Qualcomm

Qualcomm Technologies, Inc.

Qualcomm Linux Interfaces Guide

80-70018-8 Rev. AA

March 26, 2025

Contents

Tables

Figures

1 Overview of peripheral interfaces

The Qualcomm® Linux® interfaces guide describes both the low-speed and high-speed input/output (I/O) peripheral interface subsystems used in the System-on-Chip (SoC).

- The low-speed I/O interfaces operate at a lower frequency than the high-speed interfaces in the SoC. The Qualcomm universal peripheral (QUP) v3 serial engine is a hardware core that supports the following low-speed peripheral interfaces:
  - Universal asynchronous receiver/transmitter (UART)
  - Serial peripheral interface (SPI)
  - Interintegrated circuit (I2C)
  - Improved interintegrated circuit (I3C)

  These low-speed interfaces communicate with low-speed peripherals, such as sensor interface devices, Bluetooth® wireless technology devices, display or touch interface devices.
- The high-speed I/O interfaces include the following peripheral interfaces:
  - Peripheral component interconnect express (PCIe)
  - Universal serial bus (USB)

  The high-speed I/O interfaces are used to connect to high-speed devices, such as solid-state drives, network cards, external graphics cards.

The following figure shows the software stack for peripheral interfaces.

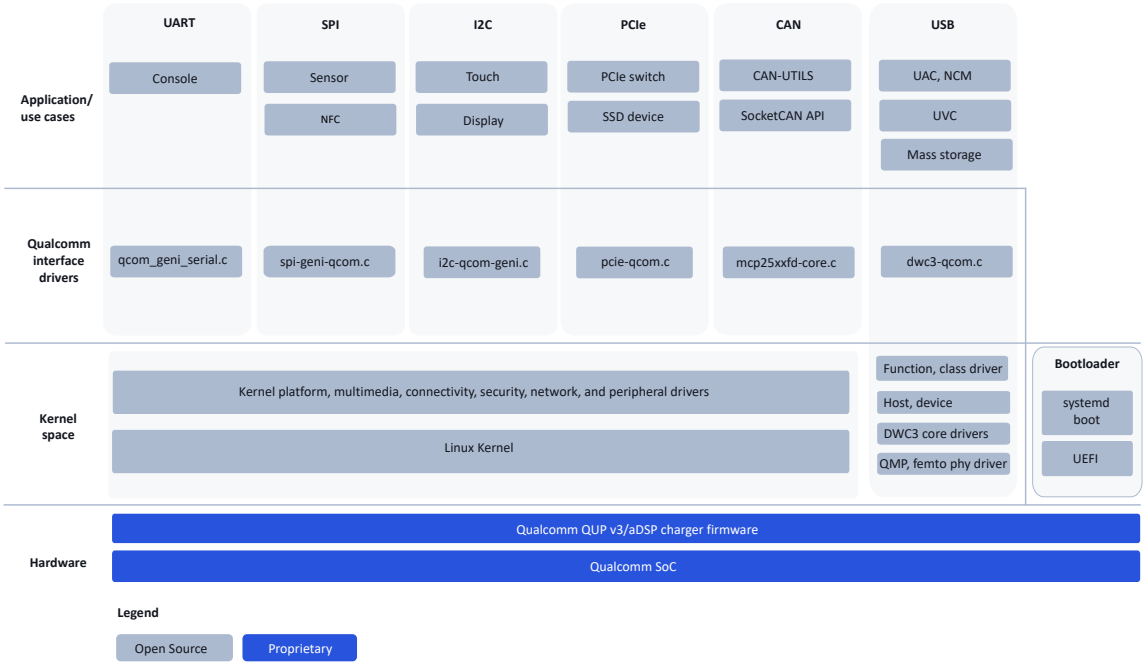


Figure 1-1 Peripheral interfaces software stack

QUP v3

Qualcomm uses QUP v3, a highly flexible and programmable hardware to support a wide range of serial interfaces. For more information about QUP v3, see the following resources:

- QUP v3 overview
- Supported transfer modes in QUP v3
- QUP v3 access control customization
- QUP v3 firmware status verification

NOTE
- The source code for boot and aDSP subsystems is available to licensed developers with authorized access.
- See Hardware SoCs that are supported on Qualcomm Linux.

This guide is your primary resource for information about enabling low-speed and high-speed I/O interfaces and making configuration changes.

NOTE Across this guide, `<chipset>` refers to QCS6490, QCS5430, QCS9075, or QCS8275. For example, for `arch/arm64/boot/dts/qcom/<chipset>.dts`, the corresponding device tree source (DTS) file is at https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts or https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi.

1.1 QCS6490 and QCS5430 interface overview

The following table lists the device interface features for QCS6490 and QCS5430.

Table 1-1 QCS6490 and QCS5430 device interfaces

2xQUP v3 serial engine		
Serial engine instances	QUPV3_0	QUPV3_1
Application processor QUP v3 serial engine	8	8
LPI QUP v3 serial engine	5	–
2xUSB controller		
Controller address	0xa600000	0x8c00000
Max Speed	USB 3.x SuperSpeed	USB 2.0 high speed
HS/SS PHY power rails	■ L1C: VDD_A_USB_HS_1P8 ■ L2B: VDD_A_USB_HS_3P1 ■ L10C:VDD_A_USB_HS_CORE ■ L6B: VDD_A_USB_SS_DP_1P2 ■ L1B:VDD_A_USB_SS_DP_CORE	■ L1C: VDD_A_USB_1_HS_1P8 ■ L2B: VDD_A_USB_1_HS_3P1 ■ L10C:VDD_A_USB_1_HS_CORE
2xPCIe controller		
Root complex	RC1	RC0
Speed	Gen3 2L (8 GT/s)	Gen3 1L (8 GT/s)
Configuration space	0x40100000 (0x100000) 1 MB	0x60100000 (0x100000) 1 MB
I/O space	0x40200000 (0x100000) 1 MB	0x60200000 (0x100000) 1 MB
Base address register space (BAR)	0x40300000 (0x3d00000)	0x60300000 (0x1fd00000) 509 MB
Power rails	■ vreg_l0c (VDD_A_PCIE_0_CORE) ■ vreg_l6b (VDD_A_PCIE_0_PLL_1P2)	■ vreg_l0c (VDD_A_PCIE_1_CORE) ■ vreg_l6b (VDD_A_PCIE_1_PLL_1P2)
Interrupts	Message signaled interrupts (MSI) and peripheral component interconnect (PCI) legacy interrupts	MSI and PCI legacy interrupts
Power management	Active-state power management (ASPM) L1/ L1ss, L0s	ASPM (L1/L1ss, L0s)

QUP v3 mapping to protocols and GPIOs in QCS6490 and QCS5430

QCS6490 and QCS5430 contain 21 QUP v3 serial engines. Of these, 16 QUP v3 serial engines are allocated for the application processor and 5 QUP v3 serial engines are allocated for the sensor low-power island (SLPI) on the application digital signal processor (aDSP).

Select only one protocol in a QUP v3 serial engine at a time. For example, simultaneous UART and I2C functions aren't supported. Each QUP v3 serial engine has up to seven lanes (I/O), which are numbered from 0 to 6.

NOTE The top-level QUP v3 serial engines are used as the application processor and boot image. The low-power island (LPI) QUP v3 serial engines are used for the sensor subsystem and Qualcomm® Trusted Execution Environment (TEE) use cases.

The following table lists the default QUP v3 mapping to protocols and GPIOs.

Table 1-2 QCS6490 and QCS5430 QUP v3 serial engine protocol mapping

QUP v3 serial engine		Protocols					QUP lane to GPIO mapping						
		UART	I3C	HS UART	SPI-M	I2C	L0	L1	L2	L3	L4	L5	L6
QUP_0	SE0	Yes	Yes	–	Yes	Yes	0	1	2	3	–	–	–
	SE1	Yes	Yes	–	Yes	Yes	4	5	6	7	–	–	–
	SE2	Yes	–	–	Yes	Yes	8	9	10	11	–	–	–
	SE3	Yes	–	–	Yes	Yes	12	13	14	15	–	–	–
	SE4	Yes	–	–	Yes	Yes	16	17	18	19	–	–	–
	SE5	Yes	–	–	Yes	Yes	20	21	22	23	–	–	–
	SE6	Yes	–	Yes	Yes	Yes	24	25	26	27	–	–	–
	SE7	Yes	–	Yes	Yes	Yes	28	29	30	31	2	3	6
QUP_1	SE0	Yes	Yes	–	Yes	Yes	32	33	34	35	–	–	–
	SE1	Yes	Yes	–	Yes	Yes	36	37	38	39	–	–	–
	SE2	Yes	–	–	Yes	Yes	40	41	42	43	–	–	–
	SE3	Yes	–	–	Yes	Yes	44	45	45	47	–	–	–
	SE4	Yes	–	–	Yes	Yes	48	49	50	51	55	54	38
	SE5	Yes	–	–	Yes	Yes	52	53	54	55	–	–	–
	SE6	Yes	–	Yes	Yes	Yes	56	57	58	59	62	63	50
	SE7	Yes	–	Yes	Yes	Yes	60	61	62	63	–	–	–
LPI_QUP	SE0	–	Yes	–	–	Yes	159	160	–	–	–	–	–
	SE1	–	Yes	–	–	Yes	161	162	–	–	–	–	–
	SE2	–	Yes	–	Yes	Yes	163	164	165	166	161	–	–
	SE5	Yes	–	–	–	Yes	171	172	171	172	–	–	–
	SE6	Yes	–	Yes	–	–	159	159	–	–	–	–	–

1.2 QCS9075 interface overview

The following table lists the device interface features in QCS9075.

Table 1-3 QCS9075 device interfaces

2xQUP v3 serial engine				
Serial engine instances	QUPV3_0	QUPV3_1	QUPV3_2	QUPV3_3

Table 1-3 QCS9075 device interfaces (cont.)

Application processor QUP v3 serial engine	6	7	7	1
3xUSB controller				
Controller address	0xa600000	0xa800000	0xa400000	
Maximum speed	USB 3.x SuperSpeed	USB 3.x SuperSpeed	USB 2.0 high speed	
HS/SS PHY power rails	■ L7A: VDD_A_USBHS_0_0P9 ■ L6C: VDD_A_USBHS_0_1P8 L9A: VDD_A_USBHS_0_3P1 ■ L7A: VDD_A_USBSS_0_0P9 ■ L1C: VDD_A_USBSS_0_1P2	■ L7A:VDD_A_USBHS_1_0P9 ■ L6C:VDD_A_USBHS_1_1P8 ■ L9A:VDD_A_USBHS_1_3P1 ■ L7A:VDD_A_USBSS_1_0P9 ■ L1C:VDD_A_USBSS_1_1P2	■ VDD_A_USBHS_2_1P8 ■ VDD_A_USBHS_2_3P1	
2xPCIe controller				
Root complex	RC1		RC0	
Speed	Gen4 2L (8 GT/s)		Gen4 4L (8 GT/s)	
Configuration space	0x40100000 (0x100000) 1 MB		0x60100000 (0x100000) 1 MB	
I/O space	0x40200000 (0x100000) 1 MB		0x60200000 (0x100000) 1 MB	
Base address Register space (BAR)	0x40300000 (0x1fd00000) 509 MB		0x60300000 (0x1fd00000) 509 MB	
Power rails	■ vreg_l5a (VDD_A_PCIE_0_CORE) ■ vreg_l1c (VDD_A_PCIE_0_PLL_1P2)		■ vreg_l5a (VDD_A_PCIE_1_CORE) ■ vreg_l1c (VDD_A_PCIE_1_PLL_1P2)	
Interrupts	MSI and PCI earlier interrupts		MSI and PCI earlier interrupts	
Power management	ASPM (L1/L1ss, L0s)		ASPM (L1/L1ss, L0s)	

QUP v3 mapping to protocols and GPIOs in QCS9075

QCS9075 has 26 QUP v3 serial engines. The following table list the protocol and GPIO mapping.

Table 1-4 QCS9075 QUP v3 serial engine protocol mapping

QUP v3 serial engine		Protocols				QUP lane to GPIO mapping						
		UART	HS UART	I2C-M	SPI-M	L0	L1	L2	L3	L4	L5	L6
QUP_0	SE0	Yes	–	Yes	Yes	20	21	22	23	–	–	–
	SE1	Yes	–	Yes	Yes	24	25	26	27	–	–	–

Table 1-4 QCS9075 QUP v3 serial engine protocol mapping (cont.)

QUP v3 serial engine		Protocols				QUP lane to GPIO mapping						
		UART	HS UART	I2C-M	SPI-M	L0	L1	L2	L3	L4	L5	L6
	SE2	Yes	Yes	Yes	Yes	36	37	38	39	–	–	–
	SE3	Yes	Yes	Yes	Yes	28	29	30	31	–	–	–
	SE4	Yes	–	Yes	Yes	32	33	34	35	–	–	–
	SE5	Yes	–	Yes	Yes	36	37	38	39	–	–	–
QUP_1	SE0	Yes	–	Yes	Yes	40	41	42	43	–	–	–
	SE1	Yes	–	Yes	Yes	42	43	40	41	–	–	–
	SE2	Yes	Yes	Yes	Yes	46	47	44	45	–	–	–
	SE3	Yes	Yes	Yes	Yes	44	45	46	47	–	–	–
	SE4	Yes	–	Yes	Yes	48	49	50	51	–	–	–
	SE5	Yes	–	Yes	Yes	52	53	54	55	–	–	–
	SE6	Yes	–	Yes	Yes	56	57	56	57	–	–	–
QUP_2	SE0	Yes	–	Yes	Yes	80	81	82	83	–	–	–
	SE1	Yes	–	Yes	Yes	84	85	99	100	–	–	–
	SE2	Yes	Yes	Yes	Yes	86	87	88	89	90	–	–
	SE3	Yes	Yes	Yes	Yes	91	92	93	94	–	–	–
	SE4	Yes	–	Yes	Yes	95	96	97	98	–	–	–
	SE5	Yes	–	Yes	Yes	99	100	84	85	–	–	–
	SE6	Yes	–	Yes	Yes	97	98	95	96	–	–	–
QUP_3	SE0	Yes	–	Yes	Yes	13	14	15	16	17	18	19

1.3 QCS8275 interface overview

The following table lists the device interface features for the QCS8275.

2xQUP v3 serial engine		
Serial engine instances	QUPV3_0	QUPV3_1
Application processor QUP v3 serial engine	8	8
2xUSB controller		
Controller address	0xa600000	0xa400000
Maximum speed	USB 3.x SuperSpeed	USB 2.0 high speed
HS/SS PHY power rails	L17A: VDDA-PHY-SUPPLY	L17A: VDDA-PLL-SUPPLY
	L15A: VDDA-PLL-SUPPLY	L17C: VDDA18-SUPPLY
	L17A: VDDA-PLL-SUPPLY	L19A: VDDA33-SUPPLY
	L18C: VDDA	–
	L19A: VDDA33-SUPPLY	–
2xPCIe controller		

Root complex	RC1	RC0
Speed	Gen4 2L (16 GT/s)	Gen4 4L (16 GT/s)
Configuration space	0x40100000 (0x100000) 1 MB	0x60100000 (0x100000) 1 MB
I/O space	0x40200000 (0x100000) 1 MB	0x60200000 (0x100000) 1 MB
Base address Register space (BAR)	0x40300000 (0x1fd00000) 509 MB	0x60300000 (0x1fd00000) 509 MB
Power rails	vreg_l6a (VDD_A_PCIE_0_CORE)	vreg_l5a (VDD_A_PCIE_1_CORE)
	vreg_l5a (VDD_A_PCIE_0_PLL_1P2)	vreg_l6a (VDD_A_PCIE_1_PLL_1P2)
Interrupts	MSI and PCI legacy interrupts	MSI and PCI legacy interrupts
Power management	ASPM (L1/L1ss, L0s)	ASPM (L1/L1ss, L0s)

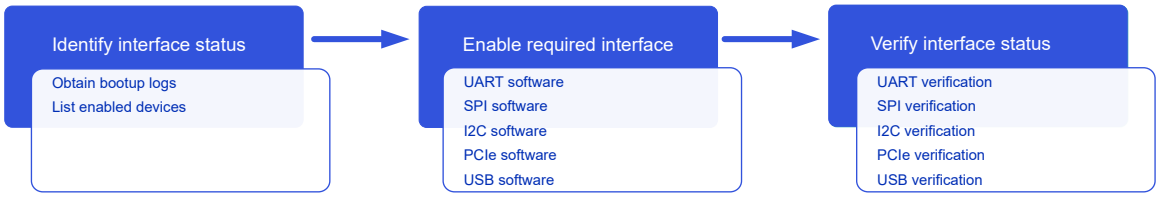
QUP v3 mapping to protocols and GPIOs in QCS8275

QCS8275 has two QUP v3 serial engines. The following table lists the protocol and GPIO mapping.

QUP v3 serial engine		Protocols				QUP lane to GPIO mapping						
		UART	HS UART	I2C-M	SPI-M	L0	L1	L2	L3	L4	L5	L6
QUP_0	SE0	Yes	–	Yes	Yes	17	18	19	20	–	–	–
	SE1	Yes	–	Yes	Yes	19	20	17	18	–	–	–
	SE2	Yes	Yes	Yes	Yes	33	34	35	36	–	–	–
	SE3	Yes	Yes	Yes	Yes	25	26	27	28	–	–	–
	SE4	Yes	–	Yes	Yes	29	30	31	32	–	–	–
	SE5	Yes	–	Yes	Yes	21	22	23	24	–	–	–
	SE6	Yes	–	Yes	Yes	80	81	82	83	–	–	–
	SE7	Yes	–	Yes	Yes	43	44	43	44	–	–	–
QUP_2	SE0	Yes	–	Yes	Yes	37	38	39	40	–	–	–
	SE1	Yes	–	Yes	Yes	39	40	37	38	–	–	–
	SE2	Yes	Yes	Yes	Yes	84	85	86	87	88	–	–
	SE3	Yes	Yes	Yes	Yes	41	42	41	42	–	–	–
	SE4	Yes	–	Yes	Yes	45	46	47	48	–	–	–
	SE5	Yes	–	Yes	Yes	49	50	51	52	–	–	–
	SE6	Yes	–	Yes	Yes	89	90	91	92	–	–	–
	SE7	Yes	–	Yes	Yes	91	92	89	90	–	–	–

2 Getting started: Set up device interface

Before you begin, set up your infrastructure as described in the Qualcomm Linux Build Guide.



2.1 Identify interface status

The default interface status indicates the status of the interfaces during bootup. To identify the interface status, you must ensure that the interfaces are registered successfully and obtain the list of enabled interfaces.

2.1.1 Obtain the bootup logs

To obtain the bootup logs of the device, do the following:

1. Open the SSH shell in permissive mode or use the ADB shell. For more information about how to run SSH, see the Use SSH section.
2. Obtain the logs of the enabled interfaces by running the following command.

   ```
   dmesg
   ```

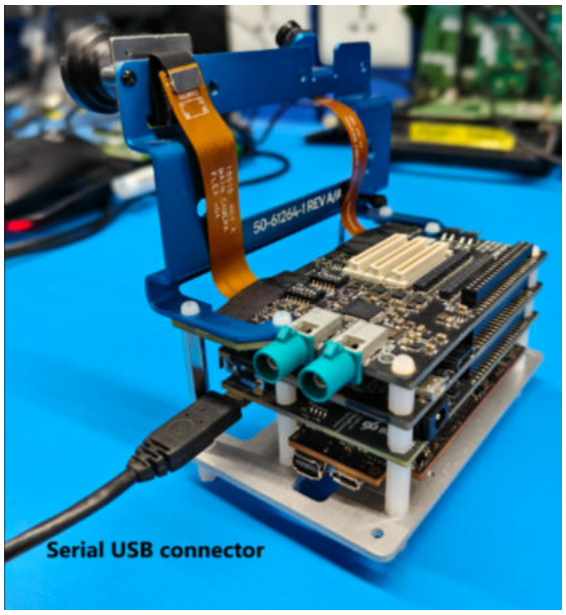
   Output:

```
[    0.434365] msm_serial: driver initialized
[    0.799123] 994000.serial: ttyMSM0 at MMIO 0x994000 (irq = 139,
base_baud = 0) is a MSM
[    0.801937] 99c000.serial: ttyHS1 at MMIO 0x99c000 (irq = 140,
base_baud = 0) is a MSM
[    0.804563] serial serial0: tty port ttyHS1 registered

[ 0.720945] usbhub_rest_vreg GPIO handle specifies active low - ignored
[    0.815241] dwc3-qcom 8c00000.usb: Adding to iommu group 4
[    5.195974] dwc3-qcom a600000.usb: Adding to iommu group 18
[    5.229464] qcom_pmic_glink pmic-glink: Failed to create device link
(0x180) with a600000.usb
[    6.195825] usb usb2: We don't know the algorithms for LPM for this
host, disabling LPM.
```

```
[    6.450338] usb 1-1: new high-speed USB device number 2 using
xhci_hcd
[    6.664583] usbcore: registered new device driver onboard-usb-hub
[    6.730998] usb 2-1: new SuperSpeed USB device number 2 using
xhci_hcd
[    7.083668] usb 2-1.1: new SuperSpeed USB device number 3 using
xhci_hcd
[    7.217287] ax88179_178a 2-1.1:1.0 eth0: register 'ax88179_178a' at
usb-0001:04:00.0-1.1, ASIX AX88179 USB 3.0 Gigabit Ethernet,
02:fe:ee:05:44:23
[    7.217415] usbcore: registered new interface driver ax88179_178a
[   10.223140] dwc3-qcom a600000.usb: request 0000000000000000 was not
queued to ep0ou
```

3. Connect the UART serial port to log in to the console.



2.1.2 List enabled interfaces on devices

To obtain the list of the enabled interfaces, do the following:

- For UART, run the following command.

  ```
  ls /dev/tty*
  ```

  Output:

```
/dev/tty    /dev/tty21 /dev/tty35  /dev/tty49  /dev/tty62   /dev/ttyp4
/dev/tty0   /dev/tty22 /dev/tty36  /dev/tty5   /dev/tty63   /dev/ttyp5
/dev/tty1   /dev/tty23 /dev/tty37  /dev/tty50  /dev/tty7    /dev/ttyp6
/dev/tty10  /dev/tty24 /dev/tty38  /dev/tty51  /dev/tty8    /dev/ttyp7
/dev/tty11  /dev/tty25 /dev/tty39  /dev/tty52  /dev/tty9    /dev/ttyp8
/dev/tty12  /dev/tty26 /dev/tty4   /dev/tty53  /dev/ttyMSM0 /dev/ttyp9
/dev/tty13  /dev/tty27 /dev/tty40  /dev/tty54  /dev/ttyS0   /dev/ttypa
/dev/tty14  /dev/tty28 /dev/tty41  /dev/tty55  /dev/ttyS1   /dev/ttypb
/dev/tty15  /dev/tty29 /dev/tty42  /dev/tty56  /dev/ttyS2   /dev/ttypc
/dev/tty16  /dev/tty3  /dev/tty43  /dev/tty57  /dev/ttyS3   /dev/ttypd
/dev/tty17  /dev/tty30 /dev/tty44  /dev/tty58  /dev/ttynull /dev/ttype
/dev/tty18  /dev/tty31 /dev/tty45  /dev/tty59  /dev/ttyp0   /dev/ttypf
```

```
/dev/tty19  /dev/tty32 /dev/tty46  /dev/tty6   /dev/ttyp1
/dev/tty2   /dev/tty33 /dev/tty47  /dev/tty60  /dev/ttyp2
/dev/tty20  /dev/tty34 /dev/tty48  /dev/tty61  /dev/ttyp3
```

- For I2C, run the following command.

  ```
  ls /dev/i2c*
  ```

  The following output is displayed.

  ```
  /dev/i2c-0  /dev/i2c-1 /dev/i2c-16
  ```

- For SPI, run the following command.

  ```
  ls /dev/spi*
  ```

  The following output is displayed.

  ```
  spidev14.0
  ```

- For PCIe, obtain the enumeration log. For more information about PCIe probe logs, see PCIe-related configurations and QPS615 switch support.

2.2 Load Linux firmware on QUP v3 serial engine

The QUP v3 serial engine loads the firmware for the required protocol onto the serial engine. The configuration of protocols (I2C, SPI, I3C) and communication modes (FIFO, GSI), is done in a secure execution environment, such as Qualcomm TEE `devcfg`. For open-source development, this configuration is performed in the Linux `devkit`, allowing for the nonsecure use cases. However, for secure use cases, Qualcomm TEE `devcfg` is used.

Default configuration

The default configuration for each serial engine, including the selected mode of data transmission and ownership, is in the `/TZ.XF.5.0/core.tz/2.0/settings/buses/qup_accesscontrol/qupv3/config/lemans/QUPAC_Access.c` file.

In the default configuration, Linux owns all the serial engines. All secure use cases are handled from Qualcomm TEE.

```
const QUPv3_se_security_permissions_type qupv3_perms_default =
{
  /*   PeriphID,         ProtocolID,               Mode,
NsOwner,         bAllowFifo, bLoad, bModExcl  */
  /*QUPV3_0_SE0*/
  /*QUPV3_0_SE1*/
  /*QUPV3_0_SE2*/
  { QUPV3_0_SE3, QUPV3_PROTOCOL_UART_4W,   QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // BT UART (2nd Hastings)
  { QUPV3_0_SE4, QUPV3_PROTOCOL_UART_2W,   QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // VIP UART/SPI (SOC SLAVE)
  /
*QUPV3_0_SE5*/
                       // Spare
  { QUPV3_1_SE0, QUPV3_PROTOCOL_I2C,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // I2C Carplay
```

```
  /*QUPV3_1_SE1*/
  { QUPV3_1_SE2, QUPV3_PROTOCOL_UART_2W,   QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // Tuner
  { QUPV3_1_SE3, QUPV3_PROTOCOL_UART_2W,   QUPV3_MODE_FIFO,
AC_HLOS,            TRUE, FALSE, FALSE }, // Debug UART
  { QUPV3_1_SE4, QUPV3_PROTOCOL_I2C,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // I2C A2B Controller & Audio port
expander
  { QUPV3_1_SE5, QUPV3_PROTOCOL_UART_2W,   QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // GNSS
  /*QUPV3_1_SE6*/
  { QUPV3_2_SE0, QUPV3_PROTOCOL_SPI,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // FPGA
  { QUPV3_2_SE1, QUPV3_PROTOCOL_I2C,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // PCIe I2C MUX
  { QUPV3_2_SE2, QUPV3_PROTOCOL_SPI,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // SPI - Audio
  { QUPV3_2_SE3, QUPV3_PROTOCOL_UART_4W,   QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // BT UART
  { QUPV3_2_SE4, QUPV3_PROTOCOL_I2C,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // I2C Display 1
  { QUPV3_2_SE5, QUPV3_PROTOCOL_I2C,       QUPV3_MODE_FIFO,
AC_HLOS,            TRUE,  TRUE, FALSE }, // I2C Sensor
  /*QUPV3_2_SE6 */
  { QUPV3_3_SE0,  QUPV3_PROTOCOL_SPI,       QUPV3_MODE_FIFO,
AC_HLOS,           TRUE,  FALSE, FALSE }, // SPI
};
```

Load Linux firmware

To load the Linux firmware, do the following:

1. Enable CONFIG_QUP_FW_LOAD in the following files:
   - `CONFIG_QCOM_QUP_FW_LOAD=y` in `arch/arm64/configs/qcom_defconfig`
   - `CONFIG_QCOM_QUP_FW_LOAD=m` in `arch/arm64/configs/qcom_vm_defconfig`
2. To load the firmware with the required protocol, include the `qcom,load-firmware` property in the device tree node.
3. Configure the transfer mode.

```
GENI_SE_INVALID,      // 0
GENI_SE_FIFO,    //1
GENI_SE_DMA,    //2
GENI_GPI_DMA,    //3
```

4. Load the firmware to the serial engine.

   For example, when QUPv3_2_SE2 (0x00888000) is loaded from Linux through the FIFO mode of communication use the following DTSI property.

```
spi16: spi@888000 {
         compatible = "qcom,geni-spi";
         reg = <0x0 0x00888000 0x0 0x4000>;
         interrupts = <GIC_SPI 584 IRQ_TYPE_LEVEL_HIGH>;
         clocks = <&gcc GCC_QUPV3_WRAP2_S2_CLK>;
         clock-names = "se";
         interconnects = <&clk_virt MASTER_QUP_CORE_2 QCOM_ICC_TAG_ALWAYS
                 &clk_virt SLAVE_QUP_CORE_2 QCOM_ICC_TAG_ALWAYS>,
                 <&gem_noc MASTER_APPSS_PROC QCOM_ICC_TAG_ALWAYS
                 &config_noc SLAVE_QUP_2 QCOM_ICC_TAG_ALWAYS>,
                 <&aggre2_noc MASTER_QUP_2 QCOM_ICC_TAG_ALWAYS
                 &mc_virt SLAVE_EBI1 QCOM_ICC_TAG_ALWAYS>;
         interconnect-names = "qup-core",
                 "qup-config",
                 "qup-memory";
         power-domains = <&rpmhpd SA8775P_CX>;
                dmas = <&gpi_dma2 0 2 QCOM_GPI_SPI>,
                <&gpi_dma2 1 2 QCOM_GPI_SPI>;
                dma-names = "tx", "rx";
         address-cells = <1>;
         #size-cells = <0>;
        qcom,load-firmware;
        xfer-mode = <1>;
         status = "disabled";
                   };
```

5. Load the GPI firmware.

```
gpi_dma2: qcom,gpi-dma@800000  {
             #dma-cells = <3>;
             compatible = "qcom,sm6350-gpi-dma";
             reg = <0x0 0x00800000 0x0 0x60000>;
             interrupts = <GIC_SPI 588 IRQ_TYPE_LEVEL_HIGH>,
                  <GIC_SPI 589 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 590 IRQ_TYPE_LEVEL_HIGH>,
                   <GIC_SPI 591 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 592 IRQ_TYPE_LEVEL_HIGH>,
                  <GIC_SPI 593 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 594 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 595 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 596 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 597 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 598 IRQ_TYPE_LEVEL_HIGH>,
                      <GIC_SPI 599 IRQ_TYPE_LEVEL_HIGH>;
             dma-channels = <12>;
             dma-channel-mask = <0xfff>;
```

```
iommus = <&apps_smmu 0x5b6 0x0>;
qcom,load-firmware;
xfer-mode = <3>;
            status = "disabled";
        };
```

2.3 Enable required interfaces

To enable an interface, do the following:

- For UART, see the UART software section.
- For SPI, see the SPI software section.
- For I2C, see the I2C software section.
- For PCIe, see the PCIe software section.
- For USB, see the USB software section.

2.4 Verify interface status

To verify the functioning of the different interfaces, do the following:

- For UART, see the Verify UART interface section.
- For SPI, see the SPI verification section.
- For I2C, see the Verify I2C interface section.
- For PCIe, verify the connected endpoint with the following command.

  ```
  lspci
  ```

  The following output is displayed.

  ```
  0001:00:00.0 PCI bridge: Qualcomm Device 010b
  0001:01:00.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:01.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:02.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:03.0 PCI bridge: Toshiba Corporation Device 0623
  0001:04:00.0 USB controller: Renesas Technology Corp. uPD720201 USB 3.0
  Host Controller (rev 03)
  0001:05:00.0 Ethernet controller: Toshiba Corporation Device 0220
  0001:05:00.1 Ethernet controller: Toshiba Corporation Device 0220
  ```

- For USB, verify the device and host as follows:
  - Device: Connect the USB Type-C port and verify the enumerated log with the host PC.

    ```
    adb devices
    ```

    The following output is displayed.

    ```
    List of devices attached
    541eb4ba        device
    ```

- Host: Connect a USB device such as a mouse, or a pen drive, and verify device detection with the following command.

  ```
  lsusb
  ```

  The following output is displayed.

  ```
  Bus 002 Device 003: ID 0b95:1790 ASIX Electronics Corp. AX88179
  Gigabit Ethernet
  Bus 002 Device 002: ID 05e3:0625 Genesys Logic, Inc. USB3.2 Hub
  Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
  Bus 001 Device 002: ID 05e3:0610 Genesys Logic, Inc. Hub
  Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
  ```

3 UART

UART devices transmit data asynchronously. Hence, a clock signal doesn't synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet, so the receiving UART knows when to start reading the bits. When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate.

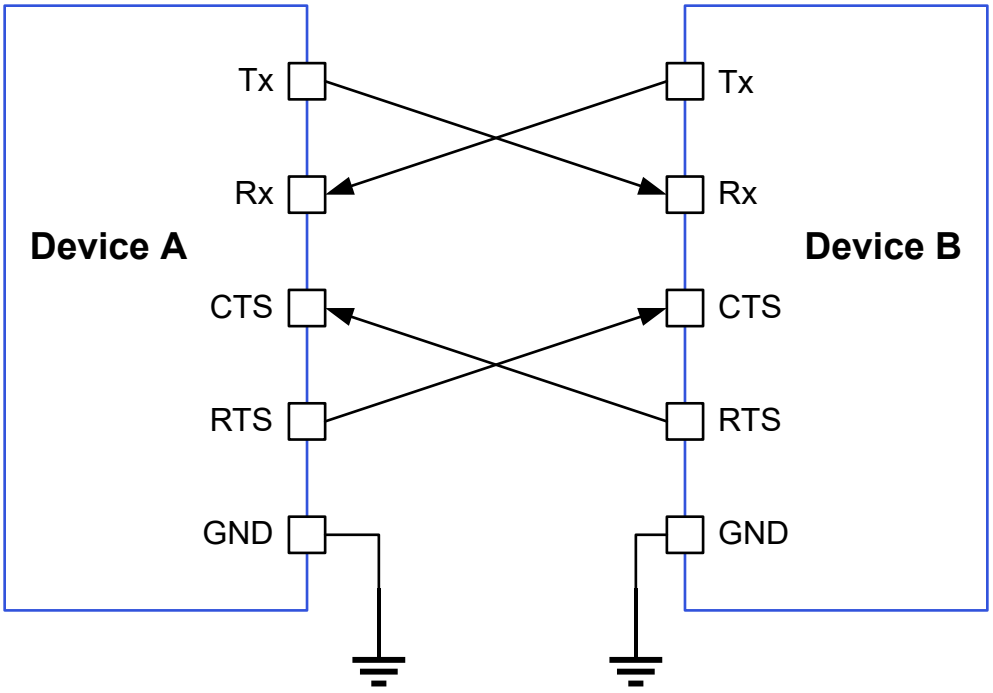


Figure 3-1 Data transfer between two UART devices

The parameters that determine successful transmission are as follows:

- Baud rate
- Start bit
- Stop bit
- Parity bit

- Data bits
- Flow control

The following figure shows a sample UART data packet.

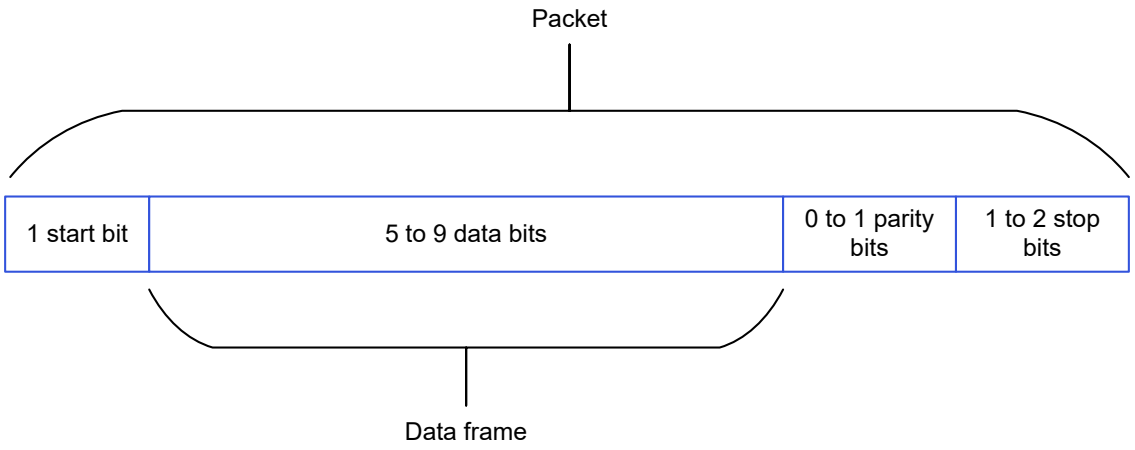


Figure 3-2 UART data packet

3.1 UART features

The following table describes the UART transfer modes for applications.

Table 3-1 UART transfer modes

Subsystem	Transfer mode	Description
Linux	▪ FIFO (low speed) ▪ CPU DMA (high speed)	▪ Supports baud rates from 300 bps up to 4 Mbps. ▪ FIFO mode transfers data between its Rx/Tx buffers and system memory ▪ DMA mode transfers data between its Rx/Tx buffers and system memory. Better performance is achieved with high-speed UART drivers supporting higher baud rates and bigger data packages. For example, the Bluetooth wireless technology connectivity module.
Boot	FIFO	▪ Rx/Tx 5 bits to 8 bits per character. ▪ Supports a maximum baud rate of 115200.
aDSP	FIFO	▪ Rx/Tx 5 bits to 8 bits per character. ▪ Supported baud rates: 115200, 230400, 460800, 921600,

Table 3-1 UART transfer modes (cont.)

Subsystem	Transfer mode	Description
		1000000, 3000000, and 6000000.

3.2 UART interface

The following table provides the paths of UART driver configurations for the different subsystems.

Table 3-2 UART interface: Linux

File type	Description
Device tree source	■ QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi ■ QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
`Pinctrl` settings	■ The pin control table for the corresponding QUP v3 serial engine is at `<workspace_path_of_LINUX_kernel_image>/sources/kernel/kernel_platform/kernel/arch/arm64/boot/dts/qcom/<chipset>.dts`. ■ For DTSI configuration examples to override the chip product, see the following DTSI files. □ QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi □ QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
Qualcomm TEE settings	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c` .

Table 3-3 UART interface: Boot (UEFI-only)

File type	Description
QUP v3 serial engine configuration	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/QcomPkg/SocPkg/<chipset>/Settings/UART/UartSettings.c`
Qualcomm TEE settings	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`

Table 3-4 UART interface: aDSP/SLPI

File type	Description
QUP v3 serial engine configuration	■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_devcfg.c ■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c
Firmware configuration settings	■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml

3.2.1 UART APIs

UART APIs for the following subsystems are listed in this section.

- Linux:https://github.com/torvalds/linux/blob/master/include/linux/tty.h
- Boot: QcomPkg/Include/HSUart.h
- aDSP: adsp_proc/core/api/buses/uart.h

3.3 UART software

This section provides information on the UART device tree configuration, and documentation for the device nodes.

Linux

For information about Kernel device instances, see https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/serial/qcom%2Cserial-geni-qcom.yaml.

For information about the UART driver files, see https://github.com/torvalds/linux/blob/master/drivers/tty/serial/qcom_geni_serial.c

```
uart7: serial@99c000 {
/* Manufacturer model of serial driver */
compatible = "qcom,geni-uart";
/* SE address and size */
reg = <0 0x0099c000 0 0x4000>;
/*Clocks for SE */
clocks = <&gcc GCC_QUPV3_WRAP0_S7_CLK>;
clock-names = "se";
/* pinctrl setting */
pinctrl-names = "default";
pinctrl-0 = <&qup_uart7_cts>, <&qup_uart7_rts>, <&qup_uart7_tx>,
<&qup_uart7_rx>;
interrupts = <GIC_SPI 608 IRQ_TYPE_LEVEL_HIGH>;
power-domains = <&rpmhpd SC7280_CX>;
```

```
operating-points-v2 = <&qup_opp_table>;
interconnects = <&clk_virt MASTER_QUP_CORE_0 0 &clk_virt SLAVE_QUP_CORE_0 0>,
<&gem_noc MASTER_APPSS_PROC 0 &cnoc2 SLAVE_QUP_0 0>;
                interconnect-names = "qup-core", "qup-config";
/* To enable QUPV3 serial engine instance for UART protocol, change Status to
OK */
                status = "disabled";
            };
        }
```

For configuration settings of the serial engine GPIOs, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

```
qup_uart7_cts: qup-uart7-cts-state {
                pins = "gpio28";
                function = "qup07";
            };

            qup_uart7_rts: qup-uart7-rts-state {
                pins = "gpio29";
                function = "qup07";
            };

            qup_uart7_tx: qup-uart7-tx-state {
                pins = "gpio30";
                function = "qup07";
            };

            qup_uart7_rx: qup-uart7-rx-state {
                pins = "gpio31";
                function = "qup07";
            };
```

NOTE The Qualcomm TEE configurations must be aligned in the `QUPAC_Access.c` file to ensure that the GPIO/QUP v3 can be used. You can access the Qualcomm TEE images at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`. Modify the required settings or see the default settings assigned for particular instances of the QUP v3 serial engine.

QUP v3 supports both 4-wire UART with flow-control enabled, and 2-wire UART without flow control enabled. The following example for Qualcomm TEE access, controls entry for both. The QUP v3 serial engine configurations to enable the UART protocol are as follows:

- Default configuration enabled for SE7 as HS UART

  ```
  { QUPV3_0_SE7, QUPV3_PROTOCOL_UART_4W, QUPV3_MODE_FIFO, AC_HLOS, TRUE,
  TRUE, FALSE },
  ```

- 2-wire UART configuration for SE5

```
uart5: serial@994000 {
compatible = "qcom,geni-uart";
reg = <0 0x00994000 0 0x4000>;
clocks = <&gcc GCC_QUPV3_WRAP0_S5_CLK>;
clock-names = "se";
pinctrl-names = "default";
pinctrl-0 =  <&qup_uart5_tx>, <&qup_uart5_rx>;
interrupts = <GIC_SPI 606 IRQ_TYPE_LEVEL_HIGH>;
power-domains = <&rpmhpd SC7280_CX>;
operating-points-v2 = <&qup_opp_table>;
interconnects = <&clk_virt MASTER_QUP_CORE_0 0 &clk_virt SLAVE_QUP_CORE_0
0>,
    <&gem_noc MASTER_APPSS_PROC 0 &cnoc2 SLAVE_QUP_0 0>;
interconnect-names = "qup-core", "qup-config";
status = "disabled";
};
```

```
{ QUPV3_0_SE5, QUPV3_PROTOCOL_UART_2W, QUPV3_MODE_FIFO,
AC_HLOS, TRUE, FALSE, FALSE },
```

Boot

The QUP v3 serial engine can be configured to UART in boot using the `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/QcomPkg/SocPkg/<chipset>/Settings/UART/UartSettings.c` file.

```
UART_PROPERTIES devices =
{
   // MAIN_PORT
   0x00994000,    // Serial Engine Base address
   0x009C0000,// qup_common base address
   0x2001c161,  // GPIO TX pin Config
   0x2000c171,  // GPIO RX Pin Config
   0,           // gpio_cts_config
   0,           // gpio_rfr_config
   0,           // clock_id_index
   (void*)0,        // bus_clock_id
   (void*)CLK_QUPV3_WRAP0_S5,     // core_clock_id
   0,          // irq number not used
   0,  //TCSR base
   0,  // TCSR offset
   0   // TCSR value

};
```

GPIO configuration

The GPIO configuration is listed in the following table.

Table 3-5 UART GPIO configuration

Bits	Parameters
[0:3]	GPIO function
[4:13]	GPIO number
[14]	Direction
[15:17]	Pull type
[18:21]	Drive strength

You must configure each GPIO based on the following bit fields.

```
<!-- GPIO configuration calculation GPIO DIR values GPIO_INPUT = 0x0
GPIO_OUTPUT = 0x1 GPIO_PULL values GPIO_NO_PULL = 0, /**< -- Do not specify a
pull. */ GPIO_PULL_DOWN = 0x1, /**< -- Pull the GPIO down. */
GPIO_KEEPER_ENABLE = 0x2, /**< -- Keeper Enable. */ GPIO_PULL_UP = 0x3, /**<
-- Pull the GPIO up. */ GPIO_DRV_STRENGTH values GPIO_2P0MA = 0, /**< --
Specify a 2 mA drive. */ GPIO_4P0MA = 0x1, /**< -- Specify a 4 mA drive. */
GPIO_6P0MA = 0x2, /**< -- Specify a 6 mA drive. */ GPIO_8P0MA = 0x3, /**< --
Specify a 8 mA drive. */ GPIO_FUNC_SELECT_Value GPIO_FS_VAL =0, //Specifies
GPIO function GPIO_FS_VAL =0X1, //Specify NON GPIO function( UART/SPI/I2C)
GPIO configuration = (GPIO_NUM & 0xFF) << 0x10 | (GPIO_FS_VAL & 0xF) << 0xC |
(GPIO_DRV_STRENGTH & 0xF) << 0x8 | (GPIO_PULL & 0xF) << 0x4 | (GPIO_DIR &
0xF) --> /* | RESERVED | GPIO NUM | DRIVE | FUNC | PULL | DIR |
------------------------------------------------------------------------ |
0000 | 0000 | 0001 | 1110 | 0001 | 0001 | 0010 | 0001 |
------------------------------------------------------------------------ */
```

aDSP

The firmware loads SSC QUP during the bootup sequence of the aDSP subsystem. Hence, the configuration file is present in the aDSP build at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c`.

The following configuration is a sample of SSC QUP SE5/6 loaded with the UART firmware in the FIFO mode.

```
                    offset,        protocol,   mode,  load_fw, dfs_mode
se_cfg se0_cfg = { 0x80000, SE_PROTOCOL_I3C,    GSI,     TRUE, TRUE  };
se_cfg se1_cfg = { 0x84000, SE_PROTOCOL_I2C,    GSI,     TRUE, TRUE  };
se_cfg se2_cfg = { 0x88000, SE_PROTOCOL_I2C,    GSI,     TRUE, TRUE  };
se_cfg se3_cfg = { 0x8C000, SE_PROTOCOL_I2C,    GSI,     FALSE, TRUE  };
se_cfg se4_cfg = { 0x90000, SE_PROTOCOL_SPI,    GSI,     TRUE, TRUE };
se_cfg se5_cfg = { 0x94000, SE_PROTOCOL_UART,   FIFO,    TRUE,FALSE };
se_cfg se6_cfg = { 0x98000, SE_PROTOCOL_UART,   FIFO,    TRUE,FALSE  };
```

GPIO configuration: Each serial engine in the QUP common driver is configured with the default GPIO configuration per protocol. The QUP v3 common driver picks the GPIO configuration according to the protocol loaded in the serial engine from `/firmware/qualcomm-linux-`

spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_instance_mapping.c.

The default GPIO configuration can be overwritten as follows.

```
{       .instance_id          =  6 ,         //Instance ID
         .qup              =  QUP_SSC,    //QUP Type
         .se_index         =  5,          //SE ID
         .se_data          =  NULL,       //devcfg_map
         .protocol_io_cfg  =  {

TLMM_MAP(TLMM_GPIO_KEEPER ,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER ),              //
SLEEP CFG

TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_6MA,TLMM_GPIO_KEEPER ),              //SP
I CFG

TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL),              //
UART CFG

TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL),              //I2
C CFG

TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER )               //I3
C CFG
                              },
         .se_exclusive     =  TRUE,
}
```

TLMM_MAP is a macro to initialize the active and sleep state GPIO configurations. For example, sample usage of the TLMM_MAP macro.

```
TLMM_MAP (active state pull type, drive strength, sleep state pull type)
```

3.4 Enable virtualization in UART

This section covers the high-level flow from a guest virtual machine client to the hardware port of the host machine, considering Kernel-based virtual machine (KVM) as the virtual machine. A foundational understanding of virtualization, hypervisor, and virtual machine technology is beneficial.

Table 3-6 Virtualization features

Virtualization components	Feature description
VirtIO	■ Provides an abstraction for a set of common emulated devices in a paravirtualized hypervisor ■ Uses UART device ports and serve application requests. Ensure VirtIO support or virtual function I/O (VFIO) at the KVM

Table 3-6 Virtualization features (cont.)

Virtualization components	Feature description
	■ Provides a common front end for the device emulations to standardize the interface and increase code reuse across platforms. ■ Supports virtualization environments that implement the VirtIO standard, such as QEMU/KVM.
Hypervisor	■ Exports a common set of emulated devices for a common application programming interface (API). ■ Implements a common set of interfaces, with the particular device emulation behind a set of back-end drivers

ZigBee use case over UART in guest virtual machine

To identify the device character name, enumerate the ZigBee interface in Linux as a serial interface. For example, if ZigBee is connected over UART, it's enumerated as `/dev/MSMx`.

NOTE This section uses `/dev/MSM0` as an example. Replace it with the actual character device ID.

1. Disable SELinux by running the following command.

   ```
   setenforce 0
   ```

2. To enable KVM, do the following:
   a. Boot the device to UEFI.
   b. Select option **17** in the BDS menu to enter UEFI menu.
   c. Select option **25** to enter OS configuration selection menu.
   d. Increment the OS type to **2** (Linux with KVM) with up arrow.
   e. Select **Enter Power Cycle** device. Wait for the device to be online.
   f. Verify the device node.

      ```
      ls /dev/kvm
      ```

3. Push the KVM image from `workspace>\builds\kvm\gunyah_k2l\svm\lemans_svm\svm\Image.gz`.

   ```
   adb push Image /mnt/overlay/
   ```

4. Push the `initrd` boot image from `workspace>pkondeti\builds\kvm\gunyah_k2l\svm\lemans_svm\svm\yocto.cpio.gz`.

   ```
   adb push yocto.cpio /mnt/overlay/
   ```

5. To connect the device to the guest virtual machine, you can use one of the following two options.
   - Option **A**: For quick emulator (QEMU) virtualizer, connect the `/dev/ttyMSM0` character device to the guest virtual machine using the following command.

     ```
     qemu-system-aarch64 \
             -M virt -m 2G \
             -initrd <your_initrd> \
             -kernel <your_kernel> \
             -device virtio-serial-pci \
     ```

```
-chardev tty,path=/dev/ttyMSM0,id=char0 \
-device virtserialport,chardev=char0,name=zigbee \
-cpu host --enable-kvm -smp 4 -nographic
```

- Option **B**: For `libvirt` virtualizer, copy the following values into the file at https://github.qualcomm.com/pkondeti/qli_virt_recipes/blob/main/example/hk-vm.xml.

```
<channel
type='dev'>


    <source path='/dev/
ttyMSM0'/>

    <target type='virtio'
name='zigbee'/>

    <alias
name='zigbee'/>


    <address type='virtio-
serial'/>

</channel>
```

6. To locate and verify the `virtio-serial` virtual serial port, run the following command to list all ports.

   ```
   ls /dev/virtio-ports/zigbee
   ```

7. Verify the virtual port connection.

   ```
   echo "hello from guest" >> /dev/virtio-ports/zigbee
   ```

   Output:

   ```
   Hello from guest
   ```

3.5 UART tools

This section provides information on various test tools and methods for the UART serial interface driver to confirm the UART data transfers.

Linux

For more details, see https://docs.kernel.org/admin-guide/serial-console.html.

3.6 Enable UART in kernel

This section provides information on how to enable UART in the kernel.

Linux

The following driver kernel configurations are required to support the UART interface.

- UART driver: https://github.com/torvalds/linux/blob/master/drivers/tty/serial/qcom_geni_serial.c
- Kernel `defconfig` file path: `<workspace_path_of_LINUX_kernel_image>/sources/kernel /kernel_platform/kernel/arch/arm64/configs/qcom_defconfig`

The following kernel configurations must be enabled.

- `CONFIG_QCOM_GENI_SE=y`
- `CONFIG_SERIAL_QCOM_GENI=y`

To enable a serial node for the loopback validation, apply the following patch to the `/arch/arm64/boot/dts/qcom/<chipset>.dtsi` file.

```
--- a/arch/arm64/boot/dts/qcom/<chipset>.dtsi
+++ b/arch/arm64/boot/dts/qcom/<chipset>.dtsi
@@ -70,6 +70,7 @@
          spi13 = &spi13;
          spi14 = &spi14;
          spi15 = &spi15;
+         serial1 = &uart7;
};


+
+&uart7 {
+    status = "ok";
+};
```

NOTE You should compile the kernel configuration and device tree changes. After compilation, you can load the images to the device to verify the interface. For information about interface verification, see the Verify UART interface section.

Boot/aDSP

For customizations, see the UART software section.

3.7 UART customization

For information about customizing UART software, see QUP v3 access control customization.

3.8 Verify UART interface

This section describes the validation procedure for the UART drivers, and the test results for the Qualcomm drivers.

Linux

To enable the UART nodes, do the following and compile the kernel configuration.

1. To change the UART status to `OK` and aliases to the specific UART node, edit the following DTSI files.
   - QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
   - QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

     NOTE Enable the SSH shell or use the ADB shell to run the commands and display the output in the SSH shell (console) window. For more information about how to run SSH, see the Use SSH section.

```
aliases {
i2c0 = &i2c0;
spi15 = &spi15;
++serial1 = &uart7;
};
uart7: serial@99c000 {
compatible = "qcom,geni-uart";
reg = <0 0x0099c000 0 0x4000>;
clocks = <&gcc GCC_QUPV3_WRAP0_S7_CLK>;
clock-names = "se";
pinctrl-names = "default";
pinctrl-0 = <&qup_uart7_cts>, <&qup_uart7_rts>, <&qup_uart7_tx>,
<&qup_uart7_rx>;
interrupts = <GIC_SPI 608 IRQ_TYPE_LEVEL_HIGH>;
power-domains = <&rpmhpd SC7280_CX>;
operating-points-v2 = <&qup_opp_table>;
interconnects = <&clk_virt MASTER_QUP_CORE_0 0 &clk_virt SLAVE_QUP_CORE_0
0>,
<&gem_noc MASTER_APPSS_PROC 0 &cnoc2 SLAVE_QUP_0 0>;
interconnect-names = "qup-core", "qup-config";
++status = "ok";
};
```

2. Disable the `if` condition in the `qcom_geni_serial.c` at https://github.com/torvalds/linux/blob/master/drivers/tty/serial/qcom_geni_serial.c file for the loopback test.

```
//if (mctrl & TIOCM_LOOP) // Disabling the if condition for loopback test
port->loopback = RX_TX_CTS_RTS_SORTED;
```

To validate the QUP v3 UART registration functionality in the Linux kernel, ensure that the UART is correctly registered with the TTY stack.

1. Disable the UART default use case in the following DTSI files.
   - QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
   - QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

   ```
   bluetooth: bluetooth {

          ++      status = "disabled";
   ```

   The following output is displayed.

   ```
   ls /dev/ttyHS1
   /dev/ttyHS1
   dmesg | grep ttyH
   [    3.355487] 99c000.serial: ttyHS1 at MMIO 0x99c000 (irq = 137,
   base_baud = 0) is a MSM
   ```

2. To verify the UART driver, do the following:
   a. Open the SSH shell in permissive mode or use the ADB shell.
   b. Register the UART.

      ```
      ls /dev/ttyHS*
      ```

      The following is a sample output.

      ```
      ls /dev/ttyHS*
      /dev/ttyHS1
      ```

      Map the `ttyHS1` port according to the aliases added for the `serial1 = &uart7` and enable the serial engine.

The UART devices registered in the kernel are listed. The UART driver follows the test sequence to enable loopback. After enabling the UART node in the DUT, run the following commands to verify that the UART instance is enabled in the DTSI file.

NOTE Open two SSH shells or use the ADB shell to write and read the data for the UART loopback. For more information about how to run SSH, see the Use SSH section.

1. Open the SSH shell in permissive mode or use the ADB shell.
2. Transfer data with the `echo` command.

   ```
   echo "This Document Is Very Much Helpful" > /dev/ttyHS1
   ```

3. Read data in the UART device node.

   ```
   cat /dev/ttyHS1
   ```

3.9 Debug UART issues

This section provides information about enabling the debug logs in the UART software driver.

Linux

UART driver logging is enabled through the dynamic debugging method. Enable `CONFIG_DYNAMIC_DEBUG` in `<workspace_path_of_LINUX_kernel_image>/sources/kernel/kernel_platform/kernel/arch/arm64/configs/qcom_defconfig` to support the dynamic debugging for kernel drivers.

To enable and view the UART driver logs in the kernel logs (`dmesg`), run the following command.

```
mount -t debugfs none /sys/kernel/debug
echo -n "file qcom_geni_serial.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file qcom-geni-se.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file serial_core.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file gpi.c +p" > /sys/kernel/debug/dynamic_debug/control
```

3.10 UART examples

For information about the upstream device tree reference, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

For information about device-tree node for the Qualcomm Linux hardware SoCs, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

4 SPI

The serial peripheral interface (SPI) is a synchronous serial data link that operates in full duplex mode. SPI is also referred as a 4-wire serial bus.

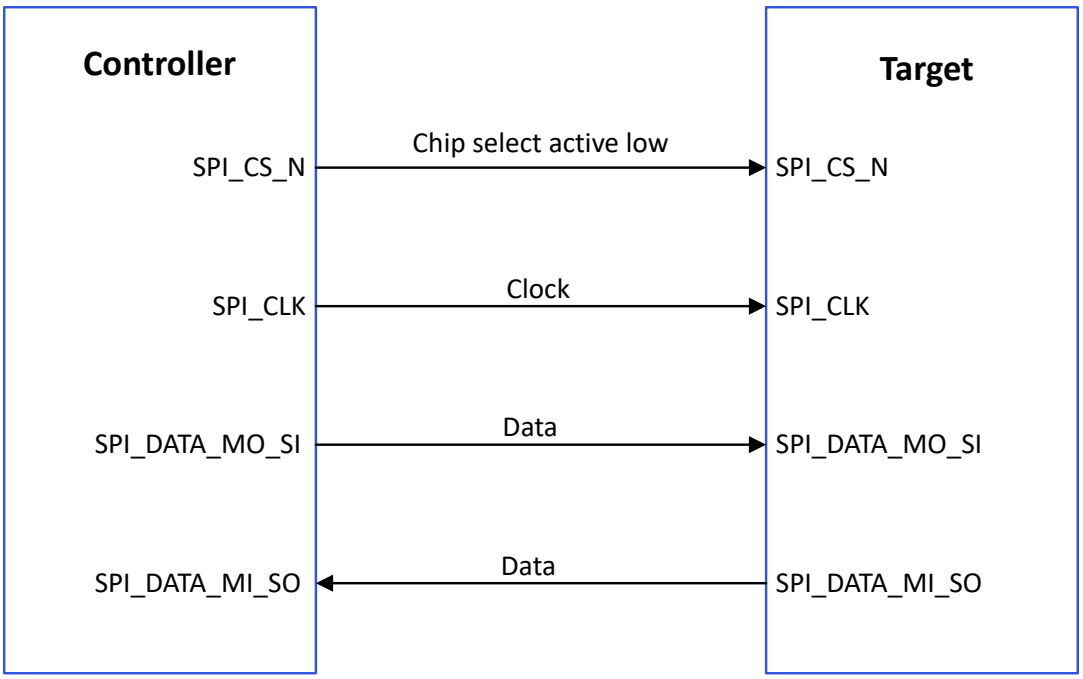


Figure 4-1 SPI data flow

The SPI core supports the bidirectional SPI standard, point-to-point, and controller-target protocol. The SPI core uses four chip-select lines (*SPI_CS#_N*) to select target devices for communication. The following two data lines support the bidirectional data transfer.

- SPI_DATA_MO_SI: Controller data output, target data input.
- SPI_DATA_MI_SO: Controller data input, target data output.

Table 4-1 Data and control signals in SPI

Data signals	MOSI: Controller data output, target data input.
	MISO: Controller data input, target data output.

Table 4-1 Data and control signals in SPI (cont.)

Control signals	SCLK: A clock generated by the controller and input to all targets.
	CS: Chip-select, a target is selected when the controller asserts its `CS_N` signal.

4.1 SPI features

This section explains the SPI serial engine transfer modes and the different scenarios where each mode is used. This section also describes the FIFO and DMA enabled in various subsystem SPI drivers.

Table 4-2 SPI transfer modes

Subsystem	Transfer mode	Description
Linux	■ FIFO (low speed) ■ CPU DMA (high speed) ■ GSI	Supports a maximum configuration speed of 50 MHz.
Boot	FIFO	■ Transfer rates up to 50 MHz. The host sets the SPI clock frequency nearest to the requested frequency. ■ 4 bits to 32 bits per word of transfer. ■ Maximum of 4 chip selects (CS) per bus. ■ GSI mode isn't supported on boot. ■ The driver executes in polling mode.
aDSP	■ Full duplex ■ Half duplex ■ Synchronous ■ Serial communication	■ There is no explicit communication framing, error checking, or defined data word length. ■ The communication is strictly at the raw bit level. ■ Transfer rates up to 50 MHz. The host sets the SPI clock frequency nearest to the requested frequency. ■ 4 bits to 32 bits per word of transfer. ■ Maximum of 4 chip selects (CS) per bus.

4.2 SPI interface

This section provides information about the subsystem drivers, kernel device tree nodes, and related documentation.

Table 4-3 SPI interface: Linux

File type	Description
Device tree source	■ QCS6490 and QCS5430:https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi ■ QCS9075:https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi ■ For information about QUP v3 serial engine device nodes, see the kernel documentation at https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/spi/qcom%2Cspi-geni-qcom.yaml
`Pinctrl` settings	■ QCS6490 and QCS5430:https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi ■ QCS9075:https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
Qualcomm TEE settings	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`

Table 4-4 SPI interface: Boot

File type	Description
QUP v3 serial engine configuration	■ QUP v3 serial engine: `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3.dtsi` ■ GPIO configurations: `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3-pinctrl.dtsi`
Qualcomm TEE settings	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`

Table 4-5 SPI interface: aDSP/SLPI/SDC

File type	Description
QUP v3 serial engine configuration	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_devcfg.c ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ▪ settings/buses/qup_common/config/<chipset> /adsp/ssc/qup_devcfg.json
Firmware configuration settings	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml

Table 4-6 SPI interface: Qualcomm TEE

File type	Description
QUP v3 serial engine configuration	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/spi/qupv3/config/<chipset>/tz/spi_devcfg_user.c ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/spi/qupv3/config/<chipset>/tz/spi_devcfg_user.h ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/spi/qupv3/config/<chipset>/tz/spi_devcfg.xml
Qualcomm TEE settings	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c

4.2.1 SPI APIs

SPI APIs for the following subsystems are listed in this section.

- Linux:
  - https://github.com/torvalds/linux/blob/master/include/uapi/linux/spi/spidev.h.
  - https://github.com/torvalds/linux/blob/master/include/linux/spi/spi.h.
- Boot: `boot_images/boot/QcomPkg/Include/SpiApi.h`
- aDSP/SDC: `adsp_proc/core/api/buses/spi_api.h`

4.3 SPI software

This section provides information on the SPI device tree configuration and documentation for the device nodes.

Linux

For device SPI details, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

For more information about kernel documentation specific to the SPI device nodes, see https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/spi/qcom%2Cspi-geni-qcom.yaml, and SPI driver file at https://github.com/torvalds/linux/blob/master/drivers/spi/spi-geni-qcom.c.

```
spi@a98000 {
compatible = "qcom,geni-spi";
reg = <0 0x00a98000 0 0x4000>;
clocks = <&gcc GCC_QUPV3_WRAP1_S6_CLK>;
clock-names = "se";
pinctrl-names = "default";
pinctrl-0 = <&qup_spi14_data_clk>, <&qup_spi14_cs>;
interrupts = <GIC_SPI 368 IRQ_TYPE_LEVEL_HIGH>;
#address-cells = <1>;
#size-cells = <0>;
power-domains = <&rpmhpd SC7280_CX>;
operating-points-v2 = <&qup_opp_table>;
interconnects = <&clk_virt MASTER_QUP_CORE_1 0 &clk_virt SLAVE_QUP_CORE_1 0>,
        <&gem_noc MASTER_APPSS_PROC 0 &cnoc2 SLAVE_QUP_1 0>;
interconnect-names = "qup-core", "qup-config";
        dmas = <&gpi_dma1 0 6 QCOM_GPI_SPI>,
           <&gpi_dma1 1 6 QCOM_GPI_SPI>;
dma-names = "tx", "rx";
status = "disabled";
};
```

For kernel documentation specific to the GPIO `pinctrl` configuration, see the following files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
- `Documentation/devicetree/bindings/pinctrl/qcom,<chipset>-tlmm.yaml`

The corresponding configurations of the QUP v3 serial engine GPIOs are present and mapped in the `pinctrl.dtsi`.

For example:

```
qup_spi14_data_clk: qup-spi14-data-clk-state {
                pins = "gpio56", "gpio57", "gpio58";
                function = "qup16";
            };

            qup_spi14_cs: qup-spi14-cs-state {
```

```
            pins = "gpio59";
            function = "qup16";
        };

        qup_spi14_cs_gpio: qup-spi14-cs-gpio-state {
            pins = "gpio59";
            function = "gpio";
        };

        qup_spi15_data_clk: qup-spi15-data-clk-state {
            pins = "gpio60", "gpio61", "gpio62";
            function = "qup17";
        };
```

Ensure that the protocol configuration for a particular serial engine uses the SPI protocol in the file at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`. Modify the required settings if needed or see the default settings assigned for QUP v3 serial engine instances.

Following is the sample configuration for enabling the SPI.

```
{ QUPV3_0_SE3, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // CAN SPI
{ QUPV3_1_SE3, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_FIFO, AC_HLOS,
FALSE, TRUE,  TRUE }, // LS1 SPI
{ QUPV3_1_SE4, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI,  AC_TZ,
FALSE, TRUE,  TRUE }, // SPI -NFC ESE
{ QUPV3_1_SE6, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI,  AC_HLOS,
FALSE, TRUE,  FALSE}, // FP
{ QUPV3_SSC_SE2, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI, AC_ADSP_Q6_ELF,
FALSE,      FALSE, FALSE }, // IMU_SPI
```

Boot

Configure the QUP v3 settings according to the Qualcomm Linux chip product requirements in the `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3.dtsi` file.

NOTE To enable the required QUP v3 serial engine, update the QUP v3 wrapper node status to `okay`, and the respective serial engine node to `disabled` state.

The following sample nodes help to configure the QUP v3 serial engine in boot.

QUP wrapper node sample

```
       /* QUPV3_0  wrapper  instance */
    TOP_QUP_0{
        compatible = "qcom,qup-controller";
        qup_id                      = /bits/  8 <(QUP_0)>;
```

```
        core_base_addr              = <QUPV3_0_CORE_BASE_ADDRESS>;
        common_base_addr            = <(QUPV3_0_CORE_COMMON_BASE_ADDRESS)>;
        se_wrapper_base_addr        = <(QUPV3_0_CORE_SE_BASE_ADDRESS)>;
        core_frequency              = <100000000>;
        qup_flags                   = <(QUP_FLAGS_UNUSED)>;
        num_se                      = /bits/ 8 <8>;
        status                      = "okay";
```

Sample node of serial engine instance

```
/*TOP_QUP_0_SE_0  Instance */
        TOP_QUP_0_SE_0{
          status                        = "disabled";  (updated to okay if you
need the instance to be enabled)
            core_offset                   = <0x00000000>;
            se_flags                      = <(USES_DDR_BUFFER |
USES_INTERNAL_DDR_MEM | ENABLE_FATAL_ON_TIMEOUT | POLLED_MODE)>;
            se_index                      = /bits/  8 <0>;
            FIFO_MODE                     = /bits/  8 <1>;
            protocol_supported            = <(I2C_SUPPORTED | UART_SUPPORTED |
SPI_SUPPORTED)>;
            interface_supported           = <CORE_IRQ>;
           * gpi_index                     = /bits/ 8 <0xFF>;
            core_irq                      = /bits/ 16 <0>;
            pdc_irq                       = /bits/ 16 <0>;
            gpio_int_num                  = /bits/ 16 <0>;
            i2c_hub                       = /bits/ 8  <0>;
            i2c_mm                        = /bits/ 16 <0>;
            SE_EXCLUSIVE                  = /bits/  8 <1>;
            pinctrl-names                 = "i2c-default", "i2c-sleep","spi-
default", "spi-sleep","uart-default", "uart-sleep";
            pinctrl-0                     = <&top_qup0_se0_i2c_active>;
            pinctrl-1                     = <&top_qup0_se0_i2c_sleep>;
            pinctrl-2                     = <&top_qup0_se0_spi_active>;
            pinctrl-3                     = <&top_qup0_se0_spi_sleep>;
            pinctrl-4                     = <&top_qup0_se0_uart_active>;
            pinctrl-5                     = <&top_qup0_se0_uart_sleep>;

            se_clock                      = "gcc_qupv3_wrap0_s0_clk";
        };
```

For `pinctrl` definitions, see the GPIO configuration file at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3-pinctrl.dtsi`.

Sample GPIO configuration:

```
#define top_qup0_se1_i2c_active_cfg
GPIO_CFG(GPIO_INPUT,GPIO_PULL_UP,GPIO_DRIVE_STRENGTH(200),GPIO_STRONG_PULL)
```

- In <chipset>-qupv3-pinctrl.dtsi modify the macro with the GPIO configuration when the requirements are different from the default configuration.
- For the macro definition, see the header file path at Settings/Include/gpio-dt.h.

Replace macro in sample TLMM node.

```
/*TOP_QUP0_se1_pinctrl*/
  top_qup0_se1_i2c_active: top_qup0_se1_i2c_active{
  --    config = <&qup0_l0_1 top_qup_i2c_active_cfg>,
              <&qup0_l1_1 top_qup_i2c_active_cfg>;
 ++  config = <&qup0_l0_1 top_qup0_se1_i2c_active_cfg>,
              <&qup0_l1_1 top_qup0_se1_i2c_active_cfg>;
  };
```

NOTE Verify Qualcomm TEE settings before changing the unified extensible firmware interface (UEFI) configuration. Ensure that the serial engine node is in FIFO_MODE and accessible from the application processor. Verify that the loaded protocol is according to the requirement.

aDSP/SDC

Firmware loading with SSC QUP is performed during the bootup sequence of the aDSP subsystem. The configuration file is present in the aDSP build at /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c and /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml.

The following configuration is a sample of the SSC QUP SE4 loaded with the SPI firmware.

```
se_cfg se4_cfg = { 0x90000, SE_PROTOCOL_SPI,       GSI,     TRUE, TRUE };
```

GPIO configuration: Each serial engine in the QUP common driver is configured with the default GPIO configuration. The GPIO configuration is picked up by the QUP common driver according to the protocol loaded for the serial engine at /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_instance_mapping.c.

The default GPIO configuration can be overwritten as follows.

```
{       .instance_id          =  5 ,         //Instance ID
         .qup              =  QUP_SSC,    //QUP Type
         .se_index         =  4,          //SE ID
         .se_data          =  NULL,       //devcfg_map
         .protocol_io_cfg  =  {

TLMM_MAP(TLMM_GPIO_KEEPER ,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER ),              //
SLEEP CFG

TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_6MA,TLMM_GPIO_KEEPER ),              //SP
I CFG

TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL),              //
UART CFG
```

```
TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL),              //I2
C CFG

TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER )               //I3
C CFG
                              },
        .se_exclusive     =  TRUE,
}
```

TLMM_MAP is a macro to initialize the active and sleep state GPIO configurations. For example, sample usage of the TLMM_MAP macro.

```
TLMM_MAP (active state pull type, drive strength, sleep state pull type)
```

Qualcomm TEE

The QUP v3 access configuration settings are at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/spi/qupv3/interface/spi_devcfg.h`.

To enable the QUP v3 serial engine for SPI in `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/spi/qupv3/config/<chipset>/tz/spi_devcfg_user.h`, add `#define TZ_USE_SPI_X` (*X* is the SPI serial engine number) as follows.

```
#define TZ_USE_SPI_13 //NFC-ese

#define TZ_USE_SPI_14 //Touch-SPI

#define TZ_USE_SPI_15 //FP sensor
```

The `TZ_USE_SPI_<num>` number is based on the serial number of the serial engine (starting from one). For example, if there are two QUPs: QUPV3_0 with seven serial engines and QUPV3_1 with eight serial engines, the user must enable QUPV3_2_SE2. The macro should be TZ_USE_SPI_9.

GPIO configuration: In the following example, drive strength and pull are configured according to the PIN starting index from MISO at `settings/buses/spi/qupv3/config/<chipset>/tz/spi_devcfg_user.c`.

```
spi_plat_device_config_user spi_device_user_config_0 =

    {2,2,2,2,-1,-1,-1},    //.drive_strength index: 0 - MISO, 1 - MOSI, 2 -
SCLK, 3 - CS_0, 4- CS_1, 5- CS_2, 6- CS_3
                                                    value: 0 - 2MA, 1 - 4MA,
2 - 6MA
    {1,1,0,1,-1,-1,-1},    //.pull index: 0 - MISO, 1 - MOSI, 2 - SCLK, 3 -
CS_0, 4- CS_1, 5- CS_2, 6- CS_3
                               value: 0 - NO_PULL, 1 = PULL_DOWN, 2 =
KEEPER, 3 = PULL_UP
    0xFF,                 //.gpii_idx
    0,                   //.mode_select  not supported for TZ
    0,                   //.flags not supported for TZ
};
```

Access control permission to the Qualcomm TEE subsystem is configured in the QUPAC policy at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`.

```
const QUPv3_se_security_permissions_type qupv3_perms_iot_rb3[] =
{
  /*   PeriphID,         ProtocolID,               Mode,  NsOwner,
bAllowFifo, bLoad, bModExcl  */
  { QUPV3_0_SE0, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // LT9611 and QPS615 I2C
  { QUPV3_0_SE1, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // APPS I2C - PCIE/ USB Type C
  { QUPV3_0_SE2, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // SMB / LS1 I2C
  { QUPV3_0_SE3, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // CAN SPI
  { QUPV3_0_SE4, QUPV3_PROTOCOL_UART_4W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // LS1 UART
  { QUPV3_0_SE5, QUPV3_PROTOCOL_UART_2W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  FALSE, FALSE }, // Debug UART
  { QUPV3_0_SE6, QUPV3_PROTOCOL_UART_2W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // WLAN UART
  { QUPV3_0_SE7, QUPV3_PROTOCOL_UART_4W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // Hastings BT
  { QUPV3_1_SE0, QUPV3_PROTOCOL_SPMI,    QUPV3_MODE_FIFO, AC_ADSP_Q6_ELF,
TRUE,  TRUE,  FALSE }, // QuP SPMI
  { QUPV3_1_SE1, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // NFC I2C
  { QUPV3_1_SE2, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // HDMI OUT for VIDEOIOBoard
  { QUPV3_1_SE3, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_FIFO, AC_HLOS,
FALSE, TRUE,  TRUE }, // LS1 SPI
  { QUPV3_1_SE4, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI,  AC_TZ,
FALSE, TRUE,  TRUE }, // SPI -NFC ESE
  { QUPV3_1_SE5, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_GSI,  AC_HLOS,
FALSE, TRUE,  FALSE}, // Legacy Touch
  { QUPV3_1_SE6, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI,  AC_HLOS,
FALSE, TRUE,  FALSE}, // FP
  /*QUPV3_1_SE7*/
};
```

4.4 SPI bringup

This section describes how to enable the Qualcomm Linux SPI drivers.

Linux

To verify SPI communication with the device, kernel configuration must be enabled. The `CONFIG_SPI_SPIDEV=m` setting is enabled in the corresponding `<chipset> defconfig` file. Enable the specific QUP v3 SPI serial engine instance in the kernel device tree.

4.5 SPI configuration

This section provides information about the SPI software driver kernel configuration and device tree node changes.

Linux

The following driver kernel configurations are required to support the SPI interface.

- Driver source file at https://github.com/torvalds/linux/blob/master/drivers/spi/spi-geni-qcom.c
- Kernel `defconfig` file at `<workspace_path_of_LINUX_kernel_image>/sources/kernel /kernel_platform/kernel/arch/arm64/configs/qcom_defconfig`

The following kernel configurations must be enabled.

- `CONFIG_QCOM_GENI_SE=y`
- `CONFIG_SPI_QCOM_GENI=m`
- `CONFIG_SPI_SPIDEV=m` to configure user space applications
- `CONFIG_QCOM_GPI_DMA=m` to enable GSI support

To enable the SPI node for the loopback validation, apply the following patch to the `/arch/arm64/boot/dts/qcom/<chipset>.dtsi` file.

```
diff --git a/arch/arm64/boot/dts/qcom/<chipset>.dtsi b/arch/arm64/boot/dts/
qcom/<chipset>.dtsi
index 82dfa3e..344e99a 100644
--- a/arch/arm64/boot/dts/qcom/<chipset>.dtsi
+++ b/arch/arm64/boot/dts/qcom/<chipset>.dtsi
@@ -6760,3 +6760,12 @@
<GIC_PPI 10 IRQ_TYPE_LEVEL_LOW>;
 };
 };
+
+&spi14 {
+        status = "ok";
+        spidev@0 {
+                compatible = "spidev";
+                spi-max-frequency = <50000000>;
+                reg = <0>;
+        };
+};

diff --git a/drivers/spi/spidev.c b/drivers/spi/spidev.c
index d13dc15..36d7914 100644
```

```
--- a/drivers/spi/spidev.c
+++ b/drivers/spi/spidev.c
@@ -84,7 +84,7 @@
static LIST_HEAD(device_list);
static DEFINE_MUTEX(device_list_lock);

-static unsigned bufsiz = 4096;
+static unsigned bufsiz = 35000;
module_param(bufsiz, uint, S_IRUGO);
MODULE_PARM_DESC(bufsiz, "data bytes in biggest supported SPI message");

@@ -742,6 +742,7 @@
     { .compatible = "semtech,sx1301", .data = &spidev_of_check },
     { .compatible = "silabs,em3581", .data = &spidev_of_check },
     { .compatible = "silabs,si3210", .data = &spidev_of_check },
+    { .compatible = "spidev"},
{},
 };
 MODULE_DEVICE_TABLE(of, spidev_dt_ids);
```

NOTE You should compile the kernel configuration and device tree changes. After the kernel is compiled, you can load the images to the device to verify the interface. For information about interface verification, see the SPI verification section.

4.6 SPI verification

This section describes the validation procedure and test results for the SPI drivers.

Linux

To cross-compile the SPI tools, do the following.

1. Access the SPI tool from `yocto/build-qcom-wayland/tmp-glibc/work-shared/<chipset>/kernel-source/tools/spi`. For more details about the SPI tool, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/tools/spi.
2. Install cross compiler.

   ```
   sudo apt-get install gcc-aarch64-linux-gnu
   ```

3. Set up the environment for cross-compilation by running the following command.

   ```
   export ARCH=arm64export CROSS_COMPILE=aarch64-linux-gnu-
   ```

4. Compile the tool by running the following command.

   ```
   make
   ```

Verify SPI device

Verify the driver by checking for `dev` node (`/dev/spidev1.0`) in the SSH shell or use the ADB shell. For more information about how to run SSH, see the Use SSH section.

1. To verify the SPI driver, do the following:
   a. Open the SSH shell in permissive mode or use the ADB shell.
   b. Mount the file system.

      ```
      mount -o remount,rw /usr
      ```

   c. Transfer files using SCP or similar tools.

      For example, `scp spidev_test root@10.92.175.138:/bin`.

   d. Assign permission to execute.

      ```
      chmod 777 spidev_test
      ```

2. Verify an SPI device. Command format. `/spidev_test -D /dev/<spidev_node>`.

   ```
   ./spidev_test -D /dev/spidev1.0
   ./spidev_test -D /dev/spidev3.0
   ```

   The following output is displayed.

   ```
   spi mode: 0x0
   bits per word: 8
   max speed: 500000 Hz (500 KHz)
   ```

Run the following command for information about usage.

```
./spidev_test -help
```

The following output is displayed.

```
-D --device   device to use (default /dev/spidev1.1)
-s --speed    max speed (Hz)
-d --delay    delay (usec)
-b --bpw      bits per word
-i --input    input data from a file (e.g. "test.bin")
-o --output   output data to a file (e.g. "results.bin")
-l --loop     loopback
-H --cpha     clock phase
-O --cpol     clock polarity
-L --lsb      least significant bit first
-C --cs-high  chip select active high
-3 --3wire    SI/SO signals shared
-v --verbose  Verbose (show tx buffer)
-p  Send data (e.g. "1234\xde\xad")
-N --no-cs    no chip select
  -R --ready    slave pulls low to pause
  -2 --dual     dual transfer
  -4 --quad     quad transfer
  -8 --octal    octal transfer
  -S --size     transfer size
  -I --iter     iterations
```

4.7 SPI debugging

This section describes the default logging method of the SPI software driver to enable logging the SPI transfer failures.

Linux

The SPI driver logs are enabled through a dynamic debugging method. Enable `CONFIG_DYNAMIC_DEBUG` in `<workspace_path_of_LINUX_kernel_image>/sources/kernel/kernel_platform/kernel/arch/arm64/configs/qcom_defconfig` to support the dynamic debugging of the kernel drivers.

To enable and view the SPI driver logs in the kernel logs (`dmesg`), run the following command.

```
mount -t debugfs none /sys/kernel/debug
echo -n "file spi-geni-qcom.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file qcom-geni-se.c  +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file spidev.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file gpi.c +p" > /sys/kernel/debug/dynamic_debug/control
```

4.8 SPI examples

For information about the upstream device tree reference, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

For information about device-tree node for the Qualcomm Linux hardware SoCs, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

5 I2C

Interintegrated circuit (I2C) is a bidirectional 2-wire bus for an efficient inter-IC control bus developed by Philips in the 1980s. Every device on the bus has its own unique address (registered with the I2C general body headed by Philips). The I2C core supports a multicontroller mode and 10-bit target address and 10-bit extendable address. For more information about I2C, see https://www.i2c-bus.org/fileadmin/ftp/i2c_bus_specification_1995.pdf.

I2C communication sequence overview

The following figure shows the communication sequence between the controller and targets in I2C.

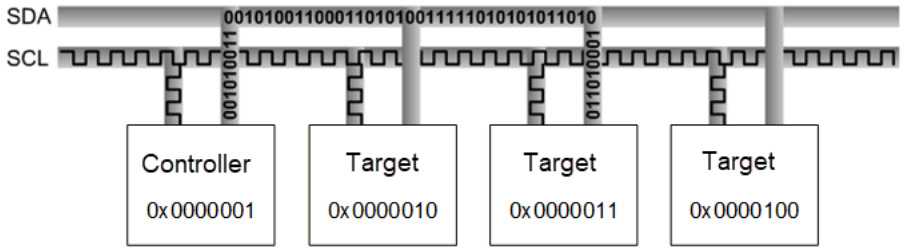


Figure 5-1 I2C sequence

For example, no device can use 1111-0XX listed in the I2C specification and high-speed mode with 3.4 MHz clock frequency.

Following are the I2C modes and supported speeds.

- Standard mode: 100 kbps
- Fast mode: 400 kbps
- Fast mode plus: 1 Mbps

The maximum supported bandwidth is 1 MHz.

I2C data packet format

The controller sends a 7-bit or 10-bit address as shown in the following figure. Along with the address, a 1-bit read/write indicating the type of operation is also sent. Data is transferred in sequences of 8 bits placed on an SDA line. For every byte transferred, the device receiving the data sends back an ACK bit (totaling nine clock pulses).

- ACK bit LOW: receives the data and is ready to accept the next byte.
- ACK bit HIGH: receives the data and can't accept further data. The controller then terminates the transmission with the STOP sequence.

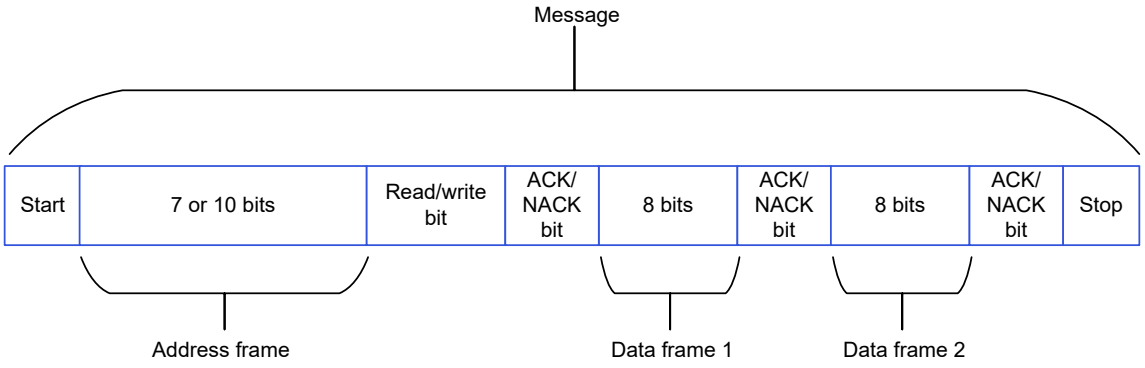
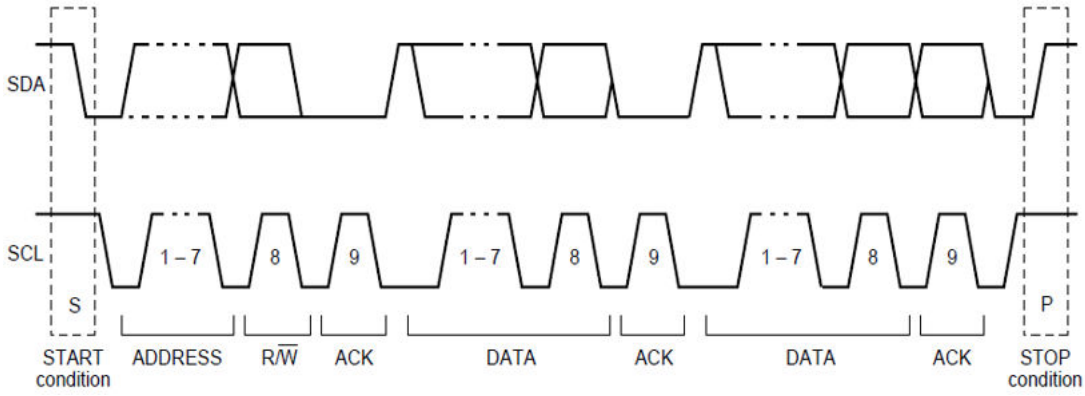


Figure 5-2 I2C data packet

I2C sequences

When the SCL is high, the SDA must remain stable and can't change. Only when the clock line is low can the data line change. However, there are two exceptions: START and STOP sequences.



5.1 I2C features

This section explains the I2C serial engine transfer modes and the different scenarios where each mode is used. The following table lists the transfer modes enabled in the various I2C subsystem drivers.

Table 5-1 I2C transfer modes

Subsystem	Transfer mode	Description
Linux	▪ FIFO (low speed) ▪ CPU DMA (high speed) ▪ GSI	▪ Supports 100 kHz, 400 kHz, and 1000 kHz bus speeds ▪ Supports 7-bit target address according to the I2C specification
Boot	FIFO	
aDSP/Qualcomm TEE/SDC	FIFO	▪ Supports 100 kHz, 400 kHz, and 1000 kHz bus speeds ▪ Supports 7-bit target address according to the I2C specification

5.2 I2C interface

This section provides information about the subsystem driver, kernel device tree nodes, and related documentation.

Table 5-2 I2C interface: Linux

File type	Description
Device tree source	■ QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi ■ QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
`Pinctrl` settings	■ QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi ■ QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
Qualcomm TEE settings	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c` .

Table 5-3 I2C interface: Boot

File type	Description
QUP v3 serial engine configuration	■ QUP v3 serial engine:`/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3.dtsi` ■ GPIO configurations:`/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3-pinctrl.dtsi`
Qualcomm TEE settings	■ `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`

Table 5-4 I2C interface: aDSP/SLPI/SDC

File type	Description
QUP v3 serial engine configuration	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_devcfg.c ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ▪ settings/buses/qup_common/config/<chipset> /adsp/ssc/qup_devcfg.json
Firmware configuration settings	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml

Table 5-5 I2C interface: Qualcomm TEE

File type	Description
QUP v3 serial engine configuration	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/i2c/qupv3/config/<chipset>/tz/i2c_devcfg_user.h ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/i2c/qupv3/config/<chipset>/tz/i2c_devcfg_user.c ▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/i2c/qupv3/config/<chipset>/tz/i2c_devcfg.xml
Qualcomm TEE settings	▪ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c

5.2.1 I2C APIs

I2C APIs for the following subsystems are listed in this section.

- Linux:
  - https://github.com/torvalds/linux/blob/master/include/linux/i2c.h.
  - https://github.com/torvalds/linux/blob/master/include/linux/i2c-dev.h.
- Boot: `boot_images/boot/QcomPkg/Include/i2c_api.h`
- aDSP/SDC/SLPI: `adsp_proc/core/api/buses/i2c_api.h`
- Qualcomm TEE: `trustzone_images/core/buses/api/i2c/qupv3/i2c_api.h`

5.3 I2C software

This section provides information on the I2C device tree configuration, and documentation for the device nodes.

Linux

For the configuration settings file, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

For more details, see the `i2c-geni-qcom.yaml` file at https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/i2c/qcom%2Ci2c-geni-qcom.yaml, and the I2C driver at the https://github.com/torvalds/linux/blob/master/drivers/i2c/busses/i2c-qcom-geni.c files.

```
i2c1: i2c@984000 {
compatible = "qcom,geni-i2c";
reg = <0 0x00984000 0 0x4000>;
clocks = <&gcc GCC_QUPV3_WRAP0_S1_CLK>;
clock-names = "se";
pinctrl-names = "default";
pinctrl-0 = <&qup_i2c1_data_clk>;
interrupts = <GIC_SPI 602 IRQ_TYPE_LEVEL_HIGH>;
#address-cells = <1>;
#size-cells = <0>;
interconnects = <&clk_virt MASTER_QUP_CORE_0 0 &clk_virt SLAVE_QUP_CORE_0 0>,
        <&gem_noc MASTER_APPSS_PROC 0 &cnoc2 SLAVE_QUP_0 0>,
        <&aggre1_noc MASTER_QUP_0 0 &mc_virt SLAVE_EBI1 0>;
interconnect-names = "qup-core", "qup-config",
        "qup-memory";
power-domains = <&rpmhpd SC7280_CX>;
required-opps = <&rpmhpd_opp_low_svs>;
            dmas = <&gpi_dma0 0 1 QCOM_GPI_I2C>,
         <&gpi_dma0 1 1 QCOM_GPI_I2C>;
dma-names = "tx", "rx";
status = "disabled";
            };
```

For kernel documentation specific to the GPIO `pinctrl` configuration, see the following files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
- `Documentation/devicetree/bindings/pinctrl/qcom,<chipset>-tlmm.yaml`

The corresponding configurations of the QUP v3 serial engine GPIOs are present and mapped in the `pinctrl.dtsi` file.

```
qup_i2c1_data_clk: qup-i2c1-data-clk-state {
                pins = "gpio4", "gpio5";
                function = "qup01";
            };
```

In the `QUPAC_Access.c` file, ensure that the particular serial engine configuration for the specified protocol is for the I2C protocol. Qualcomm TEE build: `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/`

buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c. Modify the required settings or refer to the default settings assigned for the QUP v3 serial engine instances. The following sample configuration is enabled by default in I2C.

```
/*   PeriphID,          ProtocolID,                Mode,  NsOwner,
bAllowFifo, bLoad, bModExcl  */
  { QUPV3_0_SE0, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // LT9611 and QPS615 I2C
  { QUPV3_0_SE1, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // APPS I2C - PCIE/ USB Type C
  { QUPV3_0_SE2, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // SMB / LS1 I2C

{ QUPV3_1_SE1, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // NFC I2C
 { QUPV3_1_SE2, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // HDMI OUT for
 { QUPV3_1_SE5, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_GSI,  AC_HLOS,
FALSE, TRUE,  FALSE}, // Legacy Touch
```

Boot

1. Configure I2C in the UEFI. The configuration files can be accessed from the following locations.
   - QUP v3 serial engine:/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/BOOT.MXF.1.0.c1/boot_images/boot/Settings/Soc/<chipset>/Core/Buses/qup_common/<chipset>-qupv3.dtsi
   - Qualcomm TEE settings:/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c

   NOTE For configuration settings in boot, see Boot.
2. Enable the I2C protocol in UEFI at/QcomPkg/SocPkg/<chipset>/LAA/Core.fdf.

   ```
   -#INF QcomPkg/Drivers/I2CDxe/I2CDxe.inf
   +INF QcomPkg/Drivers/I2CDxe/I2CDxe.inf
   ```
3. The application enables the I2C interface. The application then performs the read and write operation through the I2C interface. For information about I2C function usage, see boot_images/QcomPkg/QcomTestPkg/I2CApp/I2Ceeprom.c.
4. Add i2c_open->i2c_read/i2c_write->i2c_close sequentially in code.
5. Ensure that the GpiDxe.inf and I2C.efi files are loaded by verifying the device/UEFI bootup logs before you call I2c_open.

aDSP/SDC

Firmware loading with SSC QUP is performed during the bootup sequence of the aDSP subsystem. The configuration files are present in the aDSP build at /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c and/firmware/qualcomm-linux-

spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml.

The following configuration is a sample of SSC QUP SE1, SE2, and SE3 settings loaded with the I2C firmware.

```
se_cfg se1_cfg = { 0x84000, SE_PROTOCOL_I2C,    GSI,     TRUE, TRUE  };
se_cfg se2_cfg = { 0x88000, SE_PROTOCOL_I2C,    GSI,     TRUE, TRUE  };
se_cfg se3_cfg = { 0x8C000, SE_PROTOCOL_I2C,    GSI,     FALSE, TRUE  };
```

GPIO configuration: Each serial engine in the QUP v3 common driver is configured with the default GPIO configuration. The GPIO configuration is picked up by the QUP v3 common driver according to the protocol loaded in the serial engine. File path: /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_instance_mapping.c. The default GPIO configuration can be overwritten as follows.

```
{       .instance_id          =  5 ,          //Instance ID
         .qup              =  QUP_SSC,     //QUP Type
         .se_index         =  4,           //SE ID
         .se_data          =  NULL,        //devcfg_map
         .protocol_io_cfg  =  {

TLMM_MAP(TLMM_GPIO_KEEPER ,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER ),              //
SLEEP CFG

TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_6MA,TLMM_GPIO_KEEPER ),              //SP
I CFG

TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL),              //
UART CFG

TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL),              //I2
C CFG

TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER )               //I3
C CFG
                              },
         .se_exclusive     =  TRUE,
}
```

TLMM_MAP is a macro to initialize the active and sleep state GPIO configurations. For example, sample usage of the TLMM_MAP macro.

```
TLMM_MAP (active state pull type, drive strength, sleep state pull type)
```

Qualcomm TEE

The QUP v3 serial engine for I2C can be configured as follows. File path: settings/buses/i2c/qupv3/config/<chipset>/tz/i2c_devcfg_user.h.

```
#define ENABLE_I2C_08
```

The ENABLE_I2C_<num> number is based on the serial number of the serial engine (starting from 0). For example, if there are two QUPs: QUPV3_0 with seven serial engines and QUPV3_1 with eight serial engines, then the user must enable QUPV3_2_SE2. The macro should be ENABLE_I2C_08.

GPIO configuration: drive strength and pull are configured per PIN for SDA at zero index and SCL at one index. File path: settings/buses/i2c/qupv3/config/<chipset>/tz/i2c_devcfg_user.c

```
i2c_plat_device_config_user i2c_device_user_config_0 =
{
    {0,0},        //.drive_strength index: 0 - SDA, 1 - SCL
                                  value: 0 - 2MA, 1 - 4MA, 2 - 6MA
    {3,3},       //.pull  index: 0 - SDA, 1 - SCL
                       value: 0 - NO_PULL, 1 = PULL_DOWN, 2 = KEEPER, 3 =
PULL_UP
    0xFF,        //.gpii_idx
    0,           //.mode_select  not supported for TZ
    0,           //.flags not supported for TZ
};
```

QUPAC access control: controls the firmware loading and access control permission from the Qualcomm TEE subsystem are configured in the QUPAC access file at /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c.

5.4 Configure I2C interface

This section provides information about the I2C software driver kernel configuration and device tree node changes.

Linux

The following driver kernel configurations are required to support the I2C interface.

- Driver source: https://github.com/torvalds/linux/blob/master/drivers/i2c/busses/i2c-qcom-geni.c
- Kernel defconfig file path: <workspace_path_of_LINUX_kernel_image>/sources/kernel/kernel_platform/kernel/arch/arm64/configs/qcom_defconfig

The following kernel configurations are to be enabled.

- CONFIG_QCOM_GENI_SE=y
- CONFIG_I2C_CHARDEV=m
- CONFIG_I2C_QCOM_GENI=m to configure user space applications
- CONFIG_QCOM_GPI_DMA=m to enable GSI support

To enable the I2C DT node for validations, apply the following patch to the /arch/arm64/boot/dts/qcom/<chipset>.dtsi file.

```
diff --git a/arch/arm64/boot/dts/qcom/<chipset>.dtsi b/arch/arm64/boot/dts/
qcom/<chipset>.dtsi
index 8575f0b..cced7c0 100644
--- a/arch/arm64/boot/dts/qcom/<chipset>.dtsi
```

```
+++ b/arch/arm64/boot/dts/qcom/<chipset>.dtsi
@@ -6865,3 +6865,7 @@
 <GIC_PPI 10 IRQ_TYPE_LEVEL_LOW>;
 };
 };
+
+ &i2c1 {
+ status = "ok";
+};
```

NOTE You should compile the kernel configuration and device tree changes. After compilation, you can load the images to the device to verify the interface. For information about interface verification, see the Verify I2C interface section.

5.5 Verify I2C interface

This section describes the validation procedure and test results for the I2C drivers and the Qualcomm drivers.

Linux

For upstream I2C kernel test applications, see https://cdn.kernel.org/pub/software/utils/i2c-tools/i2c-tools-4.3.tar.gz.

To cross-compile tools, do the following.

1. Download `i2c-tool` from https://cdn.kernel.org/pub/software/utils/i2c-tools/i2c-tools-4.3.tar.gz.
2. Extract the tool from the downloaded `tar` file.

   ```
   tar -xzvf <i2c-tool-path>
   ```

3. Change the current directory to the `i2c-tool` path.

   ```
   cd <i2c-tool-path>
   ```

4. Install the tool.

   ```
   sudo apt-get install gcc-aarch64-linux-gnu
   ```

5. Set up the environment for cross-compilation.

   ```
   export CC=aarch64-linux-gnu-gcc
   ```

6. Compile the tool.

   ```
   make USE_STATIC_LIB=1
   ```

   The binary is generated at `<i2c-tool-path>/tools/`.

Validate the driver by checking for `dev` node (`/dev/i2c-0` and `/dev/i2c-1`) in the SSH shell or use the ADB shell. For more information about how to run SSH, see the Use SSH section.

Verify I2C driver

1. To verify the I2C driver, do the following:
   a. Open the SSH shell in permissive mode or use the ADB shell.
   b. Mount the file system.

      ```
      mount -o remount,rw /usr
      ```

   c. Transfer files using SCP or similar tools.

      For example, `scp i2cdetect root@10.92.162.185:/bin`

   d. Assign permission to execute.

      ```
      chmod 777 i2cdetect
      ```

2. Verify an I2C device with the `i2cdetect` tool. For example, `./i2cdetect -y -r <i2c_instance_num>`.

   ```
   /lib # ./i2cdetect -y -r 0
   ```

   The following output is displayed.

   ```
   0 1 2 3 4 5 6 7 8 9 a b c d e f 00: -- -- -- -- -- -- -- -- 10: -- -- --
   -- -- -- -- -- -- -- -- -- -- -- -- -- 20: -- -- -- -- -- -- -- -- --
   -- -- 2b -- -- -- -- 30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
   -- 40: -- -- -- -- 44 -- -- -- -- -- -- -- -- -- -- -- 50: -- -- -- --
   -- -- -- -- -- -- -- -- -- -- -- -- 60: -- -- -- -- -- -- -- -- -- --
   -- -- -- -- -- -- 70: -- -- -- -- -- -- -- -- /lib # exit
   ```

View I2C detect help

For more information about I2C detection, and how to use it, run the following commands.

```
/lib # ./i2cdetect --help
```

The following output is displayed.

```
Error: Unsupported option "--help"! Usage: i2cdetect [-y] [-a] [-q|-r]
I2CBUS [FIRST LAST] i2cdetect -F I2CBUS i2cdetect -l I2CBUS is an integer
or an I2C bus name If provided, FIRST and LAST limit of the probing range.
```

Identify probed device in DUT

To identify the device probed from the DUT, run the following command.

```
/lib # ./i2cdump -r 0-0xff 0 0x2b b
```

Output:

```
 WARNING! This program can confuse your I2C bus, cause data loss and
worse! will probe file /dev/i2c-0, address 0x2b, mode byte Probe range
limited to 0x00-0xff. Continue? [Y/n] Y 0 1 2 3 4 5 6 7 8 9 a b c d e f
0123456789abcdef 00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ 90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00 ................ d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................ f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 ................
```

Read I2C data from device

To read I2C data from the device, run the following commands.

```
/lib # ./i2cget 0 0x2b 4 b
```

Output:

```
 WARNING! This program can confuse your I2C bus, cause data loss and
worse! Will read from device file /dev/i2c-0, chip address 0x2b, data
address 0x04, using read byte data. Continue? [Y/n] Y 0x00
```

5.6 Debug I2C issues

The section describes the default logging method of the I2C software driver to enable logging the I2C transfer failures.

Linux

The I2C driver logs are enabled through the kernel dynamic debugging method. Enable `CONFIG_DYNAMIC_DEBUG` in `<workspace_path_of_LINUX_kernel_image>/sources/kernel/kernel_platform/kernel/arch/arm64/configs/qcom_defconfig` to support the dynamic debugging of the kernel drivers.

To enable and view the I2C driver logs in the kernel logs (`dmesg`), run the following commands.

```
mount -t debugfs none /sys/kernel/debug
echo -n "file i2c-qcom-geni.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file i2c-core-base.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file i2c-dev.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file i2c-mux.c +p" > /sys/kernel/debug/dynamic_debug/control
echo -n "file gpi.c +p" > /sys/kernel/debug/dynamic_debug/control
```

To debug the error message: `[ 8.583248] geni_i2c a94000.i2c: Invalid proto 1` for driver protocol load failures, do the following.

1. Identify the board type.

   Example: QUPV3_1_SE5 board type details

   ```
   B -    461251 - CDT Version:3,Platform ID:34,Major ID:1,Minor
   ID:0,Subtype:2
   ```

2. From the kernel log, locate the platform ID and subtype related details at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.xml`
3. Obtain the Qualcomm TEE `QUPAC_Access.c` file configurations.
4. Identify the configuration specific to the serial engine.
5. Locate the platform ID type within the `QUPAC_Access.xml` file.

6. Map this platform ID type to the corresponding `qupv3_perms` structure within the `QUPAC_Access.c` file.
7. Verify the protocol and mode configurations in the `QUPAC_Access.c` file.

5.7 I2C examples

For information about the upstream device tree reference, see the following DTSI files.

- QCS6490 and QCS5430:https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075:https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

For information about device-tree node for the Qualcomm Linux hardware SoCs, see the following DTSI files.

- QCS6490 and QCS5430:https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
- QCS9075:https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

6 I3C

The improved interintegrated circuit (I3C) interface has been developed to provide a fast, low-cost, low-power, 2-wire digital interface for connected I3C devices. The I3C interface is intended to improve upon the features of the I2C interface, preserving backward compatibility.

The 2-wire serial interface supports up to 12.5 MHz. Legacy I2C devices co-exist on the same bus. The I3C bus supports in-band support, hot join, synchronous timing supports and asynchronous time stamping. During the write operation, the transition bit is of 8-bit data parity. During read, the controller uses the transition bit to either proceed with the next data or abort the read.

The data phase can be either push-pull or open-drain, based on the I3C capability of the device (for mixed bus, the I3C controller can address the I2C target).

- The address followed by the START condition is open-drain (arbitration phase).
- The address followed by a repeated START condition is push-pull.

The following figure illustrates the I3C hardware and software entities.

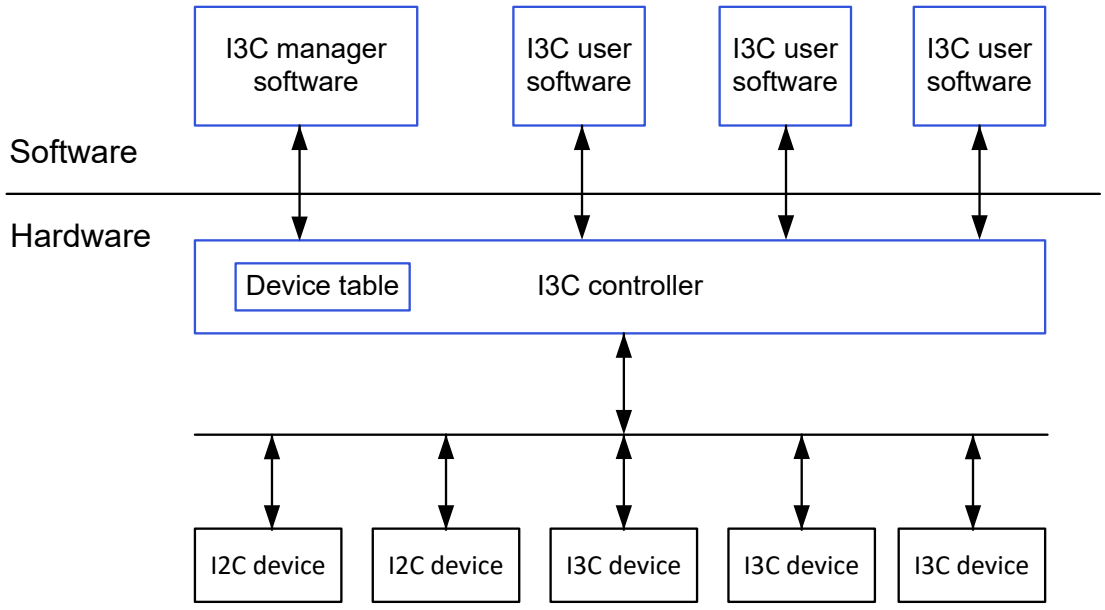


Figure 6-1 I3C block diagram

The main features of the I3C architecture are as follows:

- I3C controller driver software: software-privileged entity, capable of enumeration and bus configuration.
- I3C client software: software entities that use the I3C bus for read/write to devices.

I3C packet frame structure

A frame begins with START, command (8), data (8), transition (1), and STOP.

S or Sr	I3C dynamic client address	Read/ write	ACK	Data	T	P

Figure 6-2 I3C packet frame

Where,

- ACK: Acknowledge (SDA low)
- S: Start condition
- Sr: Repeat start condition
- P: Stop condition
- T: Transition bit alternative to ACK/NACK

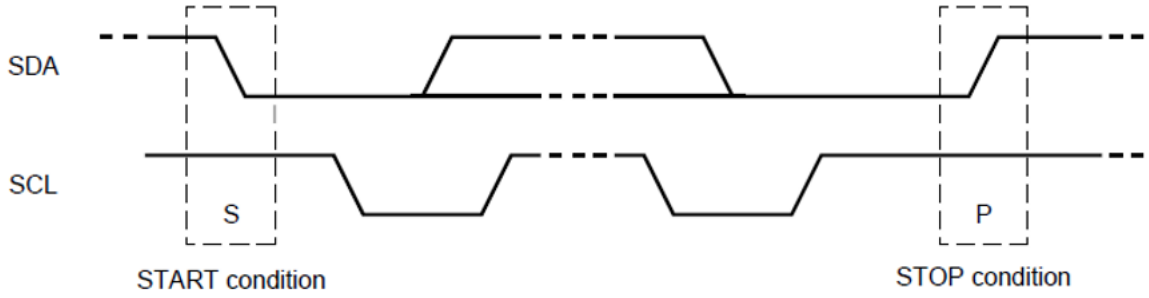


Figure 6-3 I3C sequence

I3C bus initialization

The I3C bus initialization procedure is as follows:

1. The I3C controller driver software initializes the I3C controller (firmware and configuration settings).
2. The I3C controller driver software reads the I3C configuration and the device database, including:
   - List of I2C static address devices.
   - List of I3C static address devices.

- List of expected I3C dynamic devices: vendor ID, device ID, predefined dynamic address, and associated drivers.
- Optional hot-join allowed devices (vendor ID, device ID, and predefined address).

3. Using the I3C controller software, the driver enumerates the devices on the bus (set local addresses).
4. The I3C controller driver software writes an I3C controller with the required devices and the bus characteristic information:
   - Operation frequency
   - Pure/legacy I2C mode
   - SDR/HDR enable/disable (and types)
   - IBI capable devices and expected data bytes
5. The I3C client software stack receives the following information about the I3C configuration and device database.
   - List of I2C, static address devices.
   - List of I3C, static address devices.
   - List of expected I3C dynamic devices: vendor ID, device ID, predefined dynamic address, and associated drivers.
   - Optional hot-join allowed devices (vendor ID, device ID, and predefined address).

6.1 I3C features

The aDSP/SDC I3C subsystem supports the following features.

- Address phase at frequency 400 kHz.
- Data phase at frequency 12,500 kHz.
- Backward compatibility with legacy I2C.
- Dynamic addressing of targets.
- Single data rate (SDR) mode.
- CCC commands according to the MIPI I3C specification

6.2 I3C interface

I3C use cases in the aDSP/SLPI software support sensor device communication.

NOTE I3C doesn't support Linux use cases.

The software driver and the configuration support for the I3C functionality for device communication are listed in the following table.

Table 6-1 aDSP/SLPI/SDC

File type	Description
QUP v3 serial engine configuration	■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_devcfg.c ■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_devcfg.json
Firmware configuration settings	■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c ■ /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml
Public APIs	■ api/buses/i2c_api.h ■ api/buses/qup_common.h

6.3 I3C software

The I3C firmware for the QUP v3 serial engine is loaded with SSC QUP during the bootup sequence. The configuration file is present in the aDSP build. The source code path to enable the I3C configuration for a specific QUP v3 serial engine instance is at /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.c and /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_fw/config/<chipset>/fw_devcfg.xml.

Sample of SSC QUP SE0 or SE1 loaded with the I3C firmware.

```
se_cfg se0_cfg = { 0x80000, SE_PROTOCOL_I3C,    GSI,     TRUE, TRUE  };
se_cfg se1_cfg = { 0x84000, SE_PROTOCOL_I3C,    GSI,     TRUE, TRUE  };
```

GPIO configuration: Each serial engine in the QUP common driver is configured with the default GPIO configuration. The QUP common driver selects the GPIO configuration according to the protocol loaded in the serial engine. The source code path for the GPIO configuration of a specific I3C serial engine instance is /firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/ADSP.HT.5.5.c8/adsp_proc/core/settings/buses/qup_common/config/<chipset>/adsp/ssc/qup_instance_mapping.c.

The default GPIO configuration can be overwritten as follows:

```
{ .instance_id = 1 , //Instance ID
.qup = QUP_SSC, //QUP Type
.se_index = 0, //SE ID
.se_data = NULL, //devcfg_map
.protocol_io_cfg = {
TLMM_MAP(TLMM_GPIO_KEEPER ,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER ), // SLEEP CFG
```

```
TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_6MA,TLMM_GPIO_KEEPER ),  //SPI CFG
TLMM_MAP(TLMM_GPIO_NO_PULL,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL), // UART CFG
TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_NO_PULL), //I2C CFG
TLMM_MAP(TLMM_GPIO_PULL_UP,TLMM_GPIO_2MA,TLMM_GPIO_KEEPER ) //I3C CFG
},
.se_exclusive = TRUE,
}
```

TLMM_MAP is a macro to initialize the active and sleep state GPIO configurations. For example, a sample usage of the TLMM_MAP macro is as follows:

```
TLMM_MAP (active state pull type, drive strength, sleep state pull type)
```

7 PCIe

PCIe uses a bidirectional connection to send and receive information at the same time, as shown in the following figure.

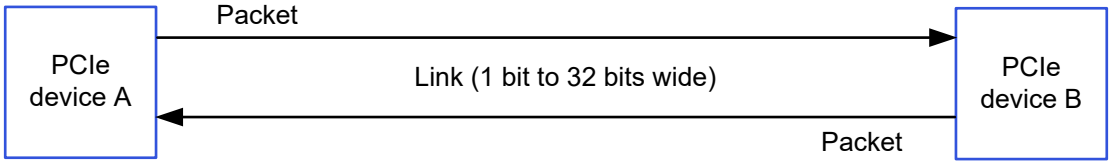


Figure 7-1 PCIe device connection link

The path between the devices is called a Link. It is made up of one or more transmit and receive pairs. One pair of the Link is called a Lane. The PCIe device connection in Qualcomm Linux devices supports 16 lanes. The number of lanes or the Link width is x4. For more information on PCIe, see https://pcisig.com/specifications/pciexpress/technical_library/pciexpress_whitepaper.pdf.

The following table lists the types of PCIe connections for devices.

Table 7-1 PCIe connections

PCIe type	Description
Root complex (RC)	Connects the CPU to the PCIe topology
Switch	Connects more than 2 ports and acts as a packet router
Bridge	Connects different buses: for example, PCIe to PCIe, or PCIe to peripheral component interconnect (PCI)
Endpoint (EP)	Resides at the bottom of the PCIe topology tree structure and has only an upstream port
Legacy endpoint	Uses older PCI bus operations to support backward compatibility

7.1 PCIe host mode enumeration feature

When a system first powers up, the configuration software running on the system host processor is aware of the existence of only Bus 0 (if PCIe is supported). The software is not aware of the bus topology or any device connected to the bus. The enumeration process discovers the various buses, devices, and functions present in the system.

When enumeration is complete, each bus in the system is numbered as follows:

- The primary bus number indicates the bus that directly connects to the primary interface of the bridge (towards the root complex).
- The secondary bus number indicates the bus that directly connects to the secondary interface of the bridge (away from the root complex).
- The subordinate bus number indicates the highest numbered bus that exists on the downstream side of the bridge.

The BDF number uniquely identifies each device. The ID-based routing method of the transaction layer packet (TLP) uses this number. The PCIe host mode enumeration process involves the following:

1. Link training
2. Scanning for devices on the bus
3. Registration

For more information on the PCIe device initialization, enumeration process, see https://www.kernel.org/doc/html/latest/PCI/index.html.

7.2 PCIe layered architecture

The following figure shows the layered architecture model of PCIe.

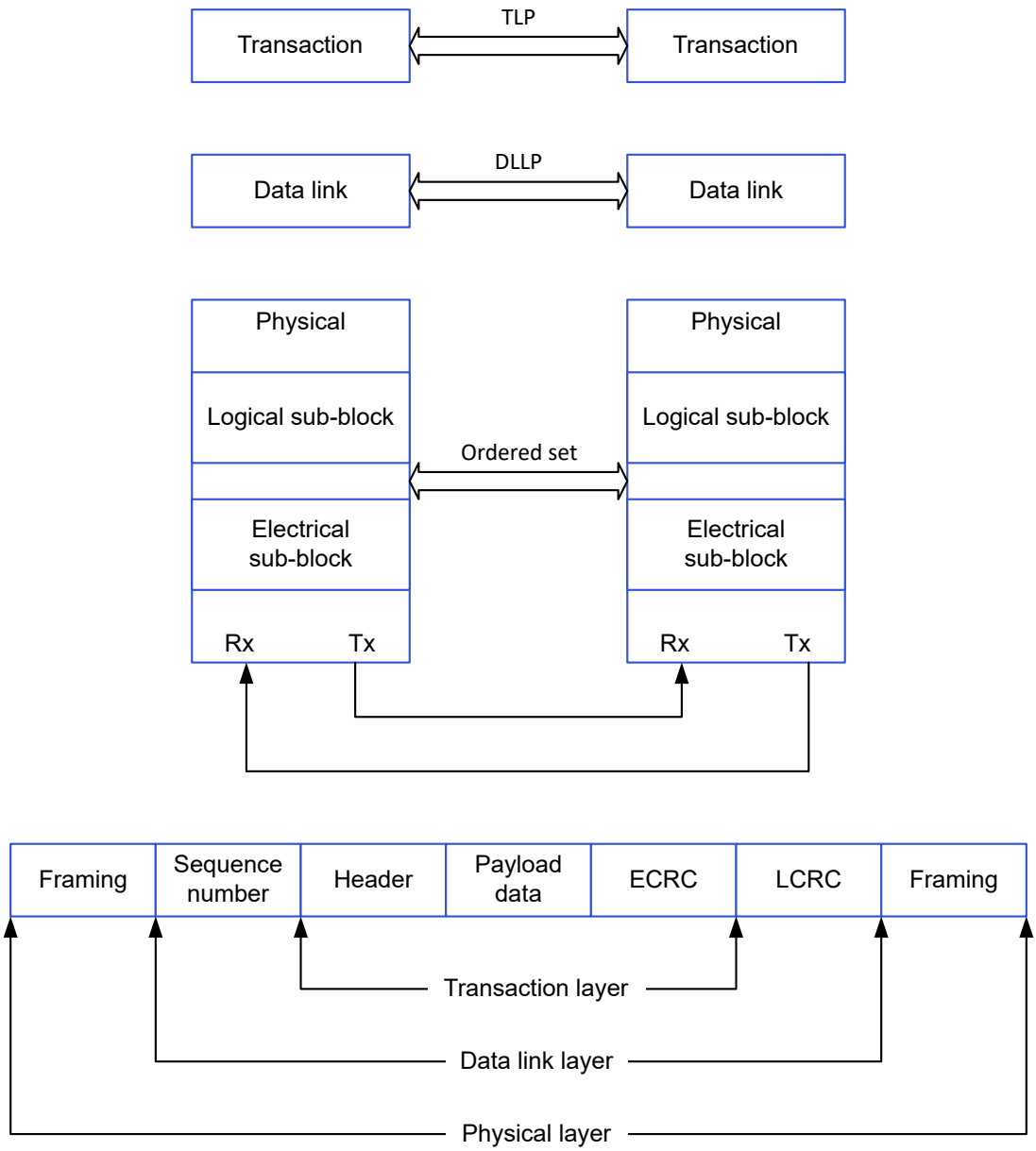


Figure 7-2 PCIe layered architecture

The transmission units exchanged are as follows.

- Ordered set between the physical layer entities.
- Data link layer packet (DLLP) between data link layer entities.
- Transaction layer packet (TLP) between transaction layer entities.

The following table lists the three layers, with the respective functions, in the PCIe architecture.

Table 7-2 Layers in PCIe architecture

Layer	Features
Physical layer	Logical sub-block: Link training, initialization, and maintenance.
	Physical sub-block: 8b/10b encoding and decoding, and parallel-to-serial and serial-to-parallel conversion.
Data link layer	Assembly and disassembly of the DLLP packet.
	Generation and validation of the link layer CRC (LCRC).
	Acknowledgment and no acknowledgment protocol (replay of TLPs in error).
Transaction layer	Assembly and disassembly of the TLP packet.
	Generation and validation of end-to-end CRC (ECRC).
	Flow control receives entity advertises for the available to receive buffer size information using DLLPs.
	Quality of service (QoS): traffic class (TC) to virtual channel (VC) mapping.
	Transaction ordering: implements the transaction ordering rule within a VC.

PCI defines a dedicated block of configuration address space for each function as shown in the following figure. The software determines the presence of a function, configures it, and checks and controls its status.

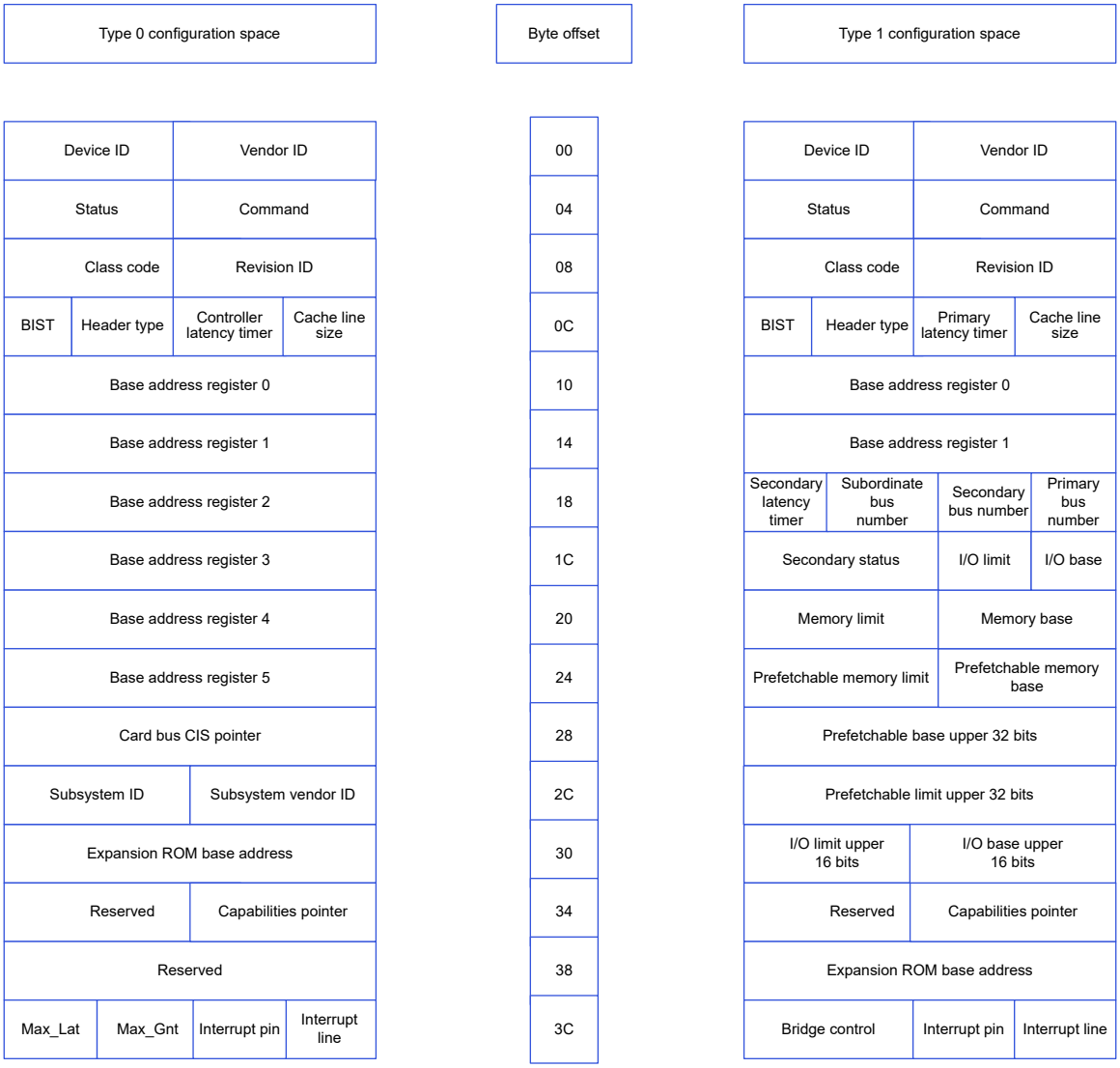


Figure 7-3 PCIe configuration address space

7.3 PCIe software

The PCIe controller driver initializes the PCIe resources and performs link training. After successful training, the controller driver calls the PCIe framework for link enumeration, such as endpoint discovery, identifying the client driver, and probing those drivers. For more information about the PCIe framework and client driver PCIe registrations, see https://www.kernel.org/doc/html/latest/PCI/index.html.

Link training

Link training comprises the following operations:

1. The PCIe driver `pcie-qcom.c` file at https://github.com/torvalds/linux/blob/master/drivers/pci/controller/dwc/pcie-qcom.c obtains the required resources such as regulators, clocks, from the device tree.
2. The PCIe driver calls Synopsys DesignWare® Core host driver `pcie-designware-host.c` file at https://github.com/torvalds/linux/blob/master/drivers/pci/controller/dwc/pcie-designware-host.c to initialize the root complex.
3. The Synopsys DesignWare Core driver performs all necessary initializations.
4. The Synopsys DesignWare Core driver calls a function pointer to perform host initialization.
5. The Qualcomm PCIe driver performs PHY power-on, enables all regulators, clocks.
6. The Synopsys DesignWare Core driver starts the link training by calling the function pointer to start the link.

Hardware initialization

The driver initializes and configures the PCIe hardware block and performs link training. The initialization occurs after the `platform _probe()` driver function is called.

```
static int qcom_pcie_probe(struct platform_device *pdev)
{

        // get PCIe resources

        ret = dw_pcie_host_init(pp);  // call DWC framework to host
intialisation like MSI, MSIx and controller init
        if (ret) {
                dev_err(dev, "cannot initialize host\n");
                goto err_phy_exit;
        }

};

static const struct of_device_id qcom_pcie_match[] = {
        { .compatible = "qcom,pcie-sc7280", .data = &cfg_1_9_0 },
        {}

};

static struct platform_driver qcom_pcie_driver = {
        .probe = qcom_pcie_probe,
        .driver = {
                .name = "qcom-pcie",
                .suppress_bind_attrs = true,
                .of_match_table = qcom_pcie_match,
```

```
        },
};
```

When the Synopsys DesignWare Core driver is initialized, it also initializes MSI, MSIx, and controllers. The driver calls a Qualcomm function pointer to start the link training.

```
static int qcom_pcie_start_link(struct dw_pcie *pci)
{
        struct qcom_pcie *pcie = to_qcom_pcie(pci);

        /* Enable Link Training state machine */
        if (pcie->cfg->ops->ltssm_enable)
                pcie->cfg->ops->ltssm_enable(pcie);

        return 0;
}

static int qcom_pcie_link_up(struct dw_pcie *pci)
{
        u16 offset = dw_pcie_find_capability(pci, PCI_CAP_ID_EXP);
        u16 val = readw(pci->dbi_base + offset + PCI_EXP_LNKSTA);

        return !!(val & PCI_EXP_LNKSTA_DLLLA); // checks the link is up or not
}
static const struct dw_pcie_ops dw_pcie_ops = {
        .link_up = qcom_pcie_link_up,
        .start_link = qcom_pcie_start_link,
};
```

The Synopsys DesignWare Core driver waits for the link to be active to enumerate the PCI framework. The Qualcomm PCIe driver enables only the link training.

NOTE The PCIE_0 root complex instance is enabled by default for the WLAN EP connection.

7.4 Enable QPS615 PCIe switch

This section describes how to enable a QPS615 PCIe switch in the Qualcomm Linux hardware SoCs. The QPS615 switch endpoint is supported on the `PCIe1` instance. The following figure shows the QPS615 endpoint and connections.

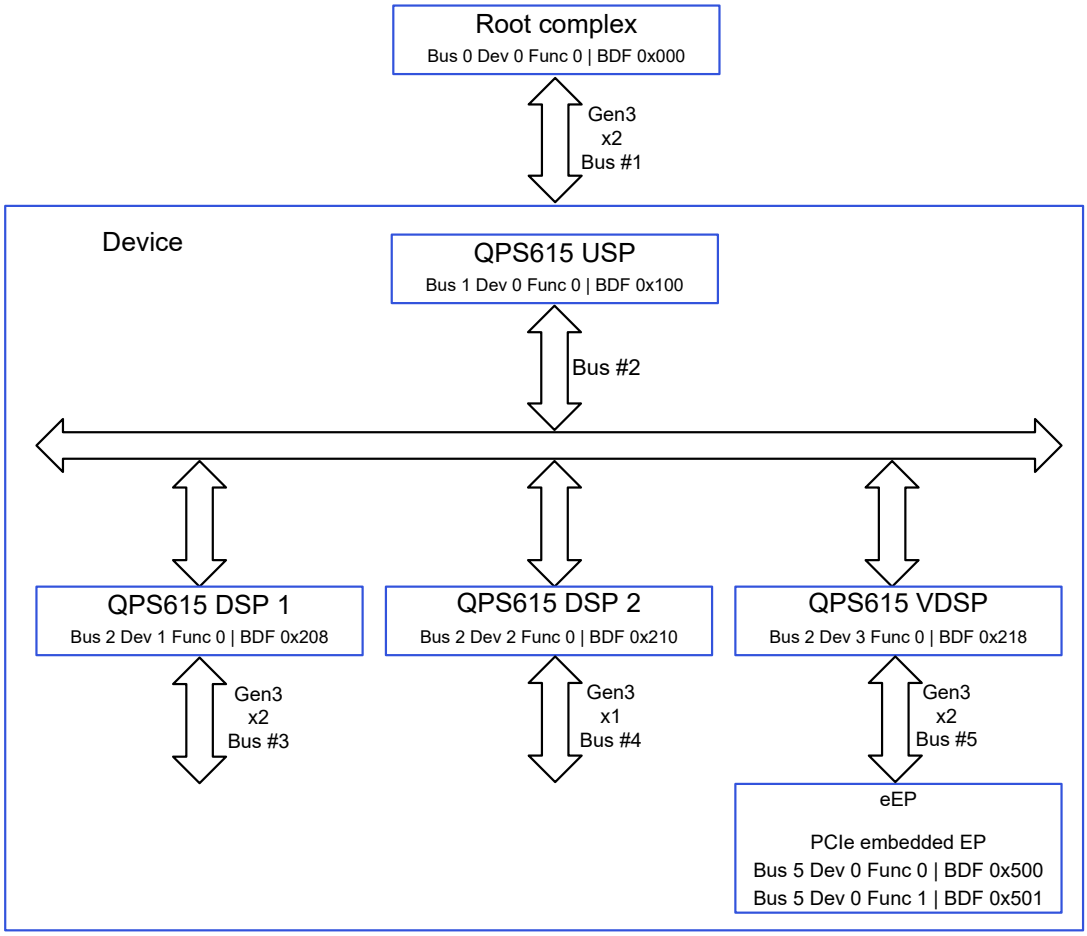


Figure 7-4 QPS615 PCIe switch connection diagram

The Qualcomm PCIe driver documentation can be accessed at the following locations:

- https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/pci/qcom%2Cpcie-sc8280xp.yaml
- https://elixir.bootlin.com/linux/v6.6.48/source/Documentation/devicetree/bindings/pci/qcom,pcie.yaml
- https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/phy/qcom%2Csc8280xp-qmp-pcie-phy.yaml

PCIe-related configurations

The following configurations are enabled by default to support the QPS615 switch. Disable the QPS615 switch default support, by reverting the code changes, to use it for a different PCIe endpoint.

To enable PCIe-related `configs`, apply the following patch to the `/arch/arm64/configs/qcom_addons.config` file.

```
QCLINUX: arm64: defconfig: qcom:
diff --git a/arch/arm64/configs/qcom_addons.config b/arch/arm64/configs/
qcom_addons.config
index 46555f9..a83f417 100644
--- a/arch/arm64/configs/qcom_addons.config
+++ b/arch/arm64/configs/qcom_addons.config
@@ -25,3 +25,4 @@
 CONFIG_VIRT_DRIVERS=y
 CONFIG_QCOM_SMP2P_SLEEPSTATE=m
 CONFIG_QCOM_SOC_DEBUG=y
+CONFIG_QCOM_QPS615_PCIE_SWITCH=y
diff --git a/arch/arm64/configs/qcom_defconfig b/arch/arm64/configs/
qcom_defconfig
index 7293680..730122c 100644
--- a/arch/arm64/configs/qcom_defconfig
+++ b/arch/arm64/configs/qcom_defconfig
@@ -173,6 +173,7 @@
 CONFIG_HOTPLUG_PCI=y
 CONFIG_HOTPLUG_PCI_ACPI=y
 CONFIG_PCI_HOST_GENERIC=y
+CONFIG_PCIE_DW_PLAT_HOST=y
 CONFIG_PCIE_QCOM=y
 CONFIG_PCI_ENDPOINT=y
 CONFIG_PCI_ENDPOINT_CONFIGFS=y
@@ -793,3 +794,5 @@
 CONFIG_CORESIGHT_TPDM=m
 CONFIG_CORESIGHT_DUMMY=m
 CONFIG_MEMTEST=y
+CONFIG_PCIEASPM=y
+CONFIG_PCIEASPM_POWER_SUPERSAVE=y
```

Always-on refclk signal to endpoint

In PCIe low-power states such as L1.1 or L1.2, the PHY stops supplying `refclk` to the endpoint. However, the `refclk` signal must be supplied to the endpoint. If the endpoint asserts `clkreq` to bring back the link to L0, then root complex must provide `refclk` to the endpoint. Some devices with PCIe QPS615 switches fail to drive the `clkreq` signal to the host from the endpoints due to the switch board design. You can add a flag to ensure `refclk` is always supplied to the endpoint.

To retain `refclk` in always-on status, apply the following patch to the `phy-qcom-qmp-pcie.c` file at https://github.com/torvalds/linux/blob/master/drivers/phy/qualcomm/phy-qcom-qmp-pcie.c.

```
phy: qcom-qmp-pcie:
--- a/drivers/phy/qualcomm/phy-qcom-qmp-pcie.c
+++ b/drivers/phy/qualcomm/phy-qcom-qmp-pcie.c
@@ -43,6 +43,8 @@
 /* QPHY_PCS_STATUS bit */
 #define PHYSTATUS                BIT(6)
 #define PHYSTATUS_4_20                BIT(7)
+/* PCS_PCIE_ENDPOINT_REFCLK_CNTRL */
+#define EPCLK_ALWAYS_ON_EN            BIT(6)

 #define PHY_INIT_COMPLETE_TIMEOUT        10000

@@ -2293,6 +2295,8 @@
     struct phy *phy;
     int mode;

+    bool refclk_always_on;
+
     struct clk_fixed_rate pipe_clk_fixed;
 };

@@ -3238,6 +3242,10 @@
     qmp_pcie_configure(pcs, tbls->pcs, tbls->pcs_num);
     qmp_pcie_configure(pcs_misc, tbls->pcs_misc, tbls->pcs_misc_num);

+    if (qmp->refclk_always_on && cfg->regs[QPHY_PCS_ENDPOINT_REFCLK_CNTRL])
+        qphy_setbits(pcs_misc, cfg->regs[QPHY_PCS_ENDPOINT_REFCLK_CNTRL],
+                EPCLK_ALWAYS_ON_EN);
+
     if (cfg->lanes >= 4 && qmp->tcsr_4ln_config) {
         qmp_pcie_configure(serdes, cfg->serdes_4ln_tbl, cfg->serdes_4ln_num);
         qmp_pcie_init_port_b(qmp, tbls);
@@ -3760,6 +3768,12 @@
     if (ret)
         goto err_node_put;

+    qmp->refclk_always_on = of_property_read_bool(dev->of_node, "qcom,refclk-
always-on");
+    if (qmp->refclk_always_on && !qmp->cfg-
>regs[QPHY_PCS_ENDPOINT_REFCLK_CNTRL]) {
+        dev_err(dev, "refclk is always on is present but refclk cntrl offset
is not present\n");
+        goto err_node_put;
+    }
+
     ret = phy_pipe_clk_register(qmp, np);
```

```
    if (ret)
        goto err_node_put;
```

Message signaled interrupt (MSI)

The current MSI mapping doesn't have all the vectors. The Qualcomm Linux hardware SoCs support eight vectors. Each vector in turn supports 32 MSIs. Therefore, the total MSIs supported are 256.

For information about adding all the MSI groups supported for this PCIe instance, see https://lore.kernel.org/linux-arm-msm/f1168212-bc6e-4570-869c-2870d6f248ad@linaro.org/T/.

QPS615 switch support

When all the GPIOs that control power to the QPS615 PCIe switch are added as fixed regulators, the QPS615 driver enables the power through the regulator framework. It also performs I2C writes to configure the QPS615 switch. The PCIe node is added as a dependency to the QPS615 node so that the PCIe driver probes after the QPS615 driver probe, which ensures that the switch is powered and ready for enumeration. When QPS615 can't toggle, the CLKREQ pin causes a device crash. This is a known limitation. Hence, the `refclk` is set to always-on.

QPS615 switch device tree bindings

To add QPS615 switch device tree binding, apply the following patch to the `/arch/arm64/boot/dts/qcom/<chipset>-addons-rb3.dts` file.

```
QCLINUX: dt-bindings: pci: qps615:
diff --git a/arch/arm64/boot/dts/qcom/<chipset>-addons-rb3.dts b/arch/arm64/
boot/dts/qcom/<chipset>-addons-rb3.dts
index 109ad08..7baf505 100644
--- a/arch/arm64/boot/dts/qcom/<chipset>-addons-rb3.dts
+++ b/arch/arm64/boot/dts/qcom/<chipset>-addons-rb3.dts
@@ -14,6 +14,82 @@
 / {
     model = "Qualcomm Technologies, Inc. <chipset>-addons RB3 platform";
     compatible = "qcom,<chipset>-addons-rb3","qcom,sc7280";
+
+    qps615_0p9_vreg: qps615-0p9-vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "qps615_0p9_vreg";
+        gpio = <&pm8350c_gpios 2 0>;
+        regulator-min-microvolt = <1000000>;
+        regulator-max-microvolt = <1000000>;
+        enable-active-high;
+        regulator-enable-ramp-delay = <4300>;
+    };
+
+    qps615_1p8_vreg: qps615-1p8-vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "qps615_1p8_vreg";
+        gpio = <&pm8350c_gpios 3 0>;
```

```
+        vin-supply = <&qps615_0p9_vreg>;
+        regulator-min-microvolt = <1800000>;
+        regulator-max-microvolt = <1800000>;
+        enable-active-high;
+        regulator-enable-ramp-delay = <10000>;
+    };
+
+    qps615_rsex_vreg: qps615-rsex-vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "qps615_rsex_vreg";
+        gpio = <&pm8350c_gpios 1 0>;
+        vin-supply = <&qps615_1p8_vreg>;
+        regulator-min-microvolt = <1800000>;
+        regulator-max-microvolt = <1800000>;
+        enable-active-high;
+        regulator-enable-ramp-delay = <10000>;
+    };
+
+    usb_hub_1p05_vreg: usb-hub-1p05-vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "usb_hub_1p05_vreg";
+        gpio = <&pm7250b_gpios 4 0>;
+        vin-supply = <&qps615_rsex_vreg>;
+        regulator-min-microvolt = <1000000>;
+        regulator-max-microvolt = <1000000>;
+        enable-active-high;
+        regulator-enable-ramp-delay = <5000>;
+    };
+
+    usb_hub_3p3_vreg: usb-hub-3p3-vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "usb_hub_3p3_vreg";
+        gpio = <&pm7250b_gpios 1 0>;
+        vin-supply = <&usb_hub_1p05_vreg>;
+        regulator-min-microvolt = <3300000>;
+        regulator-max-microvolt = <3300000>;
+        enable-active-high;
+        regulator-enable-ramp-delay = <10000>;
+    };
+
+    usb_hub_rest_vreg: usb-hub-rest-vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "usb_hub_rest_vreg";
+        gpio = <&pm8350c_gpios 4 0>;
+        vin-supply = <&usb_hub_3p3_vreg>;
+        regulator-min-microvolt = <3300000>;
+        regulator-max-microvolt = <3300000>;
```

```
+        enable-active-high;
+    };
+};
+
+&i2c0 {
+    clock-frequency = <100000>;
+    status = "okay";
+
+    qps615_switch: pcie-switch@77 {
+        compatible = "qcom,switch-i2c";
+        reg = <0x77>;
+        vdda-supply = <&usb_hub_rest_vreg>;
+        status = "okay";
+    };
 };

 &i2c1 {
@@ -216,6 +292,14 @@
     };
 };

+&pcie1 {
+    dummy-supply = <&qps615_switch>;
+};
+
+&pcie1_phy {
+    qcom,refclk-always-on;
+};
+

&pm8350c_gpios {
pm8008i-reset-state {
pm8008i_active: pm8008i-active-pins {

Documentation/devicetree/bindings/pci/qps615-switch.yaml Documentation/
devicetree/bindings/pci/qps615-switch.yaml
new file mode 100644
index 0000000..f59e068
--- /dev/null
+++ b/Documentation/devicetree/bindings/pci/qps615-switch.yaml
@@ -0,0 +1,41 @@
+# SPDX-License-Identifier: (GPL-2.0-only OR BSD-2-Clause)
+%YAML 1.2
+---
+$id: http://devicetree.org/schemas/pci/qps615-switch.yaml#
+$schema: http://devicetree.org/meta-schemas/core.yaml#
+
```

```
+title: Qualcomm Technologies, Inc. (QTI)  PCIe switch
+
+maintainers:
+  - Chundru Krishna chaitanya <quic_krichai@quicinc.com>
+
+properties:
+  compatible:
+    enum:
+      - qcom,switch-i2c
+
+  reg:
+    maxItems: 1
+
+  vdda-supply:
+    description: A phandle to the core analog power supply
+
+required:
+  - compatible
+  - reg
+  - vdda-supply
+
+additionalProperties: false
+
+examples:
+  - |
+      i2c {
+          #address-cells = <1>;
+          #size-cells = <0>;
+          qps615: pcie-switch@77 {
+              compatible = "qcom,switch-i2c";
+              reg = <0x077>;
+              vdda-supply = <&foo>;
+          };
```

QPS615 switch driver

A driver is added for powering on the QPS615 PCIe switch. It performs the basic initialization through I2C. The initialization sequence is present in the firmware image that the driver requested through the `request_firmware` API.

To add the QPS615 switch driver, apply the following patch to the `/drivers/pci/controller/Kconfig` file.

```
QCLINUX: pci: controller: misc:
diff --git a/drivers/pci/controller/Kconfig b/drivers/pci/controller/Kconfig
index c0c3f28..d68e701 100644
--- a/drivers/pci/controller/Kconfig
+++ b/drivers/pci/controller/Kconfig
@@ -345,4 +345,5 @@
```

```
 source "drivers/pci/controller/cadence/Kconfig"
 source "drivers/pci/controller/dwc/Kconfig"
 source "drivers/pci/controller/mobiveil/Kconfig"
+source "drivers/pci/controller/misc/Kconfig"
 endmenu
diff --git a/drivers/pci/controller/Makefile b/drivers/pci/controller/Makefile
index 37c8663..015f711b 100644
--- a/drivers/pci/controller/Makefile
+++ b/drivers/pci/controller/Makefile
@@ -43,6 +43,7 @@
 # pcie-hisi.o quirks are needed even without CONFIG_PCIE_DW
 obj-y                   += dwc/
 obj-y                   += mobiveil/
+obj-y                   += misc/

# The following drivers are for devices that use the generic ACPI
diff --git a/drivers/pci/controller/misc/Kconfig b/drivers/pci/controller/
misc/Kconfig
new file mode 100644
index 0000000..0acea3f
--- /dev/null
+++ b/drivers/pci/controller/misc/Kconfig
@@ -0,0 +1,10 @@

+# SPDX-License-Identifier: GPL-2.0
+config QCOM_QPS615_PCIE_SWITCH
+    bool "QCOM QPS615 PCIe Switch Driver"
+    depends on PCuart
+    help
+     This adds support to enable QPS615 PCIe switch power. And after
powering on the switch do
+     switch initialization through I2C writes. The I2C data is parsed from
the requested
+     firmware.
+
+     Say Y to compile this driver.
diff --git a/drivers/pci/controller/misc/Makefile b/drivers/pci/controller/
misc/Makefile
new file mode 100644
index 0000000..1c2b99e
--- /dev/null
+++ b/drivers/pci/controller/misc/Makefile
@@ -0,0 +1,2 @@
+# SPDX-License-Identifier: GPL-2.0
+obj-$(CONFIG_QCOM_QPS615_PCIE_SWITCH) += qps615.o

Change in arm64: dts: qcom: <chipset>-rb3g2: Add PCIe nodes.
```

```
Enable PCIe1 controller and its corresponding PHY nodes on
<chipset>-rb3g2 platform. As there are multiple endpoints connected through
PCIe switch add smmu id for each BDF.

diff --git a/arch/arm64/boot/dts/qcom/<chipset>-rb3.dts b/arch/arm64/boot/dts/
qcom/<chipset>-rb3.dts
index 5adce1f..f995a53 100644
--- a/arch/arm64/boot/dts/qcom/<chipset>-rb3.dts
+++ b/arch/arm64/boot/dts/qcom/<chipset>-rb3.dts
@@ -512,6 +512,32 @@
     bias-bus-hold;
 };

+&pcie1 {
+    perst-gpios = <&tlmm 2 GPIO_ACTIVE_LOW>;
+
+    pinctrl-0 = <&pcie1_reset_n>, <&pcie1_wake_n>;
+    pinctrl-names = "default";
+
+    iommu-map = <0x0 &apps_smmu 0x1c80 0x1>,
+            <0x100 &apps_smmu 0x1c81 0x1>,
+            <0x208 &apps_smmu 0x1c84 0x1>,
+            <0x210 &apps_smmu 0x1c85 0x1>,
+            <0x218 &apps_smmu 0x1c86 0x1>,
+            <0x300 &apps_smmu 0x1c87 0x1>,
+            <0x400 &apps_smmu 0x1c88 0x1>,
+            <0x500 &apps_smmu 0x1c89 0x1>,
+            <0x501 &apps_smmu 0x1c90 0x1>;
+
+    status = "okay";
+};
+
+&pcie1_phy {
+    vdda-phy-supply = <&vreg_l10c_0p88>;
+    vdda-pll-supply = <&vreg_l6b_1p2>;
+
+    status = "okay";
+};
+
 &qup_uart5_rx {
     drive-strength = <2>;
     bias-pull-up;
@@ -604,6 +630,21 @@
         bias-disable;
     };

+    pcie1_reset_n: pcie1-reset-n-state {
```

```
+        pins = "gpio2";
+        function = "gpio";
+        drive-strength = <16>;
+        output-low;
+        bias-disable;
+    };
+
+    pcie1_wake_n: pcie1-wake-n-state {
+        pins = "gpio3";
+        function = "gpio";
+        drive-strength = <2>;
+        bias-pull-up;
+    };
+
     qup_uart7_sleep_cts: qup-uart7-sleep-cts-state {
         pins = "gpio28";
         function = "gpio";
```

Save and restore configuration space QPS615 switch

Preserve the configuration space of the PCIe bridge device during power management suspend state. Resume the PCIe bridge configuration space during power management resume state. This method is essential for the proper operation of PCIe endpoints connected using the QPS615 PCIe switch.

```
diff --git a/drivers/pci/controller/misc/qps615.c b/drivers/pci/controller/
misc/qps615.c
index 7d63efb..10048cd 100644
--- a/drivers/pci/controller/misc/qps615.c
+++ b/drivers/pci/controller/misc/qps615.c
@@ -5,6 +5,7 @@
 #include <linux/firmware.h>
 #include <linux/i2c.h>
 #include <linux/module.h>
+#include <linux/pci.h>

 #define DRV_NAME        "qps615-switch-i2c"

@@ -235,6 +236,24 @@
 }
 module_init(qps615_i2c_init);

+static void qcom_pcie_resume_early(struct pci_dev *pdev)
+{
+    pci_restore_state(pdev);
+}
+
+DECLARE_PCI_FIXUP_CLASS_RESUME_EARLY(PCI_ANY_ID,
+                PCI_ANY_ID, PCI_CLASS_BRIDGE_PCI_NORMAL, 0,
```

```
+                qcom_pcie_resume_early);
+
+static void qcom_pcie_suspend_late(struct pci_dev *pdev)
+{
+    pci_save_state(pdev);
+}
+
+DECLARE_PCI_FIXUP_CLASS_SUSPEND_LATE(PCI_ANY_ID,
+                PCI_ANY_ID, PCI_CLASS_BRIDGE_PCI_NORMAL, 0,
+                qcom_pcie_suspend_late);
+
 MODULE_AUTHOR("Krishna Chaitanya Chundru <quic_krichai@quicinc.com>");
 MODULE_DESCRIPTION("QPS615 PCIE Switch driver");
 MODULE_LICENSE("GPL");
```

The following is a sample PCIe kernel driver log from the QPS615 device enumeration.

```
[    7.254674] qcom-pcie 1c08000.pci: supply vdda not found, using dummy
regulator
[    7.285224] qcom-pcie 1c08000.pci: supply vddpe-3v3 not found, using dummy
regulator
[    7.299688] qcom-pcie 1c08000.pci: host bridge /soc@0/pci@1c08000 ranges:
[    7.313713] qcom-pcie 1c08000.pci: IO 0x0040200000..0x00402fffff ->
0x0000000000
[    7.329778] qcom-pcie 1c08000.pci: MEM 0x0040300000..0x005fffffff ->
0x0040300000
[    7.479174] qcom-pcie 1c08000.pci: iATU: unroll T, 8 ob, 8 ib, align 4K,
limit 1024G
[    7.577792] qcom-pcie 1c08000.pci: PCIe Gen.3 x2 link up
[    7.648353] qcom-pcie 1c08000.pci: PCI host bridge to bus 0001:00
[    7.654618] pci_bus 0001:00: root bus resource [bus 00-ff]
[    7.654620] pci_bus 0001:00: root bus resource [io  0x0000-0xfffff]
[    7.654622] pci_bus 0001:00: root bus resource [mem 0x40300000-0x5fffffff]
[    7.654633] pci 0001:00:00.0: [17cb:010b] type 01 class 0x060400
[    7.665893] pci 0001:00:00.0: reg 0x10: [mem 0x00000000-0x00000fff]
[    7.684913] pci 0001:00:00.0: PME# supported from D0 D3hot D3cold
[    7.700502] pci 0001:01:00.0: [1179:0623] type 01 class 0x060400
[    7.710817] pci 0001:01:00.0: PME# supported from D0 D3hot D3cold
[    7.733698] pci 0001:01:00.0: bridge configuration invalid ([bus 00-00]),
reconfiguring
[    7.738818] pci 0001:02:01.0: [1179:0623] type 01 class 0x060400
[    7.751157] pci 0001:02:01.0: PME# supported from D0 D3hot D3cold
[    7.763174] pci 0001:02:02.0: [1179:0623] type 01 class 0x060400
[    7.777084] pci 0001:02:02.0: PME# supported from D0 D3hot D3cold
[    7.792832] pci 0001:02:03.0: [1179:0623] type 01 class 0x060400
[    7.815659] pci 0001:02:03.0: PME# supported from D0 D3hot D3cold
```

7.5 Enable USB interface through PCIe switch

This section provides instructions on how to activate a USB interface through a PCIe switch in the Qualcomm Linux hardware SoCs. The PCIE1 instance is connected to the endpoint of the QPS615 switch, and the downstream port of the QPS615 is connected to the PCIe to USB endpoint. For the PCIe to USB endpoint connections using the QPS615, see the mainboard and interposer block diagram at https://docs.qualcomm.com/bundle/publicresource/topics/80-70015-251/rb3_hardware_overview.html.

NOTE QCS9075 PCIe software doesn't support USB.

Enable power for the PCIe to USB controller

Enable the power for the PCIe to USB controller connected through the QPS615 switch, and reset the external USB hub connected using GPIO162. This method is necessary for the correct detection and functioning of the USB peripherals.

```
diff --git a/arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi b/arch/
arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi
index bf95b66..4a76a36 100644
--- a/arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi
+++ b/arch/arm64/boot/dts/qcom/qcs6490-addons-rb3gen2.dtsi
@@ -40,6 +40,37 @@
         regulator-enable-ramp-delay = <10000>;
     };

+    upd_3p3_vreg: upd_3p3_vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "upd_3p3_vreg";
+        gpio = <&pm7250b_gpios 1 0>;
+        vin-supply = <&qps615_rsex_vreg>;
+        regulator-min-microvolt = <3300000>;
+        regulator-max-microvolt = <3300000>;
+        enable-active-high;
+        regulator-enable-ramp-delay = <10000>;
+        regulator-always-on;
+    };
+
+    upd_rest_vreg: upd_rest_vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "upd_rest_vreg";
+        gpio = <&pm8350c_gpios 4 0>;
+        vin-supply = <&upd_3p3_vreg>;
+        regulator-min-microvolt = <3300000>;
+        regulator-max-microvolt = <3300000>;
+        enable-active-high;
+        regulator-always-on;
+    };
+
```

```
+    usbhub_rest_vreg: usbhub_rest_vreg {
+        compatible = "regulator-fixed";
+        regulator-name = "usbhub_rest_vreg";
+        pinctrl-names = "default";
+        pinctrl-0 = <&pcie1_usb_hub_reset_default>;
+        gpio = <&tlmm 162 GPIO_ACTIVE_LOW>;
+        enable-active-high;
+    };
 };

 &i2c0 {
@@ -165,6 +196,14 @@
         bias-pull-down;
         input-enable;
     };
+
+    pcie1_usb_hub_reset_default: pcie1_usb_hub_reset_default {
+        pins = "gpio162";
+        function = "gpio";
+        drive-strength = <2>;
+        output-high;
+        bias-pull-down;
+    };
 };
```

Avoid early handoff in PCIe to USB controller

Ensure that `usb_early-handoff` is skipped for the PCIe to USB controller since the firmware isn't loaded. This precaution is necessary to prevent any adverse impact on the bootup duration.

```
diff --git a/drivers/usb/host/pci-quirks.c b/drivers/usb/host/pci-quirks.c
index 2665832..c3d488f 100644
--- a/drivers/usb/host/pci-quirks.c
+++ b/drivers/usb/host/pci-quirks.c
@@ -1260,6 +1260,11 @@
             return;
     }

+    /* Skip handoff for Renesas PCI USB controller on QCOM SOC */
+    if ((pdev->vendor == PCI_VENDOR_ID_RENESAS) &&
+        (pcie_find_root_port(pdev)->vendor == PCI_VENDOR_ID_QCOM))
+        return;
+
     if (pdev->class != PCI_CLASS_SERIAL_USB_UHCI &&
             pdev->class != PCI_CLASS_SERIAL_USB_OHCI &&
             pdev->class != PCI_CLASS_SERIAL_USB_EHCI &&
```

Download PCIe to USB controller firmware

To download the firmware from https://www.renesas.com/us/en/products/interface/usb-switches-hubs/upd720201-usb-30-host-controller#design_development, register and log in to https://www.renesas.com/.

NOTE To prevent command failures, update the software as described in the Update software section before updating the Renesas firmware.

1. Create the `usb_fw.img` image and copy the USB firmware by running the following commands on the Linux host machine.

```
dd if=/dev/zero of=usb_fw.img bs=4k count=240

mkfs -t ext4 usb_fw.img

mkdir usb_fw

sudo mount -o loop usb_fw.img usb_fw/

sudo cp -rf renesas_usb_fw.mem usb_fw

sudo umount usb_fw
```

2. Start the device in the fastboot mode.

```
adb root

adb shell

reboot bootloader
```

3. Run the following command when the device is in the fastboot mode.

```
fastboot devices
```

Sample output:

```
7dc85f5e         fastboot
```

4. Flash the `usb_fw.img` image to the device.

```
fastboot erase usb_fw

fastboot flash usb_fw  usb_fw.img

fastboot reboot
```

Command failure sample:

```
c:\>fastboot erase usb_fw
Erasing 'usb_fw'                                   FAILED (remote:
'Check device console.')
fastboot: error: Command failed
```

5. To verify if the firmware is successfully updated, run the following command.

```
dmesg
```

Sample log after the firmware is successfully updated.

```
[    6.589462] usbcore: registered new device driver onboard-usb-hub
[    6.653277] usb 2-1: new SuperSpeed USB device number 2 using
xhci_hcd
[    7.013061] usb 2-1.1: new SuperSpeed USB device number 3 using
xhci_hcd
```

```
[    7.120657] ax88179_178a 2-1.1:1.0 eth0: register 'ax88179_178a' at
usb-0001:04:00.0-1.1, ASIX AX88179 USB 3.0 Gigabit Ethernet,
3e:9e:5e:ff:d3:fb
[    7.120767] usbcore: registered new interface driver ax88179_178a
```

PCIe kernel driver logs for PCIe to USB device enumeration reference

You can run the following commands to view the device information:

- To display device information in USB, run the following command.

  ```
  lsusb
  ```

  The following message is displayed.

  ```
  Bus 002 Device 003: ID 0b95:1790 ASIX Electronics Corp. AX88179 Gigabit
  Ethernet
  Bus 002 Device 002: ID 05e3:0625 Genesys Logic, Inc. USB3.2 Hub
  Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
  Bus 001 Device 002: ID 05e3:0610 Genesys Logic, Inc. Hub
  Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
  ```

- To display device information in PCIe, run the following command.

  ```
  lspci
  ```

  The following message is displayed.

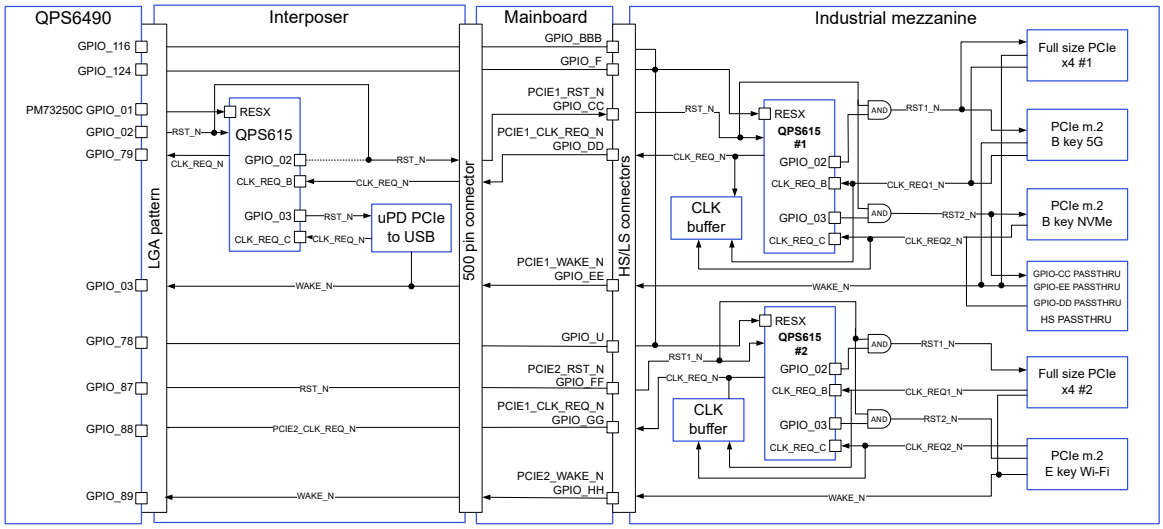
  ```
  0001:00:00.0 PCI bridge: Qualcomm Device 010b
  0001:01:00.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:01.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:02.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:03.0 PCI bridge: Toshiba Corporation Device 0623
  0001:04:00.0 USB controller: Renesas Technology Corp. uPD720201 USB 3.0
  Host Controller (rev 03)
  0001:05:00.0 Ethernet controller: Toshiba Corporation Device 0220
  0001:05:00.1 Ethernet controller: Toshiba Corporation Device 0220
  ```

7.6 Connect QPS615 switches in cascade

Connect the QPS615 switches in cascade to enable additional Ethernet, PCIe, and USB ports.

NOTE This feature is supported only in QCS6490.

The following image shows the two QPS615 switches connected in cascade to PCIE1 and another QPS615 switch connected to PCIE2.



To initiate the link training and enumeration for all endpoints, do the following:

1. To reset the QPS615 switch, toggle the RESX GPIOs for both QPS615 #1 and QPS615 #2.
2. To control the endpoint reset, trigger PERST. Both the switches share the PERST.

The devices can be attached directly to the QPS615 switch. The PCIe RC0 is attached to the WLAN endpoint. The following differences are due to the PCIe node hierarchy.

- Switch-attached devices:
  - The QPS615 PCIe tree node hierarchy is statically fixed.
  - All nodes for switching USP and DSP ports are created during PCIe initialization.
  - One of the switch DSP ports represents WLAN.
  - If you disable the WLAN node, it disables the WLAN device, but the PCIe downstream port remains enabled and returns a default maximum link width.
- Direct attach WLAN devices:
  - Disables only a single node.
  - Returns the `Invalid` argument, when WLAN is disabled.

7.7 Enable NVMe over PCIe

This section describes how to enable NVMe over PCIe for storage expansion. To verify if NVMe is connected, do the following:

1. To display PCIe device information, run the following command.

   ```
   lspci
   ```

   Output:

```
0000:00:00.0 PCI bridge: Qualcomm Device 0115
    0000:01:00.0 Network controller: Qualcomm QCNFA765 Wireless Network
Adapter (rev 01)
    0001:00:00.0 PCI bridge: Qualcomm Device 0115
    0001:01:00.0 Non-Volatile memory controller: Samsung Electronics Co
Ltd NVMe SSD Controller PM9A1/PM9A3/980PRO
```

2. Locate the PCIe logs.

```
dmesg | grep pcie
```

Output:

```
[    0.000000] Kernel command line:  root=/dev/disk/by-partlabel/system
rw rootwait console=ttyMSM0,115200n8
pcie_pme=nomsi net.ifnames=0 pci=noaer kpti=off kasan=off
kasan.stacktrace=off swiotlb=128 earlycon reboot=panic_warm
page_owner=on qcom_scm.download_mode=1 slub_debug=FZP,
zs_handle,zspage;FZPU mitigations=auto kernel.sched_pelt_multiplier=4
rcupdate.rcu_expedited=1 rcu_nocbs=0-7 no-steal-acc
vfio_iommu_type1.allow_unsafe_interrupts=1 fw_devlink.strict=1
    [    3.842071] qcom-pcie 1c00000.pci: supply vdda not found, using
dummy regulator
    [    3.845339] qcom-pcie 1c10000.pci: supply vdda not found, using
dummy regulator
    [    3.845428] qcom-pcie 1c10000.pci: supply vddpe-3v3 not found,
using dummy regulator
    [    3.845508] qcom-pcie 1c10000.pci: host bridge /pci@1c10000
ranges:
    [    3.845519] qcom-pcie 1c10000.pci:       IO
0x0060200000..0x00602fffff -> 0x0000000000
    [    3.845526] qcom-pcie 1c10000.pci:      MEM
0x0060300000..0x007fffffff -> 0x0060300000
    [    3.848154] qcom-pcie 1c00000.pci: supply vddpe-3v3 not found,
using dummy regulator
    [    3.854556] qcom-pcie 1c00000.pci: host bridge /pci@1c00000
ranges:
    [    3.860574] qcom-pcie 1c00000.pci:       IO
0x0040200000..0x00402fffff -> 0x0000000000
    [    3.867362] qcom-pcie 1c00000.pci:      MEM
0x0040300000..0x005fffffff -> 0x0040300000
    [    3.963275] qcom-pcie 1c10000.pci: iATU: unroll T, 32 ob, 8 ib,
align 4K, limit 1024G
    [    3.990385] qcom-pcie 1c00000.pci: iATU: unroll T, 32 ob, 8 ib,
align 4K, limit 1024G
    [    4.072091] qcom-pcie 1c10000.pci: PCIe Gen.4 x2 link up
    [    4.080241] qcom-pcie 1c10000.pci: PCI host bridge to bus 0001:00
    [    4.291824] pcieport 0001:00:00.0: Adding to iommu group 9
    [    4.551677] pcieport 0001:00:00.0: PME: Signaling with IRQ 256
    [    4.572484] WARNING: CPU: 3 PID: 89 at drivers/pci/
controller/dwc/pcie-qcom.c:1430 qcom_pcie_icc_update+0xf8/0x144
    [    4.595291] pc : qcom_pcie_icc_update+0xf8/0x144
    [    4.595293] lr : qcom_pcie_icc_update+0x9c/0x144
    [    4.640742]  qcom_pcie_icc_update+0xf8/0x144
    [    4.640745]  qcom_pcie_probe+0x234/0x30c
    [    4.988519] qcom-pcie 1c00000.pci: Phy link never came up
    [    5.302179] qcom-pcie 1c00000.pci: PCI host bridge to bus 0000:00
    [    5.372008] pcieport 0000:00:00.0: Adding to iommu group 10
    [    5.378905] pcieport 0000:00:00.0: PME: Signaling with IRQ 257
    [   11.564590] pcieport 0000:00:00.0: BAR 14: assigned [mem
0x40400000-0x405fffff]
```

3. To locate the NVMe directories, run the following command.

```
ls -lh /dev/nvme*
```

Output:

```
crw-------. 1 root root 508,  0 Jan  1 00:00 /dev/nvme0
    brw-rw----. 1 root disk 259, 30 Jan  1 00:00 /dev/nvme0n1
```

7.8 PCIe client driver sample

The client driver defines the `device-id` table and `pci_driver` structures, and registers with the PCIe framework. The following are a few PCIe client driver samples for reference.

- Sample data structure to hold client-specific private data.

```
struct sample_driver_data {
    int driver_data;
};
```

- Sample driver: You can provide data according to your driver-specific data structure.

```
struct sample_driver_data info = {};
```

- Sample device ID table with the driver-specific data. The client driver registers with the `0x306` device ID.

```
static const struct pci_device_id sample_pci_id_table[] = {
    { PCI_DEVICE_SUB(PCI_VENDOR_ID_QCOM, 0x0306, PCI_VENDOR_ID_QCOM,
0x010c),
        .driver_data = (kernel_ulong_t) &info },
    {  }
};
MODULE_DEVICE_TABLE(pci, sample_pci_id_table);
```

NOTE `MODULE_DEVICE_TABLE(pci, sample_pci_id_table);` is mandatory.

- Sample `pci_driver` data structure with client driver name, `pci-id` table, and callbacks. The pointer to this structure is passed while registering with the PCI frame work.

```
static struct pci_driver sample_pci_driver = {
      .name           = "Sample-PCI-Client-driver",        /* Give the
name that suites your driver description */
      .id_table       = sample_pci_id_table,               /* pci core
driver will compare from ids's supplied in this table with the enumerated
endpoints */
      .probe          = sample_pci_probe,                  /* probe
function is invoked by the PCI f/w when id matches with the endpoint id's
*/
      .remove         = sample_pci_remove,                 /* Remove
function is invoked by PCI f/w when senses the attached endpoint/function
is detached or down */
      .driver.pm      = &sample_pci_pm_ops                 /* pm ops that
will be called from the pci core driver*/
};
```

- To register with PCI firmware, call `pci_register_driver(&sample_pci_driver)` from `module_init()`.

  ```
  module_pci_driver(sample_pci_driver);
  ```

7.9 PCIe bringup

For information about PCIe bringup, see PCIe-related configurations and QPS615 switch support.

7.10 PCIe power optimization

PCIe defines two types of power management methods.

- Power management software that determines the power management capability of each device and manages each device individually
- System that doesn't require software intervention such as active state power management (ASPM)

During a time period when no packet is transmitted over the link, a device places the link into a power-saving state.

PCIe L0 link states

PCIe power management defines the following L0 link states:

- L0: active state where all PCIe transactions and other operations are enabled
- L0s: ASPM state with low-resume latency (energy saving standby state)

PCIe device states

PCIe power management defines the following device states:

- D0 (mandatory): The device is in full ON state, where there are two substates
  - $D0_{uninitialized}$: The function is present in the $D0_{uninitialized}$ state after the device comes out of reset, waiting to be enumerated and configured.
  - $D0_{active}$
    - The function is present in the $D0_{active}$ state following the completion of the enumeration and configuration process.
    - The function enters the $D0_{active}$ state when the system software enables one or more (in any combination) function parameters, such as memory space enable, I/O space enable, or BME bits.
- D1 (optional): light-sleep state
  - The function can't initiate a TLP except for the PME message
  - The function can't act as the target of transactions other than for configuration transactions.
  - The function issues a software command to enter the D1 state by programming the PM control and status register.

- D2 (optional): deep-sleep state
  - The function can't initiate a TLP except for the PME message
  - The function can't act as the target of transactions other than configuration transactions.
  - The function issues a software command to enter the D2 state by programming the PM control and status register.
- D3 (mandatory): device is the lowest power state, where the function must support both the D3 states
- $D3_{hot}$
  - The function can't initiate a TLP except for the PME message.
  - The function can't act as the target of transactions other than configuration transactions.
  - The function issues a software command to enter the $D3_{hot}$ state by programming the power state field.
- $D3_{cold}$: device enters the $D3_{cold}$ state and power is removed; when power is restored, the device enters the $D0_{uninitialized}$ state.

7.11 PCIe verification

For information about PCIe verification, see PCIe-related configurations and QPS615 switch support.

7.12 Debug PCIe issues

The `lspci` and `setpci` commands are native to Linux distributions. These commands have various levels of output. These commands also provide a useful point-in-time look at the capabilities and status of the different components trained on the PCI bus. Most of these capabilities are reflections of the configuration space registers required by the PCIe base specification. For more details, see https://pcisig.com/specifications. To view the usage instructions, run the following command.

```
lspci --help
```

The following features are useful in troubleshooting PCIe issues.

- Display device information

  ```
  lspci
  ```

  The following message is displayed.

  ```
  0001:00:00.0 PCI bridge: Qualcomm Device 010b
  0001:01:00.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:01.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:02.0 PCI bridge: Toshiba Corporation Device 0623
  0001:02:03.0 PCI bridge: Toshiba Corporation Device 0623
  0001:04:00.0 USB controller: Renesas Technology Corp. uPD720201 USB 3.0
  Host Controller (rev 03)
  0001:05:00.0 Ethernet controller: Toshiba Corporation Device 0220
  0001:05:00.1 Ethernet controller: Toshiba Corporation Device 0220
  ```

- Display PCIe device and vendor IDs in the device control register.

  ```
  lspci -nvmm
  ```

  The following message is displayed.

```
Slot:   0001:00:00.0
Class:  0604
Vendor: 17cb
Device: 010b
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0
IOMMUGroup:     33

Slot:   0001:01:00.0
Class:  0604
Vendor: 1179
Device: 0623
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0
IOMMUGroup:     33

Slot:   0001:02:01.0
Class:  0604
Vendor: 1179
Device: 0623
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0/pcie1_bus2_dev1_fn0
IOMMUGroup:     33

Slot:   0001:02:02.0
Class:  0604
Vendor: 1179
Device: 0623
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0/pcie1_bus2_dev2_fn0
IOMMUGroup:     33

Slot:   0001:02:03.0
Class:  0604
Vendor: 1179
Device: 0623
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0/pcie1_bus2_dev3_fn0
IOMMUGroup:     33

Slot:   0001:04:00.0
Class:  0c03
Vendor: 1912
Device: 0014
Rev:    03
ProgIf: 30
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0/pcie1_bus2_dev2_fn0/pcie1_bus4_dev0_fn0
IOMMUGroup:     33

Slot:   0001:05:00.0
Class:  0200
Vendor: 1179
Device: 0220
SVendor:        1179
SDevice:        0001
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0/pcie1_bus2_dev3_fn0/
qps615_eth0,qps615_eth0@pcie1_rp
IOMMUGroup:     33

Slot:   0001:05:00.1
```

```
Class:  0200
Vendor: 1179
Device: 0220
SVendor:        1179
SDevice:        0001
DTNode: /sys/firmware/devicetree/base/soc@0/pci@1c08000/pcie@0/
pcie1_bus1_dev0_fn0/pcie1_bus2_dev3_fn0/
qps615_eth1,qps615_eth1@pcie1_rp
IOMMUGroup:     33
```

For more info on PCIe debugging, see https://www.kernel.org/doc/html/v4.17/driver-api/pci.html.

7.13 PCIe examples

For information about the upstream device tree reference, see the following files.

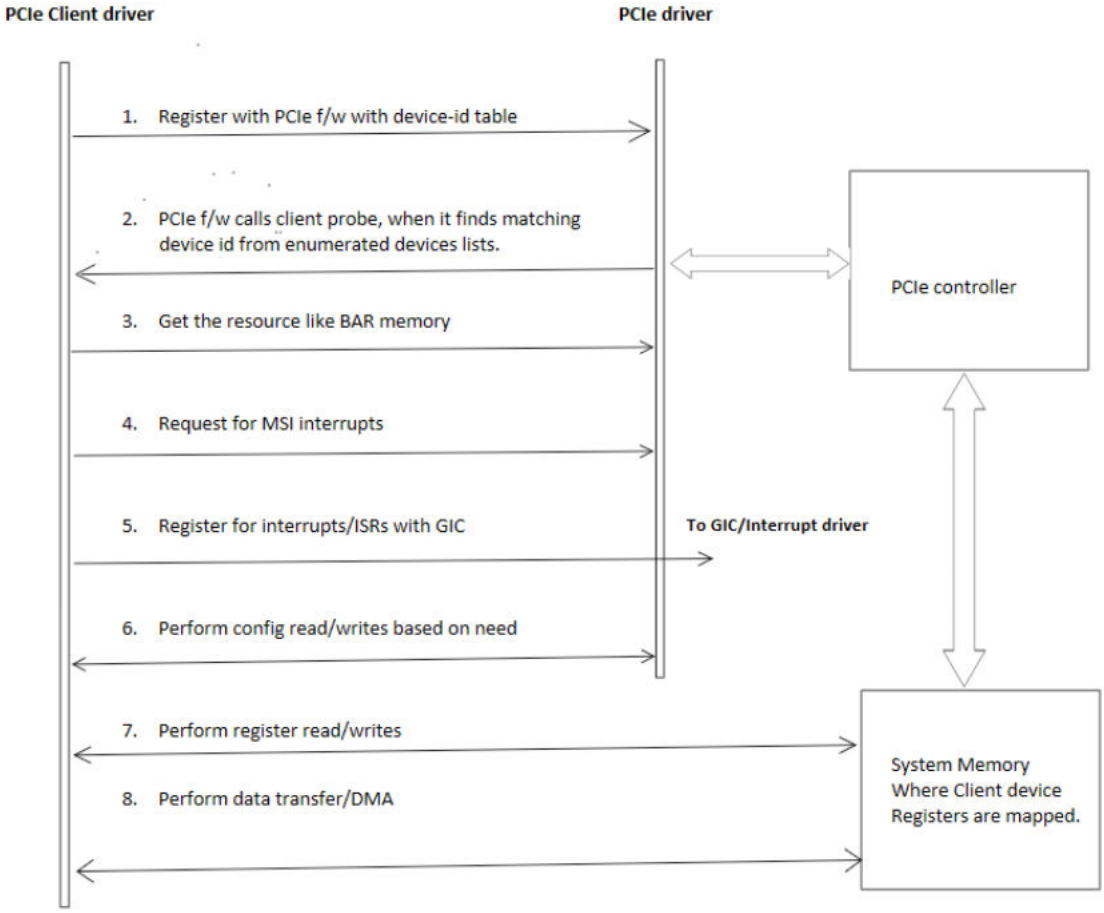
- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi
- https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/pci/controller/dwc/pcie-qcom.c?h=v6.8-rc6#n1634

For information about device-tree node for the Qualcomm Linux hardware SoCs, see the following DTSI files.

- QCS6490 and QCS5430: https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
- QCS9075: https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/sa8775p.dtsi

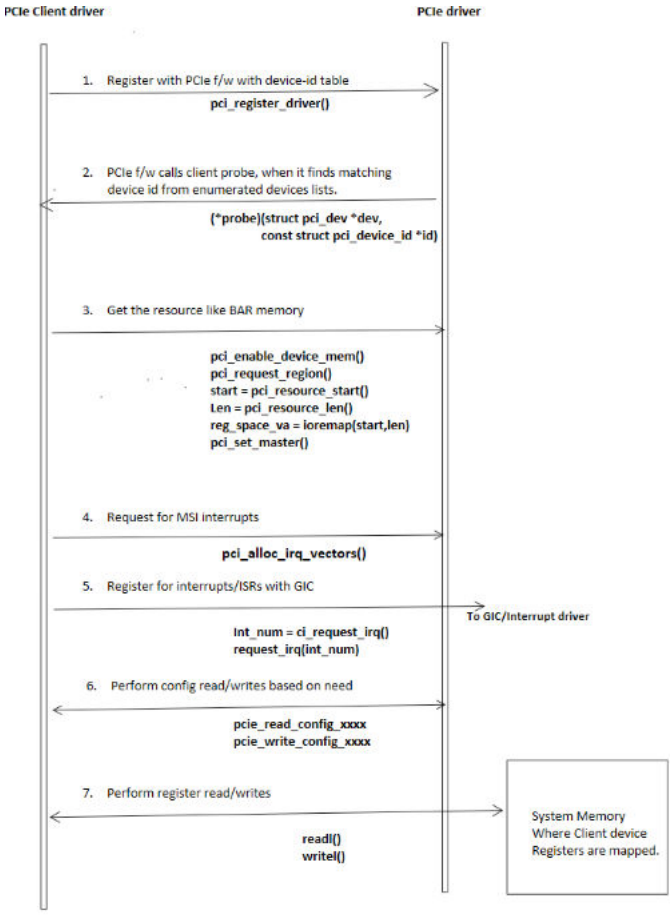
Client and PCI driver operation flow example

The following figure shows the sequence that the PCIe client driver follows to configure the PCIe driver for a client.



Client and PCI driver high-level call flow example

The following figure shows the high-level call flow and call details between the PCIe client driver and PCIe driver.



8 USB

Universal serial bus (USB) is an industry standard that allows data exchange and delivery of power between various types of electronics. It can run at different speeds such as low speed at 1.5 Mbps, full speed at 12 Mbps, high speed at 480 Mbps, SuperSpeed at 5 Gbps, and SuperSpeed Plus at 10 Gbps.

Synopsys DesignWare® Core SuperSpeed USB 3.x powers the USB on the Qualcomm SoC and is connected to two PHYs, USB2 PHY over UTMI and USB3 PHY over PIPE interfaces. These PHYs are wired to the physical Type-C port and facilitate communication to the external world.

The following are the key hardware components of the USB.

- **USB controllers**
  - The primary controller is a Synopsys DesignWare Core SuperSpeed USB 3.x controller (Gen1/Gen2).
    - Two instances of Qualcomm multipurpose PHY (QMP) for USB SuperSpeed and DisplayPort.
    - Synopsys PHY for high-speed USB.
  - The secondary controller is a Synopsys DesignWare Core high-speed USB 2.0 controller.
    - Synopsys PHY for high-speed USB
  - The tertiary controller is a Synopsys DesignWare Core high-speed USB 2.0 controller.
    - Synopsys PHY for high-speed USB.
  - QCS9075 has three USB controllers (primary USB 3.2, secondary USB 3.2, tertiary USB 2.0)
- **Synopsys DesignWare core SuperSpeed USB 3.x controller features**
  - Synopsys DesignWare Core SuperSpeed USB 3.x controller is a USB SuperSpeed-compliant controller, which can be configured in one of the following ways:
    - Peripheral-only configuration
    - Host-only configuration
    - Dual-role configuration
  - Supports all transfer types (control, bulk, interrupt, and isochronous)
  - Supports SuperSpeed bulk streams
  - Compliant with the eXtensible host controller interface (xHCI) specification
  - Host mode supports SuperSpeed (5 Gbps), high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations.

- Device mode supports SuperSpeed (5 Gbps), high-speed (480 Mbps), and full-speed (12 Mbps) operations, and up to 16 bidirectional endpoints (including the control pipe `ep0`).
- Link power management

- **USB PHY access method**
  - Register-level interface through AHB2PHY for performing PHY-related operations.
- **USB Type-C**
  - Supports USB Type-C and power delivery using the PM7325B PD controller.
  - Fully compliant with the USB Type-C 3.0 power delivery specifications.
  - Supports the PM7325B software driver updates according to the UCSI framework after the delivery controller determines the Type-C orientation, role, and mode of the connected link partner.
  - Used only for the primary USB controller.

Clocks

The following tables list the clocks and operating frequencies required for the USB controller, and the high speed and SuperSpeed PHYs to function.

Table 8-1 USB controller clocks

Clock name	Operating frequency	Description
`gcc_usb30_prim_master_clk "core_clk"`	■ 200 MHz SuperSpeed ■ 66 MHz high speed	Asynchronous to the bus clock; clock rates defined within the device tree node
`gcc_cfg_noc_usb3_prim_axi_clk "iface_clk"`	■ 200 MHz SuperSpeed ■ 66 MHz high speed	Auxiliary bus controller unit clock.
`gcc_aggre_usb3_prim_axi_clk "bus_aggr_clk"`	■ 200 MHz SuperSpeed ■ 66 MHz high speed	Clock that feeds into the `aggregator2` module, which controls data flow from the USB AXI to the NoC
`gcc_usb30_prim_mock_utmi_clk "utmi_clk"`	19.2 MHz	The internal controller `ref_clk` used for generating the ITP counter when the USB transceiver macrocell interface (UTMI)/UTMI+ Low Pin interface (ULPI) is suspended
`gcc_usb30_prim_sleep_clk "sleep_clk"`	32 kHz	Sleep clock
`gcc_usb3_sec_clkref_clk_en "xo"`	19.2 MHz	External reference clock source to the HS-PHY (PMIC)
`rpmh_cxo_clk "ref_clk_src"`	19.2 MHz	Reference clock source to the HS-PHY
`gcc_usb_phy_cfg_ahb2phy_clk "cfg_ahb_clk"`	100 MHz	Clock required for the AHB2PHY block (frequency based off PNoC frequency).

Table 8-2 High-speed PHY clocks

Clock name	Operating Frequency	Description
rpmh_cxo_clk "ref_clk_src"	19.2 MHz	Reference clock source to the HS-PHY
gcc_usb_phy_cfg_ahb2phy_clk "cfg_ahb_clk"	100 MHz	Clock required for the AHB2PHY block (frequency based off PNoC frequency)

Table 8-3 SuperSpeed PHY clocks

Clock name	Operating Frequency	Description
gcc_usb3_prim_phy_aux_clk "aux_clk"	19.2 MHz	PHY interface to PCI express (PIPE) auxiliary clock for power states
gcc_usb3_prim_phy_pipe_clk "pipe_clk"	125 MHz	Input source for the PIPE, which allows for data transfers between PHY and controller
rpmh_cxo_clk "ref_clk_src"	19.2 MHz	Parent clock for ref_clk
gcc_usb3_prim_clkref_clk "ref_clk"	19.2 MHz	Reference clock source to the SS-PHY
gcc_usb3_prim_phy_com_aux_clk "com_aux_clk"	19.2 MHz	–

Voltage rails

The following table lists the required voltage rails for the HighSpeed and SuperSpeed PHYs.

Table 8-4 USB voltage rails

Voltage rails	Voltage levels (MAX/NOM/MIN)			Device mode	Description
VREG L1C	1.8	1.7	0	Primary and secondary HS-PHY	1.8 V regulator used by both HS-PHYs in the system
VREG L2B	3.3	3.05	0	Primary and secondary HS-PHY	3.3 V regulator used by both HS-PHYs in the system
VREG L1B	0.912	0.912	0	SS-PHY	SS-PHY VDD core
VREG L10C	0.880	0.880	0	Primary and secondary HS-PHY, and SS-PHY	HS-PHY/SS-PHY VDD core

QCS9075 USB voltage rails

- HS PHY: L7A (0.88 V), L6C (1.8 V), L9A (3.3 V)
- SS PHY: L1C, L7A

Interrupts

The following table lists the various interrupts used by the USB controller to notify events.

Table 8-5 USB controller interrupts

Interrupt name	QCS6490 / QCS5430 interrupts	QCS9075 interrupts			QCS8275 interrupts		QCS615 interrupts		Interrupt events PDC wake-up kernel handling		Description
dp_hs_phy_irq	14	PDC14	PDC8	PDC10	PDC14	PDC10	9	10	D+ changes	–	Only used when the system is in VDD min/XO shutdown
dm_hs_phy_irq	15	PDC15	PDC7	PDC9	PDC15	PDC9	8	11	D-changes	–	Only used when the system is in VDD min/XO shutdown
ss_phy_irq	17	PDC12	PDC13	–	PDC12	–	6	–	LFPS detection	–	Only used when the system is in VDD min/XO shutdown
pwr_event_irq	130	287	352	444	131	444	130	663	–	USB PHY power state changes Exit/enter P3/L2	Used as the main controller wake-up handle when the system isn't in power collapse
core irq	133	292	349	442	292	442	133	664	–	■ Bus events □ Suspend □ Resume □ Reset ■ Controller event completion □ Transfer completion □ Command completion	Main USB interrupt that handles all USB controller events

Interconnect

The following table lists the various interconnects used by the USB controller.

Table 8-6 USB interconnects

Interconnect name	Controller	Target	Interconnect path bandwidths in MBps
USB-DDR	▪ MASTER_USB3_0 ▪ QCS9075 ▫ MASTER_USB3_1 ▫ MASTER_USB2 ▪ QCS8275, QCS615 ▫ MASTER_USB2	SLAVE_EBI1	▪ USB_MEMORY_AVG_HS_BW MBps_to_icc(240) ▪ USB_MEMORY_PEAK_HS_BW MBps_to_icc(700) ▪ USB_MEMORY_AVG_SS_BW MBps_to_icc(1000) ▪ USB_MEMORY_PEAK_SS_BW MBps_to_icc(2500)
APPS-USB	MASTER_APPSS_PROC	▪ SLAVE_USB3_0 ▪ QCS9075 ▫ SLAVE_USB3_1 ▫ SLAVE_USB2 ▪ QCS8275, QCS615 ▫ SLAVE_USB2	▪ APPS_USB_AVG_BW 0 ▪ APPS_USB_PEAK_BW MBps_to_icc(40)

USB controller reset using clock control

The following table lists the reset methods used for the USB controller and PHY.

Table 8-7 USB clock reset methods

Clock name	Reset control	Description
GCC_USB3_DP_PHY_PRIM_BCR "global_phy_reset"	SS-PHY	Resets the SS-PHY control and status registers
GCC_USB3_PHY_PRIM_BCR "phy_reset"	SS-PHY	Resets the SS-PHY
GCC_QUSB2PHY_PRIM_BCR "phy_reset"	HS-PHY	Resets the HS-PHY
GCC_USB30_PRIM_BCR "core_reset"	USB controller	Clock controller output to reset the USB controller

USB controller software reset using register

The following table lists the register options to reset the USB controller.

Table 8-8 USB controller resets

USB controller register	USB register bit field	Reset control	Description
DWC3_DCTL	CSFTRST [Bit 30]	USB controller	Resets the USB controller device stack.
DWC3_GCTL	CORESOFTRESET [Bit 11]	-	Global reset for the DWC3 controller

QCS9075 controller reset registers

- HS: GCC_USB2_PHY_PRIM_BCR
- SS: GCC_USB3_PHY_PRIM_BCR/ GCC_USB3PHY_PHY_PRIM_BCR
- GCC_USB3_PHY_TERT_BCR
- USB30_PRIM_GDSC
- USB30_SEC_GDSC
- USB20_PRIM_GDSC

QCS615 secondary USB controller reset registers

- GCC_USB20_SEC_BCR
- GCC_QUSB2PHY_SEC_BCR
- GCC_USB2_PHY_SEC_BCR

USB controller and SoC integration

The intellectual property and PHYs of the USB controller are integrated into the SoC as shown in the following figure.

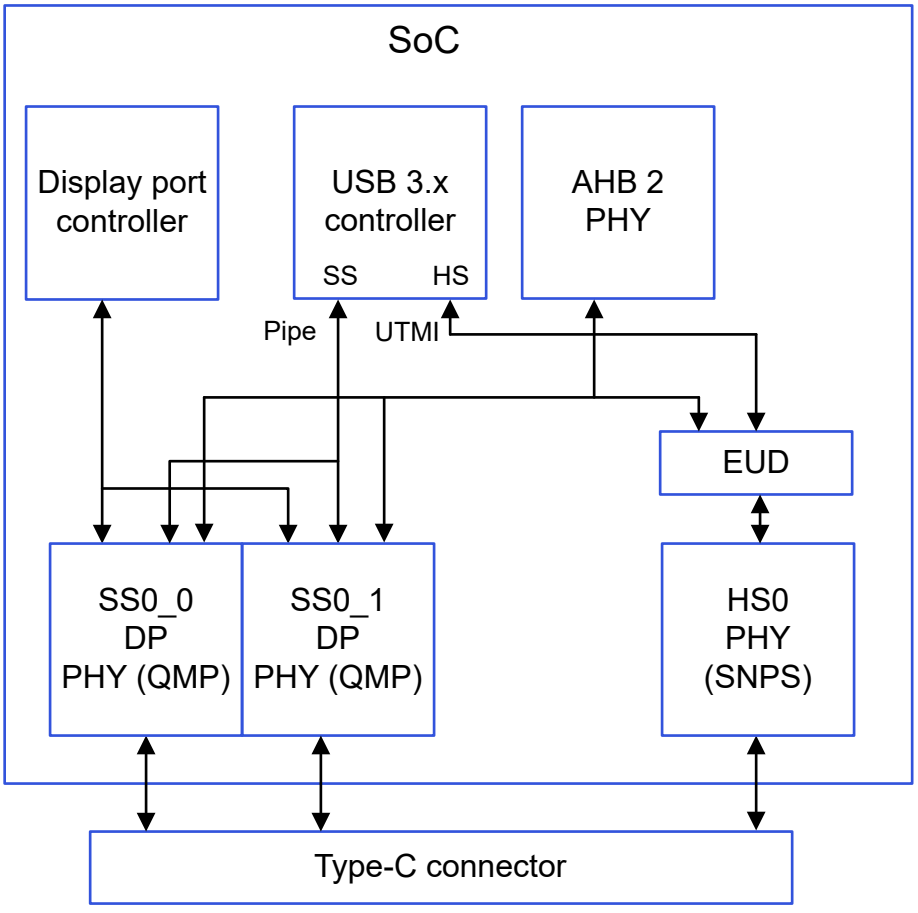


Figure 8-1 USB controller PHYs and SoC integration

The Synopsys DesignWare Core SuperSpeed USB 3.0 intellectual property controls only the core functionality and not the device specifications, such as clocks, interconnects, regulators, and GDSCs. The Synopsys DesignWare Core intellectual property is embedded inside a `Qscratch` wrapper (intellectual property and software driver), which takes up the responsibility of managing the required resources (clocks, interconnects, interrupts, GDSC, and regulators) during the probe, suspends, or resume state. Both these drivers coexist to ensure the USB functionality.

For information about the USB `Qscratch` wrapper driver, see https://github.com/torvalds/linux/blob/master/drivers/usb/dwc3/dwc3-qcom.c. For information about the controller core driver, see https://github.com/torvalds/linux/blob/master/drivers/usb/dwc3/core.c.

xHCI support

The xHCI specification describes the register-level host controller interface for USB 2.0 and later. The USB controller is compliant with xHCI specifications, and in host mode it supports SuperSpeed (5 Gbps), high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operations.

Linux standard xHCI drivers are used to operate the USB controller in host mode.

USB Type-C connector system software interface (UCSI)

NOTE QCS9075 and QCS8275 don't support USB Type-C feature.

- The USB Type-C connector system software interface is available in the Linux kernel as a library. It defines a set of registers and data structures, which are used to interface with the USB Type-C connectors on a system. The USB Type-C connectors on the platform are referred to as the platform policy manager (PPM) and the system software component is referred to as the OS policy manager (OPM).
- Qualcomm reference design uses the UCSI `Glink` driver to handle the communication between the OPM on the application processor and the PPM, which is the charger firmware running on the remote subsystem (aDSP) over PMIC GLINK.
- The USB Type-C DisplayPort alternate mode adds the capabilities of supporting additional cable details, such as DPAM version and signaling of cable. The Qualcomm reference design supports DisplayPort over Type-C and concurrent support of USB SuperSpeed and DisplayPort with a maximum of two lanes for DisplayPort (two lanes for USB SuperSpeed and two lanes for DisplayPort).
- The USB Type-C power delivery is supported by the PM7325B PD controller, fully compliant with the USB PD 3.0 specification. Using the UCSI interface, USB data-role swap and power-role swap are supported.

  NOTE QCS9075 and QCS8275 don't support data and power role swapping over Type-C connectors.

- Following are the references of the drivers involved:
  - https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/typec/ucsi/ucsi.c?h=v6.6.2
  - https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/typec/ucsi/ucsi_glink.c?h=v6.6.2
  - https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/typec/ucsi/displayport.c?h=v6.6.2
  - https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/phy/qualcomm/phy-qcom-qmp-combo.c?h=v6.6.2

- https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/dwc3/dwc3-qcom.c?h=v6.6.2
- https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/soc/qcom/pmic_glink.c?h=v6.6.2
- https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/soc/qcom/pmic_glink_altmode.c?h=v6.6.2

8.1 USB features

Qualcomm chip hardware SoCs allow working in the dual-role device (DRD) mode. DRD enables both device and host roles. The device dynamically detects the role to move to and programs the controller accordingly. The autosuspend capability is supported to shut down the USB controller when the cable is removed. Function interfaces, such as video, audio, tethering, file transfer, media transfer, and charging are supported.

Table 8-9 USB features: Linux

Feature	Description
USB_DWC3 controller	Synopsys DesignWare Core is supported by default in the Linux kernel.
USB_DWC3_QCOM	`Qscratch` wrapper using the Synopsys DesignWare Core for the USB functionality.
Runtime power management (RPM)	Linux supports RPM with the software driver file `dwc3-qcom`, for low-power mode operations.
Low-power mode (LPM)	LPM provides power-saving options from both HS/SS-PHY as well as the controller.
DRD	Dual-role device detection allows USB to work in both host and device mode.

Runtime power management

The Runtime power management feature can be enabled according to the your requirements. LPM support is added as part of https://lore.kernel.org/all/20231017131851.8299-1-quic_kriskura@quicinc.com/.

NOTE QCS9075 and QCS8275 don't support runtime power management for USB Type-A ports.

By default, USB-suspend and resume in runtime are disabled. To enable these features, run the following command:

```
echo auto > /sys/bus/platform/devices/a600000.usb/power/control
```

The default autosuspend delay is set to 5 s. To change the delay, run the following `sysfs` command:

```
echo 2000 > /sys/bus/platform/devices/a600000.usb/power/autosuspend_delay_ms
```

During a role switch, the DWC3 controller enters the suspend state, toggles GDSC, and resets the controller to ensure successful host mode peripheral enumeration.

While in host mode, for remote wake-up to work, wake-up, and autosuspend are enabled for the xHCI interface, USB root hubs, and the connected peripherals according to your requirements.

```
echo enabled > /sys/bus/platform/devices/xhci-hcd.X.auto/power/wakeup
cd /sys/bus/usb/devices/
echo enabled > usb1/power/wakeup
echo enabled > usb2/power/wakeup
```

NOTE The host controller driver (HCD) device generates the `xhci-hcd.X.auto` value, where *X* = 0, 1, 2. It indicates the number of devices connected to the USB. The connected devices are listed at `/sys/bus/platform/devices/xhci-hcd.X.auto/usb1/<node number>`. The`<node number>` value depends on the number of devices connected to the USB port.

To enable the remote wake up for an LS optical mouse, run the following commands:

```
cd /sys/bus/usb/devices/usb1/1-1/
echo auto > power/control
echo enabled > power/wakeup
```

To check the runtime status, run the following command:

```
cat /sys/bus/platform/devices/a600000.usb/power/runtime_status
```

To avoid suspending in the composition switch, remove the active UDC to rebind the composition by running the following command:

```
echo on > /sys/bus/platform/devices/a600000.usb/power/control
echo auto > /sys/bus/platform/devices/a600000.usb/power/control
```

8.2 USB architecture

The Qualcomm USB software architecture is loosely made up of two components, one is based on pure upstream, and another is a sandbox where some of the pending feature-related changes are present. The pure upstream is directly picked from the latest stable kernel long-term support (LTS) 6.6.2. The architecture uses a Yocto (release 4.0) recipe for creating the binaries.

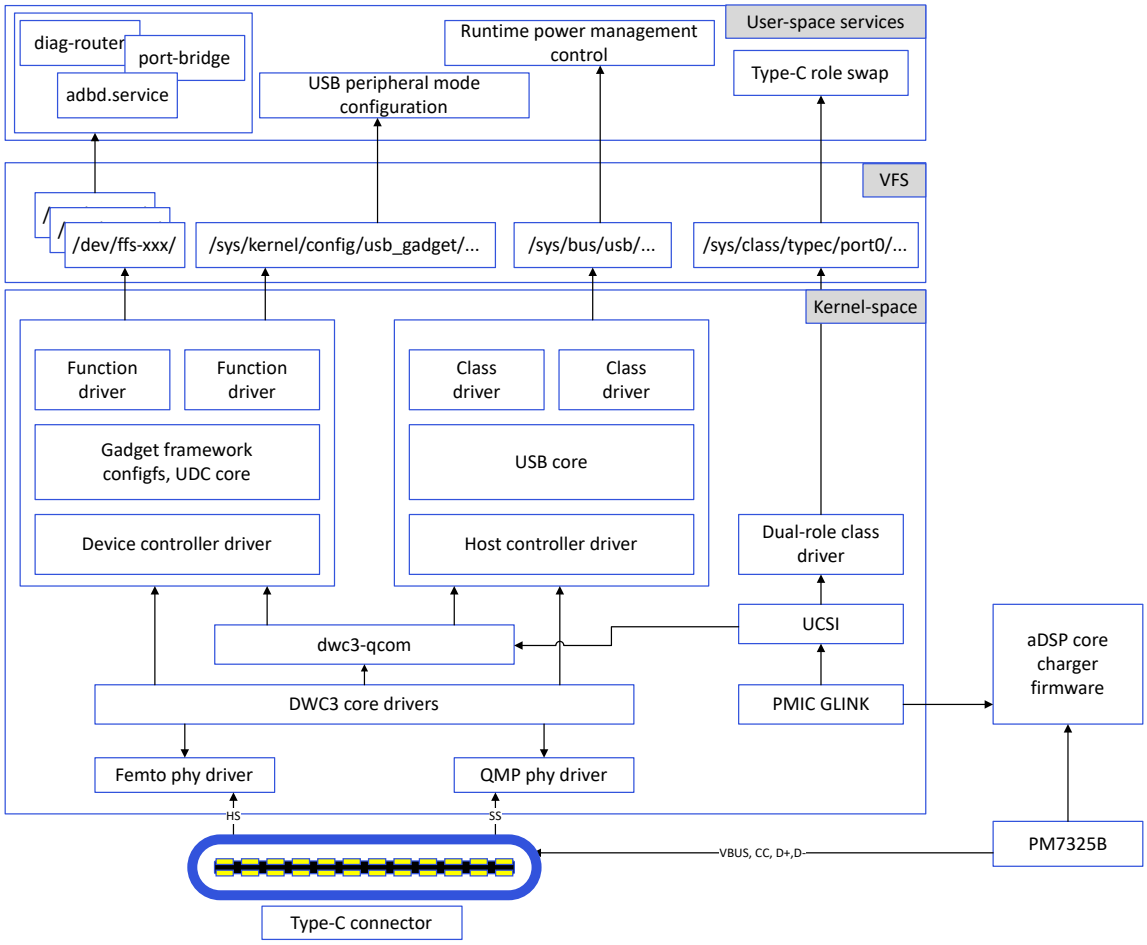


Figure 8-2 USB software architecture

The sandbox is implemented with the following Qualcomm-specific features.

- USB_DWC3_QCOM: The primary glue driver is responsible for the USB functionality on both host and device modes, depending on what's connected. The device tree entry for the glue driver consists of resources, such as clocks, power rails, and interrupts. This feature uses the DWC3 core driver as a library, which controls the actual functionality of the DWC3 controller. For more details, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/dwc3/dwc3-qcom.c?h=v6.6.2.
- Synopsys femto PHY: Qualcomm Synopsys femto PHY is the high-speed USB PHY responsible for controlling D+/D− lines, and facilitates data transfers along with charger detection. For more

details, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/phy/qualcomm/phy-qcom-snps-femto-v2.c?h=v6.6.2.

- QMP DisplayPort combo PHY: Qualcomm multipurpose PHY (QMP) display port combo PHY is designed for generic usage. A sample use case for USB would be the SuperSpeed and SuperSpeed-plus functionality as part of which the `Rx+/Rx-` and `Tx+/Tx-` can be used for data transfer. The PHY supports the display alternate mode when the display can claim a pair of lanes for the mirroring functionality while USB works in SuperSpeed. For more details, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/phy/qualcomm/phy-qcom-qmp-combo.c?h=v6.6.2.

8.3 USB interfaces

USB supports multiple interfaces to transfer audio, video, debug information, and tethering. Each of the following USB interface communication protocol is different and has its own protocols in addition to the standard USB protocol.

- Android debug bridge (ADB)
  - The USB ADB is a debug interface, providing access to the system through the USB connection. For more information about ADB, see https://developer.android.com/tools/adb.
  - The filenames are the virtual `eps` names of the `dev` node. For more information about `f_fs.c`, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/gadget/function/f_fs.c?h=v6.6.2.
  - User space service: The user space daemon `adbd.service` operates ADB and is responsible for establishing the connection between the underlying kernel driver and the host PC.
- Diagnostics (diag)
  - Diag is a diagnostics framework used for collecting log data from various subsystems, and debugging. The diag data is transmitted over USB to the host PC.
  - Kernel driver: Diag uses `f_fs.c` to expose the `/dev/ffs-diag` node, which is operated by a user space service for various operations. The `dev` node has the following three files:
    - `ep0`: control operations
    - `ep1`: read operations
    - `ep2`: write operations

    These filenames are the virtual `eps` names of the `dev` node. For more information about `f_fs.c`, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/gadget/function/f_fs.c?h=v6.6.2.
- Mass storage
  - A generic interface driver with mass storage driver is available for the device to act as a generic storage device.
  - Kernel driver: The `mass_storage` functionality is regulated by the `f_mass_storage.c` driver. This driver exposes `dev node[revisit]`, which is controlled by the host PC. For more information about mass storage, see https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/gadget/function/f_mass_storage.c?h=v6.6.2.

- Remote network driver interface specification (RNDIS)
  - RNDIS is a specification developed by Microsoft for network devices on dynamic plug and play I/O buses such as USB.
  - Kernel driver: It uses a `f_rndis` driver, which is present in the upstream Linux kernel and directly communicates with the network interfaces or stack.
- Network control model (NCM)
  - NCM is a protocol designed to offer advanced features and capabilities compared to RNDIS.
  - It's commonly found in applications where USB devices must handle demanding networking tasks.
- USB audio class 2 (UAC2)
  - UAC2 is a standard governing the communication between USB audio devices and computers.
  - UAC2 supports higher audio data transfer rates, resulting in improved audio quality and reduced latency.
- USB video class (UVC)
  - UVC is a standard that defines how video streaming devices, such as webcams, should communicate with computers over USB.
  - UVC facilitates the plug-and-play functionality for video devices.

8.4 USB software

The following USB features are supported in software.

Table 8-10 USB software features

Feature	Description
Peripheral ADB	ADB functionality is integrated over functionFS (FFS)
Peripheral mass storage	Generic Mass_storage functionality
Peripheral diag	Diag functionality is integrated over FFS
Peripheral RNDIS	Standard RNDIS protocol according to Linux kernel
Peripheral network control modem (NCM)	Standard NCM protocol according to Linux kernel
Host USB 3.0 driver (xHCI)	Both USB controllers support the xHCI architecture
Host high-speed USB	High-speed USB detection in host mode
Host HID/MS/hub driver	Class driver detection in host mode
Host video driver	UVC verified on limited webcam models
Host link power management	USB host mode LPM implementation
Peripheral USB link power management	USB device mode LPM implementation
DRD	Dual-role device support is possible (host/device mode support)
USB Type-C	Supported by PM7325B; supports current charging, CC logic, and SuperSpeed USB switch selection, all performed in the embedded controller

Table 8-10 USB software features (cont.)

Feature	Description
USB Type-C display port	Supports SuperSpeed USB + DisplayPort operating concurrently (2 DisplayPort lanes)
USB PD 2.0/3.0 charging	PD support depends on the PM7325B solution used. Fully compliant with PD 3.0
USB 3.1 Gen1	The USB controller supports USB 3.x Gen1 (5 Gbps).

8.5 USB tools

A few commonly used USB tools are listed in the following table.

Table 8-11 USB tool and download details

USB tools	Download link
Platform tools (adb/fastboot)	https://developer.android.com/tools/releases/platform-tools
USB video class (UVC) LibUVC	■ https://github.com/libuvc/libuvc ■ https://libuvc.github.io/libuvc/
UVC gadget	https://github.com/wlhe/uvc-gadget
UVC streamer	https://github.com/bsapundzhiev/uvc-streamer
UVC Video4Linux (v4l2-utils)	https://linuxtv.org/downloads/v4l-dvb-apis/driver-api/v4l2-core.html

8.6 Configure USB boot loader

The device tree parameters for tuning the high-speed and SuperSpeed USB signal quality in the boot loader can be modified using the Qualcomm DeviceTree editor (QDTE) tool.

QDTE tool is used to configure the device tree binary blob by editing the `xbl_config.elf` file, as shown in the following figure. For more information about how to configure the device tree blobs, see the QDTE section. The device tree file path at the Linux host machine is `/boot_images/boot/Settings/Soc/<Chipset>/Core/WiredConnectivity/USB/usb.dtsi`.

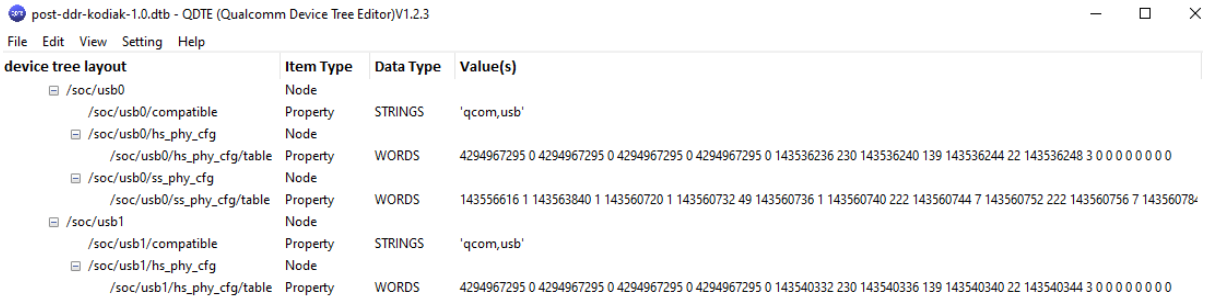


Figure 8-3 Device tree layout

The following table lists the properties for tuning the HS-USB PHY and SS USB PHY signal quality.

Table 8-12 USB configuration properties

Property name	Property description	Data type	Possible values and value range	Device behavior
path=/soc/usb0/hs_phy_cfg	Tunes the USB signal quality for the primary USB controller HS-PHY.	UINT32-array	▪ This property is an array of address, value pairs <addr, val> ▪ addr is 4 bytes in length. It can have 1 of the 4 values, range: [0x88E306C, 0x88E3070, 0x88E3074, 0x88E3078] ▪ val is of 1 byte in length, range: 0x00 to 0xFF	Improved USB signal quality for the primary USB HS-PHY.
path=/soc/usb0/ss_phy_cfg	Improves signal quality for primary USB controller SS-PHY.	UINT32-array	▪ This property is an array of address, value pairs <addr, val> ▪ addr is 4 bytes in length. Range: [0x088E8000, 0x088EB000] ▪ val is of 1 byte in length, range: 0x00 to 0xFF	Improved signal quality for the primary USB SS-PHY.
path=/soc/usb1/hs_phy_cfg	Improves the USB signal quality for the secondary USB controller HS-PHY.	UINT32-array	▪ This property is an array of address, value pairs <addr, val> ▪ addr is 4 bytes in length. It can have 1 of the 4 values, range: [0x88E406C,0x88E4070, 0x88E4074, 0x88E4078] ▪ val is of 1 byte in length, range: 0x00 to 0xFF	Improved signal quality for secondary USB HS-PHY.
Path = /sw/usb_config/fastboot_core_num	Select the USB core number for Fastboot (primary=0/ secondary=1)	UINT32	0 or 1	The Fastboot device is enumerated at either USB core0 or core1 according to the selected property.

8.7 Configure USB camera

The Qualcomm Linux devices provide driver support for USB web cameras that adhere to the USB video class (UVC) standard. The uvcvideo driver of the Linux kernel supports cameras. For more information about the uvcvideo driver, see https://www.kernel.org/doc/html/v4.19/media/v4l-drivers/uvcvideo.html.

The uvcvideo driver exposes these cameras as V4L2 video devices, which can be accessed through the character device nodes such as /dev/videoX.

In the user space, applications can manage USB cameras using the `v4l2src` GStreamer plug-in, which comes bundled with the Qualcomm Intelligent Multimedia SDK (IM SDK). Alternatively, programs such as Yavta (yet another V4L2 test application) directly interact with the V4L2 (Video4Linux2) interface to test and control camera devices.

The current release doesn't include the Yavta program by default. To obtain and cross-compile Yavta, do the following:

1. The cross-compilation environment can be established using any of the following methods.
   - Method 1: To set up the cross-compilation environment, run the following command.

     ```
     sudo apt install gcc-aarch64-linux-gnu
     ```

   - Method 2: To set up the cross-compilation environment, run the following commands:
     i. Download the cross-compiler.

     ```
     wget https://releases.linaro.org/archive/14.07/components/toolchain/
     binaries/gcc-linaro-aarch64-linux-gnu-4.9-2014.07_linux.tar.xz
     ```

     ii. Extract the cross-compiler.

     ```
     tar -xf gcc-linaro-aarch64-linux-gnu-4.9-2014.07_linux.tar.xz
     ```

     iii. Set up the environment for cross-compilation by running the following command.

     ```
     export PATH=$PATH:`pwd`/gcc-linaro-aarch64-linux-
     gnu-4.9-2014.07_linux/bin
     ```

2. Clone the Yavta repository and change the directory.

   ```
   git clone https://github.com/fastr/yavta.git
   cd yavta
   ```

3. Cross-compile the tool.

   ```
   make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
   ```

4. To push Yavta to the Qualcomm Linux hardware SoCs, do the following:
   a. Open the SSH shell in permissive mode or use the ADB shell. For more information about how to run SSH, see the Use SSH section.
   b. Mount the file system.

   ```
   mount -o remount,rw /usr
   ```

   c. Transfer files using SCP or similar tools. For more information about how to run SSH, see the Use SSH section.

   For example, `scp yavta root@10.92.162.185:/usr/bin`

   d. Assign permission to execute in Yavta.

   ```
   chmod 0777 /usr/bin/yavta
   ```

Prerequisite: Obtain image format and size

To configure the USB camera either through Yavta or GStreamer, the following steps are mandatory.

1. To know the enumeration details, plug in the USB camera and run the following command.

   ```
   lsusb
   ```

   The following output is displayed.

```
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 03f0:0959 HP, Inc w200
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

2. Identify the USB camera video node created as `/dev/videoX` from the serial console.

   ```
   ls /sys/bus/usb/devices/1-1/1-1:1.0/video4linux/
   ```

   NOTE The `1-1/1-1:1.0` value varies according to the USB device connected.

   The following output is displayed.

   ```
   video4    video5
   ```

3. To view the supported output formats and sizes using Yavta, run the following command:

   ```
   yavta /dev/video4 --enum-formats
   ```

   The following output is displayed.

   ```
   Device /dev/video4 opened: w200: w200 (usb-xhci-hcd.2.auto-1).
   - Available formats:
       Format 0: MJPG (47504a4d)
       Type: Video capture (1)
       Name: Motion-JPEG
       Frame size: 1280x720 (1/30, 1/25, 1/20, 1/15, 1/10, 1/5)
       Frame size: 800x600 (1/30)
       Frame size: 640x480 (1/30)
       Frame size: 320x240 (1/30)

       Format 1: YUYV (56595559)
       Type: Video capture (1)
       Name: YUYV 4:2:2
       Frame size: 1280x720 (1/10)
       Frame size: 800x600 (1/15)
       Frame size: 640x480 (1/30)
       Frame size: 320x240 (1/30)
   ```

4. Select the required image format and size from step 3.

Configure USB camera using Yavta

Prerequisite: Ensure that Yavta is cross-compiled, and the output format, size are identified.

- To select MJPEG format of 1280x720 output size and 30 fps, capture 10 frames under `/tmp/` with filename `testmjpeg-00000*.bin`, run the following command.

  ```
  yavta -f MJPEG -s 1280x720 -t 1/30 -c10 -F/tmp/testmjpeg /dev/video4
  ```

  The following output is displayed.

  ```
  Device /dev/video4 opened: w200: w200 (usb-xhci-hcd.2.auto-1).
  Video format set: width: 1280 height: 720 buffer size: 1843789
  Video format: MJPG (47504a4d) 1280x720
  Current frame rate: 1/30
  Setting frame rate to: 1/30
  Frame rate set: 1/30
  8 buffers requested.
  length: 1843789 offset: 0
  Buffer 0 mapped at address 0x7fabd5d000.
  length: 1843789 offset: 1847296
  Buffer 1 mapped at address 0x7fabb9a000.
  ```

```
length: 1843789 offset: 3694592
Buffer 2 mapped at address 0x7fab9d7000.
length: 1843789 offset: 5541888
Buffer 3 mapped at address 0x7fab814000.
length: 1843789 offset: 7389184
Buffer 4 mapped at address 0x7fab651000.
length: 1843789 offset: 9236480
Buffer 5 mapped at address 0x7fab48e000.
length: 1843789 offset: 11083776
Buffer 6 mapped at address 0x7fab2cb000.
length: 1843789 offset: 12931072
Buffer 7 mapped at address 0x7fab108000.
0 (0) [-] 0 57672 bytes 2459.697791 315967245.973463
1 (1) [-] 1 40816 bytes 2459.730553 315967246.005839
2 (2) [-] 2 40472 bytes 2459.763517 315967246.039485
3 (3) [-] 3 41272 bytes 2459.797002 315967246.073073
4 (4) [-] 4 42232 bytes 2459.830192 315967246.105592
5 (5) [-] 5 46024 bytes 2459.863253 315967246.139142
6 (6) [-] 6 47440 bytes 2459.896755 315967246.172384
7 (7) [-] 7 48840 bytes 2459.930006 315967246.205114
8 (0) [-] 8 50248 bytes 2459.963235 315967246.238308
9 (1) [-] 9 53136 bytes 2459.996526 315967246.272690
Captured 9 frames in 0.300183 seconds (29.981711 fps, 1559555.337911
B/s).
8 buffers released.
```

- To select YUV format of 1280x960 output size, 30 fps, capture 10 frames under `/tmp/`, the generated filename is `testyuv-00000*.bin`, run the following command.

  ```
  yavta -f YUYV -s 1280x720 -t 1/30 -c10 -F/tmp/testyuv /dev/video4
  ```

  The following output is displayed.

  ```
  Device /dev/video4 opened: w200: w200 (usb-xhci-hcd.2.auto-1).
  Video format set: width: 1280 height: 720 buffer size: 1843200
  Video format: YUYV (56595559) 1280x720
  Current frame rate: 1/10
  Setting frame rate to: 1/10
  Frame rate set: 1/10
  8 buffers requested.
  length: 1843200 offset: 0
  Buffer 0 mapped at address 0x7f9853e000.
  length: 1843200 offset: 1843200
  Buffer 1 mapped at address 0x7f9837c000.
  length: 1843200 offset: 3686400
  Buffer 2 mapped at address 0x7f981ba000.
  length: 1843200 offset: 5529600
  Buffer 3 mapped at address 0x7f97ff8000.
  length: 1843200 offset: 7372800
  Buffer 4 mapped at address 0x7f97e36000.
  length: 1843200 offset: 9216000
  Buffer 5 mapped at address 0x7f97c74000.
  length: 1843200 offset: 11059200
  Buffer 6 mapped at address 0x7f97ab2000.
  length: 1843200 offset: 12902400
  Buffer 7 mapped at address 0x7f978f0000.
  0 (3) [-] 3 1843200 bytes 2536.837027 315967323.172707
  1 (4) [-] 4 1843200 bytes 2536.937122 315967323.273059
  2 (6) [-] 6 1843200 bytes 2537.136830 315967323.472473
  3 (7) [-] 7 1843200 bytes 2537.237183 315967323.572533
  4 (2) [-] 17 1843200 bytes 2538.237266 315967324.571848
  ```

```
5 (2) [-] 24 1843200 bytes 2538.936539 315967325.271919
6 (3) [-] 25 1843200 bytes 2539.036550 315967325.372403
Warning: bytes used 0 != image size 1843200
7 (7) [E] 28 0 bytes 2539.336529 3159  325.609436
Warning: bytes used 0 != image size 1843200
8 (0) [E] 22 0 bytes 2538.737975 315967325.610149
Warning: bytes used 0 != image size 1843200
9 (1) [E] 23 0 bytes 2538.837814 315967325.610622
Captured 9 frames in 2.438312 seconds (3.691078 fps, 5291529.549951
B/s).
8 buffers released.
```

Configure USB camera using GStreamer in Qualcomm IM SDK

Qualcomm IM SDK uses GStreamer, an open-source multimedia framework to expose easy APIs and plugins in both the multimedia and machine learning domains. For information about installing the Qualcomm IM SDK, see the Getting Started section.

The Qualcomm IM SDK includes the `v4l2src` plug-in, which allows input from USB cameras with a selected format. The `waylandsink` plug-in is responsible for rendering the video output on a Wayland display.

Prerequisite: Ensure that Yavta is cross-compiled, and the output format and size are identified.

1. To set the environment variables for the Wayland display, run the following command in the serial console.

   ```
   export XDG_RUNTIME_DIR=/dev/socket/weston && export
   WAYLAND_DISPLAY=wayland-1
   ```

2. Use GStreamer commands to stream video from the camera to the UI. Ensure that you set the appropriate device ID (`/dev/videoX`) and select the correct format based on the USB camera detection.

   NOTE In GStreamer, the YUYV color format is referred to as YUY2. Hence, you must specify the YUYV format while setting up a pipeline and use **YUY2** in the caps filter.

   - For 720p, run the following command:

     ```
     gst-launch-1.0 -e v4l2src io-mode=dmabuf-import device="/dev/video0" !
     video/x-raw,format=YUY2,width=1280,height=720,framerate=10/1 !
     waylandsink fullscreen=true
     ```

     The following output is displayed.

     ```
     Y2,width=1280,height=720,framerate=10/1 ! waylandsink fullscreen=true
     Setting pipeline to PAUSED ...
     I/Adreno-UNKNOWN (1985,1985): <ReadGpuID:357>: Reading chip ID
     through GSL
     GBM_INFO::msmgbm_mapper(262)::gbm mapper instantiated
     gbm_create_device(224): Info: backend name is: msm_drm
     Pipeline is live and does not need PREROLL ...
     Pipeline is PREROLLED ...
     Setting pipeline to PLAYING ...
     New clock: GstSystemClock
     gbm_create_device(224): Info: backend name is: msm_drm
     GBM_ERR::msmgbm_bo_create(870)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
     for data fd errono: 22 (Invalid argument) drm fd: 24 data fd: 26
      GBM_ERR::msmgbm_bo_create(923)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
     ```

```
for metadata fd errono: 22 (Invalid argument) drm fd: 24 metadata
fd: 27
GBM_ERR::msmgbm_bo_create(870)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
for data fd errono: 22 (Invalid argument) drm fd: 24 data fd: 29
GBM_ERR::msmgbm_bo_create(923)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
for metadata fd errono: 22 (Invalid argument) drm fd: 24 metadata
fd: 30
GBM_ERR::msmgbm_bo_create(870)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
for data fd errono: 22 (Invalid argument) drm fd: 24 data fd: 32
GBM_ERR::msmgbm_bo_create(923)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
for metadata fd errono: 22 (Invalid argument) drm fd: 24 metadata
fd: 33
GBM_ERR::msmgbm_bo_create(870)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
for data fd errono: 22 (Invalid argument) drm fd: 24 data fd: 35
GBM_ERR::msmgbm_bo_create(923)::DRM_IOCTL_PRIME_FD_TO_HANDLE failed
for metadata fd errono: 22 (Invalid argument) drm fd: 24 metadata
fd: 36
Redistribute latency...
0:00:47.7 / 99:99:99.
```

- For 1080p, run the following command:

  ```
  gst-launch-1.0 -e v4l2src io-mode=dmabuf-import device="/dev/video0" !
  video/x-raw,format=YUY2,width=1920,height=1080,framerate=5/1 !
  waylandsink fullscreen=true
  ```

8.8 Customize USB device

This section describes the requirements for various configurations and customizations in the USB software.

Example shell script for a USB composition with diag and ADB interfaces

```
cd /sys/kernel/config/usb_gadget/adb
echo on > /sys/bus/platform/devices/a600000.usb/power/control
echo "" > UDC
mkdir functions/ffs.diag
echo "QCOM" > strings/0x409/manufacturer
echo 0x05c6 > idVendor
echo 0x901d > idProduct
echo "Diag_ADB" > configs/c.1/strings/0x409/configuration

if [ ! -d /dev/ffs-diag ]; then
mkdir -p /dev/ffs-diag
fi
if [ ! -e /dev/ffs-diag/ep0 ]; then
mount -o uid=2000,gid=2000 -t functionfs diag /dev/ffs-diag
fi

/usr/bin/diag-router &

cd configs/c.1
rm -r ffs.usb0
```

```
ln -s ../../functions/ffs.diag f1
ln -s ../../functions/ffs.usb0 f2
cd ../../ udcname=`ls -1 /sys/class/udc | head -n 1`
echo $udcname > UDC
echo auto > /sys/bus/platform/devices/a600000.usb/power/control
```

Change USB composition through QUSB service

`usb.service` starts the QUSB service at `/usr/bin/qusb` to provide users the flexibility to configure the USB gadgets. It completely removes the hassle of manually executing commands to initialize USB from `configfs`.

Prerequisite: Enable SELinux by running the following command.

```
setenforce 0
```

Example:

```
qusb [bind] [unbind] [showpid] [help]
                     [setpid [-p] <PID>] [persist <PID>]
```

Enable and configure UVC use case

The USB video device class (also USB video class or UVC) is a USB device class that describes devices capable of streaming video, such as webcams, digital camcorders, transcoders, analog video converters, and still-image cameras.

The latest revision of the USB video class specification is v1.5. The USB implementers forum describes both the basic protocol and the different payload formats in v1.5.

NOTE QCS9075 and QCS8275 don't support the UVC use case.

The tools and procedures to test the UVC are as follows:

- **UVC gadget** (device side): It's a sample application to test the `f_uvc` functional driver. It opens the video node created by `uvc_gadget` and sends MJPEG frames at a specified frame rate. This application is cross-compiled for the DUT.

  NOTE Ensure that the packages and tools required for cross-compiling a 64-bit Arm® technology compiler are installed on the host machine.

  a. The cross-compilation environment can be established using any of the following options.

  - Option **A**: Run the following command.

    ```
    sudo apt install gcc-aarch64-linux-gnu
    ```

  - Option **B**: To set up the cross-compilation environment, run the following commands:

    i. Download the cross-compiler.

    ```
    wget https://releases.linaro.org/archive/14.07/components/
    toolchain/binaries/gcc-linaro-aarch64-linux-
    gnu-4.9-2014.07_linux.tar.xz
    ```

    ii. Extract the cross-compiler.

    ```
    tar -xf gcc-linaro-aarch64-linux-gnu-4.9-2014.07_linux.tar.xz
    ```

iii. Set up the environment for cross-compilation by running the following command.

```
export PATH=$PATH:`pwd`/gcc-linaro-aarch64-linux-
gnu-4.9-2014.07_linux/bin
```

b. To build the UVC gadget tool, do the following.

i. Clone the `uvc-gadget` repository.

```
git clone https://github.com/wlhe/uvc-gadget.git
cd uvc-gadget
```

ii. Modify `Makefile` for a static build.

```
git diff
diff --git a/Makefile b/Makefile
index ccf5a34..5be54cc 100644
--- a/Makefile
+++ b/Makefile
@@ -2,7 +2,7 @@ CROSS_COMPILE   ?=
 ARCH            ?= x86
 KERNEL_DIR      ?= /usr/src/linux

-CC              := $(CROSS_COMPILE)gcc
+CC              := $(CROSS_COMPILE)gcc -static
 KERNEL_INCLUDE := -I$(KERNEL_DIR)/include -I$(KERNEL_DIR)/arch/$
(ARCH)/include
 CFLAGS          := -W -Wall -g $(KERNEL_INCLUDE)
 LDFLAGS                 := -g
```

iii. Cross-compile for `arm64` to generate the `uvc-gadget` executable.

```
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu-
```

- **UVC viewer** (host side): To receive the UVC data, open any USB webcam application on the computer.

a. To validate, push the UVC gadget application and sample image files to the device, do the following:

i. Open the SSH shell in permissive mode or use the ADB shell. For more information about how to run SSH, see the Use SSH section.

ii. Mount the file system.

```
mount -o remount,rw /usr
```

iii. Transfer files using SCP or similar tools. For more information about how to run SSH, see the Use SSH section.

For example, `scp uvc-gadget root@10.92.175.138:/usr/bin`

iv. Assign permission to execute.

```
chmod 0777 /usr/bin/uvc-gadget
```

b. Change to any UVC composition (90DF or 90CB). For example,

```
qusb setpid 90CB
```

USB is enumerated in UVC composition only when a user space video application is open. For the USB enumeration to occur, start the `uvc_gadget` application through a console (serial console or SSH shell or use the ADB shell).

- YUYV
  a. Push the `image-720.yuv` image file to `/root` or `/etc`.

     ```
     scp image-720.yuv root@10.92.175.138:/root
     ```

  b. Verify the `image-720.yuv` image with the `uvc-gadget` tool.

     ```
     uvc-gadget -u /dev/videoX -i /root/image-720.yuv -s 2 -m 2 -n 32  -t 10
     -f 0
     ```

     - Where `X` indicates the new video node created after the composition switch
     - Where `-f` is format with the following values:
       - `0` = V4L2_PIX_FMT_YUYV
       - `1` = V4L2_PIX_FMT_MJPEG

- MJPEG
  a. Push the `image-720.jpg` image file to `/root` or `/etc`.

     ```
     scp image-720.jpg root@10.92.175.138:/root
     ```

  b. Verify the `image-720.jpg` image with the `uvc-gadget` tool.

     ```
     uvc-gadget -u /dev/videoX -i /root/image-720.jpg -s 2 -m 2 -n 32  -t 10
     -f 1
     ```

     - Where `X` indicates the new video node created after the composition switch
     - Where `-f` is <format> with the following values:
       - `0` = V4L2_PIX_FMT_YUYV
       - `1` = V4L2_PIX_FMT_MJPEG

- For information about usage, run the following command.

  ```
  ./uvc-gadget -h
  ```

  The following output is displayed.

```
Usage: uvc-gadget [options]
Available options are
-b             Use bulk mode
-d             Do not use any real V4L2 capture device
-f <format>    Select frame format
        0 = V4L2_PIX_FMT_YUYV
        1 = V4L2_PIX_FMT_MJPEG
-h             Print this help screen and exit
-i image       MJPEG image
-m             Streaming mult for ISOC (b/w 0 and 2)
-n             Number of Video buffers (b/w 2 and 32)
-o <IO method> Select UVC IO method:
        0 = MMAP
        1 = USER_PTR
-r <resolution> Select frame resolution:
```

```
        0 = 360p, VGA (640x360)
        1 = 720p, WXGA (1280x720)
-s <speed>     Select USB bus speed (b/w 0 and 2)
        0 = Full Speed (FS)
        1 = High Speed (HS)
        2 = Super Speed (SS)
-t             Streaming burst (b/w 0 and 15)
-u device      UVC Video Output device
-v device      V4L2 Video Capture device
```

UAC use cases

USB audio uses isochronous, interrupt, and control transfers. All audio data is transferred over isochronous transfers. The interrupt transfers are used to relay information regarding the availability of audio clocks and control transfers are used to set volume, request sample rates.

NOTE QCS9075 and QCS8275 don't support the UAC use case.

To verify, select a USB composition with a UAC function.

```
qusb setpid 90CA
```

To capture or play back, use `tinyutils` applications such as `tinyplay/tinycap` available in the `rootfs`. The application opens the PCM nodes created in `/dev/snd/` when the UAC driver binds are successful.

The following are a few sample commands.

- List the sound card on the Linux host machine.

  ```
  arecord -L

  aplay -f S16_LE -r 44100 -c 2 -D front:CARD=qcs6490rb3gen2v,DEV=0
  441k_16bit_5min_m1dB.wav
  ```

- Play back the audio file.
  - To play the audio from device to host, push the `file.wav` file with the matching audio configuration. The following is an example of a sample configuration.
    - `cat /sys/kernel/config/usb_gadget/adb/functions/uac2.0/p_chmask`

      Output:

      ```
      3
      ```

    - `cat /sys/kernel/config/usb_gadget/adb/functions/uac2.0/c_chmask`

      Output:

      ```
      3
      ```

    - `cat /sys/kernel/config/usb_gadget/adb/functions/uac2.0/p_srate`

      Output:

      ```
      48000
      ```

    - `cat /sys/kernel/config/usb_gadget/adb/functions/uac2.0/c_srate`

      Output:

```
64000
```

- `cat /sys/kernel/config/usb_gadget/adb/functions/uac2.0/p_ssize`

  Output:

  ```
  2
  ```

- `cat /sys/kernel/config/usb_gadget/adb/functions/uac2.0/c_ssize`

  Output:

  ```
  2
  ```

  NOTE To modify the audio configuration, adjust the preceding parameters before establishing the USB composition to 90CA.

- Run the following command on the SSH shell or use the ADB shell.

  `tinyplay`

  The following output is displayed.

  ```
  Usage: tinyplay file.wav [-D card] [-d device] [-p period_size] [-n
  n_periods]
  ```

- Run the following command on a Linux host machine with an application such as `arecord` to capture the audio.

  ```
  arecord -f S16_LE -r 44100 -c 2 -D front:CARD=qcs6490rb3gen2v,DEV=0
  test1.wav
  ```

- Record an audio file.
  - Run the following command on a Linux host machine with an application such as `aplay` to play the audio.

    ```
    aplay -f S16_LE -r 44100 -c 2 -D front:CARD=qcs6490rb3gen2v,DEV=0
    441k_16bit_5min_m1dB.wav
    ```

  - Run the following command on the SSH shell of the device or use the ADB shell.

    `tinycap`

    The following output is displayed.

    ```
    Usage: tinycap file.wav [-D card] [-d device] [-c channels] [-r
    rate] [-b bits] [-a     bits_packed] [-p period_size] [-n n_periods]
    [-T capture time]
    ```

Data role swapping in USB power delivery

Data role swap (DR_SWAP) is the exchange of DFP (host) and UFP (device) roles between port partners over a USB Type-C connector. Power role swap (PR_SWAP) exchanges the source and sink roles between the port partners.

NOTE QCS9075 and QCS8275 don't support data and power role swapping over Type-C connectors.

- Data role
  - To swap data roles from host to device, run the following commands:

    ```
    cat /sys/class/typec/port0/data_role
    ```

    The following output is displayed.

    ```
    [host] device
    ```

    ```
    echo device > /sys/class/typec/port0/data_role
    cat /sys/class/typec/port0/data_role
    ```

    The following output is displayed.

    ```
    host [device]
    ```

  - To swap data roles from device to host, run the following commands:

    ```
    cat /sys/class/typec/port0/data_role
    ```

    The following output is displayed.

    ```
    host [device]
    ```

    ```
    echo host > /sys/class/typec/port0/data_role
    cat /sys/class/typec/port0/data_role
    ```

    The following output is displayed.

    ```
    [host] device
    ```

- Power role
  - To swap power roles from sink to source, run the following commands:

    ```
    cat /sys/class/typec/port0/power_role
    ```

    The following output is displayed.

    ```
    source [sink]
    ```

    ```
    echo source > /sys/class/typec/port0/power_role
    cat /sys/class/typec/port0/power_role
    ```

    The following output is displayed.

    ```
    [source] sink
    ```

  - To swap power role from source to sink, run the following commands:

    ```
    cat /sys/class/typec/port0/power_role
    ```

    The following output is displayed.

    ```
    [source] sink
    ```

    ```
    echo sink > /sys/class/typec/port0/power_role
    cat /sys/class/typec/port0/power_role
    ```

    The following output is displayed.

```
source [sink]
```

Customize with configfs

The `configfs` file system provides the converse of the `sysfs` functionality. A number of interfaces can configure Linux USB gadgets, with each interface representing a USB function.

The `qusb` executable configures the `configfs` USB gadget using the following commands.

1. Create `configfs` and mount `functionfs`.

   ```
   qusb init
   ```

2. Start `adbd/diag` services.
3. Bind `configfs` with the USB gadget application.

   ```
   qusb bind
   ```

4. Stop or unbind the USB gadget application.

   ```
   qusb unbind
   ```

5. Set diag and ADB composition.

   ```
   qusb setpid 901D
   ```

6. List the available USB compositions.

   ```
   qusb showpid
   ```

   The USB compositions are as follows.

   - A4A1 NCM
   - 4EE7 ADB
   - 900E DIAG
   - 901C DIAG + UAC2
   - 901D DIAG + ADB
   - 9015 MASS_STORAGE + ADB
   - 9024 RNDIS + ADB
   - 902A RNDIS + MASS_STORAGE
   - 902B RNDIS + ADB + MASS_STORAGE
   - 902C RNDIS + DIAG
   - 902D RNDIS + DIAG + ADB
   - 902F RNDIS + DIAG + MASS_STORAGE
   - 9060 DIAG + QDSS + ADB
   - 908C NCM + ADB
   - 90CA DIAG + UAC2 + ADB
   - 90CB DIAG + UVC + ADB
   - 90CC DIAG + UAC2 + UVC + ADB
   - 90DF DIAG + UVC

- 90E0 DIAG + UAC2 + UVC
- F000 MASS_STORAGE
- F00E RNDIS

8.9 Verify USB device

The following table lists the various methods to verify the USB device and host modes.

Table 8-13 USB device and host mode verification

USB mode	Feature	Description
USB device mode	ADB	By default, USB is enumerated in the ADB-only composition.
	Diag	See Example shell script for a USB composition with diag and ADB interfaces and Change USB composition through QUSB service.
	Mass storage	Change the USB composition (see Change USB composition through QUSB service) and select the mass_storage composition.
	Composition switch	See Example shell script for a USB composition with diag and ADB interfaces and Change USB composition through QUSB service.
	USB LPM with cable connects and disconnects	Disconnect USB manually so that the `dwc3-qcom` module changes to low-power mode.
	Network control modem (NCM)	Change the USB composition (Change USB composition through QUSB service) and select the NCM composition.
USB host mode	Host human interface device (HID) class	Switch to host mode by connecting the HID class device directly to the device under test.
	Host mass storage (MS) class	Switch to host mode by connecting the MS class device directly to the device under test.
	Host hub class	Switch to host mode by connecting the hub class device directly to the device under test.
	Host power management	Peripherals support the host-mode suspend (headsets) state when the `dwc3-qcom` module changes to low-power mode in host mode.
	USB LPM with peripheral connect and disconnect	Peripherals support the host mode suspend (headsets) when the `dwc3-qcom` module changes to low-power mode in host mode.
	USB L1 LPM with high speed	High-speed peripherals support host mode suspend (headsets) when the `dwc3-qcom` module changes to low-power mode in host mode.
	USB camera	See Configure USB camera.

8.10 Debug USB issues

This section provides information on the various methods to obtain debugging logs. The debugging methods include `regdumps`, debug `ftraces`, `configfs` nodes. The logs provide visibility into the event and controller state details when debugging issues with low-power mode entry-exit, SMMU faults, unclocked accesses.

NOTE QCS9075 and QCS8275 don't support USB Type-C feature.

Trace USB

The `debugfs` tracing provides a deeper view into each transaction over the USB line. To view the list of traces, run the following command.

```
ls /sys/kernel/debug/tracing/events/dwc3
```

NOTE Ensure that `debugfs` is mounted. If not mounted, run the following command to mount `debugfs`.

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

Following are the traces available for verifying data transfers in the xHCI/gadget stack/USB Type-C connector system software interface (UCSI).

```
dwc3_alloc_request  dwc3_event              dwc3_gadget_generic_cmd  enable
dwc3_complete_trb   dwc3_free_request       dwc3_gadget_giveback     filter
dwc3_ctrl_req       dwc3_gadget_ep_cmd      dwc3_prepare_trb
dwc3_ep_dequeue     dwc3_gadget_ep_disable  dwc3_readl
dwc3_ep_queue       dwc3_gadget_ep_enable   dwc3_writel
```

To list the traces in xHCI/host controller driver (HCD), run the following command.

```
ls /sys/kernel/debug/tracing/events/xhci-hcd
```

Following are the traces available for verifying data transfers in the xHCI/HCD.

```
enable                           xhci_handle_cmd_config_ep
filter                           xhci_handle_cmd_disable_slot
  xhci_add_endpoint                xhci_handle_cmd_reset_dev
  xhci_address_ctrl_ctx            xhci_handle_cmd_reset_ep
  xhci_address_ctx                 xhci_handle_cmd_set_deq
  xhci_alloc_dev                   xhci_handle_cmd_set_deq_ep
  xhci_alloc_virt_device           xhci_handle_cmd_stop_ep
  xhci_configure_endpoint          xhci_handle_command
  xhci_configure_endpoint_ctrl_ctx  xhci_handle_event
  xhci_dbc_alloc_request           xhci_handle_port_status
  xhci_dbc_free_request            xhci_handle_transfer
  xhci_dbc_gadget_ep_queue         xhci_hub_status_data
  xhci_dbc_giveback_request        xhci_inc_deq
  xhci_dbc_handle_event            xhci_inc_enq
  xhci_dbc_handle_transfer         xhci_queue_trb
  xhci_dbc_queue_request           xhci_ring_alloc
  xhci_dbg_address                 xhci_ring_ep_doorbell
  xhci_dbg_cancel_urb              xhci_ring_expansion
  xhci_dbg_context_change          xhci_ring_free
  xhci_dbg_init                    xhci_ring_host_doorbell
  xhci_dbg_quirks                  xhci_setup_addressable_virt_device
  xhci_dbg_reset_ep                xhci_setup_device
```

```
  xhci_dbg_ring_expansion           xhci_setup_device_slot
  xhci_discover_or_reset_device     xhci_stop_device
  xhci_free_dev                    xhci_urb_dequeue
  xhci_free_virt_device             xhci_urb_enqueue
  xhci_get_port_status              xhci_urb_giveback
  xhci_handle_cmd_addr_dev
```

To list the available events of the USB video class (UVC) gadget driver, run the following command.

```
ls /sys/kernel/debug/tracing/events/gadget
```

The following output is displayed.

```
enable                        usb_gadget_activate
  filter                        usb_gadget_clear_selfpowered
  usb_ep_alloc_request         usb_gadget_connect
  usb_ep_clear_halt            usb_gadget_deactivate
  usb_ep_dequeue               usb_gadget_disconnect
  usb_ep_disable               usb_gadget_frame_number
  usb_ep_enable                usb_gadget_giveback_request
  usb_ep_fifo_flush            usb_gadget_set_remote_wakeup
  usb_ep_fifo_status           usb_gadget_set_selfpowered
  usb_ep_free_request          usb_gadget_vbus_connect
  usb_ep_queue                 usb_gadget_vbus_disconnect
  usb_ep_set_halt              usb_gadget_vbus_draw
  usb_ep_set_maxpacket_limit   usb_gadget_wakeup
  usb_ep_set_wedge
```

To list the available events in the UCSI driver, run the following command.

```
ls /sys/kernel/debug/tracing/events/ucsi
```

The following output is displayed.

```
enable  ucsi_connector_change  ucsi_register_port  ucsi_run_command
filter  ucsi_register_altmode  ucsi_reset_ppm
```

USB regdump

The USB `debugfs` provides the following information.

- Mode of operation.

  ```
  cat /sys/kernel/debug/usb/a600000.usb/mode
  ```

  Sample output:

  ```
  device
  ```

- State and transfer ring buffer (TRB) queues to all endpoints in device mode.
- Current link state.

  ```
  cat /sys/kernel/debug/usb/a600000.usb/link_state
  ```

  Sample output.

  ```
  Sleep
  ```

- List processor (LSP) dump.

  ```
  cat /sys/kernel/debug/usb/a600000.usb/lsp_dump
  ```

  Sample output:

```
GDBGLSP[0] = 0x40000000
GDBGLSP[1] = 0x00003a80
GDBGLSP[2] = 0x38200000
GDBGLSP[3] = 0x00802000
GDBGLSP[4] = 0x126f1000
GDBGLSP[5] = 0x3a800018
GDBGLSP[6] = 0x00000a80
GDBGLSP[7] = 0xfc03f14a
GDBGLSP[8] = 0x0b803fff
GDBGLSP[9] = 0x00000000
GDBGLSP[10] = 0x000000f8
GDBGLSP[11] = 0x000000f8
GDBGLSP[12] = 0x000000f8
GDBGLSP[13] = 0x000000f8
GDBGLSP[14] = 0x000000f8
GDBGLSP[15] = 0x000000f8
```

```
ls /sys/kernel/debug/usb/a600000.usb
```

Sample output:

```
ep0in    ep11out  ep14in   ep1out  ep4in   ep6out  ep9in       regdump
ep0out   ep12in   ep14out  ep2in   ep4out  ep7in   ep9out      testmode
ep10in   ep12out  ep15in   ep2out  ep5in   ep7out  link_state
ep10out  ep13in   ep15out  ep3in   ep5out  ep8in   lsp_dump
ep11in   ep13out  ep1in    ep3out  ep6in   ep8out  mode
```

The `regdump` command provides the current state of the register space for the following registers:

- Device mode registers, such as DCTL, DSTS, and DCFG
- Global registers, such as GCTL and GSTS

```
cd /sys/kernel/debug/usb/a600000.usb
cat regdump
```

Sample output:

```
GSBUSCFG0 = 0x2222000e
GSBUSCFG1 = 0x00001700
GTXTHRCFG = 0x00000000
GRXTHRCFG = 0x00000000
GCTL = 0x00102000
GEVTEN = 0x00000000
GSTS = 0x7e800000
GUCTL1 = 0x810c1802
GSNPSID = 0x5533330a
GGPIO = 0x00000000
GUID = 0x00060500
GUCTL = 0x0d00c010
GBUSERRADDR0 = 0x00000000
GBUSERRADDR1 = 0x00000000
GPRTBIMAP0 = 0x00000000
GPRTBIMAP1 = 0x00000000
GHWPARAMS0 = 0x4020400a
GDBGFIFOSPACE = 0x00420000
GDBGLTSSM = 0x41090658
GDBGBMU = 0x20300000
GPRTBIMAP_HS0 = 0x00000000
GPRTBIMAP_HS1 = 0x00000000
```

```
GPRTBIMAP_FS0 = 0x00000000
GPRTBIMAP_FS1 = 0x00000000
GUCTL2 = 0x0198440d
VER_NUMBER = 0x00000000
VER_TYPE = 0x00000000
GUSB2PHYCFG(0) = 0x00002400
GUSB2I2CCTL(0) = 0x00000000
GUSB2PHYACC(0) = 0x00000000
GUSB3PIPECTL(0) = 0x030e0002
GTXFIFOSIZ(0) = 0x00000042
GRXFIFOSIZ(0) = 0x00000305
GEVNTADRLO(0) = 0xfffff000
GEVNTADRHI(0) = 0x0000000f
GEVNTSIZ(0) = 0x00001000
GEVNTCOUNT(0) = 0x00000000
GHWPARAMS8 = 0x000007ea
GUCTL3 = 0x00010000
GFLADJ = 0x8c80c8a0
DCFG = 0x00cc08b4
DCTL = 0x8cf00a00
DEVTEN = 0x00000257
DSTS = 0x008a5200
DGCMDPAR = 0x00000000
DGCMD = 0x00000000
DALEPENA = 0x0000000f
DEPCMDPAR2(0) = 0x00000000
DEPCMDPAR1(0) = 0xffffe000
DEPCMDPAR0(0) = 0x0000000f
DEPCMD(0) = 0x00000006
OCFG = 0x00000000
OCTL = 0x00000000
OEVT = 0x00000000
OEVTEN = 0x00000000
OSTS = 0x00000000
```

View file system attributes for host mode using sysfs

To view the bus details, run the following command.

```
lsusb
```

Sample output:

```
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 03f0:134a HP, Inc Optical Mouse
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

To list the contents of the current directory, run the following command.

```
cd /sys/bus/usb/devices/
ls
```

Sample output:

```
1-0:1.0  1-1  1-1:1.0  2-0:1.0  usb1  usb2
```

To view details about the USB devices, run the following command.

```
cat /sys/kernel/debug/usb/devices
```

Sample output:

```
T:  Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=480  MxCh= 1
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 2.00 Cls=09(hub  ) Sub=00 Prot=01 MxPS=64 #Cfgs=  1
P:  Vendor=1d6b ProdID=0002 Rev= 6.05
S:  Manufacturer=Linux 6.5.0-rc4 xhci-hcd
S:  Product=xHCI Host Controller
S:  SerialNumber=xhci-hcd.0.auto
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=256ms

T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=1.5  MxCh= 0
D:  Ver= 2.00 Cls=00(>ifc ) Sub=00 Prot=00 MxPS= 8 #Cfgs=  1
P:  Vendor=03f0 ProdID=134a Rev= 1.00
S:  Manufacturer=PixArt
S:  Product=HP USB Optical Mouse
C:* #Ifs= 1 Cfg#= 1 Atr=a0 MxPwr=100mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=03(HID  ) Sub=01 Prot=02 Driver=usbhid
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=10ms

T:  Bus=02 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#=  1 Spd=5000 MxCh= 1
B:  Alloc=  0/800 us ( 0%), #Int=  0, #Iso=  0
D:  Ver= 3.00 Cls=09(hub  ) Sub=00 Prot=03 MxPS= 9 #Cfgs=  1
P:  Vendor=1d6b ProdID=0003 Rev= 6.05
S:  Manufacturer=Linux 6.5.0-rc4 xhci-hcd
S:  Product=xHCI Host Controller
S:  SerialNumber=xhci-hcd.0.auto
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr=  0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub  ) Sub=00 Prot=00 Driver=hub
E:  Ad=81(I) Atr=03(Int.) MxPS=   4 Ivl=256ms
```

8.11 USB examples

For information about the upstream device tree reference, see https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi.

For information about device-tree node for the Qualcomm Linux hardware SoCs, see https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts.

For information about RPM changes in the USB driver and other examples, see https://patchwork.kernel.org/project/linux-usb/list/?series=793939&archive=both.

For information about how to flatten a device tree, see https://lore.kernel.org/all/af60c05b-4a0f-51b8-486a-1fc601602515@quicinc.com/ and https://lore.kernel.org/all/20231016-dwc3-refactor-v1-0-ab4a84165470@quicinc.com/.

9 CAN

Controller area network (CAN) is a message-based serial protocol. The terms CAN and CAN bus are used in the same context. The advantages of CAN are as follows:

- Uses CSMA/CD method and bitwise contention to sense the bus at the bit-level and transmit the message.
- Each message is assigned a priority. Messages with the highest priority always control the bus arbitration. Since arbitration is based on bits, there are dominant and recessive bit levels.
- Any node can transmit at any time, no primary-secondary nodes.
- All the data is transmitted through two (differential pair) lines: CAN_H and CAN_L line.
- CAN nodes are synchronized using bit-synchronization where each bit time is divided in to four segments of time quanta.
- Facilitates communication between multiple nodes through a message-based protocol, eliminating the need for extensive wiring.
- Ensures data security with error management including detection, signaling, and correction.

NOTE
- CAN is enabled by default in QCS6490 and QCS5430.
- QCS9075 and QCS8275 don't support CAN.

9.1 CAN features

The CAN protocol supports the following features.

- Multicontroller priority-based bus access
- Nondestructive content-based arbitration
- Broadcast through all-frame transfer
- Multicast frame transfer by acceptance filtering
- Remote data request
- Configuration flexibility
- Network-wide data consistency
- Error detection and error signaling

- Automatically retransmits frames that lose arbitration, lack acknowledgment, or encounter errors during transmission
- Identifies differences between temporary node errors and permanent node failures, and the automatic deactivation of faulty nodes

9.2 CAN architecture

The MCP2517 is an external configurable CAN controller, which can be accessed through the SPI line. The SPI interface supports up to 20 MHz with support for Mode0,1. The controller is connected to a QUP block with SE3 configured by a 4-pin SPI operation. The SPI controller is connected to a transceiver (MCP2561xxx) which connects to the CAN bus. Power input to the SPI controller and the CAN transceiver is through the SoC.

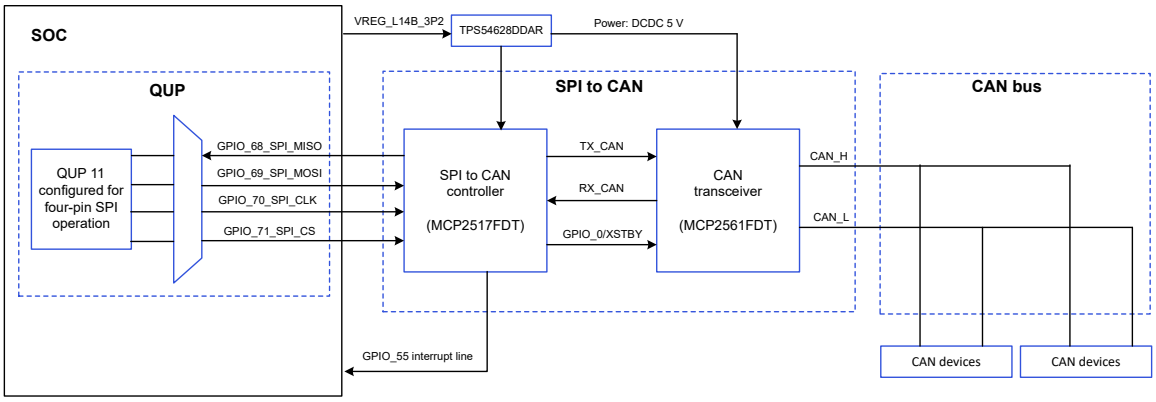


Figure 9-1 CAN architecture

The user application communicates with the network interface provided by the SocketCAN. The software driver is a client of the SPI core module in Linux, to which the QUP_SPI driver is registered with. The actual hardware is connected through the SPI pins.

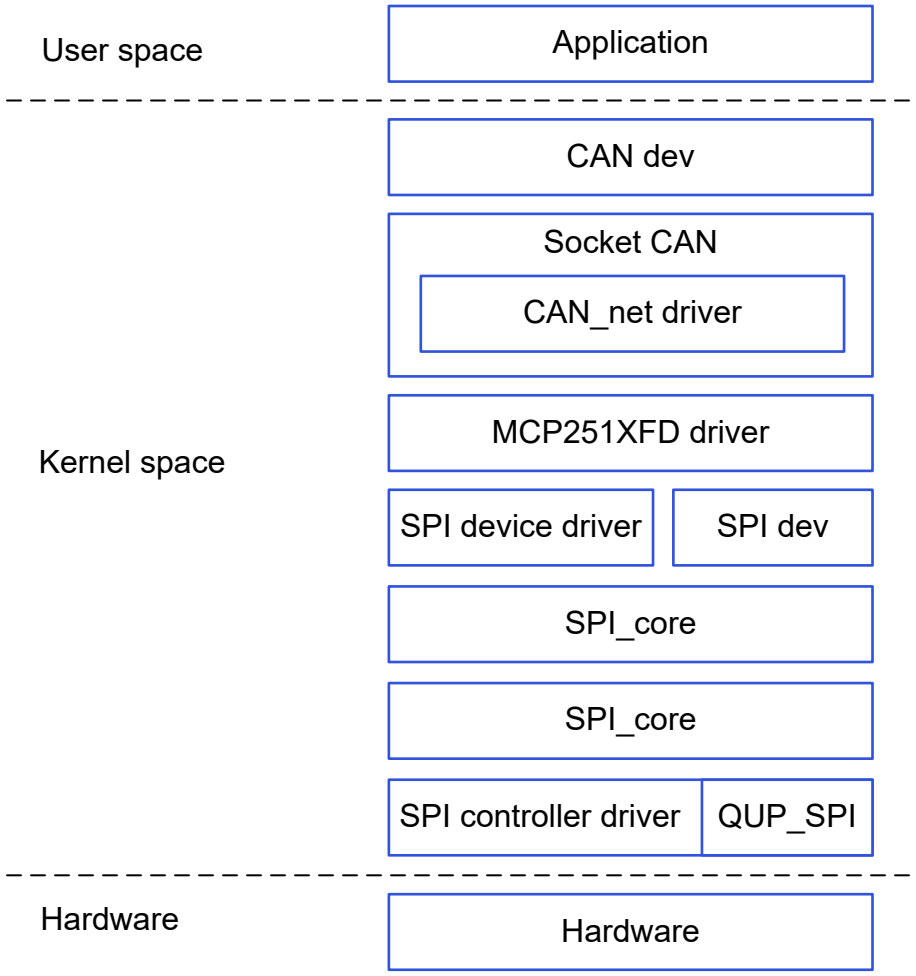


Figure 9-2 CAN software stack

9.3 CAN APIs

For CAN public APIs, see https://www.kernel.org/doc/Documentation/networking/can.txt.

9.4 CAN samples and tools

The SocketCAN user space utilities and tools to display, record, generate, and replay the CAN traffic are listed in the following table.

Table 9-1 SocketCAN utilities

Utility	Description
`candump`	To display, filter and log CAN data to files.
`canplayer`	To replay CAN log files.
`cansend`	To send a single frame.
`cangen`	To generate (random) CAN traffic.
`cansniffer`	To display CAN data content differences (just 11-bit CAN IDs).

To download CAN utilities, see https://github.com/linux-can/can-utils.

For information about CAN driver files, see `kernel/drivers/net/can/spi/mcp251xfd/mcp251xfd-core.c`.

CAN analyzer - PCAN-USB FD

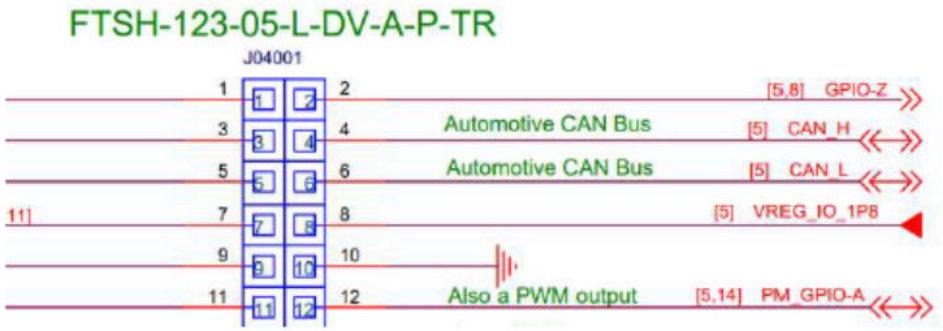
The PCAN-USB FD is a versatile CAN packet/message sniffer that can operate as a node on any CAN bus, supporting flexible data rates. It can be used to sniff and send CAN packets on a host machine over USB. The key features are as follows:

- PCAN-View GUI tool
  - Provides easy ways to sniff and send CAN packets as per-user needs.
  - Monitors and debugs CAN packets during development and testing.
- PCAN-USB FD interface
  - Enables a simple connection to CAN FD and CAN networks.
  - Compact plastic casing suitable for mobile applications.
  - Galvanic isolation of up to 500 V decouples the PC from the CAN bus.
- CAN FD standard
  - Characterized by higher bandwidth for data transfer.
  - Allows transmission of up to 64 data bytes per CAN FD frame (instead of 8).
  - Supports bit rates up to 12 Mbit/s.
  - Downward compatible with the CAN 2.0 standard.
  - CAN FD nodes can be inserted into existing CAN networks without CAN FD extensions.

Hardware setup

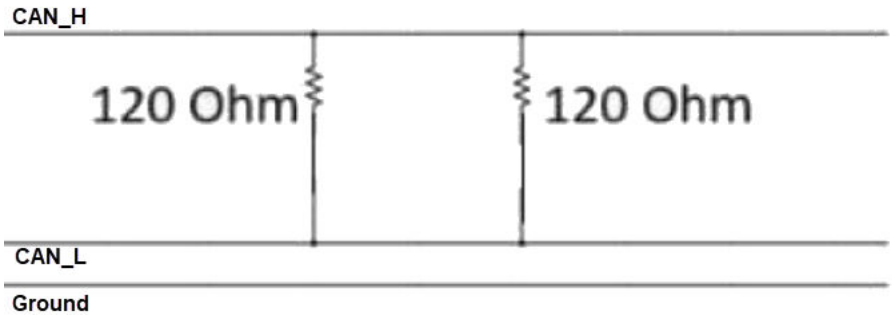
1. Connect to the DB9 connector.
   - The PCAN has a DB9 connector on which any standard DB9 connector can be plugged in.

2. Connect to the Qualcomm device.
   - Pull out the CAN_H, CAN_L, and GND lines.
   - Connect these lines to the two terminal-resistors of 120 Ω.
3. Connect CAN_H, CAN_L, and Ground.



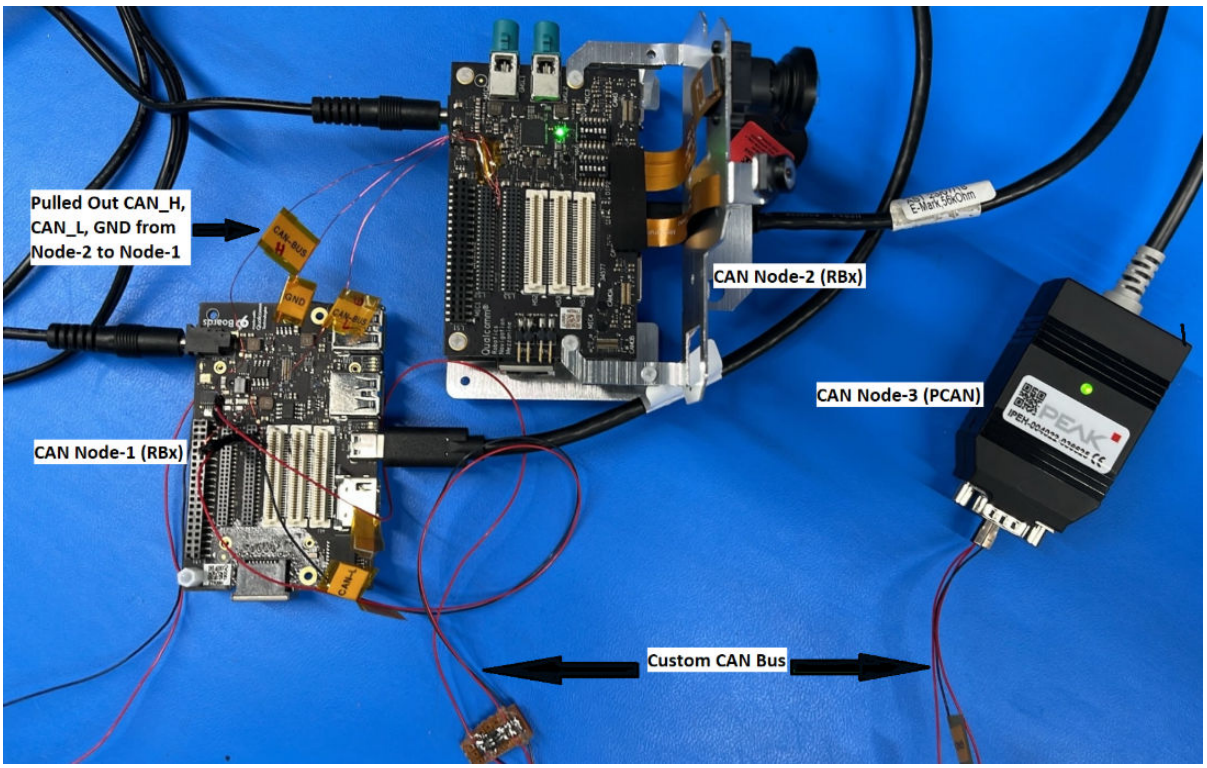
4. Connect an external terminating resistor.

   NOTE The device doesn't have an internal terminating resistor of 120 Ω. This is crucial because some tools provide built-in support for the terminating resistor.



The following figure shows the connected hardware.

Figure 9-3 CAN USB FD, external resistor, and device connection



9.5 Bringup CAN interface

After the CAN driver is built and loaded into the kernel, the following sequence of commands sends data to identify if the CAN node is up. A CAN node can be started or stopped using the commands listed in the following table.

NOTE All the network devices use `ifconfig -a`.

Table 9-2 CAN commands

Instruction	Command
Check for the can node number	`ifconfig -a`
Set a bit rate of 125 kbps	`ip link set can0 up type can bitrate 125000`
Start CAN	`ip link set can0 up`
Stop CAN	`ip link set can0 down`
Check the CAN configuration	`ip -details link show can0`
CAN internal loopback	`ip link set can0 up type can bitrate 500000 loopback on`

Table 9-2 CAN commands (cont.)

Instruction	Command
CAN FD internal loopback	`ip link set can0 up type can bitrate 1000000 dbitrate 5000000 fd on loopback on`
Send CAN FD frame	`cansend can0 213##311223344`

To start any CAN node, the bit rate for the bus must be defined, or configured if the bus-rate is already available. This configuration ensures that the CAN node is up and active. For more details about CAN utilities, see https://manpages.debian.org/testing/can-utils/index.html.

Send and receive data

Some utilities are available in the `CAN-Utils` file. SocketCAN enables CAN nodes for sending and receiving data. SocketCAN APIs are also used for custom applications.

- Send data with 11-bit ID

  ```
  cansend can0 7AF#11.22.33.44.55.66.aa.ff
  ```

- Send data with 29-bit ID

  ```
  cansend can0 111FFFFF#aa.00.cc.aa.33.61.aa.ab
  ```

- View the current incoming data on can0 node

  ```
  candump can0
  ```

9.6 Configure CAN interface

SocketCAN is an implementation of the CAN protocol for Linux. SocketCAN uses the Berkeley socket API, the Linux network stack and implements the CAN device drivers as network interfaces. The CAN socket API has been designed as similar as possible to the TCP/IP protocols to allow users, familiar with network programming, easily learn how to use CAN sockets.

Initialize hardware

The driver initializes and configures the underlying hardware block. The initialization occurs after the `driver's _probe()` function is called.

```
static struct spi_driver mcp251xfd_driver = {
    .driver = {
        .name = DEVICE_NAME,
        .pm = &mcp251xfd_pm_ops,
        .of_match_table = mcp251xfd_of_match,
    },
    .probe = mcp251xfd_probe,
    .remove = mcp251xfd_remove,
    .id_table = mcp251xfd_id_table,
};
```

```
static int mcp251xfd_probe(struct spi_device *spi)
{
    struct net_device *ndev;
    struct mcp251xfd_priv *priv;
[..]
ndev = alloc_candev(sizeof(struct mcp251xfd_priv),
                MCP251XFD_TX_OBJ_NUM_MAX);
    if (!ndev)
        return -ENOMEM;

    SET_NETDEV_DEV(ndev, &spi->dev); // The SPI node is set as the parent
node to the network device.

    ndev->netdev_ops = &mcp251xfd_netdev_ops;
[..]
    err = mcp251xfd_register(priv);
};
```

9.7 Debug CAN issues

CAN issues can be debugged using the `ftrace` tool. The MCP2517 driver supports the `dev_coredump()` API.

9.8 CAN examples

For CAN examples, see https://github.com/linux-can/can-utils/blob/master/README.md.

10 References

10.1 QUP v3 overview

The QUP v3 is a highly flexible and programmable hardware for supporting a wide range of serial interface. A single QUP v3 serial engine hardware core provides up to eight serial interfaces. The two QUP v3 hardware cores are as follows:

- 16-serial engine core
- SSC QUP v3 hardware core with five serial engines available in the SSC_I/Os

The QUP v3 supports access from multiple hardware entities in the system. Each entity has its own execution environment (EE), a separate address space, and an interrupt line. For information about the various transfer modes that can be configured in QUP v3, see Supported transfer modes in QUP v3.

The following figure shows one GSI core/engine connected with up to eight serial engines (SE). You have the flexibility to customize configurations depending on the use case or the protocol of the serial engine. For information about how to customize configurations, see QUP v3 access control customization. To verify if the QUP v3 firmware is correctly flashed, see QUP v3 firmware status verification.

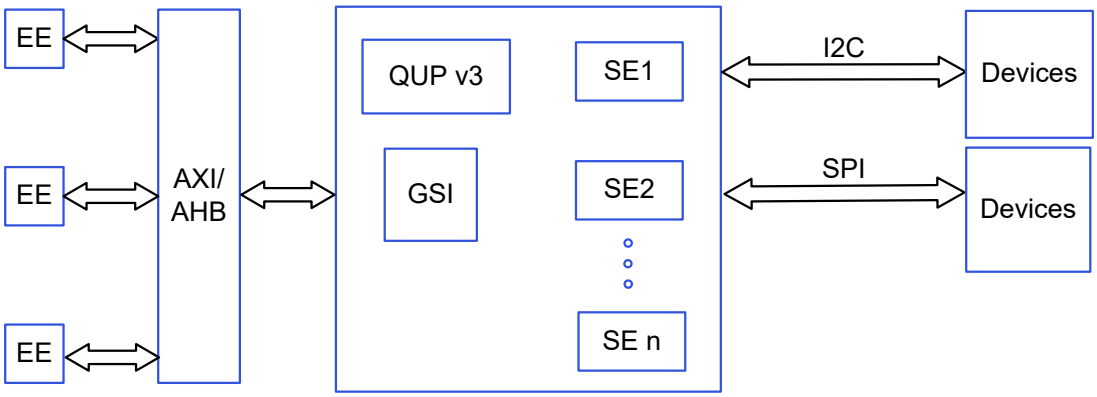
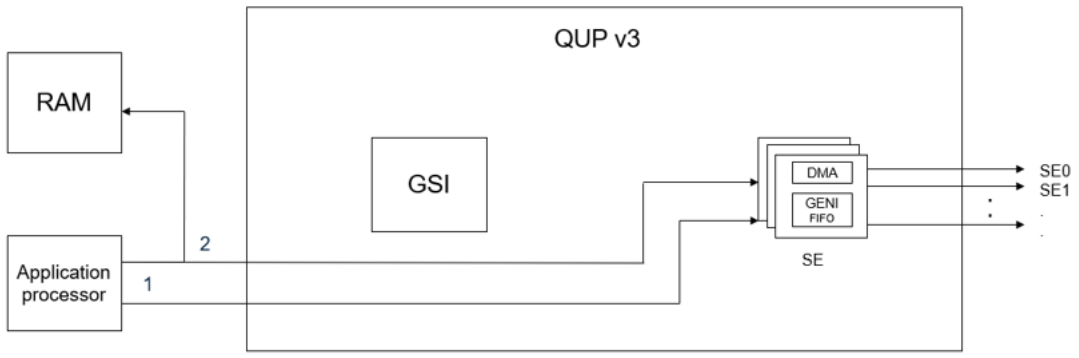


Figure 10-1 QUP v3 block diagram

10.2 Supported transfer modes in QUP v3

The following modes can be configured in the QUP v3 serial engine.

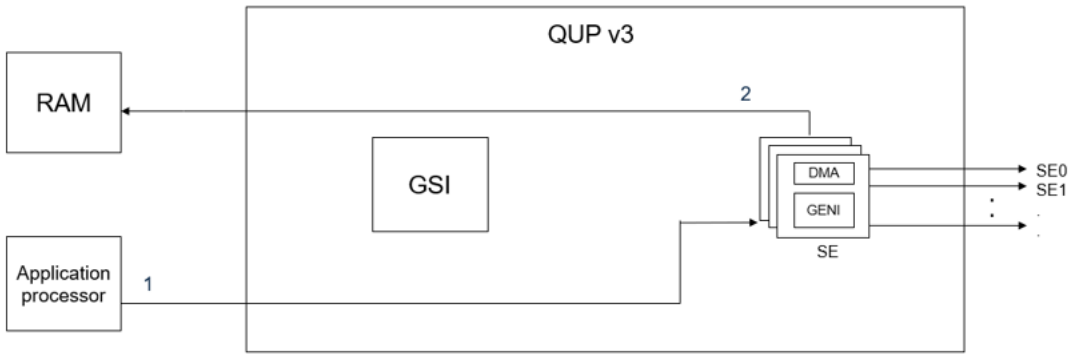
- **FIFO mode**



a. ①: The application processor configures the generic interface (GENI).

b. ②: The application processor processes Rx/Tx data. The data is transferred between GENI and memory.

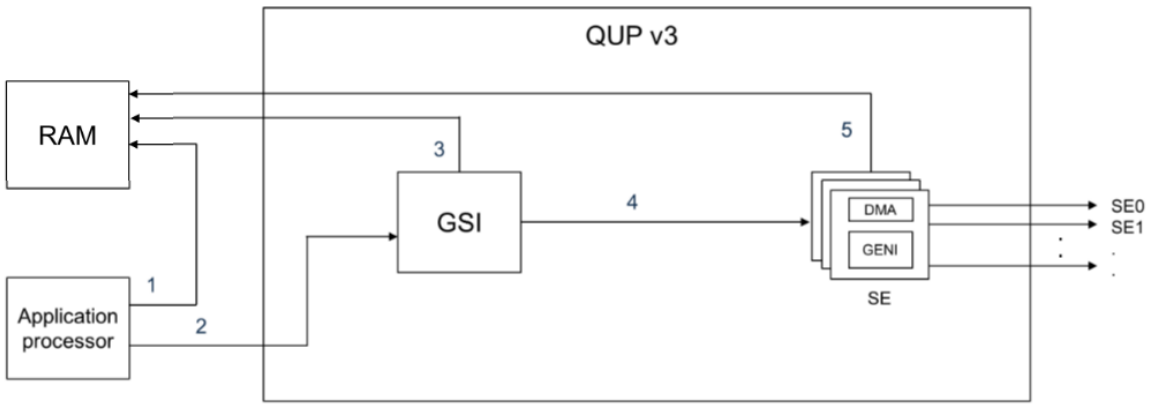
- **DMA mode**



a. ①: The application processor initializes DMA.

b. ②: The serial engine configures GENI, and initiates the transfer process.

- **GSI mode**



a. ①: The application processor prepares TRE.
b. ②: The application processor informs QUP (GSI).
c. ③: The GSI processes TRE.
d. ④: After the TRE completes processing, it's sent to GENI.
e. ⑤: The serial engine processes the Rx/Tx data.

NOTE The QUP v3 UART serial engine doesn't support the GSI mode.

10.3 QUP v3 access control customization

The QUP v3 user access file `QUPAC_Access.c` specifies the owners of the serial engine resource. Initially, it's populated according to the system I/O GPIO allocation. All serial engines must be listed to access the subsystem. It's flexible enough to list only the available serial engine on a particular device.

To customize the access control for the required serial engine protocol, configure the parameters in the `QUPAC_Access.c` file. The Qualcomm TEE image for the `QUPAC_Access.c` file is at `/firmware/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem/TZ.XF.5.0/trustzone_images/core/settings/buses/qup_accesscontrol/qupv3/config/<chipset>/QUPAC_Access.c`.

To specify the owner of the serial engine resource, modify the `QUPAC_Access.c` file to suit the board design.

The following use case specifies the default protocol that operates on an enabled serial engine. You can modify the code according to your board design.

Nonsecure mode use case in QUP v3 serial engine

The following nonsecure mode use cases are supported in the QUP v3 serial engine.

```
const QUPv3_se_security_permissions_type qupv3_perms_iot_rb3[] =
{
  /*   PeriphID,         ProtocolID,               Mode,  NsOwner,
bAllowFifo, bLoad, bModExcl  */
  { QUPV3_0_SE0, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // LT9611 and QPS615 I2C
  { QUPV3_0_SE1, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
```

```
TRUE,  TRUE,  FALSE }, // APPS I2C - PCIE/ USB Type C
  { QUPV3_0_SE2, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // SMB / LS1 I2C
  { QUPV3_0_SE3, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // CAN SPI
  { QUPV3_0_SE4, QUPV3_PROTOCOL_UART_4W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // LS1 UART
  { QUPV3_0_SE5, QUPV3_PROTOCOL_UART_2W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  FALSE, FALSE }, // Debug UART
  { QUPV3_0_SE6, QUPV3_PROTOCOL_UART_2W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // WLAN UART
  { QUPV3_0_SE7, QUPV3_PROTOCOL_UART_4W, QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // Hastings BT
  { QUPV3_1_SE0, QUPV3_PROTOCOL_SPMI,    QUPV3_MODE_FIFO, AC_ADSP_Q6_ELF,
TRUE,  TRUE,  FALSE }, // QuP SPMI
  { QUPV3_1_SE1, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // NFC I2C
  { QUPV3_1_SE2, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_FIFO, AC_HLOS,
TRUE,  TRUE,  FALSE }, // HDMI OUT for VIDEOIOBoard
  { QUPV3_1_SE3, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_FIFO, AC_HLOS,
FALSE, TRUE,  TRUE }, // LS1 SPI
  { QUPV3_1_SE4, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI,  AC_TZ,
FALSE, TRUE,  TRUE }, // SPI -NFC ESE
  { QUPV3_1_SE5, QUPV3_PROTOCOL_I2C,     QUPV3_MODE_GSI,  AC_HLOS,
FALSE, TRUE,  FALSE}, // Legacy Touch
  { QUPV3_1_SE6, QUPV3_PROTOCOL_SPI,     QUPV3_MODE_GSI,  AC_HLOS,
FALSE, TRUE,  FALSE}, // FP
  /*QUPV3_1_SE7*/
};
```

QUP v3 serial engine access list description

The following variables are passed into the `QUPv3_se_security_permissions_type` structure.

Table 10-1 Security permission variables for QUP v3

Variables	Description
`PeriphID`	Serial engine peripheral to configure and assign.
`ProtocolID`	Macro of the required protocol.
`Mode`	Macro of FIFO/GSI/DMA modes.
`NsOwner`	Holds a macro of the image that needs access.
`bAllowFifo`	The boolean flag is set to `True` if the mode is `FIFO`, else the flag is set to `False`.
`bLoad`	The boolean flag value is set to `True` to load the protocol firmware.
`bModExcl`	This flag is exclusively for Qualcomm TEE. It's set to `True` when `NsOwner` is AC_TZ.

For more information about this macro, see `settings/buses/qup_accesscontrol/qupv3/interface/QupACCommonIds.h`.

10.4 QUP v3 firmware status verification

For serial engines to work, the QUP firmware must be flashed correctly. The firmware is delivered through the metabuild at `common/core_qupv3fw/<chipset>/qupv3fw.elf`. You can verify the firmware status by checking `GENI_FW_REVISION_RO (0xa8c068)`. For example, identify the register in the kernel log for the `0000ffff` value. In the following log, the `0000ffff` error indicates that the firmware isn't flashed correctly.

```
0a8c068: 0000ffff //Invalid firmware or firmware not loaded
00a80068: 00000126 //SPI
00a88068: 00000338 //I2C
```

Modify the `QUPAC_Access.c` file configuration only if you intend to use a protocol different from the default configuration.

The following sample log is displayed when configurations don't match after loading.

```
msm_geni_serial 898000.qcom,qup_uart:msm_geni_serial_startup: Invalid FW
                    255 loaded
```

10.5 Related documents

Title	Resource
Qualcomm Technologies, Inc.	
Secure shell	https://docs.qualcomm.com/bundle/publicresource/topics/80-70018-254/how_to.html
QDTE	https://docs.qualcomm.com/bundle/publicresource/topics/80-70018-4/tools.html#qdte
Resources	
PCI bus subsystem	https://www.kernel.org/doc/html/latest/PCI/index.html
Linux user space examples	https://github.com/Digilent/linux-userspace-examples/tree/master/uart_example_linux/src
DTSI configuration examples	https://github.com/torvalds/linux/blob/master/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
Test tools and methods for the UART serial interface driver	https://docs.kernel.org/admin-guide/serial-console.html
UART Linux APIs	https://github.com/torvalds/linux/blob/master/include/linux/tty.h
UART upstream device tree reference	https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
Qualcomm Robotics RB3 Gen 2 Development Kit device tree node	https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
SPI samples	https://github.com/Digilent/linux-userspace-examples/tree/master
SPI Linux APIs	▪ https://github.com/torvalds/linux/blob/master/include/uapi/linux/spi/spidev.h ▪ https://github.com/torvalds/linux/blob/master/include/linux/spi/spi.h

Title	Resource
SPI kernel tools	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/tools/spi
SPI upstream device tree reference	https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
I2C samples	▪ https://manpages.debian.org/testing/i2c-tools/index.html ▪ https://linuxhint.com/i2c-linux-utilities/
I2C Linux APIs	▪ https://github.com/torvalds/linux/blob/master/include/linux/i2c.h ▪ https://github.com/torvalds/linux/blob/master/include/linux/i2c-dev.h
I2C upstream kernel test applications and I2C tool	https://layers.openembedded.org/layerindex/recipe/27859/
I2C upstream device tree reference	https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
PCIe device initialization and enumeration process	https://www.kernel.org/doc/html/latest/PCI/index.html
PCIe framework and client driver PCIe registrations	https://www.kernel.org/doc/html/latest/PCI/index.html
Add MSI groups supported for a PCIe instance	https://lore.kernel.org/linux-arm-msm/f1168212-bc6e-4570-869c-2870d6f248ad@linaro.org/T/
PCIe debugging	https://pcisig.com/specifications
PCIe upstream device tree reference	▪ https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi ▪ https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/pci/controller/dwc/pcie-qcom.c?h=v6.8-rc6#n1634
USB `Qscratch` wrapper driver	https://github.com/torvalds/linux/blob/master/drivers/usb/dwc3/dwc3-qcom.c
Controller core driver	https://github.com/torvalds/linux/blob/master/drivers/usb/dwc3/core.c
USB software interface drivers for Type-C connector	▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/typec/ucsi/ucsi.c?h=v6.6.2 ▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/typec/ucsi/ucsi_glink.c?h=v6.6.2 ▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/typec/ucsi/displayport.c?h=v6.6.2 ▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/phy/qualcomm/phy-qcom-qmp-combo.c?h=v6.6.2 ▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/dwc3/dwc3-qcom.c?h=v6.6.2 ▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/soc/qcom/pmic_glink.c?h=v6.6.2 ▪ https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/soc/qcom/pmic_glink_altmode.c?h=v6.6.2
LPM support	https://lore.kernel.org/all/20231017131851.8299-1-quic_kriskura@quicinc.com/
USB DW3C driver	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/dwc3/dwc3-qcom.c?h=v6.6.2
Qualcomm Synopsys femto PHY	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/phy/qualcomm/phy-qcom-snps-femto-v2.c?h=v6.6.2
QMP DisplayPort combo PHY	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/phy/qualcomm/phy-qcom-qmp-combo.c?h=v6.6.2

Title	Resource
USB ADB	https://developer.android.com/tools/adb
`f_fs.c`	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/gadget/function/f_fs.c?h=v6.6.2
Mass storage	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/drivers/usb/gadget/function/f_mass_storage.c?h=v6.6.2
Platform tools (adb/ fastboot)	https://developer.android.com/tools/releases/platform-tools
UVC LibUVC	▪ https://github.com/libuvc/libuvc ▪ https://libuvc.github.io/libuvc/
UVC gadget	https://github.com/wlhe/uvc-gadget
UVC streamer	https://github.com/bsapundzhiev/uvc-streamer
UVC Video4Linux (v4l2-utils)	https://linuxtv.org/downloads/v4l-dvb-apis/driver-api/v4l2-core.html
USB upstream device tree reference	https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/sc7280.dtsi
Qualcomm Linux hardware SoC device-tree node	https://git.linaro.org/kernel-org/linux-next.git/tree/arch/arm64/boot/dts/qcom/qcs6490-rb3gen2.dts
RPM changes in the USB driver and other examples	https://patchwork.kernel.org/project/linux-usb/list/?series=793939&archive=both
Flatten a device tree	▪ https://lore.kernel.org/all/af60c05b-4a0f-51b8-486a-1fc601602515@quicinc.com/ ▪ https://lore.kernel.org/all/20231016-dwc3-refactor-v1-0-ab4a84165470@quicinc.com/

10.6 Acronyms and terms

Acronym or term	Definition
aDSP	Application digital signal processor
ADB	Android debug bridge
ASPM	Active state power management
BDF	Bus device function
CAN	Controller area network
CRC	Cyclic redundancy check
CTS	Clear to send
DLLP	Data link layer packet
DTS	Device tree source
ECRC	End-to-end CRC
EE	Execution environment
ESE	Execute secure environment
FP	Fingerprint

Acronym or term	Definition
GPIO	General-purpose input/output
GSI	Generic software interface
HID	Human interface device
I/O	Input/output
IOMMU	Input/output memory management unit
I2C	Interintegrated circuit
I3C	Improved interintegrated circuit
LSP	List processor
MS	Mass storage
MSI	Message signaled interrupt
NCM	Network control modem
NFC	Near-field communication
NIC	Network interface card
PCIe	Peripheral component interconnect express
QoS	Quality of service
QIM	Qualcomm intelligent multimedia
RTS	Request to send
SCL	Serial clock line
SDK	Software development kit
SDL	Serial data line
SE	Serial engine
SLPI	Sensor low-power island
SPI	Serial peripheral interface
SPMI	System power management interface
SR-IOV	Single root I/O virtualization
SSC	Snapdragon sensor core
TC	Traffic class
TEE	Trusted execution environment
TLP	Transaction layer packet
UEFI	Unified extensible firmware interface
UAC	USB audio class
UART	Universal asynchronous receiver-transmitter
UCSI	USB Type-C connector system software interface
V4L2	Video4Linux2
VC	Virtual channel
Yavta	Yet another V4L2 test application

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.