MSM8626 LA Software Porting Manual

MSM8626 LA 软件移植手册

Software Product Document 软件产品文档

SP80-ND928-6 B

May 27, 2013

Submit technical questions at:
https://support.cdmatech.com/

Qualcomm Technologies, Inc. 5775 Morehouse Drive San Diego, CA 92121 U.S.A.
高通技术股份有限公司，美国加利福尼亚州圣地亚哥市莫豪斯路 5775 号，邮编 92121

Contents 目录

Figures 图目录

Tables 表目录

Revision history

Revision	Date	Description
A	May 2013	Initial release
B	May 2013	This is the bilingual version (English and Chinese).

Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

1 Introduction

1.1 Purpose

This document describes the device tree and its usage; GPIO and the sample code; I2C bus and the sample code; SPI bus and the sample code; camera architecture and the sample code; display architecture and the sample code.

1.2 Scope

This document is target for the engineers who will develop and debug on 8x26 platform.

1.3 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

Code variables appear in angle brackets, e.g., `<number>`.

Commands to be entered appear in a different font, e.g., **`copy a:*.* b:`**.

Button and key names appear in bold font, e.g., click **Save** or press **Enter**.

If you are viewing this document using a color monitor, or if you print this document to a color printer, red typeface indicates data types, blue typeface indicates attributes, and green typeface indicates system attributes.

Parameter types are indicated by arrows:

- → Designates an input parameter
- ← Designates an output parameter
- ↔ Designates a parameter used for both input and output

Shading indicates content that has been added or changed in this revision of the document.

1.4 References

Reference documents are listed in Table 1-1. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*MSM8x26 Linux Kernel BSP and Device Drivers Overview*	80-ND928-25
Q3	*MSM8x26 Linux Android Software Architecture Overview*	80-ND928-23
Q4	*MSM8x26 Software Interface*	80-NC832-2
Q5	*Linux Device Tree MSM8x26 Android SW*	80-NA157-93

1.5 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://support.cdmatech.com/.

If you do not have access to the CDMATech Support Service website, register for access or send email to support.cdmatech@qti.qualcomm.com.

1.6 Acronyms

For definitions of terms and abbreviations, see [Q1].

2 Device Tree

2.1 Introduction

In order to make life easier for embedded board vendors, device tree can be used to represent devices in the system by declaring some nodes and properties in one simple file (dts). Even for on-chip devices and other buses that don't specifically fit in an existing OF specification, device tree is also recommended.

It creates a great flexibility. In the way, the kernel can probe those and match drivers to device, without having to hard code all sorts of tables. It's also more flexible for board vendors to do minor hardware upgrades without significantly impacting the kernel code or cluttering it with special cases.

Device tree usage helps with:

- Reducing the effort required to port to a new board
- Code reusage, by making drivers generic enough to support a set of compatible devices, with the differences being represented in the device tree

For more details, refer to [Q5]. To enable device tree, CONFIG_USE_OF should be set as 'y'. OF means Open Firmware.

2.1.1 DTC

To parse device tree script into binary format, dtc (device tree compile) will be used.

You can use this command to generate dtb zImage directly if no kernel changes:

```
dtc -p 1024 -O dtb -o msm8626.dtb msm8626.dts
cat zImage msm.dtb > dtb-zimage
```

2.1.2 Bootloader

Fastboot will parse boot image and extract device tree information.

DT address = image address + page_size + kernel_actual + ramdisk_actual + second_actual

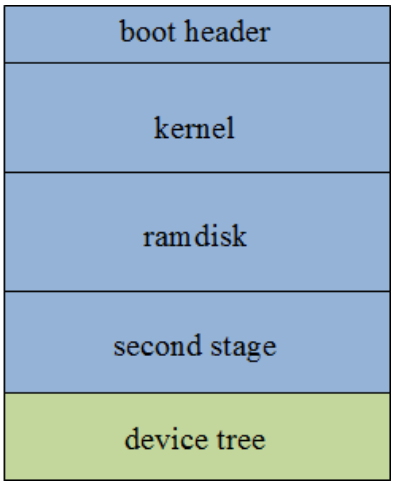

Figure 2-1 Device Tree in boot image

To pass dt table to kernel, fastboot uses the same way as atags. Kernel still uses r2 to save dt table ptr.

Table 2-1 Register table in LK

Register	ATAGS	DT
R0	Null	Null
R1	Machine type number	Unique machine ID is no longer required when bootloader passes a proper device tree. The passed-in ID should be 0xFFFFFFFF to the kernel in register r1
R2	Physical address of tagged list in system RAM	Physical pointer to the device-tree block (defined in chapter II) in RAM. Device tree can be located anywhere in system RAM, but it should be aligned on a 64 bit boundary.

The code source path is `kernel/Documentation/arm/Booting`.

2.1.3 Kernel

2.1.3.1 Relative codes

`kernel/arch/arm/kernel/devtree.c`
`kernel/drivers/of/device.c`
`include/linux/of_gpio.h`

2.1.3.2 Key functions

Table 2-2 Device Tree key API

Function Name	Description
of_platform_populate()	System init function. It will check dt table which passed by fastboot. You can check it in kernel/arch/arm/mach-msm/board-8226.c

of_match_device()	Key function in driver probe function. You should declare it in driver. For example:
	`static const struct of_device_id s5k3l1yx_dt_match[] = {`
	`{.compatible = "qcom,s5k3l1yx", .data = &s5k3l1yx_s_ctrl},`
	`{}`
	`};`
	
	`static struct platform_driver s5k3l1yx_platform_driver = {`
	`.driver = {`
	`.name = "qcom,s5k3l1yx",`
	`.owner = THIS_MODULE,`
	`.of_match_table = s5k3l1yx_dt_match,`
	`},`
	`};`
	
	`static int32_t s5k3l1yx_platform_probe(struct platform_device *pdev)`
	`{`
	`int32_t rc = 0;`
	`const struct of_device_id *match;`
	`match = of_match_device(s5k3l1yx_dt_match, &pdev->dev);`
	`rc = msm_sensor_platform_probe(pdev, match->data);`
	`return rc;`
	`}`

3 GPIO

This chapter describes the Top-Level Mode Multiplexer (TLMM) GPIO and explains how to configure it in the bootloader and kernel.

3.1 Hardware overview

8x26 have 117 GPIOs.

3.1.1 GPIO configuration

This section describes the configure register of GPIO in the MSM8x26 chipset.

Table 3-1 GPIO pysical address

HW Register	Physical address	Reset State
TLMM_GPIO_CFGn, n =[0..116]	0xFD511000+0x10*(n)	0x00000001
TLMM_GPIO_IN_OUTn , n=[0..116]	0xFD511004+0x10*(n)	0x00000000
TLMM_GPIO_OE_0	0xFD513080	0x00000000
TLMM_GPIO_OE_1	0xFD513084	0x00000000
TLMM_GPIO_OE_2	0xFD513088	0x00000000
TLMM_GPIO_OE_3	0xFD51308C	0x00000000

Table 3-2 GPIO CFG table

Bits	Field Name	Description
10	GPIO_HIHYS_EN	Controls the hihys_en for GPIO[n]
9	GPIO_OE	Controls the OE for GPIO[n] when in GPIO mode.
8:6	DRV_STRENGTH	Controls the GPIO pad drive strength. This applies regardless of the FUNC_SEL field selection. 0: DRV_2_MA 1: DRV_4_MA 2: DRV_6_MA 3: DRV_8_MA 4: DRV_10_MA 5: DRV_12_MA 6: DRV_14_MA 7: DRV_16_MA
5:2	FUNC_SEL	Many of the GPIO pads have one or more functional hardware interfaces behind them. This field controls how the pad is used. Set this to the appropriate value for the function desired

1:0	GPIO_PULL	The pad can be configured to employ an internal weak pull up, pull down, or keeper function. This applies regardless of the FUNC_SEL field selection. 0x0: NO_PULL (Disables all pull) 0x1: PULL_DOWN (Weak Pull-down) 0x2: KEEPER (Weak Keeper) 0x3: PULL_UP (Weak Pull-Up)

NOTE: In CFG register, GPIO_OE can enable gpio configuration. Don't need write TLMM_GPIO_OE_[0…3].

NOTE: In CFG register, GPIO_HIHYS_EN can enable gpio hysteresis on input mode, which is mostly useful for noisy low frequency signals like sleep clock etc.

arch/arm/mach-msm/gpiomux-v2.c

```
void __msm_gpiomux_write(unsigned gpio, struct gpiomux_setting val)
{
    uint32_t bits;

    bits = (val.drv << 6) | (val.func << 2) | val.pull;
    if (val.func == GPIOMUX_FUNC_GPIO) {
         bits |= val.dir > GPIOMUX_IN ? BIT(9) : 0;
         __raw_writel(val.dir == GPIOMUX_OUT_HIGH ? BIT(1) : 0,
              GPIO_IN_OUT(gpio));
    }
    __raw_writel(bits, GPIO_CFG(gpio));
    mb();
}
```

3.1.2 Interrupt configuration

Table 3-3 GPIO interrupt CFG table

HW Register	Physical address	Reset State
TLMM_GPIO_INTR_CFGn , n=[0..116]	0xFD511008 + 0x10 * (n)	0x000000E2
TLMM_GPIO_INTR_STATUSn , n=[0..116]	0xFD51100C + 0x10 * (n)	0x00000000

Table 3-4 INT CFG

Bits	Field Name	Description
8	DIR_CONN_EN	0: DISABLE 1: ENABLE
7:05	TARGET_PROC	0: WCSS

		1: SENSORS 2: LPA_DSP 3: RPM 4: KPSS 5: MSS 6: TZ 7: NONE
4	INTR_RAW_STATUS_EN	0: DISABLE 1: ENABLE
3:02	INTR_DECT_CTL	0: LEVEL 1: POS_EDGE 2: NEG_EDGE 3: DUAL_EDGE
1	INTR_POL_CTL	0: POLARITY_0 1: POLARITY_1
0	INTR_ENABLE	0: DISABLE 1: ENABLE

NOTE: POS_EDGE means rising edge trigger

NEG_EDGE means falling edge trigger

Both rising edge and falling edge are supported now.

Table 3-5 GPIO Wakeup interrupt

HW Register	Physical address	Reset State
TLMM_MPM_WAKEUP_INT_EN_0	0xFD512008	0x00000000
TLMM_MPM_WAKEUP_INT_EN_1	0xFD51200C	0x00000000

NOTE: Group 0:
72,71,69,68,67,66,65,64,63,62,54,52,51,50,49,48,46,41,39,37,35,33,31,29,27,21,17,13,9 ,5 ,4 ,1

NOTE: Group 1:

SDC2_DATA_3, SDC2_DATA_1, SDC1_DATA_3,
SDC1_DATA_1,115,113,111,110,109,108,107,106,38,

3.2 Software overview

3.2.1 Configuring GPIO in the bootloader

You can call gpio_tlmm_config() to config GPIO in bootloader.

`bootable/bootloader/lk/platform/msm8226/gpio.c`

```
void gpio_tlmm_config(uint32_t gpio, uint8_t func,
```

```
                        uint8_t dir, uint8_t pull,
                        uint8_t drvstr, uint32_t enable)
{
    uint32_t val = 0;
    val |= pull;
    val |= func << 2;
    val |= drvstr << 6;
    val |= enable << 9;
    writel(val, (uint32_t *)GPIO_CONFIG_ADDR(gpio));
    return;
}
```

3.2.2 Configuring GPIO in the kernel

This section describes the steps required to configure and use any of the GPIO in the MSM8x26 chipset.

For more detailed information, refer to `kernel/arch/arm/boot/dts/msm8226.dts` and `kernel/Documentation/devicetree/bindings/gpio/gpio-msm.txt`.

1. Msm GPIO declaration in device tree:

```
msmgpio: gpio@fd510000 {
     compatible = "qcom,msm-gpio";
     interrupt-controller;
     #interrupt-cells = <2>;
     reg = <0xfd510000 0x4000>;
     gpio-controller;
     #gpio-cells = <2>;
     ngpio = <117>;
     interrupts = <0 208 0>;
     qcom,direct-connect-irqs = <8>;
};
```

2. To specify gpios for a device, add device tree node:

```
device1@sample {
     compatible = "sample-device1";
     reg = <0xf991f000 0x1000>;
     gpios = <&msmgpio 45 0>;
     cs-gpios = <&msmgpio 46 0>;
};
```

3. In device1 driver code, call of_get_gpio() to get the gpio 45/46 handle. Call of_get_named_gpio() with "cs-gpios" as the name parameter can get the gpio 46 handle.

```
static int __devinit device1_probe(struct device_node *np,
                struct device1_platform_data *pdata)
{
u32 reg;
u32 gpio1;
u32 gpio2;

if (of_gpio_count(np) < 2)
     return -ENODEV;
gpio1 = of_get_gpio(np, 0);
gpio2 = of_get_gpio(np, 1);
gpio2 = of_get_named_gpio(np, “cs-gpios”,0);
gpio_request(gpio1, “gpio1”);
gpio_direction_output(gpio1,0);
gpio_request(gpio2, “gpio2”);
gpio_direction_input(gpio2);
...
                }
```

4. To specify gpio interrupt for a device, add device tree node:

```
device2@sample {
     compatible = " sample-device2";
     reg = <0xf991f000 0x1000>;
     interrupt-parent = <&msmgpio>    ;//use msmgpio as interrupt-parent
     interrupts = <17 0x2>;//offset and trigger mode
     irq-gpio = <&msmgpio 17 0>;
};
```

5. In device2 driver:

```
static int __devinit device2_probe(struct device_node *np,
                struct device1_platform_data *pdata)
{
 u32 reg;
 u32 irq-gpio;
 int irq;

 if (of_gpio_count(np) < 1)
     return -ENODEV;

 irq-gpio = of_get_gpio(np, 0);
 gpio_request(irq-gpio, “irq-gpio”);
 gpio_direction_input(irq-gpio);
```

```
irq = gpio_to_irq(irq-gpio);
request_threaded_irq(irq,  NULL, sample_irq_handler, 0x0, DRIVER_NAME,
NULL);
...
    }
```

3.2.3 Step by step

Check `kernel/msm-3.4/drivers/i2c/busses/i2c-gpio.c`.

1. Get gpio information:

   pdata->sda_pin = `of_get_gpio(np, 0);`

   pdata->scl_pin = `of_get_gpio(np, 1);`

2. Get properties:

```
of_property_read_u32(np, "i2c-gpio,delay-us", &pdata->udelay)
of_property_read_u32(np, "i2c-gpio,timeout-ms", &reg)
of_property_read_bool(np, "i2c-gpio,sda-open-drain")
of_property_read_bool(np, "i2c-gpio,scl-open-drain")
of_property_read_bool(np, "i2c-gpio,scl-output-only")
```

3. Get of_device_id:

```
static const struct of_device_id i2c_gpio_dt_ids[] = {
{ .compatible = "i2c-gpio", },
{ /* sentinel */ }
};
```

4. Add dts:

   Use a unique name here. Here set device name as “i2c-gpio@sample”. Based on the value in `of_device_id i2c_gpio_dt_ids[]`, compatible is "i2c-gpio". There are two gpios. In this case, use gpio 45, 46 as example, 45 as sda, 46 as scl. Then set properties based on requirement (clk rate 20K, timeout value 100ms).

```
Set i2c-gpio,delay-us as 50
Set i2c-gpio,timeout-ms as 100
Set i2c-gpio,sda-open-drain as 1
Set i2c-gpio,scl-open-drain as 1
Set i2c-gpio,scl-output-only as 1
```

   Summary:

```
i2c-gpio@sample {
     compatible = "i2c-gpio"
```

```
        gpios = <&msmgpio 45 0
                    &msmgpio 46 0>;
        i2c-gpio,delay-us = <50>;
        i2c-gpio,timeout-ms = <100>;
        i2c-gpio,sda-open-drain = <1>;
        i2c-gpio,scl-open-drain = <1>;
        i2c-gpio,scl-output-only = <1>;
};
```

If use `of_get_named_gpio()` instead of `of_get_gpio()`, for example:

```
pdata->sda_pin = of_get_named_gpio(np, "sample,i2c-sda",0);
pdata->scl_pin = of_get_named_gpio(np, "sample,i2c-scl",0);
```

then the dts should be:

```
i2c-gpio@sample {
        compatible = "i2c-gpio"
        sample,i2c-sda = <&msmgpio 45 0>;
           sample,i2c-scl = <&msmgpio 46 0>;
        i2c-gpio,delay-us = <50>;
        i2c-gpio,timeout-ms = <100>;
        i2c-gpio,sda-open-drain = <1>;
        i2c-gpio,scl-open-drain = <1>;
        i2c-gpio,scl-output-only = <1>;
};
```

4 I2C

This chapter describes the Inter-Integrated Circuit (I2C) and explains how to configure it in the kernel.

4.1 Hardware overview

4.1.1 Qualcomm universal serial engine

The Qualcomm Universal Serial Engine (QUP) provides a general purpose data path engine to support multiple mini cores. Each mini core implements protocol-specific logic. The common FIFO provides a consistent system IO buffer and system DMA model across widely varying external interface types. For example, one pair of FIFO buffers can support SPI and I2C mini cores independently.

4.1.2 I2C core

The I2C core supports I2C Standard (100 kHz) and Fast (400 kHz). The following key features have been added for the MSM8x26:

- BAM integration
- Support for I2C tag version

4.1.3 QUP base addresses and IRQs

To match the labeling in the software interface manual, each QUP is identified by BLSP core and QUP core (1 to 6).

NOTE: The MSM8x26 chipset contains only one BLSP core.

Table 4-1 QUP physical address

BLSP hardware ID	QUP Core	Physical address
BLSP1	BLSP 1 QUP 1	0xF9923000
BLSP1	BLSP 1 QUP 2	0xF9924000
BLSP1	BLSP 1 QUP 3	0xF9925000
BLSP1	BLSP 1 QUP 4	0xF9926000
BLSP1	BLSP 1 QUP 5	0xF9927000
BLSP1	BLSP 1 QUP 6	0xF9928000

Table 4-2 QUP IRQ

BLSP hardware ID	QUP Core	IRQ
BLSP1	BLSP 1 QUP 1	95
BLSP1	BLSP 1 QUP 2	96
BLSP1	BLSP 1 QUP 3	97
BLSP1	BLSP 1 QUP 4	98
BLSP1	BLSP 1 QUP 5	99
BLSP1	BLSP 1 QUP 6	100

4.2 Software overview

4.2.1 Configuring the QUP core as I2C in the kernel

This section describes the steps required to configure and use any of the QUP cores which is available in the MSM8x26 chipset as an I2C device.

By default, Qualcomm has preconfigured BLSP1_QUP4 and BLSP1_QUP5 as I2C. For more detailed information, refer to `/kernel/arch/arm/boot/dts/msm226.dtsi, kernel/Documentation/devicetree/bindings/i2c/i2c-qup.txt.`

4.2.2 Code modification

The following examples show how to configure BLSP1_QUP5 as I2C for the MSM8226 chipset.

File to modify:

`/kernel/arch/arm/boot/dts/msm8226.dts`

1. Add a new device tree node.

```
i2c@f9927000 { /* BLSP1 QUP5 */
     cell-index = <5>; //BUS ID can be any # recommend OEM to use large
# so won't conflict with Qualcomm id
     compatible = "qcom,i2c-qup"; //Manufacturer model
     #address-cells = <1>;// address for slave chips
     #size-cells = <0>; // size for slave chips
     reg-names = "qup_phys_addr"; //Name of QUP_BASE
     reg = <0xf9927000 0x1000>; //BLSP1 QUP5 BASE address and size
     interrupt-names = "qup_err_intr"; //Interrupt Name
     interrupts = <0 99 0>;
     gpios = <&msmgpio 19 0>, /* SCL */
          <&msmgpio 18 0>; /* SDA */
    qcom,i2c-bus-freq = <100000>; //Output Clk frequency (can be 100KHz,
or 400KHz)
     qcom,i2c-src-freq = <19200000>; //Source clock frequency
};
```

2. Add a clock node.

This step describes how to modify the clock table.

File to modify:
`Project_Root/kernel/arch/arm/mach-msm/clock-8226.c`

The code for adding a clock node:

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f9927000.i2c"),
  /*
  Add a node to QUP Core clock.
  Note: In clock regime QUP cores are label #1 to #6.
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_qup5_i2c_apps_clk.c, "f9927000.i2c"),
```

3. Set up the GPIO.

NOTE: This step is based on the MSM8226-ES release.

File to modify:
`Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

```
static struct gpiomux_setting gpio_i2c_config = {
  .func = GPIOMUX_FUNC_3, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE for UART
};

static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
    .gpio      = 18,  //BLSP1_QUP5 I2C_SDA
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_i2c_config,
    },
  },
  {
    .gpio      = 19,  //BLSP1_QUP5 I2C_SCL
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_i2c_config,
    },
  },
```

Registering the GPIOs:

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs, ARRAY_SIZE(msm_blsp_configs));
}
```

4.2.3 Quick verification

This section describes how to verify the I2C bus.

Ensure that the bus is registered.

If you enter all the information properly, you should see I2C bus registered under **/dev/i2c-**, where cell-index matches the bus number.

```
adb shell --> Get adb shell
cd /dev/
ls i2c* --> to List all the I2C buses
root@android:/dev # ls i2c*
ls i2c*
i2c-0
i2c-10
i2c-2
```

If bus is properly registered, test it using a simple test program. This example is created in the following directory: `/vendor/qcom/proprietary/kernel-tests/i2c-test/`.

Once you compile and run the program, if the I2C bus is properly programmed and the slave device responds, you will see the following output:

```
root@android:/data # ./i2c-test
./i2c-test
Device Open successfull [3]
I2C RDWR Returned 1
```

If an error occurs, you will see the following output:

```
./i2c-test
Device Open successfull [3]
I2C RDWR Returned -1
```

If I2C RDWR returns -1, check the kernel log for the driver error message. If you see the following error message:

```
[ 6131.397699] qup_i2c f9927000.i2c: I2C slave addr:0x54 not connected
```

Typically, the slave device doesn't send an acknowledgment. The bus is properly configured and at least the start bit and address bit are sent from the bus, but the slave refuses it and doesn't acknowledge it.

At this point, the debugging should focus on the slave device to make sure it is properly powered up and ready to accept messages.

If you see the following error message:

```
[ 6190.209880] qup_i2c f9927000.i2c: Transaction timed out, SL-AD = 0x54
[ 6190.216389] qup_i2c f9927000.i2c: I2C Status: 132100
[ 6190.221247] qup_i2c f9927000.i2c: QUP Status: 0
[ 6190.225857] qup_i2c f9927000.i2c: OP Flags: 10
```

This error message could be due to multiple issues:

- Invalid software configuration
- Invalid hardware configuration
- Slave device issues

4.3 Debugging tips

If you do a simple read/write, but the I2C always fails, the following sections provide debugging tips.

- Check SDA/SCL Idling.
- Scope the BUS to ensure that the SDA/SCL is idling at the high logic level. If it is not idling high, there is either a hardware configuration problem or the GPIO settings are invalid.
- Set a breakpoint at the line where the error message is coming. For example, at the Transaction timed out message:

```
static int
qup_i2c_xfer(struct i2c_adapter *adap, struct i2c_msg msgs[], int num)
{
...//Put a breakpoint inside if statement.
  if (!timeout) {
    uint32_t istatus = readl_relaxed(dev->base +
 QUP_I2C_STATUS);
```

4.4 Step by step

4.4.1 Registering a slave device using the device tree

Once the I2C bus is properly verified, you can create a slave device driver and register it with I2C bus. Refer to the following files for examples.

For an I2C slave device, refer to `msm8226-qrd.dts`.

For Synaptic Touch Screen driver registration, refer to `kernel/drivers/input/touchscreen/synaptics_i2c_rmi4.c`.

The following example shows the minimum requirement for properly registering a slave device using the device tree.

4.4.2 Creating a device tree node

File to modify:

`/kernel/arch/arm/boot/dts/msm8226-qrd.dts`

Add a new device tree node:

```
i2c@f9927000 { /* BLSP1 QUP5 */
   synaptics@20 {//Slave driver and slave Address
        compatible = "synaptics,rmi4";//Manufacture, model

        reg = <0x20>;//Slave Address
        interrupt-parent = <&msmgpio>;//GPIO handler
        interrupts = <17 0x2>;//GPIO # will be converted to gpio_irq
        vdd-supply = <&pm8226_l19>;
        vcc_i2c-supply = <&pm8226_lvs1>;
        synaptics,reset-gpio = <&msmgpio 16 0x00>;//Pass a GPIO
        synaptics,irq-gpio = <&msmgpio 17 0x00>;
        synaptics,button-map = <139 102 158>;
        synaptics,i2c-pull-up;
        synaptics,reg-en;
   };
};
```

4.4.3 Sample slave driver

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/i2c.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <linux/gpio.h>
#include <linux/debugfs.h>
```

```
#include <linux/seq_file.h>
#include <linux/regulator/consumer.h>
#include <linux/string.h>
#include <linux/of_gpio.h>

#ifdef CONFIG_OF //Open firmware must be defined for dts useage
static struct of_device_id qcom_i2c_test_table[] = {
   { .compatible = "qcom,i2c-test",}, //Compatible node must match dts
   { },
};
#else
#define qcom_i2c_test_table NULL
#endif

//I2C slave id supported by driver
static const struct i2c_device_id qcom_id[] = {
   { "qcom_i2c_test", 0 },
   { }
};

static int i2c_test_test_transfer(struct i2c_client *client)
{
  struct i2c_msg xfer; //I2C transfer structure
  u8 buf = 0x55;  //data to transfer
  xfer.addr = client->addr;
  xfer.flags = 0;
  xfer.len = 1;
  xfer.buf = &buf;

  return i2c_transfer(client->adapter, &xfer, 1);
}

static int __devinit i2c_test_probe(struct i2c_client *client,
        const struct i2c_device_id *id)
{
  int irq_gpio = -1;
  int irq;
  int addr;
  //Parse data using dt.
  if(client->dev.of_node){
    irq_gpio = of_get_named_gpio_flags(client->dev.of_node,
"qcom_i2c_test,irq-gpio", 0, NULL);
  }
  irq = client->irq; //GPIO irq #. already converted to gpio_to_irq
```

```
  addr = client->addr; //Slave Addr
  dev_err(&client->dev, "gpio [%d] irq [%d] gpio_irq [%d] Slaveaddr [%x]
\n", irq_gpio, irq,
     gpio_to_irq(irq_gpio), addr);

  //You can initiate a I2C transfer anytime using i2c_client *client
structure
  i2c_test_test_transfer(client);

  return 0;
}

//I2C Driver Info
static struct i2c_driver i2c_test_driver = {
    .driver = {
         .name     = "qcom_i2c_test",
         .owner    = THIS_MODULE,
         .of_match_table = qcom_i2c_test_table,
    },
    .probe        = i2c_test_probe,
    .id_table       = qcom_id,
};

//Easy wrapper to do driver init
module_i2c_driver(i2c_test_driver);

MODULE_DESCRIPTION("I2C TEST");
MODULE_LICENSE("GPL v2");
```

In the kernel log, the following message indicates the Device Tree was successfully configured.

```
<3>[    2.670731] qcom_i2c_test 2-0052: gpio [61] irq [306] gpio_irq [306]
                  Slaveaddr [52]
```

5 UART

This chapter describes the UART and explains how to configure it in the bootloader and kernel.

5.1 Hardware overview

5.1.1 BLSP

BLSP is a new design that replaces the legacy GSBI core in the MSM8x26 chipset families. All legacy GSBI software registers have been removed.

Each BLSP block includes six QUP and six UART cores. Instead of the legacy data mover (ADM), BAM is used as a hardware data mover. Each BLSP peripheral is statically connected to a pair of BAM pipes, consists of 26 pipes that can be used for data move operations and supports both BAM and non-BAM based data transfer.

5.1.2 UART core

Key features are added for the chipset:

- BAM support
- Single-character mode

The UART core is used for transmitting and receiving data through a serial interface. It is used for communicating with other UART protocol devices. Configuration of this mode is primarily defined by the UART_DM_MR1 and UART_DM_MR2 registers.

5.1.3 Base addresses

To match the labeling in the software interface manual, each UART is identified by the BLSP core 1 and UART core (1 to 6).

Table 5-1 UART physical address

BLSP hardware ID	UART Core	Physical address (UART_DM_BASE_ADDRESS)
BLSP1	BLSP 1 UART 1	0xF991D000
BLSP1	BLSP 1 UART 2	0xF991E000
BLSP1	BLSP 1 UART 3	0xF991F000
BLSP1	BLSP 1 UART 4	0xF9920000
BLSP1	BLSP 1 UART 5	0xF9921000
BLSP1	BLSP 1 UART 6	0xF9922000

5.1.4 IRQ numbers

To match the labeling in the software interface manual, each UART is identified by the BLSP core 1 and UART core (1 to 6).

Table 5-2 UART Interrupt Table

BLSP hardware ID	UART Core	IRQ #
BLSP1	BLSP 1 UART 1	107
BLSP1	BLSP 1 UART 2	108
BLSP1	BLSP 1 UART 3	109
BLSP1	BLSP 1 UART 4	110
BLSP1	BLSP 1 UART 5	111
BLSP1	BLSP 1 UART 6	112

5.2 Software overview

5.2.1 UART in the bootloader

By default, Qualcomm is configured the BLSP1 UART3 in the bootloader to use. This section describes the changes required to configure a different UART for debugging purpose.

5.2.1.1 Enabling the UART for debugging

File to modify:

```
Project_Root/bootable/bootloader/lk/project/msm8226.mk
```

Set the flag, WITH_DEBUG_UART, to TRUE:

```
DEFINES += WITH_DEBUG_UART=1
```

5.2.1.2 Setting the base address

File to modify:

```
Project_Root/bootable/bootloader/lk/target/msm8226/init.c
```

Set the correct base address:

```
void target_early_init(void)
{
#if WITH_DEBUG_UART
  /*
  First argument represents the ID (can be any since it's not used)
  Second argument, if it is a GSBI base, must be 0
  Third Argument is the physical address for UART CORE defined in
  /bootable/bootloader/lk/platform/msm8226/include/platform/iomap.h
  */
```

```
  uart_dm_init(1, 0, BLSP1_UART2_BASE);//it is uart[0..5] instead of
uart[1..6]
#endif
```

5.2.1.3 Configuring the clock

Two clocks are required for the UART to properly work on the MSM8226 device:

- BLSP AHB clock
- UART core clock

File to modify:

`Project_Root/bootable/bootloader/lk/platform/msm8226/acpuclock.c`

Enable the clocks:

```
void clock_config_uart_dm(uint8_t id)
{
  int ret;
  /*
    NOTE: In clock regime clocks are # from 1 to 6 so UART0 would
    be identified as UART1
  */
  //iface_clk is BLSP clk
  ret = clk_get_set_enable("uart3_iface_clk", 0, 1);

  //core_clock is UART clock.
  ret = clk_get_set_enable("uart3_core_clk", 7372800, 1);
}
```

Register the clocks with the clock regime. The BLSP1_AHB clock is enabled by default.

5.2.1.4 Configuring the GPIO

File to modify:

`Project_Root/bootable/bootloader/lk/platform/msm8226/gpio.c`

Configure the correct GPIO:

```
void gpio_config_uart_dm(uint8_t id)
{
  /*
  Configure the RX/TX GPIO
  Argument 1: GPIO #
  Argument 2: Function (Please see device pinout for more information)
  Argument 3: Input/Ouput (Can be 0/1)
  Argument 4: Should be no PULL
  Argument 5: Drive strength
```

```
        Argument 6: Output Enable (Can be 0/1)
        */
        gpio_tlmm_config(9, 2, GPIO_INPUT, GPIO_NO_PULL,
        GPIO_8MA, GPIO_DISABLE);
        gpio_tlmm_config(8, 2, GPIO_OUTPUT, GPIO_NO_PULL,
        GPIO_8MA, GPIO_DISABLE);
      }
```

5.2.1.5 Debugging tips

If the UART is properly configured, the following message is displayed on the serial console:

Android Bootloader - UART_DM Initialized!!!

If you do not see a message, use the following debugging tips.

- Set a breakpoint

  Put a breakpoint in the bootloader inside the uart_dm_init function line at msm_boot_uart_dm_write(uart_dm_base, data, 44) in this file:

  /bootable/bootloader/lk/platform/msm_shared/uart_dm.c

- Check properly configured GPIOs

  Check the GPIO configuration register, GPIO_CFGn, to ensure that the GPIO settings valid.

5.2.2 Low-speed UART in the kernel

Low-speed UART is a FIFO-based UART driver designed for small data transfer at a slow rate, such as console debugging or IrDA transfer. If a large amount of data is transferred, or for a high-speed transfer, Qualcomm recommends using the high-speed UART driver.

By default, BLSP1 UART3 Base 0xF991F000 is preconfigured as low-speed UART.

5.2.2.1 Code modification

The following examples describe how to configure BLSP1 UART3 to use the low-speed UART driver on the MSM8226 chipset.

The same methods can be applied to the MSM8x26 chipset, but with the following modifications:

Device Tree Source -- /kernel/arch/arm/boot/dts/msm8226.dtsi

Clk Table -- /kernel/arch/arm/mach-msm/clock-8226.c

In this file, update the msm_clocks_8226[] table.

GPIO Table -- /kernel/arch/arm/mach-msm/board-8226-gpiomux.c

1. Creating a device tree node

   For more details, refer to the device tree source:

   /kernel/arch/arm/boot/dts/msm8226.dtsi

NOTE: Currently, there is no ID field to explicitly assign to the UART. Therefore, the first UART registered is ttyHSL0, the second one is ttyHSL1, etc.

File to modify:
`/kernel/arch/arm/boot/dts/msm8226.dtsi`

Add a new Device Tree node:

```
serial@f991f000 { //0xF991F000 is the UART_DM Base Address
  compatible = "qcom,msm-lsuart-v14"; //manufacture, model of serial
driver
  reg = <0xf991f000 0x1000>; //Base address UART_DM and size
  /*
    First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
    Second Field: Interrupt #
    Third field: Trigger type, keep 0
    For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
  */
  interrupts = <0 109 0>;
  status = "ok"; //Status OK enables it
};
```

2. Set up the clocks

File to modify:
`Project_Root/kernel/arch/arm/mach-msm/clock-8226.c`

Add a clock node:

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f991f000.serial"),
  /*
  Add a node to UART Core clock.
  Note: In clock regime UART clocks are label #1 to #6.
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_uart3_apps_clk.c, "f991f000.serial"),
```

3. Set up the GPIO

File to modify:
`Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

a. Create a configuration structure:

```
static struct gpiomux_setting gpio_uart_config = {
```

```
  .func = GPIOMUX_FUNC_2, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE for UART
};
```

b. Create a GPIO array:

```
static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
    .gpio      = 8,  //BLSP1 UART3 TX
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_uart_config,
    },
  },
  {
    .gpio      = 9,  //BLSP1 UART3 RX
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_uart_config,
    },
  },
```

c. Register the GPIOs:

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs,
ARRAY_SIZE(msm_blsp_configs));
```

4. Debugging tips

This section describes how to verify the low-speed UART.

a. Checking the registration

Ensure that the UART is properly registered with the TTY stack. Run the following commands:

```
adb shell -> start a new shell
ls /dev/ttyHSL* -> Make sure UART is properly registered
```

If you do not see your device, check your code modification to ensure that all the information is defined and correct.

b. Checking the Internal loopback

i Run the following commands to enable loopback:

```
adb shell
mount -t debugfs none /sys/kernel/debug  -> mount debug fs
cd /sys/kernel/debug/msm_serial_hsl      -> directory for Low
Speed UART
echo 1 > loopback.#                      -> enable loopback. # is
device #
cat loopback.#                           -> make sure returns 1
```

ii Open another shell to dump the UART Rx data:

```
adb shell
cat /dev/ttyHSL#  ->Dump any data UART Receive
```

iii Transmit some test data through a separate shell:

```
adb shell
echo "This Document Is Very Much Helpful" > /dev/ttyHSL# -
>Transfer data
```

If the loopback works:

You will see your test message loop continuously in the command shell until you exit the cat program. This is because of the internal loopback and how the cat program opens the UART.

You can assume the UART is properly configured and only the GPIO settings need to be confirmed.

c. Loopback does not work

If loopback does not work, check the clock settings.

Before checking the clock, ensure that the UART is still in the Active state. Open the UART from the shell:

```
adb shell
cat /dev/ttyHSL#  ->Dump any data UART Receive
```

d. Loopback works, but there is no output

If the loopback works but you do not see any signal output, check the GPIO settings.

5.2.3 High-speed UART in the kernel

UART_DM can be configured as a BAM-based UART. This driver is designed for high-speed, large data transfers such as BlueTooth communication.

5.2.3.1 BLSP BAM address and IRQ

Each BLSP core has a master BAM hardware block and one dedicated IRQ.

Table 5-3 BLSP BAM physical address

BLSP hardware ID	UART Core	Physical address (BLSP_BAM, IRQ)
BLSP1	BLSP 1 UART[1:6]	0xF9904000, 238

5.2.3.2 BAM pipe assignment

Each BLSP UART core has two unique pipes (Consumer and Producer) pre-allocated for data mover operations. Table UartPipeAssignment identifies the pipe to use for each BLSP UART_DM core.

Table 5-4 BAM pipe

BLSP hardware ID	UART Core	Pipes (Consumer and Producer)
BLSP1	BLSP 1 UART1	0,1
BLSP1	BLSP 1 UART2	2,3
BLSP1	BLSP 1 UART3	4,5
BLSP1	BLSP 1 UART4	6,7
BLSP1	BLSP 1 UART5	8,9
BLSP1	BLSP 1 UART6	10,11

5.2.3.3 Code modification

The following examples show how to configure BLSP1_UART2 as a high-speed UART on the MSM8226 chipset.

Device Tree Source -- `/kernel/arch/arm/boot/dts/msm8226.dtsi`

Clk Table -- `/kernel/arch/arm/mach-msm/clock-8226.c`

In this file, update the msm_clocks_8226[] table.

GPIO Table -- `/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

ARM TrustZone Changes -- `/trustzone_images/core/hwengines/bam/8x26/bamtgtcfgdata_tz.h`

1. Create a device tree node

   For detailed information, refer to the Device Tree documentation: `/kernel/Documentation/devicetree/bindings/tty/serial/msm_serial_hs.txt.`

   File to modify: `/kernel/arch/arm/boot/dts/msm8226.dtsi`

   Or any dts/dtsi file used.

   The following elements are the minimum requirement:

```
  uart2: uart@f991e000 { //0xF991E000 is the UART_DM Base address for
BLSP1_UART2
    /*
      UART ID, Recommend OEM to use Large ID so does not conflict with
      Qualcomm configured ID.
      cell-index 102 would represent as /dev/ttyHS102
    */
```

```
            cell-index = <102>; //UART ID, Recommend OEM to use Large ID # that
        < 255
            compatible = "qcom,msm-hsuart-v14"; //manufacture, model (must be
        same)
            status = "ok"; //"ok" or "okay" to enable
            /*
            First Row UART_DM Base and Size always 0x1000
            Second Row is BAM address, size always 0x19000
            For BLSP1 UART0:5 Bam Address = 0xF9904000
            */
            reg = <0xf991e000 0x1000>,
            <0xf9904000 0x19000>;
            reg-names = "core_mem", "bam_mem"; //Keep the same names
            /*
            First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
            Second Field: Interrupt #
            Third field: Trigger type, keep 0
            For more
        information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
            First Row UART_DM IRQ #
            Second Row is BAM IRQ
            For BLSP1 UART0:5 IRQ = 238
            */
            interrupts = <0 108 0>, <0 238 0>;
            interrupt-names = "core_irq", "bam_irq"; //Keep same
            qcom,bam-tx-ep-pipe-index = <2>; //BAM Consumer Pipe
            qcom,bam-rx-ep-pipe-index = <3>; //BAM Producer Pipe
          };
```

2. Set up the clocks

NOTE: This section is based on the MSM8226-ES release. The clocks do not support Device Tree Format.

File to modify:

`Project_Root/kernel/arch/arm/mach-msm/clock-8226.c`

Add a node to the UART core clock:

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f991e000.uart"),
  /*
  Add a node to UART core clock
  Note: In clock regime UART clocks are labeled #1 to 6, so UART2 is
```

```
        BLSP1 UART 1
        */
        CLK_LOOKUP("core_clk", gcc_blsp1_uart2_apps_clk.c, "f991e000.uart"),
```

3. Set up the GPIO

NOTE: This section is based on the MSM8226-ES release; currently, the GPIO driver does not fully support the Device Tree format.

File to modify:
`Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

a. Create a configuration structure:

```
static struct gpiomux_setting gpio_uart_config = {
  .func = GPIOMUX_FUNC_2, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE for UART
};
```

b. Create a GPIO array:

```
static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
       .gpio      = 4,              /* BLSP2 UART TX */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
  {
       .gpio      = 5,              /* BLSP2 UART RX */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
  {
       .gpio      = 6,              /* BLSP2 UART CTS */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
  {
       .gpio      = 7,              /* BLSP2 UART RFR */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
```

```
  },
```

c. Register the GPIOs:

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  /*
    Make sure the same GPIO # not registered twice or used by
    another subsystem such as modem.
  */
  msm_gpiomux_install(msm_blsp_configs,
ARRAY_SIZE(msm_blsp_configs));
```

4. Change TrustZone

   Each BLSP can be used by any subsystem such as Modem and LPASS. It is important to give ownership of the BAM Pipes to Applications processor.

   File to modify:
   `/trustzone_images/core/hwengines/bam/8226/bamtgtcfgdata_tz.h`

   Allocate BLSP BAM pipes with the applications core:

```
/*
  bam_tgt_blsp1_secconfig = BLSP1 Core

  Each Bit You set represent Pipe #.
    For example Bit 0 = Pipe # 0
                Bit 5 = Pipe # 5

  Any bit set on TZBSP_VMID_AP owns by APPs
*/
bam_sec_config_type bam_tgt_blsp1_secconfig =
{
    { //Since BLSP1_UART2 is pipe 2, 3.  Bit 2, 3 should be set to APPS
        {0x00C0FFFF, TZBSP_VMID_AP}, //Bit 2,3 Should be Set
        {0x003F0000, TZBSP_VMID_LPASS } //Make sure Bit 2,3 is Cleared
    }
};
```

5.2.3.4 Debugging tips

This section describes how to verify the high-speed UART.

- Check the registration

Ensure that the UART is properly registered with the TTY stack. Run the following commands:

```
adb shell -> start a new shell
ls /dev/ttyHS* -> Make sure UART is properly registered
```

If you do not see your device, check your code modification to ensure that all the information is defined and correct.

- Check the Internal loopback

  a. Run the following commands to enable loopback:

  ```
  adb shell
  mount -t debugfs none /sys/kernel/debug  -> mount debug fs
  cd /sys/kernel/debug/msm_serial_hs       -> directory for High Speed
  UART
  echo 1 > loopback.#                      -> enable loopback. # is
  device #
  cat loopback.#                           -> make sure returns 1
  ```

  b. Open another shell to dump the UART Rx data:

  ```
  adb shell
  cat /dev/ttyHS#  ->Dump any data UART Receive
  ```

  c. Transmit some test data through a separate shell:

  ```
  adb shell
  echo "This Is A Helpful Document" > /dev/ttyHS# ->Transfer data
  ```

  If the loopback works:

  You will see your test message loop continuously in the command shell until you exit the cat program. This is because of the internal loopback and how the cat program opens the UART.

  You can assume the UART is properly configured and only the GPIO settings need to be confirmed.

- Loopback does not work
  - Ensure that the applications processor can access the designated pipe.

Table 5-5 BAM pipe

BLSP Pipe	PIPE_TRUST_REG	PIPE_CTRL_REG
BLSP1_P 0	0xF9905030	0xF9905000
BLSP1_P 1	0xF9906030	0xF9906000
BLSP1_P 2	0xF9907030	0xF9907000
BLSP1_P 3	0xF9908030	0xF9908000
BLSP1_P 4	0xF9909030	0xF9909000
BLSP1_P 5	0xF990a030	0xF990a000

Once the system boots up, attach a JTAG to the RPM and CortexA7 cores (t32 cmd: sys.m.a).

Break all the CortexA7 cores (t32 cmd: b).

Turn on the BLSP_AHB_CLK by ensuring that bits 17 and 15 are set for the APCS_CLOCK_BRANCH_ENA_VOTE register
(APCS_CLOCK_BRANCH_ENA_VOTE = 0xFC401484).

NOTE: Only Bit 17 is required for the MSM8x26 Chipset.

- Ensure that BLSP_PIPE_TRUST_REG is set to zero for applications processor. (Read using Secure mode.)

```
D AZ:0xF9907030 -- BLSP1 pipe 2 \\
D AZ:0xF9908030 -- BLSP1 pipe 3
```

- Ensure that you can read/write to PIPE_CTRL_REG in Non-secure mode.

```
D.S A:0x0:0xF9907000 \%LE \%LONG 0xABCD -- Writing pipe 2 \\
D.S A:0x0:0xF9908000 \%LE \%LONG 0xABCD -- Writing pipe 3 \\
data.in A:0xF9907000 /long -- Reading pipe 2 \\
data.in A:0xF9908000 /long -- Reading pipe 3
```

If the above test failed, check the TrustZone modifications.

- Check the clock settings

Before checking the clock, ensure that the UART is still in the Active state. Open the UART from the shell:

```
adb shell
cat /dev/ttyHS#  ->Dump any data UART Receive
```

- Loopback works, but there is no output

If the loopback works but you do not see any signal output, check the GPIO settings.

Optional configurations

Once basic UART functionality is verified, you can add the following settings to enhance the UART_DM functionality.

- Runtime GPIO configuration

Optionally, the UART driver supports runtime GPIO configuration so that during the Active state, the GPIO is configured as UART mode. During the Suspend state, the GPIO is converted back to the default, user-specified Suspend/Power Saving mode.

File to modify:

```
Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c
```

d. Create a GPIO active/suspend configuration structure.

```
  //GPIO Configuration during Active State
  static struct gpiomux_setting gpio_uart_active_config = {
    .func = GPIOMUX_FUNC_2, //Please look @ GPIO function table for
proper value.
    .drv = GPIOMUX_DRV_8MA,  //Drive Strength
    .pull = GPIOMUX_PULL_NONE,//Should be PULL NONE
  };
 //Obtain recommended suspend settings from HW engineer
  static struct gpiomux_setting gpio_uart_suspend_config = {
    .func = GPIOMUX_FUNC_GPIO, //SUSPEND Configuration
    .drv = GPIOMUX_DRV_2MA, //Drive Strength
    .pull = GPIOMUX_PULL_NONE, //PULL Configuration
  };
```

e. Create a GPIO array.

```
  static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
    {
      .gpio      = 4,               /* BLSP2 UART TX */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
    {
      .gpio      = 5,               /* BLSP2 UART RX */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
    {
      .gpio      = 6,               /* BLSP2 UART CTS */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
    {
      .gpio      = 7,               /* BLSP2 UART RFR */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
```

```
    },
```

f. Register the GPIOs.

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs,
ARRAY_SIZE(msm_blsp_configs));
```

g. Add GPIO numbers to the Device Tree.

File to modify:

`/kernel/arch/arm/boot/dts/msm8226.dtsi`

Or any dts/dtsi file used.

The following additional settings to the Device Tree are required to enable runtime GPIO configuration with the UART core.

```
  //Add following additional nodes to enable RunTime GPIO
Configuration
  uart2: uart@f991e000 { //0xF991E000 is the UART_DM Base address for
BLSP1_UART2
    /*
    Please look @
/kernel/Documentation/devicetree/bindings/gpio/gpio-msm.txt
    for more info on how to specify GPIO
    First Field = Controller name
    Second Field = GPIO #
    Third Field = Not Used.
    */
    qcom,tx-gpio = <&msmgpio 4 0>;
    qcom,rx-gpio = <&msmgpio 5 0>;
    qcom,cts-gpio = <&msmgpio 6 0>;
    qcom,rfr-gpio = <&msmgpio 7 0>;
  };
```

For the latest details, refer to the Device Tree documentation:

`/kernel/Documentation/devicetree/bindings/tty/serial/msm_serial_hs.txt`。

6 Serial Peripheral Interface

This chapter describes the Serial Peripheral Interface (SPI) and explains how to configure it in the kernel.

6.1 Hardware overview

6.1.1 SPI core

The SPI allows full-half-duplex, synchronous, serial communication between a master and slave. There is no explicit communication framing, error checking, or defined data word length.

Key features:

- Supports up to 48 MHz
- Supports 4 to 32 bits per word of transfer
- Supports a maximum of four Chip Selects (CS) per bus
- Supports BAM (new for MSM8x26 chipsets)

Table 6-1 Base address

BLSP hardware ID	QUP Core	Physical address
BLSP1	BLSP 1 QUP 1	0xF9923000
BLSP1	BLSP 1 QUP 2	0xF9924000
BLSP1	BLSP 1 QUP 3	0xF9925000
BLSP1	BLSP 1 QUP 4	0xF9926000
BLSP1	BLSP 1 QUP 5	0xF9927000
BLSP1	BLSP 1 QUP 6	0xF9928000

The SPI core shares the same base address as the QUP core; or the correct values.

6.2 Software overview

6.2.1 Configuring the QUP core as SPI in the kernel

This section describes the steps required to configure and use any of the QUP cores available in the MSM8x26 chipset as an SPI device.

By default, Qualcomm has preconfigured BLSP1_QUP0 as SPI. For more detailed information, refer to `/kernel/arch/arm/boot/dts/msm8226.dtsi`.

6.2.1.1 Code modification

The following examples show how to configure BLSP1_QUP1 as SPI for the MSM8226 chipset. The same methods can be applied to the MSM8x26 chipset, but with the following modifications:

Device Tree Source -- `/kernel/arch/arm/boot/dts/msm8226.dtsi`

Clk Table -- `/kernel/arch/arm/mach-msm/clock-8226.c`

In this file, update the msm_clocks_8226[] table.

GPIO Table -- `/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

TrustZone Changes -- `/trustzone_images/core/hwengines/bam/8x26/bamtgtcfgdata_tz.h`

1. Creating a device tree

   File to modify:
   `/kernel/arch/arm/boot/dts/msm8226.dts`

   Add a new Device Tree node:

```
spi_0: spi@f9923000 { /* BLSP1 QUP1 */
     compatible = "qcom,spi-qup-v2";//Manufactur and Model
     #address-cells = <1>;
     #size-cells = <0>;
     reg-names = "spi_physical", "spi_bam_physical";
     reg = <0xf9923000 0x1000>,//Base address for BLSP1_QUP1 and size
          <0xf9904000 0xF000>;
     interrupt-names = "spi_irq", "spi_bam_irq";
     interrupts = <0 95 0>, <0 238 0>;
                /*
                First Field: 0 SPI interrupt (Shared Peripheral
Interrupt)
                Second Field: Interrupt #
                Third field: Trigger type, keep 0
                For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
                */

     spi-max-frequency = <19200000>;//Maximum supported frequency in HZ

     gpios = <&msmgpio 3 0>, /* CLK  */
          <&msmgpio 1 0>, /* MISO */
          <&msmgpio 0 0>; /* MOSI */
     cs-gpios = <&msmgpio 22 0>;

     qcom,infinite-mode = <0>;
     qcom,use-bam;
     qcom,ver-reg-exists;
```

```
        qcom,bam-consumer-pipe-index = <12>;// BAM consumer PIPE
        qcom,bam-producer-pipe-index = <13>;// BAM producer PIPE
    };
```

For latest detail please follow
`/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt`.

2. Set the clocks

   This step describes how to modify the clock table.

   File to modify:
   `Project_Root/kernel/arch/arm/mach-msm/clock-8226.c`

   Add a clock node:

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f9923000.spi"),
  /*
  Add a node to QUP Core clock.
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_qup1_spi_apps_clk.c, "f9923000.spi"),
```

3. Set the GPIO

   File to modify:
   `Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

   Create a configuration structure:

```
static struct gpiomux_setting gpio_spi_config = {
  .func = GPIOMUX_FUNC_1, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE
};
```

4. Create a GPIO array

```
static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
    .gpio      = 0,  //BLSP1_QUP1 MOSI
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_spi_config,
    },
  },
  {
    .gpio      = 1,  //BLSP1_QUP5 MISO
```

```
            .settings = {
              [GPIOMUX_SUSPENDED] = &gpio_spi_config,
            },
          },
          {
            .gpio      = 2,  //BLSP1_QUP5 CS
            .settings = {
              [GPIOMUX_SUSPENDED] = &gpio_spi_cs_config,
            },
          },
          {
            .gpio      = 3,  //BLSP1_QUP5 CLK
            .settings = {
              [GPIOMUX_SUSPENDED] = &gpio_spi_config,
            },
          },
```

Register the GPIOs:

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs, ARRAY_SIZE(msm_blsp_configs));
```

For the latest details, refer to the Device Tree documentation:

`/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt`。

6.2.2 Quick verification

If you enter all the information properly, you should see the SPI bus registered under `/sys/class/spi_master/spi`, where cell-index matches the bus number.

```
adb shell --> Get adb shell
cd /sys/class/spi_master to list all the spi master
root@android:/sys/class/spi_master # ls
ls
spi0
spi6
spi7
```

6.2.2.1 Registering a slave device using the device tree

Once the SPI bus is registered, you can create a slave device driver and register it with the SPI master. For examples of SPI slave devices, refer to the following files:

```
/kernel/arch/arm/boot/dts/msm8226.dts
/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt
/kernel/Documentation/devicetree/bindings/spi/spi-bus.txt
```

The following example shows the minimum requirements for registering a slave device.

1. Create a Device Tree Node

   File to modify:

   ```
   /kernel/arch/arm/boot/dts/msm8226.dts
   ```

   Add a new Device Tree node:

   ```
   spi@f9923000 {
     qcom-spi-test@0 { //Slave driver and CS ID
       compatible = "qcom,spi-test"; //Manufacture, and Model
       reg = <0>; //Same as CS ID
       spi-max-frequency = <4800000>; //Max Frequency for Device
       /*
       * Following elements are optional.  For more detail respect to
       * SPI please see
   /kernel/Documentation/devicetree/bindings/spi/spi-bus.txt
       * For general settings respect to device tree look @ sdcc device
   tree
       */
       spi-cpol; //CPOL bit set for SPI polarity
       spi-cpha; //CPHA bit set for SPI Phase
       spi-cs-high; //CS Active High
       interrupt-parent = <&msmgpio>; //GPIO Handler
       interrupts = <61 0>; //GPIO # and flags
       qcom_spi_test,irq-gpio = <&msmgpio 61 0x00>; //Pass a GPIO
     };
   };
   ```

2. Sample Slave Device Driver

   ```
   #include <linux/module.h>
   #include <linux/init.h>
   #include <linux/delay.h>
   #include <linux/spi/spi.h>
   #include <linux/interrupt.h>
   #include <linux/slab.h>
   #include <linux/gpio.h>
   #include <linux/debugfs.h>
   ```

```
#include <linux/seq_file.h>
#include <linux/regulator/consumer.h>
#include <linux/string.h>
#include <linux/of_gpio.h>

#ifdef CONFIG_OF //Open firmware must be defined for dts useage
static struct of_device_id qcom_spi_test_table[] = {
{ .compatible = "qcom,spi-test",}, //Compatible node must match dts
{ },
};
#else
#define qcom_spi_test_table NULL
#endif


#define BUFFER_SIZE 4<<10
struct spi_message spi_msg;
struct spi_transfer spi_xfer;
u8 *tx_buf; //This needs to be DMA friendly buffer
static int spi_test_transfer(struct spi_device *spi)
{
spi_message_init(&spi_msg);

spi_xfer.tx_buf = tx_buf;
spi_xfer.len = BUFFER_SIZE;
spi_xfer.bits_per_word = 8;
spi_xfer.speed_hz = spi->max_speed_hz;

spi_message_add_tail(&spi_xfer, &spi_msg);

return spi_sync(spi, &spi_msg);
}


static int __devinit spi_test_probe(struct spi_device *spi)
{

int irq_gpio = -1;
int irq;
int cs;
int cpha,cpol,cs_high;
u32 max_speed;

dev_err(&spi->dev, "s\n", __func__);
```

```
1
2                //allocate memory for transfer
3           tx_buf = kmalloc(BUFFER_SIZE, GFP_ATOMIC);
4           if(tx_buf == NULL){
5                dev_err(&spi->dev, "s: mem alloc failed\n", __func__);
6                return -ENOMEM;
7           }
8           //Parse data using dt.
9           if(spi->dev.of_node){
10            irq_gpio = of_get_named_gpio_flags(spi->dev.of_node,
11          "qcom_spi_test,irq-gpio", 0, NULL);
12          }
13          irq = spi->irq;
14          cs = spi->chip_select;
15          cpha = ( spi->mode & SPI_CPHA ) ? 1:0;
16          cpol = ( spi->mode & SPI_CPOL ) ? 1:0;
17          cs_high = ( spi->mode & SPI_CS_HIGH ) ? 1:0;
18          max_speed = spi->max_speed_hz;
19          dev_err(&spi->dev, "gpio [d] irq [d] gpio_irq [d] cs [x] CPHA [x] CPOL
20          [x] CS_HIGH [x]\n",
21               irq_gpio, irq, gpio_to_irq(irq_gpio), cs, cpha, cpol, cs_high);
22
23          dev_err(&spi->dev, "Max_speed [d]\n", max_speed );
24
25          //Once you have a spi_device structure you can do a transfer anytime
26          spi->bits_per_word = 8;
27          dev_err(&spi->dev, "SPI sync returned [d]\n",  spi_test_transfer(spi));
28          return 0;
29          }
30
31
32          //SPI Driver Info
33          static struct spi_driver spi_test_driver = {
34          .driver = {
35               .name     = "qcom_spi_test",
36               .owner    = THIS_MODULE,
37               .of_match_table = qcom_spi_test_table,
38          },
39          .probe        = spi_test_probe,
40          };
41
42
43          static int __init spi_test_init(void)
44          {
45          return spi_register_driver(&spi_test_driver);
```

```
}

static void __exit spi_test_exit(void)
{
spi_unregister_driver(&spi_test_driver);
}


module_init(spi_test_init);
module_exit(spi_test_exit);
MODULE_DESCRIPTION("SPI TEST");
MODULE_LICENSE("GPL v2");
```

In the kernel log, the following message indicates the Device Tree was successfully configured.

```
<3>[    2.503571] qcom_spi_test spi6.0: spi_test_probe
<3>[    2.507305] qcom_spi_test spi6.0: gpio [61] irq [306] gpio_irq [306]
                       cs [0] CPHA [1] CPOL [1] CS_HIGH [1]
<3>[    2.516825] qcom_spi_test spi6.0: Max_speed [4800000]
<3>[    2.521932] qcom_spi_test spi6.0: SPI sync returned [0]
```

6.2.3 Optional: enable data mover mode (BAM)

The SPI can operate in Data Mover mode (BAM) or FIFO-based mode. If large amounts of data are to be transferred, Qualcomm recommends enabling BAM to offload the CPU.

6.2.3.1 BLSP BAM address and IRQ

Each BLSP core has a master BAM hardware block and one dedicated IRQ.

Table 6-2 BLSP BAM Physical Address

BLSP HW ID	QUP Cores	BLSP_BAM, IRQ
BLSP1	BLSP1_QUP[1:6]	0xF9904000,238

Table 6-3 BAM pipe assignment

BLSP HW ID	QUP Cores	PIPE(in, out)
BLSP1	BLSP1_QUP_1	12,13
BLSP1	BLSP1_QUP_2	14,15
BLSP1	BLSP1_QUP_3	16,17
BLSP1	BLSP1_QUP_4	18,19
BLSP1	BLSP1_QUP_5	20,21
BLSP1	BLSP1_QUP_6	22,23

Each BLSP QUP core has two unique pipes (Consumer and Producer) pre-allocated for data mover operations. The MSM8x26 chipset contains only one BLSP core.

6.2.3.2 Code Modification

This section describes how to enable BAM mode for the SPI.

1. Device Tree modification:

   The following additional settings to the Device Tree are required to enable BAM with SPI core:

   For latest detail, please follow

```
/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt
  //Add following additional nodes device tree to enable BAM
  spi@f9923000 {
    /*
    Modify the reg field as below to add BLSP BAM base address.
    First Row BLSP_QUP Base and Size always 0x1000
    Second Row is BAM address, size always 0x19000
    For BLSP1 QUP0:5 Bam Address = 0xF9904000
    */
    reg = <0xf9928000 0x1000>,
          <0xf9904000 0x19000>;
    reg-names = "spi_physical", "spi_bam_physical"; //Keep the same
names


    /*
    Replace the interrupt field.
    First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
    Second Field: Interrupt #
    Third field: Trigger type, keep 0
    For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
    First Row BLSP_QUP IRQ #
    Second Row is BAM IRQ
    For BLSP1 QUP0:5 IRQ = 238
    */
    interrupts = <0 100 0>, <0 238 0>;
    interrupt-names = "spi_irq", "spi_bam_irq"; //Keep same

    /*
    Add Consumer and Producer Pipes
    */
    qcom,use-bam; //enable BAM-mode
```

```
    qcom,bam-consumer-pipe-index = <22>; //Consumer PIPE from table
MSM8226 BLSP SPI Pipe Assignment
    qcom,bam-producer-pipe-index = <23>; //Producer PIPE from table
MSM8226 BLSP SPI Pipe Assignment
    qcom,ver-reg-exists; //Version register exists (True for 8226)
  };
```

2. TrustZone Changes:

   Each BLSP can be used by any subsystem such as Modem and LPASS. It is important to give ownership of the BAM pipes to applications processor.

   File to modify:
   `/trustzone_images/core/hwengines/bam/8226/bamtgtcfgdata_tz.h`

   Allocate BLSP BAM pipes with the applications core:

```
/*
  bam_tgt_blsp1_secconfig = BLSP1 Core
  Each Bit You set represent Pipe #.
    For example Bit 0 = Pipe # 0
                Bit 5 = Pipe # 5

  Any bit set on TZBSP_VMID_AP owns by APPs
*/
bam_sec_config_type bam_tgt_blsp1_secconfig =
{
    { //Since BLSP1_QUP1 is pipe 12, 13.  Bit 12, 13 should be set to
APPS
        {0x00003FFF, TZBSP_VMID_AP}, //Bit 11,12 Should be Set
        {0x00FFC000, TZBSP_VMID_LPASS } //Make sure Bit 12,13 is Cleared
    }
};
```

3. Debugging Tips

   If FIFO mode SPI works but BAM mode SPI does not work, ensure that the application processor can access the designated pipes.

   Once the system boots up, attach a JTAG to the RPM and CortexA7 cores (t32 cmd: sys.m.a).

   Break all the CortexA7 cores (t32 cmd: b).

   Turn on the BLSP_AHB_CLK by ensuring that bits 17 and 15 are set for the APCS_CLOCK_BRANCH_ENA_VOTE register (APCS_CLOCK_BRANCH_ENA_VOTE = 0xFC401484).

   - Ensure that BLSP_PIPE_TRUST_REG is set to zero for applications processor. (Read using Secure mode.)

```
D AZ:0xF9905030 -- BLSP1 pipe 12 \\
D AZ:0xF9906030 -- BLSP1 pipe 13
```

- Ensure that you can read/write to PIPE_CTRL_REG in Non-secure mode.

```
D.S A:0x0:0xF9905000 \LE \LONG 0xABCD -- Writing pipe 12 \\
D.S A:0x0:0xF9906000 \LE \LONG 0xABCD -- Writing pipe 13 \\
data.in A:0xF9905000 /long -- Reading pipe 12 \\
data.in A:0xF9906000 /long -- Reading pipe 23
```

If the above test failed, check the TrustZone modifications.

7 Camera

This chapter provides the information for OEM manufacturers to port the Camera module on MSM8x26 build. MSM8x26 has one VFE(4.0 lite) , and support both Bayer camera and SOC camera module. We will focus on how to port a Bayer camera to MSM8x26 platform.

7.1 Introduction

Device tree is introduced to MSM8x26 build to describe the hardware info instead of legacy kernel driver. There are several parts which need to be modified to port a new camera module to MSM 8x26 platform. It gives a guideline about how to bring up camera on a new hardware platform below. We assume to bring up sensor s5k3l1yx in this document.

The camera driver consists of a kernel part and a userspace part. The kernel part driver is shown as device tree source file and a kernel driver. The postfix "_lib" will be used for the userspace driver instead of "_u". To bring up a new sensor, we need to modify/add following files for sensor s5k3l1yx.

Kernelspace camera device tree source file:

```
/kernel/arch/arm/boot/dts/msm8226-camera-sensor-liquid.dtsi
```

Kernelspace camera driver file:

```
/kernel/drivers/media/video/msmb/sensor/s5k3l1yx.c
```

User space camera driver:

```
/vendor/qcom/proprietary/mm-camera/mm-camera2/media-
controller/modules/sensors/sensor_libs/s5k3l1yx/s5k3l1yx_lib.c
```

NOTE: Files below will not be used for MSM8x26 anymore.

```
/vendor/qcom/proprietary/mm-
camera/server/hardware/sensor/s5k3l1yx/s5k3l1yx_u.c
/vendor/qcom/proprietary/mm-
camera/server/hardware/sensor/s5k3l1yx/s5k3l1yx_u.h
```

7.2 Kernel porting

7.2.1 Device tree porting

Sensor-specific information is described in one of the following device tree files. The sensor information depends on the camera sensor used on your project:

`/arch/arm/boot/dts/msm8x26-camera-sensor.dtsi`

This file contains the basic sensor hardware information that needs to be defined, following is the sample code and description for each field.

Table 7-1 Camera properties

Properties	Description	Sample Codes	Suggestion for OEMs
compatible	should be "qcom" followed by sensor name	compatible = "qcom,s5k3l1yx";	Same as the sensor name used in the project
reg	should contain i2c slave address of the camera sensor and length of data field which is 0x0	reg = <0x6e 0x0>;	
qcom,slave-id	should contain i2c slave address, device id address and expected id read value	qcom,slave-id = <0x6e 0x0 0x3121>;	
qcom,csiphy-sd-index	should contain csiphy instance that will used to receive sensor data - 0, 1	qcom,csiphy-sd-index = <0>;	
qcom,csid-sd-index	should contain csid core instance that will used to receive sensor data - 0, 1	qcom,csid-sd-index = <0>;	
qcom,led-flash-sd-index	should contain phandle to flash source node if flash is supported for this sensor	qcom,flash-src-index = <&led_flash0>;	
qcom,mount-angle	should contain the physical mount angle of the sensor on the target - 0, 90, 180, 360	qcom,mount-angle = <0>;	
qcom,sensor-name	should contain unique sensor name to differentiate from other sensor	qcom,sensor-name = "s5k3l1yx";	Same as the compatible field.

Properties	Description	Sample Codes	Suggestion for OEMs
cam_vdig-supply	- cam_vdig-supply : should contain regulator from which digital voltage is supplied	cam_vdig-supply = <&pm8941_l3>;	The parameters are based on the HW design
cam_vana-supply	- cam_vana-supply : should contain regulator from which analog voltage is supplied	cam_vana-supply = <&pm8941_l17>;	
cam_vio-supply	- cam_vio-supply : should contain regulator from which IO voltage is supplied	cam_vio-supply = <&pm8941_lvs3>;	
cam_vaf-supply	- cam_vaf-supply : should contain regulator from which AF voltage is supplied	cam_vaf-supply = <&pm8941_l23>;	
qcom,cam-vreg-name	- qcom,cam-vreg-name : should contain names of all regulators needed by this sensor	qcom,cam-vreg-name = "cam_vdig", "cam_vio", "cam_vana", "cam_vaf";	
qcom,cam-vreg-type	- "cam_vdig", "cam_vana", "cam_vio", "cam_vaf" - qcom,cam-vreg-type : should contain regulator type for regulators mentioned in qcom,cam-vreg-name (in same order) - "cam_vdig", "cam_vana", "cam_vio", "cam_vaf" - 0 for LDO and 1 for LVS	qcom,cam-vreg-type = <0 1 0 0>;	
qcom,cam-vreg-min-voltage	- qcom,cam-vreg-min-voltage : should contain minimum voltage level for regulators mentioned in qcom,cam-vreg-name property (in the same order)	qcom,cam-vreg-min-voltage = <1225000 0 2850000 3000000>;	
qcom,cam-vreg-max-voltage	- qcom,cam-vreg-max-voltage : should contain maximum voltage level for regulators mentioned in qcom,cam-vreg-name property (in the same order)	qcom,cam-vreg-max-voltage = <1225000 0 2850000 3000000>;	
qcom,cam-vreg-op-voltage	- qcom,cam-vreg-op-mode : should contain optimum voltage level for regulators mentioned in qcom,cam-vreg-name property (in the same order)	qcom,cam-vreg-op-mode = <105000 0 80000 100000>;	

Properties	Description	Sample Codes	Suggestion for OEMs
qcom,gpio-no-mux	- qcom,gpio-no-mux : should contain field to indicate whether gpio mux table is available - 1 if gpio mux is not available, 0 otherwise	qcom,gpio-no-mux = <0>;	The parameters are based on the HW design, do not change unless really necessary.
gpios	- gpios : should contain phandle to gpio controller node and array of #gpio-cells specifying specific gpio (controller specific)	gpios = <&msmgpio 15 0>, <&msmgpio 90 0>;	
qcom,gpio-reset	- qcom,gpio-reset: should contain index to gpio used by sensors reset_n	qcom,gpio-reset = <1>;	
qcom,gpio-req-tbl-num	- qcom,gpio-req-tbl-num : should contain index to gpios specific to this sensor	qcom,gpio-req-tbl-num = <0 1>;	
qcom,gpio-req-tbl-flags	- qcom,gpio-req-tbl-flags : should contain direction of gpios present in qcom,gpio-req-tbl-num property (in the same order)	qcom,gpio-req-tbl-flags = <1 0>;	
qcom,gpio-req-tbl-label	- qcom,gpio-req-tbl-label : should contain name of gpios present in qcom,gpio-req-tbl-num property (in the same order)	qcom,gpio-req-tbl-label = "CAMIF_MCLK", "CAM_RESET1";	
qcom,gpio-set-tbl-num	- qcom,gpio-set-tbl-num : should contain index of gpios that need to be configured by msm	qcom,gpio-set-tbl-num = <1 1>;	
qcom,gpio-set-tbl-flags	- qcom,gpio-set-tbl-flags : should contain value to be configured for the gpios present in qcom,gpio-set-tbl-num property (in the same order)	qcom,gpio-set-tbl-flags = <0 2>;	
qcom,gpio-set-tbl-delay	- qcom,gpio-set-tbl-delay : should contain amount of delay after configuring gpios as specified in gpio_set_tbl_flags property (in the same order)	qcom,gpio-set-tbl-delay = <1000 30000>;	

Properties	Description	Sample Codes	Suggestion for OEMs
qcom,csi-lane-assign qcom,csi-lane-mask	- qcom,csi-lane-assign : should contain lane assignment value to map CSIPHY lanes to CSID lanes - qcom,csi-lane-mask : should contain lane mask that specifies CSIPHY lanes to be enabled	qcom,csi-lane-assign = <0x4320>; qcom,csi-lane-mask = <0x1F>;	
qcom,sensor-position	should contain the mount angle of the camera sensor - 0 -> back camera - 1 -> front camera	qcom,sensor-position = <0>;	
qcom,sensor-mode	should contain format of data that sensor streams - 0 -> bayer format - 1 -> yuv format	qcom,sensor-mode = <1>;	
status = "ok";	Camera module status	status = "ok";	Keep unchanged.

Following is a sample device tree source file for camera sensor s5k3l1yx.

```
&cci {
   /*6e is the I2C slave Id of the sensor*/
   qcom,camera@6e {
      compatible = "qcom,s5k3l1yx";
      reg = <0x6e 0x0>;
      qcom,slave-id = <0x6e 0x0 0x3121>;
      qcom,csiphy-sd-index = <0>;
      qcom,csid-sd-index = <0>;
      /*
      - led_flash0, led_flash1 which is defined in
./arch/arm/boot/dts/msm8226-camera.dtsi
      */
      qcom,flash-src-index = <&led_flash0>;

      qcom,mount-angle = <0>;
      qcom,sensor-name = "s5k3l1yx";
      /*
      pm8941_l3 is defined in ./arch/arm/boot/dts/msm8226-regulator.dtsi
      */
      cam_vdig-supply = <&pm8941_l3>;
      cam_vana-supply = <&pm8941_l17>;
```

```
            cam_vio-supply = <&pm8941_lvs3>;
            cam_vaf-supply = <&pm8941_l23>;
            qcom,cam-vreg-name = "cam_vdig", "cam_vio", "cam_vana",
                      "cam_vaf";
            qcom,cam-vreg-type = <0 1 0 0>;
            qcom,cam-vreg-min-voltage = <1225000 0 2850000 3000000>;
            qcom,cam-vreg-max-voltage = <1225000 0 2850000 3000000>;
            qcom,cam-vreg-op-mode = <105000 0 80000 100000>;
            qcom,gpio-no-mux = <0>;
            gpios = <&msmgpio 15 0>,
               <&msmgpio 90 0>;
            qcom,gpio-reset = <1>;
            qcom,gpio-req-tbl-num = <0 1>;
            qcom,gpio-req-tbl-flags = <1 0>;
            qcom,gpio-req-tbl-label = "CAMIF_MCLK",
                      "CAM_RESET1";
            qcom,gpio-set-tbl-num = <1 1>;
            qcom,gpio-set-tbl-flags = <0 2>;
            qcom,gpio-set-tbl-delay = <1000 30000>;
            qcom,csi-lane-assign = <0x4320>;
            qcom,csi-lane-mask = <0x1F>;
            qcom,sensor-position = <0>;
            qcom,sensor-mode = <1>;
            status = "ok";
         };

         qcom,camera@6c {
            compatible = "qcom,ov2720";
            …
         };
      };
```

NOTE: Detail document for camera device tree is in

`/kernel/Documentation/devicetree/bindings/media/video/msm-cci.txt`。

7.2.2 Kernel driver porting

Kernel driver for the sensor need to be modified based on the sensor capabilities:
`/kernel/drivers/media/video/msmb/sensor/s5k3l1yx.c`

The relationship between kernel driver and device tree source file (dtsi) file depends on the sensor name, so make sure have identical info between related fields.

```
#include "msm_sensor.h"
/*Sensor name must be same as the name in device tree file*/
#define S5K3L1YX_SENSOR_NAME "s5k3l1yx"
DEFINE_MSM_MUTEX(s5k3l1yx_mut);

static struct msm_sensor_ctrl_t s5k3l1yx_s_ctrl;

static struct msm_sensor_power_setting s5k3l1yx_power_setting[] = {
   {
      .seq_type = SENSOR_VREG,
      .seq_val = CAM_VDIG,
      .config_val = 0,
      .delay = 0,
   },
…
   {
      .order = 0,
   },
};

static const struct i2c_device_id s5k3l1yx_i2c_id[] = {
   {S5K3L1YX_SENSOR_NAME, (kernel_ulong_t)&s5k3l1yx_s_ctrl},
   { }
};

static struct i2c_driver s5k3l1yx_i2c_driver = {
   .id_table = s5k3l1yx_i2c_id,
   .probe = msm_sensor_i2c_probe,
   .driver = {
      .name = S5K3L1YX_SENSOR_NAME,
   },
};

static struct msm_camera_i2c_client s5k3l1yx_sensor_i2c_client = {
   .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
};
/*The compatible must be same as what is defined in device tree file*/
static const struct of_device_id s5k3l1yx_dt_match[] = {
   {.compatible = "qcom,s5k3l1yx", .data = &s5k3l1yx_s_ctrl},
   {}
};

MODULE_DEVICE_TABLE(of, s5k3l1yx_dt_match);
/*must match with the definition in device tree source file*/
```

```
static struct platform_driver s5k3l1yx_platform_driver = {
   .driver = {
      .name = "qcom,s5k3l1yx",
      .owner = THIS_MODULE,
      .of_match_table = s5k3l1yx_dt_match,
   },
};

static int32_t s5k3l1yx_platform_probe(struct platform_device *pdev)
{
   int32_t rc = 0;
   const struct of_device_id *match;
   match = of_match_device(s5k3l1yx_dt_match, &pdev->dev);
   rc = msm_sensor_platform_probe(pdev, match->data);
   return rc;
}

static int __init s5k3l1yx_init_module(void)
{
   int32_t rc = 0;
   pr_info("%s:%d\n", __func__, __LINE__);
   rc = platform_driver_probe(&s5k3l1yx_platform_driver,
      s5k3l1yx_platform_probe);
   if (!rc)
      return rc;
   pr_err("%s:%d rc %d\n", __func__, __LINE__, rc);
   return i2c_add_driver(&s5k3l1yx_i2c_driver);
}

static void __exit s5k3l1yx_exit_module(void)
{
   pr_info("%s:%d\n", __func__, __LINE__);
   if (s5k3l1yx_s_ctrl.pdev) {
      msm_sensor_free_sensor_data(&s5k3l1yx_s_ctrl);
      platform_driver_unregister(&s5k3l1yx_platform_driver);
   } else
      i2c_del_driver(&s5k3l1yx_i2c_driver);
   return;
}

static struct msm_sensor_ctrl_t s5k3l1yx_s_ctrl = {
   .sensor_i2c_client = &s5k3l1yx_sensor_i2c_client,
   .power_setting_array.power_setting = s5k3l1yx_power_setting,
   .power_setting_array.size = ARRAY_SIZE(s5k3l1yx_power_setting),
```

```
    .msm_sensor_mutex = &s5k3l1yx_mut,
    .sensor_v4l2_subdev_info = s5k3l1yx_subdev_info,
    .sensor_v4l2_subdev_info_size = ARRAY_SIZE(s5k3l1yx_subdev_info),
};

module_init(s5k3l1yx_init_module);
module_exit(s5k3l1yx_exit_module);
MODULE_DESCRIPTION("s5k3l1yx");
MODULE_LICENSE("GPL v2");
```

When kernel need to support a new sensor, some makefile or configuration should be modified, followings are the list of the compile options should be modified.

```
/drivers/media/platform/msm/camera_v2/Kconfig
…
config S5K3L1YX
  bool "Sensor S5K3L1YX (BAYER 12M)"
  depends on MSMB_CAMERA
  ---help---
    Samsung 12 MP Bayer Sensor with auto focus, uses
    4 mipi lanes, preview config = 1984 * 1508 at 30 fps,
    snapshot config = 4000 * 3000 at 20 fps,
    hfr video at 60, 90 and 120 fps.
…

/drivers/media/platform/msm/camera_v2/sensor/Makefile
…
 obj-$(CONFIG_MSMB_CAMERA) += cci/ io/ csiphy/ csid/
 obj-$(CONFIG_MSM_CAMERA_SENSOR) += msm_sensor.o
 obj-$(CONFIG_S5K3L1YX) += s5k3l1yx.o
 obj-$(CONFIG_OV2720) += ov2720.o
…

arch/arm/configs/msm8x26-perf_defconfig
arch/arm/configs/msm8x26_defconfig
…
 CONFIG_MSM_ISPIF=y
 CONFIG_S5K3L1YX=y
 CONFIG_IMX135=y
…
Also We need to provide clock configuration for the sensor
/arch/arm/mach-msm/clock-8x26.c
…
```

```
  CLK_LOOKUP("cam_src_clk", mclk0_clk_src.c, "6e.qcom,camera"),
  CLK_LOOKUP("cam_src_clk", mclk0_clk_src.c, "20.qcom,camera"),
  CLK_LOOKUP("cam_src_clk", mclk1_clk_src.c, "90.qcom,camera"),
  CLK_LOOKUP("cam_clk", camss_mclk0_clk.c, "6e.qcom,camera"),
  CLK_LOOKUP("cam_clk", camss_mclk0_clk.c, "20.qcom,camera"),
  CLK_LOOKUP("cam_clk", camss_mclk2_clk.c, "6c.qcom,camera"),
…
```

7.3 User space porting

For user space porting, following are the files that need to be changed.

`/vendor/qcom/proprietary/mm-camera/mm-camera2/media-controller/modules/sensors/sensor_libs/s5k3l1yx/s5k3l1yx_lib.c`

The only external interface of s5k3l1yx_lib.c is

```
void *s5k3l1yx_open_lib(void)
{
  return &sensor_lib_ptr;
}
```

This lib will be loaded when camera deamon process is started, kernel will find the handle based on the sensor name in dtb(device tree binary file), so please make sure the file name and function name follow the request for your sensor.

`<your_sensor_name>/<your_sensor_name>_lib.c`

`void *<your_sensor_name>_open_lib(void)`

Chromatix headers should also be updated for the sensor and put in the following folder:

`mm-camera2/media-controller/modules/sensors/chromatix/0301/libchromatix/chromatix_s5k3l1yx/common/`

`chromatix_s5k3l1yx_common.h`

`chromatix_s5k3l1yx_preview.h`

`chromatix_s5k3l1yx_video.h`

7.3.1 Porting power settings

The power setting of sensor contains the info of the power supply, IO control and Mclk for the sensor. Usually, we don’t need to modify this part, because most of the configuration is done in device tree file, but you can change the field “`.delay = 0`” to match the timing you want.

```
static struct msm_sensor_power_setting power_setting[] = {
  {
```

```
            .seq_type = SENSOR_VREG,
            .seq_val = CAM_VDIG,
            .config_val = 0,
            .delay = 0,
            .data = NULL,
          },
          {
            .seq_type = SENSOR_VREG,
            .seq_val = CAM_VANA,
            .config_val = 0,
            .delay = 0,
            .data = NULL,
          },
        …
        }
```

7.3.2 Porting camera Interface configuration

This part defines the interface configuration of sensor, which includes I2C, MIPI. Please make sure the settings are not conflict with the similar info that is mentioned in dtsi file.

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {
  /* sensor slave address */
  .slave_addr = 0x6E,
  /* sensor address type */
  .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
  /* sensor id info*/
  .sensor_id_info = {
    /* sensor id register address */
    .sensor_id_reg_addr = 0x0000,
    /* sensor id */
    .sensor_id = 0x3121,
  },
  /* power up / down setting */
  .power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
  },
};
static sensor_output_t sensor_output = {
  .output_format = SENSOR_BAYER,
  .connection_mode = SENSOR_MIPI_CSI,
  .raw_output = SENSOR_10_BIT_DIRECT,
};
static struct csi_lane_params_t csi_lane_params = {
```

```
  .csi_lane_assign = 0x4320,
  .csi_lane_mask = 0x1F,
  .csi_if = 1,
  .csid_core = {0},
};
```

7.3.3 Porting sensor output configuration

This part contains the configuration or settings for different sensor modes, which includes sensor I2C register settings, CSI parameters, crop parameters, sensor output info and chormatix array for each sensor mode. Those configuration should have the same array size as how many modes the sensor supports.

```
static struct msm_camera_i2c_reg_setting res_settings[] = {
  {
    /* res0_reg_array is the I2C settings from sensor vendor*/
    .reg_setting = res0_reg_array,
    .size = ARRAY_SIZE(res0_reg_array),
    .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
    .data_type = MSM_CAMERA_I2C_BYTE_DATA,
    .delay = 0,
  },
  …
};

static struct msm_camera_csi2_params *csi_params[] = {
  &s5k3l1yx_csi_params, /* csi param might be different for different
sensor mode*/
…
};

static struct sensor_crop_parms_t crop_params[] = {
  {0, 0, 0, 0}, /* All 0 means no crop */
…
};

static struct sensor_lib_out_info_t sensor_out_info[] = {
  {
    /* vt_pixel_clk  = .line_length_pclk* frame_length_lines*frame rate */
    /* op_pixel_clk  = VFE working clk */
    .x_output = 4000,
    .y_output = 3016,
    .line_length_pclk = 5336,
    .frame_length_lines = 3052,
```

```
            .vt_pixel_clk = 330000000,
            .op_pixel_clk = 264000000,
            .binning_factor = 1,
            .max_fps = 22.0,
            .min_fps = 22.0,
          },
        …
        };

        static struct sensor_lib_chromatix_t s5k3l1yx_chromatix[] = {
          {
            S5K3L1YX_LOAD_CHROMATIX(preview), /* RES0 */
            S5K3L1YX_LOAD_CHROMATIX(default_video), /* RES0 */
          },
          …
        };
```

7.3.4 Lens info porting

Lens info is needed in camera process flow, such as exposure calculation. So correct lens info should be filled to the following structures.

```
/*Lens info, filled based on lens and sensor spec */
static sensor_lens_info_t default_lens_info = {
  .focal_length = 1.15,
  .pix_size = 1.4,
  .f_number = 2.6,
  .total_f_dist = 1.5,
  .hor_view_angle = 54.8,
  .ver_view_angle = 42.5,
};
```

7.3.5 Exposure configuration porting

Usually different sensor has different method for exposure control. So after sensor is up, we need to port the exposure configuration part to bring up exposure control for the sensor.

Which includes the following functions:

- s5k3l1yx_real_to_register_gain – converts a real logical gain to the register value of sensor gain control register.
- s5k3l1yx_register_to_real_gain – converts a sensor gain register value to a real logical gain.
- s5k3l1yx_calculate_exposure – get next exposure configuration for both exposure time and gain.
- s5k3l1yx_fill_exposure_array – prepare the next exposure configure array.

```
/*===========================================================================
==
 * FUNCTION - s5k3l1yx_real_to_register_gain -
 *
 * DESCRIPTION:
 *===========================================================================
=*/
static uint16_t s5k3l1yx_real_to_register_gain(float gain)
{
  uint16_t reg_gain;

  if (gain < 1.0)
    gain = 1.0;

  if (gain > 16.0)
    gain = 16.0;

  reg_gain = (uint16_t)(gain * 32.0);

  return reg_gain;
}

/*===========================================================================
==
 * FUNCTION - s5k3l1yx_register_to_real_gain -
 *
 * DESCRIPTION:
 *===========================================================================
=*/
static float s5k3l1yx_register_to_real_gain(uint16_t reg_gain)
{
  float gain;

  if (reg_gain > 0x0200)
    reg_gain = 0x0200;

  gain = (float) reg_gain / 32.0;

  return gain;
}

/*===========================================================================
==
 * FUNCTION - s5k3l1yx_calculate_exposure -
 *
```

```
 * DESCRIPTION:
 *============================================================================
=*/
static int32_t s5k3l1yx_calculate_exposure(float real_gain,
  uint16_t line_count, sensor_exposure_info_t *exp_info)
{
  if (!exp_info) {
    return -1;
  }
  exp_info->reg_gain = s5k3l1yx_real_to_register_gain(real_gain);
  float sensor_real_gain = s5k3l1yx_register_to_real_gain(exp_info-
>reg_gain);
  exp_info->digital_gain = real_gain / sensor_real_gain;
  exp_info->line_count = line_count;
  return 0;
}

/*============================================================================
==
 * FUNCTION - s5k3l1yx_fill_exposure_array -
 *
 * DESCRIPTION:
 *============================================================================
=*/
static int32_t s5k3l1yx_fill_exposure_array(uint16_t gain, uint32_t line,
  uint32_t fl_lines, struct msm_camera_i2c_seq_reg_setting *reg_setting)
{
  reg_setting->reg_setting[0].reg_addr =
    sensor_lib_ptr.output_reg_addr->frame_length_lines;
  reg_setting->reg_setting[0].reg_data[0] = (fl_lines & 0xFF00) >> 8;
  reg_setting->reg_setting[0].reg_data[1] = (fl_lines & 0xFF);
  reg_setting->reg_setting[0].reg_data_size = 2;
  reg_setting->reg_setting[1].reg_addr =
    sensor_lib_ptr.exp_gain_info->coarse_int_time_addr;
  reg_setting->reg_setting[1].reg_data[0] = (line & 0xFF00) >> 8;
  reg_setting->reg_setting[1].reg_data[1] = (line & 0xFF);
  reg_setting->reg_setting[1].reg_data_size = 2;
  reg_setting->reg_setting[2].reg_addr =
    sensor_lib_ptr.exp_gain_info->global_gain_addr;
  reg_setting->reg_setting[2].reg_data[0] = (gain & 0xFF00) >> 8;
  reg_setting->reg_setting[2].reg_data[1] = (gain & 0xFF);
  reg_setting->reg_setting[2].reg_data_size = 2;
  reg_setting->size = 3;
  reg_setting->addr_type = MSM_CAMERA_I2C_WORD_ADDR;
  reg_setting->delay = 0;
```

```
  return 0;
}
```

7.4 Acuator porting

msm_actuator_ctrl_t contains all the information about the actuator setting related to sensors like i2c addr, set_info, focal length, etc. All the info is loaded from the chromatix file for the sensor.

Generic functions provided by msm_actuator.c:

/kernel/drivers/media/video/msm/actuators/actuator.c

HW description for Actuator is put in device tree:

/kernel/arch/arm/boot/dts/msm8x26-camera-sensor.dtsi

```
&cci {
  actuator0: qcom,actuator@18 {
    cell-index = <0>;
    reg = <0x18 0x0>;
    compatible = "qcom,actuator";
    qcom,cci-master = <0>;
  };
```

AF parameters will be loaded from hear files in Actuator.c

```
static actuator_ctrl_t actuators[] = {
#include "af_main_cam_0.h"
#include "af_main_cam_1.h"
#include "af_main_cam_2.h"
};
```

In af header file, the driver engineer should take care of the structure of :

actuator_params_t, which contains the af drive ic address, register patten, etc.

It is important for AF working.

```
  /* actuator_params_t */
    {
      /* i2c_addr */
      0xE4,
      /* i2c_data_type */
      MSM_ACTUATOR_BYTE_DATA,
      /* i2c_addr_type */
      MSM_ACTUATOR_BYTE_ADDR,
```

```
            /* act_type */
            ACTUATOR_PIEZO,
            /* data_size */
            8,
            /* af_restore_pos */
            0,
            /* msm_actuator_reg_tbl_t */
            {
              /* reg_tbl_size */
              1,
              /* msm_actuator_reg_params_t */
              {
                /* reg_write_type;hw_mask; reg_addr; hw_shift >>; data_shift <<
      */
                {MSM_ACTUATOR_WRITE_DAC, 0x00000080, 0x0000, 0, 0},
              },
            },
```

8 Display

8.1 Introduction

This chapter provides information for OEM manufacturers to port the Display Panel on MSM8626 build. MSM8626 just supports one display panel with MIPI DSI interface, both video mode and command mode are supported, and the maximum resolution is FWVGA(1280X800).

This chapter will focus on how to port DSI panel in Linux kernel. Please read panel vendors' specification for more details of the IC driver before you start to port your display panel.

8.2 Kernel porting

8.2.1 Properties

One big change/advantage in 8626 build is to use device tree to describe hardware, comparing to hardcoding every detail of a device into board specific files in previous chipset.

Display panel configuration is part of device trees as well, a binding is a description of how a device is described in the device tree. you can find Bindings for display panel at kernel\Documentation\devicetree\bindings\fb\mdss-dsi-panel.txt.

The below table describe those display properties one by one with suggestion to OEM manufacturers.

Table 8-1 Display properties

Properties	Description	Sample Codes	Suggestion for OEMs
compatible	Must be "qcom,mdss-dsi-panel"	compatible = "qcom,mdss-dsi-panel";	Keep unchanged
status	A string that has to be set to "okay/ok" to enable the panel driver. By default this property will be set to "disable". Will be set to "ok/okay" status for specific platforms.	status = "disable";	Set it to "disable" in panel dtsi, and set it to "ok" in MSM8x26 dtsi
qcom,dsi-ctrl-phandle	Specifies the phandle for the DSI controller that this panel will be mapped to.	qcom,dsi-ctrl-phandle = <&mdss_dsi0>;	Keep unchanged
qcom,mdss-pan-res	A two dimensional array that specifies the panel resolution.	qcom,mdss-pan-res = <720 1280>;	Customize it based on your panel specification
qcom,mdss-pan-bpp	Specifies the panel bits per pixel. Default value is 24(rgb888). 18 = for rgb666 and 16 = for rgb565	qcom,mdss-pan-bpp = <24>;	Customize it based on your panel specification

qcom,panel-phy-regulatorSettings	An array of length 7 that specifies the PHY regulator settings for the panel.	qcom,panel-phy-regulatorSettings = [03 01 01 00 20 00 01]; /* Regulator settings */	Keep unchanged
qcom,panel-phy-timingSettings	An array of length 12 that specifies the PHY timing settings for the panel.	qcom,panel-phy-timingSettings = [69 29 1f 00 55 55 19 2a 2a 03 04 00];	Keep unchanged when you try to bring up this panel at the beginng, and create case for help for fine tuning parameters if the panel failed to bring up
qcom,panel-phy-strengthCtrl	An array of length 2 that specifies the PHY strengthCtrl settings for the panel.	qcom,panel-phy-strengthCtrl = [77 06];	Keep unchanged
qcom,panel-phy-bistCtrl	An array of length 6 that specifies the PHY BIST ctrl settings for the panel.	qcom,panel-phy-bistCtrl = [00 00 b1 ff 00 00]; /* BIST Ctrl settings */	Keep unchanged
qcom,panel-phy-laneConfig	An array of length 45 that specifies the PHY lane configuration settings for the panel.	qcom,panel-phy-laneConfig = [00 c2 45 00 00 00 00 01 75 /* lane0 config */ 00 c2 45 00 00 00 00 01 75 /* lane1 config */ 00 c2 45 00 00 00 00 01 75 /* lane2 config */ 00 c2 45 00 00 00 00 01 75 /* lane3 config */ 00 02 45 00 00 00 00 01 97]; /* Clk ln config */	Keep unchanged
qcom,mdss-panel-on-cmds	An array of variable length that lists the init commands of the panel. Please refer dsi_cmd_desc structure definition at \kernel\drivers\video\msm\mdss\mdss_dsi.h for the format.	qcom,panel-on-cmds = [23 01 00 00 0a 02 b0 00 23 01 00 00 0a 02 b2 00];	Customize it based on your panel specification
qcom,mdss-panel-off-cmds	An array of variable length that lists the panel off commands.	qcom,panel-off-cmds = [05 01 00 00 32 02 28 00 05 01 00 00 78 02 10 00];	customize it based on your panel specification
label	A string used as a descriptive name of the panel.	label = "toshiba 720p video mode dsi panel";	Customize it based on your panel specification
qcom,enable-gpio	Specifies the panel lcd/display enable gpio.	qcom,enable-gpio = <&msmgpio 58 0>;	Customize it based on your panel specification, you may need change mdss_dsi_panel_reset() in mdss_dsi_panel.c
qcom,rst-gpio	Specifies the panel reset gpio.	qcom,rst-gpio = <&pm8941_gpios 19	Customize it based on your panel

		0>;	specification
qcom,mdss-pan-porch-values	An array of size 6 that specifies the panel blanking values.	qcom,mdss-pan-porch-values = <32 12 144 3 4 9>;	Customize it based on your panel specification
qcom,mdss-pan-underflow-clr	Specifies the controller settings for the panel underflow clear settings. Default value is 0xff.	qcom,mdss-pan-underflow-clr = <0xff>;	Keep unchanged
qcom,mdss-pan-bl-ctrl	A string that specifies the implementation of backlight control for this panel. "bl_ctrl_pwm" = Backlight controlled by PWM gpio. "bl_ctrl_wled" = Backlight controlled by WLED. "bl_ctrl_dcs_cmds" = Backlight controlled by DCS commands.	qcom,mdss-pan-bl-ctrl = "bl_ctrl_wled";	Customize it based on your panel specification
qcom,mdss-pan-bl-levels	Specifies the backlight levels supported by the panel. Default range is 1 to 255.	qcom,mdss-pan-bl-levels = <1 255>;	Keep unchanged
qcom,mdss-pan-dsi-mode	Specifies the panel operating mode. 0 = enable video mode(default mode). 1 = enable command mode.	qcom,mdss-pan-dsi-mode = <0>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-h-pulse-mode	Specifies the pulse mode option for the panel. 0 = Don't send hsa/he following vs/ve packet(default) 1 = Send hsa/he following vs/ve packet	qcom,mdss-pan-dsi-h-pulse-mode = <0>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-h-power-stop	An Array of size 3 that specifies the power mode during horizontal porch and sync periods of the panel. 0 = high speed mode(default mode). 1 = Low power mode for horizontal porches and sync pulse.	qcom,mdss-pan-dsi-h-power-stop = <0 0 0>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-bllp-power-stop	An Array of size 2 that specifies the power mode during blanking period and after EOF(end of frame). 0 = high speed mode(default mode). 1 = Low power mode during blanking and EOF.	qcom,mdss-pan-dsi-bllp-power-stop = <1 1>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-traffic-mode	Specifies the panel traffic mode. 0 = non burst with sync pulses (default mode). 1 = non burst with sync start event. 2 = burst mode.	qcom,mdss-pan-dsi-traffic-mode = <1>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-dst-format	Specifies the destination format. 0 = DSI_VIDEO_DST_FORMAT_RGB565. 1 = DSI_VIDEO_DST_FORMAT_RGB666. 2 = DSI_VIDEO_DST_FORMAT_RGB666_LOOSE. 3 = DSI_VIDEO_DST_FORMAT_RGB888 (Default format) 6 = DSI_CMD_DST_FORMAT_RGB565 7 = DSI_CMD_DST_FORMAT_RGB666 8 = DSI_CMD_DST_FORMAT_RGB888	qcom,mdss-pan-dsi-dst-format = <3>;	Customize it based on your panel specification

qcom,mdss-pan-dsi-vc	Specifies the virtual channel identefier. 0 = default value.	qcom,mdss-pan-dsi-vc = <0>;	Keep unchanged
qcom,mdss-pan-dsi-rgb-swap	Specifies the R, G and B channel ordering. 0 = DSI_RGB_SWAP_RGB (default value) 1 = DSI_RGB_SWAP_RBG 2 = DSI_RGB_SWAP_BGR 3 = DSI_RGB_SWAP_BRG 4 = DSI_RGB_SWAP_GRB 5 = DSI_RGB_SWAP_GBR	qcom,mdss-pan-dsi-rgb-swap = <0>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-data-lanes	An array that specifies the data lanes enabled. <1 1 0 0> = data lanes 1 and 2 are enabled.(default).	qcom,mdss-pan-dsi-data-lanes = <1 1 1 1>;	Customize it based on your panel specification
qcom,mdss-pan-dsi-t-clk	An array that specifies the byte clock cycles before and after each mode switch.	qcom,mdss-pan-dsi-t-clk = <0x1b 0x04>;	Keep unchanged when you try to bring up this panel at the beginng, and create case for help for fine tuning parameters if the panel failed to bring up
qcom,mdss-pan-dsi-stream	Specifies the packet stream to be used. 0 = stream 0 (default) 1 = stream 1	qcom,mdss-pan-dsi-stream = <0>;	Keep unchanged
qcom,mdss-pan-dsi-mdp-tr	Specifies the trigger mechanism to be used for MDP path. 0 = no trigger 2 = Tear check signal line used for trigger 4 = Triggered by software (default mode) 6 = Software trigger and TE	qcom,mdss-pan-dsi-mdp-tr = <0x0>;	Keep unchanged
qcom,mdss-pan-dsi-dma-tr	Specifies the trigger mechanism to be used for DMA path. 0 = no trigger 2 = Tear check signal line used for trigger 4 = Triggered by software (default mode) 5 = Software trigger and start/end of frame trigger. 6 = Software trigger and TE	qcom,mdss-pan-dsi-dma-tr = <0x04>;	Keep unchanged
qcom,mdss-pan-dsi-frame-rate	Specifies the frame rate for the panel. 60 = 60 frames per second (default)	qcom,mdss-pan-frame-rate = <60>;	Customize it based on your panel specification

8.2.2 Sample

Both CDP8x26 and QRD8x26 phone adopt Truly NT35590 720p panel , the reference display configuration files is `kernel\arch\arm\boot\dts\ dsi-panel-nt35590-720p-video.dtsi`, the sample codes are copied below.

```
/ {
        qcom,mdss_dsi_nt35590_720p_video {
```

```
            compatible = "qcom,mdss-dsi-panel";
            label = "nt35590 720p video mode dsi panel";
            status = "disable";
            qcom,dsi-ctrl-phandle = <&mdss_dsi0>;
            qcom,rst-gpio = <&msmgpio 25 0>;
            qcom,mdss-pan-res = <720 1280>;
            qcom,mdss-pan-bpp = <24>;
            qcom,mdss-pan-dest = "display_1";
            qcom,mdss-pan-porch-values = <164 8 140 1 1 6>;
            qcom,mdss-pan-underflow-clr = <0xff>;
            qcom,mdss-pan-bl-ctrl = "bl_ctrl_wled";
            qcom,mdss-pan-bl-levels = <1 255>;
            qcom,mdss-pan-dsi-mode = <0>;
            qcom,mdss-pan-dsi-h-pulse-mode = <1>;
            qcom,mdss-pan-dsi-h-power-stop = <0 0 0>;
            qcom,mdss-pan-dsi-bllp-power-stop = <1 1>;
            qcom,mdss-pan-dsi-traffic-mode = <2>;
            qcom,mdss-pan-dsi-dst-format = <3>;
            qcom,mdss-pan-dsi-vc = <0>;
            qcom,mdss-pan-dsi-rgb-swap = <0>;
            qcom,mdss-pan-dsi-data-lanes = <1 1 1 1>; /* 4 lanes */
            qcom,mdss-pan-dsi-dlane-swap = <0>;
            qcom,mdss-pan-dsi-t-clk = <0x2c 0x20>;
            qcom,mdss-pan-dsi-stream = <0>;
            qcom,mdss-pan-dsi-mdp-tr = <0x0>;
            qcom,mdss-pan-dsi-dma-tr = <0x04>;
            qcom,mdss-pan-frame-rate = <60>;
            qcom,panel-phy-regulatorSettings = [07 09 03 00  /* Regualotor
        settings */
                                        20 00 01];
            qcom,panel-phy-timingSettings = [7d 25 1d 00 37 33
                                        22 27 1e 03 04 00];
            qcom,panel-phy-strengthCtrl = [ff 06];
            qcom,panel-phy-bistCtrl = [00 00 b1 ff           /* BIST Ctrl
        settings */
                                    00 00];
            qcom,panel-phy-laneConfig = [00 00 00 00 00 00 00 01 97 /* lane0
        config */
                                      00 00 00 00 05 00 00 01 97 /* lane1 config */
                                      00 00 00 00 0a 00 00 01 97 /* lane2 config */
                                      00 00 00 00 0f 00 00 01 97 /* lane3 config */
                                      00 c0 00 00 00 00 00 01 bb]; /* Clk ln config */
            qcom,panel-on-cmds = [29 01 00 00 00 02 FF EE
                                  29 01 00 00 00 02 26 08
                                  29 01 00 00 00 02 26 00
```

```
                        29 01 00 00 00 02 FF 00
                        29 01 00 00 78 02 29 00];

        qcom,on-cmds-dsi-state = "DSI_LP_MODE";
        qcom,panel-off-cmds = [05 01 00 00 32 02 28 00
                        05 01 00 00 78 02 10 00];
        qcom,off-cmds-dsi-state = "DSI_HS_MODE";
    };
};
```

The display configuration is included into super MSM8x26 device tree with the following codes snippets copied from msm8226-qrd.dts.

```
/include/ "dsi-panel-nt35590-720p-video.dtsi"

/ {
    model = "Qualcomm MSM 8226 QRD";
    compatible = "qcom,msm8226-qrd", "qcom,msm8226";
    qcom,msm-id = <145 11 0>;

    serial@f991f000 {
        status = "ok";
    };

    qcom,mdss_dsi_nt35590_720p_video {
        status = "ok";
    };
};
```

8.2.3 Step by step

1. Create your own panel configuration device tree in kernel\arch\arm\boot\dts\ folder, for example, dsi-panel-<vendor>-<res>-video.dtsi.
2. Copy the content from dsi-panel-nt35590-720p-video.dtsi into your own panel device tree file, dsi-panel-<vendor>-<res>-video.dtsi.
3. Modify the panel vendor information, such as.

```
qcom,mdss_dsi_<vendor>_<res>_video {
    compatible = "qcom,mdss-dsi-panel";
    label = "<vendor> <res> video mode dsi panel";
```

4. Modify the panel basic information, such as, panel resolution, bpp, based on your panel specification.

```
qcom,mdss-pan-res = <720 1280>;
```

```
qcom,mdss-pan-bpp = <24>;
```

5. Modify the panel power on and power off command sequence, based on your panel specification.

```
qcom,panel-on-cmds = [05 01 00 00 78 02 11 00
                      05 01 00 00 78 02 29 00];
qcom,on-cmds-dsi-state = "DSI_LP_MODE";
qcom,panel-off-cmds = [05 01 00 00 32 02 28 00
                      05 01 00 00 78 02 10 00];
qcom,off-cmds-dsi-state = "DSI_LP_MODE";
```

6. Modify the panel porch value based on your panel specification.

```
qcom,mdss-pan-porch-values = <32 12 144 3 4 9>;
```

7. Modify the panel gpio configuration based on your hardware schematics, you may need change

```
qcom,rst-gpio = <&msmgpio 25 0>;
```

   we use GPIO25 for panel reset, if you change it to other GPIO, you need change arch/arm/mach-msm/board-8226-gpiomux.c as well.

8. Add your own panel device tree into MSM8626 main device tree.

```
/include/ "dsi-panel-<vendor>-<res>-video.dtsi"
…… ……
    qcom,mdss_dsi@fd922800 {
        qcom,mdss_dsi_<vendor>_<res>_video {
            status = "ok";
        };
};
```

9. Rebuild and load your boot image, and check whether the panel can be lighted, if not, please refer Q2 to debug the panel driver further.

1 简介

1.1 目的

本文档描述了设备树及其用法、GPIO 及示例代码、I2C 总线及示例代码、SPI 总线及示例代码、相机架构及示例代码、显示屏架构及示例代码。

1.2 范围

本文档适用于在 8x26 平台进行开发调试的工程师。

1.3 约定

函数声明，函数名称，类型声明以及代码举例需要使用不同的字体，例如`#include`。

代码变量应出现在角括号中，例如`<number>`。

命令在输入时应使用不同的字体，例如 **`copy a:*.* b:`**。

按钮和关键字应使用粗体字体，例如 click **Save** or press **Enter**。

如果你正在使用彩色显示器查看此文件，或使用彩色打印机打印文档时，红色字体显示的数据类型，蓝色字体显示属性，而绿色字体显示系统属性。

参数类型是用箭头表示：

→ 指定一个输入参数

← 指定一个输出参数

↔ 指定一个用于输入和输出参数

1.4 参考

参考文件，其中可能包括高通的文档、标准和资源，具体列在 Table 1-1。对于不再适用参考的文档已从本表中删除，因此，参考文档的引用顺序可能不连续。

Table 1-1 参考文档和标准

编号	文档	
高通文档		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*MSM8x26 Linux Kernel BSP and Device Drivers Overview*	80-ND928-25
Q3	*MSM8x26 Linux Android Software Architecture Overview*	80-ND928-23
Q4	*MSM8x26 Software Interface*	80-NC832-2

编号	文档	
Q5	*Linux Device Tree MSM8x26 Android SW*	80-NA157-93

1.5 技术支持

如果您想得到关于本指南的支持细节，请登陆至 https://support.cdmatech.com/。

如果您无法登陆 CDMA 技术支持服务网站，请注册或发送电子邮件至 support.cdmatech@qualcomm.com。

1.6 缩写词

术语和缩略词的定义请参考文档[Q1]。

2 设备树

2.1 简介

设备树可通过在简单的文件（dts）中声明节点和属性来描绘系统设备，这样使得嵌板供应商的工作更容易。并且，设备树也适用于不符合现有 OF 规范的芯片设备和其他总线。

同时，内核可以更灵活的进行探测并能匹配设备和驱动，而无需硬编码各类表格。此外，对于单板供应商来说，可以进行硬件设备的细微升级而无需明显影响或扰乱内核代码。

使用设备树有利于：

- 减少移植到新的单板所需的工作
- 进行代码复用，通过使驱动器足够通用来支持一组便携设备，并且可呈现设备区别

更多细节，请参考[Q5]。设置 CONFIG_USE_OF 为 y，启用设备树。其中，OF 表示打开固件。

2.1.1 DTC

使用 dtc (设备树编译器) 将设备树脚本解析成二进制格式。

如果内核没有变化，可使用以下命令生成 dtb zImage：

```
dtc -p 1024 -O dtb -o msm8626.dtb msm8626.dts
cat zImage msm.dtb > dtb-zimage
```

2.1.2 Bootloader

Fastboot 可解析引导映像并提取设备树信息。

DT address = image address + page_size + kernel_actual + ramdisk_actual + second_actual

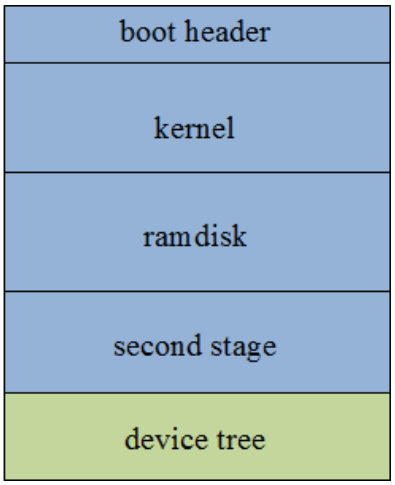

Figure 2-1 引导映像中的设备树

Fastboot 使用和 atags 同样的方式将 dt table 传给内核。内核仍通过 r2 保存 dt table ptr。

Table 2-1 LK 中的注册表

寄存器	ATAGS	DT
R0	Null	Null
R1	机器型号	当 bootloader 传递正确的设备树时，不再需要唯一的 machine ID。在 register r1，传入内核的 ID 应为 0xFFFFFFFF
R2	系统 RAM 中标记列表的物理地址	RAM 中设备树块（在第二章中定义）的物理指针。 设备树可位于系统 RAM 中的任何地方，但其应该在 64 位边界上对齐。

源代码路径为：`kernel/Documentation/arm/Booting`。

2.1.3 内核

2.1.3.1 相关代码

```
kernel/arch/arm/kernel/devtree.c
kernel/drivers/of/device.c
include/linux/of_gpio.h
```

2.1.3.2 主要函数

Table 2-2 设备树主要 API

函数	描述
`of_platform_populate()`	系统 init 函数，可在 kernel/arch/arm/mach-msm/board-8226.c 中查看。该函数会检查通过 fastboot 传递的 dt table。

of_match_device()	是驱动器重要的探测函数，应在驱动中进行声明。
	例如：
	`static const struct of_device_id s5k3l1yx_dt_match[] = {`
	`{.compatible = "qcom,s5k3l1yx", .data = &s5k3l1yx_s_ctrl},`
	`{}`
	`};`
	
	`static struct platform_driver s5k3l1yx_platform_driver = {`
	`.driver = {`
	`.name = "qcom,s5k3l1yx",`
	`.owner = THIS_MODULE,`
	`.of_match_table = s5k3l1yx_dt_match,`
	`},`
	`};`
	
	`static int32_t s5k3l1yx_platform_probe(struct platform_device *pdev)`
	`{`
	`int32_t rc = 0;`
	`const struct of_device_id *match;`
	`match = of_match_device(s5k3l1yx_dt_match, &pdev->dev);`
	`rc = msm_sensor_platform_probe(pdev, match->data);`
	`return rc;`
	`}`

1

3 GPIO

本章介绍了顶级模式复用器(TLMM)GPIO 并描述了如何在 bootloader 及内核中对其进行配置。

3.1 硬件概述

8x26 有 117 个 GPIO。

3.1.1 GPIO配置

本节描述了 MSM8x26 芯片集 GPIO 的配置寄存器。

Table 3-1 GPIO 物理地址

硬件寄存器	物理地址	复位状态
TLMM_GPIO_CFGn, n =[0..116]	0xFD511000+0x10*(n)	0x00000001
TLMM_GPIO_IN_OUTn , n=[0..116]	0xFD511004+0x10*(n)	0x00000000
TLMM_GPIO_OE_0	0xFD513080	0x00000000
TLMM_GPIO_OE_1	0xFD513084	0x00000000
TLMM_GPIO_OE_2	0xFD513088	0x00000000
TLMM_GPIO_OE_3	0xFD51308C	0x00000000

Table 3-2 GPIO CFG 表

Bits	字段	描述
10	GPIO_HIHYS_EN	控制 hihys_en
9	GPIO_OE	在 GPIO 模式下，控制 OE
8:6	DRV_STRENGTH	控制GPIO pad驱动强度。并且不管如何选择FUNC_SEL字段，都适用。 0: DRV_2_MA 1: DRV_4_MA 2: DRV_6_MA 3: DRV_8_MA 4: DRV_10_MA 5: DRV_12_MA 6: DRV_14_MA 7: DRV_16_MA
5:2	FUNC_SEL	很多GPIO pad后背有一个或多个功能硬件接口。该字段控制了如何使用pad。因此，需将其设为函数所需值。
1:0	GPIO_PULL	可通过配置该pad启用内部弱pull up、pull down或keeper函数。并且不管如何选择FUNC_SEL字段，

		都适用。 0x0: NO_PULL (Disables all pull) 0x1: PULL_DOWN (Weak Pull-down) 0x2: KEEPER (Weak Keeper) 0x3: PULL_UP (Weak Pull-Up)

NOTE: 在 CFG 寄存器中，GPIO_OE 能启用 GPIO 配置。无需写入 TLMM_GPIO_OE_[0…3]。

NOTE: 在 CFG 寄存器中，GPIO_HIHYS_EN 能启用输入模式的 GPIO 滞后，其主要适用于噪声低频信号，如睡眠时钟等。

```
arch/arm/mach-msm/gpiomux-v2.c

void __msm_gpiomux_write(unsigned gpio, struct gpiomux_setting val)
{
   uint32_t bits;

   bits = (val.drv << 6) | (val.func << 2) | val.pull;
   if (val.func == GPIOMUX_FUNC_GPIO) {
        bits |= val.dir > GPIOMUX_IN ? BIT(9) : 0;
        __raw_writel(val.dir == GPIOMUX_OUT_HIGH ? BIT(1) : 0,
             GPIO_IN_OUT(gpio));
   }
   __raw_writel(bits, GPIO_CFG(gpio));
   mb();
}
```

3.1.2 中断配置

Table 3-3 GPIO 中断 CFG 表

硬件寄存器	物理地址	复位状态
TLMM_GPIO_INTR_CFGn , n=[0..116]	0xFD511008 + 0x10 * (n)	0x000000E2
TLMM_GPIO_INTR_STATUSn , n=[0..116]	0xFD51100C + 0x10 * (n)	0x00000000

Table 3-4 INT CFG

Bits	字段	描述
8	DIR_CONN_EN	0: DISABLE 1: ENABLE
7:05	TARGET_PROC	0: WCSS 1: SENSORS 2: LPA_DSP 3: RPM 4: KPSS 5: MSS

		6: TZ 7: NONE
4	INTR_RAW_STATUS_EN	0: DISABLE 1: ENABLE
3:02	INTR_DECT_CTL	0: LEVEL 1: POS_EDGE 2: NEG_EDGE 3: DUAL_EDGE
1	INTR_POL_CTL	0: POLARITY_0 1: POLARITY_1
0	INTR_ENABLE	0: DISABLE 1: ENABLE

NOTE: POS_EDGE 表示正缘触发

NEG_EDGE 表示负缘触发

目前支持正缘触发和负缘触发。

Table 3-5 GPIO 唤醒中断

硬件寄存器	物理地址	复位状态
TLMM_MPM_WAKEUP_INT_EN_0	0xFD512008	0x00000000
TLMM_MPM_WAKEUP_INT_EN_1	0xFD51200C	0x00000000

NOTE: Group 0:
72,71,69,68,67,66,65,64,63,62,54,52,51,50,49,48,46,41,39,37,35,33,31,29,27,21,17,13,9 ,5 ,4 ,1

NOTE: Group 1:

SDC2_DATA_3, SDC2_DATA_1, SDC1_DATA_3,
SDC1_DATA_1,115,113,111,110,109,108,107,106,38,

3.2 软件概述

3.2.1 在bootloader中配置GPIO

可通过调用 gpio_tlmm_config()来配置 GPIO。

bootable/bootloader/lk/platform/msm8226/gpio.c

```
void gpio_tlmm_config(uint32_t gpio, uint8_t func,
                      uint8_t dir, uint8_t pull,
                      uint8_t drvstr, uint32_t enable)
{
   uint32_t val = 0;
```

```
    val |= pull;
    val |= func << 2;
    val |= drvstr << 6;
    val |= enable << 9;
    writel(val, (uint32_t *)GPIO_CONFIG_ADDR(gpio));
    return;
}
```

3.2.2 在内核配置GPIO

本节描述了配置及使用 MSM8x26 芯片集 GPIO 的步骤。

更多详细信息，请参考 kernel/arch/arm/boot/dts/msm8226.dts 和 kernel/Documentation/devicetree/bindings/gpio/gpio-msm.txt。

1. 设备树中 MSM GPIO 定义如下：

```
msmgpio: gpio@fd510000 {
     compatible = "qcom,msm-gpio";
     interrupt-controller;
     #interrupt-cells = <2>;
     reg = <0xfd510000 0x4000>;
     gpio-controller;
     #gpio-cells = <2>;
     ngpio = <117>;
     interrupts = <0 208 0>;
     qcom,direct-connect-irqs = <8>;
};
```

2. 添加设备树节点，说明设备的 GPIO：

```
device1@sample {
     compatible = "sample-device1";
     reg = <0xf991f000 0x1000>;
     gpios = <&msmgpio 45 0>;
     cs-gpios = <&msmgpio 46 0>;
};
```

3. 在 device1 的驱动代码中，通过调用 of_get_gpio()可获得 GPIO45/46 句柄，通过调用以 cs-gpios 作为名称参数的 of_get_named_gpio()可获得 GPIO46 句柄。

```
static int __devinit device1_probe(struct device_node *np,
                struct device1_platform_data *pdata)
```

```
{
u32 reg;
u32 gpio1;
u32 gpio2;

if (of_gpio_count(np) < 2)
     return -ENODEV;
gpio1 = of_get_gpio(np, 0);
gpio2 = of_get_gpio(np, 1);
gpio2 = of_get_named_gpio(np, “cs-gpios”,0);
gpio_request(gpio1, “gpio1”);
gpio_direction_output(gpio1,0);
gpio_request(gpio2, “gpio2”);
gpio_direction_input(gpio2);
...
              }
```

4. 添加设备树节点，说明设备的 GPIO 中断：

```
device2@sample {
     compatible = " sample-device2";
     reg = <0xf991f000 0x1000>;
     interrupt-parent = <&msmgpio>    ;//use msmgpio as interrupt-parent
     interrupts = <17 0x2>;//offset and trigger mode
     irq-gpio = <&msmgpio 17 0>;
};
```

5. 在 device2 驱动中：

```
static int __devinit device2_probe(struct device_node *np,
               struct device1_platform_data *pdata)
{
 u32 reg;
 u32 irq-gpio;
 int irq;

 if (of_gpio_count(np) < 1)
     return -ENODEV;

 irq-gpio = of_get_gpio(np, 0);
 gpio_request(irq-gpio, “irq-gpio”);
 gpio_direction_input(irq-gpio);
 irq = gpio_to_irq(irq-gpio);
```

```
request_threaded_irq(irq,  NULL, sample_irq_handler, 0x0, DRIVER_NAME,
NULL);
...
            }
```

3.2.3 步骤

检查 kernel/msm-3.4/drivers/i2c/busses/i2c-gpio.c。

1. 获取 GPIO 信息：

   pdata->sda_pin = of_get_gpio(np, 0);

   pdata->scl_pin = of_get_gpio(np, 1);

2. 获取 properties：

```
of_property_read_u32(np, "i2c-gpio,delay-us", &pdata->udelay)
of_property_read_u32(np, "i2c-gpio,timeout-ms", &reg)
of_property_read_bool(np, "i2c-gpio,sda-open-drain")
of_property_read_bool(np, "i2c-gpio,scl-open-drain")
of_property_read_bool(np, "i2c-gpio,scl-output-only")
```

3. 获取 of_device_id：

```
static const struct of_device_id i2c_gpio_dt_ids[] = {
{ .compatible = "i2c-gpio", },
{ /* sentinel */ }
};
```

4. 添加 dts：

   需使用唯一的名称。比如这里将设备名设为“i2c-gpio@sample”。根据 of_device_id i2c_gpio_dt_ids[]中的值，推断 compatible 为 i2c-gpio。共有两个 GPIO，在这种情况下，使用 GPIO45 和 GPIO46 作为例子，45 作为 sda， 46 作为 sda；根据需求设置属性（clk 速率为 20K，超时值为 100ms）。

```
Set i2c-gpio,delay-us as 50
Set i2c-gpio,timeout-ms as 100
Set i2c-gpio,sda-open-drain as 1
Set i2c-gpio,scl-open-drain as 1
Set i2c-gpio,scl-output-only as 1
```

   总结为：

```
i2c-gpio@sample {
     compatible = "i2c-gpio"
     gpios = <&msmgpio 45 0
                 &msmgpio 46 0>;
     i2c-gpio,delay-us = <50>;
     i2c-gpio,timeout-ms = <100>;
     i2c-gpio,sda-open-drain = <1>;
     i2c-gpio,scl-open-drain = <1>;
     i2c-gpio,scl-output-only = <1>;
};
```

如果使用 of_get_named_gpio()而不是 of_get_gpio()，比如：

```
pdata->sda_pin = of_get_named_gpio(np, "sample,i2c-sda",0);
pdata->scl_pin = of_get_named_gpio(np, "sample,i2c-scl",0);
```

dts 应该是：

```
i2c-gpio@sample {
     compatible = "i2c-gpio"
     sample,i2c-sda = <&msmgpio 45 0>;
        sample,i2c-scl = <&msmgpio 46 0>;
     i2c-gpio,delay-us = <50>;
     i2c-gpio,timeout-ms = <100>;
     i2c-gpio,sda-open-drain = <1>;
     i2c-gpio,scl-open-drain = <1>;
     i2c-gpio,scl-output-only = <1>;
};
```

4 I2C

本章描述了 Inter-Integrated Circuit (I2C)并说明了如何在内核对其进行配置。

4.1 硬件概述

4.1.1 高通通用串行引擎

高通通用串行引擎 (QUP)提供了通用的数据路径引擎来支持多个微型核心。每个微型核心可实现特定协议的逻辑。常见的 FIFO 为不同的外部接口提供了一致的系统输入输出缓冲和系统 DMA 模型。例如,一对 FIFO 缓冲可独立支持 SPI 和 I2C 微型核心。

4.1.2 I2C核心

I2C 核心支持 I2C Standard (100 kHz) 及 Fast (400 kHz)。MSM8x26 增加了以下主要功能：

- BAM 整合
- 支持 I2C 标记版本

4.1.3 QUP基准地址及IRQs

通过 BLSP 核心以及 QUP 核心（1 到 6）识别每个 QUP，来匹配软件界面手册中的标记。

NOTE: MSM8x26 芯片集只包含一个 BLSP 核心。

Table 4-1 QUP 物理地址

BLSP 硬件 ID	QUP 核心	物理地址
BLSP1	BLSP 1 QUP 1	0xF9923000
BLSP1	BLSP 1 QUP 2	0xF9924000
BLSP1	BLSP 1 QUP 3	0xF9925000
BLSP1	BLSP 1 QUP 4	0xF9926000
BLSP1	BLSP 1 QUP 5	0xF9927000
BLSP1	BLSP 1 QUP 6	0xF9928000

Table 4-2 QUP IRQ

BLSP 硬件 ID	QUP 核心	IRQ
BLSP1	BLSP 1 QUP 1	95

BLSP1	BLSP 1 QUP 2	96
BLSP1	BLSP 1 QUP 3	97
BLSP1	BLSP 1 QUP 4	98
BLSP1	BLSP 1 QUP 5	99
BLSP1	BLSP 1 QUP 6	100

4.2 软件概述

4.2.1 在内核中将QUP核心配置为I2C

本节描述了如何将 MSM8x26 芯片集中的任一可用 QUP 核心配置为及用作 I2C 设备。

高通默认预配置 BLSP1_QUP1 为 I2 C。更多详细信息，请参考 `/kernel/arch/arm/boot/dts/msm226.dtsi` 及 `kernel/Documentation/devicetree/bindings/i2c/i2c-qup.txt`。

4.2.2 代码变更

以下示例描述了对于 MSM8226 芯片集，如何将 BLSP1_QUP5 配置为 I2C。

需修改的文件如下：

`/kernel/arch/arm/boot/dts/msm8226.dts`

1. 添加新的设备树节点。

```
i2c@f9927000 { /* BLSP1 QUP5 */
     cell-index = <5>; //BUS ID can be any # recommend OEM to use large
# so won't conflict with Qualcomm id
     compatible = "qcom,i2c-qup"; //Manufacturer model
     #address-cells = <1>;// address for slave chips
     #size-cells = <0>; // size for slave chips
     reg-names = "qup_phys_addr"; //Name of QUP_BASE
     reg = <0xf9927000 0x1000>; //BLSP1 QUP5 BASE address and size
     interrupt-names = "qup_err_intr"; //Interrupt Name
     interrupts = <0 99 0>;
     gpios = <&msmgpio 19 0>, /* SCL */
          <&msmgpio 18 0>; /* SDA */
    qcom,i2c-bus-freq = <100000>; //Output Clk frequency (can be 100KHz,
or 400KHz)
     qcom,i2c-src-freq = <19200000>; //Source clock frequency
};
```

2. 添加时钟节点。

该步骤描述了如何更改时钟表。

需修改以下文件：

Project_Root/kernel/arch/arm/mach-msm/clock-8226.c

添加时钟节点的代码如下：

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f9927000.i2c"),
  /*
  Add a node to QUP Core clock.
  Note: In clock regime QUP cores are label #1 to #6.
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_qup5_i2c_apps_clk.c, "f9927000.i2c"),
```

3. 设置 GPIO。

NOTE: 这一步骤是建立在 MSM8226-ES 发布版本的基础上。

需修改以下文件：

Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c

```
static struct gpiomux_setting gpio_i2c_config = {
  .func = GPIOMUX_FUNC_3, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE for UART
};

static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
    .gpio      = 18,  //BLSP1_QUP5 I2C_SDA
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_i2c_config,
    },
  },
  {
    .gpio      = 19,  //BLSP1_QUP5 I2C_SCL
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_i2c_config,
    },
  },
```

注册 GPIO：

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
```

```
        ..
        //Registers the GPIO with GPIO class
        msm_gpiomux_install(msm_blsp_configs, ARRAY_SIZE(msm_blsp_configs));
    }
```

4.2.3 快速验证

本节描述了如何验证 I2C 总线。

确保总线已注册。

如果信息已正确输入，那么 I2C 总线注册在/dev/i2c-下，并且 cell-index 和总线编号匹配。

```
adb shell --> Get adb shell
cd /dev/
ls i2c* --> to List all the I2C buses
root@android:/dev # ls i2c*
ls i2c*
i2c-0
i2c-10
i2c-2
```

如果总线已正确注册，通过简单程序可对其进行测试。示例在以下目录中创建：`/vendor/qcom/proprietary/kernel-tests/i2c-test/`。

一旦编译并运行了程序，如果 I2C 总线已正确编制，并且子设备能响应，会看到以下输出信息：

```
root@android:/data # ./i2c-test
./i2c-test
Device Open successfull [3]
I2C RDWR Returned 1
```

如果出现错误，会看到以下输出信息：

```
./i2c-test
Device Open successfull [3]
I2C RDWR Returned -1
```

如果 I2C RDWR 返回-1，在内核日志中检查驱动错误信息。如果看到以下错误信息：

```
[ 6131.397699] qup_i2c f9927000.i2c: I2C slave addr:0x54 not connected
```

通常情况下，子设备不会发送确认消息。总线已正确配置，并且至少发送了起始位和地址位，但子设备不承认并拒绝接收。

这时，调试主要集中在子设备上。确保子设备已正确上电并准备接受消息。

如果看到以下错误消息：

```
[ 6190.209880] qup_i2c f9927000.i2c: Transaction timed out, SL-AD = 0x54
[ 6190.216389] qup_i2c f9927000.i2c: I2C Status: 132100
[ 6190.221247] qup_i2c f9927000.i2c: QUP Status: 0
[ 6190.225857] qup_i2c f9927000.i2c: OP Flags: 10
```

这些消息是由多个问题引起的：

- 无效的软件配置
- 无效的硬件配置
- 子设备故障

4.3 调试建议

在进行简单读写时，如果 I2C 总是失败，以下提供了一些调试建议。

- 检查 SDA/SCL Idling。
- 检查总线，确认 SDA/SCL 闲置在逻辑高层。否则，要么是硬件配置问题要么是 GPIO 设置无效。
- 在错误消息行设置断点。例如，在交互超时消息中：

```
static int
qup_i2c_xfer(struct i2c_adapter *adap, struct i2c_msg msgs[], int num)
{
...//Put a breakpoint inside if statement.
  if (!timeout) {
    uint32_t istatus = readl_relaxed(dev->base +
 QUP_I2C_STATUS);
```

4.4 步骤

4.4.1 使用设备树注册备用设备

验证 I2C 总线后，创建备用设备驱动，并将其注册在 I2C 总线。可参考以下文件。

关于 I2C 备用设备，请参考 `msm8226-qrd.dts`。

关于注册 Synaptic 触屏驱动，请参考 `kernel/drivers/input/touchscreen/synaptics_i2c_rmi4.c`。

以下示例说明了正确使用设备树注册备用设备的最低要求。

4.4.2 创建设备树节点

修改以下文件：

/kernel/arch/arm/boot/dts/msm8226-qrd.dts

添加新的设备树节点：

```
i2c@f9927000 { /* BLSP1 QUP5 */
   synaptics@20 {//Slave driver and slave Address
        compatible = "synaptics,rmi4";//Manufacture, model

        reg = <0x20>;//Slave Address
        interrupt-parent = <&msmgpio>;//GPIO handler
        interrupts = <17 0x2>;//GPIO # will be converted to gpio_irq
        vdd-supply = <&pm8226_l19>;
        vcc_i2c-supply = <&pm8226_lvs1>;
        synaptics,reset-gpio = <&msmgpio 16 0x00>;//Pass a GPIO
        synaptics,irq-gpio = <&msmgpio 17 0x00>;
        synaptics,button-map = <139 102 158>;
        synaptics,i2c-pull-up;
        synaptics,reg-en;
   };
};
```

4.4.3 备用设备示例

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/i2c.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <linux/gpio.h>
#include <linux/debugfs.h>
#include <linux/seq_file.h>
#include <linux/regulator/consumer.h>
#include <linux/string.h>
#include <linux/of_gpio.h>

#ifdef CONFIG_OF //Open firmware must be defined for dts useage
static struct of_device_id qcom_i2c_test_table[] = {
   { .compatible = "qcom,i2c-test",}, //Compatible node must match dts
   { },
};
```

```
#else
#define qcom_i2c_test_table NULL
#endif

//I2C slave id supported by driver
static const struct i2c_device_id qcom_id[] = {
   { "qcom_i2c_test", 0 },
   { }
};

static int i2c_test_test_transfer(struct i2c_client *client)
{
  struct i2c_msg xfer; //I2C transfer structure
  u8 buf = 0x55;  //data to transfer
  xfer.addr = client->addr;
  xfer.flags = 0;
  xfer.len = 1;
  xfer.buf = &buf;

  return i2c_transfer(client->adapter, &xfer, 1);
}

static int __devinit i2c_test_probe(struct i2c_client *client,
        const struct i2c_device_id *id)
{
  int irq_gpio = -1;
  int irq;
  int addr;
  //Parse data using dt.
  if(client->dev.of_node){
    irq_gpio = of_get_named_gpio_flags(client->dev.of_node,
"qcom_i2c_test,irq-gpio", 0, NULL);
  }
  irq = client->irq; //GPIO irq #. already converted to gpio_to_irq
  addr = client->addr; //Slave Addr
  dev_err(&client->dev, "gpio [%d] irq [%d] gpio_irq [%d] Slaveaddr [%x]
\n", irq_gpio, irq,
    gpio_to_irq(irq_gpio), addr);

  //You can initiate a I2C transfer anytime using i2c_client *client
structure
  i2c_test_test_transfer(client);

  return 0;
}
```

```
1
2         //I2C Driver Info
3         static struct i2c_driver i2c_test_driver = {
4             .driver = {
5                  .name     = "qcom_i2c_test",
6                  .owner    = THIS_MODULE,
7                  .of_match_table = qcom_i2c_test_table,
8             },
9             .probe         = i2c_test_probe,
10            .id_table       = qcom_id,
11        };
12
13        //Easy wrapper to do driver init
14        module_i2c_driver(i2c_test_driver);
15
16        MODULE_DESCRIPTION("I2C TEST");
17        MODULE_LICENSE("GPL v2");
18
19        在内核日志中，以下消息表示已成功配置设备树。
20
21        <3>[    2.670731] qcom_i2c_test 2-0052: gpio [61] irq [306] gpio_irq [306]
22                          Slaveaddr [52]
```

5 UART

本章介绍了 UART 并说明了如何在 bootloader 和内核中配置 UART。

5.1 硬件概述

5.1.1 BLSP

BLSP 是在 MSM8x26 芯片族中代替传统 GSBI 核心的新设计。所有传统 GSBI 软件寄存器都已被移除。

每个 BLSP 块包含六个 QUP 和六个 UART 核心。BAM 代替了旧数据移动器（ADM）被用作硬件数据移动器。每个 BLSP 外围设备静态连接到一对 BAM 管道，该外围设备包含 26 个可用于数据移动的管道，并支持 BAM 和非 BAM 数据传输。

5.1.2 UART核心

芯片集增加了以下功能：

- 支持 BAM
- 单字符模式

UART 核心通过串口传输和接收数据，用于和其他 UART 协议设备进行数据通信。这种模式的配置主要是由 UART_DM_MR1 和 UART_DM_MR2 寄存器定义的。

5.1.3 基址

每个 UART 是通过 BLSP 核心 1 和 UART 核心（1 到 6）识别的，这样可以匹配软件接口手册中的标签。

Table 5-1 UART 物理地址

BLSP 硬件 ID	UART 核心	物理地址 (UART_DM_BASE_ADDRESS)
BLSP1	BLSP 1 UART 1	0xF991D000
BLSP1	BLSP 1 UART 2	0xF991E000
BLSP1	BLSP 1 UART 3	0xF991F000
BLSP1	BLSP 1 UART 4	0xF9920000
BLSP1	BLSP 1 UART 5	0xF9921000
BLSP1	BLSP 1 UART 6	0xF9922000

5.1.4 IRQ编号

每个 UART 是通过 BLSP 核心 1 和 UART 核心（1 到 6）识别的，这样可以匹配软件接口手册中的标签。

Table 5-2 UART 中断表

BLSP 硬件 ID	UART 核心	IRQ #
BLSP1	BLSP 1 UART 1	107
BLSP1	BLSP 1 UART 2	108
BLSP1	BLSP 1 UART 3	109
BLSP1	BLSP 1 UART 4	110
BLSP1	BLSP 1 UART 5	111
BLSP1	BLSP 1 UART 6	112

5.2 软件概述

5.2.1 Bootloader中的UART

高通默认在 bootloader 中配置 BLSP1 UART3。本节描述了配置一个不同的 UART 所需的变更，该 UART 用于调试。

5.2.1.1 启用UART用于调试

需修改的文件如下：

```
Project_Root/bootable/bootloader/lk/project/msm8226.mk
```

设置 flag, WITH_DEBUG_UART 为 TRUE：

```
DEFINES += WITH_DEBUG_UART=1
```

5.2.1.2 设置基址

需修改的文件如下：

```
Project_Root/bootable/bootloader/lk/target/msm8226/init.c
```

设置正确的基址：

```
void target_early_init(void)
{
#if WITH_DEBUG_UART
  /*
  First argument represents the ID (can be any since it's not used)
  Second argument, if it is a GSBI base, must be 0
  Third Argument is the physical address for UART CORE defined in
  /bootable/bootloader/lk/platform/msm8226/include/platform/iomap.h
```

```
  */
  uart_dm_init(1, 0, BLSP1_UART2_BASE);//it is uart[0..5] instead of
uart[1..6]
#endif
```

5.2.1.3 配置时钟

需要以下两类时钟，这样 UART 可以在 MSM8226 设备上正确运行：

- BLSP AHB 时钟
- UART 核心时钟

需修改的文件如下：

```
Project_Root/bootable/bootloader/lk/platform/msm8226/acpuclock.c
```

启用时钟：

```
void clock_config_uart_dm(uint8_t id)
{
  int ret;
  /*
    NOTE: In clock regime clocks are # from 1 to 6 so UART0 would
    be identified as UART1
  */
  //iface_clk is BLSP clk
  ret = clk_get_set_enable("uart3_iface_clk", 0, 1);

  //core_clock is UART clock.
  ret = clk_get_set_enable("uart3_core_clk", 7372800, 1);
}
```

在时钟体系中进行注册。默认启用 BLSP1_AHB 时钟。

5.2.1.4 配置GPIO

需要修改的文件如下：

```
Project_Root/bootable/bootloader/lk/platform/msm8226/gpio.c
```

配置正确的 GPIO：

```
void gpio_config_uart_dm(uint8_t id)
{
  /*
  Configure the RX/TX GPIO
  Argument 1: GPIO #
  Argument 2: Function (Please see device pinout for more information)
  Argument 3: Input/Ouput (Can be 0/1)
```

```
  Argument 4: Should be no PULL
  Argument 5: Drive strength
  Argument 6: Output Enable (Can be 0/1)
  */
  gpio_tlmm_config(9, 2, GPIO_INPUT, GPIO_NO_PULL,
  GPIO_8MA, GPIO_DISABLE);
  gpio_tlmm_config(8, 2, GPIO_OUTPUT, GPIO_NO_PULL,
  GPIO_8MA, GPIO_DISABLE);
}
```

5.2.1.5 调试建议

如果正确配置了 UART，串行控制台会显示以下消息：

Android Bootloader - UART_DM Initialized!!!

如果没看到此消息，参考以下建议：

- 设置断点

在以下文档 msm_boot_uart_dm_write(uart_dm_base, data, 44)的 uart_dm_init 功能行，添加断点到 bootloader：

`/bootable/bootloader/lk/platform/msm_shared/uart_dm.c`

- 检查正确设置的 GPIO

检查 GPIO 配置寄存器，GPIO_CFGn，确保 GPIO 设置有效。

5.2.2 内核中的低速UART

低速 UART 是基于 FIFO 的 UART 驱动，用于低速传输较小的数据，比如控制台调试及 IrDA 传输。如要传输大量的数据或者需进行高速数据传输，高通推荐使用高速 UART 驱动。

BLSP1 UART3 Base 0xF991F000 默认预配置为低速 UART。

5.2.2.1 代码变更

以下示例描述了如何配置 BLSP1 UART3，以便在 MSM8226 芯片集上使用低速 UART 驱动。

同样的方法适用于 MSM8x26 芯片集，但有以下改动：

设备树源-- `/kernel/arch/arm/boot/dts/msm8226.dtsi`

Clk 表 -- `/kernel/arch/arm/mach-msm/clock-8226.c`

在此文件中，更新 msm_clocks_8226[]表。

GPIO 表 -- `/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

1. 创建设备树节点

更多细节，请参考设备树源：
/kernel/arch/arm/boot/dts/msm8226.dtsi

NOTE: UART 目前没有指定的 ID 字段。因此，第一个注册的 UART 是 ttyHSL0，第二个是 ttyHSL1。

需修改以下文件：
/kernel/arch/arm/boot/dts/msm8226.dtsi

增加新的设备树节点：

```
serial@f991f000 { //0xF991F000 is the UART_DM Base Address
  compatible = "qcom,msm-lsuart-v14"; //manufacture, model of serial
driver
  reg = <0xf991f000 0x1000>; //Base address UART_DM and size
  /*
    First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
    Second Field: Interrupt #
    Third field: Trigger type, keep 0
    For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
  */
  interrupts = <0 109 0>;
  status = "ok"; //Status OK enables it
};
```

2. 设置时钟

修改以下文件：
Project_Root/kernel/arch/arm/mach-msm/clock-8226.c

添加时钟节点：

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f991f000.serial"),
  /*
  Add a node to UART Core clock.
  Note: In clock regime UART clocks are label #1 to #6.
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_uart3_apps_clk.c, "f991f000.serial"),
```

3. 设置 GPIO

修改以下文件：

Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c

a. 创建如下配置结构：

```
static struct gpiomux_setting gpio_uart_config = {
  .func = GPIOMUX_FUNC_2, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE for UART
};
```

b. 创建 GPIO 数组：

```
static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
    .gpio      = 8,  //BLSP1 UART3 TX
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_uart_config,
    },
  },
  {
    .gpio      = 9,  //BLSP1 UART3 RX
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_uart_config,
    },
  },
```

c. 注册 GPIO：

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs,
ARRAY_SIZE(msm_blsp_configs));
```

4. 调试建议

以下描述了如何验证低速 UART。

a. 检查注册情况

运行以下命令，确保 UART 正确注册到 TTY 栈：

```
adb shell -> start a new shell
ls /dev/ttyHSL* -> Make sure UART is properly registered
```

如果没有看到设备，查看代码更正，确保已定义所有信息并且信息是正确的。

b. 检查内部回路

i 运行以下命令启用回路：

```
adb shell
mount -t debugfs none /sys/kernel/debug  -> mount debug fs
cd /sys/kernel/debug/msm_serial_hsl      -> directory for Low
Speed UART
echo 1 > loopback.#                      -> enable loopback. # is
device #
cat loopback.#                           -> make sure returns 1
```

ii 打开另一个 shell，转储 UART Rx 数据：

```
adb shell
cat /dev/ttyHSL#  ->Dump any data UART Receive
```

iii 通过单独的 shell 传输测试数据：

```
adb shell
echo "This Document Is Very Much Helpful" > /dev/ttyHSL# -
>Transfer data
```

如果回路可以运行：

由于内部回路以及 cat 程序打开 UART 的方式，可以连续在命令 shell 中看到测试消息环路直到退出 cat 程序。

假设已正确配置 UART，并且只有 GPIO 设置需要确认。

c. 回路不能运行：

如果回路不能运行，检查时钟设置。

检查时钟之前，从 shell 打开 UART，确认 UART 处于激活状态：

```
adb shell
cat /dev/ttyHSL#  ->Dump any data UART Receive
```

d. 回路可以运行，但没有信号输出

如果回路可以运行，但是没有任何信号输出，请检查 GPIO 设置。

5.2.3 内核中的高速UART

UART_DM 可被配置为基于 BAM 的 UART。此驱动是专为高速、大数据传输设计的，比如蓝牙通信。

5.2.3.1 BLSP BAM地址及IRQ

每个 BLSP 核心有一个主要的 BAM 硬件块和一个专用的 IRQ。

Table 5-3 BLSP BAM 物理地址

BLSP 硬件 ID	UART 核心	(BLSP_BAM, IRQ)
BLSP1	BLSP 1 UART[1:6]	0xF9904000, 238

5.2.3.2 BAM管道分配

每个 BLSP UART 核心有两个为数据移动预分配的独特管道(Consumer and Producer)。UART Pipe Assignment 表列出了每个 BLSP UART_DM 核心所用的管道。

Table 5-4 BAM 管道

BLSP 硬件 ID	UART 核心	Pipes (Consumer and Producer)
BLSP1	BLSP 1 UART1	0,1
BLSP1	BLSP 1 UART2	2,3
BLSP1	BLSP 1 UART3	4,5
BLSP1	BLSP 1 UART4	6,7
BLSP1	BLSP 1 UART5	8,9
BLSP1	BLSP 1 UART6	10,11

5.2.3.3 代码变更

以下示例说明了如何在 MSM8226 芯片集上将 BLSP1_UART2 配置为高速 UART。

设备树源 -- `/kernel/arch/arm/boot/dts/msm8226.dtsi`

Clk 表 -- `/kernel/arch/arm/mach-msm/clock-8226.c`

在此文件中，更新 msm_clocks_8226[]表。

GPIO 表 -- `/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

ARM TrustZone 变更 -- `/trustzone_images/core/hwengines/bam/8x26/bamtgtcfgdata_tz.h`

1. 创建设备树节点

   详细信息，请参考以下设备树文档：
   `/kernel/Documentation/devicetree/bindings/tty/serial/msm_serial_hs.txt.`

   修改以下文件：
   `/kernel/arch/arm/boot/dts/msm8226.dtsi`
   或者任何使用的 dts/dtsi 文件。

   以下列出了最低要求：

```
  uart2: uart@f991e000 { //0xF991E000 is the UART_DM Base address for
BLSP1_UART2
    /*
      UART ID, Recommend OEM to use Large ID so does not conflict with
      Qualcomm configured ID.
      cell-index 102 would represent as /dev/ttyHS102
    */
    cell-index = <102>; //UART ID, Recommend OEM to use Large ID # that
< 255
```

```
            compatible = "qcom,msm-hsuart-v14"; //manufacture, model (must be
        same)
            status = "ok"; //"ok" or "okay" to enable
            /*
            First Row UART_DM Base and Size always 0x1000
            Second Row is BAM address, size always 0x19000
            For BLSP1 UART0:5 Bam Address = 0xF9904000
            */
            reg = <0xf991e000 0x1000>,
            <0xf9904000 0x19000>;
            reg-names = "core_mem", "bam_mem"; //Keep the same names
            /*
            First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
            Second Field: Interrupt #
            Third field: Trigger type, keep 0
            For more
        information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
            First Row UART_DM IRQ #
            Second Row is BAM IRQ
            For BLSP1 UART0:5 IRQ = 238
            */
            interrupts = <0 108 0>, <0 238 0>;
            interrupt-names = "core_irq", "bam_irq"; //Keep same
            qcom,bam-tx-ep-pipe-index = <2>; //BAM Consumer Pipe
            qcom,bam-rx-ep-pipe-index = <3>; //BAM Producer Pipe
          };
```

2. 设置时钟

NOTE: 这部分内容基于 MSM8226-ES 发布版本。这些时钟不支持设备树格式。

修改以下文件：

Project_Root/kernel/arch/arm/mach-msm/clock-8226.c

在 UART 核心时钟上添加节点：

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f991e000.uart"),
  /*
  Add a node to UART core clock
  Note: In clock regime UART clocks are labeled #1 to 6, so UART2 is
  BLSP1 UART 1
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_uart2_apps_clk.c, "f991e000.uart"),
```

3. 设置 GPIO。

NOTE: 这部分内容基于 MSM8226-ES 发布版本，GPIO 驱动目前不完全支持设备树格式。

修改以下文件：

Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c

a. 创建如下配置结构：

```
static struct gpiomux_setting gpio_uart_config = {
  .func = GPIOMUX_FUNC_2, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE for UART
};
```

b. 创建 GPIO 数组：

```
static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
       .gpio      = 4,               /* BLSP2 UART TX */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
  {
       .gpio      = 5,               /* BLSP2 UART RX */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
  {
       .gpio      = 6,               /* BLSP2 UART CTS */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
  {
       .gpio      = 7,               /* BLSP2 UART RFR */
       .settings = {
            [GPIOMUX_SUSPENDED] = &gpio_uart_config,
       },
  },
```

c. 注册 GPIO：

```
void __init msm_8226_init_gpiomux(void)
```

```
        {
          rc = msm_gpiomux_init(NR_GPIO_IRQS);
          ..
          /*
            Make sure the same GPIO # not registered twice or used by
            another subsystem such as modem.
          */
          msm_gpiomux_install(msm_blsp_configs,
        ARRAY_SIZE(msm_blsp_configs));
```

4. 更改 TrustZone

   任何子系统（比如 Modem 和 LPASS）都可以使用每个 BLSP。需将 BAM pipe 的控制权交给应用处理器。

   修改以下文件：

   ```
   /trustzone_images/core/hwengines/bam/8226/bamtgtcfgdata_tz.h
   ```

   将应用核心分配给 BLSP BAM 管道：

   ```
   /*
     bam_tgt_blsp1_secconfig = BLSP1 Core

     Each Bit You set represent Pipe #.
       For example Bit 0 = Pipe # 0
                   Bit 5 = Pipe # 5

     Any bit set on TZBSP_VMID_AP owns by APPs
   */
   bam_sec_config_type bam_tgt_blsp1_secconfig =
   {
       { //Since BLSP1_UART2 is pipe 2, 3.  Bit 2, 3 should be set to APPS
           {0x00C0FFFF, TZBSP_VMID_AP}, //Bit 2,3 Should be Set
           {0x003F0000, TZBSP_VMID_LPASS } //Make sure Bit 2,3 is Cleared
       }
};
```

5.2.3.4 调试建议

本节描述了如何验证高速 UART。

- 检查注册情况

  运行以下命令，确保 UART 正确注册到 TTY 栈：

  ```
  adb shell -> start a new shell
  ls /dev/ttyHS* -> Make sure UART is properly registered
  ```

如果没有看到设备，查看代码更正，确保已定义所有信息并且信息是正确的。

- 检查内部回路

a. 运行以下命令启用内部回路：

```
adb shell
mount -t debugfs none /sys/kernel/debug  -> mount debug fs
cd /sys/kernel/debug/msm_serial_hs       -> directory for High Speed
UART
echo 1 > loopback.#                      -> enable loopback. # is
device #
cat loopback.#                           -> make sure returns 1
```

b. 打开另一个 shell，转储UART Rx 数据：

```
adb shell
cat /dev/ttyHS#  ->Dump any data UART Receive
```

c. 通过单独的 shell 传输测试数据：

```
adb shell
echo "This Is A Helpful Document" > /dev/ttyHS# ->Transfer data
```

如果回路可以运行：

由于内部回路及 cat 程序打开 UART 的方式，可以连续在 shell 命令中看到测试消息环路直到退出 cat 程序。

假设已正确配置 UART，并且只有 GPIO 设置需要确认。

- 回路不能运行
  - 确保应用处理器能接入指定的管道。

Table 5-5 BAM 管道

BLSP 管道	PIPE_TRUST_REG	PIPE_CTRL_REG
BLSP1_P 0	0xF9905030	0xF9905000
BLSP1_P 1	0xF9906030	0xF9906000
BLSP1_P 2	0xF9907030	0xF9907000
BLSP1_P 3	0xF9908030	0xF9908000
BLSP1_P 4	0xF9909030	0xF9909000
BLSP1_P 5	0xF990a030	0xF990a000

一旦系统启动，将 JTAG 附到 RPM 和 Cortex A7 核心上(t32 cmd: sys.m.a)。

断开所有 Cortex A7 核心(t32 cmd: b)。

确保 APCS_CLOCK_BRANCH_ENA_VOTE 寄存器 (APCS_CLOCK_BRANCH_ENA_VOTE = 0xFC401484)已设置 bit 15 和 bit 17，打开 BLSP_AHB_CLK。

NOTE: MSM8x26 芯片集只需要 bit 17。

- 确保应用处理器 BLSP_PIPE_TRUST_REG 设为零。（通过安全模式读取）

```
D AZ:0xF9907030 -- BLSP1 pipe 2 \\
D AZ:0xF9908030 -- BLSP1 pipe 3
```

- 确认在非安全模式，可以读/写 PIPE_CTRL_REG。

```
D.S A:0x0:0xF9907000 \%LE \%LONG 0xABCD -- Writing pipe 2 \\
D.S A:0x0:0xF9908000 \%LE \%LONG 0xABCD -- Writing pipe 3 \\
data.in A:0xF9907000 /long -- Reading pipe 2 \\
data.in A:0xF9908000 /long -- Reading pipe 3
```

如果以上测试失败，检查 TrustZone 更正。

- 检查时钟设置

检查时钟设置前，确保 UART 处于激活状态。从 shell 打开 UART：

```
adb shell
cat /dev/ttyHS#  ->Dump any data UART Receive
```

- 回路可以运行，但没有信号输出

如果回路可以运行，但是没有任何信号输出，请检查 GPIO 设置。

可选配置：

一旦基本 UART 功能得到验证，可增加以下设置，加强 UART_DM 功能。

- 运行时间 GPIO 配置

UART 驱动可以选择支持运行时间 GPIO 配置，这样在激活状态下，GPIO 配置为 UART 模式。在暂停状态下，GPIO 转换至默认的暂停/省电模式，该模式是由用户指定的。

修改以下文件：

```
Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c
```

d. 创建 GPIO 激活/暂停配置结构：

```
  //GPIO Configuration during Active State
  static struct gpiomux_setting gpio_uart_active_config = {
    .func = GPIOMUX_FUNC_2, //Please look @ GPIO function table for
proper value.
    .drv = GPIOMUX_DRV_8MA,  //Drive Strength
    .pull = GPIOMUX_PULL_NONE,//Should be PULL NONE
  };
 //Obtain recommended suspend settings from HW engineer
  static struct gpiomux_setting gpio_uart_suspend_config = {
    .func = GPIOMUX_FUNC_GPIO, //SUSPEND Configuration
    .drv = GPIOMUX_DRV_2MA, //Drive Strength
    .pull = GPIOMUX_PULL_NONE, //PULL Configuration
  };
```

e. 创建 GPIO 数组：

```
  static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
```

```
    {
      .gpio      = 4,              /* BLSP2 UART TX */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
    {
      .gpio      = 5,              /* BLSP2 UART RX */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
    {
      .gpio      = 6,              /* BLSP2 UART CTS */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
    {
      .gpio      = 7,              /* BLSP2 UART RFR */
      .settings = {
    [GPIOMUX_ACTIVE] =    &gpio_uart_active_config,
    [GPIOMUX_SUSPENDED] = &gpio_uart_suspend_config,
      },
    },
```

f. 注册 GPIO：

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs,
ARRAY_SIZE(msm_blsp_configs));
```

g. 在设备树上添加 GPIO 编号：

修改以下文件：

```
/kernel/arch/arm/boot/dts/msm8226.dtsi
```

或者任何使用的 dts/dtsi 文件。

要启用带有 UART 核心的运行时间 GPIO 配置，须遵循以下设备树配置。

```
//Add following additional nodes to enable RunTime GPIO Configuration
uart2: uart@f991e000 { //0xF991E000 is the UART_DM Base address for BLSP1_UART2
    /*
    Please look @ /kernel/Documentation/devicetree/bindings/gpio/gpio-msm.txt
    for more info on how to specify GPIO
    First Field = Controller name
    Second Field = GPIO #
    Third Field = Not Used.
    */
    qcom,tx-gpio = <&msmgpio 4 0>;
    qcom,rx-gpio = <&msmgpio 5 0>;
    qcom,cts-gpio = <&msmgpio 6 0>;
    qcom,rfr-gpio = <&msmgpio 7 0>;
  };
```

最新细节，请参考以下设备树文档：

/kernel/Documentation/devicetree/bindings/tty/serial/msm_serial_hs.txt。

6 串行外设接口

本章描述了串行外设接口(SPI)并说明了如何在内核中对其进行配置。

6.1 硬件概述

6.1.1 SPI核心

SPI 支持主备设备间的全双工、半双工、同步和串行通信。并且没有明确的通信框架、错误检查或定义的数据字长。

主要功能：

- 支持最高 48MHz 频段
- 支持 4 到 32 位字长的传输
- 支持每总线最多四个芯片选择(CS)
- 支持 BAM（对于 8x26 芯片集来说是全新的）

Table 6-1 基址

BLSP 硬件 ID	QUP 核心	物理地址
BLSP1	BLSP 1 QUP 1	0xF9923000
BLSP1	BLSP 1 QUP 2	0xF9924000
BLSP1	BLSP 1 QUP 3	0xF9925000
BLSP1	BLSP 1 QUP 4	0xF9926000
BLSP1	BLSP 1 QUP 5	0xF9927000
BLSP1	BLSP 1 QUP 6	0xF9928000

SPI 核心和 QUP 核心共享相同的值或基址。

6.2 软件概述

6.2.1 在内核中配置QUP核心为SPI

本节描述了如何将 MSM8x26 芯片集中的任一可用 QUP 核心配置为并用作 SPI 设备。

高通默认预配置 BLSP1_QUP0 为 SPI。更多详细信息，请参考 `/kernel/arch/arm/boot/dts/msm8226.dtsi`。

6.2.1.1 代码变更

以下示例描述了对于 MSM8226 芯片集，如何将 BLSP1_QUP1 配置为 SPI。同样的方法适用于 MSM8x26 芯片集，但需进行如下变更：

设备树源 -- `/kernel/arch/arm/boot/dts/msm8226.dtsi`

Clk 表格 -- `/kernel/arch/arm/mach-msm/clock-8226.c`

在该文件中，更新 msm_clocks_8226 表。

GPIO 表格 -- `/kernel/arch/arm/mach-msm/board-8226-gpiomux.c`

TrustZone 变更 -- `/trustzone_images/core/hwengines/bam/8x26/bamtgtcfgdata_tz.h`

1. 创建设备树

   修改以下文件：
   `/kernel/arch/arm/boot/dts/msm8226.dts`

   增加新的设备树节点：

```
spi_0: spi@f9923000 { /* BLSP1 QUP1 */
     compatible = "qcom,spi-qup-v2";//Manufactur and Model
     #address-cells = <1>;
     #size-cells = <0>;
     reg-names = "spi_physical", "spi_bam_physical";
     reg = <0xf9923000 0x1000>,//Base address for BLSP1_QUP1 and size
          <0xf9904000 0xF000>;
     interrupt-names = "spi_irq", "spi_bam_irq";
     interrupts = <0 95 0>, <0 238 0>;
                /*
                First Field: 0 SPI interrupt (Shared Peripheral
Interrupt)
                Second Field: Interrupt #
                Third field: Trigger type, keep 0
                For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
                */

     spi-max-frequency = <19200000>;//Maximum supported frequency in HZ

     gpios = <&msmgpio 3 0>, /* CLK  */
          <&msmgpio 1 0>, /* MISO */
          <&msmgpio 0 0>; /* MOSI */
     cs-gpios = <&msmgpio 22 0>;

     qcom,infinite-mode = <0>;
     qcom,use-bam;
     qcom,ver-reg-exists;
```

```
        qcom,bam-consumer-pipe-index = <12>;// BAM consumer PIPE
        qcom,bam-producer-pipe-index = <13>;// BAM producer PIPE
    };
```

最新细节，请遵循：
/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt。

2. 设置时钟

该步骤描述了如何修改时钟表格。

修改以下文件：
Project_Root/kernel/arch/arm/mach-msm/clock-8226.c

增加时钟节点：

```
static struct clk_lookup msm_clocks_8226[] = {
  //Add node to BLSP1 AHB Clock
  CLK_LOOKUP("iface_clk", gcc_blsp1_ahb_clk.c, "f9923000.spi"),
  /*
  Add a node to QUP Core clock.
  */
  CLK_LOOKUP("core_clk", gcc_blsp1_qup1_spi_apps_clk.c, "f9923000.spi"),
```

3. 设置 GPIO

修改以下文件：
Project_Root/kernel/arch/arm/mach-msm/board-8226-gpiomux.c

创建如下配置结构：

```
static struct gpiomux_setting gpio_spi_config = {
  .func = GPIOMUX_FUNC_1, //Please look @ GPIO function table for
correct function
  .drv = GPIOMUX_DRV_8MA, //Drive Strength
  .pull = GPIOMUX_PULL_NONE, //Should be PULL NONE
};
```

4. 创建 GPIO 数组

```
static struct msm_gpiomux_config msm_blsp_configs[] __initdata = {
  {
    .gpio      = 0,  //BLSP1_QUP1 MOSI
    .settings = {
      [GPIOMUX_SUSPENDED] = &gpio_spi_config,
    },
  },
  {
    .gpio      = 1,  //BLSP1_QUP5 MISO
```

```
            .settings = {
              [GPIOMUX_SUSPENDED] = &gpio_spi_config,
            },
          },
          {
            .gpio      = 2,  //BLSP1_QUP5 CS
            .settings = {
              [GPIOMUX_SUSPENDED] = &gpio_spi_cs_config,
            },
          },
          {
            .gpio      = 3,  //BLSP1_QUP5 CLK
            .settings = {
              [GPIOMUX_SUSPENDED] = &gpio_spi_config,
            },
          },
```

注册 GPIO：

```
void __init msm_8226_init_gpiomux(void)
{
  rc = msm_gpiomux_init(NR_GPIO_IRQS);
  ..
  //Registers the GPIO with GPIO class
  msm_gpiomux_install(msm_blsp_configs, ARRAY_SIZE(msm_blsp_configs));
```

最新细节，请参考设备树文档：

/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt。

6.2.2 快速确认

如果所有信息已正确输入，可以看到 SPI 总线注册到/sys/class/spi_master/spi，并且 cell-index 匹配总线编号。

```
adb shell --> Get adb shell
cd /sys/class/spi_master to list all the spi master
root@android:/sys/class/spi_master # ls
ls
spi0
spi6
spi7
```

6.2.2.1 通过设备树注册备用设备

SPI 总线注册后，可以创建备用设备，并将其注册到 SPI 主用设备。关于 SPI 备用设备，可参考以下文件：

/kernel/arch/arm/boot/dts/msm8226.dts
/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt
/kernel/Documentation/devicetree/bindings/spi/spi-bus.txt

以下示例显示了注册备用设备的最低要求。

1. 创建设备树节点

   修改以下文件：
   /kernel/arch/arm/boot/dts/msm8226.dts

   增加新的设备树节点：

```
spi@f9923000 {
  qcom-spi-test@0 { //Slave driver and CS ID
    compatible = "qcom,spi-test"; //Manufacture, and Model
    reg = <0>; //Same as CS ID
    spi-max-frequency = <4800000>; //Max Frequency for Device
    /*
    * Following elements are optional.  For more detail respect to
    * SPI please see
/kernel/Documentation/devicetree/bindings/spi/spi-bus.txt
    * For general settings respect to device tree look @ sdcc device
tree
    */
    spi-cpol; //CPOL bit set for SPI polarity
    spi-cpha; //CPHA bit set for SPI Phase
    spi-cs-high; //CS Active High
    interrupt-parent = <&msmgpio>; //GPIO Handler
    interrupts = <61 0>; //GPIO # and flags
    qcom_spi_test,irq-gpio = <&msmgpio 61 0x00>; //Pass a GPIO
  };
};
```

2. 示例备用设备驱动如下：

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/spi/spi.h>
#include <linux/interrupt.h>
#include <linux/slab.h>
#include <linux/gpio.h>
#include <linux/debugfs.h>
#include <linux/seq_file.h>
#include <linux/regulator/consumer.h>
#include <linux/string.h>
#include <linux/of_gpio.h>
```

```
#ifdef CONFIG_OF //Open firmware must be defined for dts useage
static struct of_device_id qcom_spi_test_table[] = {
{ .compatible = "qcom,spi-test",}, //Compatible node must match dts
{ },
};
#else
#define qcom_spi_test_table NULL
#endif


#define BUFFER_SIZE 4<<10
struct spi_message spi_msg;
struct spi_transfer spi_xfer;
u8 *tx_buf; //This needs to be DMA friendly buffer
static int spi_test_transfer(struct spi_device *spi)
{
spi_message_init(&spi_msg);

spi_xfer.tx_buf = tx_buf;
spi_xfer.len = BUFFER_SIZE;
spi_xfer.bits_per_word = 8;
spi_xfer.speed_hz = spi->max_speed_hz;

spi_message_add_tail(&spi_xfer, &spi_msg);

return spi_sync(spi, &spi_msg);
}


static int __devinit spi_test_probe(struct spi_device *spi)
{

int irq_gpio = -1;
int irq;
int cs;
int cpha,cpol,cs_high;
u32 max_speed;

dev_err(&spi->dev, "s\n", __func__);

        //allocate memory for transfer
tx_buf = kmalloc(BUFFER_SIZE, GFP_ATOMIC);
if(tx_buf == NULL){
```

```
1               dev_err(&spi->dev, "s: mem alloc failed\n", __func__);
2               return -ENOMEM;
3           }
4           //Parse data using dt.
5           if(spi->dev.of_node){
6             irq_gpio = of_get_named_gpio_flags(spi->dev.of_node,
7           "qcom_spi_test,irq-gpio", 0, NULL);
8           }
9           irq = spi->irq;
10          cs = spi->chip_select;
11          cpha = ( spi->mode & SPI_CPHA ) ? 1:0;
12          cpol = ( spi->mode & SPI_CPOL ) ? 1:0;
13          cs_high = ( spi->mode & SPI_CS_HIGH ) ? 1:0;
14          max_speed = spi->max_speed_hz;
15          dev_err(&spi->dev, "gpio [d] irq [d] gpio_irq [d] cs [x] CPHA [x] CPOL
16          [x] CS_HIGH [x]\n",
17               irq_gpio, irq, gpio_to_irq(irq_gpio), cs, cpha, cpol, cs_high);
18
19          dev_err(&spi->dev, "Max_speed [d]\n", max_speed );
20
21          //Once you have a spi_device structure you can do a transfer anytime
22          spi->bits_per_word = 8;
23          dev_err(&spi->dev, "SPI sync returned [d]\n",  spi_test_transfer(spi));
24          return 0;
25          }
26
27
28          //SPI Driver Info
29          static struct spi_driver spi_test_driver = {
30          .driver = {
31               .name     = "qcom_spi_test",
32               .owner    = THIS_MODULE,
33               .of_match_table = qcom_spi_test_table,
34          },
35          .probe        = spi_test_probe,
36          };
37
38
39          static int __init spi_test_init(void)
40          {
41          return spi_register_driver(&spi_test_driver);
42          }
43
44          static void __exit spi_test_exit(void)
45          {
```

```
    spi_unregister_driver(&spi_test_driver);
    }


    module_init(spi_test_init);
    module_exit(spi_test_exit);
    MODULE_DESCRIPTION("SPI TEST");
    MODULE_LICENSE("GPL v2");
```

在内核日志中，以下消息表明已正确配置设备树。

```
<3>[    2.503571] qcom_spi_test spi6.0: spi_test_probe
<3>[    2.507305] qcom_spi_test spi6.0: gpio [61] irq [306] gpio_irq [306]
                       cs [0] CPHA [1] CPOL [1] CS_HIGH [1]
<3>[    2.516825] qcom_spi_test spi6.0: Max_speed [4800000]
<3>[    2.521932] qcom_spi_test spi6.0: SPI sync returned [0]
```

6.2.3 可选：启用数据移动模式 (BAM)

SPI 可以在数据移动模式(BAM)或基于 FIFO 的模式下运行。如需传输大量数据，高通推荐启用 BAM 模式以卸载 CPU。

6.2.3.1 BLSP BAM地址及IRQ

每个 BLSP 内核有一个主用 BAM 硬件块和一个专用 IRQ。

Table 6-2 BLSP BAM 物理地址

BLSP 硬件 ID	QUP 核心	BLSP_BAM, IRQ
BLSP1	BLSP1_QUP[1:6]	0xF9904000,238

Table 6-3 BAM 管道分配

BLSP 硬件 ID	QUP 核心	PIPE(in, out)
BLSP1	BLSP1_QUP_1	12,13
BLSP1	BLSP1_QUP_2	14,15
BLSP1	BLSP1_QUP_3	16,17
BLSP1	BLSP1_QUP_4	18,19
BLSP1	BLSP1_QUP_5	20,21
BLSP1	BLSP1_QUP_6	22,23

每个 BLSP QUP 核心有两个预分配给数据移动操作的独特管道(Consumer and Producer)。MSM8x26 芯片集只包含一个 BLSP 核心。

6.2.3.2 代码变更

本节描述了如何启用 SPI 的 BAM 模式。

1. 设备树变更：

   如要启用带有 SPI 核心的 BAM，需遵循以下设备树配置：

   最新细节如下：

```
/kernel/Documentation/devicetree/bindings/spi/spi_qsd.txt
  //Add following additional nodes device tree to enable BAM
  spi@f9923000 {
    /*
    Modify the reg field as below to add BLSP BAM base address.
    First Row BLSP_QUP Base and Size always 0x1000
    Second Row is BAM address, size always 0x19000
    For BLSP1 QUP0:5 Bam Address = 0xF9904000
    */
    reg = <0xf9928000 0x1000>,
          <0xf9904000 0x19000>;
    reg-names = "spi_physical", "spi_bam_physical"; //Keep the same
names


    /*
    Replace the interrupt field.
    First Field: 0 SPI interrupt (Shared Peripheral Interrupt)
    Second Field: Interrupt #
    Third field: Trigger type, keep 0
    For more
information:/kernel/Documentation/devicetree/bindings/arm/gic.txt
    First Row BLSP_QUP IRQ #
    Second Row is BAM IRQ
    For BLSP1 QUP0:5 IRQ = 238
    */
    interrupts = <0 100 0>, <0 238 0>;
    interrupt-names = "spi_irq", "spi_bam_irq"; //Keep same

    /*
    Add Consumer and Producer Pipes
    */
    qcom,use-bam; //enable BAM-mode
    qcom,bam-consumer-pipe-index = <22>; //Consumer PIPE from table
MSM8226 BLSP SPI Pipe Assignment
    qcom,bam-producer-pipe-index = <23>; //Producer PIPE from table
MSM8226 BLSP SPI Pipe Assignment
    qcom,ver-reg-exists; //Version register exists (True for 8226)
  };
```

2. TrustZone 变更：

任何子系统（比如 Modem 和 LPASS）都可以使用每个 BLSP。需将 BAM pipe 的控制权交给应用处理器。

修改以下文件：

```
/trustzone_images/core/hwengines/bam/8226/bamtgtcfgdata_tz.h
```

将应用核心分配给 BLSP BAM 管道：

```
/*
  bam_tgt_blsp1_secconfig = BLSP1 Core
  Each Bit You set represent Pipe #.
    For example Bit 0 = Pipe # 0
                Bit 5 = Pipe # 5

  Any bit set on TZBSP_VMID_AP owns by APPs
*/
bam_sec_config_type bam_tgt_blsp1_secconfig =
{
    { //Since BLSP1_QUP1 is pipe 12, 13.  Bit 12, 13 should be set to
APPS
        {0x00003FFF, TZBSP_VMID_AP}, //Bit 11,12 Should be Set
        {0x00FFC000, TZBSP_VMID_LPASS } //Make sure Bit 12,13 is Cleared
    }
};
```

3. 调试建议

如果 FIFO 模式 SPI 可以运行，但 BAM 模式 SPI 不能运行，确认应用处理器可以接入指定的管道。

一旦系统启动，将 JTAG 附到 RPM 和 Cortex A7 核心上(t32 cmd: sys.m.a)。

断开所有 Cortex A7 核心(t32 cmd: b)。

确保 APCS_CLOCK_BRANCH_ENA_VOTE 寄存器
(APCS_CLOCK_BRANCH_ENA_VOTE = 0xFC401484)已设置 bit 15 和 bit 17，打开 BLSP_AHB_CLK。

- 确保应用处理器 BLSP_PIPE_TRUST_REG 设为零。(通过安全模式读取)

```
D AZ:0xF9905030 -- BLSP1 pipe 12 \\
D AZ:0xF9906030 -- BLSP1 pipe 13
```

- 确认在非安全模式下，可以读/写 PIPE_CTRL_REG。

```
D.S A:0x0:0xF9905000 \LE \LONG 0xABCD -- Writing pipe 12 \\
D.S A:0x0:0xF9906000 \LE \LONG 0xABCD -- Writing pipe 13 \\
data.in A:0xF9905000 /long -- Reading pipe 12 \\
data.in A:0xF9906000 /long -- Reading pipe 23
```

如果以上测试失败，请检查 TrustZoned 更正。

7 相机

本章为 OEM 制造商提供如何在 MSM8x26 构建上移植相机模块的相关信息。其中重点讨论了如何将 Bayer 相机移植到 MSM8x26 平台。MSM8x26 具有一个 VFE(4.0 lite)，同时支持 Bayer 相机和 SOC 相机模块。

7.1 简介

MSM8x26 构建引入了设备树而不是传统内核驱动来描述硬件信息；要移植新的相机模块到 MSM8x26 平台，还需要做一些改动。本章提供了关于如何在以下新的硬件平台成功调试相机的指导信息。在本文档中，假设成功调试传感器 s5k3l1yx。

相机驱动由内核和用户空间组成。内核驱动显示为设备树源文件及内核驱动。用户空间驱动使用后缀“_lib”而不是“_u”。需要修改/添加以下文件，用于调试传感器 s5k3l1yx 的新驱动。

内核空间相机设备树源文件：

```
/kernel/arch/arm/boot/dts/msm8226-camera-sensor-liquid.dtsi
```

内核空间相机驱动文件：

```
/kernel/drivers/media/video/msmb/sensor/s5k3l1yx.c
```

用户空间相机驱动：

```
/vendor/qcom/proprietary/mm-camera/mm-camera2/media-
controller/modules/sensors/sensor_libs/s5k3l1yx/s5k3l1yx_lib.c
```

NOTE: 以下文件不再适用于 MSM8x26。

```
/vendor/qcom/proprietary/mm-
camera/server/hardware/sensor/s5k3l1yx/s5k3l1yx_u.c
/vendor/qcom/proprietary/mm-
camera/server/hardware/sensor/s5k3l1yx/s5k3l1yx_u.h
```

7.2 内核移植

7.2.1 设备树移植

以下设备树文件描述了传感器具体信息。这些信息取决于项目使用的相机传感器：

```
/arch/arm/boot/dts/msm8x26-camera-sensor.dtsi
```

该文件包含需定义的传感器基本硬件信息。下表列出了每字段的示例代码并进行了描述。

Table 7-1 相机属性

属性	描述	示例代码	给 OEM 的建议
compatible	应是"qcom"，之后是传感器名称	compatible = "qcom,s5k3l1yx";	和项目使用的传感器名称相同
reg	应包含相机传感器的 i2c 备用地址以及数据字段长度，长度为 0x0	reg = <0x6e 0x0>;	
qcom,slave-id	应包含 i2c 备用地址，设备 ID 地址及预计的 ID 读取值	qcom,slave-id = <0x6e 0x0 0x3121>;	
qcom,csiphy-sd-index	应包含用于接收传感器数据- 0, 1 的 csiphy 实例	qcom,csiphy-sd-index = <0>;	
qcom,csid-sd-index	应包含用于接收传感器数据- 0, 1 的 csid 核心实例	qcom,csid-sd-index = <0>;	
qcom,led-flash-sd-index	如果传感器支持，应包含 phandle 用于烧制源节点	qcom,flash-src-index = <&led_flash0>;	
qcom,mount-angle	在目标- 0、90、180、360 上，应包含传感器的物理安装弯角	qcom,mount-angle = <0>;	
qcom,sensor-name	应包含传感器的唯一名称以区分于其他传感器	qcom,sensor-name = "s5k3l1yx";	和 compatible 字段相同

属性	描述	示例代码	给 OEM 的建议
cam_vdig-supply	- cam_vdig-supply：应包含提供数字电压的调节器	cam_vdig-supply = <&pm8941_l3>;	这些参数是以硬件设计为基础的
cam_vana-supply	- cam_vana-supply：应包含提供模拟电压的调节器	cam_vana-supply = <&pm8941_l17>;	
cam_vio-supply	- cam_vio-supply：应包含提供 IO 电压的调节器	cam_vio-supply = <&pm8941_lvs3>;	
cam_vaf-supply	- cam_vaf-supply：应包含提供 AF 电压的调节器	cam_vaf-supply = <&pm8941_l23>;	
qcom,cam-vreg-name	- qcom,cam-vreg-name：应包含此传感器所需的所有调节器的名称	qcom,cam-vreg-name = "cam_vdig", "cam_vio", "cam_vana", "cam_vaf";	
qcom,cam-vreg-type	- qcom,cam-vreg-type：应包含 qcom,cam-vreg-name（按同样的顺序）提到的调节器种类 - "cam_vdig", "cam_vana", "cam_vio", "cam_vaf" - 0 用于 LDO，1 用于 LVS	qcom,cam-vreg-type = <0 1 0 0>;	
qcom,cam-vreg-min-voltage	- qcom,cam-vreg-min-voltage：应包含 qcom,cam-vreg-name property（按同样的顺序）提到的调节器最低电压电平	qcom,cam-vreg-min-voltage = <1225000 0 2850000 3000000>;	
qcom,cam-vreg-max-voltage	- qcom,cam-vreg-max-voltage：应包含 qcom,cam-vreg-name property（按同样的顺序）提到的调节器最高电压电平	qcom,cam-vreg-max-voltage = <1225000 0 2850000 3000000>;	
qcom,cam-vreg-op-voltage	- qcom,cam-vreg-op-mode: 应包含 qcom,cam-vreg-name property（按同样的顺序）提到的调节器最佳电压电平	qcom,cam-vreg-op-mode = <105000 0 80000 100000>;	

属性	描述	示例代码	给 OEM 的建议
qcom,gpio-no-mux	- qcom,gpio-no-mux：应包含可以表示 gpio mux 表格是否可用的字段 -1 如表示 gpio mux 表格不可用，那么 0 表示表格可用	qcom,gpio-no-mux = <0>;	这些参数是以硬件设计为基础的；除非需要，否则不要更改。
gpios	- gpios：应包含 GPIO 控制器的句柄及指明具体 GPIO(controller specific)的#gpio-cells 数组	gpios = <&msmgpio 15 0>, <&msmgpio 90 0>;	
qcom,gpio-reset	- qcom,gpio-reset：应包含传感器 reset_n 使用的 GPIO 的编号	qcom,gpio-reset = <1>;	
qcom,gpio-req-tbl-num	- qcom,gpio-req-tbl-num：应包含该传感器特定的 GPIO 的编号	qcom,gpio-req-tbl-num = <0 1>;	
qcom,gpio-req-tbl-flags	- qcom,gpio-req-tbl-flags：应包含 qcom,gpio-req-tbl-num property（按相同的顺序）中存在的 GPIO 的方向	qcom,gpio-req-tbl-flags = <1 0>;	
qcom,gpio-req-tbl-label	- qcom,gpio-req-tbl-label：应包含 qcom,gpio-req-tbl-num property（按相同的顺序）中存在的 GPIO 的名称	qcom,gpio-req-tbl-label = "CAMIF_MCLK", "CAM_RESET1";	
qcom,gpio-set-tbl-num	- qcom,gpio-set-tbl-num：应包含需 MSM 配置的 GPIO 的编号	qcom,gpio-set-tbl-num = <1 1>;	
qcom,gpio-set-tbl-flags	- qcom,gpio-set-tbl-flags：应包含 qcom,gpio-req-tbl-num property（按相同的顺序）中存在的 GPIO 需配置的值	qcom,gpio-set-tbl-flags = <0 2>;	
qcom,gpio-set-tbl-delay	- qcom,gpio-set-tbl-delay：应包含配置 gpio-req-tbl-num property（按相同的顺序）中规定的 GPIO 后所需的时延量	qcom,gpio-set-tbl-delay = <1000 30000>;	

属性	描述	示例代码	给 OEM 的建议
qcom,csi-lane-assign qcom,csi-lane-mask	- qcom,csi-lane-assign：应包含 lane 分配值，这样可将 CSIPHY lanes 映射到 CSID lanes - qcom,csi-lane-mask：应包含指定需启用的 CSIPHY lanes 的 lane mask	qcom,csi-lane-assign = <0x4320>; qcom,csi-lane-mask = <0x1F>;	
qcom,sensor-position	应包含相机传感器的安装弯角 - 0 -> 后置摄像头 - 1-> 前置摄像头	qcom,sensor-position = <0>;	
qcom,sensor-mode	应包含传感器的数据输出格式 - 0 -> bayer 格式 - 1 -> yuv 格式	qcom,sensor-mode = <1>;	
status = "ok";	相机模块状态	status = "ok";	保持不变

以下是相机传感器 s5k3l1yx 的示例设备树文件：

```
&cci {
   /*6e is the I2C slave Id of the sensor*/
   qcom,camera@6e {
      compatible = "qcom,s5k3l1yx";
      reg = <0x6e 0x0>;
      qcom,slave-id = <0x6e 0x0 0x3121>;
      qcom,csiphy-sd-index = <0>;
      qcom,csid-sd-index = <0>;
      /*
      - led_flash0, led_flash1 which is defined in
./arch/arm/boot/dts/msm8226-camera.dtsi
      */
      qcom,flash-src-index = <&led_flash0>;

      qcom,mount-angle = <0>;
      qcom,sensor-name = "s5k3l1yx";
      /*
      pm8941_l3 is defined in ./arch/arm/boot/dts/msm8226-regulator.dtsi
      */
      cam_vdig-supply = <&pm8941_l3>;
      cam_vana-supply = <&pm8941_l17>;
      cam_vio-supply = <&pm8941_lvs3>;
```

```
            cam_vaf-supply = <&pm8941_l23>;
            qcom,cam-vreg-name = "cam_vdig", "cam_vio", "cam_vana",
                      "cam_vaf";
            qcom,cam-vreg-type = <0 1 0 0>;
            qcom,cam-vreg-min-voltage = <1225000 0 2850000 3000000>;
            qcom,cam-vreg-max-voltage = <1225000 0 2850000 3000000>;
            qcom,cam-vreg-op-mode = <105000 0 80000 100000>;
            qcom,gpio-no-mux = <0>;
            gpios = <&msmgpio 15 0>,
               <&msmgpio 90 0>;
            qcom,gpio-reset = <1>;
            qcom,gpio-req-tbl-num = <0 1>;
            qcom,gpio-req-tbl-flags = <1 0>;
            qcom,gpio-req-tbl-label = "CAMIF_MCLK",
                      "CAM_RESET1";
            qcom,gpio-set-tbl-num = <1 1>;
            qcom,gpio-set-tbl-flags = <0 2>;
            qcom,gpio-set-tbl-delay = <1000 30000>;
            qcom,csi-lane-assign = <0x4320>;
            qcom,csi-lane-mask = <0x1F>;
            qcom,sensor-position = <0>;
            qcom,sensor-mode = <1>;
            status = "ok";
         };

         qcom,camera@6c {
            compatible = "qcom,ov2720";
            …
         };
      };
```

NOTE: 相机设备树的具体文档如下：

/kernel/Documentation/devicetree/bindings/media/video/msm-cci.txt。

7.2.2 内核驱动移植

传感器的内核驱动需根据传感器性能进行修改：

/kernel/drivers/media/video/msmb/sensor/s5k3l1yx.c

内核驱动和设备树源文件(dtsi)的关系取决于传感器名称，因此需确定相关字段信息相同。

```
#include "msm_sensor.h"
/*Sensor name must be same as the name in device tree file*/
#define S5K3L1YX_SENSOR_NAME "s5k3l1yx"
```

```
DEFINE_MSM_MUTEX(s5k3l1yx_mut);

static struct msm_sensor_ctrl_t s5k3l1yx_s_ctrl;

static struct msm_sensor_power_setting s5k3l1yx_power_setting[] = {
   {
      .seq_type = SENSOR_VREG,
      .seq_val = CAM_VDIG,
      .config_val = 0,
      .delay = 0,
   },
…
   {
      .order = 0,
   },
};

static const struct i2c_device_id s5k3l1yx_i2c_id[] = {
   {S5K3L1YX_SENSOR_NAME, (kernel_ulong_t)&s5k3l1yx_s_ctrl},
   { }
};

static struct i2c_driver s5k3l1yx_i2c_driver = {
   .id_table = s5k3l1yx_i2c_id,
   .probe = msm_sensor_i2c_probe,
   .driver = {
      .name = S5K3L1YX_SENSOR_NAME,
   },
};

static struct msm_camera_i2c_client s5k3l1yx_sensor_i2c_client = {
   .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
};
/*The compatible must be same as what is defined in device tree file*/
static const struct of_device_id s5k3l1yx_dt_match[] = {
   {.compatible = "qcom,s5k3l1yx", .data = &s5k3l1yx_s_ctrl},
   {}
};

MODULE_DEVICE_TABLE(of, s5k3l1yx_dt_match);
/*must match with the definition in device tree source file*/
static struct platform_driver s5k3l1yx_platform_driver = {
   .driver = {
      .name = "qcom,s5k3l1yx",
```

```
        .owner = THIS_MODULE,
        .of_match_table = s5k3l1yx_dt_match,
    },
};

static int32_t s5k3l1yx_platform_probe(struct platform_device *pdev)
{
    int32_t rc = 0;
    const struct of_device_id *match;
    match = of_match_device(s5k3l1yx_dt_match, &pdev->dev);
    rc = msm_sensor_platform_probe(pdev, match->data);
    return rc;
}

static int __init s5k3l1yx_init_module(void)
{
    int32_t rc = 0;
    pr_info("%s:%d\n", __func__, __LINE__);
    rc = platform_driver_probe(&s5k3l1yx_platform_driver,
        s5k3l1yx_platform_probe);
    if (!rc)
        return rc;
    pr_err("%s:%d rc %d\n", __func__, __LINE__, rc);
    return i2c_add_driver(&s5k3l1yx_i2c_driver);
}

static void __exit s5k3l1yx_exit_module(void)
{
    pr_info("%s:%d\n", __func__, __LINE__);
    if (s5k3l1yx_s_ctrl.pdev) {
        msm_sensor_free_sensor_data(&s5k3l1yx_s_ctrl);
        platform_driver_unregister(&s5k3l1yx_platform_driver);
    } else
        i2c_del_driver(&s5k3l1yx_i2c_driver);
    return;
}

static struct msm_sensor_ctrl_t s5k3l1yx_s_ctrl = {
    .sensor_i2c_client = &s5k3l1yx_sensor_i2c_client,
    .power_setting_array.power_setting = s5k3l1yx_power_setting,
    .power_setting_array.size = ARRAY_SIZE(s5k3l1yx_power_setting),
    .msm_sensor_mutex = &s5k3l1yx_mut,
    .sensor_v4l2_subdev_info = s5k3l1yx_subdev_info,
    .sensor_v4l2_subdev_info_size = ARRAY_SIZE(s5k3l1yx_subdev_info),
```

```
};

module_init(s5k3l1yx_init_module);
module_exit(s5k3l1yx_exit_module);
MODULE_DESCRIPTION("s5k3l1yx");
MODULE_LICENSE("GPL v2");
```

当内核需要支持一个新的传感器，应修改某些生成文件或配置文件。

以下列出了需修改的编译选项。

```
/drivers/media/platform/msm/camera_v2/Kconfig
…
config S5K3L1YX
  bool "Sensor S5K3L1YX (BAYER 12M)"
  depends on MSMB_CAMERA
  ---help---
    Samsung 12 MP Bayer Sensor with auto focus, uses
    4 mipi lanes, preview config = 1984 * 1508 at 30 fps,
    snapshot config = 4000 * 3000 at 20 fps,
    hfr video at 60, 90 and 120 fps.
…

/drivers/media/platform/msm/camera_v2/sensor/Makefile
…
 obj-$(CONFIG_MSMB_CAMERA) += cci/ io/ csiphy/ csid/
 obj-$(CONFIG_MSM_CAMERA_SENSOR) += msm_sensor.o
 obj-$(CONFIG_S5K3L1YX) += s5k3l1yx.o
 obj-$(CONFIG_OV2720) += ov2720.o
…

arch/arm/configs/msm8x26-perf_defconfig
arch/arm/configs/msm8x26_defconfig
…
 CONFIG_MSM_ISPIF=y
 CONFIG_S5K3L1YX=y
 CONFIG_IMX135=y
…
Also We need to provide clock configuration for the sensor
/arch/arm/mach-msm/clock-8x26.c
…
  CLK_LOOKUP("cam_src_clk", mclk0_clk_src.c, "6e.qcom,camera"),
  CLK_LOOKUP("cam_src_clk", mclk0_clk_src.c, "20.qcom,camera"),
  CLK_LOOKUP("cam_src_clk", mclk1_clk_src.c, "90.qcom,camera"),
```

```
  CLK_LOOKUP("cam_clk", camss_mclk0_clk.c, "6e.qcom,camera"),
  CLK_LOOKUP("cam_clk", camss_mclk0_clk.c, "20.qcom,camera"),
  CLK_LOOKUP("cam_clk", camss_mclk2_clk.c, "6c.qcom,camera"),
…
```

7.3 用户空间移植

关于用户空间移植，需修改以下文件：

`/vendor/qcom/proprietary/mm-camera/mm-camera2/media-controller/modules/sensors/sensor_libs/s5k3l1yx/s5k3l1yx_lib.c`

s5k3l1yx_lib.c 唯一的外部接口是：

```
void *s5k3l1yx_open_lib(void)
{
  return &sensor_lib_ptr;
}
```

当相机虚拟光驱进程开始时，会加载 lib；并且内核会基于 dtb (设备树二进制文件)中的传感器名称找到处理器。因此，请确认文件名称和功能名称符合传感器要求。

`<your_sensor_name>/<your_sensor_name>_lib.c`

`void *<your_sensor_name>_open_lib(void)`

需更新传感器的 Chromatix header，并将其放在如下文件夹中：

`mm-camera2/media-controller/modules/sensors/chromatix/0301/libchromatix/chromatix_s5k3l1yx/common/`

`chromatix_s5k3l1yx_common.h`

`chromatix_s5k3l1yx_preview.h`

`chromatix_s5k3l1yx_video.h`

7.3.1 移植电源设置

传感器的电源设置包括传感器电源信息，IO 控制和 Mclk。通常情况下，无需修改该部分内容，因为大部分配置已在设备树文件中完成；但可以修改`.delay = 0`字段来和时间匹配。

```
static struct msm_sensor_power_setting power_setting[] = {
  {
    .seq_type = SENSOR_VREG,
    .seq_val = CAM_VDIG,
    .config_val = 0,
    .delay = 0,
    .data = NULL,
  },
```

```
  {
    .seq_type = SENSOR_VREG,
    .seq_val = CAM_VANA,
    .config_val = 0,
    .delay = 0,
    .data = NULL,
  },
…
}
```

7.3.2 移植相机接口配置

这部分内容定义了传感器接口配置，包括 I2C 和 MIPI。请确认设置不会和 dtsi 文件中的类似内容冲突。

```
static struct msm_camera_sensor_slave_info sensor_slave_info = {
  /* sensor slave address */
  .slave_addr = 0x6E,
  /* sensor address type */
  .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
  /* sensor id info*/
  .sensor_id_info = {
    /* sensor id register address */
    .sensor_id_reg_addr = 0x0000,
    /* sensor id */
    .sensor_id = 0x3121,
  },
  /* power up / down setting */
  .power_setting_array = {
    .power_setting = power_setting,
    .size = ARRAY_SIZE(power_setting),
  },
};
static sensor_output_t sensor_output = {
  .output_format = SENSOR_BAYER,
  .connection_mode = SENSOR_MIPI_CSI,
  .raw_output = SENSOR_10_BIT_DIRECT,
};
static struct csi_lane_params_t csi_lane_params = {
  .csi_lane_assign = 0x4320,
  .csi_lane_mask = 0x1F,
  .csi_if = 1,
  .csid_core = {0},
};
```

7.3.3 移植传感器输出配置

这部分内容介绍了传感器不同模式的配置和设置，包含各模式的 I2C 寄存器设备、CSI 参数、crop 参数、传感器输出信息和 chormatix 数组。这些配置的数组大小应和传感器支持的模式种类相同。

```
static struct msm_camera_i2c_reg_setting res_settings[] = {
  {
    /* res0_reg_array is the I2C settings from sensor vendor*/
    .reg_setting = res0_reg_array,
    .size = ARRAY_SIZE(res0_reg_array),
    .addr_type = MSM_CAMERA_I2C_WORD_ADDR,
    .data_type = MSM_CAMERA_I2C_BYTE_DATA,
    .delay = 0,
  },
  …
};

static struct msm_camera_csi2_params *csi_params[] = {
  &s5k3l1yx_csi_params, /* csi param might be different for different
sensor mode*/
…
};

static struct sensor_crop_parms_t crop_params[] = {
  {0, 0, 0, 0}, /* All 0 means no crop */
…
};

static struct sensor_lib_out_info_t sensor_out_info[] = {
  {
    /* vt_pixel_clk  = .line_length_pclk* frame_length_lines*frame rate */
    /* op_pixel_clk  = VFE working clk */
    .x_output = 4000,
    .y_output = 3016,
    .line_length_pclk = 5336,
    .frame_length_lines = 3052,
    .vt_pixel_clk = 330000000,
    .op_pixel_clk = 264000000,
    .binning_factor = 1,
    .max_fps = 22.0,
    .min_fps = 22.0,
  },
```

```
  …
};

static struct sensor_lib_chromatix_t s5k3l1yx_chromatix[] = {
  {
    S5K3L1YX_LOAD_CHROMATIX(preview), /* RES0 */
    S5K3L1YX_LOAD_CHROMATIX(default_video), /* RES0 */
  },
  …
};
```

7.3.4 镜头信息移植

相机处理流程(比如曝光计算)需要镜头信息。因此，以下结构需添加正确的镜头信息。

```
/*Lens info, filled based on lens and sensor spec */
static sensor_lens_info_t default_lens_info = {
  .focal_length = 1.15,
  .pix_size = 1.4,
  .f_number = 2.6,
  .total_f_dist = 1.5,
  .hor_view_angle = 54.8,
  .ver_view_angle = 42.5,
};
```

7.3.5 曝光配置移植

通常情况下，不同传感器有不同的曝光控制方式。所以，当传感器打开后，需移植曝光配置以成功调试传感器的曝光控制。

包括以下函数：

- s5k3l1yx_real_to_register_gain – 将实际逻辑增益转换为寄存器值。
- s5k3l1yx_register_to_real_gain – 将寄存器值转换成真实的逻辑增益。
- s5k3l1yx_calculate_exposure – 获取曝光时间和增益的另一个曝光配置.
- s5k3l1yx_fill_exposure_array – 准备另一个曝光配置数组。

```
/*============================================================================
==
 * FUNCTION - s5k3l1yx_real_to_register_gain -
 *
 * DESCRIPTION:
 *==========================================================================
=*/
static uint16_t s5k3l1yx_real_to_register_gain(float gain)
```

```
{
  uint16_t reg_gain;

  if (gain < 1.0)
    gain = 1.0;

  if (gain > 16.0)
    gain = 16.0;

  reg_gain = (uint16_t)(gain * 32.0);

  return reg_gain;
}

/*==========================================================================
==
 * FUNCTION - s5k3l1yx_register_to_real_gain -
 *
 * DESCRIPTION:
 *==========================================================================
=*/
static float s5k3l1yx_register_to_real_gain(uint16_t reg_gain)
{
  float gain;

  if (reg_gain > 0x0200)
    reg_gain = 0x0200;

  gain = (float) reg_gain / 32.0;

  return gain;
}

/*==========================================================================
==
 * FUNCTION - s5k3l1yx_calculate_exposure -
 *
 * DESCRIPTION:
 *==========================================================================
=*/
static int32_t s5k3l1yx_calculate_exposure(float real_gain,
  uint16_t line_count, sensor_exposure_info_t *exp_info)
{
  if (!exp_info) {
    return -1;
```

```
  }
  exp_info->reg_gain = s5k3l1yx_real_to_register_gain(real_gain);
  float sensor_real_gain = s5k3l1yx_register_to_real_gain(exp_info-
>reg_gain);
  exp_info->digital_gain = real_gain / sensor_real_gain;
  exp_info->line_count = line_count;
  return 0;
}

/*============================================================================
==
 * FUNCTION - s5k3l1yx_fill_exposure_array -
 *
 * DESCRIPTION:
 *============================================================================
=*/
static int32_t s5k3l1yx_fill_exposure_array(uint16_t gain, uint32_t line,
  uint32_t fl_lines, struct msm_camera_i2c_seq_reg_setting *reg_setting)
{
  reg_setting->reg_setting[0].reg_addr =
    sensor_lib_ptr.output_reg_addr->frame_length_lines;
  reg_setting->reg_setting[0].reg_data[0] = (fl_lines & 0xFF00) >> 8;
  reg_setting->reg_setting[0].reg_data[1] = (fl_lines & 0xFF);
  reg_setting->reg_setting[0].reg_data_size = 2;
  reg_setting->reg_setting[1].reg_addr =
    sensor_lib_ptr.exp_gain_info->coarse_int_time_addr;
  reg_setting->reg_setting[1].reg_data[0] = (line & 0xFF00) >> 8;
  reg_setting->reg_setting[1].reg_data[1] = (line & 0xFF);
  reg_setting->reg_setting[1].reg_data_size = 2;
  reg_setting->reg_setting[2].reg_addr =
    sensor_lib_ptr.exp_gain_info->global_gain_addr;
  reg_setting->reg_setting[2].reg_data[0] = (gain & 0xFF00) >> 8;
  reg_setting->reg_setting[2].reg_data[1] = (gain & 0xFF);
  reg_setting->reg_setting[2].reg_data_size = 2;
  reg_setting->size = 3;
  reg_setting->addr_type = MSM_CAMERA_I2C_WORD_ADDR;
  reg_setting->delay = 0;
  return 0;
}
```

7.4 行位移植

msm_actuator_ctrl_t 包含传感器行位设置的所有信息，例如：i2c addr, set_info, focal length 等。所有信息是通过 chromatix 文件加载的。

msm_actuator.c 提供了通用函数：

/kernel/drivers/media/video/msm/actuators/actuator.c

设备树中包含了对行位的硬件描述：

/kernel/arch/arm/boot/dts/msm8x26-camera-sensor.dtsi

```
&cci {
  actuator0: qcom,actuator@18 {
    cell-index = <0>;
    reg = <0x18 0x0>;
    compatible = "qcom,actuator";
    qcom,cci-master = <0>;
  };
```

可从 Actuator.c 的 hear 文件加载 AF 参数：

```
static actuator_ctrl_t actuators[] = {
#include "af_main_cam_0.h"
#include "af_main_cam_1.h"
#include "af_main_cam_2.h"
};
```

在 af header 文件中，驱动工程师应注意到以下结构：

actuator_params_t，包含 af driver ic 地址，寄存器模式等。

这对 AF 的运行起了重要作用。

```
  /* actuator_params_t */
    {
      /* i2c_addr */
      0xE4,
      /* i2c_data_type */
      MSM_ACTUATOR_BYTE_DATA,
      /* i2c_addr_type */
      MSM_ACTUATOR_BYTE_ADDR,
      /* act_type */
      ACTUATOR_PIEZO,
      /* data_size */
      8,
      /* af_restore_pos */
      0,
      /* msm_actuator_reg_tbl_t */
```

```
            {
              /* reg_tbl_size */
              1,
              /* msm_actuator_reg_params_t */
              {
                /* reg_write_type;hw_mask; reg_addr; hw_shift >>; data_shift <<
    */
                {MSM_ACTUATOR_WRITE_DAC, 0x00000080, 0x0000, 0, 0},
              },
            },
```

8 显示屏

8.1 简介

本章为 OEM 厂商提供在 MSM8626 构建上移植显示面板的相关信息。MSM8626 只支持带有 MIPI DSI 接口的一个显示面板，以及视频模式和命令模式。面板最高分辨率为 FWVGA (1280X800)。

本章重点介绍如何在 Linux 内核移植 DSI 面板。移植显示面板前，请参阅面板供应商的相关规范。

8.2 内核移植

8.2.1 属性

相对于之前的芯片集需将设备的每个细节硬编码到单板的具体文件，8626 构建的一大改变/优势在于使用设备树来描述硬件。

显示面板配置同样是设备树的一部分, 绑定是用来描述设备树中的设备。关于显示面板的绑定，请参阅 kernel\Documentation\devicetree\bindings\fb\mdss-dsi-panel.txt。

下表一一描述了显示屏的属性，并给 OEM 厂商提供了建议。

Table 8-1 显示屏属性

属性	描述	示例代码	给 OEM 的建议
compatible	必须是“qcom,mdss-dsi-panel”	compatible = "qcom,mdss-dsi-panel";	保持不变
status	字符串需设置为“okay/ok”来启用面板驱动。该属性默认设为”disable”，在特定平台也可设为”ok/okay”状态	status = "disable";	在 panel dtsi 中将其设为“disable”，在 MSM8x26 dtsi 中将其设为“OK”
qcom,dsi-ctrl-phandle	指定 DSI 控制器的 phandle，面板将会映射到该控制器上	qcom,dsi-ctrl-phandle = <&mdss_dsi0>;	保持不变
qcom,mdss-pan-res	指定了面板分辨率的二维数组	qcom,mdss-pan-res = <720 1280>;	根据面板规格定制
qcom,mdss-pan-bpp	指定 pixel 的 bits 组成。 默认值为 24(rgb888). 18 = for rgb666 and 16 = for rgb565	qcom,mdss-pan-bpp = <24>;	根据面板规格定制

qcom,panel-phy-regulatorSettings	指定了面板 PHY 调节器设置的数组，该数组长度为 7	qcom,panel-phy-regulatorSettings = [03 01 01 00 20 00 01]; /* Regulator settings */	保持不变
qcom,panel-phy-timingSettings	指定了面板 PHY 定时设置的数组，该数组长度为 12	qcom,panel-phy-timingSettings = [69 29 1f 00 55 55 19 2a 2a 03 04 00];	最初尝试调试面板时，保持不变。如果调试失败，创建 case 进行参数优化
qcom,panel-phy-strengthCtrl	指定了面板 PHY strengthCtrl 设置的数组，该数组长度为 2	qcom,panel-phy-strengthCtrl = [77 06];	保持不变
qcom,panel-phy-bistCtrl	指定了面板 PHY BIST ctrl 设置的数组，该数组长度为 6	qcom,panel-phy-bistCtrl = [00 00 b1 ff 00 00]; /* BIST Ctrl settings */	保持不变
qcom,panel-phy-laneConfig	指定了面板 PHY lane 配置设置的数组，该数组长度为 45	qcom,panel-phy-laneConfig = [00 c2 45 00 00 00 00 01 75 /* lane0 config */ 00 c2 45 00 00 00 00 01 75 /* lane1 config */ 00 c2 45 00 00 00 00 01 75 /* lane2 config */ 00 c2 45 00 00 00 00 01 75 /* lane3 config */ 00 02 45 00 00 00 00 01 97]; /* Clk ln config */	保持不变
qcom,mdss-panel-on-cmds	列出面板 init 命令的数组，该数组长度可变。格式相关信息，请参考 \kernel\drivers\video\msm\mdss\mdss_dsi.h 的 dsi_cmd_desc 结构定义。	qcom,panel-on-cmds = [23 01 00 00 0a 02 b0 00 23 01 00 00 0a 02 b2 00];	根据面板规格定制
qcom,mdss-panel-off-cmds	列出关闭面板命令的数组，该数组长度可变。	qcom,panel-off-cmds = [05 01 00 00 32 02 28 00 05 01 00 00 78 02 10 00];	根据面板规格定制
Label	作为面板描述性名称的字符串	label = "toshiba 720p video mode dsi panel";	根据面板规格定制
qcom,enable-gpio	指定了面板的 lcd/display enable GPIO	qcom,enable-gpio = <&msmgpio 58 0>;	根据面板规格定制，可能需要更改 mdss_dsi_panel.c 中的 mdss_dsi_panel_reset()
qcom,rst-gpio	指定了面板的 reset GPIO。	qcom,rst-gpio = <&pm8941_gpios 19 0>;	根据面板规格定制
qcom,mdss-pan-porch-	指定了面板消隐值的数组，该数组大小为 6。	qcom,mdss-pan-porch-values = <32	根据面板规格定制

values		12 144 3 4 9>;	
qcom,mdss-pan-underflow-clr	指定了面板下溢清除的控制器设置。默认值为0xff。	qcom,mdss-pan-underflow-clr = <0xff>;	保持不变
qcom,mdss-pan-bl-ctrl	指定了面板背光控制执行的字符串。 "bl_ctrl_pwm"表示由 PWM gpio 控制的背光 "bl_ctrl_wled"表示由 WLED 控制的背光 "bl_ctrl_dcs_cmds"表示由 DCS 命令控制的背光	qcom,mdss-pan-bl-ctrl = "bl_ctrl_wled";	根据面板规格定制
qcom,mdss-pan-bl-levels	指定了面板支持的背光水平。 默认范围从 1 到 255。	qcom,mdss-pan-bl-levels = <1 255>;	保持不变
qcom,mdss-pan-dsi-mode	指定了面板运行模式。 0 表示启用视频模式 （默认模式） 1 表示启用命令模式	qcom,mdss-pan-dsi-mode = <0>;	根据面板规格定制
qcom,mdss-pan-dsi-h-pulse-mode	指定了面板的脉冲模式选项。 0 表示 Don't send hsa/he following vs/ve packet (默认选项) 1 Send hsa/he following vs/ve packet	qcom,mdss-pan-dsi-h-pulse-mode = <0>;	根据面板规格定制
qcom,mdss-pan-dsi-h-power-stop	在 horizontal porch and sync pulse 时，指定功率模式的数组，该数组大小为 3。 0 表示高速模式（默认模式） 1 表示 horizontal porches and sync pulse 的低速模式	qcom,mdss-pan-dsi-h-power-stop = <0 0 0>;	根据面板规格定制
qcom,mdss-pan-dsi-bllp-power-stop	在消隐期及 EOF（帧结尾）之后，指定功率模式的数组，该数组大小为 2。 0 表示高速模式（默认模式） 1 表示消隐期及 EOF 的低功耗模式	qcom,mdss-pan-dsi-bllp-power-stop = <1 1>;	根据面板规格定制
qcom,mdss-pan-dsi-traffic-mode	指定了面板业务模式。 0 = non burst with sync pulses (default mode). 1 = non burst with sync start event. 2 = burst mode.	qcom,mdss-pan-dsi-traffic-mode = <1>;	根据面板规格定制
qcom,mdss-pan-dsi-dst-format	指定目标格式。 0 = DSI_VIDEO_DST_FORMAT_RGB565. 1 = DSI_VIDEO_DST_FORMAT_RGB666. 2 = DSI_VIDEO_DST_FORMAT_RGB666_LOOSE. 3 = DSI_VIDEO_DST_FORMAT_RGB888 (Default format) 6 = DSI_CMD_DST_FORMAT_RGB565 7 = DSI_CMD_DST_FORMAT_RGB666 8 = DSI_CMD_DST_FORMAT_RGB888	qcom,mdss-pan-dsi-dst-format = <3>;	根据面板规格定制
qcom,mdss-pan-dsi-vc	指定虚拟信道标识。 0 为默认值。	qcom,mdss-pan-dsi-vc = <0>;	保持不变

qcom,mdss-pan-dsi-rgb-swap	指定 R、G、B 通道排序。 0 = DSI_RGB_SWAP_RGB (默认值) 1 = DSI_RGB_SWAP_RBG 2 = DSI_RGB_SWAP_BGR 3 = DSI_RGB_SWAP_BRG 4 = DSI_RGB_SWAP_GRB 5 = DSI_RGB_SWAP_GBR	qcom,mdss-pan-dsi-rgb-swap = <0>;	根据面板规格定制
qcom,mdss-pan-dsi-data-lanes	指定了启用的 data lane 的数组。 <1 1 0 0>表示 data lane 1 和 data lane 2 已启用（默认）。	qcom,mdss-pan-dsi-data-lanes = <1 1 1 1>;	根据面板规格定制
qcom,mdss-pan-dsi-t-clk	每次模式切换前后，指定字节时钟周期的数组。	qcom,mdss-pan-dsi-t-clk = <0x1b 0x04>;	最初尝试面板调试时，保持不变。如果调试失败，创建 case 进行参数优化
qcom,mdss-pan-dsi-stream	指定要使用的分组流。 0 = stream 0 (默认) 1 = stream 1	qcom,mdss-pan-dsi-stream = <0>;	保持不变
qcom,mdss-pan-dsi-mdp-tr	指定用于 MDP 路径的触发机制 0 = no trigger 2 = Tear check signal line used for trigger 4 = Triggered by software (default mode) 6 = Software trigger and TE	qcom,mdss-pan-dsi-mdp-tr = <0x0>;	保持不变
qcom,mdss-pan-dsi-dma-tr	指定用于 DMA 路径的触发机制 0 = no trigger 2 = Tear check signal line used for trigger 4 = Triggered by software (default mode) 5 = Software trigger and start/end of frame trigger. 6 = Software trigger and TE	qcom,mdss-pan-dsi-dma-tr = <0x04>;	保持不变
qcom,mdss-pan-dsi-frame-rate	指定面板的帧频。 60 表示每秒 60 帧（默认）	qcom,mdss-pan-frame-rate = <60>;	根据面板规格定制

8.2.2 示例

CDP8x26 和 QRD8x26 都使用了 Truly NT35590 720p 面板，可参考的显示屏配置文档为：`kernel\arch\arm\boot\dts\ dsi-panel-nt35590-720p-video.dtsi`。

示例代码如下：

```
/ {
        qcom,mdss_dsi_nt35590_720p_video {
            compatible = "qcom,mdss-dsi-panel";
            label = "nt35590 720p video mode dsi panel";
            status = "disable";
            qcom,dsi-ctrl-phandle = <&mdss_dsi0>;
            qcom,rst-gpio = <&msmgpio 25 0>;
            qcom,mdss-pan-res = <720 1280>;
            qcom,mdss-pan-bpp = <24>;
            qcom,mdss-pan-dest = "display_1";
            qcom,mdss-pan-porch-values = <164 8 140 1 1 6>;
```

```
                qcom,mdss-pan-underflow-clr = <0xff>;
                qcom,mdss-pan-bl-ctrl = "bl_ctrl_wled";
                qcom,mdss-pan-bl-levels = <1 255>;
                qcom,mdss-pan-dsi-mode = <0>;
                qcom,mdss-pan-dsi-h-pulse-mode = <1>;
                qcom,mdss-pan-dsi-h-power-stop = <0 0 0>;
                qcom,mdss-pan-dsi-bllp-power-stop = <1 1>;
                qcom,mdss-pan-dsi-traffic-mode = <2>;
                qcom,mdss-pan-dsi-dst-format = <3>;
                qcom,mdss-pan-dsi-vc = <0>;
                qcom,mdss-pan-dsi-rgb-swap = <0>;
                qcom,mdss-pan-dsi-data-lanes = <1 1 1 1>; /* 4 lanes */
                qcom,mdss-pan-dsi-dlane-swap = <0>;
                qcom,mdss-pan-dsi-t-clk = <0x2c 0x20>;
                qcom,mdss-pan-dsi-stream = <0>;
                qcom,mdss-pan-dsi-mdp-tr = <0x0>;
                qcom,mdss-pan-dsi-dma-tr = <0x04>;
                qcom,mdss-pan-frame-rate = <60>;
                qcom,panel-phy-regulatorSettings = [07 09 03 00  /* Regualotor
        settings */
                                        20 00 01];
                qcom,panel-phy-timingSettings = [7d 25 1d 00 37 33
                                        22 27 1e 03 04 00];
                qcom,panel-phy-strengthCtrl = [ff 06];
                qcom,panel-phy-bistCtrl = [00 00 b1 ff           /* BIST Ctrl
        settings */
                                    00 00];
                qcom,panel-phy-laneConfig = [00 00 00 00 00 00 00 01 97 /* lane0
        config */
                                      00 00 00 00 05 00 00 01 97 /* lane1 config */
                                      00 00 00 00 0a 00 00 01 97 /* lane2 config */
                                      00 00 00 00 0f 00 00 01 97 /* lane3 config */
                                      00 c0 00 00 00 00 00 01 bb]; /* Clk ln config */
                qcom,panel-on-cmds = [29 01 00 00 00 02 FF EE
                                29 01 00 00 00 02 26 08
                                29 01 00 00 00 02 26 00

                                29 01 00 00 00 02 FF 00
                                29 01 00 00 78 02 29 00];

                qcom,on-cmds-dsi-state = "DSI_LP_MODE";
                qcom,panel-off-cmds = [05 01 00 00 32 02 28 00
                                05 01 00 00 78 02 10 00];
                qcom,off-cmds-dsi-state = "DSI_HS_MODE";
            };
```

```
};
```

超级 MSM8x26 设备树包含了带有以下代码片段的显示屏配置，这些代码片段是从 MSM8226-qrd.dts 复制的。

```
/include/ "dsi-panel-nt35590-720p-video.dtsi"

/ {
    model = "Qualcomm MSM 8226 QRD";
    compatible = "qcom,msm8226-qrd", "qcom,msm8226";
    qcom,msm-id = <145 11 0>;

    serial@f991f000 {
         status = "ok";
    };

    qcom,mdss_dsi_nt35590_720p_video {
         status = "ok";
    };
};
```

8.2.3 步骤

1. 在 kernel\arch\arm\boot\dts\ folder 中创建面板配置设备树。比如：dsi-panel-<vendor>-<res>-video.dtsi。
2. 将 dsi-panel-nt35590-720p-video.dtsi 的内容复制到设备树文件 dsi-panel-<vendor>-<res>-video.dtsi。
3. 按如下所示，修改面板 vendor 信息：

```
qcom,mdss_dsi_<vendor>_<res>_video {
     compatible = "qcom,mdss-dsi-panel";
     label = "<vendor> <res> video mode dsi panel";
```

4. 根据面板规范修改基本信息。比如：面板分辨率，bpp。

```
qcom,mdss-pan-res = <720 1280>;
qcom,mdss-pan-bpp = <24>;
```

5. 根据面板规范修改开关指令序列。

```
qcom,panel-on-cmds = [05 01 00 00 78 02 11 00
                      05 01 00 00 78 02 29 00];
qcom,on-cmds-dsi-state = "DSI_LP_MODE";
qcom,panel-off-cmds = [05 01 00 00 32 02 28 00
                      05 01 00 00 78 02 10 00];
```

```
qcom,off-cmds-dsi-state = "DSI_LP_MODE";
```

6. 根据面板规范修改面板 porch 值。

```
qcom,mdss-pan-porch-values = <32 12 144 3 4 9>;
```

7. 根据硬件设置修改面板 GPIO 配置，可能需要修改以下命令：

```
qcom,rst-gpio = <&msmgpio 25 0>;
```

GPIO25 用于面板复位。如改为其他 GPIO，还需修改 arch/arm/mach-msm/board-8226-gpiomux.c。

8. 添加面板设备树到 MSM8626 主设备树。

```
/include/ "dsi-panel-<vendor>-<res>-video.dtsi"
…… ……
    qcom,mdss_dsi@fd922800 {
        qcom,mdss_dsi_<vendor>_<res>_video {
            status = "ok";
        };
};
```

9. 重新构建及加载引导映像，并检查面板是否可亮，如果不能，参考 Q2，进一步调试面板驱动。