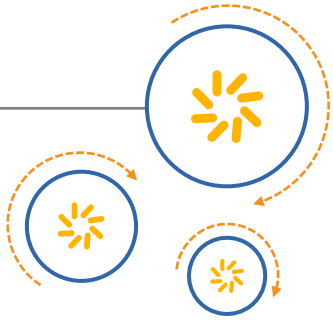
Qualcomm Technologies, Inc.



IPQ40x8 and IPQ40x9 Software

Configuration Guide

80-Y9571-4 Rev. L

December 7, 2016

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	July 2015	Initial release
B	October 2015	Numerous updates to the document for the IPQ40x8 release. Read the document in its entirety.
C	October 2015	Added DK03 Support in section 4.3. Updated section 1.3.
D	November 2015	Updated sections 2.3, 3, and 1.3.
E	January 2016	Added the sections 4.2 and 3.5.
F	April 2016	Added section 3.6 and updated the section 4.3.
G	May 2016	Updated section 1.3.
H	June 2016	Updated Sections 3.4 and 3.6.
J	September 2016	Updated Chapter 4.3.
K	November 2016	Updated sections 6.1, 6.2, and 6.3.
L	December 2016	Added section 3.9.

Contents

Figures

1 Overview

This document describes the salient functionalities of the IPQ40x8/IPQ40x9 software. In addition, the boot process, flash memory specifications, and Linux pin control drivers of these chipsets are also described.

FCC NOTICE:

This kit is designed to allow:

(1) Product developers to evaluate electronic components, circuitry, or software associated with the kit to determine whether to incorporate such items in a finished product and

(2) Software developers to write software applications for use with the end product. This kit is not a finished product and when assembled may not be resold or otherwise marketed unless all required FCC equipment authorizations are first obtained. Operation is subject to the condition that this product not cause harmful interference to licensed radio stations and that this product accept harmful interference. Unless the assembled kit is designed to operate under part 15, part 18 or part 95 of the FCC's rules, the operator of the kit must operate under the authority of an FCC license holder or must secure an experimental authorization under part 5 of the FCC's rules.

1.1 Acronyms

The following terms and acronyms are used in the discussion of this document:

EDMA	Ethernet Direct Memory Access
ESS	Ethernet Switch Subsystem
GPIO	General-Purpose I/O
HLOS	High-Level Operating System
IRQ	Interrupt Request
WSS	Wi-Fi Sub System
PBL	Primary Boot Loader
QRFS	Qualcomm Receiving Flow Steering
RFS	Receiving Flow Steering
SBL	Secondary Boot Loader
TLMM	Top-Level Mode Multiplexer

1.2 Architecture

Figure 1-1 shows the IPQ40x8/IPQ40x9 hardware architecture.

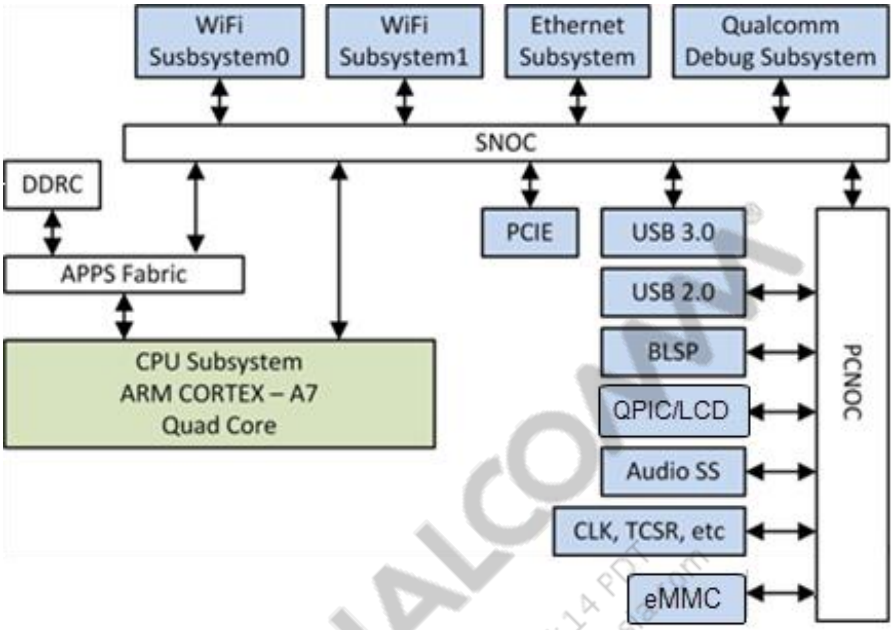


Figure 1-1 IPQ40x8/IPQ40x9 hardware architecture

The following is the CPU specification for IPQ40x8 and IPQ40x9 chipsets:

Quad Cortex-A7 application processors	Qualcomm supplies a reference networking application platform consisting of Linux with OpenWrt. Configuration files tailor the networking software to suit targeted applications. Core 0 is the first CPU to power up.
Dual Tensilica (WSS)	Responsible for Wi-Fi connectivity.

1.3 System features

The salient system features of the IPQ40x8 and IPQ40x9 chipsets are trusted computing and power management. Implementation of these features is distributed throughout the hardware and software.

The following power management techniques are used in IPQ40x8 and IPQ40x9:

ARM A7 CPU frequency scaling	▪ CPU frequency can be scaled up and down based on the CPU load. ▪ Performance governor is turned **on** during the boot process ▪ Supported CPU Frequencies for this release are: ▫ 716 MHz ▫ 500 MHz ▫ 200 MHz ▫ 48 MHz

Dynamic clock gating is enabled	Software detects the inactivity in parts of the system and shuts down the clock to these parts accordingly
DDR and NOC	DDR runs at 672 MHz (BGA) and 537 MHz (QFN) SNOC at 200 MHz PCNOC at 100 MHz Frequency scaling is not supported for DDR and NOC

Use these commands to switch governor and CPU frequency:

```
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo 716000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

System frequencies

	Min	Max
Cortex-A7	48 MHz	716 MHz
DDR	672/537 MHz	672/537 MHz
SNOC	200 MHz	200 MHz
PCNOC	100 MHz	100 MHz

- Cortex-A7 frequencies are changed based on CPU load.
- DDR and NOC frequencies are always at the maximum in the system **on** state.

1.4 Reference software platform (QSDK)

The Qualcomm® Software Development Kit (QSDK) for IPQ40x8 and IPQ40x9 comprises the following components:

- Linux kernel
- SBL
- Qualcomm Secure Execution Environment software
- U-Boot: High-level boot loader for Linux
- OpenWrt: Embedded Linux distribution reference software and framework for customizing and building a networking application from existing and value-add components
- Board configuration files for the reference designs supplied by Qualcomm
- Application profiles that tailor the OpenWrt networking software to suit various application classes
- SoC drivers
- Qualcomm Wi-Fi proprietary driver package

The IPQ40x8 includes several cryptographic engines, which are useful for accelerating a variety of cryptographic operations. The QSDK includes a choice of cryptographic APIs that help the software work with the cryptographic engines:

- Standard Linux CryptoAPI
- OpenBSD Cryptographic Framework (OCF)

1.5 Related documentation

These documents provide specialized programming information for IPQ40x8/IPQ40x9.

Doc number	Title
80-Y9347-18	*IPQ4018 Access Point SoC Device Specification*
80-Y9347-28	*IPQ4028 Access Point SoC Device Specification*
80-Y9348-28	*IPQ4018/IPQ4028 Device Revision Guide*
80-Y6399-3	*IPQ4018/IPQ4019/IPQ4028/IPQ4029 QSDK Setup And User Guide*
80-Y9571-5	*Lauterbach Debugger for IPQ4018/IPQ4019/IPQ4028/IPQ4029 Application Note*
80-Y9571-6	*IPQ4018/IPQ4028/IPQ4019/IPQ4029 Performance Management Application Note*
80-Y9571-7	*IPQ4018/IPQ4028/IPQ4019/IPQ4029 Home Switch Software Development Kit User Guide*
80-Y9571-8	*IPQ4018/IPQ4028/IPQ4019/IPQ4029 SOHO Switch Software Development Kit Reference Manual*
80-Y9571-9	*IPQ4018/IPQ4028/IPQ4019/IPQ4029 Switch Software Development Kit Diagnostic Shell User Guide*
80-Y9571-10	*IPQ4018/IPQ4028/IPQ4019/IPQ4029 SOHO Switch UCI Command User Guide*
80-Y9572-1	*IPQ4019.ILQ.1.1 AND IPQ4019.ILQ.1.1.1 Firmware Build Guide*
80-Y9700-1	*IPQ4018/IPQ4028 AP.DK01 Hardware Reference Guide*
80-Y9700-2	*IPQ4018/IPQ4028 AP.DK01 Setup Guide*
80-Y9700-3	*IPQ4018/IPQ4028/IPQ4019/IPQ4029 RF Test User Guide*

2 Boot process

This chapter describes the boot operation workflow (see Figure 2-1) and the different boot loader components. On the IPQ40x8 and IPQ40x9 chipsets, the processor receives power and starts running on Cortex-A7. Cortex-A7 initiates a chain-of-trust sequence of events that file authentication to ensure that the intended and trusted firmware and software components are loaded. The chain-of-trust continues till the software is loaded into DDR.

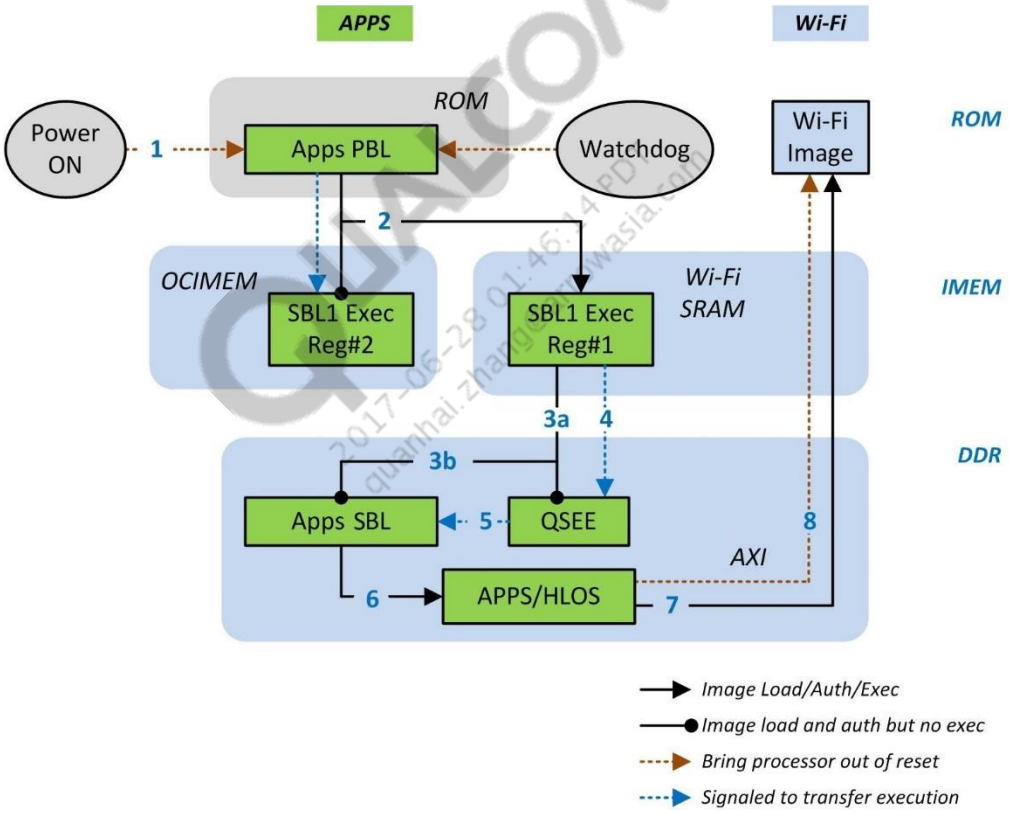


Figure 2-1 Boot flow

2.1 PBL

Primary boot loader (PBL) is burnt into the boot ROM of the Cortex-A7 processor at the factory. Its functions are:

- Hardware initialization
- Boot device detection – eMMC/NOR/NAND
- Load SBL from the first block of the detected boot device into system WSS IMEM and OC IMEM
- Authenticate SBL image
- Transfer control to SBL

2.2 SBL

Secondary boot loader (SBL) performs boot-specific initialization before transferring control to the application bootloader. It runs on a single core of the Cortex-A7 processor. Its functions are:

- Initializing DDR memory using DDR parameters read from flash.
- Loading the QSEE image; can read the QSEE image from NOR/NAND/eMMC to PCDDR3.
- Loading APPSBL; can read the SBL image from NOR/NAND/eMMC to PCDDR3.
- Handing control to APPSBL.

2.3 QSEE

Qualcomm Secure Extension Environment (QSEE) software runs in privilege mode of Trustzone. Its functions are:

- On a reset (cold boot), performing security configuration of the IPQ40x8/IPQ40x9 SoC.
- Brings the secondary cores out of reset.
- Performs context dump collection for crash dumps.
- Offers runtime services to HLOS via the SCM interface.

2.4 APPSBL

Applications boot loader (APPSBL) is U-Boot. Its primary functions are:

- Loading the kernel into PCDDR3 memory and execute
- Passing boot arguments to the kernel

U-Boot also enables reflashing of the kernel and the file system. It can access these interfaces:

SPI-NOR flash	Store and retrieve kernel, fs, etc.
NAND flash	Store and retrieve kernel, fs, etc.
eMMC flash	Store and retrieve kernel, fs, etc.
Ethernet	Upgrade the SBL, QSEE, U-Boot, kernel, and file system via TFTP.

3 Flash memory

The following sections describe the manner in which the IPQ40x8/IPQ40x9 flash memory is organized. This information is useful during the loading of images into flash memory, and booting the device using the image from flash memory.

3.1 NOR flash layout (Premium – 32 MBytes)

Partition	Size In KBytes	Comments
SBL1	256	Secondary boot loader
MIBIB	128	Partition table
QSEE	384	TZ
CDT	64	Platform ID and DDR configuration
DDRPARAMS	64	TBD
APPSBLENV	64	U-Boot ENV variables
APPSBL	512	U-Boot
ART	64	Wi-Fi calibration data
HLOS	4096	Kernel + DTB
Rootfs	22528	File system

3.2 NOR flash layout (Standard – 16 MBytes)

Partition	Size In KBytes	Comments
SBL1	256	Secondary boot loader
MIBIB	128	Partition table
QSEE	384	TZ
CDT	64	Platform ID and DDR configuration
DDRPARAMS	64	TBD
APPSBLENV	64	U-Boot ENV Variables
APPSBL	512	U-Boot
ART	64	Wi-Fi calibration data
HLOS	4096	Kernel + DTB
Rootfs	10752	File system

3.3 NOR and NAND flash layout

NOR			NAND		
Partition	***Size In KBytes***	***Comments***	***Partition***	***Size In KBytes***	***Comments***
SBL1	256	Secondary boot loader	rootfs	65536	Ubinized Kernel + DTB + File system
MIBIB	128	Partition table			
QSEE	384	TZ			
CDT	64	Platform ID and DDR configuration			
DDRPARAMS	64	TBD			
APPSBLENV	64	U-Boot ENV variables			
APPSBL	512	U-Boot			
ART	64	Wi-Fi calibration data			

3.4 NAND flash layout

Partition	Size In KBytes	Comments
SBL1	1024	Secondary boot loader
MIBIB	1024	Partition table
BOOTCONFIG	1024	Contains failsafe partition information
QSEE	1024	TZ
QSEE_1	1024	Alternate partition
CDT	512	Platform ID and DDR configuration
CDT_1	512	Alternate partition
BOOTCONFIG1	512	Contains failsafe partition information
APPSBLENV	512	U-Boot ENV variables
APPSBL	2048	U-Boot
APPSBL_1	2048	Alternate partition
ART	512	Wi-Fi calibration data
Rootfs	66560	Ubinized Kernel + DTB + File system
rootfs_1	52224	Alternate partition

NOTE: When system upgrade is done, the alternate partitions become the active partitions and vice versa. Each time system upgrade is done, alternate and active partitions get switched.

NOTE: Size of rootfs_1 partition has been made lesser than rootfs in order to retain the existing flash partition layout since failsafe feature was added later.

3.5 NAND flash page layout

This section describes the diagrammatic representation of NAND flash structure.

Blocks

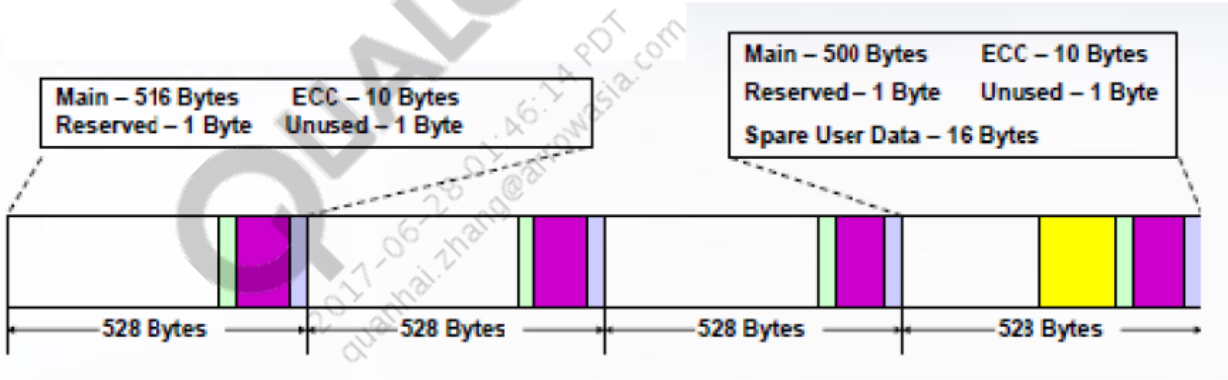
A block is the smallest erasable unit of a NAND flash. Blocks are further broken up into units called pages.

Pages

A page is the smallest programmable unit of a NAND flash. Pages are further broken up into units called main data and spare data.

The NAND controller, which is part of the Qualcomm chipset, has a built-in ECC engine which performs the ECC detection and correction on pages that are read. It calculates the ECC for a page that is to be written and stores it in the ECC area of the page's spare region. The 4/8 bit BCH ECC algorithm is the ECC algorithm currently being used.

The page layout used is shown below. Flash layout is uniform across all SBL components upto Linux kernel.



Bad block management

Bad blocks contain one or more invalid bits whose reliability is not guaranteed. Bad blocks may be present when the device is shipped, or may develop during the lifetime of the device.

IPQ4016 uses the scheme "Skip block method" for managing bad blocks. The algorithm starts by reading the spare area of the first page of each block and checks if the first or the second byte is not 0xff. Based on this, bad block table is created. This is done at every boot. The data is sequentially programmed page-by-page into the target flash device. When the target address corresponds to an invalid block address, that block is skipped and the data is stored in the next good block. Since this invalid block is skipped, the data in the spare array indicating the presence of the invalid block is kept. Hence the user's system can build a table of invalid blocks addresses by reading the spare area of all the blocks at boot time. Because the invalid block exists, the user's data should not exceed the size of good blocks.

A graphical representation of bad block scheme is shown below:

Device		Image
Block 0	Image 0	Block 0
1	1	1
2	2	2
Block 3 is bad	skip	3
4	3	4
5	4	5
6	5	6
Block 7 is bad	skip	7
8	6	8
9	7	9
Block 10 is bad	skip	10
11	8	11
12	9	12
13	10	13
....	...	....
n-4	n-7	n-4
n-3	n-6	n-3
n-2	n-5	n-2
n-1	n-4	n-1
n	n-3	n

Figure 3-1 Bad block scheme

3.6 eMMC flash layout

Partition	Size In KBytes	Comments
GPT	16.5	Partition table
SBL1	512	Secondary boot loader
BOOTCONFIG	512	Contains failsafe partition information
QSEE	512	TZ
QSEE_1	512	Alternate partition
CDT	256	Platform ID and DDR Configuration
CDT_1	256	Alternate partition
BOOTCONFIG1	256	Contains failsafe partition information
APPSBLENV	256	U-Boot ENV variables
APPSBL	1024	U-Boot
APPSBL_1	1024	Alternate partition
ART	256	Wi-Fi calibration data
HLOS	8192	Kernel + DTB
HLOS_1	8192	Alternate partition
rootfs	65536	File system
rootfs_1	65536	Alternate partition
rootfs_data	1310720	File system data
Backup GPT	16	Backup partition table

NOTE: When system upgrade is done, the alternate partitions become the active partitions and vice versa. Each time system upgrade is done, alternate and active partitions get switched.

3.7 NOR and eMMC flash layout

NOR			eMMC		
Partition	***Size in KBytes***	***Comments***	***Partition***	***Size in KBytes***	***Comments***
SBL1	256	Secondary Boot Loader	GPT	16.5	Partition table
MIBIB	128	Partition table	HLOS	8192	Kernel + DTB
QSEE	384	TZ	rootfs	65536	File system
CDT	64	Platform ID and DDR Configuration	rootfs_data	1310720	File system Data
DDRPARAMS	64	TBD	Backup GPT	16	Backup partition table
APPSBLENV	64	Uboot ENV Variables			
APPSBL	512	U-Boot			
ART	64	Wi-Fi Calibration data			

3.8 DDR layout (256 MBytes)

	Start address	Size (MBytes)
Linux HLOS 1	0x80000000	112
APPS BL(U-Boot)	0x87000000	4
SBL (BOOT)	0x87400000	1
CNSS_DEBUG	0x87500000	6
CPU Context Dump	0x87B00000	0.5
TZ_APPS	0x87B80000	2.5
SMEM	0x87E00000	0.5
TZ	0x87E80000	1.5
Linux HLOS 2 *	0x88000000	128

NOTE:

- NAND and eMMC layouts are applicable only for IPQ40x9.
- Linux HLOS 2 is not available for 128 MBytes DDR (standard profile).

3.9 Kernel memory protection

Starting with QCA_Networking_2016.SPF.4.0 release, support has been added for kernel memory protection capability, which enforces the following memory protection in kernel sections. Kernel generates an exception when memory protection is violated if the configuration is enabled.

- .text (contains the code): Only Read/Execute and no Write
- .rodata (contains Read-only data): Only Read and no Write/Execute
- .data (contains read/write data): Only Read/Write and no Execute

This feature is not enabled by default. To enable this feature, run **make kernel_menuconfig** command, and select CONFIG_ARM_KERNMEM_PERMS and CONFIG_DEBUG_RODATA.

4 Linux pin control drivers

The Linux pinctrl implementation provides a framework for configuring pins with appropriate attributes of drive strength, pull ups etc., as well as supporting multiplexing of pins for different use cases. The clients/consumers of pins interact with the framework through a combination of device tree bindings and interface APIs.

- Kernel documentation on pinctrl is available at: **documentation/pinctrl.txt**.
- Target specific documentation for pinctrl is available at: documentation/devicetree/bindings/pinctrl/qcom,ipq40xx-pinctrl.txt

Target specific documentation includes details on the available pins, supported functions, and examples on the usage.

4.1 Configure GPIO for SPI0 functionality

The GPIO pads can have one or more functional hardware interfaces. Update the "FUNC_SEL" field in the TLMM GPIO configuration register to control the pad how it is used. The pinctrl driver in Linux kernel supports this configuration.

To configure GPIO pins, do the following:

1. Add the entry under pinctrl@0x01000000 node in the appropriate dts file.
2. Include the added entry in the parent node.

The following is an example for DK04 to configure GPIO for SPI0 functionality:

3. In qcom-ipq40xx-ap.dk04.1.dtsi, configure GPIO 12, GPIO 13, GPIO 14, GPIO 15 for SPI0 functionality by adding the below node under pinctrl node.

```
    spi_0_pins: spi_0_pinmux {
  mux {
       pins = "gpio12", "gpio13", "gpio14", "gpio15";
       function = "blsp_spi0";
       bias-disable;
       };
     };
```

   Supported generic properties are:

   - Pins – the list of pins that the properties in the node apply to. Pin numbers are included as "gpioxx", "gpioyy"
   - Function – the mux function to select. The fun_sel name can be referred from pinctrl driver in drivers/pinctrl/qcom/pinctrl-ipq40xx.c in ipq40xx_groups [] which is described later in this subsection.

- bias-disable – disable any pin bias
- bias-pull-up – pull up the pin
- bias-pull-down – pull down the pin
- drive-strength – sink or source at most X mA
- input-enable – enable input on pin (no effect on output)
- input-disable – disable input on pin (no effect on output)
- output-low – set the pin to output mode with low-level
- output-high – set the pin to output mode with high-level

4. Include the spi pinctrl node in the SPI parent node as follows:

```
spi_0: spi@78b5000 { /* BLSP1 QUP1 */
            pinctrl-0 = <&spi_0_pins>;
            pinctrl-names = "default";
            status = "ok";
            m25p80@0 {
            #address-cells = <1>;
            #size-cells = <1>;
            reg = <0>;
            compatible = "n25q128a11";
            linux,modalias = "m25p80", "n25q128a11";
            spi-max-frequency = <24000000>;
            use-default-sizes;
            };
           };
```

The following are the GPIO fun_sel strings for SPI0 in Linux pinctrl driver:

PINGROUP(12, blsp_spi0, blsp_i2c1, NA, wcss0_dbg24, wcss1_dbg24, NA, NA, NA, NA, NA, NA, NA, NA, NA),

PINGROUP(13, blsp_spi0, blsp_i2c1, NA, wcss0_dbg25, wcss1_dbg25, NA, NA, NA, NA, NA, NA, NA, NA, NA),

PINGROUP(14, blsp_spi0, NA, wcss0_dbg26, wcss1_dbg26, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA),

PINGROUP(15, blsp_spi0, pcie_clk0, NA, wcss0_dbg27, wcss1_dbg27, NA, NA, NA, NA, NA, NA, NA, NA, NA),

4.2 Configure pin as GPIO

By default, all pins are in the GPIO mode. If the pin needs to be configured for an alternate functionality, the function name is to be specified in the dts using the 'function' property. If it is not specified, the default function will remain as GPIO.

The following is an example for configuring pins 7 and 9 as GPIO:

```
leds_pins: led_pinmux {
                    mux {
                         pins = "gpio7", "gpio9";
                         drive-strength = <2>;
                         bias-none;
                         output-high;
                         };
                       };
gpio-leds {
             compatible = "gpio-leds";
             pinctrl-0 = <&leds_pins>;
};
```

4.3 Linux OS device tree to register devices

The Linux operating system (OS) uses a device tree to find and register the devices in the system. Kernel documentation on device tree is available at **documentation/devicetree/usage-model.txt**. Target-specific documentation is available at **documentation/devicetree/bindings**.

The device tree is typically located at **linux/arch/arm/boot/dts** with the example tree structure as shown in Figure 4-1.

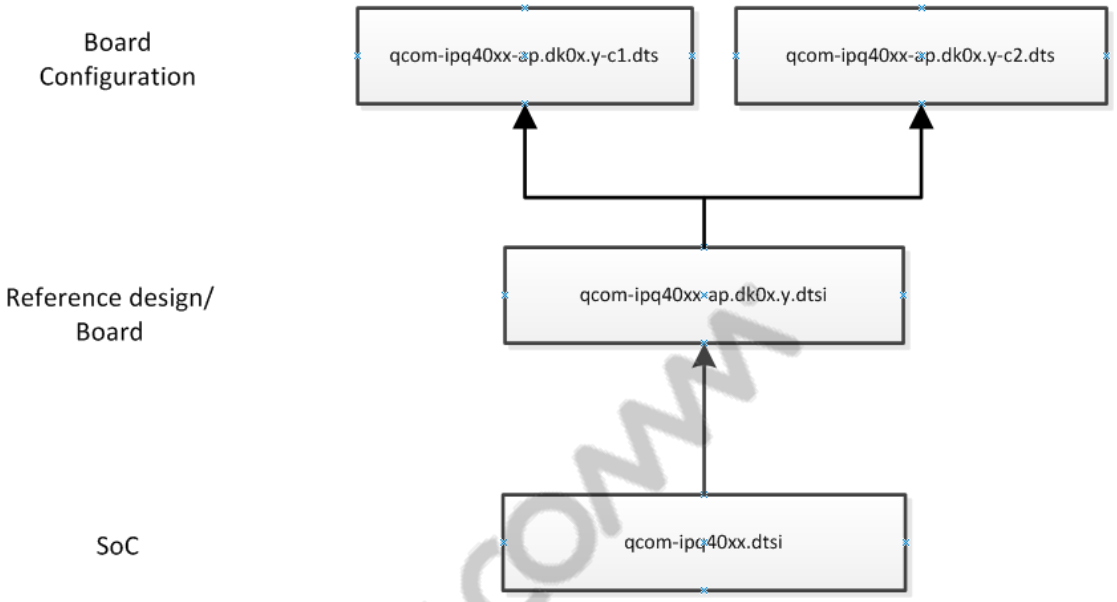


Figure 4-1 Example of a Linux OS device tree structure

The following are the different device tree structure specifications:

qcom-ipq40xx-ap.dk01.1-c1.dts:

IPQ4018 2X2 2G + 2X2 5G DBDC, EJTAG, SPI NOR flash (32 MBytes), UART, DDR3L, PSGMII, USB3.0/2.0, USB2.0

qcom-ipq40xx-ap.dk01.1-c2.dts/qcom-ipq40xx-ap.dk05.1-c1.dts:

IPQ4018 2X2 2G + 2X2 5G DBDC, EJTAG, SPI NOR flash (16 MBytes) and SPI NAND flash (128 MBytes), UART, DDR3L, PSGMII, USB3.0/2.0, USB2.0

qcom-ipq40xx-ap.dk04.1-c1.dts:

IPQ4019 2X2 2G + 2X2 5G DBDC, SPI NOR, QPIC NAND flash, PSGMII, MDIO, USB3.0/2.0, USB2.0, SD/MMC, EJTAG, UART0, UART1, LCD, PCIE2.0, LEDC

qcom-ipq40xx-ap.dk04.1-c2.dts:

IPQ4019 2X2 2G + 2X2 5G DBDC, SPI NOR, PSGMII, MDIO, USB3.0/2.0, USB2.0, Audio/PCM, EJTAG, UART0, UART1, PCIE2.0, LEDC

qcom-ipq40xx-ap.dk04.1-c3.dts:

Similar to qcom-ipq40xx-ap.dk04.1-c1.dts with NOR + eMMC boot

qcom-ipq40xx-ap.dk04.1-c5.dts:

Similar to qcom-ipq40xx-ap.dk04.1-c1.dts with NOR + SPI NAND boot

qcom-ipq40xx-ap.dk06.1-c1.dts:

IPQ4019 2X2 2G + 2X2 5G DBDC, QPIC NAND flash, PSGMII, MDIO, M2 (USB3.0), USB2.0, EJTAG, UART0, UART1, LCD, M2 PCIE2.0, LEDC

qcom-ipq40xx-ap.dk07.1-c1.dts:

IPQ4019 2X2 2G + 2X2 5G DBDC, SPI NOR, QPIC NAND flash, PSGMII, MDIO, USB3.0/2.0, USB2.0, SD, EJTAG, UART0, UART1, LCD, PCIE2.0, LEDC

qcom-ipq40xx-ap.dk01.1-cx.dts is also applicable for AP.DK03 platforms.

5 EDMA drivers

The EDMA hardware module provides the Tx/Rx queuing interface to Ethernet switch subsystem (ESS) module. The ESS/Switch module supports 5 GMAC interfaces. These GMAC interfaces are not directly accessible to the host operating system; they must be accessed through the EDMA queuing interface. The 5 GMAC interfaces are divided into two groups:

	ESS port mappings
LAN	0, 1, 2, 3, 4
WAN	0, 5

Port 0 (CPU port) acts as an interface between the EDMA and ESS modules. This port is added to both the LAN and WAN groups. Hence it is required to make it a VLAN trunk port.

The EDMA driver configures the EDMA queuing interface. It also creates two virtual network devices: eth0 (WAN) and eth1 (LAN). All packets received on the eth0 interface are forwarded to the ESS WAN group and all packets received on eth1 interfaces are forwarded to the ESS LAN group.

The EDMA driver uses the default VLAN tags to access LAN and WAN groups on ESS: 1 (LAN), 2 (WAN). These default tags can be modified and/or disabled through the Linux sysctl interface. In this case, the default tag support is disabled in the EDMA driver and users must configure tags through Linux vconfig or an equivalent interface.

5.1 EDMA driver features

This section describes the EDMA driver functionalities.

5.1.1 Queues

The EDMA hardware supports 16 Tx queues and 8 Rx queues. The EDMA driver currently makes use of all 16 Tx queues (4 queues per processor core) and 4 Rx queues (one queue per processor core).

The EDMA driver registers 4 Tx queues per eth interface (for lockless queueing across cores) and a total of 4 Rx queues for two eth interfaces, with Linux networking infrastructure.

The Tx queue selection process is as follows:

1. Identify the per-core block of 4 Tx queues (out of total 16 Tx queues) based on processor core.
2. Identify the per-interface block of 2 Tx queues (out of 4 Tx queues per processor core) based on interface (eth0/eth1).
3. Identify the final Tx queue (out of 2 Tx queues per interface per processor core) based on skb>priority.

The Rx queue is selected through RSS/RFS. The EDMA hardware supports 7-bits 2/3/5 tuple hash calculation for RSS decision. These 7-bits are used to index 128 entries into a hash table. The EDMA driver evenly distributes the hash across 4 Rx queues.

RFS suggestion is pushed from ESS 5-Tuple RFS table. This suggestion overrides any RSS decision taken by the EDMA hardware. The EDMA driver configures ESS RFS table based on Linux RFS decisions. The EDMA driver registers hooks with Linux RFS subsystem to add/remove RFS rules.

5.1.2 L4 offload

The EDMA driver supports:

- The following L4 offload features (through hardware):
  - TCP segmentation offload for IPv4/IPv6 packets
  - TCP/UDP/Generic checksum offload
- These features along with VLAN outer tag insertion/removal offload.
- These features for frames with/without PPPoE header.

5.1.3 Scatter-gather

The EDMA driver supports gather operation in the Tx direction. It can sniff through a Linux skbuff with nr_frags > 0 and configure the EDMA Tx descriptors accordingly. None of the L4 offload features are lost with scattered skbuff.

The EDMA driver supports scatter operation in the Rx direction when supporting for jumbo frames is enabled through device tree. All scatters are stored in paged buffers and only initial few bytes are copied to skbuff head area to take care of eth_type_trans () requirements. eth_type_trans() expects MAC header to be available in the skbuff head area.

5.1.4 QoS

The EDMA hardware/driver supports these mechanisms for QoS:

- Configure WRR weights per EDMA Tx queue for scheduling.
- Map EDMA Tx queue to ESS virtual queue. Extended QoS features and BW control can be configured through ESS virtual queues.

5.1.5 Interrupt affinity

In EDMA, there are 16 Tx interrupts and 8 Rx interrupts. Out of these, all 16 Tx queues and 4 Rx queues are used. 4 Tx queues are mapped to each core, while 1 Rx queue is mapped to each core.

Tx and Tx completion are handled in different cores

TxQ0-TxQ3	Tx is on Core0, while Tx completion is handled on Core2.
TxQ4-TxQ7	Tx is on Core1, while Tx completion is handled on Core3.
TxQ8-TxQ11	Tx is on Core2, while Tx completion is handled on Core0.
TxQ12-TxQ15	Tx is on Core3, while Tx completion is handled on Core1.

Rx and Rx completion are handled on the same core

RxQ0	Core0 is used
RxQ2	Core1 is used
RxQ4	Core2 is used
RxQ6	Core3 is used

The recommended interrupt affinity configuration (see section 3.2)

Tx Q0, Tx Q1, Tx Q2, Tx Q3	Core2
Tx Q4, Tx Q5, Tx Q6, Tx Q7	Core3
Tx Q8, Tx Q9, Tx Q10, Tx Q11	Core0
Tx Q12, Tx Q13, Tx Q14, Tx Q15	Core1
Rx Q0	Core0
Rx Q2	Core1
Rx Q4	Core2
Rx Q6	Core3

5.1.6 ATH_HDR insertion/removal

The EDMA driver supports conversion of ATH_HDR (Qualcomm header used by various Qualcomm switches) to in band information that can be passed in the Tx descriptor. It also inserts ATH_HDR based on the Rx descriptor in band information. At this point, this feature supports adding/removing ATH_HDR for STP/RSTP.

Format of ATH_HDR for transmit side

15:14	version	0x10
13:11	priority	Maps to skb>priority
10:8	action	0'b000, no map
7	from_cpu	Maps to FROM_CPU field in transmit packet descriptor
6:0	dp_bit_map	Depends on STP protocol; maps to DP_BITMAP field in transmit packet descriptor

Format of ATH_HDR for receive side

15:14	version	0x10
13:11	priority	Frame priority
10:6	frame type	0x4
5:4		Reserved
3	tagged-frame	Ingress frame is tagged
2:0	src_port_num	Ingress port number from the port id field in the Rx descriptor

5.1.7 Miscellaneous

The EDMA driver adds flow cookie and RSS hash output received from the ESS/EDMA hardware to skbuff. The flow cookie is used by shortcut forwarding engine. The RSS hash is used by Linux RPS/RFS subsystems.

5.2 EDMA driver configuration

This section describes the EDMA configuration settings:

5.2.1 Default VLAN tag for LAN/WAN groups

The EDMA driver assigns default VLAN tags to the eth0/eth1 interfaces. By default, eth0 (WAN) is assigned VLAN tag 2, while eth1 (LAN) is assigned VLAN tag 1. These tags are inserted and removed from packets by EDMA hardware. Linux OS is not aware of these tags.

Use the following commands to update the default VLAN configuration.

```
echo X >  /proc/sys/net/edma/edma_default_wtag (for WAN)
echo Y >  /proc/sys/net/edma/edma_default_ltag (for LAN)
```

There may be a need to use VLANs in network topology and/or make Linux OS aware of these tags by creating custom VLAN interfaces like eth0.2, eth1.1. In such a case, configure default VLAN tag to "0" and create VLANs on Linux OS:

```
vconfig add eth0 2
vconfig add eth1 1
```

NOTE: If the value of default VLAN tags is changed, then update the switch VLAN configuration.

5.2.2 Interrupt affinity for Tx/Rx queues

NOTE: These commands set the interrupt affinity; the affinity will be set automatically by a script. This section is for user information only; users do not need to do anything for this.

/* Tx interrupt infinity */

```
echo 4 >  /proc/irq/97/smp_affinity
echo 4 >  /proc/irq/98/smp_affinity
echo 4 >  /proc/irq/99/smp_affinity
echo 4 >  /proc/irq/100/smp_affinity
echo 8 >  /proc/irq/101/smp_affinity
echo 8 >  /proc/irq/102/smp_affinity
```

```
echo 8 >  /proc/irq/103/smp_affinity
echo 8 >   /proc/irq/104/smp_affinity
echo 1 >  /proc/irq/105/smp_affinity
echo 1 >  /proc/irq/106/smp_affinity
echo 1 >  /proc/irq/107/smp_affinity
echo 1 >  /proc/irq/108/smp_affinity
echo 2 >  /proc/irq/109/smp_affinity
echo 2 >  /proc/irq/110/smp_affinity
echo 2 >  /proc/irq/111/smp_affinity
echo 2 >  /proc/irq/112/smp_affinity
```

/* Rx interrupt affinity */

```
echo 1 > /proc/irq/272/smp_affinity
echo 2 > /proc/irq/274/smp_affinity
echo 4 > /proc/irq/276/smp_affinity
echo 8 > /proc/irq/278/smp_affinity
echo 1 > /proc/irq/273/smp_affinity
echo 2 > /proc/irq/275/smp_affinity
echo 4 > /proc/irq/277/smp_affinity
echo 8 > /proc/irq/279/smp_affinity
```

5.2.3 XPS/RFS configuration

Each interface (eth0 and eth1) has four Linux queues allotted for Tx. 4 Rx queues are shared between interfaces. Usually, Tx queue and Tx completion are assigned to different CPUs for better performance of unidirection flows.

NOTE: These commands map CPUs to the Linux queues.
This section is for user information only; users need not do anything for this.

```
echo 1 > /sys/class/net/eth0/queues/tx-0/xps_cpus
echo 2 > /sys/class/net/eth0/queues/tx-1/xps_cpus
echo 4 > /sys/class/net/eth0/queues/tx-2/xps_cpus
echo 8 > /sys/class/net/eth0/queues/tx-3/xps_cpus
echo 1 > /sys/class/net/eth0/queues/rx-0/rps_cpus
echo 2 > /sys/class/net/eth0/queues/rx-1/rps_cpus
echo 4 > /sys/class/net/eth0/queues/rx-2/rps_cpus
echo 8 > /sys/class/net/eth0/queues/rx-3/rps_cpus
echo 1 > /sys/class/net/eth1/queues/tx-0/xps_cpus
echo 2 > /sys/class/net/eth1/queues/tx-1/xps_cpus
echo 4 > /sys/class/net/eth1/queues/tx-2/xps_cpus
echo 8 > /sys/class/net/eth1/queues/tx-3/xps_cpus
echo 1 > /sys/class/net/eth1/queues/rx-0/rps_cpus
echo 2 > /sys/class/net/eth1/queues/rx-1/rps_cpus
echo 4 > /sys/class/net/eth1/queues/rx-2/rps_cpus
echo 8 > /sys/class/net/eth1/queues/rx-3/rps_cpus
```

This configuration is required to enable RFS for EDMA driver. *rps_flow_cnt* denotes the number of flows are supported per queue and *rps_sock_flow_entries* denotes the number of local socket flows are supported.

```
echo 256 > /sys/class/net/eth0/queues/rx-0/rps_flow_cnt
echo 256 > /sys/class/net/eth0/queues/rx-1/rps_flow_cnt
echo 256 > /sys/class/net/eth0/queues/rx-2/rps_flow_cnt
echo 256 > /sys/class/net/eth0/queues/rx-3/rps_flow_cnt
echo 1024 > /proc/sys/net/core/rps_sock_flow_entries
```

5.2.4 Page mode support

EDMA supports page mode. However, configuration changes are needed to enable this mode.

To configure, use one of the following options:

1. Change the board DTS (device tree) file. The device tree has a parameter "qcom, page-mode". This needs to be set to 1 in order to enable page mode processing in the EDMA driver. Both interfaces move to page mode when this configuration is enabled.

   Or

   Change the mode runtime. Once the kernel boots up, modify the file /etc/modules.d/45-qca-edma to:

   ```
   essedma overwrite_mode=1 page_mode=1
   ```

2. Save the file and reboot.

5.2.5 WRR configuration for Tx queues

The EDMA/ESS hardware supports weight-based Tx queue scheduling. Weight per queue can be configured using the following command:

```
echo 0x000Y000X  > /proc/sys/net/edma/weight_assigned_to_queues
```

Here, X = Tx queue_id (0–15)

Y = weight associated with the queue (0x0-0xF)

This command must be called for each queue individually. For example, weights of queue_id 1, queue_id 5, and queue_id 10 can be set to 4, 6, and 3, respectively using these commands:

```
echo 0x00040001  >  /proc/sys/net/edma/weight_assigned_to_queues
echo 0x00060005  >  /proc/sys/net/edma/weight_assigned_to_queues
echo 0x0003000a  >  /proc/sys/net/edma/weight_assigned_to_queues
```

5.2.6 EDMA Tx queue to ESS virtual queue mapping

The EDMA hardware supports EDMA Tx queue to ESS virtual queue mapping. Use ESS virtual queues to configure QoS per ESS port.

Use the following command to configure EDMA Tx queue to ESS virtual queue.

echo 0x000Y000X > /proc/sys/net/edma/queue_to_virtual_queue_map

Here, X = Tx queue_id (0–15).

Y = ESS virtual queue_id (0−7)

Call this command for each queue individually. For example, queue_id 1, queue_id 5 and queue_id 10 can be assigned virtual queues 4, 6, and 3 respectively using these commands:

```
echo 0x00040001  >  /proc/sys/net/edma/queue_to_virtual_queue_map
echo 0x00060005  >  /proc/sys/net/edma/queue_to_virtual_queue_map
echo 0x0003000a  >  /proc/sys/net/edma/queue_to_virtual_queue_map
```

5.2.7 Configuration through ethtool

ethtool can be used to enable and disable certain features during the runtime. By default, all net_device features are enabled. To enable/disable these features at runtime, use the following commands:

```
ethtool –K ethX  tx off/on   /*  Disable/enable tx checksum  */
ethtool –K ethX rx off/on   /*  Disable/enable rx checksum  */
ethtool –K ethX  tso off/on   /* disable/enable TCP segmentation */
ethtool –K ethX sg off/on   /* disable/enable scatter gather */
ethtool –K ethX  gro off/on  /* disable/enable generic receive offload */
ethtool –K ethX gso off/on  /* disable/enable generic segmentation offload */
ethtool –S ethX   /* Read statistics */
```

5.2.8 ESS WAN port ID

The WAN port number can be configured through DTS (device tree) parameter "**qcom,port_id_wan"**

By default, this DTS parameter is set to 0x5. This means that the EDMA driver supports port 5 as WAN port, while Port 0–4 are to be as LAN port. The SSDK (switch driver) configuration needs to be changed to support different WAN port prior to updating the EDMA WAN port configuration.

5.2.9 MAC address for LAN/WAN interfaces

MAC addresses for both interfaces are stored in the ART table. U-Boot reads the ART table and updates "**local-mac-address**" parameter for both the interfaces (LAN and WAN), under EDMA node in the device tree file. This device tree parameter is parsed by the EDMA driver and is used to configure MAC address in net_device structures for LAN and WAN interfaces.

In case, U-Boot does not fill the MAC address, the EDMA driver automatically assigns random MAC addresses to net_devices.

5.2.10 STP/RSTP ether type

To enable STP/RSTP ATH_HDR support:

```
echo 1 > /proc/sys/net/edma/enable_stp_rstp
```

To configure ether_ type:

```
echo 0xYYYY > /proc/sys/net/edma/athr_hdr_eth_type
```

The EDMA driver Tx path extracts ATH_HDR immediately after destination and source MAC addresses in the Ethernet header. There are 4 bytes in the format:

Destination MAC	Source MAC	ATH_HDR eth_type	ATH_HDR
6 bytes	6 bytes	2 bytes	2 bytes

The EDMA driver Rx path adds ATH_HDR immediately after destination and source MAC addresses in the Ethernet header. There are 4 bytes in the format:

Destination MAC	Source MAC	ATH_HDR eth_type	ATH_HDR
6 bytes	6 bytes	2 bytes	2 bytes

ATH_HDR eth_type is configured through sysctl as mentioned above.

5.2.11 Priority tagging enable/disable on switch

EDMA backpressures flow when one of the Rx queues are full. Backpressure is not priority-based. Both high and low priority flows may get back pressured. To avoid this situation, the EDMA driver starts dropping low priority packets (tagged as priority == 0 by switch) when Rx queue reaches upper limit threshold. This helps reduce impact of backpressure on high priority flows.

By default, switch is configured to priority tag incoming packets based on IP TOS value. However, if use case demands backpressure for both high and priority flows without any drop in the EDMA driver then switch configuration can be changed to not priority tag incoming packets.

This can be done by using one of the following methods:

1. Configure using SSDK shell commands

```
ssdk_sh qos ptMode set 5 dscp disable
ssdk_sh qos ptMode set 4 dscp disable
ssdk_sh qos ptMode set 3 dscp disable
ssdk_sh qos ptMode set 2 dscp disable
ssdk_sh qos ptMode set 1 dscp disable
```

2. Remove following section from /etc/config/network script

```
config switch_ext
        option device 'switch0'
        option name 'QosPtMode'
        option port_id '1'
        option mode 'dscp'
        option status 'enable'

config switch_ext
        option device 'switch0'
        option name 'QosPtMode'
        option port_id '2'
        option mode 'dscp'
        option status 'enable'

config switch_ext
        option device 'switch0'
        option name 'QosPtMode'
        option port_id '3'
        option mode 'dscp'
        option status 'enable'
```

```
config switch_ext
        option device 'switch0'
        option name 'QosPtMode'
        option port_id '4'
        option mode 'dscp'
        option status 'enable'

config switch_ext
        option device 'switch0'
        option name 'QosPtMode'
        option port_id '5'
        option mode 'dscp'
        option status 'enable'
```

6 SFE processes

Qualcomm has provided a technology that allows bridging and routing to be accelerated using the same processor that Linux is running on. The technology is called a Shortcut Forwarding Engine (SFE). SFE operates by hooking into the Linux firewall. Flows that are possible to accelerate are identified and maintained in a database. As packets traverse out of the netdev layer, the 5-tuple is checked against the database and packets are accelerated to the output netdev without traversing the Linux stack.

6.1 SFE kernel modules

Location	Description
/lib/modules/3.14/shortcut-fe.ko	SFE core. It receives packets from connection manager, translates packets according to internal forward table, and transmits packets. It cannot work independently. It is controlled by connection manager.
/lib/modules/3.14/shortcut-fe-cm.ko	SFE connection manager. It adds or deletes forward entry in shortcut forward engine core. Therefore, it decides whether to accelerate a connection.
/etc/init.d/shortcut-fe	This file checks compatibility, and decides whether to load shortcut forward engine connection manager or not.
/lib/modules/3.14/shortcut-fe-ipv6.ko	SFE core for IPv6. . It receives packets from connection manager, translates packets according to internal forward table, and transmits packets. It cannot work independently. It is controlled by connection manager.
/lib/modules/3.14/shortcut-fe-drv.ko	SFE adapter for ECM. It converts ECM message to SFE-required format, and converts SFE-returned value to ECM message and to transmit to ECM.

6.2 Kernel module dependency

Kernel module	Depends on
shortcut-fe-cm.ko	shortcut-fe-drv.ko, shortcut-fe.ko, shortcut-fe-ipv6.ko, nf_conntrack.ko

6.3 Enable shortcut forward engine

The shortcut forward engine is enabled when the kernel modules “shortcut-fe.ko”, “shortcut-fe-ipv6”, and “shortcut-fe-drv.ko” are loaded. The kernel modules “shortcut-fe.ko” and “shortcut-fe-ipv6.ko” are always loaded by default. To avoid conflict with other accelerators, the kernel

module “shortcut-fe-cm.ko” is not automatically loaded on some platform. User can manually enable it by inserting the kernel module “shortcut-fe-cm.ko”.

6.4 Disable shortcut forward engine

To disable the shortcut forward engine, remove the “shortcut-fe-cm.ko” kernel module.

6.5 Dump debug information of shortcut forward engine

To dump debug information of the shortcut forward engine, execute the following shell script:

```
#!/bin/sh
#@sfe_dump
#@example : sfe_dump
sfe_dump(){
    [ -e "/dev/sfe_ipv4" ] || {
         dev_num=$(cat /sys/sfe_ipv4/debug_dev)
         mknod /dev/sfe_ipv4 c $dev_num 0
    }
    [ -e "/dev/sfe_ipv6" ] || {
         dev_num=$(cat /sys/sfe_ipv6/debug_dev)
         mknod /dev/sfe_ipv6 c $dev_num 0
    }
    cat /dev/sfe_ipv4
    cat /dev/sfe_ipv6
}
```

Format of debug information

The debug information is formatted as an XML file and contains three top elements:

- Connections

Fields	Description
Src_dev, src_ip, src_ip_xlate, src_port, src_port_xlate	5-tuple in original direction
Dest_dev,dest_ip, dest_ip_xlate, dest_port, dest_port_xlate,	5-tuple in reply direction
Src_rx_pkts, src_rx_bytes	# of received packets and bytes in original direction
Dest_rx_packets, dest_rx_bytes	# of received packets and bytes in reply direction

- Exception statistics: Statistics about all kinds of exceptions which cause shortcut forward engine not to forward packet.

Global statistics

Fields	Description
Num_of_connections	Total number of active connections
Pkts_forwarded	Number of successfully forwarded packets
Pkts_not_forwarded	Number of packets which are received by shortcut but not forwarded by shortcut
Create_request	Number of creation requests from connection manager
Create_collisions	Number of collisions which means a connection already exist when create it
Destroy_requests	Number of destroy requests from the connection manager
Destroy_misses	Number of destroys missing which means a connection does not exist while destroying it
Flushes	Number of deletions which are caused by the protocol state rather than the connection manager
Hash_hits	Number of hits when looking up connection for incoming packets
Hash_reorder	Number of reorder operations which move the identified connection to the head of hash bucket.

7 QRFS operations

Qualcomm Receiving Flow Steering (QRFS) steers ESS packets to particular CPU cores. It focuses on packets from ESS to WLAN and tries to steer ESS packets to the CPU where the WLAN is binding. For example, WLAN 2.4 GHz is binding to CPU core 2, and if the packets are from Ethernet to WLAN 2.4 GHz, QRFS steers all of these packets to CPU core 2.

The WLAN in an IPQ40x8 system is supported by the Qualcomm AP 10.4 framework.

The AP 10.4 framework is used as documented in the *AP 10.4 CLI Users Guide* (80-Y8052-1) and the *AP 10.4 Programmer's Guide* (80-Y8053-1) with the exception that the CFGs must be ignored; these are handled in QSDK framework and need not be run.

This chapter describes the configurations and programming APIs provided by the QRFS module.

7.1 Configure QRFS using UCI

QRFS is enabled by default and can be changed with these UCI commands:

Start QRFS	*/etc/init.d/qrfs start*
Stop QRFS	*/etc/init.d/qrfs stop*
Enable QRFS	*/etc/init.d/qrfs enable*
Disable QRFS	*/etc/init.d/qrfs_disable*

7.2 View QRFS debug details

The proc: file system shows the QRFS debug messages:

Status of QRFS	*cat /proc/qrfs/enable*
Debug level	*cat /proc/qrfs/debug*
Rules were set	*cat /proc/qrfs/rule*
Interfaces is managed	*cat /proc/qrfs/vif*
Connections (5-tuples) are managed	*cat /proc/qrfs/connection*

7.3 Programing API definitions

The QRFS kernel module could support any network device with the hardware RFS feature and ESS/EDMA device. A network device that wants to support the QRFS feature should implement these APIs in net device driver or kernel module.

7.3.1 int rfs_ess_device_register(struct rfs_device *dev)

Definition	Register a device which supports RFS features	
Prototype	int rfs_ess_device_register(	

	struct rfs_device *dev	struct rfs_device {
		char *name;
		/*Device name*/
		rfs_dev_mac_rule_cb mac_rule_cb;
		/*MAC based RFS rule callback*/
		rfs_dev_ip4_rule_cb ip4_rule_cb;
		/*IPv4 based RFS rule callback */
		rfs_dev_ip6_rule_cb ip6_rule_cb;
		/*IPv4 based RFS rule callback */
		}

Return value	0 on success, negative for failure	

7.3.2 int rfs_ess_device_unregister(struct rfs_device *dev)

Definition	Unregister a RFS device previously registered	
Prototype	int rfs_ess_device_unregister(	

	struct rfs_device *dev	struct rfs_device {
		char *name;
		/*Device name*/
		rfs_dev_mac_rule_cb mac_rule_cb;
		/*MAC based RFS rule callback*/
		rfs_dev_ip4_rule_cb ip4_rule_cb;
		/*IPv4 based RFS rule callback */
		rfs_dev_ip6_rule_cb ip6_rule_cb;
		/*IPv6based RFS rule callback */
		}

Return value	0 on success, negative for failure	

7.3.3 typedef int (*rfs_dev_mac_rule_cb)(uint16_t vid, uint8_t *mac, uint8_t ldb, int is_set)

Definition	Call back function to set a MAC address based RFS rule; it should be implemented by a RFS device.	
Prototype	int (*rfs_dev_mac_rule_cb)(	
	uint16_t vid,	VLAN ID of the ingress interface from which the flows come.
	uint8_t *mac,	Destination MAC address of the flows
	uint8_t ldb	Load balance attribute, currently it's equal to CPU core ID. The EDMA driver is responsible for the correct mapping from LB to Rx queues (so as to the right cores).
	int is_set	1: set 0: delete
	)	
Return value	0 on success, negative for failure	

7.3.4 typedef int (*rfs_dev_ip4_rule_cb)(uint16_t vid, uint32_t ipaddr, uint8_t *mac, uint8_t ldb, int is_set);

Definition	Call back function to set an IPv4 address based RFS rule; it should be implemented by a RFS device.	
Prototype	int (*rfs_dev_ip4_rule_cb)(	
	uint16_t vid,	VLAN ID of the ingress interface from which the flows come.
	uint32_t ipaddr,	Destination IP address of the flows
	uint8_t *mac,	Destination MAC address of the flows
	uint8_t ldb	Load balance attribute, currently it's equal to CPU core ID. The EDMA driver is responsible for the correct mapping from LB to Rx queues (so as to the right cores).
	int is_set	1: set 0: delete
	)	
Return Value	0 on success, negative for failure	

7.3.5 typedef int (*rfs_dev_ip6_rule_cb)(uint16_t vid, uint8_t *ipaddr, uint8_t *mac, uint8_t ldb, int is_set)

Definition	Call back function to set an IPv4 address based RFS rule; it should be implemented by a RFS device.	
Prototype	int (*rfs_dev_ip6_rule_cb)(	
	uint16_t vid,	VLAN ID of the ingress interface from which the flows come.
	uint8_t *ipaddr,	Destination IPv6 address of the flows
	uint8_t *mac,	Destination MAC address of the flows
	uint8_t ldb	Load balance attribute, currently it's equal to CPU core ID. The EDMA driver is responsible for the correct mapping from LB to Rx queues (so as to the right cores).
	int is_set	1: set 0: delete
	)	
Return Value	0 on success, negative for failure	

7.3.6 Linux tuple based RFS

Definition	The net device should implement Linux 5-tuple based RFS. For more details, see the Linux kernel documents.	
Prototype	int (*ndo_rx_flow_steer)(	
	struct net_device *dev	Linux net device
	const struct sk_buff *skb,	Packet which will be steering
	u16 rxq_index,	Rx queue index
	u32 flow_id	Identification of a flow
	)	
Return Value	0 on success, negative for failure	

8 HNAT

For host network address translation (HNAT), both static NAT and NAPT functions are supported. The HNAT module is built into SSDK kernel module. When SSDK module is loaded, the HNAT function is disabled by default.

HNAT supports two modes. When mode 0 is enabled, it uses full 1 K NPAT entries, but does not synchronize corresponding counters to Linux kernel. When mode 1 is enabled, it synchronizes the counters to Linux kernel but supports only 8 NAPT entries. By default, mode 0 is enabled.

In the *AP 10.4 Programmer's Guide*, the HNAT Wi-Fi offload section is not applicable for IPQ40xx chipsets.

8.1 HNAT enable UCI configuration

Enable HNAT mode 0:	`config switch_ext`
	`  option device 'switch0'`
	`  option name 'NatGlobal'`
	`  option status 'enable'`
	`  option sync 'disable'`

Enable HNAT mode 1:	`config switch_ext`
	`  option device 'switch0'`
	`  option name 'NatGlobal'`
	`  option status 'enable'`
	`  option sync 'enable'`

8.2 HNAT disable UCI configuration

To disable HNAT, enter the following shell command:

```
ssdk_sh nat global set disable disable
```

Disable HNAT:	`config switch_ext`
	`  option device 'switch0'`
	`  option name 'NatGlobal'`
	`  option status 'disable'`
	`  option sync 'disable'`

8.3 Static NAT configuration

To configure static NAT, enter the following iptables command:

```
iptables -t nat -A POSTROUTING -p tcp -s 192.168.11.10 -j SNAT --to
192.168.10.1
```

NOTE: It needs to be configured after HNAT is enabled.

8.4 View HNAT entries

To view NAT entries, enter the following command:

```
ssdk_sh nat naptentry show
```

9 Debugging

All debug interfaces for the IPQ40x8/ IPQ40x9 are accessible to an external debugger through a standard JTAG connector. Qualcomm recommends using the Trace32 in-circuit debugger from Lauterbach GmbH that supports the Cortex-A7 debugging. See Lauterbach Debugger for *IPQ4018/IPQ4019/IPQ4028/IPQ4029 Application Note* (80-Y9571-5) for additional information.

Figure 9-1 shows the Lauterbach power debug interface USB 3 connected between the JTAG header on the board and a USB 3.0 interface on a PC.

Figure 9-1 Set-up for debugging with Lauterbach debugger