Qualcomm Technologies, Inc.

Linux Android PMIC GPIO Software

User Guide

80-P9301-88 Rev. E

June 19, 2018

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

Revision history

Revision	Date	Description
A	July 2017	Initial release
B	August 2017	Updated GPIO HLOS files and locations table, and example procedures for configuring PMIC GPIOs
C	November 2017	Updated configuration examples; added `pinctrl_select_state()` example
D	March 2018	Added support for SDM670
E	June 2018	Added support for SM8150

Contents

1 Introduction

1.1 Purpose

This document describes the general-purpose input/output (GPIO) configuration for the following devices:

- PM845, PMI8998, and PM8005 for the SDM845 chipset
- PM670 and PM670A/PM670L for the SDM670 chipset
- PM8150, PM8150L/A, and PM8150B for the SM8150 chipset

Unless suggested otherwise, use the PM8998 examples as a reference for PM670, PM845, and PM8150 GPIO configuration. For additional GPIO pin information, refer to the following documents:

- *SDM845 PMIC Software Design Review* (80-P9301-59)
- *PM670 Pin Assignment Spreadsheet* (80-PD119-1A)
- *PM670A/PM670L Pin Assignment Spreadsheet* (80-PD120-1A)
- *SM8150 PMIC Software Design Review* (80-PF777-59)

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `#include`.

Code variables appear in angle brackets, for example, <*`number`*>.

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

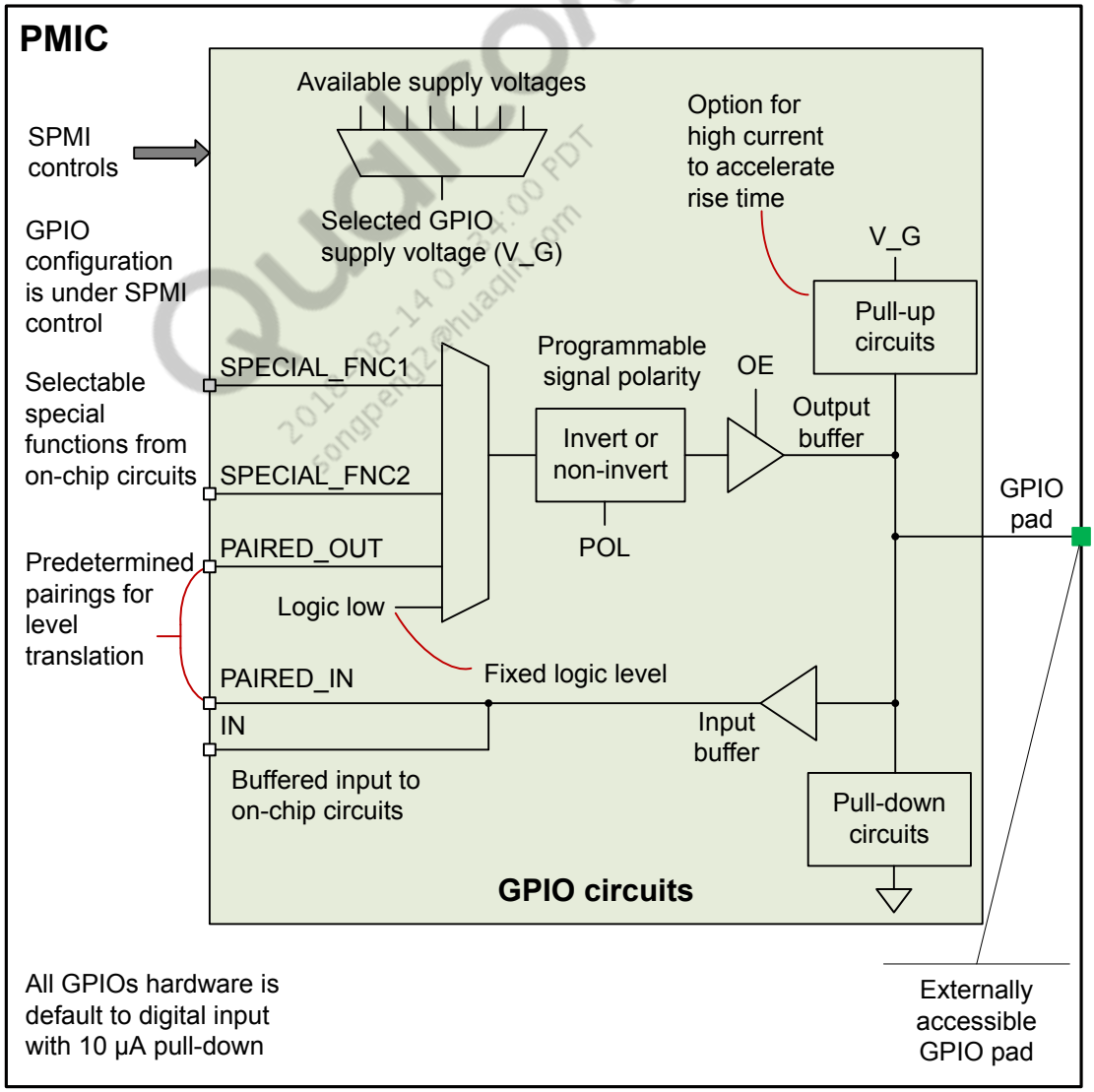
1.3 Technical assistance

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

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

2 Overview

The PMIC for this target has two types of GPIOs to support a low voltage range (GPIO_LV) and a medium voltage range (GPIO_MV). It does not have multipurpose pins (MPPs). The following block diagram describes the PMIC GPIOs.



2.1 PMIC GPIO modes

The PMIC GPIO includes modes configurable for system uses cases and tests.

Configure the unused GPIOs as digital inputs with their internal pull-downs enabled.

Digital Input mode

GPIO is configured as a digital hysteresis input buffer.

GPIO type	Digital Input mode support
GPIO_LV	1.8 V digital signal
GPIO_MV	1.8 V or phone voltage (VPH)

Digital Output mode

GPIO is programmed as a digital output buffer. Output is determined by internal digital signals, special functions, logic 0 or 1, and so on.

GPIO_LV and GPIO_MV have programmable drive strength. Their digital output drivers are configured as one of the following:

- CMOS driver
- Open drain NMOS – PMOS is disabled
- Open drain PMOS – NMOS is disabled
- No Drive mode – Both PMOS and NMOS are disabled

Digital Input and Output mode

Both the digital input hysteresis buffer and digital output buffer are enabled.

The digital input buffer works similar to the Digital Input mode, and the digital output buffer works similar to the Digital Output mode.

Analog Pass-through mode

Analog Pass-through mode access is available in the PM8998 and PMI8998 GPIOs.

NOTE The GPIO does not provide the analog output buffer feature in this mode.

2.2 Special functions

A few GPIOs are internally connected within their respective PMICs to the output signals similar to clocks, or LPG and PWM signals.

NOTE For a list of PM670 special functions and GPIO pins, refer to *PM670 Pin Assignment Spreadsheet* (80-PD119-1A). For a list of PM670A/L special functions and GPIO pins, refer to *PM670A/PM670L Pin Assignment Spreadsheet* (80-PD120-1A).

For a list of PM8150, PM8150L/A, and PM8150B special functions and GPIO pins, refer to *SM8150 PMIC Software Design Review* (80-PF777-59).

These connections are enabled by configuring the respective GPIOs into the Special Function mode. The following tables list the available special function connections:

PM8998 GPIO	Special function 1	Special function 2
1	BUA module	–
13	SLEEP_CLK_2	DIV_CLK_1
14	SLEEP_CLK_2	DIV_CLK_2
16	SLEEP_CLK_2	DIV_CLK_3

PMI8998 GPIO	Special function 1	Special function 2
1	WLED EXT_FET_CTL	–
2	LPG/PWM_1	–
5	LPG/PWM_2	–
6	QNOVO EXT_FET_CTL	–
8	LPG/PWM_6	–

2.3 Register information

EN_CTL register (address – 0x00000046, reset state – 0x80)

Bit	Bit name	Description
7	PERPH_EN	GPIO master enable: ■ 0 – Sets GPIO_PAD to HI-Z and disables the block ■ 1 – Enables GPIO

MODE_CTL register (address – 0x00000040, reset state – 0x00)

Bit	Bit name	Description
1:0	MODE	GPIO mode: ■ 0 – Digital input ■ 1 – Digital output ■ 2 – Digital input and output ■ 3 – Analog pass-through ■ 0x0 – DIG_IN ■ 0x1 – DIG_OUT ■ 0x2 – DIG_INOUT ■ 0x3 – ANA_PASS_THRU

DIG_VIN_CTL register (address – 0x00000041, reset state – 0x00)

Bit	Bit name	Description
2:0	VOLTAGE_SEL	Select voltage source: ■ 000 – VIN0 ■ 001 – VIN1

DIG_OUT_SOURCE_CTL register (address – 0x00000044, reset state – 0x00)

Bit	Bit name	Description
7	OUTPUT_INVERT	Invert the output: ■ 0x0 – NO_INVERT ■ 0x1 – INVERT
3:0	OUTPUT_SOURCE_SEL	Select output source: ■ 0x0 – Low ■ 0x1 – PAIRED_GPIO ■ 0x2 – SPECIAL_FUNCTION1 ■ 0x3– SPECIAL_FUNCTION2 ■ 0x4– SPECIAL_FUNCTION3 (not used) ■ 0x5 – SPECIAL_FUNCTION4 (not used)

3 Software drivers

The XBL loader (SBL) supports the GPIO module.

GPIO XBL files and locations

File type	Location
Header	\boot_images\QcomPkg\Include\api\pmic\pm\pm_gpio.h
Source	\boot_images\QcomPkg\Library\PmicLib\drivers\gpio\src\pm_gpio.c \boot_images\QcomPkg\Library\PmicLib\drivers\gpio\src\pm_gpio_driver.c

UEFI supports the GPIO module via protocol.

GPIO UEFI protocol files and locations

File type	Location
Header	\boot_images\QcomPkg\Include\Protocol\EFIPmicGpio.h
Source	\boot_images\QcomPkg\Drivers\PmicDxe\PmicGpioProtocol.c

The pinctrl-spmi-gpio chip driver is aligned with the SDM845 pinctrl framework.

GPIO HLOS files and locations

File type	Location
DTSI binding doc	/Documentation/devicetree/bindings/pinctrl/qcom,pmic-gpio.txt
Header	/kernel/include/dt-bindings/pinctrl/qcom,pmic-gpio.h
DTSI	/kernel/arch/arm64/boot/dts/qcom/pmi8998.dtsi /kernel/arch/arm64/boot/dts/qcom/pm8998.dtsi /kernel/arch/arm64/boot/dts/qcom/pm8005.dtsi /kernel/arch/arm64/boot/dts/qcom/sdm845-pinctrl.dtsi /kernel/arch/arm64/boot/dts/qcom/sdm845-pinctrl-overlay.dtsi
Source	/kernel/drivers/pinctrl/qcom/pinctrl-spmi-gpio.c

For SM8150 GPIO software related to PM8150, PM8150L/A, and PM8150B, refer to one of the following:

- *SM8150 Non-HLOS PMIC Software Overview* (80-PF777-63)
- *SM8150 Linux Android PMIC Software Overview* (80-PF777-64)

3.1 XBL core functions

The following functions support the PMIC GPIO module in XBL loader (SBL).

For voltage and regulator levels, refer to one of the following:

- *PM670 Power Management Device Specification* (80-PD119-1)
- *PM8998/PM845 Power Management Device Specification* (80-P1086-1)
- *PM8150 Power Management Device Specification* (80-PD995-1)
- *PM8150B Power Management Device Specification* (80-PD996-1)
- *PM8150L/A Power Management Device Specification* (80-PD997-1)

RELATED INFORMATION

"Software drivers" on page 10

3.1.1 pm_gpio_status_get( )

This function returns the status of a PMIC GPIO.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	GPIO	GPIO identifier
out	gpio_status	Pointer to the GPIO status

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The pm_driver_init() function must be executed first.

3.1.2 pm_gpio_config_bias_voltage( )

This function configures the selected GPIO for bias voltage.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	perph_index	GPIO to configure for bias voltage
in	volt_src	GPIO voltage source

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The `pm_driver_init()` function must be executed first.

Example

Configure GPIO_5 bias voltage to VIN0:

```
errFlag = pm_gpio_config_bias_voltage(0, PM_GPIO_5, PM_GPIO_VIN0
);
```

3.1.3 pm_gpio_config_digital_input( )

This function configures the selected GPIO as a digital input.

Parameters

in	`pmic_chip`	Primary PMIC chip – 0; secondary PMIC chip – 1
in	`perph_index`	GPIO to configure as a digital input
in	`i_src_pulls`	Current source pulls
in	`voltage_source`	GPIO voltage source
in	`out_buffer_strength`	GPIO output buffer strength
in	`source`	Selects the source

Returns

SUCCESS or Error (`pm_err_flag_type`)

Dependencies

The `pm_driver_init()` function must be executed first.

Example

- Ensure that the GPIO is available to enable the keypad scan module to drive and sense the keypad.
- Configure the keypad drive signal as an open_drain output with low drive strength.
- Configure the keypad sense module as an input with 1.5 µA pull-up + 30 µA boost. For example, reference voltage VIN0 (1.8 V) for drive and sense lines.
- Configure GPIO_5 as a sense line:

```
errFlag = pm_gpio_config_digital_input(0, PM_GPIO_5,
                              PM_GPIO_I_SOURCE_PULL_UP_1_5uA_PLUS_30uA_BOOST,
                              PM_GPIO_VIN0,
                              PM_GPIO_OUT_BUFFER_OFF,
                              PM_GPIO_SOURCE_GND);
```

3.1.4 pm_gpio_config_digital_output( )

This function configures the selected GPIO as a digital output.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	perph_index	GPIO configures as DO
in	out_buffer_config	GPIO output buffer configuration
in	volt_src	GPIO voltage source
in	src	Selects the source
in	out_buffer_strength	GPIO output buffer strength
in	out_inversion	Inverts the EXT_PIN output

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The pm_driver_init() function must be executed first.

Example

- Ensure that the GPIO is available to enable the keypad scan module to drive and sense the keypad.
- Configure the keypad drive signal as an open_drain output with low drive strength.
- Configure the keypad sense module as an input with 1.5 µA pull-up + 30 µA boost. For example, reference voltage VIN0 (1.8 V) for drive and sense lines.
- Configure GPIO_5 as a drive signal:

```
errFlag = pm_gpio_config_digital_output(0, PM_GPIO_5,
                                        PM_GPIO_OUT_BUFFER_CONFIG_OPEN_DRAIN,
                                        PM_GPIO_VIN0,
                                        PM_GPIO_SOURCE_SPECIAL_FUNCTION1,
                                        PM_GPIO_OUT_BUFFER_LOW,
                                        FALSE);
```

3.1.5 pm_gpio_config_digital_input_output( )

This function configures the selected GPIO as bidirectional.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	perph_index	GPIO to configure as a digital output
in	src	Selects the source

in	i_src_pulls	Current source pulls
in	volt_src	GPIO voltage source
in	out_buffer_config	GPIO output buffer configuration
in	out_buffer_strength	GPIO output buffer strength

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The pm_driver_init() function must be executed first.

3.1.6 pm_gpio_set_voltage_source( )

This function sets the voltage source.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	perph_index	GPIO configures the voltage source
in	voltage_source	GPIO voltage source

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The pm_driver_init() function must be executed first.

Example

Set voltage source to VIN0 for GPIO_5:

```
errFlag = pm_gpio_set_voltage_source(0, PM_GPIO_5, PM_GPIO_VIN0);
```

3.1.7 pm_gpio_set_output_buffer_configuration( )

This function sets the output buffer configuration.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	perph_index	GPIO for which the current source pulls are configured
in	out_buffer_config	GPIO output buffer configuration

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The `pm_driver_init()` function must be executed first.

Example

For GPIO_5, set the output buffer configuration to CMOS in Bias mode:

```
errFlag = pm_gpio_set_output_buffer_configuration(0, PM_GPIO_5,
                                                  PM_GPIO_OUT_BUFFER_CONFIG_CMOS);
```

3.1.8 pm_gpio_set_inversion_configuration( )

This function sets the inversion configuration.

Parameters

in	`pmic_chip`	Primary PMIC chip – 0; secondary PMIC chip – 1
in	`perph_index`	GPIO for which the current source pulls are configured
in	`inversion`	TRUE – Enable inversion; FALSE – Disable inversion

Returns

SUCCESS or Error (`pm_err_flag_type`)

Dependencies

The `pm_driver_init()` function must be executed first.

Example

For GPIO_5, set inversion to TRUE:

```
errFlag = pm_gpio_set_inversion_configuration(0, PM_GPIO_5, TRUE);
```

3.1.9 pm_gpio_set_output_buffer_drive_strength( )

This function sets the output buffer drive strength.

Parameters

in	`pmic_chip`	Primary PMIC chip – 0; secondary PMIC chip – 1
in	`perph_index`	GPIO for which the output buffer drive strength is configured
in	`out_buffer_strength`	GPIO output buffer drive strength

Returns

SUCCESS or Error (`pm_err_flag_type`)

Dependencies

The `pm_driver_init()` function must be executed first.

Example

For GPIO_5, set the output buffer drive strength to HIGH:

```
errFlag = pm_gpio_set_output_buffer_drv_strength(0, PM_GPIO_5,
                                                 PM_GPIO_OUT_BUFFER_HIGH);
```

3.1.10 pm_gpio_set_source_configuration( )

This function sets the source configuration.

Parameters

in	pmic_chip	Primary PMIC chip – 0; secondary PMIC chip – 1
in	perph_index	GPIO for which the source configuration is selected
in	src	Selects the source

Returns

SUCCESS or Error (pm_err_flag_type)

Dependencies

The pm_driver_init() function must be executed first.

Example

Select Pair In as the source to serve GPIO_5 as an output in the Level Translator mode:

```
errFlag = pm_gpio_set_src_configuration(0, PM_GPIO_5,
                                        PM_GPIO_SOURCE_PAIRED_GPIO);
```

3.2 Use UEFI protocol to control GPIO

The UEFI protocol can be used to control GPIO. This code example shows how to configure GPIO_3 to a digital output low or high with 1.8 V.

1. In the related .inf file, add gQcomPmicGpioProtocolGuid under [Protocols]:

```
[Protocols]
 gQcomPmicGpioProtocolGuid
```

2. In the driver file, add the following #include for the header file:

```
#include <Protocol/EFIPmicGpio.h>
```

3. Locate the PMIC GPIO protocol.

```
EFI_QCOM_PMIC_GPIO_PROTOCOL *PmicGpioProtocol = NULL;
Status = gBS->LocateProtocol(&gQcomPmicGpioProtocolGuid, NULL,(VOID**)
&PmicGpioProtocol))
if (Status != EFI_SUCCESS )
{
//Error Handling
}
```

4. Configure PMIC GPIO_3 to output 0 V (logic low).

```
//arg 1: PMIC Index, first PMIC (PM8998) index=0
//arg 2: GPIO, set to GPIO3
//arg 3: voltage source: VIN0 sourced by S3(1.8V)
//arg 4: Logic source: GND for general purpose
//arg 5: Buffer strength MEDIUM
//arg 6: Invert output or not. FALSE = don't invert GND signal
Status = PmicGpioProtocol->ConfigDigitalOutput(
0,
EFI_PM_GPIO_3,
EFI_PM_GPIO_OUT_BUFFER_CONFIG_CMOS,
EFI_PM_GPIO_VIN0,
EFI_PM_GPIO_SOURCE_GND,
EFI_PM_GPIO_OUT_BUFFER_MEDIUM,
FALSE ))
if (Status != EFI_SUCCESS )
{
//Error Handling
}
```

5. Configure PMIC GPIO_3 to 1.8 V output (logic high).

```
//arg 1: PMIC Index, first PMIC (PM8998) index=0
//arg 2: GPIO, set to GPIO3
//arg 3: voltage source
//arg 4: Logic source: GND for general purpose
//arg 5: Buffer strength MEDIUM
//arg 6: Invert output or not. TRUE = invert GND signal
Status = PmicGpioProtocol->ConfigDigitalOutput(
0,
EFI_PM_GPIO_3,
EFI_PM_GPIO_OUT_BUFFER_CONFIG_CMOS,
EFI_PM_GPIO_VIN0,
EFI_PM_GPIO_SOURCE_GND,
EFI_PM_GPIO_OUT_BUFFER_MEDIUM,
TRUE );
if (Status != EFI_SUCCESS )
{
//Error Handling
}
```

RELATED INFORMATION

"Software drivers" on page 10

3.3 Kernel

The following parameter tables describe the kernel DTSI properties required to configure the root node and child node.

3.3.1 Required root node properties

Parameter	Description
`compatible`	Must be qcom,spmi-gpio to specify the GPIO driver
`reg`	Register base of the GPIO block and length
`interrupts`	Must contain an array of encoded interrupt specifiers for each available GPIO
`gpio-controller`	Marks the device node as a GPIO controller
`#gpio-cells`	A PMIC pin number and a 32-bit flag field are encoded to specify the GPIO configuration; set the value to 2

3.3.2 Optional root node properties

Parameter	Description
`qcom,gpios-disallowed`	■ Array of the GPIO hardware numbers corresponding to GPIOs which the APSS processor is not allowed to configure ■ Hardware numbers are indexed beginning at 1 ■ Interrupt resources for these GPIOs must not be defined in the `interrupts` and `interrupt-names` properties ■ GPIOs defined in this array are not registered as pins in the pinctrl device or GPIOs in the GPIO chip

3.3.3 Required child node properties

Parameter	Description	Possible values
`pins`	List of GPIO pins affected by the properties specified in this child node	■ GPIO1 – GPIO_26 for PM8998 ■ GPIO1 – GPIO_14 for PMI8998
`function`	Specifies the alternative function to be configured for the specified pins	■ Valid values: □ normal – Constant output □ paired □ func1 – Special function 1 □ func2 – Special function 2 □ dtest1 □ dtest2 □ dtest3 □ dtest4 ■ Values supported by LV/MV GPIO subtypes: □ func3 □ func4 □ analog

3.3.4 Optional child node properties

Parameter	Description	Possible values
bias-disable	Specified pins must be configured as no pull.	–
bias-pull-down	Specified pins must be configured as pull down.	–
bias-pull-up	Specified pins must be configured as pull up.	–
qcom,pull-up-strength	If selected, this property specifies the strength to use for the pull up.	■ 0 – 30 µA (PMIC_GPIO_PULL_UP_30) ■ 1 – 1.5 µA (PMIC_GPIO_PULL_UP_1P5) ■ 2 – 31.5 µA (PMIC_GPIO_PULL_UP_31P5) ■ 3 – 1.5 µA + 30 µA boost (PMIC_GPIO_PULL_UP_1P5_30) If this property is omitted, 30 µA (default) strength is used if the pull up is selected.
bias-high-impedance	Specified pins are put in High-Z mode and disabled.	–
input-enable	Specified pins are put in Input mode.	–
output-high	Specified pins are configured in output mode, inverted output. **Note:** If set to normal function with output-high, the GPIO outputs inverted GND, which is constant HIGH. This setting can be used to invert other outputs, e.g., special function outputs.	–
output-low	Specified pins are configured in output mode, noninverted output. **Note:** If set to the normal function with output-low, the GPIO outputs noninverted GND, which is constant LOW. This setting can be used to keep other outputs, such as special function signals, noninverted.	–
power-source	Selects the power source for the specified pins	■ 0 – VIN0 □ 1.8 V for PM8998 GPIO_1 to 18 □ 3.6 V for PM8998 GPIO_19 to 26 □ 1.8 V for PMI8998 GPIO_2, 5, 9, 10, 12, and 14 □ 3.6 V for PMI8998 GPIO_1, 3, 4, 6, 7, 8, 11, and 13 ■ 1 – VIN1 □ 1.8 V for PM8998 GPIO_19 to 26 □ 1.8 V for all PMI8998 GPIOs

Parameter	Description	Possible values
qcom,drive-strength	Selects the drive strength for the specified pins	Value drive strengths: ■ 0 – None (PMIC_GPIO_STRENGTH_NO) ■ 1 – HIGH (PMIC_GPIO_STRENGTH_HIGH) 0.9 mA at 1.8 V - 1.9 mA at 2.6 V ■ 2 – MEDIUM (PMIC_GPIO_STRENGTH_ MED) 0.6 mA at 1.8V - 1.25 mA at 2.6 V ■ 3 – LOW (PMIC_GPIO_STRENGTH_LOW) 0.15 mA at 1.8 V - 0.3 mA at 2.6 V
drive-push-pull	Specified pins are configured in push-pull mode. **Note:** CMOS configuration for digital output.	–
drive-open-drain	Specified pins are configured in open-drain mode. **Note:** NMOS configuration for digital output.	–
drive-open-source	Specified pins are configured in open-source mode. **Note:** PMOS configuration for digital output.	–
qcom,atest	Selects the ATEST rail to route to GPIO when it is configured in analog-pass-through mode by specifying the analog option	■ 0 – PMIC_GPIO_AOUT_ATEST1 ■ 1 – PMIC_GPIO_AOUT_ATEST2 ■ 2 – PMIC_GPIO_AOUT_ATEST3 ■ 3 – PMIC_GPIO_AOUT_ATEST4
qcom,dtest-buffer	Selects the DTEST rail to route to GPIO when configured as a digital input	■ 0 – PMIC_GPIO_DIN_DTEST1 ■ 1 – PMIC_GPIO_DIN_DTEST2 ■ 2 – PMIC_GPIO_DIN_DTEST3 ■ 3 – PMIC_GPIO_DIN_DTEST4

3.4 Configuration examples

The following examples describe how to configure PMIC GPIOs.

3.4.1 Configure PM8998 GPIO_16 as driver-controllable digital output HIGH or LOW

1. Configure GPIO_16 as digital output.

```
&pm8998_gpios {
    gpio16_dig_out {
        gpio16_dig_out_default: gpio16_dig_out_default {
            pins = "gpio16"; /* GPIO 16 */
            function = "normal"; /* normal output */
            power-source = <0>; /* VIN0 */
            output-low; /* digital output, no invert */
            input-disable; /* prevent GPIO from being set to DIO */
```

```
        };
    };
};
```

2. Add GPIO_16 to the client DTSI.

```
test_func {
        compatible = "qcom,test";
        pinctrl-names = "default";
        pinctrl-0 = <&gpio16_dig_out_default>;
        test-gpios = <&pm8998_gpios 16 GPIO_ACTIVE_LOW>;
    };
```

3. Control GPIO_16 in the client driver.

```
int gpio16_test;
gpio16_test = of_get_named_gpio(dev.of_node,"test-gpios", 0);
gpio_request(gpio16_test, "GPIO16");
gpio_direction_output(gpio16_test, 1); //Output HIGH
gpio_direction_output(gpio16_test, 0); //Output LOW
```

3.4.2 Configure PM8998 GPIO_4 as interrupt wake-up resources with a falling-edge trigger

1. Modify the GPIO status.

```
&pm8998_gpios {
    wake_in {
        wake_in_default: wake_in_default {
            pins = "gpio4"; /* GPIO 4 */
            function = "normal"; /* normal input */
            bias-pull-up; /* enable pull up */
            qcom,pull-up-strength = <0>; /* 30uA pull up */
            power-source = <0>; /* VIN0 */
            input-enable /* digital input */
        };
    };
};
```

2. Claim GPIO as an interrupt in the driver-related DTSI file.

```
test_func {
    compatible = "qcom,test-gpio";
    interrupt-parent = <&spmi_bus>;
    test1,irq-gpio = <&pm8998_gpios 4 GPIO_ACTIVE_LOW>;
    pinctrl-names = "default";
    pinctrl-0 = <&wake_in_default>;
};
```

3. Configure driver usage.

```
int test_gpio, test_irq;
test_gpio = of_get_named_gpio(pdev,"test1,irq-gpio", 0);
```

```
test_irq = gpio_to_irq(test_gpio);
rc = request_irq(test_irq, test_gpio_irq_handler, IRQ_TYPE_EDGE_FALLING,
"gpio_test_1", NULL);
enable_irq_wake(test_irq);
```

4. Handle a triggered interrupt.

```
static irqreturn_t
test_gpio_irq_handler(int irq, void *dev_id)
{
//do what would you like to do

return IRQ_HANDLED;
}
```

RELATED INFORMATION

"Software drivers" on page 10

3.4.3 Configure PM8998 GPIO_16 to 32 kHz clock

1. Configure GPIO_16 to special function 1 for 32 kHz clock.

```
&pm8998_gpios {
   clk_out {
        clk_out_default: clk_out_default {
            pins = "gpio16"; /* GPIO 16 */
            function = "func1"; /* Special Function 1 */
            bias-disable; /* No Pull */
            power-source = <0>; /* VIN0 */
            output-low; /* digital output, no invert */
          input-disable; /* prevent GPIO from being set to DIO */
        };
   };
};
```

2. Add pinctrl-names and pinctrl-0 in the consumer node. For the pinctrl framework to configure GPIO_16, the consumer node must use the default GPIO node. It is unnecessary to explicitly enable the GPIO with the `gpio_direction_output()` function.

```
test_func {
…
…
        pinctrl-names = "default";
        pinctrl-0 = <&clk_out_default>;
   };
```

3.4.4 Configure PMI8998 GPIO_5 PWM for blink LEDs

1. Configure PMI8998 GPIO_5 to output PWM from LPG2.

```
&pmi8998_gpios {
    pwm_out {
        pwm_out_default: pwm_out_default {
            pins = "gpio5"; /* GPIO 5 */
            function = "func1"; /* Special Function 1 */
            bias-disable; /* No Pull */
            power-source = <0>; /* VIN0 */
            output-low; /* digital output, no invert */
            qcom,drive-strength = <3>; /* LOW strength */
            drive-push-pull; /* CMOS */
        };
    };
};
pmi8998_pwm_2: pwm@b200 {
    compatible = "qcom,qpnp-pwm";
    reg = <0xb200 0x100>,
            <0xb042 0x7e>;
    reg-names = "qpnp-lpg-channel-base",
            "qpnp-lpg-lut-base";
    qcom,channel-id = <2>;
    qcom,supported-sizes = <6>, <9>;
    qcom,ramp-index = <1>;
    qcom,force-pwm-size = <9>;
    qcom,mode-select = <0>; /* PWM */
    qcom,period = <100000>;
    #pwm-cells = <2>;
    status = "okay";
    qcom,pwm {
        qcom,duty = <50000>;
        label = "pwm";
    }
};
```

2. Add pinctrl-names and pinctrl-0 in the consumer node. For the pinctrl framework to configure PMI8998 GPIO_5 at runtime, the consumer node must use the default GPIO node. It is unnecessary to explicitly enable the GPIO with the `gpio_direction_output()` function.

```
test_func {
…
…
        pinctrl-names = "default";
        pinctrl-0 = <&pwm_out_default>;
   };
```

3.4.5 Configure PM8998 GPIO_9 as an ADC

This example shows how to configure the PM8998 GPIO as analog input and set up ADC read via the device tree with the following settings.

Prerequisites:

- GPIO – 9
- Channel name – gpio9_adc
- Input voltage range – 0 to 3 V
- Calibration type – Absolute
- No interpolation to other units
- 100 µs delay (recommended for GPIO channels)
- Average 4 ADC samples

1. Ensure that the GPIO is disabled in the sdm845-pinctrl.dtsi file and that it is not being used by other processors.

```
&pm8998_gpios {
    gpio9_adc {
        gpio9_adc_default: gpio9_adc_default {
            pins = "gpio9"; /* GPIO 9 */
            function = "normal"; /* normal */
            bias-high-impedance; /* DISABLE GPIO9 for ADC*/
        };
    };
};
```

2. Add pinctrl-names and pinctrl-0 in the consumer node. For the pinctrl framework to configure PM8998 GPIO_9 at runtime, the consumer node must use the default GPIO node. It is unnecessary to explicitly enable the GPIO using the `gpio_direction_output()` function.

```
Ensure that the GPIO is disabled in the sdm845-pinctrl.dtsi file and that
it isclient_node {
…
…
        pinctrl-names = "default";
        pinctrl-0 = <&gpio9_adc_default>;
   };
```

3. Set the ADC channel for GPIO in the msm-pmcobalt.dtsi file.

```
chan@13 {
    label = "gpio9_adc";
    reg = <0x13>; // channel for GPIO9
    qcom,decimation = <2>;
    qcom,pre-div-channel-scaling = <0>; //1:1 scaling
    qcom,calibration-type = "absolute";
```

```
    qcom,scale-function = <2>;
    qcom,hw-settle-time = <2>;
    qcom,fast-avg-setup = <0>;
};
```

4. Add to the client node that needs the VADC channel A/D.

```
client_node {
    qcom,test-vadc = <&pm8998_vadc>;
};
```

NOTE The consumer name passed to the driver when calling the `qpnp_get_vadc()` function is used to associate the client with the corresponding device.

5. Associate the client with the corresponding device and get the device structure.

```
struct qpnp_vadc_chip *vadc_dev;
vadc_dev = qpnp_get_vadc(chip->dev, "test");
```

6. Read the VADC channel using the QPNP ADC API.

```
struct qpnp_vadc_result result;
err = qpnp_vadc_read(vadc_dev, P_MUX2_1_1, &result); //Read the GPIO9 VADC
channel with 1:1 scaling
*adc = (int) result.physical;
*adc = *adc / 1000; /* uV to mV */
```

RELATED INFORMATION
"Software drivers" on page 10

3.4.6 Configure pinctrl_select_state()

GPIO is part of the pinctrl framework and as such can be controlled by pinctrl APIs. This can be useful in situations where a GPIO needs to be dynamically controlled ON or OFF with a special function source. In this example, GPIO_16 is set to be enabled as special function 2 to output DIV_CLK3, and can be disabled to High-Z.

1. Create multiple states for the GPIO in sdm845-pinctrl.dtsi for ENABLED and DISABLED states.

```
&pm8998_gpios {
    gpio16_div_clk_test {
        gpio16_div_clk_test_enable: gpio16_div_clk_test_enable {
            pins = "gpio16"; /* GPIO 16 */
            function = "func2"; /* special function 2 */
            bias-disable; /* disable pull up & down */
            power-source = <0>;
            output-low; /* digital output, no invert */
            input-disable; /* prevent GPIO from being set to DIO */
        };
        gpio16_div_clk_test_disable: gpio16_div_clk_test_disable {
            pins = "gpio16"; /* GPIO 16 */
            function = "normal"; /* normal */
            power-source = <0>;
            bias-high-impedance; /* DISABLE GPIO */
```

```
        };
    };
};
```

2. Add GPIO_16 to client DTSI node.

```
test_func {
        compatible = "qcom,test";
        pinctrl-names = "enable", "disable";
        pinctrl-0 = <&gpio16_div_clk_test_enable>;
        pinctrl-1 = <&gpio16_div_clk_test_disable>;
};
```

3. In the client driver's code, add the following:

```
struct pinctrl *pctrl;
struct pinctrl_state *pins_enable;
struct pinctrl_state *pins_disable;
int ret;
pctrl = devm_pinctrl_get(dev);
if (IS_ERR(pctrl)) {
    dev_err(dev, "failed to get pinctrl\n");
     return PTR_ERR(pctrl);
};
pins_enable = pinctrl_lookup_state(pctrl, "enable");
if (IS_ERR(pins_enable)) {
    ret = PTR_ERR(pins_enable);
    dev_err(dev, "Could not get active pinstates, err:%d\n", ret);
    return PTR_ERR(pins_enable);
};
pins_disable = pinctrl_lookup_state(pctrl, "disable");
if (IS_ERR(pins_disable)) {
    ret = PTR_ERR(pins_disable);
    dev_err(dev, "Could not get sleep pinstates, err:%d\n", ret);
    return PTR_ERR(pins_disable);
};
```

4. In the client driver's code, dynamically enable or disable GPIO_16 as special function 2:

```
if (gpio_enable)
    ret = pinctrl_select_state(pctrl, pins_enable);
else
    ret = pinctrl_select_state(pctrl, pins_disable);
```

4 Debug

To debug GPIOs, collect GPIO register dumps via ADB shell or JTAG.

4.1 Collect register dumps via ADB shell

To collect GPIO PM8998 register dumps:

```
cd /sys/kernel/debug/regmap/spmi0-00
echo 0xC000> address
echo 0x2000> count
cat data
```

To collect GPIO PM8I998 register dumps:

```
cd /sys/kernel/debug/regmap/spmi0-02
echo 0xC000> address
echo 0xDFF> count
cat data
```


4.2 Collect register dumps via JTAG

1. To run the JTAG tool, type the following command on any T32 window (Apps or AOP T32 are recommended):

   ```
   do pmicdump.cmm
   ```

   GLUE build location:

   \Common\Core\tools\tools\systemdrivers\pmic\hoya

   NOTE To run the script from the AOP T32 window, first attach to the Apps T32 window via `sys.m.a` command.

   - It is unnecessary to modify the following settings irrespective of the T32 window the script is being run from:
     - Owner – DEFAULT
     - AccessMode – EZAXI
   - If CX is collapsed or in retention during sleep, modify the following:
     - Owner – AOP
     - AccessMode – A

2. Choose one of the following dump options:
   - Click **Dump** for a full dump of all PMICs on the target.
   - Click **Dump + Analyze** for a dump from all PMICs on the target and parsed PMIC dump output.

   Optional:

   a. Select **Log File** to save logs to the default or custom location.
   b. Deselect **Break on Dump** to collect PMIC dump while T32 is running.
3. Collect the dump:
   a. Deselect **Default Settings**.
   b. Select the PMIC option (PMIC A, PMIC B, etc.) for which the dump must be collected.
   c. Click **Collect Dump**. Output is similar to the following example:

```
Collecting PM8998v1.1 pmic register dump...
Collecting PMI8998v1.1 pmic register dump...
Collecting PM8005v1.1 pmic register dump...
PMIC register dump sent to c:\temp\pmicdump.xml
Parsing PM8998v1.1 pmic register dump...
PMIC register dump parsed to c:\temp\pmicdump.xml.PM8998.txt
Parsing PMI8998v1.1 pmic register dump...
PMIC register dump parsed to c:\temp\pmicdump.xml.PMI8998.txt
Parsing PM8005v1.1 pmic register dump...
PMIC register dump parsed to c:\temp\pmicdump.xml.PM8005.txt
```

Postrequisites:

For details on additional JTAG/T32 script tools, see *SDM845 Non-HLOS PMIC Software Overview* (80-P9301-26) or *SDM670 Non-HLOS PMIC Software Overview* (80-PD126-24).

4.3 Access GPIO control through sysfs

Use the following procedure to access PMIC GPIOs via ADB.

1. Read the gpiochip and the corresponding GPIO label.

   Labels identify which gpiochip* represents the pinctrl blocks. In the following example, PM8998 pinctrl is second in the list, followed by PM8005 and PMI8998. These labels correspond to gpiochip1005, gpiochip992, and gpiochip994.

```
cd /sys/class/gpio
cat gpiochip*/label
3400000.pinctrl
c440000.qcom,spmi:qcom,pm8998@0:pinctrl@c000
qcom-wcd-pinctrl
ipa
ipa
master-kernel
slave-kernel
master-kernel
slave-kernel
master-kernel
```

```
slave-kernel
master-kernel
slave-kernel
sleepstate
smp2p
smp2p
smp2p
smp2p
smp2p
smp2p
smp2p
smp2p
smp2p
smp2p
c440000.qcom,spmi:qcom,pm8005@4:pinctrl@c000
c440000.qcom,spmi:qcom,pmi8998@2:pinctrl@c000
ls
export
gpiochip0
gpiochip1005
gpiochip315
gpiochip320
gpiochip352
gpiochip384
gpiochip416
gpiochip448
gpiochip480
gpiochip512
gpiochip544
gpiochip576
gpiochip608
gpiochip640
gpiochip672
gpiochip704
gpiochip736
gpiochip768
gpiochip800
gpiochip832
gpiochip864
gpiochip896
gpiochip928
gpiochip960
gpiochip992
gpiochip994
unexport
PM8998 pinctrl is gpiochip1005
```

```
PM8005 pinctrl is gpiochip992
PMI8998 pinctrl is gpiochip994
```

NOTE GPIOs disallowed in DTSI are not populated in the gpiochip list.

- PMI8998 default DTSI disallows GPIOs 4, 7, and 13
- PM8998 default DTSI disallows GPIOs 3, 15, 20, 22, 24, 25, and 26
- PM8005 default DTSI disallows GPIO_3 and 4

2. Export the GPIO to control it dynamically. The following is an example for PMI8998 GPIO_5:

```
echo 997 > export
ls
export
gpio997
gpiochip0
gpiochip1005
…
…
cd gpio997
ls
active_low  device  direction  edge  power  subsystem  uevent  value
```

NOTE If a kernel-level driver obtains, requests, and uses a GPIO (via `of_get_named_gpio()` function or similar), the GPIO cannot be exported for sysfs control.

3. Assign the GPIO direction and output.

```
cd gpio997
echo out > direction
cat direction
out

echo 1 > value
```

RELATED INFORMATION
"GPIO number-to-gpiochip mapping" on page 32

4.4 Check runtime GPIO status

GPIO status can be checked in /d/pinctrl.

```
sdm845: # cd d/pinctrl
sdm845:/d/pinctrl # ls
3400000.pinctrl                              pinctrl-devices
c440000.qcom,spmi:qcom,pm8005@4:pinctrl@c000  pinctrl-handles
c440000.qcom,spmi:qcom,pm8998@0:pinctrl@c000  pinctrl-maps
c440000.qcom,spmi:qcom,pmi8998@2:pinctrl@c000  qcom-wcd-pinctrl
sdm845:/d/pinctrl # cd c440000.qcom,spmi:qcom,pmi8998@2:pinctrl@c000
sdm845:/d/pinctrl/c440000.qcom,spmi:qcom,pmi8998@2:pinctrl@c000 # ls
gpio-ranges     pinconf-groups  pingroups         pinmux-pins
```

```
pinconf-config  pinconf-pins    pinmux-functions  pins
sdm845:/d/pinctrl/c440000.qcom,spmi:qcom,pmi8998@2:pinctrl@c000 # cat pins
registered pins: 11
pin 0 (gpio1)
pin 1 (gpio2)
pin 2 (gpio3)
pin 3 (gpio5)
pin 4 (gpio6)
pin 5 (gpio8)
pin 6 (gpio9)
pin 7 (gpio10)
pin 8 (gpio11)
pin 9 (gpio12)
pin 10 (gpio14)
sdm845:/d/pinctrl/c440000.qcom,spmi:qcom,pmi8998@2:pinctrl@c000 # cat pinconf-
groups
Pin config settings per pin group
Format: group (name): configs
0 (gpio1): input bias disabled, input bias high impedance, input bias pull
down, input bias pull up, output drive open drain, output drive open source,
output drive push pull, input enabled, pin output (0 level), pin power source
(0 selector), pull up strength (5), drive-strength (3), atest (0) gpio1 :
out  func1   vin
-0 no pull                    push-pull  low  low
1 (gpio2): input bias disabled, input bias high impedance, input bias pull
down, input bias pull up, output drive open drain, output drive open source,
output drive push pull, input enabled, pin output (0 level), pin power source
(0 selector), pull up strength (5), drive-strength (1), atest (0) gpio2 :
out  normal  vin
-0 no pull                    push-pull  low  high
2 (gpio3): input bias disabled, input bias high impedance, input bias pull
down, input bias pull up, output drive open drain, output drive open source,
output drive push pull, input enabled, pin output (0 level), pin power source
(0 selector), pull up strength (5), drive-strength (1), atest (0) gpio3 :
in   normal  vin
-0 no pull                    push-pull  low  high
3 (gpio5): input bias disabled, input bias high impedance, input bias pull
down, input bias pull up, output drive open drain, output drive open source,
output drive push pull, input enabled, pin output (1 level), pin power source
(0 selector), pull up strength (5), drive-strength (1), atest (0) gpio5 :
out  normal  vin
-0 no pull                    push-pull  high
```

A GPIO number-to-gpiochip mapping

The examples listed are based on default settings. Corresponding gpiochip and GPIO numbers are different if the list of disallowed GPIOs is different. For the most up-to-date list of disallowed GPIOs, refer to the DTSI.

Example PM8005 GPIO gpiochip mapping

gpiochip	PM8005 GPIO number
992	1
993	2

Example PM8998 GPIO gpiochip mapping

gpiochip	PM8998 GPIO number
1005	1
1006	2
1007	4
1008	5
1009	6
1010	7
1011	8
1012	9
1013	10
1014	11
1015	12
1016	13
1017	14
1018	16
1019	17
1020	18
1021	19
1022	21
1023	23

Example PMI8998 GPIO gpiochip mapping

gpiochip	PMI8998 GPIO number
994	1
995	2
996	3
997	5
998	6
999	8
1000	9
1001	10
1002	11
1003	12
1004	14

B References

B.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
PM670 Pin Assignment Spreadsheet	80-PD119-1A
PM670A/PM670L Pin Assignment Spreadsheet	80-PD120-1A
PM670 and PM670A/PM670L Power Management ICS Design Guidelines	80-PD119-5A
PM8998, PM8005, and PMI8998 Power Management ICS Design Guidelines	80-P1086-5
PM8150 Power Management IC Design Guidelines	80-PD995-5
PM8150B Power Management IC Design Guidelines	80-PD996-5
PM8150L/A Power Management IC Design Guidelines	80-PD997-5
SDM670 Linux Android PMIC Software Overview	80-PD126-4
SDM845 Linux Android PMIC Software Overview	80-P9301-30
SM8150 Linux Android PMIC Software Overview	80-PF777-64
SDM670 Non-HLOS PMIC Software Overview	80-PD126-24
SDM845 Non-HLOS PMIC Software Overview	80-P9301-26
SM8150 Non-HLOS PMIC Software Overview	80-PF777-63
Non-HLOS ADC PMIC Software User Guide	80-P2484-71
Linux Android PMIC ADC Software User Guide	80-P2484-78
PM670 Hardware Register Description	80-PD119-2X
PM8998/PM845 Hardware Register Description	80-P1086-2X
PMI8998 Hardware Register Description	80-P1087-2X
PM670 Power Management Device Specification	80-PD119-1
PM8998/PM845 Power Management Device Specification	80-P1086-1
PM8150 Device Specification	80-PD995-1
PM8150B Device Specification	80-PD996-1
PM8150L/A Device Specification	80-PD997-1

B.2 Acronyms and terms

Acronym or term	Definition
ADB	Android debug bridge
ADC	Analog-to-digital conversion
AI or AIN	Analog input
AO or AOUT	Analog output
CS	Current sink
DO	Digital output
FET	Field effect transistor
GPIO	General-purpose input/output
LDO	Low drop out
LV	Low voltage
MPP	Multipurpose pin
MV	Medium voltage
PWM	Pulse width modulation
SBL	Secondary boot loader
VPH	Phone voltage