor etc.

dpaa2_caam_enqueue()

Name

dpaa2_caam_enqueue — Enqueue a frame descriptor (FD) into a QMan frame queue. Returns 0 if OK, -EBUSY if QMan driver fails to enqueue the FD for some reason or if congestion is detected.

Synopsis

```
int dpaa2_caam_enqueue(struct device *dev, struct caam_request *req);
```

Arguments

dev: DPSECI device.

req: pointer to the request structure the driver application should fill while submitting a job to driver, containing a callback function and its parameter, Queue Manager S/Gs for input and output, a per-context structure containing the CAAM shared descriptor etc.

Please refer to the source code for usage examples.

Supporting Documentation

DPAA1-specific SEC details - Queue Interface (QI)

DPAA2-specific SEC details - Data Path SEC Interface (DPSECI)

4.2.10 Universal Serial Bus Interfaces

4.2.10.1 USB 3.0 Host/Peripheral Linux Driver User Manual

Description

The driver supports xHCI SuperSpeed (SS) Dual-Role-Device (DRD) controller

Main features of xHCI controller

- Supports operation as a standalone USB xHCI host controller
- USB dual-role operation and can be configured as host or device
- Super-speed (5 GT/s), High-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations
- Supports operation as a standalone single port USB
- Supports eight programmable, bidirectional USB endpoints

Modes of Operation

- Host Mode: SS/HS/FS/LS
- Device Mode: SS/HS/FS

NOTE

Super-speed operation is not supported when OTG is enabled

NOTE

This document explains working of **HS Host and HS Device** in Linux

Module Loading

The default kernel configuration enables support for USB_DWC3 as built-in kernel module.

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description

`Device Drivers--->`
` USB support --->`
	Enables USB host controller support

Table continues on the next page...

Table continued from the previous page...

Kernel Configure Tree View Options	Description
`[*] Support for Host-side USB`	

`Device Drivers--->`
 	
` USB support --->`
` <*>     xHCI HCD (USB 3.0) support`
	Enables XHCI Host Controller Driver and transaction translator

`Device Drivers--->`
 	
` USB support --->`
` <*>     USB Mass Storage support`
 	
`  [ ]       USB Mass Storage verbose debug`
	Enable support for USB mass storage devices. This is the driver needed for USB flash devices, and memory sticks

`<*> Sound card support  --->`
`<*>   Advanced Linux Sound Architecture   --->`
`   <*>   OSS Mixer API`
`   <*>   OSS PCM (digital audio) API`
`   [*]     OSS PCM (digital audio) API -`
`Include plugin system`
`   [*]   Support old ALSA API`
`   [*]   USB sound devices   --->`
`     <*>   USB Audio/MIDI driver`
	Enables support for USB Audio devices. This driver is needed for USB microphone.

`Device Drivers--->`
 	
` USB support --->`
`  <*>   USB Gadget Support --->`
 	
`    <M>   USB Gadget Drivers`
`    < >     USB functions configurable through`
`configfs`
`    < >     Gadget Zero (DEVELOPMENT)`
`    <M>     Ethernet Gadget (with CDC Ethernet`
`support)`
`    [*]       RNDIS support`
`    [ ]       Ethernet Emulation Model (EEM)`
`support`
`    < >     Network Control Model (NCM) support`
`    < >     Gadget Filesystem`
`    < >     Function Filesystem`
`    <M>     Mass Storage Gadget`
`    < >     Serial Gadget (with CDC ACM and CDC`
`OBEX support)`
	Note: Required only for USB Gadget/Peripheral Support
	• Enable driver for peripheral/device controller
	• Enable Ethernet Gadget Client driver
	• Enable Mass Storage Client driver

Table continues on the next page...

Table continued from the previous page...

Kernel Configure Tree View Options	Description

```
Device Drivers--->

 <*>   DesignWare USB3 DRD Core Support
          DWC3 Mode Selection (Dual Role mode)
--->
```
	Enable XHCI DRD Core Support

Compile-time Configuration Options

Option	Values	Default Value	Description	
CONFIG_USB	y/m/n	y	Enables USB host controller	
CONFIG_USB_XHCI_HCD	y/m/n	y	Enables XHCI HCD	
CONFIG_USB_DWC3	y/m/n	y	Enables DWC3 Controller	
CONFIG_USB_GADGET	y/m/n	n	Enables USB peripheral device	
CONFIG_USB_ETH	y/m/n	n	Enable Ethernet style communication	
CONFIG_USB_MASS_STORAGE	m/n	n	Enable USB Mass Storage disk drive	
CONFIG_SOUND	y/m/n	y	Enables Sound Card Support	
CONFIG_SND	y/m/n	y	Enables ALSA (Advanced Linux Sound Architecture)	
CONFIG_SND_MIXER_OSS	y/m/n	y	Enables OSS Mixer API	
CONFIG_SND_PCM_OSS	y/m/n	y	Enables OSS PCM (digital audio) API	
CONFIG_SND_PCM_OSS_PLUGINS	y/n	y	Enables OSS PCM (digital audio) API - Include plugin system	
CONFIG_SND_SUPPORT_OLD_API	y/n	y	Enables old ALSA API	
CONFIG_SND_USB	y/n	n	Enables USB sound devices	
CONFIG_SND_USB_AUDIO	y/m/n	n	Enables USB Audio/ MIDI driver	

NOTE: USB Audio configuration options default value is listed for LS1021A platform.

Source Files

The driver source is maintained in the Linux kernel source tree in below files

Table continued from the previous page...

Source File	Description
drivers/usb/host/xhci-*	xhci platform driver
drivers/usb/gadget/mass_storage.c	USB Mass Storage
drivers/usb/gadget/ether.c	Ethernet gadget driver

Device Tree Binding for Host

```
usb@3100000 {
               compatible = "snps,dwc3";
               reg = <0x0 0x3100000 0x0 0x10000>;
               interrupts = <GIC_SPI 93 IRQ_TYPE_LEVEL_HIGH>;
              dr_mode = "host;
            };
```

Device Tree Binding for Peripheral

Note: with multiple USB controller, just one can be peripheral mode at a time.

```
usb@3100000 {
               compatible = "snps,dwc3";
               reg = <0x0 0x3100000 0x0 0x10000>;
               interrupts = <GIC_SPI 93 IRQ_TYPE_LEVEL_HIGH>;
              dr_mode = "peripheral;
               maximum-speed = "super-speed";
          };
```

Host Testing

Following are serial console logs that appear during bootup if dr_mode set to host in device-tree

```
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned bus number 1
xhci-hcd xhci-hcd.0.auto: irq 125, io mem 0x03100000
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
xhci-hcd xhci-hcd.0.auto: xHCI Host Controller
xhci-hcd xhci-hcd.0.auto: new USB bus registered, assigned bus number 2
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 1 port detected
usbcore: registered new interface driver usb-storage
```

Following are serial-console logs after connecting a USB flash drive

```
For High-Speed Device attach
usb 1-1.2: new high-speed USB device number 3 using xhci-hcd
usb-storage 1-1.2:1.0: USB Mass Storage device detected
scsi0 : usb-storage 1-1.2:1.0
scsi 0:0:0:0: Direct-Access     SanDisk  Cruzer           7.01 PQ: 0 ANSI: 0 CCS
sd 0:0:0:0: [sda] 1957887 512-byte logical blocks: (1.00 GB/955 MiB)
sd 0:0:0:0: Attached scsi generic sg0 type 0
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] No Caching mode page found
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] Attached SCSI removable disk

For Super-Speed Device attach
# usb 2-1: new SuperSpeed USB device number 2 using xhci-hcd
usb 2-1: Parent hub missing LPM exit latency info.  Power management will be impacted.
usb-storage 2-1:1.0: USB Mass Storage device detected
scsi0 : usb-storage 2-1:1.0
scsi 0:0:0:0: Direct-Access     SanDisk  Extreme          0001 PQ: 0 ANSI: 6
sd 0:0:0:0: [sda] 31277232 512-byte logical blocks: (16.0 GB/14.9 GiB)
sd 0:0:0:0: Attached scsi generic sg0 type 0
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO or FUA
 sda:
sd 0:0:0:0: [sda] Attached SCSI removable disk
FAT-fs (sda): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
```

Make filesystem and mount connected USB flash drive using below commands

```
root@freescale /$ fdisk -l

Disk /dev/sda: 16.0 GB, 16013942784 bytes
255 heads, 63 sectors/track, 1946 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks  Id System
/dev/sda1               1        1946    15631213+ 83 Linux
root@freescale /$
root@freescale /$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
shm                     516684         0    516684   0% /dev/shm
rwfs                       512         0       512   0% /mnt/rwfs
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$ fdisk /dev/sda

The number of cylinders for this disk is set to 1946.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
```

```
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): d
Selected partition 1

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-1946, default 1): Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-1946, default 1946): Using default value 1946

Command (m for help): w
The partition table has been alter sda: sda1
ed!

Calling ioctl() to re-read partition table
root@freescale /$
root@freescale /$
root@freescale /$ fdisk -l

Disk /dev/sda: 16.0 GB, 16013942784 bytes
255 heads, 63 sectors/track, 1946 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks  Id System
/dev/sda1               1        1946    15631213+ 83 Linux
root@freescale /$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
shm                     516684         0    516684   0% /dev/shm
rwfs                       512         0       512   0% /mnt/rwfs
root@freescale /$ mkdir my_mnt
root@freescale /$
root@freescale /$
root@freescale /$ mkfs.ext2 /dev/sda1
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
977280 inodes, 3907803 blocks
195390 blocks (5%) reserved for the super user
First data block=0
Maximum filesystem blocks=4194304
120 block groups
32768 blocks per group, 32768 fragments per group
8144 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$ mount /dev/sda1 my_mnt/
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$ df
```

```
Filesystem           1K-blocks      Used Available Use% Mounted on
shm                     516684         0    516684   0% /dev/shm
rwfs                       512         0       512   0% /mnt/rwfs
/dev/sda1             15385852        20  14604272   0% /my_mnt
root@freescale /$
```

Test by wring/reading data on mount drive

```
root@freescale /$ dd if=/dev/urandom of=/tmp/123 bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes (100.0MB) copied, 54.535026 seconds, 1.8MB/s
root@freescale /$
root@freescale /$
root@freescale /$
root@freescale /$ cp /tmp/123 /my_mnt/.
root@freescale /$ sync
root@freescale /$ ls /my_mnt/
123         lost+found
root@freescale /$
```

Peripheral testing with Win7 as Host

NOTE

In gadget mode standard USB cables with micro plug should be used.

Below Message will appear during bootup if dr_mode set as peripheral in device-tree

```
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb

usbcore: registered new interface driver usb-storage
```

Make sure "dr_mode" contains "peripheral" string

```
root@freescale /$# cat /proc/device-tree/soc/usb\@3100000/dwc3/dr_mode
peripheral root@freescale /$
```

Move all below modeules to platform

```
fs/configfs/configfs.ko
driver/usb/gadget/libcomposite.ko
driver/usb/gadget/g_mass_storage.ko
driver/usb/gadget/u_rndis.ko
driver/usb/gadget/u_ether.ko
driver/usb/gadget/usb_f_ecm.ko
driver/usb/gadget/usb_f_ecm_subset.ko
driver/usb/gadget/usb_f_rndis.ko
driver/usb/gadget/g_ether.ko
```

Mass Storage Gadget

```
To use ramdisk as a backing store use the following

root@freescale /$ mkdir /mnt/ramdrive
root@freescale /$ mount -t tmpfs tmpfs /mnt/ramdrive -o size=600M
```

```
root@freescale /$ dd if=/dev/zero of=/mnt/ramdrive/vfat-file bs=1M count=500
root@freescale /$ mke2fs -F /mnt/ramdrive/vfat-file
root@freescale /$ insmod configfs.ko
root@freescale /$ insmod libcomposite.ko
root@freescale /$ insmod usb_f_mass_storage.ko
root@freescale /$ insmod g_mass_storage.ko file=/mnt/ramdrive/vfat-file stall=n
```

We will get below messages

```
[   39.987594] g_mass_storage gadget: Mass Storage Function, version: 2009/09/11
[   39.994822] g_mass_storage gadget: Number of LUNs=1
[   39.989240]  lun0: LUN: file: /home/backing_file_20mb
[   39.994367] g_mass_storage gadget: Mass Storage Gadget, version: 2009/09/11
[   39.990902] g_mass_storage gadget: userspace failed to provide iSerialNumber
[   39.987547] g_mass_storage gadget: g_mass_storage ready
```

Attached ***USB3.0 only*** gadget cable to host and you will get below message. Now Storage is ready to use.

```
g_mass_storage gadget: super-speed config #1: Linux File-Backed Storage
```

Speaker and Microphone

1. Aplay utility can be used to list the available sound cards e.g. Here Jabra 410 USB speaker is detected as a second sound card and can be addressed as **–D hw:1,0 OR –c1**:

```
[root@freescale ~]$ aplay –l**** List of PLAYBACK Hardware Devices ****
  card 0: FSLVF610TWRBOAR [FSL-VF610-TWR-BOARD], device 0: HiFi sgtl5000-0 [ ]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
  card 1: USB [Jabra SPEAK 410 USB], device 0: USB Audio [USB Audio]  Subdevices: 1/1  Subdevice #0:
subdevice #0
```

2. Sample wav file can be played using the below command:

```
[root@freescale ~]$ aplay -D hw:1,0 LYNC_fsringing.wav
Playing WAVE 'LYNC_fsringing.wav' : Signed 16 bit Little Endian, Rate 48000 Hz, Stereo
```

3. Sample wav file can be recorded using the below command:

```
[root@freescale ~]$ arecord -f S16_LE -t wav -Dhw:1,0 -r 16000 foobar.wav -d 5
Recording WAVE 'foobar.wav' : Signed 16 bit Little Endian, Rate 16000 Hz, Mono
```

NOTE: If recorded audio is not played, try to use "-D plughw:1,0" in above command.

4. Audio controls can be checked using the below command, control details and name of the controls can be checked from output of "amixer –c1" as below:

```
[root@freescale ~]$ amixer -c1 controls
numid=3,iface=MIXER,name='PCM Playback Switch'
numid=4,iface=MIXER,name='PCM Playback Volume'
numid=5,iface=MIXER,name='Headset Capture Switch'
numid=6,iface=MIXER,name='Headset Capture Volume'
numid=2,iface=PCM,name='Capture Channel Map'
numid=1,iface=PCM,name='Playback Channel Map'

[root@freescale ~]$ amixer -c1
Simple mixer control 'PCM',0  Capabilities: pvolume pvolume-joined pswitch pswitch-joined penum
```

```
  Playback channels: Mono
  Limits: Playback 0 - 11
  Mono: Playback 4 [36%] [-20.00dB] [on]

Simple mixer control 'Headset',0  Capabilities: cvolume cvolume-joined cswitch cswitch-joined penum
  Capture channels: Mono
  Limits: Capture 0 - 7
  Mono: Capture 5 [71%] [0.00dB] [on]
```

For Example, in above output there are two controls named “PCM” and “Headset” for Speaker and microphone respectively.

Sample Audio controls Usage:

a. mute/unmute

```
[root@freescale ~]$ amixer -c1 set PCM  mute
Simple mixer control 'PCM',0
  Capabilities: pvolume pvolume-joined pswitch pswitch-joined
  Playback channels: Mono
  Limits: Playback 0 - 11
  Mono: Playback 2 [18%] [-28.00dB] [off]
[root@freescale ~]$ amixer -c1 set PCM  unmute
Simple mixer control 'PCM',0
  Capabilities: pvolume pvolume-joined pswitch pswitch-joined
  Playback channels: Mono
  Limits: Playback 0 - 11
  Mono: Playback 2 [18%] [-28.00dB] [on]
```

b. volume up/down – Below commands are trying to set volume to 11 and 2 performing volume up and down respectively.

```
root@freescale ~]$ amixer -c1 set PCM 11
Simple mixer control 'PCM',0
Capabilities: pvolume pvolume-joined pswitch pswitch-joined
Playback channels: Mono
Limits: Playback 0 - 11
Mono: Playback 11 [100%] [8.00dB] [on]
[root@freescale ~]$ amixer -c1 set PCM 2
Simple mixer control 'PCM',0
Capabilities: pvolume pvolume-joined pswitch pswitch-joined
Playback channels: Mono
Limits: Playback 0 - 11
Mono: Playback 2 [18%] [-28.00dB] [on]
```

Ethernet Gadget

```
To use Ethernet gadget use the following

root@freescale /$ insmod configfs.ko
root@freescale /$ insmod libcomposite.ko
root@freescale /$ insmod u_ether.ko
root@freescale /$ insmod u_rndis.ko
root@freescale /$ insmod usb_f_ecm.ko
root@freescale /$ insmod usb_f_ecm_subset.ko
root@freescale /$ insmod usb_f_rndis.ko
root@freescale /$ insmod g_ether.ko
```

We will get below messages

```
[   28.692611] using random self ethernet address
[   28.697156] using random host ethernet address
[   28.694271] usb0: HOST MAC 82:96:69:7e:a5:7d
[   28.698928] usb0: MAC 72:00:a5:80:2b:e8
[   28.692586] using random self ethernet address
[   28.697080] using random host ethernet address
[   28.691368] g_ether gadget: Ethernet Gadget, version: Memorial Day 2008
[   28.698028] g_ether gadget: g_ether ready
```

Make sure USB0 ethernet interface is available after this

```
root@freescale /$ ifconfig -a
can0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:158

can1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          NOARP  MTU:16  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:159

eth0      Link encap:Ethernet  HWaddr 00:E0:0C:BC:E5:60
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 00:E0:0C:BC:E5:61
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth2      Link encap:Ethernet  HWaddr 00:E0:0C:BC:E5:62
          inet addr:10.232.132.212  Bcast:10.232.135.255  Mask:255.255.252.0
          inet6 addr: fe80::2e0:cff:febc:e562/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2311 errors:0 dropped:3 overruns:0 frame:0
          TX packets:66 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:290810 (283.9 KiB)  TX bytes:8976 (8.7 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:2 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:100 (100.0 B)  TX bytes:100 (100.0 B)

sit0      Link encap:IPv6-in-IPv4
          NOARP  MTU:1480  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usb0      Link encap:Ethernet  HWaddr 72:00:A5:80:2B:E8
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Attached the cable with Win7 and Configure RNDS interface in windows under "Control Panel -> Network and Internet -> Network Connections" and set IP Address

Set IP Address in Platform and start Ping

```
root@freescale /$ ifconfig usb0 10.232.1.11
root@freescale /$
root@freescale /$
root@freescale /$ ping usb   10.232.1.10
PING 10.232.1.10 (10.232.1.10): 56 data bytes
64 bytes from 10.232.1.10: seq=0 ttl=128 time=5.294 ms
64 bytes from 10.232.1.10: seq=1 ttl=128 time=6.101 ms
64 bytes from 10.232.1.10: seq=2 ttl=128 time=4.170 ms
64 bytes from 10.232.1.10: seq=3 ttl=128 time=4.233 ms
```

Known Bugs, Limitations, or Technical Issues

- Linux only allows one peripheral at one time. Make sure that when one of DWC3 controller is set as peripheral, then the others should not be set to the same mode.
- For USB host mode, some Pen drives such as Kingston / Transcend / SiliconPower / Samtec might have a compatibility issue.
- Some USB micro ports might have a OTG3.0 cable compatibility issue. An OTG 2.0 cable and USB standard port will work fine.

4.2.11 Watchdog

Module Loading
Watchdog device driver support kernel built-in mode.

U-Boot Configuration

Runtime options

Env Variable	Env Description	Sub option	Option Description
bootargs	Kernel command line argument passed to kernel	setenv othbootargs wdt_period=35	Sets the watchdog timer period timeout

Kernel Configure Options

Kernel Configure Tree View Options

Kernel Configure Tree View Options	Description

`Device Drivers --->`
`  [*] Watchdog Timer Support --->`
`  [*] Disable watchdog shutdown on close`
`  [*] IMX2+ Watchdog`
	IMX2 Watchdog Timer

Compile-time Configuration Options

Option	Values	Default Value	Description
CONFIG_IMX2_WDT	y/n	y	IMX2 Watchdog Timer

Source Files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/watchdog/imx2_wdt.c	IMX2 Watchdog Timer

User Space Application

The following applications will be used during functional or performance testing. Please refer to the UM document for the detailed build procedure.

Command Name	Description	Package Name
watch	watchdog is a daemon for watchdog feeding	watchdog

Verification in Linux

- Set NFS rootfs. Build a rootfs image which includes watchdog daemon.
- Set boot parameter. On the U-Boot prompt, set following parameter:

  Set nfsargs:

  ```
  setenv bootargs wdt_period=35 root=/dev/nfs rw nfsroot=$serverip:$rootpath ip=$ipaddr:$serverip:
  $gatewayip:$netmask:$hostname:$netdev:off
  console=$consoledev,$baudrate $othbootargs
  ```

Set nfsboot

```
run nfsargs;tftp $loadaddr $bootfile;tftp $fdtaddr $fdtfile;bootm $loadaddr - $fdtaddr
run nfsboot
```

NOTE

`wdt_period` is a watchdog timeout period. Set this parameter with the proper value depending on your board bus frequency.

`wdt_period` is inversely proportional to watchdog expiry time ie. the higher the `wdt_period`, the lower the watchdog expiry time. So if `wdt_period` is increased to high, watchdog will expiry early.

NOTE

When using watchdog as wake-up source with the default Ubuntu root filesystem, add `watchdog-device = /dev/watchdog` to `/etc/watchdog.conf`

4.2.12 Networking

4.2.12.1 Interface naming

Following section documents the association between physical interfaces and networking interfaces as presented by software.

Interface naming in U-Boot

The following figure shows the Ethernet ports as presented in U-Boot. Note that not all ports are available.

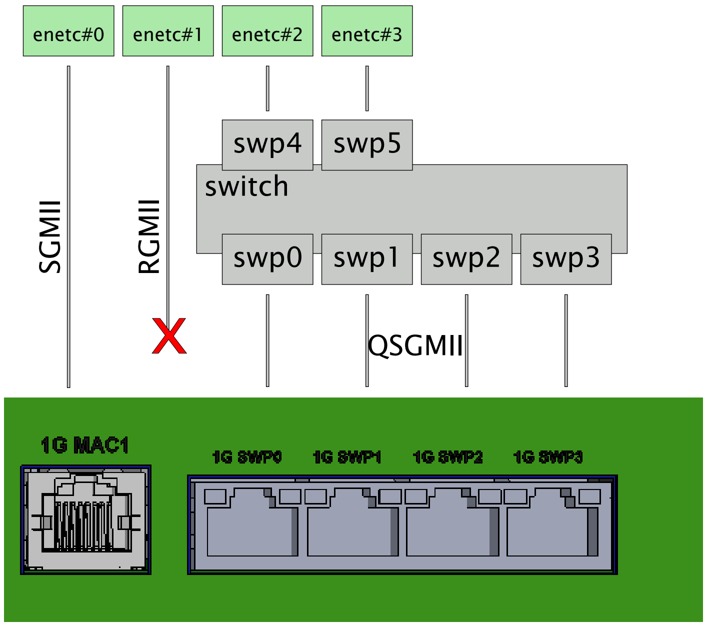


Figure 4. U-boot network interfaces on RDB

RDB port	QDS port	U-Boot interface	PCI function	Comments
1G MAC1	• Top port with **SGMII** add-on card in slot 1 • Single port of **SXGMII** add-on card in slot 1	***enetc#0***	0000:00:00.0	
N/A	On-board port	***enetc#1***	0000:00:00.1	*enetc#1* is presented in U-Boot on all boards. RGMII is only populated on QDS though, therefore, this interface is not functional on RDB.
Internal	Internal	***enetc#2***	0000:00:00.2	Connected internally (MAC to MAC) to the Ethernet switch. This interface can be used to access remote hosts connected to switch ports.
Internal	Internal	***enetc#3***	0000:00:00.6	Connected internally (MAC to MAC) to the Ethernet switch. This interface is present if bit 851 is set in RCW.
1G SWP0 to **1G SWP3**	4 ports on QS(X)GMII add-on card in slot 2. Note that SWP0 is top port on **QSGMII** add-on card and bottom port on **QSXGMII** add-on card.	N/A	0000:00:00.5	As of BSP0.3 switch ports can be used in U-Boot both to access remote hosts (through *enetc#2*) and to switch between external ports.

Interface naming in Linux

The following figure shows how Ethernet ports are presented in Linux.

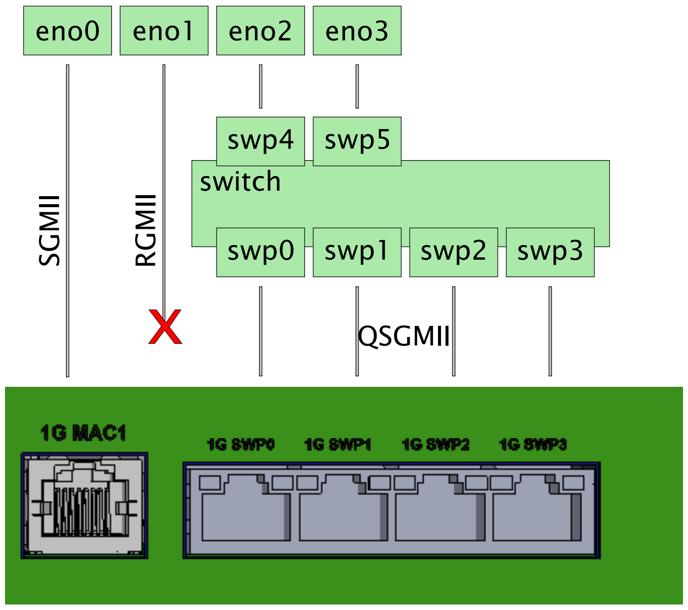


Figure 5. Linux network interfaces on RDB

RDB port	QDS port	Linux netdev	PCI function	Comments
1G MAC1	Top port with **SGMII** add-on card in slot 1 Single port of **SXGMII** add-on card in slot 1	***eno0***	0000:00:00.0	

N/A	On-board port	***eno1***	0000:00:00.1	RGMII interface is not present on RDB board and the associated ENETC interface is disabled in device tree:
				
				&enetc_port1 {
				
				status = "disabled";
				
				};
				
				On QDS this requires an RCW that explicitly enables RGMII (bits 833-838 and bits 860-861 must be reset)

Internal	Internal	***eno2***	0000:00:00.2	Connected internally (MAC to MAC) to ***swp4***. This is used to carry traffic between the switch and software running on Arm cores.

Table continues on the next page...

Table continued from the previous page...

RDB port	QDS port	Linux netdev	PCI function	Comments
Internal	Internal	***eno3***	0000:00:00.6	Connected internally (MAC to MAC) to ***swp5***. This is used to carry switch control traffic between the switch and Linux bridge. This interface is present if bit 851 is set in RCW.
1G SWP0 to **1G SWP3**	4 ports on QS(X)GMII add-on card in slot 2. Note that SWP0 is top port on **QSGMII** add-on card and bottom port on **QSXGMII** add-on card.	***swp0*** to ***swp3***	0000:00:00.5	By default, switching is not enabled on these ports. For detail on how to enable switching across these ports, see Felix Ethernet switch.
Internal	Internal	***swp4***		Connected internally (MAC to MAC) to *eno2*.
Internal	Internal	***swp5***		Connected internally (MAC to MAC) to *eno3*.

4.2.12.2 ENETC Ethernet controller

Kernel Configuration Options

Driver modules

The following table describes the Ethernet driver modules.

Driver	Description
"fsl-enetc"	ENETC Physical Function (PF) Ethernet driver
"fsl-enetc-vf"	ENETC Virtual Function (VF) Ethernet driver

Config tree view

To enable the ENETC PF and VF driver modules via `make menuconfig`:

```
Device Drivers  --->
    [*]    Network device support  --->
        [*]    Ethernet driver support  --->
            [*]    Freescale devices
            <*>        ENETC PF driver
            <*>        ENETC VF driver
```

Config option identifiers

Option	Values	Description
CONFIG_FSL_ENETC	y/m/n	ENETC Physical Function (PF) Ethernet driver
CONFIG_FSL_ENETC_VF	y/m/n	ENETC Virtual Function (VF) Ethernet driver

Device tree node

The ENETC drivers are **PCI device drivers**, and the ENETC PCI Root Complex Integrated Endpoint (RCIE) is described through the following PCIe device tree node:

```
pcie@1f0000000 {    /* rcie_enetc */
    compatible = "pci-host-ecam-generic";
    reg = <0x01 0xf0000000 0x0 0x100000>;
    #address-cells = <3>;
    #size-cells = <2>;
    #interrupt-cells = <1>;
    msi-parent = <&its>;
    device_type= "pci";
    bus-range= <0x0 0x0>;
    dma-coherent;
    msi-map = <0 &its 0 0xe>;
    iommu-map = <0 &smmu 0x4000 0xe>;
    ranges = <…>
            /* PF0-6 BAR0 - non-prefetchable memory */
            /* PF0-6, BAR2 - prefetchable memory */
            /* PF0, VF-BAR0 - non-prefetchable memory */
            /* PF0, VF-BAR2 - prefetchable memory */
            /* PF1, VF-BAR0 - non-prefetchable memory*/
            /* PF1, VF-BAR1 - prefetchable memory */
            […]
            enetc_port0: pci@0,0 {
                reg = <0x000000 0 0 0 0>;
            };
            enetc_port1: pci@0,1 {
                reg = <0x000100 0 0 0 0>;
            };
            enetc_port2: pci@0,2 {
                reg = <0x000200 0 0 0 0>;
            };
            […]
            enetc_port3: pci@0,6 {
                reg = <0x000600 0 0 0 0>;
            };
};
```

Source files

Source file	Description
enetc_pf.c, enetc_pf.h	ENETC PF driver, ENETC PSI and Port specific code

Table continues on the next page...

Table continued from the previous page...

Source file	Description
enetc_vf.c	ENETC VF driver, ENETC VSI specific code
enetc.c, enetc.h	Packet processing and other PF and VF common logic
enetc_hw.h	ENETC h/w specific defines (reg offsets, BDR structs etc.)
enetc_ethtool.c	ethtool support
enetc_cbdr.c	ENETC Control Buffer Descriptor Ring support
enetc_msg.c	ENETC VF-PF Messaging support

Linux runtime verification

ENETC PF driver probing

```
# modprobe fsl-
enetc
iommu: Adding device 0000:00:00.0 to group
0
fsl_enetc 0000:00:00.0: enabling device (0400 ->
0402)
[…]
fsl_enetc 0000:00:00.0 eth0: ENETC PF driver
v0.9
iommu: Adding device 0000:00:00.1 to group
1
fsl_enetc 0000:00:00.1: enabling device (0400 ->
0402)
[…]
fsl_enetc 0000:00:00.1 eth1: ENETC PF driver
v0.9
iommu: Adding device 0000:00:00.2 to group
2
fsl_enetc 0000:00:00.2: enabling device (0400 ->
0402)
[…]
fsl_enetc 0000:00:00.2 eth2: ENETC PF driver
v0.9
iommu: Adding device 0000:00:00.6 to group
3
fsl_enetc 0000:00:00.6: enabling device (0400 ->
0402)
[…]
fsl_enetc 0000:00:00.6 eth3: ENETC PF driver v0.9
#
```

ENETC VF driver probing

For example: probing ENETC VF0 of ENETC PF0 (Port0)

```
# echo 1 > /sys/bus/pci/devices/0000\:00\:00.0/sriov_numvfs
fsl_enetc 0000:00:00.0: SR-IOV start, 1 VFs
# modprobe fsl-enetc-
```

```
vf
iommu: Adding device 0000:00:01.0 to group 4
fsl_enetc_vf 0000:00:01.0: enabling device (0000 -> 0002)
fsl_enetc_vf 0000:00:01.0 eth4: ENETC VF driver v0.9
#
```

Supported features

The following table provides the list of supported ENETC interfaces.

Table 11. Supported ENETC interfaces

Ethernet Ports	PCIe PF id	Interface support	Supported VFs
Port 0	0	External SGMII/USXGMII (10/100/1G/2.5Gbps)	2
Port 1	1	External RGMII (10/100/1Gbps)	2
Port 2	2	Internal - EthernetSwitch@2.5Gbps (SGMII/USXGMII)	n/a
Port 3	6	Internal - EthernetSwitch @ 1Gbps (RGMII)	n/a

Supported Linux Ethernet driver features

Overview of supported Linux Ethernet driver features:

- PF and VF PCI Endpoint drivers
- Basic net-device features
- Multi-queue support – 1 Rx queue per CPU, 8 Tx queues
- Per Rx/Tx queue group MSI-X support (interrupt vector)
- SMMU support
- Rx H/W checksum offload for L3 INET_CSUM (CHECKSUM_COMPLETE)
- Unicast and multicast MAC filtering H/W offload
- VLAN filtering H/W offload
- VLAN insertion/ extraction H/W offload
- Scatter-gather (S/G) on Rx and Tx
- Jumbo frames of up to 9.6 Kbyte
- Rx flow hashing (RSS)
- Rx flow steering (RFS)
- Statistics and debug H/W counters (ethtool -S) and register dump (ethtool -d)
- VF primary MAC addressconfig and MAC anti-spoofing
- QoS – TC offloading with H/W MQPRIO

4.2.12.3 Felix Ethernet switch

4.2.12.3.1 Modules and dependencies

The driver for the LS1028A L2Switch or the Microsemi “Felix” is a two-layer driver: a common driver (shared with the Microsemi Ocelot switch driver) and a PCI driver built on top of it.

The two driver modules are described in the following table:

Module	Runtime module dependencies	Description
mscc_ocelot_common.ko	-	Common functionalities for both Ocelot and Felix drivers
mscc_felix.ko	mscc_ocelot_common.ko	The LS1028A L2Switch PCI driver

4.2.12.3.1.1 Make menuconfig path

The L2Switch driver module and its dependencies can be enabled for build using the `make menuconfig` command and selecting the below options:

```
→ Device Drivers → Network device support → Ethernet driver support → Microsemi devices →
<*> Ocelot switch driver
< > Ocelot switch driver on Ocelot
<*> Felix switch driver
```

4.2.12.3.1.2 Config defines

Module	Menuconfig name	CONFIG	Build options
mscc_ocelot_common.ko	Ocelot switch driver	CONFIG_MSCC_OCELOT_SWITCH	y/m
mscc_felix.ko	Felix switch driver	CONFIG_MSCC_FELIX_SWITCH	y/m[1]

1. Building options must confirm with the fact that the mscc_ocelot_common module dependency must be satisfied before load mscc_felix.

4.2.12.3.2 Device Tree bindings

For configuring the L2Switch ports and the attached PHYs the driver uses a custom Ethernet device tree binding named **port** node. These port nodes can only be defined within the L2Switch PCI device node ‘**pci@0,5**’. The total number of available switch ports on LS1028A is six. Depending on board design and port type (internal or external) there are two supported link modes specified by below switch port device tree bindings.

Required properties:

- **reg** – identifies the switch port id and can be within the range [0 ... 5]

1) The switch port is an external port connected to a MDIO configurable phy:

For this binding, the follwing properties are required, their bindings already defined in ethernet.txt, under *Documentation/devicetree/bindings/net/** in the kernel source tree.

Required:

- **phy-handle** – reference to a PHY device node for switch ports [0 … 3]. Defined in ethernet.txt.

- **phy-connection-type** – operation mode of PHY; for example, for LS1028ARDB this property should be set to **“qsgmii”**

2) The swicth port is an internal port, or, in some cases, has a fixed-link external connection:

In this case, the switch port node defines a fixed link connection, as specified by "fixed-link.txt", under *Documentation/devicetree/bindings/net/** in the kernel source tree.

Required:

- **fixed-link** – used primarily for ports #4 and #5 as they are internally (inside the SoC) MAC-to-MAC connected to the ENETC (Ethernet controller) endpoints ENETC PF2 (that is, ENETC Port 2) and ENETC PF6 (that is, ENETC Port 3), respectively. The "fixed-link" node is defined in "fixed-link.txt".

The only optional property is **cpu-ethernet**, custom for internal ports, and it is a phandle to an Ethernet node representing a network device physically connected to one of the switch ports. This property enables the CPU port function on the designated switch port. The CPU port function allows injecting frames on a specific switch port, the injected frames bypass the switch frame processing core. Consequently, it also allows extraction of specific control or normal Ethernet frames. The property should only be used for the SoC internal switch ports, ports #4 and #5, as they are MAC-to-MAC connected to corresponding ENETC ports.

Example

The below example enables all switch ports and configures the CPU port function on port #5. Ports 0 to 3 are connected to QSGMII PHY, while ports 4 and 5 are configured in fixed link mode:

```
pci@0,5 {
[..]
    switch_port0: port@0 {
        reg = <0>;
        phy-handle = <&qsgmii_phy1>;
        phy-connection-type = "qsgmii";
    };
    switch_port1: port@1 {
        reg = <1>;
        phy-handle = <&qsgmii_phy2>;
        phy-connection-type = "qsgmii";
    };
    switch_port2: port@2 {
        reg = <2>;
        phy-handle = <&qsgmii_phy3>;
        phy-connection-type = "qsgmii";
    };
    switch_port3: port@3 {
        reg = <3>;
        phy-handle = <&qsgmii_phy4>;
        phy-connection-type = "qsgmii";
    };
    port@4 {
        reg = <4>;
        fixed-link {
            speed = <1000>;
            full-duplex;
        };
    };
    port@5 {
        reg = <5>;
        cpu-ethernet = <&enetc_port3>
        fixed-link {
            speed = <1000>;
            full-duplex;
        };
    };
};
enetc_port3: pci@0,6 {
        reg = <0x000600 0 0 0 0>;
        fixed-link {
                speed = <1000>;
                full-duplex;
        };
};
```

4.2.12.3.3 Linux usage

4.2.12.3.3.1 Driver probing

A successful driver probe will produce the following output in the boot log:

```
[1.074990] mscc_felix 0000:00:00.5: Felix Switch Driver - version 0.3 probed
```

If the driver is successfully probed, each switch port defined in the device tree will have its own network device interface. After udev applies the networking rules, the switch network interfaces are renamed to **swpX**, where X represents the port number.

NOTE

If CPU port function is enabled on any switch port, the network device connected to it must be ready or else the driver probing will fail.

4.2.12.3.3.2 Connecting to the host CPU

On the LS1028A SoC, the L2Switch is connected to the host CPU via two SoC internal MAC-to-MAC port connections between the switch and corresponding ENETC Ethernet endpoints:

- ENETC PF2 (or ENETC Port 2) and switch port #4
- ENETC PF6 (or ENETC Port 3) and switch port #5

Besides these MAC-to-MAC connections with the host CPU, the L2Switch IP allows for a single switch port to work in CPU port mode.

The following table summarizes the differences between CPU and non-CPU port modes:

Feature	**CPU port**	**Non-CPU port**
Allows frame injection/extraction	Yes	No
Tags frames with custom header	Yes	No
Destination for control frames,for example, STP	Yes	No
PTP offload	Yes	No
Flow control	No	Yes
Requires peer net device as proxy[1]	No	Yes

1. In non-CPU mode the user needs to use the peer network interface to send and receive packets instead of the actual switch port interface. In CPU port mode, the user will use the net device interface of the switch port or the bridge interface.

The below figure shows the connection of the L2Switch with the host CPU:

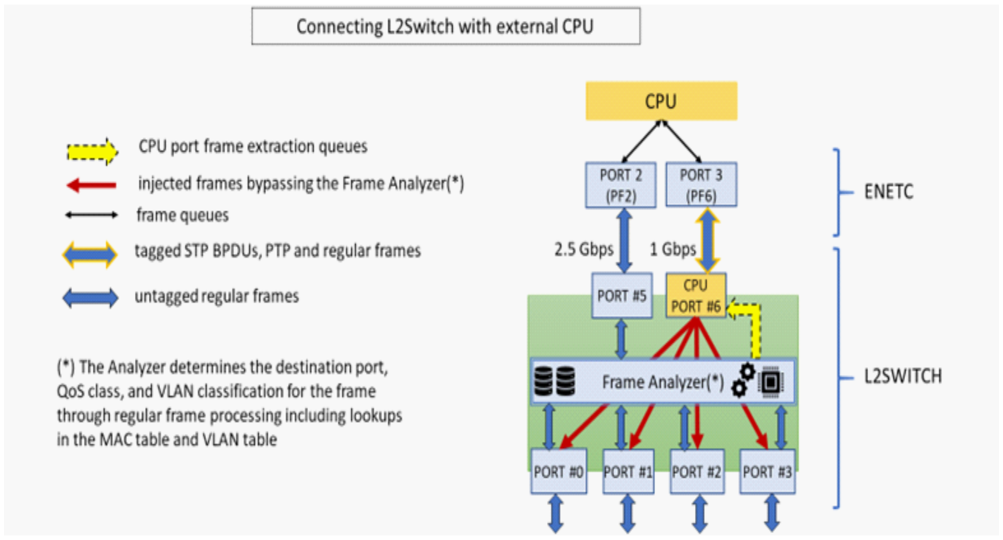

Connecting L2Switch with external CPU
CPU port frame extraction queues
injected frames bypassing the Frame Analyzer(*)
frame queues
tagged STP BPDUs, PTP and regular frames
untagged regular frames
(*) The Analyzer determines the destination port, QoS class, and VLAN classification for the frame through regular frame processing including lookups in the MAC table and VLAN table
CPU
PORT 2 (PF2)
PORT 3 (PF6)
ENETC
2.5 Gbps
1 Gbps
PORT #5
CPU PORT #6
Frame Analyzer(*)
L2SWITCH
PORT #0
PORT #1
PORT #2
PORT #3

Figure 6. Connecting L2Switch with host CPU

4.2.12.3.3.2.1 CPU port mode

The CPU port works by providing the means for the host CPU to choose a switch destination port for a frame and address control frames from known protocols, such as, STP. When this mode is enabled on one of the internal switch ports the user may use the Linux network interface assigned to that port as:

- standalone network interface - see CPU port without bridge on page 102
- child interface for a bridge interface - see CPU port with simple bridge configuration and L2 forwarding support on page 99

The two important features of CPU mode are: frame injection and frame extraction.

Frame injection

By using a custom tag or injection header prepended before the Ethernet frame header, the driver can instruct the L2Switch to forward the frame on a specific port and bypass the frame analyzer. The analyzer determines the destination port, QoS class, and VLAN classification for the frame through normal frame processing including lookups in the MAC table and VLAN table.

The tagged frame transmission is done from the peer network endpoint device. The peer net device is designated by the **cpu-ethernet** device tree property. On reception the L2Switch will strip the header, apply the frame updates (for example, write timestamp on PTP frames) and put the frame on the egress queue of the destination port.

Once configured for injection the switch port accepts only tagged frames.

Frame extraction

The L2Switch can intercept a variety of control frames or just normal frames (unicast or multicast) and redirect them to the CPU port. When a frame exits the CPU port it is also prepended, similarly to injection, by a custom tag or extraction header. This header needs to be stripped off and decoded by the switch driver to extract the ingress switch port number on which the frame was received.

Once configured for extraction the switch port emits only tagged frames.

4.2.12.3.3.2.2 Non-CPU port mode

If one of the two SoC internal ports works in non-CPU port mode then the decision to forward the frame to the host CPU or accept a frame from it depends exclusively on the frame analyzer and the MAC and VLAN tables. As mentioned earlier, the frames transiting a non-CPU port are not carrying any custom tags. In this mode only, the peer net device port (that is, ENETC port) acts like a proxy for the switch port, therefore, the user is required to use the peer's Linux network interface for sending and receiving packets from/to the switch.

4.2.12.3.3.3 L2Switch configuration examples

4.2.12.3.3.3.1 CPU port with simple bridge configuration and L2 forwarding support

The below figure describes the basic setup required to test the switch with CPU port configuration and L2 forwarding at the same time.

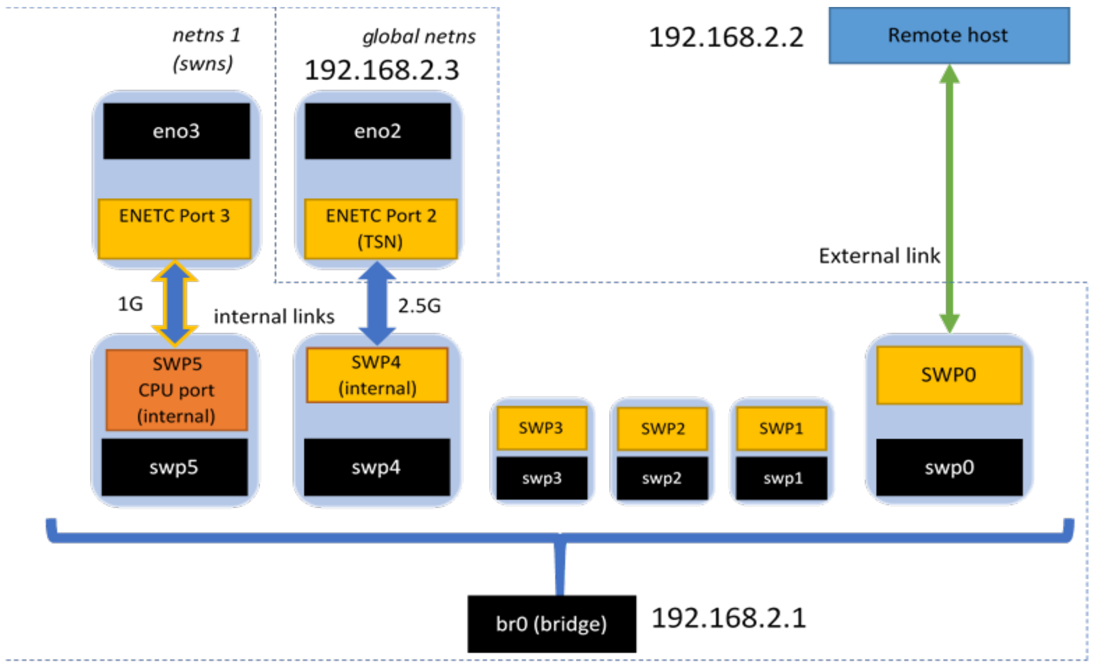


Figure 7. Basic L2Switch config with bridge interface

The setup can be explained as follows:

- Remote host with IP 192.168.2.2 connected to **SWP0**
- All **swp0** ... **swp5** interfaces are grouped under the bridge interface **br0**
- The **br0** interface must be configured with an IP address; in this example the IP is 192.168.2.1
- The **swp5** represents the **CPU port** that can be used to monitor the traffic from L2Switch side
- The **eno3** represents the ENETC Port 3 internally connected to SWP5 over a 1G link
- Both **swp5** and **eno3** net interfaces must be up
- The **eno2** (ENETC Port 2 with TSN capabilities) can be also brought up and assigned an IP address
- The L2Switch provides external connectivity for **eno2** through internal SWP4 in non-CPU mode

The following script configures the L2Switch for the above setup:

- Creates a bridge interface **br0**
- For each interface:
  - — sets the MAC address
  - — adds it to the bridge interface
  - — brings up the interface

- Brings up the **eno2** and **eno3** ENETC interfaces
- Assigns the bridge, switch port interfaces and eno3 to the same net namespace (**swns**)

```
#!/bin/bash
#
# Simple switch configuration
#
# Assume both ENETC and Felix drivers are already loaded
#

BRIDGE=br0
MAC_ROOT=bc:8d:bf:7c:5b
SW_NETNS=swns
EXEC_SWNS="ip netns exec $SW_NETNS"

# Create bridge namespace
ip netns add $SW_NETNS
# Create bridge device in net namespace
$EXEC_SWNS ip link add name $BRIDGE type bridge
$EXEC_SWNS ip link set $BRIDGE up

# Configure switch ports
#  * set MAC address
#  * bring up interface
#  * move net device into the bridge net namespace
#  * set bridge device as master
swps=($(ls /sys/bus/pci/devices/0000:00:00.5/net/))
nr=${#swps[@]}
for (( i=0; i<$nr; i++ ))
do
    echo "adding ${swps[$i]} to brigde .."
    ip link set ${swps[$i]} address $MAC_ROOT:$(echo "${swps[$i]}" | tr -dc '0-9')
    ip link set ${swps[$i]} netns $SW_NETNS
    $EXEC_SWNS ip link set ${swps[$i]} master $BRIDGE
    $EXEC_SWNS ip link set ${swps[$i]} up
done

# bring up ENETC ports connected to switch ports
enetc2=$(ls /sys/bus/pci/devices/0000:00:00.2/net/)
ip link set $enetc2 up

# move ENETC port connected to switch CPU port in bridge ns
enetc3=$(ls /sys/bus/pci/devices/0000:00:00.6/net/)
ip link set $enetc3 netns $SW_NETNS
$EXEC_SWNS ip link set $enetc3 up

# Check configuration
$EXEC_SWNS bridge link show
```

The script output appears as follows:

```
br0: port 1(swp0) entered blocking state
br0: port 1(swp0) entered disabled state
device swp0 entered promiscuous mode
mscc_felix 0000:00:00.5 swp0: Unsupported PHY speed: 0
Vitesse VSC8514 1f8100000:10: attached PHY driver [Vitesse VSC8514] (mii_bus:phy_addr=1f8100000:10,
irq=POLL)
br0: port 1(swp0) entered blocking state
br0: port 1(swp0) entered forwarding state
```

```
8021q: adding VLAN 0 to HW filter on device swp0
br0: port 2(swp1) entered blocking state
br0: port 2(swp1) entered disabled state
device swp1 entered promiscuous mode
mscc_felix 0000:00:00.5 swp1: Unsupported PHY speed: 0
Vitesse VSC8514 1f8100000:11: attached PHY driver [Vitesse VSC8514] (mii_bus:phy_addr=1f8100000:11,
irq=POLL)
br0: port 2(swp1) entered blocking state
br0: port 2(swp1) entered forwarding state
8021q: adding VLAN 0 to HW filter on device swp1
br0: port 3(swp2) entered blocking state
br0: port 3(swp2) entered disabled state
device swp2 entered promiscuous mode
mscc_felix 0000:00:00.5 swp2: Unsupported PHY speed: 0
Vitesse VSC8514 1f8100000:12: attached PHY driver [Vitesse VSC8514] (mii_bus:phy_addr=1f8100000:12,
irq=POLL)
br0: port 3(swp2) entered blocking state
br0: port 3(swp2) entered forwarding state
8021q: adding VLAN 0 to HW filter on device swp2
br0: port 4(swp3) entered blocking state
br0: port 4(swp3) entered disabled state
device swp3 entered promiscuous mode
mscc_felix 0000:00:00.5 swp3: Unsupported PHY speed: 0
Vitesse VSC8514 1f8100000:13: attached PHY driver [Vitesse VSC8514] (mii_bus:phy_addr=1f8100000:13,
irq=POLL)
br0: port 4(swp3) entered blocking state
br0: port 4(swp3) entered forwarding state
8021q: adding VLAN 0 to HW filter on device swp3
br0: port 5(swp4) entered blocking state
br0: port 5(swp4) entered disabled state
device swp4 entered promiscuous mode
mscc_felix 0000:00:00.5 swp4: Link is Up - 1Gbps/Full - flow control off
Generic PHY fixed-0:02: attached PHY driver [Generic PHY] (mii_bus:phy_addr=fixed-0:02, irq=POLL)
br0: port 5(swp4) entered blocking state
br0: port 5(swp4) entered forwarding state
8021q: adding VLAN 0 to HW filter on device swp4
br0: port 6(swp5) entered blocking state
br0: port 6(swp5) entered disabled state
device swp5 entered promiscuous mode
mscc_felix 0000:00:00.5 swp5: Link is Up - 1Gbps/Full - flow control off
Generic PHY fixed-0:03: attached PHY driver [Generic PHY] (mii_bus:phy_addr=fixed-0:03, irq=POLL)
br0: port 6(swp5) entered blocking state
br0: port 6(swp5) entered forwarding state
8021q: adding VLAN 0 to HW filter on device swp5
Generic PHY fixed-0:00: attached PHY driver [Generic PHY] (mii_bus:phy_addr=fixed-0:00, irq=POLL)
8021q: adding VLAN 0 to HW filter on device eno2
mscc_felix 0000:00:00.5 swp0: Link is Down
Generic PHY fixed-0:01: attached PHY driver [Generic PHY] (mii_bus:phy_addr=fixed-0:01, irq=POLL)
8021q: adding VLAN 0 to HW filter on device eno3
mscc_felix 0000:00:00.5 swp1: Link is Down
mscc_felix 0000:00:00.5 swp2: Link is Down
mscc_felix 0000:00:00.5 swp3: Link is Down
mscc_felix 0000:00:00.5 swp4: Link is Up - 1Gbps/Full - flow control off
br0: port 1(swp0) entered disabled state
br0: port 2(swp1) entered disabled state
br0: port 3(swp2) entered disabled state
br0: port 4(swp3) entered disabled state
mscc_felix 0000:00:00.5 swp5: Link is Up - 1Gbps/Full - flow control off
fsl_enetc 0000:00:00.2 eno2: Link is Up - 1Gbps/Full - flow control off
fsl_enetc 0000:00:00.6 eno3: Link is Up - 1Gbps/Full - flow control off
```

```
mscc_felix 0000:00:00.5 swp0: Link is Up - 1Gbps/Full - flow control rx/tx
br0: port 1(swp0) entered blocking state
br0: port 1(swp0) entered forwarding state
Generic PHY fixed-0:00: attached PHY driver [Generic PHY] (mii_bus:phy_addr=fixed-0:00, irq=POLL)
8021q: adding VLAN 0 to HW filter on device eno2
fsl_enetc 0000:00:00.2 eno2: Link is Up - 1Gbps/Full - flow control off
```

When configuring the L2Switch ports in a bridge interface we need to set the IP on this interface. In this case the Linux network stack will take care of sending packets to switch ports and forward the ingress port traffic to the bridge interface.

Adding an IP address for **br0**:

```

bash-4.4# $EXEC_SWNS ip addr add 192.168.2.1/24 dev br0
bash-4.4# $EXEC_SWNS ip addr show dev br0
2: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1468 qdisc noqueue state UP group default qlen 1000
    link/ether bc:8d:bf:7c:5b:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 scope global br0
       valid_lft forever preferred_lft forever
```

Once the IP address is set on the **br0** interface, the bridge is ready to receive and send packets:

```

bash-4.4# $EXEC_SWNS  ip -s link show dev br0
2: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1468 qdisc noqueue state UP mode DEFAULT group default
qlen 1000
    link/ether bc:8d:bf:7c:5b:00 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    0          0        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    0          0        0       0       0       0
bash-4.4#
bash-4.4#  $S ping 192.168.2.2 -c3
PING 192.168.2.2 (192.168.2.2) 56(84) bytes of data.
64 bytes from 192.168.2.2: icmp_seq=1 ttl=64 time=0.331 ms
64 bytes from 192.168.2.2: icmp_seq=2 ttl=64 time=0.098 ms
64 bytes from 192.168.2.2: icmp_seq=3 ttl=64 time=0.152 ms

--- 192.168.2.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2033ms
rtt min/avg/max/mdev = 0.098/0.193/0.331/0.100 ms
bash-4.4# $S ip -s link show dev br0
2: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1468 qdisc noqueue state UP mode DEFAULT group default
qlen 1000
    link/ether bc:8d:bf:7c:5b:00 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    344        5        0       0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    378        5        0       0       0       0
bash-4.4#
```

4.2.12.3.3.3.2 CPU port without bridge

In this configuration mode, the traffic received on all external ports is forwarded to the CPU port as in previous example (bridge mode), however, the L2 forwarding does not work by default. Another difference from bridge mode is that each switch port interface can be used independently to send and receive packets. Figure 8. on page 103 shows the high-level view of this mode.

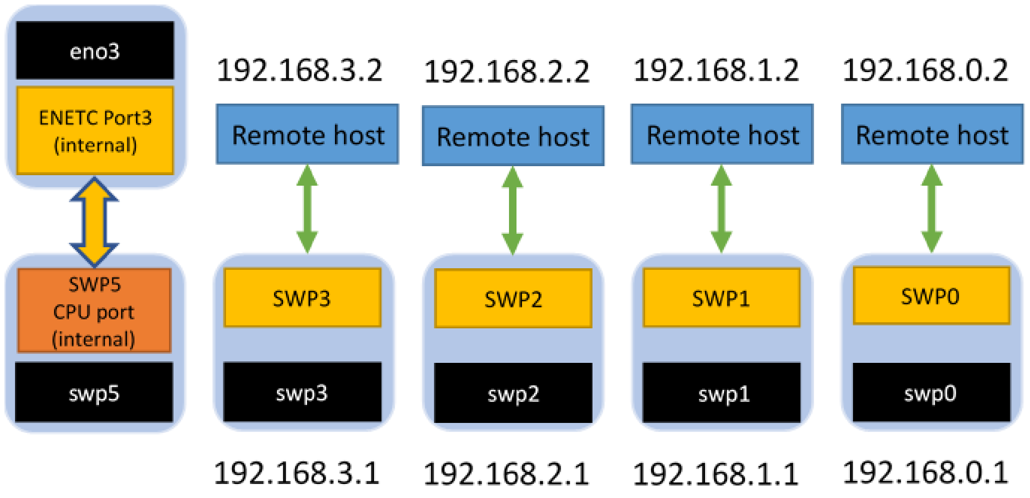


Figure 8. L2Switch configuration without bridge interface

The configuration script on the switch side is as follows:

```
#!/bin/bash
#
# Simple switch configuration without bridge
# Assume both ENETC and Felix drivers are already loaded

MAC_ROOT=bc:8d:bf:7c:5b
swpip=192.168

# Configure switch ports
swps=($(ls /sys/bus/pci/devices/0000:00:00.5/net/))
let nr=${#swps[@]}
for (( i=0; i<&nr; i++ ))
do
    ip link set ${swps[$i]} address $MAC_ROOT:$(echo "${swps[$i]}" | tr -dc '0-9')
    ip addr add ${swpip}.${i}.1/24 dev ${swps[$i]}
    ip link set ${swps[$i]} up
done

# bring up ENETC Port3
seth=$(ls /sys/bus/pci/devices/0000:00:00.6/net/)
ip link set $seth up
```

4.2.13 TSN

4.2.13.1 Tsntool user manual

The tsntool is a tool to set the hardware TSN capability ethernet ports.

Source code

Use the following link to get the access of tsntool source code: https://github.com/openil/tsntool.git.

tsntool commands support

The following table describes the tsntool supported commands.

Command	Description
help	List commands support
version	Show software version
verbose	Debug on/off for tsntool
quit	Quit prompt mode
qbvset	Set time gate scheduling config for <ifname>
qbvget	Get time scheduling entries for <ifname>
cbstreamidset	Set stream identify table
cbstreamidget	Get stream identify table and counters
qcisfiset	Set stream filter instance
qcisfiget	Get stream filter instance
qcisgiset	Set stream gate instance
qcisgiget	Get stream gate instance
qcisficounterget	Get stream filter counters
qcifmiset	Set flow metering instance
qcifmiget	Get flow metering instance
cbsset	Set TCs credit-based shaper configure
cbsget	Get TCs credit-based shaper status
qbuset	Set one 8-bit vector showing the preemptable traffic class
qbugetstatus	Not supported
tsdset	Not supported
tsdget	Not supported
ctset	Set cut through queue status (specific for LS1028A switch)
cbgen	Set sequence generate configure (specific for LS1028A switch)
cbrec	Set sequence recover configure (specific for ls1028 switch)
pcpmap	Set queues map to PCP tag (specific for LS1028A switch)
sendpkt	Not supported
regtool	Register read/write of bar0 of PFs (specific for LS1028A ENETC)
ptptool	Not supported

Tsntool command parameters

The following table describes the tsntool supported command parameters.

Parameters	Description
qbvset command	
--device <ifname>	Interface, such as eth0/sw0p0
--entryfile <filename>	A file script to input gatelist format: #'NUMBER' 'GATE_VLAUE' 'TIME_LONG' # #NUMBER: # 't' or 'T' head. Plus entry number. Duplicate entry number will got error. # #GATE_VALUE: # format: xxxxxxxxb . # The MS bit corresponds to traffic class 7. The LS bit to traffic class 0. # A bit value of 0 indicates closed, A bit value of 1 indicates open. # #TIME_LONG: # nanoseconds. Do not input 0 time long. t0 11101111b 10000 t1 11011111b 10000 **NOTE**: entryfile parameter is must set. If not set, a vi text editor prompt is required to input the gate list.
--basetime <value>	AdminBaseTime
--cycletime <value>	AdminCycleTime
--cycleextend <value>	AdminCycleTimeExtension
--enable | --disable	enable: enable the qbv for this port disable: disable the qbv for this port Default to set enable if no enable or disable input
--maxsdu <value>	queueMaxSDU
--initgate <value>	AdminGateStates
--configchange	ConfigChange. Default set to 1.
--configchangetime <value>	ConfigChangeTime
`cbstreamidget` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	Index entry number in this controller. Must have.

Table continues on the next page...

Table continued from the previous page...

Parameters	**Description**
`qcisfiset` **command**	
--device <ifname>	interface, such as eth0/sw0p0
--enable | --disable	enable: enable the entry for this index disable: disable the entry for this index default to set enable if no enable or disable input
--maxsdu <value>	Maximum SDU size.
--flowmeterid <value>	Flow meter instance identifier index number.
--index <value>	StreamFilterInstance. Index entry number in this controller.
--streamhandle <value>	StreamHandleSpec This value corresponds to “ tsnStreamIdHandle ” of “ cbstreamidset ” command.
--priority <value>	PrioritySpec
--gateid <value>	StreamGateInstanceID
--oversizeenable	StreamBlockedDueToOversizeFrameEnable
--oversize	StreamBlockedDueToOversizeFrame
`qcisfiget` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	Index entry number in this controller. Must have.
`qcisgiset` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	Index entry number in this controller. Must have.
--enable | --disable	enable: enable the entry for this index. PSFPGateEnabled disable: disable the entry for this index Default to set enable if no enable or disable input
--configchange	configchange
--enblkinvrx	PSFPGateClosedDueToInvalidRxEnable
--blkinvrx	PSFPGateClosedDueToInvalidRx
--initgate	PSFPAdminGateStates
--initipv	AdminIPV
--cycletime	Default not set. Get by gatelistfile.
--cycletimeext	PSFPAdminCycleTimeExtension
--basetime	PSFPAdminBaseTime

Table continues on the next page...

Table continued from the previous page...

Parameters	**Description**
--gatelistfile	PSFPAdminControlList. A file input the gate list: #'NUMBER' 'GATE_VLAUE' 'IPV' 'TIME_LONG' 'OCTET_MAX' # #NUMBER: # 't' or 'T' head. Plus entry number. Duplicate entry number will got error. # #GATE_VALUE: # format: xb # The MS bit corresponds to traffic class 7. The LS bit to traffic class 0. # A bit value of 0 indicates closed, A bit value of 1 indicates open. # #IPV: # 0~7 # #TIME_LONG: # nanoseconds. Do not input 0 time long. # #OCTET_MAX: # The maximum number of octets that are permitted to pass the gate. # If zero, there is no maximum. t0 1b -1 50000 10
`qcisgiget` **command**	
--device <ifname>	interface like eth0/sw0p0
--index <value>	index entry number in this controller. Must have.
`qcifmiset` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	Index entry number in this controller. Must have.
--disable	Not set disable then to be set enable.
--cir <value>	cir. kbit/s.
--cbs <value>	cbs. octets.

Table continues on the next page...

Table continued from the previous page...

Parameters	**Description**
--eir <value>	eir.kbit/s.
--ebs <value>	ebs.octets.
--cf	cf. couple flag.
--cm	cm. color mode.
--dropyellow	drop yellow.
--markred_enable	mark red enable.
--markred	mark red.
`qcifmiget` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	Index entry number in this controller. Must have.
`qbuset` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--preemptable <value>	8 bit hex value. Example: 0xfe The MS bit corresponds to traffic class 7. The LS bit to traffic class 0. A bit value of 0 indicates express. A bit value of 1 indicates preemptable.
`cbsset` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--tc <value>	Traffic class number.
--percentage <value>	Set percentage of tc limitation.
--all <tc-percent:tc-percent...>	Not supported.
`cbsget` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--tc <value>	Traffic class number.
`regtool` **command**	
	Usage: regtool { pf number } { offset } [data]
	pf number: pf number for the pci resource to act on
	offset : offset into pci memory region to act upon
	data : data to be written
`ctset` **command**	
--device <ifname>	Interface, such as eth0/sw0p0

Table continues on the next page...

Table continued from the previous page...

Parameters	**Description**
--queue_stat <value>	Specifies which priority queues have to be processed in cut-through mode of operation. Bit 0 corresponds to priority 0, Bit 1 corresponds to priority 1 so on.
`cbgen` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	Index entry number in this controller. Must have. This value corresponds to " tsnStreamIdHandle " of " cbstreamidset " command.
--iport_mask <value>	INPUT_PORT_MASK: If the packet is from input port belonging to this port mask, then it is a known stream and Sequence generation parameters can be applied.
--split_mask <value>	SPLIT_MASK: Port mask used to add redundant paths (or ports) if split is enabled (STREAM_SPLIT) for a stream. This is OR'ed with the final port mask determined by the forwarding engine.
--seq_len <value>	SEQ_SPACE_LOG2: Min value is 1 and maximum value is 28. tsnSeqGenSpace = 2**SEQ_SPACE_LOG2 For example, if this value is 12, the valid sequence numbers are from 0x0 to 0xFFF.
--seq_num <value>	GEN_REC_SEQ_NUM: The sequence number to be used for outgoing packet passed to SEQ_GEN function. Note: Only lower 16 bits are sent in RED_TAG.
`cbrec` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--index <value>	index entry number in this controller. Must have. This value corresponds to " tsnStreamIdHandle " of " cbstreamidset " command.
--seq_len <value>	SEQ_SPACE_LOG2: Min value is 1 and maximum value is 28. tsnSeqRecSeqSpace = 2**SEQ_REC_SPACE_LOG2 For example, if this value is 12, the valid sequence numbers are from 0x0 to 0xFFF.
--his_len <value>	SEQ_HISTORY_LEN: Refer to SEQ_HISTORY, Min 1 and Max 32.
--rtag_pop_en	REDTAG_POP: If True, then the redundancy tag is popped by rewriter.

Table continues on the next page...

Table continued from the previous page...

Parameters	Description
`pcpmap` **command**	
--device <ifname>	Interface, such as eth0/sw0p0
--enable	Enable pcp to traffic class for frames.

Shortcut keys

- When you input any command, use Tab key to get list of related commands. For example:

  ```
  tsntool> qbv
  ```

  Pressing Tab key after input above command, you can get list of related “qbv*” start commands. If there is only one option, the complete command comes automatically.
- When you input a command and do not remember the parameter names, you can just input “--” and press Tab key. This shortcut shows all parameters of the command. If you input half of the parameter name, press Tab key can list all the related names.
- Non-interactive mode: In non-interactive mode, input the command list following up the tsntool command.

  For example, if a command in interactive mode is as follows:

  ```
  tsntool> qbuset --device eth0 --preemptable 0xfe
  ```

  The same command in non-interactive mode (for example, Linux shell) can be input as follows:

  ```
  tsntool qbuset --device eth0 --preemptable 0xfe
  ```

4.2.13.2 Kernel configuration

In the kernel, select the CONFIG:

```

| Symbol: TSN
[=y]

  | Type  :
boolean

  | Prompt: 802.1 Time-Sensitive Networking
support
  |
Location:

  |     -> Networking support (NET
[=y])
  |    -> Networking
options

  |   Depends on: NET [=y] && VLAN_8021Q [=y] && PTP_1588_CLOCK
[=y]

 | Symbol: ENETC_TSN
[=y]
```

```
  | Type  :
boolean

  | Prompt: TSN Support for NXP ENETC
driver
  |
Location:

  |     -> Device
Drivers

  |       -> Network device support (NETDEVICES
[=y])
  |         -> Ethernet driver support (ETHERNET
[=y])
  |        -> Freescale devices (NET_VENDOR_FREESCALE
[=y])
  |   Defined at drivers/net/ethernet/freescale/enetc/Kconfig:
41
  |   Depends on: NETDEVICES [=y] && ETHERNET [=y] && NET_VENDOR_FREESCALE [=y] && FSL_ENETC [=m] &&
TSN [=y]

  | Symbol: FSL_ENETC_PTP_CLOCK [=y]
  | Type  : tristate
  | Prompt: ENETC PTP clock driver
  |   Location:
  |     -> Device Drivers
  |       -> Network device support (NETDEVICES [=y])
  |         -> Ethernet driver support (ETHERNET [=y])
  |           -> Freescale devices (NET_VENDOR_FREESCALE [=y])


  | Symbol: FSL_ENETC_HW_TIMESTAMPING [=y]
  | Type  : boolean
  | Prompt: ENETC hardware timestamping support
  |   Location:
  |     -> Device Drivers
  |       -> Network device support (NETDEVICES [=y])
  |         -> Ethernet driver support (ETHERNET [=y])
  |           -> Freescale devices (NET_VENDOR_FREESCALE [=y])

| Symbol: MSCC_FELIX_SWITCH_TSN [=y]
     | Type  : tristate
     |  Prompt: TSN on FELIX switch driver
     | Location:
     |     -> Device Drivers
     |         -> Network device support (NETDEVICES [=y])
     |             -> Ethernet driver support (ETHERNET [=y])
     |                 -> Microsemi devices (NET_VENDOR_MICROSEMI [=y])
     |                     -> Ocelot switch driver (MSCC_OCELOT_SWITCH [=y])
     |                      -> FELIX switch driver (MSCC_FELIX_SWITCH [=y])
     |  Defined at drivers/net/ethernet/mscc/Kconfig:38
```

4.2.13.3 Basic TSN Configure examples on ENETC

The **tsntool** is an application configuration tool to configure the TSN capability. You can find files **/usr/bin/tsntool** and **/usr/lib/libtsn.so** in the rootfs. Run **tsntool** to start the setting shell.

4.2.13.3.1 Linuxptp test

To test 1588 synchronization on ENETC interface, connect ENETC interface on two boards using back-to-back method. (For example, eth0 to eth0.)

The Linux booting log shows the following:

```
…
pps pps0: new PPS source ptp0
…
```

Check PTP clock and timestamping capability:

```
# ethtool -T eth0
Time stamping parameters for eth0:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 0
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
        on                    (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                  (HWTSTAMP_FILTER_NONE)
        all                   (HWTSTAMP_FILTER_ALL)
```

Configure IP address and run `ptp4l` on two boards:

```
# ifconfig eth0 <ip_addr>
# ptp4l -i eth0 -p /dev/ptp0 -m
```

After running the above commands, one board is selected as master automatically, and the slave board prints synchronization messages.

For 802.1AS testing, use configure file **gPTP.cfg** in linuxptp source. Run the following commands on boards instead.

```
# ptp4l -i eth0 -p /dev/ptp0 -f gPTP.cfg -m
```

4.2.13.3.2 Qbv test

Following are the qbv tests:

Case 1: Basic gates close

```
cat > qbv0.txt << EOF
t0      00000000b               20000
EOF

#Exmplain:
# 'NUMBER'         :     t0
# 'GATE_VLAUE'     :     00000000b
# 'TIME_LONG'         :     20000 ns

cp libtsn.so /lib
./tsntool
```

```
tsntool> verbose
tsntool> qbvset --device eth0 --entryfile ./qbv0.txt

ethtool -S eth0
ping 192.168.0.2 -c 1   #should not pass any frame since gates all off.
```

Case 2: basetime test

The following result is based on case 1 qbv1.txt gate list.

```

#create 1s gate
cat > qbv1.txt << EOF
t0  11111111b      10000
t1  00000000b      99990000
EOF

tsntool> regtool 0 0x18
tsntool> regtool 0 0x1c
#read the current time and add 0x2000000000, about 2 minutes, for example 0x2000011234 as result
tsntool> qbvset --device eth0 --entryfile qbv1.txt --basetime 0x2000011234
tsntool> qbvget --device eth0 #You can check configchange time
tsntool> regtool 0 0x11a10 #check pending status, 0x1 means time gate is working
#waiting to change state, ping remote computer
ping 192.168.0.2 -A -s 1000
#The reply time will be about 100ms
```

Since 10000 ns max limit package size 1250 bytes.

```
ping 192.168.0.2 -c 1 -s 1300  #frame should not pass
```

Case 3: Qbv performance test

Test ENETC port0(MAC0)

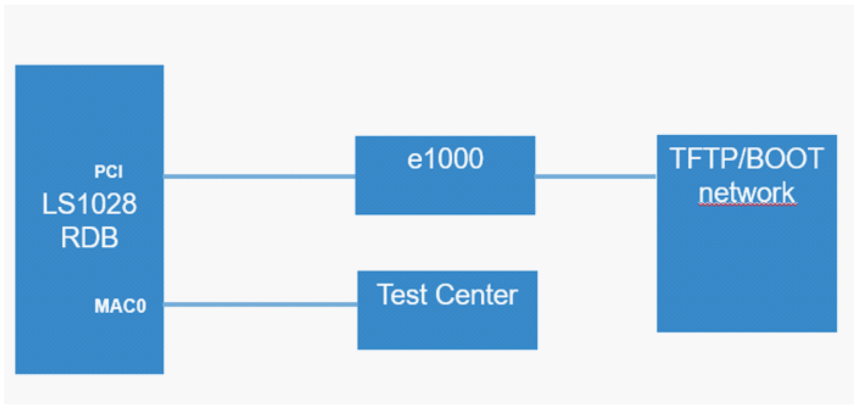


Figure 9. Setup

```

cat > qbv5.txt << EOF
t0  11111111b       1000000
t1  00000000b      1000000
EOF

qbvset --device eth0 --entryfile qbv5.txt
./pktgen/pktgen_twoqueue.sh -i eth0 -q 3 -n 0
```

```
#The stream would got about half line rate
```

4.2.13.3.3 Qci test cases

Perform background setting as follows:

- Set eth0 MAC address:

  ```
  ip link set eth0 address 10:00:80:00:00:00
  ```

  Testcenter MAC address **99:aa:bb:cc:dd:ee** as example

- SETUP 2 as hardware setup:

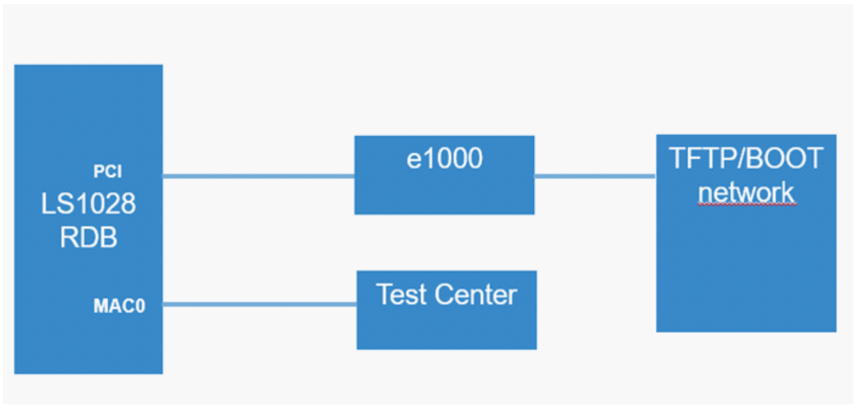


Figure 10. SETUP2

Test SFI No Streamhandle

Follow the following procedure to test no streamhandle for stream filter.

1. Set a close gate stream id 2, no stream identify package check and other streams will pass gate.

   ```
   tsntool> qcisfiset --device eth0 --index 2 --gateid 2
   ```

   Streams no streamhandle should pass this filter.

   ```
   tsntool> qcisfiget --device eth0 --index 2
   ```

2. Send one frame from Test center.

   ```
   tsntool> qcisfiget --device eth0 --index 2
   ```

3. Set Stream Gate entry 2.

   ```
   tsntool> qcisgiset --device eth0 --index 2 --initgate 1
   ```

4. Send one frame from Test center.

   ```
   tsntool> qcisfiget --device eth0 --index 2
   ```

5. Set Stream Gate entry 2, gate close permanently.

   ```
   tsntool> qcisgiset --device eth0 --index 2 --initgate 0
   ```

6. Send one frame from Test center.

```
tsntool>qcisfiget --device eth0 --index 2
#should look result like below:
match   pass   gate_drop   sdu_pass   sdu_drop   red
   1   0   1   1   0   0
```

Test Null Stream identify Entry

Follow the following procedure to set main stream by close gate.

1. Set Stream identify Null stream identify entry 1.

```
tsntool> cbstreamidset --device eth0 --index 1 --nullstreamid --nulldmac 0x000000800010
        --nulltagged 3 --nullvid 10 --streamhandle 100
```

2. Get SID index 1.

```
tsntool> cbstreamidget --device eth0 --index 1
```

3. Set Stream filer entry 1.

```
tsntool> qcisfiset --device eth0 --streamhandle 100 --index 1 --gateid 1
```

4. Set Stream Gate entry 1.

```
tsntool> qcisgiset --device eth0 --index 1 --initgate 0
```

5. Send one frame from Test center.

```
tsntool> qcisfiget --device eth0 --index 1
#should look like below:
match   pass   gate_drop   sdu_pass   sdu_drop   red
   1   0   1   1   0   0
```

Test Source Stream identify Entry

1. Keep Stream Filter entry 1 and Stream gate entry 1.
2. Add stream2 in test center: SMAC is 66:55:44:33:22:11 DMAC:20:00:80:00:00:00
3. Set Stream identify Source stream identify entry 3.

```
tsntool> cbstreamidset --device eth0 --index 3 --sourcemacvid --sourcemac 0x112233445566 --
sourcetagged 3 --sourcevid 20 --streamhandle 100
```

4. Send frame from test center. The frame passes to stream filter index 1.

```
tsntool> qcisfiget --device eth0 --index 1
```

SGI stream gate list

```
cat > sgi1.txt << EOF
t0  0b  -1   1000     0
t1  1b  -1   1000     0
EOF

tsntool> qcisfiset --device eth0 --index 2 --gateid 2
tsntool> qcisgiset --device eth0 --index 2 --initgate 1 --gatelistfile sgi1.txt
#fluding frame size 64bytes at test center
tsntool> qcisfiget --device eth0 --index 2
#check the frames dropped and passed, should be same.
```

FMI test

Only send green color frames, set test center speed to 10000000 bsp/s:

```

tsntool> qcisfiset --device eth0 --index 2 --gateid 2 --flowmeterid 2
tsntool> qcifmiset --device eth0 --index 2 --cm --cf --cbs 1500 --cir 50000 --ebs 1500 --eir 5000000
```

Below setting shows drop frames.

```
tsntool> qcifmiset --device eth0 --index 2 --cm --cf --cbs 1500 --cir 50000 --ebs 1500 --eir 2000000
```

The following example shows how to get information of color frame counters showing at application layer.

```
tsntool> qcifmiget --device eth0 --index 2
======================================================================
bytecount   drop   dr0_green   dr1_green   dr2_yellow   remark_yellow   dr3_red   remark_red
1c89   0   4c   0   0   0   0   0
======================================================================
   index = 2
   cir = c34c
   cbs = 5dc
   eir = 4c4b3c
   ebs = 5dc
   couple flag
   color mode
```

4.2.13.3.4 Qbu test

Set frame path from eth0 to external by link enetc MAC0 - SWP0.

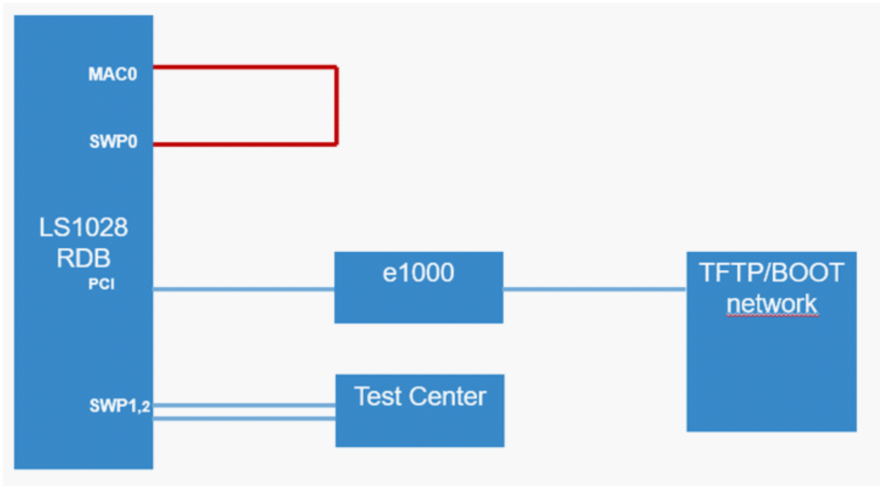


Figure 11. QBU test setup

NOTE

- 0x11f10 Port MAC Merge Frame Assembly OK Count Register
- 0x11f18 Port MAC Merge Fragment Count TX Register (MAC_MERGE_MMFCTXR)

If link ENETC port0 to SWP0:

```
#Set switch to merge enable:
devmem 0x1fc100048 32 0x111     #DEV_GMII:MM_CONFIG:ENABLE_CONFIG

#ENETC port setting
ip link set eth0 address 90:e2:ba:ff:ff:ff

tsntool> qbuset --device eth0 --preemptable 0xfe

./pktgen/pktgen_twoqueue.sh -i eth0 -q 0 -s 100 -n 2000 -m 90:e2:ba:ff:ff:ff

tsntool> regtool 0 0x11f18   #check tx merge counter, if non-zero means Qbu working.
```

4.2.13.3.5 Qav test

Following is the hardware test diagram:

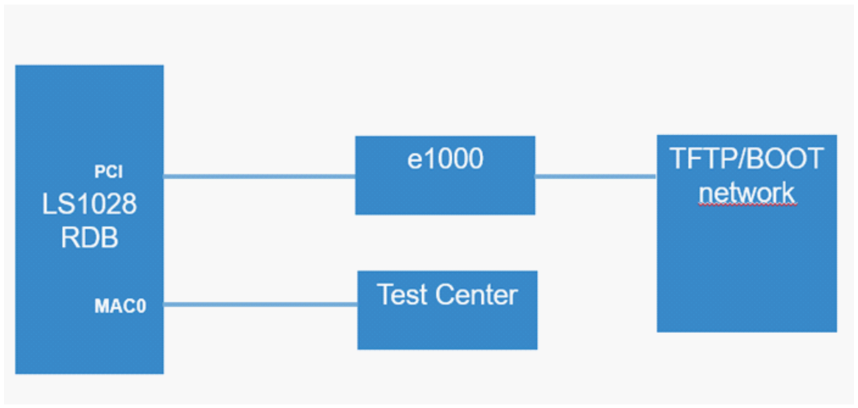


Figure 12. Hardware test diagram (Setup2)

```
cbsset --device eth0 --tc 7 --percentage 60
cbsset --device eth0 --tc 6 --percentage 20
./pktgen/pktgen_twoqueue.sh -i eth0 -q 6 -s 1500 -n 0 -m 10:11:12:13:14:15
#check the test center result, TC6 should 1/3 frames of TC7.
```

Check one queue:

```
./pktgen/pktgen_sample01_simple.sh -i eth0 -q 7 -s 500 -n 0
```

This gets approx 60% percentage line rate.

4.2.13.4 Basic TSN configure examples on switch

4.2.13.4.1 Switch configuration

The following figure shows the switch configuration.

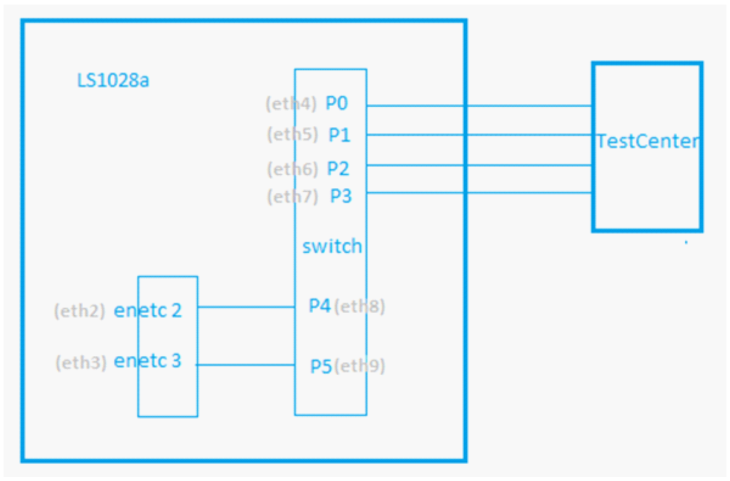


Figure 13. Switch configuration

```
ls /sys/bus/pci/devices/0000:00:00.5/net/
#Get switch device interfaces: eth4 eth5 eth6 eth7 eth8 eth9

ip link add name switch type bridge
ip link set switch up
ip link set eth4 master switch && ip link set eth4 up
ip link set eth5 master switch && ip link set eth5 up
ip link set eth6 master switch && ip link set eth6 up
ip link set eth7 master switch && ip link set eth7 up
ip link set eth8 master switch && ip link set eth8 up
ip link set eth9 master switch && ip link set eth9 up
```

4.2.13.4.2 Enable timestamp on switch

Follow the following commands to enable timestamp on switch:

```
#Init PTP:
devmem 0x1fc0900a0 w 0x00000004

#Get PTP real time(second):
devmem 0x1fc0900c4
```

4.2.13.4.3 Qbv test

The following figure shows the Qbv test setup.

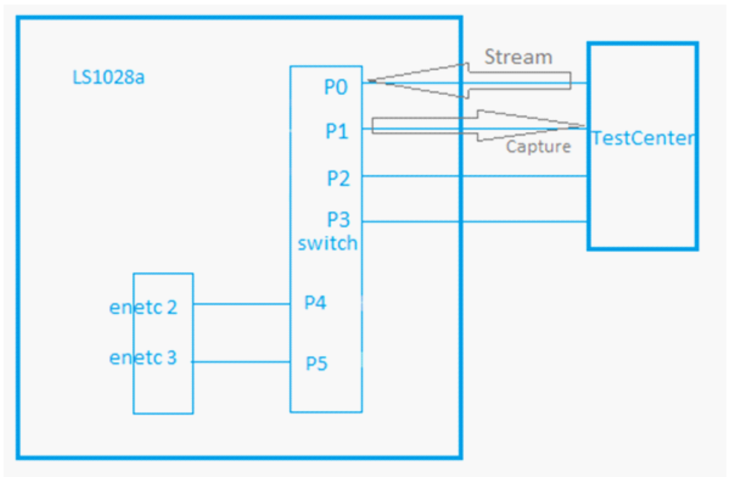


Figure 14. Qbv test setup

Basic gates close

```
echo  “t0  00000000b  20000”  >  qbv0.txt
#Explain:
# 'NUMBER'         :     t0
# 'GATE_VLAUE'     :     00000000b
# 'TIME_LONG'         :     20000 ns

cp libtsn.so /lib
./tsntool
tsntool> verbose
tsntool> qbvset --device eth5 --entryfile ./qbv0.txt

#send one broadcast frame to eth4 from TestCenter.
ethtool -S eth5
#Should not get any frame from eth5 on TestCenter.

echo  “t0  11111111b  20000”  >  qbv0.txt
tsntool> qbvset --device eth5 --entryfile ./qbv0.txt

#send one broadcast frame to eth4 on TestCenter.
ethtool -S eth5
#Should get one frame from eth5 on TestCenter.
```

Basetime test

```
#Get current second time:
devmem 0x1fc0900c4

tsntool> qbvset --device eth5 --entryfile ./qbv0.txt  --basetime  0x205100000000
# ’basetime’  :  (current_second_time+offset)<<32 + current_nsecond_time

#send one broadcast frame to eth4 on TestCenter.
#Frame could not pass eth5 until time offset.
```

Gate time limitation

```
echo  “t0  00000000b  20000”  >  qbv0.txt
echo  “t1  11111111b  100”  >>  qbv0.txt
tsntool> qbvset --device eth5 --entryfile ./qbv0.txt
tsntool> qbuset --device eth5 --preemptable 0xff

#send one broadcast frame(frame size = 1500) to eth4 on TestCenter.
#Frame could not pass eth5.
# We can get one fragment frame(frame size less than 1500) from eth5 on TestCenter.
```

Qbv performance test

```

cat > qbv5.txt << EOF
t0  11111111b       1000000
t1  00000000b      1000000
EOF

qbvset --device eth5 --entryfile qbv5.txt

#send 1G rate stream to eth4 on TestCenter.
#The stream would get about half line rate from eth5.
```

4.2.13.4.4 Qbu test

The following figure shows the Qbu test setup.

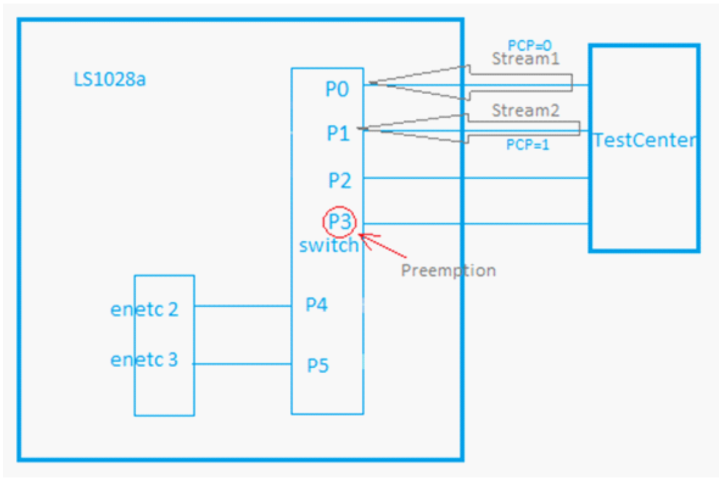
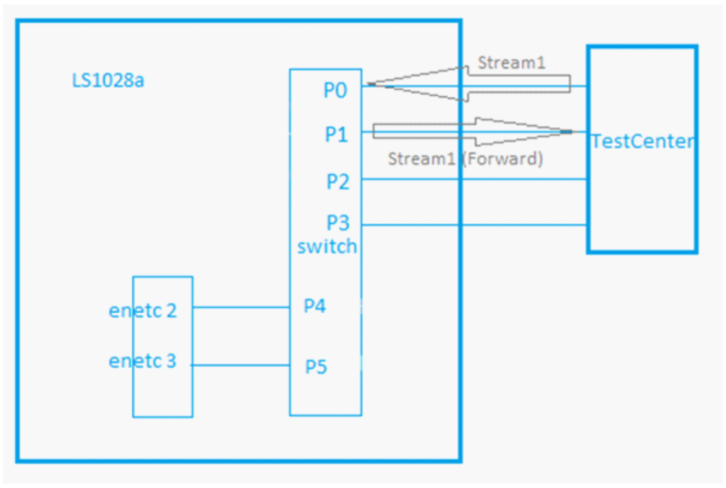


Figure 15. Qbu test setup

```

# map PCP value of VLan to queues on port 0
tsntool>pcpmap -device eth4 -enable

# map PCP value of VLan to queues on port 1
tsntool>pcpmap -device eth5 -enable

#set queue 1 to be preemptable
tsntool> qbuset --device eth7 --preemptable 0x02

# Send two streams from TestCenter, then check the number of additional mPackets transmitted by PMAC:
devmem 0x1fc010e48 32  0x3  &&  devmem 0x1fc010280
```

4.2.13.4.5 Qci test cases

The following figure shows the Qci test setup.



Stream identify

```
# Set a stream to eth4 on TestCenter.
# Edit the stream, set destination MAC : 00:01:83:fe:12:01, Vlan ID : 1

tsntool> cbstreamidset --device eth5 --nullstreamid --nulldmac 0x000183fe1201 --nullvid 1 --
streamhandle 1

tsntool> qcisfiset --device eth4 --streamhandle 1 --index 1 --gateid 1 --priority 0 --flowmeterid 68

# Send one frame.
ethtool -S eth5
ethtool -S eth6
# Only eth5 can get the frame.
```

Stream gate control

```
echo "t0      1b      3       50000       200" > sgi.txt

tsntool> qcisgiset --device eth4 --enable --index 1 --initgate 1  --initipv 0 --gatelistfile sgi.txt --
basetime 0x0
#Explain:
# 'index'    :    gateid
# 'basetime'    :    The same as Qbv set

# Send one frame on TestCenter.
ethtool -S eth5
# Frame could pass, and green_prio_3 has increased.

echo "t0      0b      3       50000       200" > sgi.txt

tsntool> qcisgiset --device eth4 --enable --index 1 --initgate 1  --initipv 0 --gatelistfile sgi.txt --
basetime 0x0

# Send one frame on TestCenter.
ethtool -S eth5
# Frame could not pass.
```

SFI maxSDU test

```
tsntool> qcisfiset --device eth4 --streamhandle 1 --index 1 --gateid 1 --priority 0 --flowmeterid 68
--maxsdu 200

# Send one frame(frame size > 200) on TestCenter.
ethtool -S eth5
# Frame could not pass.
```

FMI test

```
tsntool>qcifmiset --device eth4 --index 68  --cir 100000 --cbs 4000 --ebs 4000 --eir 100000

# Send one stream (rate = 100M) on TestCenter.
ethtool -S eth4
# All frames pass and get all green frames.

# Send one stream (rate = 200M) on TestCenter.
ethtool -S eth4
# All frames pass and get green and yellow frames.

# Send one stream (rate = 300M) on TestCenter.
ethtool -S eth4
# Not all frames could pass and get green, yellow and red frames.

# map CFI value of VLan to dp value on port 0 to recognize yellow frames.
tsntool>pcpmap –device eth4 –enable

# Send one yellow stream (rate = 100M) on TestCenter.
ethtool -S eth4
# All frames pass and get all yellow frames.

# Send one yellow stream (rate = 200M) on TestCenter.
ethtool -S eth4
# Not all frames could pass and get yellow and red frames.

# Test cf mode.
tsntool> qcifmiset --device eth4 --index 68  --cir 100000 --cbs 4000 --ebs 4000 --eir 100000 --cf

# Send one yellow stream (rate = 200M) on TestCenter.
ethtool -S eth4
# All frames pass and get all yellow frames. (use CIR as well as EIR)

# Send one yellow stream (rate = 300M) on TestCenter.
ethtool -S eth4
# Not all frames could pass and get yellow and red frames.
```

4.2.13.4.6 Qav test case

The following figure shows Qav test case setup.

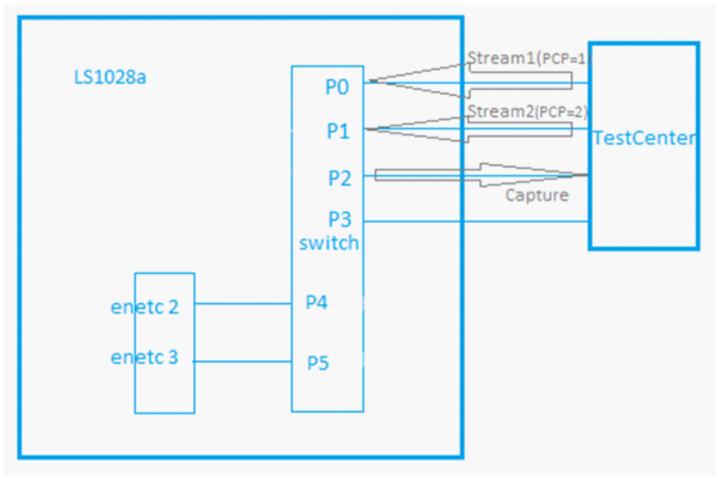


Figure 17. Qav test case setup

```
# map PCP value of VLan to queues on port 0
tsntool>pcpmap -device eth4 -enable

# map PCP value of VLan to queues on port 1
tsntool>pcpmap -device eth5 -enable

tsntool> cbsset --device eth6 --tc 1 --percentage 20
tsntool> cbsset --device eth6 --tc 2 --percentage 40
#Explain:
# ' percentage '  : 40 means bandwidth limited of queue is Max_bandwidth*40%, 400M.
# Send two streams from Test center, then check the frames count.
ethtool -S eth6
# Frames of queue 1 is half of queue2.
# Note: Stream rate must lager than bandwidth limited of queue.

# Capture frames on eth6 on TestCenter.
# Get Frame sequence is:
(PCP=1), (PCP=2), (PCP=2), (PCP=1), (PCP=2), (PCP=2),…
```

4.2.13.4.7 Seamless redundancy test case

The following figure shows the Seamless redundancy test case setup.

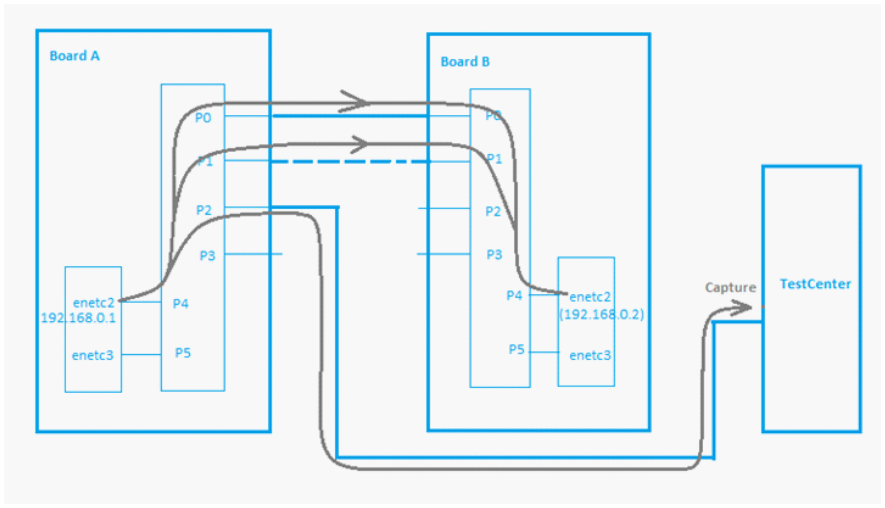


Figure 18. Seamless redundancy test case

Sequence generator test

```
# Get MAC address of eth2: 7E:A8:8C:9B:41:DD

# On board A:
ifconfig eth2 192.168.0.1 up
ping 192.168.0.2
# Network is connected.

# On board A:
tsntool> cbstreamidset --device eth4 -streamhandle 1 --nulldmac 0x7EA88C9B41DD --nullvid 1

tsntool> cbgen --device eth4 --index 1 --iport_mask 0x3f --split_mask 0x07 --seq_len 16 --seq_num 2048

# Capture frames on eth6 on TestCenter.
ping 192.168.0.2
# We can get frames from eth6 on TestCenter, each frame add the sequence number: 23450801, 23450802,
23450803…

# On board B:
tcpdump -i eth2 -w eth2.pcap
# We also can get a copy frames from eth2 on board B, which is transmit through eth4 of board A.
```

Sequence recover test

```
# On board B:
ifconfig eth2 192.168.0.2 up
ifconfig eth2
# On board B:
tsntool> cbstreamidset --device eth8 -streamhandle 1 --nulldmac 0x7EA88C9B41DD --nullvid 1

tsntool> cbrec --device eth4 --index 1 --seq_len 16 --his_len 31 --rtag_pop_en

tcpdump -i eth2 -w eth2.pcap

# On board A:
ping 192.168.0.2
```

```
# Then on board B, we can get a frame without sequence tags from board A.


# Connect eth5 of board A with eth5 of board B.
# On board B:
tsntool> cbrec --device eth5 --index 1 --seq_len 16 --his_len 31 --rtag_pop_en
tcpdump -i eth2 -w eth2.pcap

# On board A:
ping 192.168.0.2

# On board B, we can get only one frame without sequence tag, another frame from eth4 or eth5 is dropped.
```

4.2.14 Multimedia

4.2.14.1 DisplayPort and LCD controller

Description

This section describes how to configure and test LCD controller and eDP controller drivers for the LS1028ARDB and LS1028AQDS boards.

RCW configuration

The following table describes RCW for LCD controller on the LS1028ARDB and LS1028AQDS boards.

Board	RCW
LS1028ARDB	HWA_CGA_M3_CLK_SEL = 2
LS1028AQDS	HWA_CGA_M3_CLK_SEL = 2

Kernel configure options tree view

The following table describes the tree view of the kernel configuration options.

Options	Description

`Device Drivers --->`
`<*> Graphics support  --->`
`    [*] ARM Mali Display Processor`
`    [*] DRM Support for Freescale i.mx`
`    [*] IMX8 HD Display Controller`
`    [*] Bootup logo--->`
`    [*] Standard 224-color Linux logo`
	Enable LCD controller driver, DRM driver and HD display (eDP) controller driver.
	Choose 224 color Linux logo.

Device tree configuration

The following is default device tree configuration for LS1028A RDB and QDS board:

```
&edp {
        fsl,no_edid;
```

```
          resolution = "3840x2160@60",
                       "1920x1080@60",
                       "1280x720@60",
                       "720x480@60";
          lane_mapping = <0x4e>;
          edp_link_rate = <0x6>;
          edp_num_lanes = <0x4>;
          status = "okay";
 };
```

If there is no edid supported by display panel, “fsl, no_edid” is needed to specify no edid mode is used. For this case, “resolution” property is used to specify which resolutions can be supported by the display panel. Currently, the kernel driver has build in display configuration for the above four resolutions. You can remove any resolution if not supported by the panel. If you want to add the other resolution support by no edid mode, display parameter should be added in kernel driver of DisplayPort.

In case the edid data read from display panel or display monitor is used for display configuration, remove the “fsl,no_edid” and “resolution” property.

Source files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
Drivers/gpu/drm/arm/malidp_*	LCD controller driver source
Drivers/gpu/drm/imx/	eDP controller driver source

Verification in Linux

DisplayPort supports four resolutions: 480p (720x480p60), 720p (1280x720p60), 1080p (1920x1080p60) and 4k (3840x2160p60). Follow the below procedure to provide support for 480p, 720p, 1080p, or 4k resolution.

1. Downloading and extracting DP firmware.

   You can either download the DP firmware package **ls1028a-dp-fw.bin** from flexbuild or you can find it in BSP release binary packages. The binary **ls1028a-dp-fw.bin** is a self-extractable archive, and you must accept EULA first before file extraction. The following procedure shows how to extract DP firmware in Linux host machine:

```

$ chmod +x ls1028a-dp-fw.bin

$ ./ls1028a-dp-fw.bin
Welcome to NXP ls1028a-dp-fw.bin

You need to read and accept the EULA before continue..

LA_OPT_BASE_LICENSE v26 June 2018



IMPORTANT.  Read the following NXP Software License Agreement ("Agreement")
completely.    By selecting the "I Accept" button at the end of this page, you
indicate that you accept the terms of the Agreement and you acknowledge that
you have the authority, for yourself or on behalf of your company, to bind your
company to these terms.  You may then download or install the file.

…
```

```
Do you accept the EULA you just read? (y/N) y
EULA has been accepted. The files will be unpacked at 'ls1028a-dp-fw'

Unpacking file ...... done
```

The DP firmware **mhdp_fw_x_x_xx-dptx-hdcp-mcu2.bin** can be found in the directory **ls1028a-dp-fw** along with the copyright files.

2. Loading DP firmware in u-boot.

   HDP firmware binary is loaded during U-Boot. At U-Boot prompt, copy the firmware binary from any storage medium (NOR flash or SD/eMMC) to DDR memory.

   Use the following command to load the binary:

   ```

   => hdp load <address > <offset>
   ```

   where:

   - address - address where the firmware binary starts in DDR memory
   - offset - IRAM offset in the firmware binary (8192 default)

   For example:

   ```
   hdp load 0x98000000 0x2000
   ```

   If flash images are built by flex-builder, DP firmware will be burn in Flash or SD card. To know about the location of DP firmware, see LS1028A BSP Memory Layout. Use the following commands to load DP firmware at U-Boot:

   - Get DP firmware with XSPI flash boot

     ```

     =>  sf probe; sf read 0x98000000 0x900000 0x40000
     =>  hdp load 0x20900000 0x2000
     Trying load HDP firmware from flexspi..
     Loading hdp firmware from 0x0000000020900000 offset 0x0000000000002000
     Loading hdp firmware Complete
     ```

   - Get DP firmware with SD or eMMC boot

     ```

     =>  mmc info; mmc read 0xa0000000 0x4800 0x200;  hdp load 0xa0000000 0x2000
     Trying load HDP firmware from SD..
     Device: FSL_SDHC
     Manufacturer ID: 3
     OEM: 5054
     Name: SL16G
     Bus Speed: 50000000
     Mode : SD High Speed (50MHz)
     Rd Block Len: 512
     SD version 3.0
     High Capacity: Yes
     Capacity: 14.5 GiB
     Bus Width: 4-bit
     Erase Group Size: 512 Bytes

     MMC read: dev # 0, block # 18432, count 512 ... 512 blocks read: OK
     Loading hdp firmware from 0x00000000a0000000 offset 0x0000000000002000
     Loading hdp firmware Complete
     ```

3. Add bootargs as per the resolution.
   - To support 480p resolution, add the following argument to bootargs (the minimum CMA size is 64M bytes):

     ```
     video=720x480-32@60 cma=256M
     ```

   - To support 720p resolution, add the following argument to bootargs:

     ```
     video=1280x720-32@60 cma=256M
     ```

   - To support 1080p resolution, add the following argument to bootargs:

     ```
     video=1920x1080-32@60 cma=256M
     ```

   - To support 4k resolution, add the following arguments to bootargs:

     ```
     video=3840x2160-32@60 cma=256M
     ```

4. Boot up the kernel, you can see penguins on the DP monitor.

   Refer to the following kernel boot up log:

   ```
   [    2.480536] [drm] found ARM Mali-DP500 version r1p2
   [    2.485571] i.mx8-hdp f1f0000.phy: lane_mapping 0x4e
   [    2.490561] i.mx8-hdp f1f0000.phy: edp_link_rate 0x06
   [    2.495631] i.mx8-hdp f1f0000.phy: dp_num_lanes 0x04
   [    2.500778] [drm] Started firmware!
   [    2.504281] [drm] CDN_API_CheckAlive returned ret = 0
   [    2.509354] [drm] Firmware version: 23029, Lib version: 20691
   [    2.515145] [drm] CDN_API_MainControl_blocking (ret = 0 resp = 1)
   [    2.521298] [drm] CDN_API_General_Test_Echo_Ext_blocking (ret = 0 echo_resp = echo test)
   [    2.529429] [drm] CDN_API_General_Write_Register_blockin ... setting LANES_CONFIG
   [    2.537001] [drm] pixel engine reset
   [    2.540599] [drm] CDN_*_Write_Register_blocking ... setting LANES_CONFIG 4e
   [    2.549371] [drm] AFE_init
   [    2.552095] [drm] deasserted reset
   [    2.555585] Wait for A2 ACK
   [    2.579947] [drm] AFE_power exit
   [    2.583184] [drm] CDN_API_DPTX_SetVideo_blocking (ret = 0)
   [    2.588772] mali-dp f080000.display: bound f1f0000.phy (ops imx_hdp_imx_ops)
   [    2.595960] [drm] Supports vblank timestamp caching Rev 2 (21.10.2013).
   [    2.602602] [drm] No driver support for vblank timestamp query.
   [    2.620556] [drm] pixel engine reset
   [    2.620569] [drm] CDN_*_Write_Register_blocking ... setting LANES_CONFIG 4e
   [    2.622354] [drm] AFE_init
   [    2.622370] [drm] deasserted reset
   [    2.622459] Wait for A2 ACK
   [    2.644648] [drm] AFE_power exit
   [    2.644654] [drm] CDN_API_DPTX_SetVideo_blocking (ret = 0)
   [    2.644675] [drm] CDN_API_DPTX_SetHostCap_blocking (ret = 0)
   [    2.647306] [drm] INFO: Full link training started
   [    2.649727] [drm] INFO: Clock recovery phase finished
   [    2.650540] [drm] INFO: Channel equalization phase finished
   [    2.650541] [drm] (last part meaning training finished)
   [    2.650573] [drm] INFO: Get Read Link Status (ret = 0) resp: rate: 20,
   [    2.650575] [drm] lanes: 4, vswing 0..3: 2 2 2, preemp 0..3: 2 1 1
   [    2.650761] [drm] CDN_API_DPTX_Set_VIC_blocking (ret = 0)
   [    2.650766] [drm] CDN_API_DPTX_SetVideo_blocking (ret = 0)
   [    2.698031] Console: switching to colour frame buffer device 480x135
   ```

```
[    2.791774] [drm] pixel engine reset
[    2.791786] [drm] CDN_*_Write_Register_blocking ... setting LANES_CONFIG 4e
[    2.793593] [drm] AFE_init
[    2.793606] [drm] deasserted reset
[    2.793693] Wait for A2 ACK
[    2.815577] [drm] AFE_power exit
[    2.815583] [drm] CDN_API_DPTX_SetVideo_blocking (ret = 0)
[    2.815603] [drm] CDN_API_DPTX_SetHostCap_blocking (ret = 0)
[    2.818239] [drm] INFO: Full link training started
[    2.820665] [drm] INFO: Clock recovery phase finished
[    2.821478] [drm] INFO: Channel equalization phase finished
[    2.821479] [drm] (last part meaning training finished)
[    2.821511] [drm] INFO: Get Read Link Status (ret = 0) resp: rate: 20,
[    2.821513] [drm] lanes: 4, vswing 0..3: 2 2 2, preemp 0..3: 2 1 1
[    2.821698] [drm] CDN_API_DPTX_Set_VIC_blocking (ret = 0)
[    2.821704] [drm] CDN_API_DPTX_SetVideo_blocking (ret = 0)
[    2.822714] [drm] HDMI/DP Cable Plug In
[    2.897464] mali-dp f080000.display: fb0:  frame buffer device
[    2.903577] [drm] Initialized mali-dp 1.0.0 20160106 for f080000.display on minor 0
```

4.2.14.2 Graphics processing unit (GPU)

Description

The GPU driver supports NXP Graphics Processing Unit (GPU).

Module loading

GPU driver supports either kernel built-in or in module.

Kernel Configure Tree View Options	Description

`Device Drivers--->`
`<*> MXC Vivante GPU support`
	Enable GPU module

Compile-time configuration options

Option	Values	Default Value	Description
CONFIG_MXC_GPU_VIV=y	y/m/n	y	Enables GPU controller

Source files

The GPU driver source is maintained in the Linux kernel source tree.

Source File	Description
drivers/mxc/*	GPU module support

Test procedure

Follow the below procedure to use GPU.

1. Boot up the kernel. The messages appear as follows:

   ```
   ...
   Galcore version 6.2.4.163672
   ...
   ```

   You will see GPU device filename as follows:

   ```
   root@localhost:~# ll /dev/galcore
   crw------- 1 root root 199, 0 Jan 28  2018 /dev/galcore
   ```

2. All the test cases are located under the **/opt** directory:

   ```
   root@localhost:/opt# ls /opt
   cl11  es20  tiger  vdk
   ```

   a. OpenCL demo

      Below is FFT as example:

      ```
      root@ localhost: cd /opt/cl11/fft
      root@localhost:/opt/cl11/fft# ./fft 128
      Block size: 16
      Print result: yes
      Initializing device(s)...
      Get the Device info and select Device...
      # of Devices Available = 1
      # of Compute Units = 1
      # compute units = 1
      Creating Command Queue...
      log2(fft size) = log2(128)=7
      Compiling  radix-2 FFT Program for GPU...
      creating radix-2 kernels...
      Creating kernel fft_radix2 0 (p=1)...
      Creating kernel fft_radix2 1 (p=2)...
      Creating kernel fft_radix2 2 (p=4)...
      Creating kernel fft_radix2 3 (p=8)...
      Creating kernel fft_radix2 4 (p=16)...
      Creating kernel fft_radix2 5 (p=32)...
      Creating kernel fft_radix2 6 (p=64)...
      Setting kernel args for kernel 0 (p=1)...
      Setting kernel args for kernel 1 (p=2)...
      Setting kernel args for kernel 2 (p=4)...
      Setting kernel args for kernel 3 (p=8)...
      Setting kernel args for kernel 4 (p=16)...
      Setting kernel args for kernel 5 (p=32)...
      Setting kernel args for kernel 6 (p=64)...
      running kernel 0 (p=1)...
      running kernel 1 (p=2)...
      running kernel 2 (p=4)...
      running kernel 3 (p=8)...
      running kernel 4 (p=16)...
      running kernel 5 (p=32)...
      running kernel 6 (p=64)...
      Kernel execution time on GPU (kernel 0) :   0.000342 seconds
      Kernel execution time on GPU (kernel 1) :   0.000215 seconds
      ```

```
Kernel execution time on GPU (kernel 2) :   0.000002 seconds
Kernel execution time on GPU (kernel 3) :   0.000600 seconds
Kernel execution time on GPU (kernel 4) :   0.000655 seconds
Kernel execution time on GPU (kernel 5) :   0.000821 seconds
Kernel execution time on GPU (kernel 6) :   0.000868 seconds
Total Kernel execution time on GPU :   0.003503 seconds
Successful.
```

b. OpenGLES demo

```
root@localhost:~# weston --tty=1 &
root@ localhost: cd /opt/es20/vv_launcher
root@localhost:/opt/es20/vv_launcher # ./vv_launcher
```

c. OpenVG demo

```
root@localhost:~# weston --tty=1 &
root@localhost:~# cd /opt/tiger/
root@localhost:/opt/tiger# ./tiger
```

d. OpenVDK demo

```
root@localhost:~# weston --tty=1 &
root@localhost:~# cd /opt/vdk
root@localhost:/opt/vdk# ./tutorial1
```

4.2.14.3 Synchronous audio interface (SAI)

Description

This section describes how to configure and test SAI audio driver for the LS1028ARDB and LS1028AQDS boards. The integrated I2S module is NXP's Synchronous Audio Interface (SAI). The codec is SGTL5000 stereo audio codec.

RCW configuration

The following table describes RCW for audio on the LS1028ARDB and LS1028AQDS boards.

Board	RCW
LS1028ARDB	rcw_1300_audio.rcw, EC1_SAI4_5_PMUX = 2
LS1028AQDS	rcw_1300_audio.rcw, SDHC1_BASE_PMUX = 3

Kernel configure options tree view

The following table describes the tree view of the kernel configuration options.

Options	Description

Table continues on the next page...

Table continued from the previous page...

<table>
<tr><td>

```
Device Drivers --->
<*> I2C support  --->
    [*] Enable compatibility bits for old user-space
    [*] I2C device interface
    [*] I2C bus multiplexing support
        Multiplexer I2C Chip support  --->
        <*> Philips PCA954x I2C Mux/switches
    [*] Autoselect pertinent helper modules
    I2C Hardware Bus support  --->
        <*> IMX I2C interface

<*> Voltage and Current Regulator Support --->
    [*] Regulator debug support
    [*] Fixed voltage regulator support

<*> Sound card support
    <*> Advanced Linux Sound Architecture ->
        [*] OSS PCM (digital audio) API
        [*]     OSS PCM (digital audio) API - Include
plugin system
        [*] Support old ALSA API
        [*] Verbose procfs contents
            ALSA for SoC audio support  --->
            SoC Audio for Freescale CPUs  --->
          <*> Synchronous Audio Interface (SAI) module
support
            CODEC drivers --->
            <*> Freescale SGTL5000 CODEC
            <*> ASoC Simple sound card
support
<*> DMA Engine support   --->
    <*> Freescale eDMA engine support support
```

</td><td>Enable ALSA SoC driver, I2C driver and EDMA driver.</td></tr>
</table>

Identifier

The following table describes the configure identifiers that are used in kernel source code and default configuration files.

Option	Values	Default Value	Description
CONFIG_I2C_IMX	y/m/n	y	I2C driver needed for configuring SGTL5000
CONFIG_SOUND	y/m/n	y	Enable sound card support
CONFIG_SND	y/m/n	y	Enable advanced Linux sound architecture support
CONFIG_SND_PCM_OSS	y/m/n	y	Enable OSS digital audio
CONFIG_SND_PCM_OSS_PLUGINS	y/m/n	y	Support conversion of channels, formats and rates
CONFIG_SND_SUPPORT_OLD_API	y/m/n	y	Enable support old ALSA API

Table continues on the next page...

Table continued from the previous page...

Option	Values	Default Value	Description
CONFIG_SND_SOC_FSL_SAI	y/m/n	y	Enable SAI module support
CONFIG_SND_SOC_GENERIC_DMAENGINE_PCM	y/m/n	y	Enable generic DMA engine for PCM
CONFIG_SND_SIMPLE_CARD	y/m/n	y	Enable generic simple sound card support
CONFIG_SND_SOC_SGTL5000	y/m/n	y	Enable codec SGTL5000 support
CONFIG_FSL_EMDA	y/m/n	y	Enable eDMA engine support

Source files

The driver source is maintained in the Linux kernel source tree.

Source File	Description
sound/soc/fsl	ALSA SoC driver source

Verification in Linux

Follow the below procedure for the verification.

1. The kernel boot process displays the following messages:

```
sgtl5000 2-000a: sgtl5000 revision 0x11
sgtl5000 2-000a: Using internal LDO instead of VDDD
......
asoc-simple-card sound:  sgtl5000 <-> f130000.sai mapping ok
......
ALSA device list:
  #0: f130000.sai-sgtl5000
```

2. On the LS1028ARDB board:
    a. Set the switch SW5[8] = ON.
    b. To configure BRDCFG3[2] = 1, run the following command at U-Boot prompt.

    ```
    i2c mw 0x66 0x53 0x4
    ```

    c. The lineout interface is J34.
3. On the LS1028AQDS board:
    a. Set the switch SW6[1] = OFF and switch SW6[2] = ON.
    b. Connect the jumpers for J59.

   c. To configure BRDCFG11[7:6] = 0b10 and BRDCFG4[0] = 0b1, run the following commands at U-Boot prompt.

      ```
      i2c mw 0x66 0x54 0x1;
      i2c mw 0x66 0x5b 0x93
      ```

   d. The lineout interface is J57.

4. Run the following `aplay` command to test playback.

   ```
   aplay -f S16_LE -r 44100 -t wav -c 2 44k-16bit-stereo.wav
   ```

5. Use alsamixer to adjust the volume for playing by the option “PCM”.

Date of release: 04/2019

Document identifier: LS1028ABSPUG