Qualcomm Technologies, Inc.

PMIC APSS Interface Specification and Operational Description

September 2016

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

LM80-P0598-11 Rev B

Revision history

Revision	Date	Description
B	September 2016	Update for ‘E’ part
A	June 1, 2015	Initial release

Contents

1 Introduction 7

1.1 Purpose 7
1.2 Acronyms, abbreviations, and terms 7
1.3 Scope 8
1.4 Conventions 8

2 Regulator API 9

2.1 Introduction 9
2.2 regulator_get 9
2.3 regulator_put 9
2.4 regulator_enable 10
2.5 regulator_disable 10
2.6 regulator_is_enabled 11
2.7 regulator_set_voltage 11
2.8 regulator_get_voltage 11
2.9 regulator_set_optimum_mode 12
2.10 Regulator configuration 12
2.10.1 Regulator declaration and constraints 12
2.10.2 Regulator consumer supply list 14
2.10.3 Regulator supply chain 14

3 GPIO API 15

3.1 Introduction 15
3.2 gpio_request 15
3.3 gpio_free 15
3.4 gpio_set_value_cansleep 16
3.5 gpio_get_value_cansleep 16
3.6 gpio_to_irq 17
3.7 pm8xxx_gpio_config 17

4 MPP API 18

4.1 Introduction 18
4.2 pm8xxx_mpp_config 18

5 PMIC IRQ API 19

5.1 Introduction 19
5.2 request_threaded_irq 19
5.3 free_irq 20
5.4 enable_irq 20
5.5 disable_irq 20
5.6 irq_set_irq_wake 21
5.7 pm8xxx_get_irq_stat 21

6 PMIC Charger API 23

6.1 Introduction 23
6.2 pm8xxx_charger_enable 23

6.3 pm8xxx_is_usb_chg_plugged_in 23
6.4 pm8xxx_is_dc_chg_plugged_in 24
6.5 pm8xxx_is_battery_present 24
6.6 pm8xxx_disable_input_current_limit 25
6.7 pm8xxx_set_usb_power_supply_type 25
6.8 pm8xxx_disable_source_current 26
6.9 pm8xxx_regulate_input_voltage 26
6.10 pm8xxx_is_battery_charging 27
6.11 pm8xxx_batt_temperature 27
6.12 pm8xxx_usb_ovp_set_threshold 27
6.13 pm8xxx_usb_ovp_set_hystersis 28
6.14 pm8xxx_usb_ovp_disable 28
6.15 power_supply_get_by_name 29
6.16 power_supply_changed 29
6.17 power_supply_am_i_supplied 29
6.18 power_supply_set_battery_charged 30
6.19 power_supply_set_current_limit 30
6.20 power_supply_set_online 31
6.21 power_supply_set_charge_type 31
6.22 Configuration 31

7 Battery Alarm API 34

7.1 Introduction 34
7.2 pm8xxx_batt_alarm_enable 34
7.3 pm8xxx_batt_alarm_disable 34
7.4 pm8xxx_batt_alarm_threshold_set 35
7.5 pm8xxx_batt_alarm_status_read 35
7.6 pm8xxx_batt_alarm_register_notifier 36
7.7 pm8xxx_batt_alarm_unregister_notifier 36
7.8 pm8xxx_batt_alarm_hold_time_set 36
7.9 pm8xxx_batt_alarm_pwm_rate_set 37

8 BMS API 38

8.1 Introduction 38
8.2 pm8xxx_bms_get_battery_current 38
8.3 pm8xxx_bms_get_percent_charge 38
8.4 pm8xxx_bms_charging_began 39
8.5 pm8xxx_bms_charging_end 39
8.6 pm8xxx_bms_get_simultaneous_battery_voltage_and_ current 39
8.7 pm8xxx_bms_get_fcc 40
8.8 Configuration 40

9 CCADC API 42

9.1 Introduction 42
9.2 pm8xxx_cc_adjust_for_gain 42
9.3 pm8xxx_calib_ccadc 42
9.4 pm8xxx_ccadc_get_battery_current 43

10 PMIC ADC API 44

10.1 Introduction 44
10.2 pm8xxx_adc_read 44
10.3 pm8xxx_adc_mpp_config_read 44
10.4 pm8xxx_adc_btm_start 45
10.5 pm8xxx_adc_btm_ end 45
10.6 pm8xxx_adc_btm_configure 45

10.7 ADC Sysfs 46

11 PWM API 47

11.1 Introduction 47
11.2 pwm_request 47
11.3 pwm_free 48
11.4 pwm_config 48
11.5 pwm_enable 49
11.6 pwm_disable 49
11.7 pm8xxx_pwm_config_period 50
11.8 pm8xxx_pwm_config_pwm_value 50
11.9 pm8xxx_pwm_lut_config 51
11.10 pm8xxx_pwm_lut_enable 51

12 LED API 53

12.1 Introduction 53
12.2 led_trigger_register_simple 53
12.3 led_trigger_event 54
12.4 led_trigger_blink 54
12.5 LED configuration 54

13 Vibrator API 56

13.1 Introduction 56
13.2 pm8xxx_vibrator_config 56

14 Speaker API 57

14.1 Introduction 57
14.2 pm8xxx_spk_enable 57
14.3 pm8xxx_spk_mute 57
14.4 pm8xxx_spk_gain 58

15 PMIC Thermal API 59

15.1 Introduction 59
15.2 pm8xxx_tm_core_data 59

16 RTC API 60

16.1 Introduction 60

17 PMIC Clock API 61

17.1 Introduction 61
17.2 pm8xxx_aux_clk_control 61

18 PMIC Power-key API 62

18.1 Introduction 62
18.2 Configuration 62

19 PMIC Keypad API 63

19.1 Introduction 63
19.2 Keypad configuration 63

20 Miscellaneous API 64

20.1 Introduction 64

20.2 pm8xxx_reset_pwr_off 64
20.3 pm8xxx_smpl_control 64
20.4 pm8xxx_smpl_set_delay 65
20.5 pm8xxx_coincell_chg_config 65
20.6 pm8xxx_watchdog_reset_control 66
20.7 pm8xxx_stay_on 66
20.8 pm8xxx_hard_reset_config 66
20.9 pm8xxx_uart_gpio_mux_ctrl 67
20.10 pm8xxx_preload_dvdd 67

EXHIBIT 1 69

1 Introduction

1.1 Purpose

This document describes the Power Management Integrated Circuit (PMIC) API for A-family chips. The Qualcomm® Snapdragon™ APQ8064E is equipped with PMIC PMM8920, which contains two chips – PMIC PM8921 and a PM8821.

NOTE: This document provides a description of chipset capabilities. Not all features are available, nor are all features supported in the software.

NOTE: Enabling some features may require additional licensing fees.

1.2 Acronyms, abbreviations, and terms

Table 1-1 provides definitions for the acronyms, abbreviations, and terms used in this document.

Table 1-1 Acronyms, abbreviations, and terms

Term	Definition
ADB	Android Debug Bridge
ADC	Analog-to-Digital Converter
ASIC	Application Specific Integrated Circuit
ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
BMS	Battery Monitoring System
GPIO	General Purpose Input/Output
IRQ	Interrupt ReQuest
LED	Light Emitting Diode
MPP	Multi-Purpose Pin
OEM	Original Equipment Manufacturer
PD (pd)	Pull Down
PMIC	Power Management Integrated Circuit
PWM	Pulse Width Modulation
RPM	Remote Power Management
RTC	Real Time Clock
SS (ss)	Sleep Selectable
USB	Universal Serial Port

1.3 Scope

This document is written for engineers who are integrating the PMIC chipset drivers into their code.

1.4 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:`**.

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.

2 Regulator API

2.1 Introduction

The Linux regulator framework is designed to be consumer-centric. A given consumer need not be aware of any other consumers of a regulator; the consumer only needs to specify requirements in terms of the state the consumer needs the regulator to be in. This design allows for drivers on one system to work on another system with a different regulator configuration.

The **regulator_get()** API, which a consumer uses to request a regulator handle, facilitates this by using a consumer-specified regulator name. This allows consumer drivers to hardcode regulator names that are not board specific. The regulator framework takes care of aggregating requests from multiple consumers of one regulator to ensure that a mutually acceptable state is reached.

All regulator APIs that are available to consumers are specified in:

$KERNEL/include/linux/regulator/consumer.h

2.2 regulator_get

Gets access to the specific supply regulator.

Prototype

```
struct regulator * regulator_get
(
       struct device *dev,
       const char *id
)
```

Parameters

→	`dev`	A device struct pointer to your driver's device
→	`id`	Character pointer to the regulator consumer name defined by the driver or to the regulator supply

Return value

- 0 – Success
- < 0 – Error

2.3 regulator_put

Releases the regulator.

Prototype

```
void regulator_put(
       struct regulator *regulator
)
```

Parameters

→	`regulator`	A pointer to the struct regulator

Return value

None.

2.4 regulator_enable

Requests that the regulator be enabled with the regulator output at the predefined voltage or current value.

Prototype

```
int regulator_enable(
      struct regulator *regulator
)
```

Parameters

→	`regulator`	A pointer to the struct regulator

Return value

- 0 – Success
- < 0 – Error

2.5 regulator_disable

Disables the regulator output voltage or current. Calls to regulator_enable must be balanced with calls to regulator_disable.

Prototype

```
int regulator_disable(
      struct regulator *regulator
)
```

Parameters

→	`regulator`	A pointer to the struct regulator

Return value

- ≥ 0 – Success
- < 0 – Error

2.6 regulator_is_enabled

Tests if the regulator output is enabled.

Prototype

```
int regulator_is_enabled(
      struct regulator *regulator
)
```

Parameters

→	`regulator`	A pointer to struct regulator

Return value

- > 0 – Regulator driver backing the client has requested that the device be enabled.
- 0 – Regulator driver backing the client has not requested that the device be enabled.
- > 0 – ERRNO error value.

2.7 regulator_set_voltage

Dynamically changes the supply voltage to match system operating points. The range specified by min_uV and max_uV will be aggregated with other consumers’ ranges and with the constraint range usually defined in platform board file or device tree.

Prototype

```
int regulator_set_voltage(
      struct regulator *regulator,
      int min_uV,
      int max_uV
)
```

Parameters

→	`regulator`	A pointer to struct regulator
→	`min_uV`	Minimal microvolts to be set
→	`max_uV`	Maximal microvolts to be set

Return value

- 0 – Success
- < 0 – Error

2.8 regulator_get_voltage

Returns the configured output voltage whether the regulator is enabled or disabled and should NOT be used to determine regulator output state.

Prototype

```
int regulator_get_voltage(
      struct regulator *regulator
)
```

Parameters

→	`regulator`	A pointer to struct regulator

Return value

- > 0 – Currently configured output voltage in microvolts
- 0 – Success
- < 0 – Error

2.9 regulator_set_optimum_mode

Sets the mode of the regulator so it can output at least the specified current. This will be summed with other consumers’ current requirements before making the mode selection decision.

Prototype

```
int regulator_set_optimum_mode
(
      struct regulator *regulator,
      int load_uA
)
```

Parameters

→	`Regulator`	A pointer to struct regulator
→	`load_uA`	Load current in micro Amps

Return value

- >0 – Success.
- <= 0 – Failure.

2.10 Regulator configuration

2.10.1 Regulator declaration and constraints

Regulator declaration and constraints tables specify (or control) which regulators are available in the system and what their constraints are. Each entry contains a mix of regulator framework values and driver specific platform data values. The majority of regulators are controlled by the rpm-regulator driver. Some Apps-only regulators are controlled by the pm8xxx-regulator driver.

2.10.1.1 RPM Control

```
static struct rpm_regulator_init_data
msm_rpm_regulator_init_data[] __devinitdata = {
          /* ID a_on pd ss min_uV   max_uV  supply sys_uA freq */
    RPM_SMPS(S1,  1, 1, 0, 1225000, 1225000, NULL, 100000, 3p20),
    RPM_SMPS(S3,  0, 1, 0, 500000, 1150000, NULL, 100000, 4p80),
    RPM_SMPS(S4,  1, 1, 0, 1800000, 1800000, NULL, 100000, 3p20),
         /* ID a_on pd ss min_uV   max_uV  supply  sys_uA init_ip */
    RPM_LDO(L1,  1, 1, 0, 1050000, 1050000, "8921_s4", 0, 10000),
};
```

Regulator framework values

- a_on – Always_on (disable is not an option)
- min_uV – Minimum allowed voltage
- max_uV – Maximum allowed voltage
- supply – Parent regulator

Driver specific values

- ID – Driver specific ID
- pd – Pull-down enabled when Off
- ss – Sleep selectable (for sleep set; vote-able via the backdoor API)
- sys_uA – System load current
- init_ip – Initial peak current set in base RPM request

2.10.1.2 Local control (pm8xxx-regulator)

NOTE: Refer to Kernel/arch/arm/mach-msm/board-8064-regulator.c in the kernel code downloaded from www.codeaurora.org for the specific 8064.

```
struct pm8xxx_regulator_platform_data
msm_pm8xxx_regulator_pdata[] __devinitdata = {
                 /* ID  a_on pd  min_uV   max_uV  en_t supply sys_uA */
PM8xxx_VREG_INIT_SMPS(S1,    1, 1, 1225000, 1225000, 500, NULL, 100000),
PM8xxx_VREG_INIT_SMPS(S4,    1, 1, 1800000, 1800000, 500, NULL, 100000),
PM8xxx_VREG_INIT_LDO(L1,     1, 1, 1050000, 1050000, 200, "8921_s4", 0),
};
```

Regulator framework values

- a_on – Always_on (disable is not an option)
- min_uV – Minimum allowed voltage
- max_uV – Maximum allowed voltage
- supply – Parent regulator

Driver-specific values:

- ID – Driver specific ID
- pd – Pull-down enabled when off
- en_t – Enable time (in μs)
- sys_uA – System load current

2.10.2 Regulator consumer supply list

A consumer supply list links consumers to a given regulator. The following is an example of a consumer supply list of PM8921 LDO4 as the supply:

```
VREG_CONSUMERS(L4) = {
  REGULATOR_SUPPLY("8921_l4",       NULL),
  REGULATOR_SUPPLY("HSUSB_1p8",     "msm_otg"),
  REGULATOR_SUPPLY("iris_vddxo",    "wcnss_wlan.0"),
};
```

This list specifies consumer names HSUSB_1p8 and iris_vddxo. Each entry is a supply name and device name tuple.

- Uniqueness constraint: each name tuple must be unique among all regulators registered for a board.
- “HSUSB_1p8” would be the string for ‘id’ in regulator_get for the USB driver, and so is “iris_vddxo” for the WLAN driver.

2.10.3 Regulator supply chain

Regulators are defined with a link to a potential parent regulator. This allows the framework to capture arbitrarily complicated trees of subregulated regulators.

regulator_enable and **regulator_disable** calls propagate through the tree.

For example, assume that S1 is the parent regulator of L2. **regulator_enable(reg_l2)** will result in S1 being enabled and then L2 automatically inside of the regulator framework.

This capability is important so that a consumer only needs to be aware of the regulator(s) to which it is directly connected.

3 GPIO API

3.1 Introduction

Make sure a GPIO is configured correctly in the board file, or in the device tree for future kernel.

Use the Linux standard gpiolib API to control a PMIC GPIO when it is configured as a digital input and output.

PMIC GPIOs are mapped to the system GPIO map.

Use the Linux standard interrupt API to handle a GPIO as interrupt. In this case, the GPIO should be configured as a digital input.

3.2 gpio_request

Requests a GPIO.

Prototype

```
int gpio_request
(
      unsigned gpio,
      const char *label
)
```

Parameters

→	`gpio`	GPIO number
→	`label`	Non-null string for diagnostics

Return value

- ERRNO – Error value
- 0 – Success

3.3 gpio_free

Releases the previously-claimed GPIO.

Prototype

```
void gpio_free
(
      unsigned gpio
)
```

Parameters

→	`gpio`	GPIO number

Return value

None.

3.4 gpio_set_value_cansleep

Sets a GPIO value; this function might sleep.

Prototype

```
void gpio_set_value_cansleep
(
      unsigned gpio,
      int value
)
```

Parameters

→	`Gpio`	GPIO number
→	`Value`	Boolean ▪ 0 – Low ▪ Nonzero – High

Return value

- None.

3.5 gpio_get_value_cansleep

Gets a GPIO value. This function might sleep.

Prototype

```
int gpio_get_value_cansleep
(
      unsigned gpio
)
```

Parameters

→	`gpio`	GPIO number

Return value

0 or non-zero – Might sleep

3.6 gpio_to_irq

Maps GPIO numbers to IRQ numbers. This API returns the corresponding IRQ number of a GPIO number.

Prototype

```
int gpio_to_irq
(
      unsigned gpio
)
```

Parameters

→	`gpio`	GPIO number

Return value

IRQ number.

3.7 pm8xxx_gpio_config

Configures the setting for an individual GPIO pin.

Prototype

```
int pm8xxx_gpio_config
(
      int               gpio,
      struct pm_gpio    *param,
)
```

Parameters

→	`gpio`	GPIO ID
→	`param`	Pointer to pm_gpio data structure

Return value

- ERRNO – Error value
- 0 – Success

4 MPP API

4.1 Introduction

Multi-Purpose Pins (MPP) can be used for different purposes, such as digital or analog input or output. MPPs need correct configuration before being used. This is done in board files or in device tree files.

When an MPP is configured as a digital signal pin, it can be used through GPIO API.

When an MPP is configured as an analog signal or special function, use the corresponding API, such as Analog to Digital Converter (ADC) API for analog input, or Pulse Width Modulation (PWM) API for an output signal requiring PWM modulation.

Header file

include/linux/mfd/pm8xxx/mpp.h

4.2 pm8xxx_mpp_config

Configures the setting for an individual MPP pin.

Prototype

```
int pm8xxx_mpp_config
(
      unsigned                     mpp,
      struct pm8xxx_mpp_config_data  *config,
)
```

Parameters

→	mpp	MPP ID
→	config	Pointer to pm8xxx_mpp_config_data data structure

Return value

- ERRNO – Error value
- 0 – Success

5 PMIC IRQ API

5.1 Introduction

PMIC interrupts are all mapped to the kernel's interrupt map. PMIC 'base' IRQ is usually determined after APQ IRQ map.

Use standard Linux IRQ API for PMIC interrupts, which include:

- Passing system-wide IRQ# in board file to a driver.
- Calling Linux interrupt API to request, enable, disable, and free an IRQ.
- Calling Linux interrupt set_wake API for wake up interrupt.

Header file

include/linux/mfd/pm8xxx/irq.h

include/linux/mfd/pm8xxx/pm8xxx.h

5.2 request_threaded_irq

Allocates interrupt resources and enables the interrupt line and IRQ handling. From the point this call is made, the handler function may be invoked. To set up a threaded IRQ handler for your device, you need to supply handler and thread_fn.

Prototype

```
int request_threaded_irq (
      unsigned int irq,
      irq_handler_t handler,
      irq_handler_t thread_fn,
      unsigned long irqflags,
      const char * devname,
      void * dev_id
)
```

Parameters

→	`irq`	Interrupt line to allocate
→	`handler`	Function to be called when the IRQ occurs
→	`thread_fn`	Function called from the IRQ handler thread. If NULL, no irq thread is created
→	`irqflags`	Interrupt type flags
→	`devname`	An ascii name for the claiming device
→	`dev_id`	A cookie passed back to the handler function

Return value

- ERRNO – Error value
- 0 – Success

5.3 free_irq

Removes an interrupt handler. The handler is removed and if the interrupt line is no longer in use by any driver, it is disabled.

Prototype

```
void free_irq (
      unsigned int    irq,
      void *      dev_id
)
```

Parameters

→	`irq`	Interrupt line to free
→	`dev_id`	Device identity to free

Return value

None.

5.4 enable_irq

Enables handling of an IRQ by undoing the effect of one call to disable_irq. If this matches the last disable, processing of interrupts on this IRQ line is re-enabled.

Prototype

```
void enable_irq (
      unsigned int irq
)
```

Parameters

→	`irq`	Interrupt to enable

Return value

None.

5.5 disable_irq

Disables the selected interrupt line. This function waits for any pending IRQ handlers for this interrupt to complete before returning. If this function is used while holding a resource, the needed IRQ handler will deadlock.

Prototype

```
void disable_irq (
      unsigned int irq
)
```

Parameters

→	irq	Interrupt to disable

Return value

None.

5.6 irq_set_irq_wake

Notifies the kernel that an interrupt is expected to wake up the system if enabled. If an interrupt line has the capability to wake up the system when the system is in suspend mode, this function can be called to notify the kernel interrupt chip driver that a wake up is expected if the ON parameter is TRUE.

Prototype

```
int irq_set_irq_wake(
            unsigned int irq,
            unsigned int on
);
```

Parameters

→	irq	Interrupt number
→	on	True or false

These two macros can be used:

- enable_irq_wake(unsigned int irq)
- disable_irq_wake(unsigned int irq)

Return value

- ERRNO – Error value
- 0 – Success

5.7 pm8xxx_get_irq_stat

Gets the status of the IRQ line. The pm8xxx GPIO and MPP rely on the interrupt block to read the values on their pins. This function is used to facilitate reading the real-time status of a GPIO or an MPP line. The caller must convert the GPIO number to an IRQ number.

This API is intended for other PMIC drivers to query the real time status of an interrupt line, such as for a PMIC GPIO chip driver to query the status of a GPIO as interrupt. It is not intended for other purposes.

Prototype

```
int pm8xxx_get_irq_stat
(
      struct pm_irq_chip   *chip,
      int              irq
)
```

Parameters

→	`chip`	Pointer to identify a PMIC IRQ controller
→	`Irq`	The IRQ number

Return value

- ERRNO – Error value
- 0 – Success

6 PMIC Charger API

6.1 Introduction

The Linux Power Supply framework and API should be used by the user space application as well as any kernel driver. PMIC charger driver supports the API through the power-supply framework. The remaining proprietary API will be converted to power supply API.

6.2 pm8xxx_charger_enable

Enables/disables battery charging current; the device will still draw current from the charging source.

NOTE: This API will be replaced by power_supply_set_online().

Prototype

```
Int pm8xxx_charger_enable (
      bool enable
}
```

Parameters

→	`enable`	Enable the charger

Return value

This function returns an integer:

- Zero – Success
- Non-zero – Fail

6.3 pm8xxx_is_usb_chg_plugged_in

Returns if the USB is plugged in. If the USB is under voltage or over voltage, this will return False.

NOTE: This API will be replaced by using get_property of POWER_SUPPLY_PROP_ONLINE for POWER_SUPPLY_TYPE_USB.

Prototype

```
int pm8xxx_is_usb_chg_plugged_in(
        void
)
```

Return value

This function returns an integer:

- Zero – False
- Non-zero – True

6.4 pm8xxx_is_dc_chg_plugged_in

Returns if DC is plugged in. If DC is under voltage or over voltage this will return False.

NOTE: This API will be replaced by using get_property of POWER_SUPPLY_PROP_ONLINE for POWER_SUPPLY_TYPE_MAINS.

Prototype

```
int pm8xxx_is_dc_chg_plugged_in(
        void
)
```

Return value

Returns an integer:

- Zero – False
- Non-zero – True

6.5 pm8xxx_is_battery_present

Returns whether the PMIC sees the battery present.

NOTE: This API will be replaced by using get_property of POWER_SUPPLY_PROP_PRESENT for POWER_SUPPLY_TYPE_BATTERY.

Prototype

```
int pm8xxx_is_battery_present(
        void
)
```

Return value

This function returns an integer:

- Zero – False
- Non-zero – True

6.6 pm8xxx_disable_input_current_limit

Disables the input current limit. Disabling the charge current limit causes the present current limits to have no monitoring. An adequate charger capable of supplying high current while sustaining VIN_MIN is required if input current limiting is disabled.

NOTE: This API will be replaced by another API through the power supply framework.

Prototype

```
int pm8xxx_disable_input_current_limit(
      bool disable
)
```

Parameters

→	`disable`	Disable the current limit

Return value

This function returns integer:

- Zero – Success
- Non-zero – Failure

6.7 pm8xxx_set_usb_power_supply_type

Allows one set of a specific USB power_supply_type. USB drivers can distinguish between types of USB connections and set the appropriate type for the USB supply.

NOTE: This API will be replaced by using set_property of POWER_SUPPLY_PROP_CHARGE_TYPE for POWER_SUPPLY_TYPE_USB or by power_supply_set_charge_type().

Prototype

```
int pm8xxx_set_usb_power_supply_type(
     enum power_supply_type type
)
```

Parameters

→	`POWER_SUPPLY_TYPE_BATTERY`	Power supply from battery
→	`POWER_SUPPLY_TYPE_UPS`	Power supply from UPS
→	`POWER_SUPPLY_TYPE_MAINS`	Power supply from AC charger
→	`POWER_SUPPLY_TYPE_DSP`	Standard Downstream Port (DSP)
→	`POWER_SUPPLY_TYPE_USB_DCP`	Dedicated Charging Port (DCP)
→	`POWER_SUPPLY_TYPE_USB_CDP`	Charging Downstream Port (CDP)
→	`POWER_SUPPLY_TYPE_USB_ACA`	Accessory Charger Adapters (ACA)

Return value

This function returns an integer:

- Zero – Success
- Non-zero – Failure

6.8 pm8xxx_disable_source_current

Stops all charging activities and disables any current drawn from the charger. The battery provides the system current.

NOTE: This API will be replaced by power_supply_set_current_limit().

Prototype

```
int pm8xxx_disable_source_current(
      bool disable
)
```

Parameters

→	`disable`	Disable the source current

Return value

This function returns an integer:

- Zero – Success
- Non-zero – Failure

6.9 pm8xxx_regulate_input_voltage

This function allows values from 4300 mV to 6500 mV.

Prototype

```
int pm8xxx_regulate_input_voltage
(
      int voltage
)
```

Parameters

→	`voltage`	Voltage in millivolts to regulate

Return value

- Zero – Success
- Non-zero – Failure

6.10 pm8xxx_is_battery_charging

Checks if the battery is charging.

NOTE: This API will be replaced by power_supply_am_i_supplied().

Prototype

```
bool pm8xxx_is_battery_charging(
      int *source
)
```

Parameters

→	`source`	Finite state machine

When the battery is charging the source is updated to reflect which charger, USB or DC, is charging the battery.

Return value

Bool – Indicates whether or not the battery is being charged.

6.11 pm8xxx_batt_temperature

Gets battery temperature.

NOTE: This API will be replaced by using get_property() of POWER_SUPPLY_PROP_TEMPERATURE for POWER_SUPPLY_TYPE_BATTERY.

Prototype

```
int pm8xxx_batt_temperature(
      void
)
```

Return value

Temp in C.

6.12 pm8xxx_usb_ovp_set_threshold

Sets the USB threshold as defined by enum usb_ov_threshold.

Prototype

```
int pm8xxx_usb_ovp_set_threshold(
      enum pm8xxx_usb_ov_threshold ov
)
```

Parameters

→	`pm8xxx_usb_ov_threshold`	▪ PM_USB_OV_5P5V ▪ PM_USB_OV_6V ▪ PM_USB_OV_6P5V ▪ PM_USB_OV_7V

Return value

- Zero – Success
- Non-zero – Failure

6.13 pm8xxx_usb_ovp_set_hystersis

Sets the debounce time for USB insertion/removal detection.

Prototype

```
int pm8xxx_usb_ovp_set_hystersis(
        enum pm8xxx_usb_debounce_time ms
)
```

Parameters

→	`pm8xxx_usb_debounce_time`	▪ PM_USB_BYPASS_DEBOUNCER ▪ PM_USB_DEBOUNCE_20P5MS ▪ PM_USB_DEBOUNCE_40P5MS ▪ PM_USB_DEBOUNCE_80P5MS

Return value

- Zero – Success
- Non-zero – Failure

6.14 pm8xxx_usb_ovp_disable

Disables the USB OVP. When disabled, there is no over-voltage protection. The USB voltage is fed to the PMIC as is. This should be disabled only when there is over-voltage protection circuitry present outside the PMIC chip.

Prototype

```
int pm8xxx_usb_ovp_disable(
        int disable
)
```

Parameters

→	`disable`	Disable the USB OVP

Return value

- Zero – Success
- Non-zero – Failure

6.15 power_supply_get_by_name

Gets the pointer to the struct power_supply by its name.

For convenience, the power supply drivers have the name of “battery”, “usb”, and “dc”, which can be used as ‘name’ to access the supported API by the supply driver.

Prototype

```
struct power_supply *power_supply_get_by_name
(
      char *name
)
```

Parameters

→	name	The name of the power supply

Return value

Pointer to the struct power_supply.

6.16 power_supply_changed

Notifies the application layer of the change in the power supply. This API will eventually call external_power_changed() of the struct power_supply and issue an Android UEvent (CHANGE) to the application layer.

Prototype

```
void power_supply_changed
(
      struct power_supply *psy
)
```

Parameters

→	psy	Pointer to the struct power_supply

Return value

None.

6.17 power_supply_am_i_supplied

Determines if the power is received.

Prototype

```
int power_supply_am_i_supplied(
      struct power_supply *psy
)
```

Parameters

→	psy	Pointer to the struct power_supply

Return value

- Zero – No
- Non-zero – Yes

6.18 power_supply_set_battery_charged

Sets the battery charged.

Prototype

```
int power_supply_set_battery_charged(
      struct power_supply *psy
)
```

Parameters

→	psy	Pointer to the struct power_supply

Return value

Integer.

6.19 power_supply_set_current_limit

Sets the current limit.

Prototype

```
int power_supply_set_current_limit(
      struct power_supply *psy,
      int limit
)
```

Parameters

→	psy	Pointer to the struct power_supply
→	limit	Current limit

Return value

Integer.

6.20 power_supply_set_online

Sets the online power supply.

Prototype

```
int power_supply_set_online(
      struct power_supply *psy,
      bool enable
)
```

Parameters

→	`psy`	Pointer to the struct power_supply
→	`enable`	Enable a power supply

Return value

Integer.

6.21 power_supply_set_charge_type

Sets the charge type.

Prototype

```
int power_supply_set_charge_type(
      struct power_supply *psy,
      int type
)
```

Parameters

→	`psy`	Pointer to the struct power_supply
→	`type`	Charger type

Return value

Integer.

6.22 Configuration

OEMs are responsible for making sure that the struct pm8xxx_charger_platform_data has the correct data for the batteries in OEM devices.

The following list includes the current version of platform data, which is here for reference only and could be changed in later releases.

NOTE: Refer to Kernel/arch/arm/mach-msm/board-8064-pmic.c in the kernel code downloaded from www.codeaurora.org for the specific 8064.

- struct pm8xxx_charger_platform_data:

```
@ttrkl_time:  max trckl charging time in minutes
             valid range 1 to 64 mins. PON default 15 min
   @update_time:  how often the userland be updated of the charging (msec)
   @alarm_low_mv: the voltage (mV) when low battery alarm is triggered
   @alarm_high_mv:the voltage (mV) when high battery alarm is triggered
   @max_voltage:  the max voltage (mV) the battery should be charged up to
   @min_voltage:  the voltage (mV) where charging method switches from
             trickle to fast. This is also the minimum voltage the
             system operates at
   @uvd_thresh_voltage:the USB falling UVD threshold (mV) (PM8917 only)
   @safe_current_ma:   The upper limit of current allowed to be pushed in
                  battery. This ends up writing in a one time
                  programmable register.
   @resume_voltage_delta:   the (mV) drop to wait for before resume charging
                  after the battery has been fully charged
   @resume_charge_percent:  the % SOC the charger will drop to after the
                  battery is fully charged before resuming charging.
   @term_current: the charger current (mA) at which EOC happens
   @cool_temp:    the temperature (degC) at which the battery is
             considered cool charging current and voltage is reduced.
             Use INT_MIN to indicate not valid.
   @warm_temp:    the temperature (degC) at which the battery is
             considered warm charging current and voltage is reduced
             Use INT_MIN to indicate not valid.
   @temp_check_period: The polling interval in seconds to check battery
             temeperature if it has gone to cool or warm temperature area
   @max_bat_chg_current:    Max charge current of the battery in mA
                  Usually 70% of full charge capacity
   @cool_bat_chg_current:   chg current (mA) when the battery is cool
   @warm_bat_chg_current:   chg current (mA) when the battery is warm
   @cool_bat_voltage:  chg voltage (mV) when the battery is cool
   @warm_bat_voltage:  chg voltage (mV) when the battery is warm
   @get_batt_capacity_percent:
             a board specific function to return battery capaticy.
             If null - a default one will be used
   @dc_unplug_check:   enables the reverse boosting fix for the DC_IN line
             however, this should only be enabled for devices which control the
             DC OVP FETs otherwise this option should remain disabled
   @has_dc_supply:report DC online if this bit is set in board file
   @trkl_voltage: the trkl voltage in (mV) below which hw controlled
             trkl charging happens with linear charger
   @weak_voltage: the weak voltage (mV) below which hw controlled
             trkl charging happens with switching mode charger
```

```
@trkl_current: the trkl current in (mA) to use for trkl charging phase
@weak_current: the weak current in (mA) to use for weak charging phase
@vin_min: the input voltage regulation point (mV) - if the voltage falls
          below this, the charger reduces charge current or stops charging
         temporarily
@thermal_mitigation:the array of charge currents to use as temperature
               increases
@thermal_levels:    the number of thermal mitigation levels supported
@cold_thr:if high battery will be cold when VBAT_THERM goes above
          80% of VREF_THERM (typically 1.8volts), if low the
          battery will be considered cold if VBAT_THERM goes above
          70% of VREF_THERM. Hardware defaults to low.
@hot_thr: if high the battery will be considered hot when the
          VBAT_THERM goes below 35% of VREF_THERM, if low the
          battery will be considered hot when VBAT_THERM goes
          below 25% of VREF_THERM. Hardware defaults to low.
@rconn_mohm:   resistance in milliOhm from the vbat sense to ground
          with the battery terminals shorted. This indicates
          resistance of the pads, connectors, battery terminals
          and rsense.
@led_src_config:    Power source for anode of charger indicator LED.
```

7 Battery Alarm API

7.1 Introduction

The Battery Alarm module provides alarm (interrupt) capability to notify the device when a programmed voltage threshold is crossed.

The two thresholds in the module are for:

- Higher voltage
- Lower voltage

7.2 pm8xxx_batt_alarm_enable

Enables one of the battery voltage threshold comparators (upper or lower).

Prototype

```
int pm8xxx_batt_alarm_enable(
      enum pm8xxx_batt_alarm_comparator comparator
)
```

Parameters

→	`pm8xxx_batt_alarm_comparator`	▪ PM8XXX_BATT_ALARM_LOWER_COMPARATOR ▪ PM8XXX_BATT_ALARM_UPPER_COMPARATOR

Return value

- Zero – Success
- ERRNO – Failure

7.3 pm8xxx_batt_alarm_disable

Disables one of the battery voltage threshold comparators.

Prototype

```
int pm8xxx_batt_alarm_disable(
      enum pm8xxx_batt_alarm_comparator comparator
)
```

Parameters

→	`pm8xxx_batt_alarm_comparator`	▪ PM8XXX_BATT_ALARM_LOWER_COMPARATOR ▪ PM8XXX_BATT_ALARM_UPPER_COMPARATOR

Return value

- Zero – Success
- ERRNO – Failure

7.4 pm8xxx_batt_alarm_threshold_set

Sets the lower or upper alarm threshold.

Prototype

```
int pm8xxx_batt_alarm_threshold_set(
   enum pm8xxx_batt_alarm_comparator comparator,
   int threshold_mV
)
```

Parameters

→	`pm8xxx_batt_alarm_comparator`	▪ PM8XXX_BATT_ALARM_LOWER_COMPARATOR ▪ PM8XXX_BATT_ALARM_UPPER_COMPARATOR
→	`threshold_mV`	Battery voltage threshold in millivolts points = 2500-5675 mV in 25 mV steps

Return value

- Zero – Success
- ERRNO – Failure

7.5 pm8xxx_batt_alarm_status_read

Gets the status of both threshold comparators.

Prototype

```
int pm8xxx_batt_alarm_status_read(
void
)
```

Return value

- 0 = Error
- 0 = Battery voltage OK
- BIT(0) set = Battery voltage below lower threshold
- BIT(1) set = Battery voltage above upper threshold

7.6 pm8xxx_batt_alarm_register_notifier

Registers a notifier to run when a battery voltage change interrupt fires.

Prototype

```
int pm8xxx_batt_alarm_register_notifier(
      struct notifier_block *nb
)
```

Parameters

→	nb	nb→notifier_call must point to a function of this form – int (*notifier_call)(struct notifier_block *nb, unsigned long status, void *unused);

Return value

- Zero – Success
- ERRNO – Failure

7.7 pm8xxx_batt_alarm_unregister_notifier

Unregisters a notifier so that it is no longer run when a battery voltage change interrupt fires.

Prototype

```
int pm8xxx_batt_alarm_unregister_notifier(
      struct notifier_block *nb
)
```

Parameters

→	nb	nb→notifier_call must point to a function of this form - int (*notifier_call)(struct notifier_block *nb, unsigned long status, void *unused);

Return value

- Zero – Success
- ERRNO – Failure

7.8 pm8xxx_batt_alarm_hold_time_set

Sets the hold time for the battery alarm. Using a larger value reduces the number of times that the interrupt triggers while the battery voltage is near one of the thresholds.

Prototype

```
int pm8xxx_batt_alarm_hold_time_set(
      enum pm8xxx_batt_alarm_hold_time hold_time
)
```

Parameters

→	`pm8xxx_batt_alarm_hold_time`	▪ PM8XXX_BATT_ALARM_HOLD_TIME_0p125_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_0p25_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_0p5_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_1_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_2_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_4_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_8_MS ▪ PM8XXX_BATT_ALARM_HOLD_TIME_16_MS

Return value

- Zero – Success
- ERRNO – Failure

7.9 pm8xxx_batt_alarm_pwm_rate_set

Sets the rate at which the battery alarm module enables the threshold comparators. The rate is determined by the following equation:

f_update = (1024 Hz) / (clock_divider * (2 ^ clock_scaler))

Thus, the update rate can range from 0.25 Hz to 128 Hz. Utilizing PWM with a small frequency helps to reduce the number of times that the interrupt triggers while the battery voltage is near one of the thresholds.

Prototype

```
int pm8xxx_batt_alarm_pwm_rate_set(
        int use_pwm,
        int clock_scaler,
        int clock_divider
)
```

Parameters

→	`use_pwm`	1 – Use PWM update rate 0 – Comparators always active
→	`clock_scaler`	PWM clock scaler; range 2 to 9
→	`clock_divider`	PWM clock divider; range 2 to 8

Return value

- Zero – Success
- ERRNO – Failure

8 BMS API

8.1 Introduction

The BMS (Battery Monitoring System) driver monitors the battery charge flow using coulomb counting. BMS provides a nonstandard API for the charger driver to report to the kernel power supply framework, which then reports to the user space. BMS replies on the service provided by the CCADC driver.

8.2 pm8xxx_bms_get_battery_current

Returns the battery current based on vsense resitor in microamperes. The pointer result is where the voltage will be updated. A negative result means that the current is flowing in the battery during battery charging.

NOTE: This API will be replaced by a power_supply API.

Prototype

```
int pm8xxx_bms_get_battery_current(
      int *result
)
```

Parameters

→	`*result`	Battery current

Return value

- Zero – Success
- ERRNO – There was a problem when reading vsense

8.3 pm8xxx_bms_get_percent_charge

Gets the current battery charge, in percent.

NOTE: This API will be replaced by a power_supply API.

Prototype

```
int pm8xxx_bms_get_percent_charge(
void
)
```

Return value

Current battery charge in percent.

8.4 pm8xxx_bms_charging_began

Notifies the BMS driver that charging has started, keeping track of charge cycles.

NOTE: This API will be replaced by a power_supply API.

Prototype

```
void pm8xxx_bms_charging_began(
      void
)
```

Return value

None.

8.5 pm8xxx_bms_charging_end

Notifies the BMS driver that charging has stopped, keeping track of charge cycles.

NOTE: This API will be replaced by a power_supply API.

Prototype

```
void pm8xxx_bms_charging_end(
      int is_battery_full
)
```

Parameters

→	`is_battery_full`	▪ 0 – Not full ▪ 1 – Full

Return value

None.

8.6 pm8xxx_bms_get_simultaneous_battery_voltage_and_current

Takes simultaneous battery voltage and current readings. The API puts the BMS in override mode but keeps coulumb counting on. It is useful when IR compensation needs to be implemented.

Prototype

```
int pm8xxx_bms_get_simultaneous_battery_voltage_and_current(
    int *ibat_ua,
    int *vbat_uv
)
```

Parameters

→	*ibat_ua	Pointer to the battery current
→	*vbat_uv	Pointer to the battery voltage

Return value

None.

8.7 pm8xxx_bms_get_fcc

Gets fcc in mAh of the battery depending on its age and temperature.

NOTE: This API will be replaced by a power_supply API.

Prototype

```
int pm8xxx_bms_get_fcc(
        void
)
```

Return value

The fcc in mAh of the battery.

8.8 Configuration

OEMs are responsible for making sure that the struct pm8xxx_bms_platform_data has the correct data for the batteries in OEM devices.

The following list includes the current version of platform data, which is for reference only and could be changed in later releases.

NOTE: Refer to Kernel/arch/arm/mach-msm/board-8064-pmic.c in the kernel code downloaded from www.codeaurora.org for the specific 8064.

- struct pm8xxx_bms_platform_data:

```
@bms_cdata:     'struct pm8xxx_bms_core_data'. Should not change.
@batt_type:     allows to force chose battery calibration data
@r_sense_uohm:  sense resistor value in (micro Ohms)
@i_test:   current at which the unusable charger cutoff is to be
           calculated or the peak system current (mA)
```

```
@v_cutoff:the loaded voltage at which the battery
          is considered empty(mV)
@enable_fcc_learning:    if set the driver will learn full charge
               capacity of the battery upon end of charge
@max_voltage_uv:    max voltage the battery can be charged to
@rconn_mohm:   the resistence of the battery connector
@shutdown_soc_valid_limit: the upper limit of the soc difference
               between shutdown soc and new soc
@ignore_shutdown_soc:    a boolean to use or not use shutdown soc
@adjust_shutdown_low_threshold: the threshold to start adjustment
               when soc during discharge is low
@chg_term_ua:  the threshold of current for end of charging
```

9 CCADC API

9.1 Introduction

Coulomb Counter Analog-to-Digital Converter (CCADC) is an ADC dedicated for the BMS.

9.2 pm8xxx_cc_adjust_for_gain

Adjusts the voltage read from CCADC for gain compensation. Structure pointer of type adc_arb_btm_param * is provided by the client for threshold warm/cold, interval, and functions to call when warm/cold events are triggered.

Prototype

```
s64 pm8xxx_cc_adjust_for_gain(
      s64 uv
)
```

Parameters

→	uv	The voltage that needs to be gain compensated in microVolts

Return value

uint32_t

9.3 pm8xxx_calib_ccadc

This API configures the BATT_THERM channel parameters for warm/cold thresholds. Structure pointer of type adc_arb_btm_param * is provided by the client for threshold warm/cold, interval, and functions to call when warm/cold events are triggered.

Prototype

```
void pm8xxx_calib_ccadc(
      void
)
```

Return value

uint32_t

9.4 pm8xxx_ccadc_get_battery_current

This API configures the BATT_THERM channel parameters for warm/cold thresholds. Structure pointer of type adc_arb_btm_param * is provided by the client to provide for threshold warm/cold, interval, and functions to call when warm/cold events are triggered.

Prototype

```
int pm8xxx_ccadc_get_battery_current(
      int *bat_current
)
```

Parameters

→	`*bat_current`	Return the battery current based on vsense resitor in microamperes

Return value

uint32_t

10 PMIC ADC API

10.1 Introduction

The PMIC ADC driver is registered with the Linux Hardware Monitoring (hwmon) framework. Hwmon framework provides the API to the user space application, but no API for kernel use. The driver supports non-framework API for kernel use.

10.2 pm8xxx_adc_read

Performs ADC read on the channel.

Prototype

```
uint32_t pm8xxx_adc_read(
      enum pm8xxx_adc_channels channel,
      struct pm8xxx_adc_chan_result *result
)
```

Parameters

→	`channel`	Input channel to perform the ADC read
→	`*result`	Structure pointer of type adc_chan_result in which the ADC read results are stored

Return value

uint32_t

10.3 pm8xxx_adc_mpp_config_read

Configures the PM8XXX MPP to AMUX6 and performs an ADC read.

Prototype

```
uint32_t pm8xxx_adc_mpp_config_read(
      uint32_t mpp_num,
      enum pm8xxx_adc_channels channel,
      struct pm8xxx_adc_chan_result *result
)
```

Parameters

→	`mpp_num`	PM8XXX MPP number to configure to AMUX6
→	`channel`	Input channel to perform the ADC read. ▪ ADC_MPP_1_AMUX6' if the input voltage is less than 1.8 V ▪ ADC_MPP_2_AMUX6' if the input voltage is greater than 1.8 V
→	`result`	Structure pointer of type adc_chan_result in which the ADC read results are stored

Return value

uint32_t

10.4 pm8xxx_adc_btm_start

Configures the BTM registers and starts monitoring the BATT_THERM channel for threshold warm/cold temperature set by the battery client.

Prototype

```
uint32_t pm8xxx_adc_btm_start(
      void
)
```

Return value

uint32_t

10.5 pm8xxx_adc_btm_ end

Causes the BTM registers to stop monitoring the BATT_THERM channel for warm/cold events and disables the interval timer.

Prototype

```
uint32_t pm8xxx_adc_btm_end(
      void
)
```

Return value

uint32_t

10.6 pm8xxx_adc_btm_configure

Configures the BATT_THERM channel parameters for warm/cold thresholds. Structure pointer of type adc_arb_btm_param * is provided by the client for threshold warm/cold, interval, and functions to call when warm/cold events are triggered.

Prototype

```
uint32_t pm8xxx_adc_btm_configure(
      struct pm8xxx_adc_arb_btm_param *
)
```

Parameters

→	`pm8xxx_adc_arb_btm_param*`	Structure pointer of type adc_arb_btm_param * which client provides for threshold warm/cold, interval and functions to call when warm/cold events are triggered

Return value

uint32_t

10.7 ADC Sysfs

Sysfs provides a way to read ADC values of specific pins from Android Debug Bridge (ADB).

ADB Commands

```
ADB console: cd sys/devices/platform/msm_ssbi.0/pm8xxx-core/pm8xxx-adc/
```

Parameters

```
ADB console: ls
```

ADC channels are listed below.

→	125v	1.25 V reference voltage
→	625mv	625 mV reference voltage
→	batt_id	Battery id
→	batt_therm	Battery temperature
→	chg_temp	Charger temperature
→	dcin	Voltage of DCIN
→	ibat	Battery current
→	ichg	Charger current
→	pa_therm0	Temperature of power amplifier 0
→	pa_therm1	Temperature of power amplifier 1
→	pmic_therm	Temperature of PMIC die
→	usbin	Voltage of USBIN
→	vbat	Battery voltage
→	vcoin	Voltage of coin cell
→	vph_pwr	Voltage of VPH_PWR
→	xo_therm	Temperature of XO

Example

Read the ADC value of battery ID:

```
ADB console: cat batt_id .
```

11 PWM API

11.1 Introduction

As of kernel version 3.4, there is no PWM framework in the Linux kernel, except for a header file specifying a few PWM API prototypes.

PWM API prototypes are supported by the pm8xxx-pwm driver, with these constraints:

- An ‘int’ or ‘unsigned int’ of 32-bit period in pwm_config() cannot represent a range from nano seconds to hundreds of seconds.
- The duty cycle is in the same units as the period, and cannot be lower than one (1).

The QuIC specific API is provided to meet the requirement not covered by the API prototypes.

- PWM frequency can be calculated using this equation:

  F_pwm = F_clk / (p * 2^n * 2^m)

  where

  - F_clk is the clock source frequency: 19.2MHz, 32768Hz or 1024Hz
  - p is the pre-divider: 1, 2, 3, 5, or 6
  - n is the number of bits for PWM value (size): 6 or 9
  - m is the divider exponent: [0..7]

Header files

- linux/pwm.h
- linux/mfd/pm8xxx/pwm.h

11.2 pwm_request

Requests access to a PWM device.

Prototype

```
pwm_device *pwm_request
(
      int        pwm_id,
      const char *lable
)
```

Parameters

→	`pwm_id`	PWM ID or channel
→	`label`	Label to identify user

Return value

This function returns a pointer to pwm_device:

- ENODEV – There is no valid PWM device available
- EINVAL – Invalid PWM ID
- EBUSY – Requested PWM ID is currently busy
- Pointer – Success; address of the PWM data structure of requested ID

11.3 pwm_free

Releases a PWM device.

Prototype

```
void pwm_free
(
      struct pwm_device    *pwm
)
```

Parameters

→	`pwm`	Address of a PWM data structure

Return value

None.

11.4 pwm_config

Changes the configuration of a PWM device.

Prototype

```
int pwm_config
(
      struct pwm_device    *pwm,
      int              duty_us,
      int              period_us
)
```

Parameters

→	`pwm`	Address of the intended PWM device
→	`duty_us`	Period in micro-seconds
→	`period_us`	Duty cycle in micro-seconds

Return value

- ENODEV – No valid PWM device available.
- EINVAL – Invalid PWM handle, parameters, or the intended PWM is not in use.
- 0 – Success

11.5 pwm_enable

Toggles the output of a PWM device.

Prototype

```
int pwm_enable
(
      struct pwm_device    *pwm
)
```

Parameters

→	pwm	Address of a PWM data structure

Return value

- ENODEV – No valid PWM device available.
- EINVAL – Invalid PWM handle or the intended PWM is not in use.
- 0 – Success

11.6 pwm_disable

Stops toggling the output of a PWM device.

Prototype

```
int pwm_disable
(
      struct pwm_device    *pwm
)
```

Parameters

→	pwm	Address of a PWM data structure

Return value

None.

Dependencies

A valid PWM address and pwm→chip

11.7 pm8xxx_pwm_config_period

Changes the period of a PWM device with the chip specific parameters in struct pm8xxx_pwm_period. Use pm8xxx_pwm_config_pwm_value() to change the duty cycle as raw PWM value.

Prototype

```
int pm8xxx_pwm_config_period
(
      struct pwm_device         *pwm,
      struct pm8xxx_pwm_period  *period
)

struct pm8xxx_pwm_period {
      enum pm_pwm_size     pwm_size;
      enum pm_pwm_clk      clk;
      enum pm_pwm_pre_div  pre_div;
      int                  pre_div_exp;
}
```

Parameters

→	`pwm`	Address of the intended PWM device
→	`period`	Period in struct pm8xxx_pwm_period

Return value

- ENODEV–- No valid PWM device available
- EINVAL – Invalid PWM ID
- 0 – Success

11.8 pm8xxx_pwm_config_pwm_value

Changes the raw value of a PWM duty cycle. The raw PWM value ranges from 0 to (2^pwm_size – 1), such as 63 for pwm_size=6. PWM period should be configured using pm8xxx_pwm_config_period first.

Prototype

```
int pm8xxx_pwm_config_pwm_value
(
      struct pwm_device    *pwm,
      int             pwm_value
)
```

Parameters

→	`pwm`	Address of the intended PWM device
→	`pwm_value`	Duty cycle in raw PWM value less than 2^pwm_size

Return value

- ENODEV – No valid PWM device available
- EINVAL – Invalid PWM ID
- 0 – Success

11.9 pm8xxx_pwm_lut_config

Changes the PWM configuration to use Look Up Table (LUT).

Prototype

```
int pm8xxx_pwm_lut_config
(
      struct pwm_device    *pwm,
      int              period_us,
      int              duty_pct[],
      int              duty_time_ms,
      int              start_idx,
      int              idx_len,
      int              pause_lo,
      int              pause_hi,
      int              flags
)
```

Parameters

→	`pwm`	Address of the intended PWM device
→	`Period_us`	Period in micro-seconds
→	`duty_pct`	Array of duty cycles in percent. Ex: 20, 50
→	`duty_time_ms`	Time for each duty cycle in milli-seconds
→	`start_idx`	Start index in look-up table from 0 to MAX -1
→	`idx_len`	Number of index
→	`pause_lo`	Pause time in milli-seconds at low index
→	`pause_hi`	Pause time in milli-seconds at high index
→	`flags`	Control flags

Return value

- ENODEV – No valid PWM device available.
- EINVAL – Invalid PWM ID, invalid duty percentage, unsupported LPG module, wrong LUT size or index, exceed LUT limit, or out-of-range period.
- 0 – Success

11.10 pm8xxx_pwm_lut_enable

Controls a PWM device to start/stop LUT ramp.

Prototype

```
int pm8xxx_pwm_lut_enable
(
      struct pwm_device    *pwm,
      int             start
)
```

Parameters

→	`pwm`	Address of the intended PWM device
→	`start`	▪ 1 – Start ▪ 0 – Stop

Return value

- ENODEV – No valid PWM device available
- EINVAL – Invalid PWM ID, unsupported LPG module
- 0 – Success

12 LED API

12.1 Introduction

There is a Linux class or framework driver for LEDs. The Linux standard LED trigger mechanism and API are recommended to access LED functions.

Example of using LED trigger for an LED:

- Declare a default_trigger for a LED in a target's board file for the LED driver.
- Define a static 'struct led_trigger *' variable in your driver.
- Register it using an API such as led_register_trigger_simple() in your driver.
- Trigger the LED with an API such as led_trigger_event() in your driver.
  Up to 256 levels of LED brightness can be supported with 0 to be LED_OFF.

Header files

linux/leds.h

linux/leds-pm8xxx.h

12.2 led_trigger_register_simple

Registers a simple trigger for the name which should be existant on a target.

Prototype

```
void led_trigger_register_simple(
      const char *name,
      struct led_trigger **trigger
)
```

Parameters

→	`name`	The name of the trigger to be registered, which should match the declared default_trigger on a target
→	`trigger`	The pointer to a 'struct led_trigger *', which would be filled and returned

Return value

None.

12.3 led_trigger_event

Sends an LED brightness event to the LED driver, and, if necessary, cancels the software blink timer that implements blinking when the hardware does not.

Prototype

```
void led_trigger_event(
      struct led_trigger *trigger,
      enum led_brightness event
)
```

Parameters

→	`trigger`	The trigger to send a brightness event
→	`brightness`	The brightness event to to trigger

Return value

None.

12.4 led_trigger_blink

Sets blinking, with software fallback blinking; it attempts to use the hardware acceleration if possible, but falls back to software blinking if there is no hardware blinking or if the LED refuses the passed values.

Prototype

```
void led_trigger_blink(
      struct led_trigger *trigger,
      unsigned long *delay_on,
      unsigned long *delay_off
)
```

Parameters

→	`trigger`	The trigger to start blinking
→	`delay_on`	The time it should be on (in ms)
→	`delay_off`	The time it should ble off (in ms)

Return value

None.

12.5 LED configuration

The LED must be configured using the driver's platform data. Besides 'struct led_platform_data' required by the LED class (or framework) driver, a PMIC chip specific 'struct pm8xxx_led_config' must be filled as well.

```
struct pm8xxx_led_config {
        u8      id;
        u8      mode;
        u16     max_current;
        int     pwm_channel;
        u32     pwm_period_us;
        bool    default_state;
        struct pm8xxx_pwm_duty_cycles *pwm_duty_cycles;
        struct wled_config_data *wled_cfg;
};
```

The LED driver has dependence on the PWM configuration if PWM mode is configured for an LED.

Complex LED, like WLED, has its own struct wled_config_data, which must be configured if being used on a platform.

13 Vibrator API

13.1 Introduction

Vibrator is a timed_output device driver. The timed_output class driver is a staging driver located at:

- drivers/staging/android/timed_output.[hc]

Use the timed_output class driver API when available. Use the proprietary API for configuration.

13.2 pm8xxx_vibrator_config

Configures the vibrator.

Prototype

```
int pm8xxx_vibrator_config
(
      struct pm8xxx_vib_config       *vib_config,
)
```

Parameters

→	`vib_config`	Pointer to pm8xxx_vib_config data structure

Return value

- ERRNO – Error value
- 0 – Success

14 Speaker API

14.1 Introduction

Class D Speaker amplifier can be provided by a PMIC chip. The proprietary API is supported by the pm8xxx-spk driver.

Header file

- linux/mfd/pm8xxx/spk.h

14.2 pm8xxx_spk_enable

Enables or disables a speaker.

Prototype

```
int pm8xxx_spk_enable
(
      int    enable
)
```

Parameters

→	`enable`	▪ 1 – Enable a speaker ▪ 0 – disable a speaker

Return value

- ERRNO – Error value
- 0 – Success

14.3 pm8xxx_spk_mute

Enables mute or unmute mode of a speaker.

Prototype

```
int pm8xxx_spk_mute
(
      bool   mute
)
```

Parameters

→	`mute`	▪ True – Mute ▪ False – Unmute

Return value

- ERRNO – Error value
- 0 – Success

14.4 pm8xxx_spk_gain

Sets gain for a speaker.

Prototype

```
int pm8xxx_spk_gain
(
      u8    gain
)
```

Parameters

→	`gain`	Set the gain for a speaker

Return value

- ERRNO – Error value
- 0 – Success

15 PMIC Thermal API

15.1 Introduction

This is the thermal driver for the PMIC die thermistor. Many PMIC chips have a three-stage thermistor to sense a range of temperatures. If an ADC channel is allocated to sense the die temperature, the PMIC thermal driver can be configured to read the ADC for real-time temperature report.

The PMIC thermal driver is registered with Linux Thermal class driver to handle one thermal zone.

Use the Linux Thermal class API in the user space for thermal management.

15.2 pm8xxx_tm_core_data

This data structure can be modified for a board's specific configures.

Prototype

```
struct pm8xxx_tm_core_data {
      int                  adc_channel;
      unsigned long        default_no_adc_temp;
      enum pm8xxx_tm_adc_type   adc_type;
…
};)
```

Parameters

→	adc_channel	If an ADC channel is connected for PMIC die thermistor, specify it
→	adc_type	enum pm8xxx_tm_adc_type
→	default_no_adc_temp	Default temperature if no ADC is used to read real time temp

16 RTC API

16.1 Introduction

Each PMIC has one RTC (Real Time Clock) module. The PMIC RTC device driver is registered with the Linux RTC class driver.

Use the Linux RTC class API from the user space for real-time clock service.

The PMIC RTC alarm can be configured to power up the device when the device is off or in shutdown mode. This feature may be disabled because of no UI support.

17 PMIC Clock API

17.1 Introduction

Some PMIC chips may provide clocks derived from the XO clock source. A simple and proprietary API is supported to configure such a clock, e.g., an MP3 clock.

Header file

- linux/mfd/pm8xxx/misc.h

17.2 pm8xxx_aux_clk_control

Configures the specified PMIC clock.

Prototype

```
int pm8xxx_aux_clk_control
(
      enum pm8xxx_aux_clk_id         clk_id,
      enum pm8xxx_aux_clk_div        divider,
      bool                      enable
)
```

Parameters

→	`clk_id`	clk_id to specify which PMIC clock
→	`divider`	The divider to get the desired clock frequency
→	`enable`	To enable or disable this clock

Return value

- ERRNO – Error value
- 0 – Success

18 PMIC Power-key API

18.1 Introduction

Each PMIC has a KPDPWR line connected to the power key of the device. This key serves two purposes:

- Power-on source
- Standard Linux key after boot-up; it uses the Linux-input framework to report keypress and release events to the userspace

The configurable parameters provided by the driver for this key are :

- Debounce time
- Default pull-up and pull-down configuration
- Wakeup capability

18.2 Configuration

Power key parameters can be configured by using driver's platform data. This is the struct of platform data:

```
struct pm8xxx_pwrkey_platform_data  {
        bool pull_up;
        u32  kpd_trigger_delay_us;
        u32  wakeup;
};
```

where

- pull up – Power on register control for pull up/down configuration
- kpd_trigger_delay_us – Time delay for power key state change interrupt trigger, i.e., debounce time
- wakeup – Configure power key as wakeup source

19 PMIC Keypad API

19.1 Introduction

Some PMICs have a built-in keypad controller (8 x 18 – matrix configuration) to support a physical keypad. The PMIC GPIOs can be configured as sense (GPIOs 1 to 8) and drive (GPIOs 9 to 23) lines for the keypad controller.

- The minimum drive X sense (row X column) configuration is 5 x 5.
- The key-mapping specific to the device (row, column, key type) can be specified to the driver.

The keypad driver uses the linux-input framework to report the key press and release events to the userspace.

19.2 Keypad configuration

The important parameters to be specified for the keypad configuration are:

```
struct pm8xxx_keypad_platform_data  {
        .num_rows      = <rows in matrix configuration>,
        .num_cols      = <columns in matrix configuration>,
        .rows_gpio_start = PM8XXX_GPIO_PM_TO_SYS(<gpio number>),
        .cols_gpio_start = PM8XXX_GPIO_PM_TO_SYS(<gpio number>),
        .keymap_data     =  <keymap of the device>,
}
```

The keymap for the device is specified as an array of integers, with each entry encoded as:

```
KEY(row, column, linux keycode)

Example:
KEY(0, 0, KEY_VOLUMEUP)
KEY(0, 1, KEY_BACK)

KEY is defined in: include/linux/input/matrix_keypad.h
```

20 Miscellaneous API

20.1 Introduction

Miscellaneous features are usually supported by a power-on module and other small hardware modules.

20.2 pm8xxx_reset_pwr_off

Switches all PM8XXX PMIC chips attached to the system to either reset or shutdown when they are turned off.

Prototype

```
Int   pm8xxx_reset_pwr_off
(
      int    reset
)
```

Parameters

→	`reset`	▪ 0 – Shutdown PMIC ▪ 1 – Shutdown, then restart PMIC

Return value

- ERRNO – Error value
- 0 – Success

20.3 pm8xxx_smpl_control

Enables or disables the SMPL detection module. If SMPL detection is enabled, the PMIC will automatically reset itself when a sufficiently long power loss event occurs. If SMPL detection is disabled, the PMIC will shut down when power loss occurs.

Prototype

```
int pm8xxx_smpl_control
(
      int    enable
)
```

Parameters

→	`enable`	▪ 0 – Shutdown PMIC on power loss ▪ 1 – Reset PMIC on power loss

Return value

- ERRNO – Error value
- 0 – Success

20.4 pm8xxx_smpl_set_delay

Sets the time delay of the SMPL detection module. If power is reapplied within this interval, the PMIC resets automatically. The SMPL detection module must be enabled for this delay time to take effect.

Prototype

```
int pm8xxx_smpl_set_delay
(
      Enum pm8xxx_smpl_delay    delay
)
```

Parameters

→	`delay`	▪ PM8XXX_SMPL_DELAY_0p5 – 0.5 sec ▪ PM8XXX_SMPL_DELAY_1p0 – 1 sec ▪ PM8XXX_SMPL_DELAY_1p5 – 1.5 sec ▪ PM8XXX_SMPL_DELAY_2p0 – 2 sec

Return value

- ERRNO – Error value
- 0 – Success

20.5 pm8xxx_coincell_chg_config

Enables or disables the coincell charger, and configures its voltage and resistor settings.

Prototype

```
int pm8xxx_coincell_chg_config
(
      struct pm8xxx_coincell_chg     *chg_config
)
```

Parameters

→	`chg_config`	Pointer to pm8xxx_coincell_chg data structure

Return value

- ERRNO – Error value
- 0 – Success

20.6 pm8xxx_watchdog_reset_control

Enables or disables the PMIC watchdog reset detection feature. If watchdog reset detection is enabled, the PMIC will reset itself when PS_HOLD goes low. If it is not enabled, the PMIC will shut down when PS_HOLD goes low.

Prototype

```
int pm8xxx_watchdog_reset_control
(
      int         enable
)
```

Parameters

→	enable	▪ 0 – Shutdown PMIC when PS_HOLD goes low ▪ 1 – Reset PMIC when PS_HOLD goes low

Return value

- ERRNO – Error value
- 0 – Success

20.7 pm8xxx_stay_on

Enables the PMIC stay-on feature, which allows the PMIC to ignore the MSM™ PS_HOLD=low signal so that special functions, such as debugging, can be performed. This feature should not be used in any product release.

Prototype

```
int pm8xxx_stay_on
(
      void
)
```

Return value

- ERRNO – Error value
- 0 – Success

20.8 pm8xxx_hard_reset_config

Allows different reset configurations.

Prototype

```
int pm8xxx_hard_reset_config
(
      enum pm8xxx_pon_config         config
)
```

Parameters

→	config	▪ PM8XXX_DISABLE_HARD_RESET – Disable hard reset ▪ PM8XXX_SHUTDOWN_ON_HARD_RESET – Turn off system on hard reset ▪ PM8XXX_RESTART_ON_HARD_RESET – Restart system on hard reset

Return value

- ERRNO – Error value
- 0 – Success

20.9 pm8xxx_uart_gpio_mux_ctrl

Configures the UART path.

Prototype

```
int pm8xxx_uart_gpio_mux_ctrl
(
      enum pm8xxx_uart_path_sel      uart_path_sel
)
```

Parameters

→	uart_path_del	▪ UART_NONE ▪ UART_TX1_RX1 ▪ UART_TX2_RX2 ▪ UART_TX3_RX3

Return value

- ERRNO – Error value
- 0 – Success

20.10 pm8xxx_preload_dvdd

Helps reduce fluctuations in the dVdd voltage during startup at the cost of additional Off-state current draw. This API should only be called if dVdd startup issues are suspected.

Prototype

```
int pm8xxx_preload_dvdd
(
      void
)
```

Return value

- ERRNO – Error value
- 0 – Success

EXHIBIT 1

PLEASE READ THIS LICENSE AGREEMENT ("AGREEMENT") CAREFULLY. THIS AGREEMENT IS A BINDING LEGAL AGREEMENT ENTERED INTO BY AND BETWEEN YOU (OR IF YOU ARE ENTERING INTO THIS AGREEMENT ON BEHALF OF AN ENTITY, THEN THE ENTITY THAT YOU REPRESENT) AND QUALCOMM TECHNOLOGIES, INC. ("QTI" "WE" "OUR" OR "US"). THIS IS THE AGREEMENT THAT APPLIES TO YOUR USE OF THE DESIGNATED AND/OR ATTACHED DOCUMENTATION AND ANY UPDATES OR IMPROVEMENTS THEREOF (COLLECTIVELY, "MATERIALS"). BY USING OR COMPLETING THE INSTALLATION OF THE MATERIALS, YOU ARE ACCEPTING THIS AGREEMENT AND YOU AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS, QTI IS UNWILLING TO AND DOES NOT LICENSE THE MATERIALS TO YOU. IF YOU DO NOT AGREE TO THESE TERMS YOU MUST DISCONTINUE AND YOU MAY NOT USE THE MATERIALS OR RETAIN ANY COPIES OF THE MATERIALS. ANY USE OR POSSESSION OF THE MATERIALS BY YOU IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

1.1 **License.** Subject to the terms and conditions of this Agreement, including, without limitation, the restrictions, conditions, limitations and exclusions set forth in this Agreement, Qualcomm Technologies, Inc. ("QTI") hereby grants to you a nonexclusive, limited license under QTI's copyrights to use the attached Materials; and to reproduce and redistribute a reasonable number of copies of the Materials. You may not use Qualcomm Technologies or its affiliates or subsidiaries name, logo or trademarks; and copyright, trademark, patent and any other notices that appear on the Materials may not be removed or obscured. QTI shall be free to use suggestions, feedback or other information received from You, without obligation of any kind to You. QTI may immediately terminate this Agreement upon your breach. Upon termination of this Agreement, Sections 1.2-4 shall survive.

1.2 **Indemnification.** You agree to indemnify and hold harmless QTI and its officers, directors, employees and successors and assigns against any and all third party claims, demands, causes of action, losses, liabilities, damages, costs and expenses, incurred by QTI (including but not limited to costs of defense, investigation and reasonable attorney's fees) arising out of, resulting from or related to: (i) any breach of this Agreement by You; and (ii) your acts, omissions, products and services. If requested by QTI, You agree to defend QTI in connection with any third party claims, demands, or causes of action resulting from, arising out of or in connection with any of the foregoing.

1.3 **Ownership.** QTI (or its licensors) shall retain title and all ownership rights in and to the Materials and all copies thereof, and nothing herein shall be deemed to grant any right to You under any of QTI's or its affiliates' patents. You shall not subject the Materials to any third party license terms (e.g., open source license terms). You shall not use the Materials for the purpose of identifying or providing evidence to support any potential patent infringement claim against QTI, its affiliates, or any of QTI's or QTI's affiliates' suppliers and/or direct or indirect customers. QTI hereby reserves all rights not expressly granted herein.

1.4 **WARRANTY DISCLAIMER.** YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT THE USE OF THE MATERIALS IS AT YOUR SOLE RISK. THE MATERIALS AND TECHNICAL SUPPORT, IF ANY, ARE PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED. QTI ITS LICENSORS AND AFFILIATES MAKE NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE MATERIALS OR ANY OTHER INFORMATION OR DOCUMENTATION PROVIDED UNDER THIS AGREEMENT, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT, OR ANY EXPRESS OR IMPLIED WARRANTY ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. NOTHING CONTAINED IN THIS AGREEMENT SHALL BE CONSTRUED AS (I) A WARRANTY OR REPRESENTATION BY QTI, ITS LICENSORS OR AFFILIATES AS TO THE VALIDITY OR SCOPE OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT OR (II) A WARRANTY OR REPRESENTATION BY QTI THAT ANY MANUFACTURE OR USE WILL BE FREE FROM INFRINGEMENT OF PATENTS, COPYRIGHTS OR OTHER INTELLECTUAL PROPERTY RIGHTS OF OTHERS, AND IT SHALL BE THE SOLE RESPONSIBILITY OF YOU TO MAKE SUCH DETERMINATION AS IS NECESSARY WITH RESPECT TO THE ACQUISITION OF LICENSES UNDER PATENTS AND OTHER INTELLECTUAL PROPERTY OF THIRD PARTIES.

1.5 **LIMITATION OF LIABILITY.** IN NO EVENT SHALL QTI, QTI'S AFFILIATES OR ITS LICENSORS BE LIABLE TO YOU FOR ANY INCIDENTAL, CONSEQUENTIAL OR SPECIAL DAMAGES, INCLUDING BUT NOT LIMITED TO ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE, OR THE DELIVERY OR FAILURE TO DELIVER, ANY OF THE MATERIALS, OR ANY BREACH OF ANY OBLIGATION UNDER THIS AGREEMENT, EVEN IF QTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE FOREGOING LIMITATION OF LIABILITY SHALL REMAIN IN FULL FORCE AND EFFECT REGARDLESS OF WHETHER YOUR REMEDIES HEREUNDER ARE DETERMINED TO HAVE FAILED OF THEIR ESSENTIAL PURPOSE. THE ENTIRE LIABILITY OF QTI, QTI's AFFILIATES AND ITS LICENSORS, AND THE SOLE AND EXCLUSIVE REMEDY OF YOU, FOR ANY CLAIM OR CAUSE OF ACTION ARISING HEREUNDER (WHETHER IN CONTRACT, TORT, OR OTHERWISE) SHALL NOT EXCEED US$10.

2. **COMPLIANCE WITH LAWS; APPLICABLE LAW.** You agree to comply with all applicable local, international and national laws and regulations and with U.S. Export Administration Regulations, as they apply to the subject matter of this Agreement. This Agreement is governed by the laws of the State of California, excluding California's choice of law rules.

3. **CONTRACTING PARTIES.** If the Materials are downloaded on any computer owned by a corporation or other legal entity, then this Agreement is formed by and between QTI and such entity. The individual accepting the terms of this Agreement represents and warrants to QTI that they have the authority to bind such entity to the terms and conditions of this Agreement.

4. **MISCELLANEOUS PROVISIONS.** This Agreement, together with all exhibits attached hereto, which are incorporated herein by this reference, constitutes the entire agreement between QTI and You and supersedes all prior negotiations, representations and agreements between the parties with respect to the subject matter hereof. No addition or modification of this Agreement shall be effective unless made in writing and signed by the respective representatives of QTI and You. The restrictions, limitations, exclusions and conditions set forth in this Agreement shall apply even if QTI or any of its affiliates becomes aware of or fails to act in a manner to address any violation or failure to comply therewith. You hereby acknowledge and agree that the restrictions, limitations, conditions and exclusions imposed in this Agreement on the rights granted in this Agreement are not a derogation of the benefits of such rights. You further acknowledges that, in the absence of such restrictions, limitations, conditions and exclusions, QTI would not have entered into this Agreement with You. Each party shall be responsible for and shall bear its own expenses in connection with this Agreement. If any of the provisions of this Agreement are determined to be invalid, illegal, or otherwise unenforceable, the remaining provisions shall remain in full force and effect. This Agreement is entered into solely in the English language, and if for any reason any other language version is prepared by any party, it shall be solely for convenience and the English version shall govern and control all aspects. If You are located in the province of Quebec, Canada, the following applies: The Parties hereby confirm they have requested this Agreement and all related documents be prepared in English.