PM8921™ PMIC Modem

Interface Specification and Operational Description

80-N2283-25 A

August 9, 2011

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

Qualcomm Confidential and Proprietary

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Contents

Figures

Tables

Revision history

Revision	Date	Description
A	Aug 2011	Initial release

1 Introduction

1.1 Purpose

This document presents the software driver for Qualcomm's PM8921™ Power Management Integrated Circuit (PMIC).

NOTE: The APIs presented in this document are mainly for the MSM8960™ ASIC. For other targets/platforms using the PM8921, see the PMIC document for the specific target/platform.

1.2 Scope

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

1.3 Conventions

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

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

Parameter types are indicated by arrows:

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

1.4 References

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 Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*MSM8960™ Chipset (RTR860X, PM8921™, WCD9310, WCN3660) Schematics and Design Guidelines*	80-N1622-5

1.5 Technical assistance

For assistance or clarification on information in this guide, submit a case to Qualcomm CDMA Technologies at https://support.cdmatech.com/.

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

1.6 Acronyms

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

2 Functional Overview

The PM8921 device (shown in Figure 2-1) integrates all wireless handset power management, general housekeeping, and user interface support functions into a single mixed-signal IC. Its versatile design is suitable for CDMA and non-CDMA handsets, as well as other wireless products, such as PC cards, modems, PDAs, etc.

Figure 2-1 is a PM8921 IC functional block diagram.

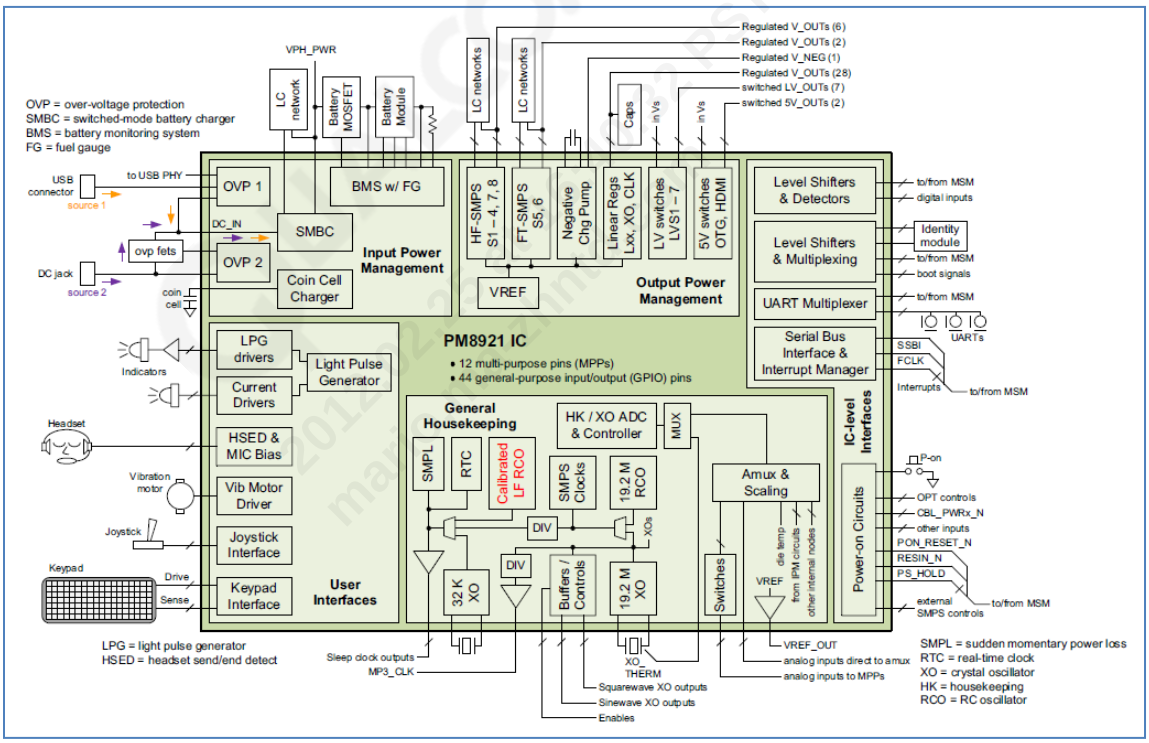

Figure 2-1 PM8921 IC functional block diagram

2.1 Power management

The power management portion accepts power from common sources (such as battery, external charger, adapter, USB_VBUS, coin cell backup) and generates all the regulated voltages needed to power the appropriate handset electronics. It monitors and controls the power sources, detecting which sources are applied, and verifying that they are within acceptable operational limits. It also coordinates battery and coin cell recharging while maintaining the handset electronics supply voltages.

2.2 Voltage regulation

On-chip voltage regulators generate 37 programmable output voltages using a combination of switched-mode power supplies and low dropout voltage regulators, all derived from a common trimmed voltage reference. Independent regulated power sources support various electronic functions (while avoiding signal corruption between diverse circuits), allow power management sequencing, and meet different voltage level requirements.

2.3 General housekeeping

The device's general housekeeping functions include a 16-position analog multiplexer that has five internal connections and supports connections to 18 external pins. The five internal connections are used to monitor on-chip functions, such as the temperature sensor. Six external connections are hardwired to access input power nodes, such as Charger Voltage (VCHG), Battery Voltage (VBATT), etc.

2.4 Inputs and outputs

The IC includes 12 Multipurpose Pins (MPPs) that can be configured as analog inputs that are routed by an analog switch network to create five multiplexer inputs. These inputs are available to monitor system parameters, such as temperature and battery ID.

The multiplexer output's offset and gain are adjusted, and then buffered and routed to the Mobile Station Modem™ (MSM™) device for analog-to-digital conversion.

2.5 Clocks and counters

Various oscillator, clock, and counter circuits are provided to initialize and maintain valid pulse waveforms and measure time intervals for higher-level handset functions.

2.6 Temperature control

A dedicated controller manages the Temperature-Compensated Crystal Oscillator (TCXO) warm-up and signal buffering, and key Parameters are monitored to protect against detrimental conditions.

2.7 Handset-level user interfaces

Handset-level user interfaces are also supported. The IC includes six backlight or LED drivers with brightness (current) control that can be used for keypad, LCD, camera Flash, and general purpose drivers.

A vibration motor driver alerts handset users of incoming calls, and a speaker driver with volume control can be used for audio alerts or speakerphone and melody ringer applications.

2.8 IC-level interfaces

IC-level interfaces include the three-line Serial Bus Interface (SBI) used by the MSM device to control and report the status of the PM8921 IC. This bus is supplemented by an interrupt manager for time-critical information.

Another dedicated IC interface circuit monitors multiple trigger events and controls the power-on/power-off sequences.

A Universal Serial Bus On The Go (USB-OTG) transceiver is included for interfacing the MSM device to computers as a USB peripheral, or connecting the MSM to other peripherals. Reusable Identification Module (RUIM)-level translators allow the MSM to interface with external modules.

3 PMIC Interface

The three-line SBI allows efficient initialization, control of device operating modes and parameters, and verification of programmed parameters. The MSM's SBI controller is the Master and the PM8921 IC is a Slave. The three SBI pins have dedicated functions:

- The Serial Bus Strobe Signal (SBST) – Used to initiate serial data transfers
- The Serial Bus Data Transfer (SBDT) line – Bidirectional and is used to transfer data into and out of the PM8921 IC
- The Serial Bus Clock (SBCK) – Synchronizes data transfers

Handset designers use the Application Programming Interface (API) to program the PM8921 IC, indirectly exercising the SBI. A software designer only needs to include pm.h to access the PM8921 API.

3.1 Power-on initialization API

While the PS_HOLD signal from the MSM is low, the PM8921 IC is in one of its OFF states. Under this condition, the power-on circuits continually monitor five events that can trigger a power-on sequence:

- The keypad power-on button is pressed and the KPD_PWR_N signal is pulled low; PM_KPD_PWR_KEY_ON_IRQ_HDL
- An external supply source is detected (the voltage on either the VCHG or USB_VBUS pin exceeds its valid threshold); PM_VALID_CHG_IRQ_HDL, PM_VALID_USB_CHG_IRQ_HDL
- The real-time clock alarm is triggered; PM_RTC_ALRM_IRQ_HDL
- A serial cable is inserted and both the CBL0PWR_N (MPP3) and CBL1PWR_N (MPP4) pins are pulled low; PM_CABLE_IN_IRQ_HDL
- A Sudden Momentary Power Loss (SMPL) condition is detected and an SMPL recovery is initiated; PM_SMPL_IRQ_HDL

3.1.1 pm_init()

This function performs the *early stage* initialization of the PMIC for operation. It is up to the user to change this function to target user needs.

Parameters

None

Description

This function initializes the PM60x0 for operation; it:

- Initializes the PMIC software driver memory
- Initializes the LDOs voltages
- Enables and starts the 32 kHz external sleep crystal

Return value

None

Dependencies

rflib_init() must have been called.

Side effects

Interrupts are disabled while writing and reading to/from SBI.

Example

None

3.1.2 pm_init_delayed()

Parameters

None

Description

This function initializes the PM60x0 ISR services. It disables all PMIC IRQs and registers the PMIC ISR with the GPIO software driver.

Return value

None

Dependencies

This function should be called after executing the following functions:

- pm_init()
- tramp_init()
- gpio_int_init()

Side effects

None

Example

3.1.3 is_pm_init_done ()

Parameters

None

Description

This function is used by outside clients to determine if the PMIC software driver has been initialized.

Return value

Boolean:

- TRUE
- FALSE

Dependencies

None

Side effects

Example

3.1.4 boot_pm_init ()

Parameters

```
unsigned int boot_pm_init
(
 void* vptr  ,
unsigned char numComms
 )
```

→	*vptr	Void pointer – This is a void pointer that can be utilized; more information is needed by the boot build PMIC software.
→	numComms	Unsigned char – This is the number of devices that are going to be initialized during the boot sequence.

Description

This function is used by outside clients to determine if the PMIC software driver has been initialized.

Return value

Unsigned int – Number of errors that occurred during initialization

Dependencies

Boot build

NOTE: This function should only be called during boot initialization.

Side effects

Example

4 PMIC Model API

4.1 pm_get_pmic_model()

This function returns the PMIC's model name, e.g., PM6610™, PM6650™, PM7540™, PM8921™, etc.

Parameters

None

Description

Use this function to return the PMIC model name.

Return value

pm_model_type indicates the PMIC chip model; it should return PMIC_IS_PM8921 when PM8921 is used.

Dependencies

This function should be called after executing pm_init() function.

Side effects

None

Example

```
pm_model_type    pm_model = PMIC_IS_INVALID;
/* Get The PMIC model (model/tier = HT, VLT, etc). */
pm_model = pm_get_pmic_model();
switch( pm_model )
{
    Case PMIC_IS_PM7540:
         …
         Break;
    Case PMIC_IS_PM8028:
         …
         Break;
    Case PMIC_IS_PM8921:
         …
         Break;
    default:
         …
         Break;
}
```

4.2 pm_get_pmic_revision()

This function returns the PMIC's chip revision, e.g., A0, B1, etc.

Parameters

None

Description

Use this function to return the PMIC revision ID.

Return value

uint8 variable represent PMIC revision ID.

Dependencies

This function should be called after executing pm_init() function.

Side effects

None

Example

```
uint8 pmic_rev_id = pm_get_pmic_revision();
if (pmic_rev_id < 2)
{
    //What if feature not supported
    …
}
```

5 Interrupt Manager API

The PM8921 Interrupt Manager supports the MSM's interrupt processing. Each interrupt event has the following associated SBI bits:

- Interrupt Mask (read/write) – When set, this bit allows the MSM device to ignore the interrupting event. Setting the mask bit prevents the PM8921 IC from generating a latched status and driving the MSM_INT_N signal low.
- Interrupt Real-Time Status (read only) – This bit follows the real-time interrupt event status (active or inactive).
- Interrupt Latched Status (read only) – This bit is set when the interrupt event is active and the interrupt mask bit is cleared. It stays set until the interrupt clear bit is set. This bit can only be read while the interrupt mask bit is cleared.
- Interrupt Clear (read/write) – Setting this bit clears the interrupt event's latched status. It is cleared automatically after the latched status is read, which has no effect other than allowing it to be set later to again clear the event's latched status.

When one or more interrupt events occur that are not masked, the MSM_INT_N signal is forced low to notify the MSM that at least one interrupt has occurred. This signal stays low until the corresponding interrupts are cleared by the MSM through the appropriate clear bits. When all interrupts have been cleared (or masked) by the MSM device, the MSM_INT_N signal returns high.

There are four special interrupts, OVERTEMP, RTCRST, SMPL, and WDOG. Special interrupts are not cleared by a PM8921 reset, do not provide true real-time status bits, and provide latched versions of the real-time status bits whether the Mask bit is set. These special characteristics are necessary because the associated interrupts often occur just before or during a PM8921 IC reset. The IC latches the interrupt just before resetting, and these particular interrupts are backed up by the coin cell. When the PM8921 IC restarts, the latched interrupts inform the MSM that they were triggered.

NOTE: The transparent latch holds the interrupt event during the read strobe even if it transitions midway through the read.

Table 5-1 lists the IRQs available in PM8921.

Table 5-1 PM8921 IRQ list

IRQ
PM_VALID_CHG_RT_ST
PM_INVALID_CHG_RT_ST
PM_BATT_REPLACE_RT_ST
PM_VBATT_DET_LOW_RT_ST

IRQ
PM_VCP_RT_ST
PM_T_STAT_CHANGED_RT_ST
PM_BAT_STAT_CHANGED_RT_ST
PM_VBAT_DET_RT_ST
PM_BAT_FET_ON_RT_ST
PM_KPD_PWR_KEY_ON_RT_ST
PM_KPD_PWR_KEY_OFF_RT_ST
PM_RTC_ALRM_RT_ST
PM_PWR_RST_RT_ST
PM_SMPL_RT_ST
PM_CABLE_IN_RT_ST
PM_OVER_TEMP_RT_ST
PM_KPD_PWRON_EVENT_RT_ST
PM_MPP01_CHGED_ST_RT_ST
PM_MPP02_CHGED_ST_RT_ST
PM_MPP03_CHGED_ST_RT_ST
PM_MPP04_CHGED_ST_RT_ST
PM_MPP05_CHGED_ST_RT_ST
PM_MPP06_CHGED_ST_RT_ST
PM_MPP07_CHGED_ST_RT_ST
PM_MPP08_CHGED_ST_RT_ST
PM_MPP10_CHGED_ST_RT_ST
PM_MPP11_CHGED_ST_RT_ST
PM_MPP12_CHGED_ST_RT_ST
PM_GPIO1_CHGED_ST_RT_ST
PM_GPIO2_CHGED_ST_RT_ST
PM_GPIO3_CHGED_ST_RT_ST
PM_GPIO4_CHGED_ST_RT_ST
PM_HSEDOUT_NC_0_RT_ST
PM_HSEDOUT_NO_0_RT_ST
PM_HSEDOUT_NC_1_RT_ST
PM_HSEDOUT_NO_1_RT_ST
PM_HSEDOUT_NC_2_RT_ST
PM_HSEDOUT_NO_2_RT_ST
PM_ATCDONE_RT_ST
PM_ATCFAIL_RT_ST
PM_CHGDONE_RT_ST
PM_CHGFAIL_RT_ST
PM_CHGSTATE_RT_ST
PM_FASTCHG_RT_ST
PM_BATTCONNECT_RT_ST
PM_BATTTEMP_RT_ST

IRQ
PM_CHGHOT_RT_ST
PM_CHGGONE_RT_ST
PM_VALID_CHG_IRQ_HDL
PM_INVALID_CHG_IRQ_HDL
PM_BAT_STAT_CHANGED_IRQ_HDL
PM_CHG_WDOG_IRQ_HDL
PM_CHGDONE_IRQ_HDL
PM_CHGFAIL_IRQ_HDL
PM_CHGSTATE_IRQ_HDL
PM_FASTCHG_IRQ_HDL
PM_TRKLCHG_IRQ_HDL
PM_VBAT_DET_IRQ_HDL
PM_VBAT_OV_IRQ_HDL
PM_BATTCONNECT_IRQ_HDL
PM_BATTREMOVE_IRQ_HDL
PM_BATTTEMP_IRQ_HDL
PM_COLDBATT_IRQ_HDL
PM_HOTBATT_IRQ_HDL
PM_VCP_IRQ_HDL
PM_MPP11_CHGED_ST_IRQ_HDL
PM_HSED_NC_1_IRQ_ID
PM_HSED_NO_1_IRQ_ID
PM_KPD_PWROFF_IRQ_ID
PM_KPD_PWRON_IRQ_ID
PM_KPD_STATE_CHANGED_IRQ_ID
PM_KPD_STUCK_IRQ_ID
PM_KPD_SWITCH_IRQ_ID
PM_ADC_EOC_TO_BTM_IRQ
PM_ADC_EOC_TO_USR_IRQ_ID
PM_ADC_EOC_TO_SEC_IRQ_ID
PM_ADC_EOC_TO_MDM_IRQ_ID
PM_WDOG_TOUT_IRQ_ID
PM_UICC1_IRQ_ID
PM_UICC2_IRQ_ID
PM__8901_MPP01_CHGED_ST

5.1 pm_set_irq_handle ()

This function register a PMIC interrupt service routine.

Parameters

```
pm_err_flag_type pm_set_irq_handle
(
  pm_irq_hdl_type       irq_id,
  pm_isr_ptr_type       handler
)
```

→	irq_id	pm_irq_hdl_type – See Table 5-1
→	handler	pm_isr_ptr_type – The function that will service the IRQ (ISR)

Description

Use this function to:

- Enable the desired PMIC IRQ and assign to it an IRQ
- Disable the desired PMIC IRQ by assigning to it a NULL interrupt service routine

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED

Dependencies

Side effects

Example

5.2 pm_get_rt_status ()

This function gets the real-time status on a PMIC interrupt.

Parameters

```
pm_err_flag_type pm_get_rt_status
(
   pm_irq_hdl_type   rt_status_id,
   boolean           *rt_status
)
```

→	rt_status_id	pm_irq_hdl_type – See to Table 5-1
→	*rt_status	Pointer of boolean: ▪ TRUE – Interrupt is triggered ▪ FALSE – Interrupt is not triggered

Description

This function returns the current bit-wise (not latch) status of the select PMIC IRQ. Use this function for polling the desired PMIC real-time status bit if the IRQ is not enabled. An interrupt will not be triggered until the corresponding IRQ is enabled.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED

Dependencies

Side effects

Example

5.3 pm_get_irq_status ()

This function finds out if an interrupt is triggered.

Parameters

```
pm_err_flag_type pm_get_irq_status
(
   pm_irq_hdl_type irq_id,
   boolean         *status
)
```

→	irq_id	pm_irq_hdl_type – See Table 5-1
→	*status	Pointer of boolean: ▪ TRUE – Interrupt is triggered ▪ FALSE – Interrupt is not triggered

Description

This function returns the status of the different PMIC IRQs. An interrupt will not be triggered until the corresponding IRQ is enabled. The IRQ status bit will stay active until the corresponding IRQ is cleared.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED

Dependencies

Side effects

Example

5.4 pm_clear_irq ()

This function resets the selected IRQ.

Parameters

```
pm_err_flag_type pm_clear_irq
(
   pm_irq_hdl_type irq
)
```

→	irq	pm_irq_hdl_type – See Table 5-1

Description

This function resets/clears the selected IRQ.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED

Dependencies

Side effects

Example

5.5 pm_clear_irq_handle ()

This function clears the PMIC interrupt handler for a given IRQ ID.

Parameters

```
pm_err_flag_type pm_clear_irq_handle
(
   pm_irq_hdl_type      irq_id,     /* Which IRQ */
   pm_isr_ptr_type      handler     /* ISR to service the IRQ */
)
```

→	irq_id	pm_irq_hdl_type – See Table 5-1
→	handler	pm_isr_ptr_type – The function that will service the IRQ (ISR)

Description

This function may be called more than once to clear multiple callback functions for the same IRQ ID.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED

Dependencies

Side effects

Example

5.6 pm_config_irq()

This function configures PMIC IRQs for a specific trigger type.

Parameters

```
pm_err_flag_type pm_config_irq
(
   pm_irq_hdl_type        irq_id,     /* Which IRQ */
   pm_irq_trigger_type    irq_trigger, /* The trigger type for the IRQ. */
   pm_sec_irq_domain_type irq_domain,   /* interrupt domain SEC/USR/MDM */
   boolean                irq_permission /* The permission of the IRQ */
)
```

→	`irq_id`	pm_irq_hdl_type – See Table 5-1
→	`irq_trigger`	pm_irq_trigger_type: ▪ PM_IRQ_TRIG_BE – Both RE and FE triggered ▪ PM_IRQ_TRIG_BE_X – Both high and low level triggered ▪ PM_IRQ_TRIG_RE – RE triggered ▪ PM_IRQ_TRIG_HIGH_LVL – High level triggered ▪ PM_IRQ_TRIG_FE – FE triggered ▪ PM_IRQ_TRIG_LOW_LVL – Low level triggered ▪ PM_IRQ_TRIG_DISABLE – Disable RE and FE trigger ▪ PM_IRQ_TRIG_DISABLE_X – Disable HIGH and LOW trigger ▪ PM_IRQ_CLEAR – Clear IRQ ▪ PM_IRQ_TRIG_DEFAULT – Use default trigger settings
→	`irq_domain`	pm_sec_irq_domain_type: ▪ PM_IRQ_SEC ▪ PM_IRQ_MDM ▪ PM_IRQ_USR
→	`irq_permission`	Boolean ▪ TRUE ▪ FALSE

Description

Use this function to configure PMIC IRQs for a specific trigger type.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED

Dependencies

Side effects

Example

6 Power-on API

6.1 Cable insertion power-on

An alternate method of initiating the power-on sequence is cable insertion. This function is general purpose and is not specific to this application; however, the intention is to allow the insertion of a serial cable to power on the handset. Two pins are provided for this function, CBLPWRON1 and CBLPWRON0. To initiate a power-on sequence, both pins must be pulled to the Logic Low state and held there until the PS_HOLD is asserted by the system microprocessor. An interrupt is set to enable the microprocessor to determine the power-on triggering event. A second type of interrupt (MPP3 and MPP4 IRQs) alerts the microprocessor if the state of any one of the two pins has changed from a logic low to logic high.

This API allows you to:

- Monitor the state of pins CBLPWRON1 (MPP4) and CBLPWRON0 (MPP3):
  - Configure MPP3 and MPP4 as digital inputs. For a description of the pm_mpp_config_digital_input() function, see Section 11.1.
  - Use pm_get_rt_status() (IDs – PM_MPP3_CHGED_RT_ST and PM_MPP4_CHGED_RT_ST) to poll the current state of the cable power-on pins.
  - Use pm_set_irq_handle() (handle – PM_CABLE_IN_IRQ_HDL) to enable/disable the ability of the PM8921to alert the microprocessor if the state of any one of the two pins has changed from a logic low to logic high.
- Enable/disable the cable power-on feature through pm_cbl_power_switch().
- Enable/disable the cable power pins pull-up resistor through pm_cbl_pwr_on_pull_up_switch().

NOTE: See the corresponding user guide and device specification document to determine if this feature is supported by the PM8921.

6.1.1 pm_get_power_on_status()

This function returns the phone power-on status. It should be called at the earliest possible moment at bootup.

Parameters

```
pm_err_flag_type pm_get_power_on_status
(
    uint32* pwr_on_status
)
```

→	pwr_on_status	Power on status: ▪ PM_PWR_ON_EVENT_KEYPAD – 0x01 ▪ PM_PWR_ON_EVENT_RTC – 0x02 ▪ PM_PWR_ON_EVENT_CABLE – 0x04 ▪ PM_PWR_ON_EVENT_SMPL – 0x08 ▪ PM_PWR_ON_EVENT_USB_CHG – 0x20 ▪ (Hard Reset) – 0x100 ▪ (General-purpose trigger) – 0x200

Description

Use this function to configure the power-on status.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the pwr_on_status parameter
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – This PMIC model does not support cable power-on
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Which event causes the device to power-on */
err = pm_get_power_on_status(&pwr_on_status);

/* Find out if there was a PMIC API error. */
if (err == PM_ERR_FLAG__SUCCESS)
{
     If(pwr_on_status == PM_PWR_ON_EVENT_SMPL)
     {
            ….
     }
}
```

6.1.2 pm_clear_power_on_status()

This function clears the phone power-on status. It should be called before the phone powers down.

NOTE: In PM8921, the power-on status register is read only, so there is nothing to clear.

Parameters

```
pm_err_flag_type pm_clear_power_on_status
(
    void
)
```

Description

Use this function to clear the phone power-on status. It should be called before the phone powers down.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – This PMIC model does not support cable power on
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

- Interrupts are disabled while communicating with the PMIC.
- The microprocessor will be halted for one millisecond if option PM_CBL_PWR_ON__DIS_PERMANENTLY is selected.

Example

```
/* Clear power on-status */
err = pm_clear_power_on_status();

"Find out if there was a PMIC API error."
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

6.1.3 pm_cbl_pwr_on_pull_up_switch()

This function enables/disables the corresponding cable power-on pin pull-up resistor.

Parameters

```
pm_err_flag_type pm_cbl_pwr_on_pull_up_switch
(
    pm_switch_cmd_type          OnOff,
    pm_cbl_pwr_on_type          cable,
)
```

→	OnOff	pm_switch_cmd_type: ▪ PM_OFF_CMD – Disable the pull-up ▪ PM_ON_CMD – Enable the pull-up
→	cable	pm_cbl_pwr_on_type: ▪ PM_CBL_PWR_ON__0 – CBLPWRON0 pin ▪ PM_CBL_PWR_ON__1 – CBLPWRON1 pin

Description

Use this function to enable or disable the CBLPWRON0 and CBLPWRON1 pin pull-up resistors. The pull-up is effective only when cable power is enabled.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the cable parameter
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – This PMIC model does not support cable power-on
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
"Enable Cable Power ON pin 0 pull up resistor.”
err |= pm_cbl_pwr_on_pull_up_switch(PM_ON_CMD, PM_CBL_PWR_ON__0);

"Find out if there was a PMIC API error."
if (err != PM_ERR_FLAG__SUCCESS)
{
   MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

6.2 Watchdog reset detect

If this feature is enabled, the PMIC resets and restarts if the PS_HOLD pin goes low. This feature also triggers an IRQ if the watchdog timeout IRQ (PM_WDOG_TOUT_IRQ_HDL) is enabled.

If this feature is disabled, the PMIC shuts off if the PS_HOLD pin goes low. The IRQ does not trigger even if the IRQ is enabled.

6.2.1 pm_watchdog_reset_detect_switch()

This function enables/disables the watchdog reset detect.

Parameters

```
pm_err_flag_type pm_watchdog_reset_detect_switch
(
    pm_switch_cmd_type          OnOff,
)
```

→	OnOff	pm_switch_cmd_type: ▪ PM_OFF_CMD – Disable the detect ▪ PM_ON_CMD – Enable the detect

Description

Use this function to enable or disable the watchdog reset detect feature.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – This PMIC model does not support cable power-on
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

Interrupts are LOCKED while communicating with the PMIC.

Example

```
"Enable watchdog reset detection."
err |= pm_watchdog_reset_detect_switch(PM_ON_CMD);

"Find out if there was a PMIC API error."
if (err != PM_ERR_FLAG__SUCCESS)
{
   MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

6.2.2 pm_wdog_status_get()

This function returns the PMIC watchdog reset status.

Parameters

```
pm_err_flag_type pm_wdog_status_get
(
    boolean *dog_status
)
```

→	*dog_status	Pointer of boolean: ▪ TRUE – Watchdog reset has occurred. ▪ FALSE– No watchdog reset has occurred

Description

Use this function to read the PMIC watchdog reset status.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the dog_status parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

Interrupts are LOCKED while communicating with the PMIC.

Example

```
//Read watchdog reset status
boolean dog_status;
err = pm_wdog_status_get(&dog_status);
```

6.3 Undervoltage lockout

6.3.1 pm_uvlo_threshold_set()

This function sets the Undervoltage Lockout (UVLO) threshold voltage.

Parameters

```
pm_err_flag_type pm_uvlo_threshold_set
(
    uint16 mVolts,
)
```

→	mVolts	Undervoltage lockout threshold voltage; valid range is 1500 to 3050 mV

Description

Use this function to set the undervoltage lockout threshold voltage.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mVolts parameter
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – This PMIC model does not support cable power-on
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

```
"Set the UVLO threshold to 2250 mV ”
err |= pm_uvlo_threshold_set(2250);

"Find out if there was a PMIC API error."
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

6.3.2 pm_uvlo_delay_set()

This function sets the undervoltage lockout detection delay.

Parameters

```
pm_err_flag_type pm_uvlo_delay_set
(
    uint16 mSeconds,
)
```

→	mSeconds	Time delay for UVLO detection in milliseconds; valid range is 0 to 64 (0, 1, 2, 4, 8, 16, 32, 64)

Description

Use this function to set the undervoltage lockout detection delay.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mSeconds parameter
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – This PMIC model does not support cable power-on
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

```
"Set the uvlo delay time to 32 mSec”
err |= pm_uvlo_delay_set(32);

"Find out if there was a PMIC API error."
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

6.4 Hard reset

6.4.1 pm_pwron_hard_reset_enable()

This function controls the PMIC behavior after hard reset.

Parameters

```
pm_err_flag_type pm_pwron_hard_reset_enable
(
    boolean enable
)
```

→	enable	boolean ▪ TRUE – A power-on sequence will be orchestrated after hard reset ▪ FALSE– PMIC will remain power off after hard reset

Description

Use this API to control the PMIC behavior after hard reset, either powering up or remaining power-off.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

```
//Hard Reset Feature Configuration
(void)pm_pwron_hard_reset_enable(TRUE);
(void)pm_pwron_hard_reset_delay_timer_set(150);
(void)pm_pwron_hard_reset_debounce_timer_set(100);
```

6.4.2 pm_pwron_hard_reset_delay_timer_set()

This function is used to set the PMIC hard reset delay timer.

Parameters

```
pm_err_flag_type pm_pwron_hard_reset_delay_timer_set
(
    int delay_timer_ms
)
```

→	delay_timer_ms	Int – The value for the hard reset delay timer in milliseconds; the allowable range for this parameter is 0 to 2000 milliseconds (will be set to one of the following discrete values: 0, 10, 50, 100, 250, 500, 1000, 2000)

Description

Use this API to set the PMIC hard reset delay timer.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

Refer to the example in Section 6.4.1.

6.4.3 pm_pwron_hard_reset_debounce_timer_set()

This function is used to set the PMIC hard reset debounce timer.

Parameters

```
pm_err_flag_type pm_pwron_hard_reset_debounce_timer_set
(
    int debounce_timer_ms
)
```

→	debounce_timer_ms	Int – The hard reset debounce timer in milliseconds; the allowable range for this parameter is 0 to 10256 milliseconds (will be set to one of the following discrete values: 0, 32, 56, 80, 128, 184, 272, 408, 608, 904, 1352, 2048, 3072, 4480, 6720, 10256)

Description

Us this API to set the PMIC hard reset debounce timer.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

Refer to the example in Section 6.4.1.

7 Input Power Management

7.1 Boot Charging API

7.1.1 pm_boot_chg_set_battery_thresholds ()

This function sets the battery alarm thresholds for weak battery and dead battery.

Parameters

```
pm_err_flag_type pm_boot_chg_set_battery_thresholds
(
   uint32 dead_battery_threshold,
   uint32 weak_battery_threshold
)
```

→	dead_battery_threshold	uint32 – The dead battery threshold is the voltage threshold in millivolts that when the battery is below this value will trigger the dead battery alarm
→	weak_battery_threshold	uint32 – The weak battery threshold is the voltage threshold in millivolts that when the battery is below this value will trigger the weak battery alarm

Description

Use this function to configure the pulse charger as follows:

- Use the charger or battery transistor (high side or low side) to pulse charge the battery.
- Set the TON, TOFF, and TDONE timers.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

The battery alarm (if applicable) will be enabled in the same function after the battery thresholds are set.

Side effects

Example

7.1.2 pm_boot_chg_get_weak_battery_status ()

This function checks if the battery is weak.

Parameters

```
pm_err_flag_type pm_boot_chg_get_weak_battery_status
(
   boolean *weak
)
```

→	*weak	Pointer of boolean: ▪ TRUE – The battery voltage is below the weak battery threshold ▪ FALSE – The battery voltage is above the weak battery threshold

Description

This function checks if the battery is weak.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.3 pm_boot_chg_get_battery_present ()

This function checks if the battery is present.

Parameters

```
pm_err_flag_type pm_boot_chg_get_battery_present
(
   boolean *present
)
```

→	*present	Pointer of boolean: ▪ TRUE – Battery is present ▪ FALSE – Battery is not present

Description

This function checks if the battery is present.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.4 pm_boot_chg_get_attached_charger ()

This function gets the attached charger type.

Parameters

```
pm_err_flag_type pm_boot_chg_get_attached_charger
(
   pm_boot_chg_attached_chgr_type *type
)
```

→	*type	Pointer of pm_boot_chg_attached_chgr_type: ▪ PM_BOOT_CHG_ATTACHED_CHGR__NONE ▪ PM_BOOT_CHG_ATTACHED_CHGR__USB ▪ PM_BOOT_CHG_ATTACHED_CHGR__DCIN ▪ PM_BOOT_CHG_ATTACHED_CHGR__BOTH

Description

This function gets the attached charger type.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.5 pm_boot_chg_set_iusb_max ()

This function sets the maximum USB charge current.

Parameters

```
pm_err_flag_type pm_boot_chg_set_iusb_max
(
   uint32 value_ma
)
```

→	value_ma	uint32 – The maximum value USB charge current in mA

Description

This function sets the maximum USB charge current. USB charge current is the current going through the USB cable.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.6 pm_boot_chg_set_ibat_max ()

This function sets the maximum fast charge battery current limit (IBAT).

Parameters

```
pm_err_flag_type pm_boot_chg_set_ibat_max
(
   uint32 value_ma
)
```

→	value_ma	uint32 –The maximum IBAT value in mA

Description

This function sets the maximum fast charge battery current limit.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.7 pm_boot_chg_set_itrkl ()

This function sets the system trickle charging current.

Parameters

```
pm_err_flag_type pm_boot_chg_set_itrkl
(
   uint32 value_ma
)
```

→	value_ma	uint32 –The system trickle charging current value in mA

Description

This function sets the system trickle charging current.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.8 pm_boot_chg_set_mode ()

This function sets the charge mode.

Parameters

```
pm_err_flag_type pm_boot_chg_set_mode
(
   pm_boot_chg_mode_type mode
)
```

→	mode	pm_boot_chg_mode_type: ▪ PM_BOOT_CHG_MODE__CHARGE_OFF– Mode used to set BOOT-CHG hardware state machine to "charge off" state ▪ PM_BOOT_CHG_MODE__CHARGE_ON – Mode used to set BOOT-CHG hardware state machine to "charge on" state ▪ PM_BOOT_CHG_MODE__ATC_OFF – Mode used to set BOOT-CHG hardware state machine to "ATC (Auto Trickle Charge) off" state ▪ PM_BOOT_CHG_MODE__ATC_ON – Mode used to set BOOT-CHG hardware state machine to "ATC (Auto Trickle Charge) on" state

Description

This function sets the charge mode, such as enabling charging and disable charging, etc.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.1.9 pm_boot_chg_enable_led ()

This function sets the charge LED and the LED source if applicable.

Parameters

```
pm_err_flag_type pm_boot_chg_enable_led
(
   boolean enable,
   pm_boot_chg_led_source_type source
)
```

→	enable	boolean: ▪ enable/disable the charger LED
→	source	pm_boot_chg_led_source_type: ▪ PM_BOOT_CHG_LED_SRC__GROUND ▪ PM_SBOOT_CHG_LED_SRC__VPH_PWR ▪ PM_BOOT_CHG_LED_SRC__BOOST ▪ PM_BOOT_CHG_LED_SRC__INTERNAL

Description

This function sets the charge LED and the LED source if applicable.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

Dependencies

Side effects

Example

7.2 Battery voltage driver API

The following interface functions are internal VBATT functions to support the VBATT interface using an integrated PMIC comparator with the Qualcomm PMIC.

7.2.1 pm_set_comparator_window()

This function sets the threshold, PM_BAT_ALRM_THRESH_R_W.

Parameters

```
void pm_set_comparator_window
(
    void
)
```

Description

Use this function to set the threshold PM_BAT_ALRM_THRESH_R_W.

Return value

None

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- vbatt_init()
- vbatt_task_init()

Side effects

None

Example

```
/* set PM_BAT_ALRM_THRESH_R_W on the basis of pm_calculate_next_window */
pm_set_comparator_window();
```

7.2.2 pm_vbatt_comp_window_change_detector()

This function:

- Checks for the direction of a voltage change
- Modifies the appropriate data structure to inform the application task from the ARM11® that the battery voltage has changed

Parameters

```
void pm_vbatt_comp_window_change_detector
(
    void
)
```

Description

Use this function to check for the direction of a voltage change, and to modify the appropriate data structure to inform the application task from the ARM11 that the battery voltage has changed.

Return value

None

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- vbatt_init()
- vbatt_task_init()

Side effects

None

Example

```
/* Check for direction of voltage change and inform that the battery-
voltage has changed. */
pm_vbatt_comp_window_change_detector();
pm_set_comparator_window();
```

7.2.3 pm_vbatt_voltage_change_registered()

This function resets the pm_vbatt_status_voltage.pm_did_batt_voltage_change flag.

Parameters

```
boolean pm_vbatt_voltage_change_registered
(
    void
)
```

Description

Once the voltage is changed, the vbatt_task is responsible for servicing the clients. However, first the pm_vbatt_status_voltage.pm_did_batt_voltage_change flag is set. After servicing is complete, use this function to reset this flag. The caller is vbatt_task because it services all clients and then calls this function.

Return value

This function returns:

- TRUE – Servicing pending
- FALSE – Servicing done

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- vbatt_init()
- vbatt_task_init()

Side effects

None

Example

```
if ( pm_vbatt_voltage_change_registered () )
{
   printf ( "VBATT task has serviced all cleints... " );
}
```

7.2.4 pm_comparator_vbatt_read()

This function returns the voltage incident at the battery terminals in mV.

Parameters

```
pm_battery_voltage_type pm_comparator_vbatt_read
(
    void
)
```

Description

Use this function to return the voltage incident at the battery terminals in mV.

Return value

This function returns pm_battery_voltage_type. Battery voltage is between 2800 and 4300 mV.

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- vbatt_init()
- vbatt_task_init()

Side effects

None

Example

```
pm_battery_voltage_type batt_voltage = pm_comparator_vbatt_read () ;
```

7.2.5 pm_disable_comparator_alarm()

This function disables the battery alarm and clears the interrupt.

Parameters

```
pm_err_flag_type pm_disable_comparator_alarm
(
    void
)
```

Description

Use this function to disable the battery alarm and clear the interrupt.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

```
if ( ! pm_disable_comparator_alarm() )
{
     printf ( "Battery Alarm Disabled " ) ;
}
```

7.2.6 pm_enable_comparator_alarm()

This function enables the battery alarm and clears the interrupt.

Parameters

```
pm_err_flag_type pm_enable_comparator_alarm
(
    void
)
```

Description

Use this function to enable the battery alarm and clear the interrupt.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

None

Example

```
if ( ! pm_enable_comparator_alarm () )
{
     printf ( "Battery Alarm Is Enabled " ) ;
}
```

7.3 Switch Mode Battery Charger API

The following interface functions are internal SMBC (Switch Mode Battery Charger) functions to support the Qualcomm PMIC SMBC module.

7.3.1 pm_smbc_set_mode()

This function allows/inhibits the SMBC module to send power-on-triggers to the Power-On (PON) module with regard to the charge path.

Parameters

```
pm_err_flag_type pm_smbc_set_mode
(
   Int                 externalResourceIndex,
   pm_smbc_mode_type   mode
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	mode	pm_smbc_mode_type: ▪ PM_SMBC_MODE__BOOT_FAIL ▪ PM_SMBC_MODE__BOOT_DONE ▪ PM_SMBC_MODE__BATT_TEMP_SENSING_OFF ▪ PM_SMBC_MODE__BATT_TEMP_SENSING_ON ▪ PM_SMBC_MODE__CHARGE_OFF ▪ PM_SMBC_MODE__CHARGE_ON ▪ PM_SMBC_MODE__CHARGE_PAUSE ▪ PM_SMBC_MODE__CHARGE_RESUME ▪ PM_SMBC_MODE__FORCE_ATC_OFF ▪ PM_SMBC_MODE__FORCE_ATC_ON ▪ PM_SMBC_MODE__ATC_FAILED_CLEAR_ON ▪ PM_SMBC_MODE__CHG_FAILED_CLEAR_ON ▪ PM_SMBC_MODE__USB_SUSPENDED_OFF ▪ PM_SMBC_MODE__USB_SUSPENDED_ON

Description

Use this function to set the threshold PM_BAT_ALRM_THRESH_R_W.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – Current processor does not have the access to the SMBC module

- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.2 pm_smbc_set_pwron_trig()

This function allows/inhibits the SMBC module to send power-on-trigger to the PON module wrt the charge path.

Parameters

```
pm_err_flag_type pm_smbc_set_pwron_trig
(
   Int                      externalResourceIndex,
   pm_smbc_cmd_type         enable,
   pm_smbc_chg_path_type    path
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	enable	pm_smbc_cmd_type: ▪ PM_SMBC_CMD__DISABLE ▪ PM_SMBC_CMD__ENABLE
→	path	pm_smbc_chg_path_type: ▪ PM_SMBC_CHGR_PATH__USB ▪ PM_SMBC_CHGR_PATH__DCIN

Description

This function allows/inhibits the SMBC module to send power-on-triggers to the PON module when a charge device is plugged into the phone.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correc.
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Input Parameter three is out of range

Dependencies

Side effects

Example

7.3.3 pm_smbc_get_batfet_present()

This function gets the availability of the BATFET.

Parameters

```
pm_err_flag_type pm_smbc_get_batfet_present
(
   Int       externalResourceIndex,
   Boolean   *present
)
```

→	externalResourceIndex	Int –The external SMBC ID
→	*present	Pointer of boolean: ▪ present – The status of BATFET

Description

This function checks if the BATFET is present.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.4 pm_smbc_set_default_vdd()

This function sets the default VDD value.

Parameters

```
pm_err_flag_type pm_smbc_set_default_vdd
(
   Int                      externalResourceIndex,
   pm_smbc_vdd_default_type vdd
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	vdd	pm_smbc_vdd_default_type: ▪ PM_SMBC_VDD_DEFAULT__LOW ▪ PM_SMBC_VDD_DEFAULT__STANDARD

Description

This function sets the default VDD (power supply) value.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.5 pm_smbc_set_batt_temp_config()

This function sets the battery temperature configurations.

Parameters

```
pm_err_flag_type pm_smbc_set_batt_temp_config
(
   Int                                 externalResourceIndex,
   pm_smbc_batt_temp_ref_gnd_type      ref_gnd,
   pm_smbc_batt_temp_cold_thr_type     cold_thr,
   pm_smbc_batt_temp_hot_thr_type      hot_thr
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	ref_gnd	pm_smbc_batt_temp_ref_gnd_type: ▪ PM_SMBC_BATT_TEMP_REF_GND__PACK ▪ PM_SMBC_BATT_TEMP_REF_GND__GND
→	cold_thr	pm_smbc_batt_temp_cold_thr_type: ▪ PM_SMBC_BATT_TEMP_COLD_THR__LOW ▪ PM_SMBC_BATT_TEMP_COLD_THR__HIGH
→	hot_thr	pm_smbc_batt_temp_hot_thr_type: ▪ PM_SMBC_BATT_TEMP_HOT_THR__LOW ▪ PM_SMBC_BATT_TEMP_HOT_THR__HIGH

Description

This function sets the battery temp configurations, such as the ground connection for the battery thermistor, the cold battery temp threshold, the hot battery temp threshold, etc.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Input Parameter three is out of range
- PM_ERR_FLAG__PAR4_OUT_OF_RANGE – Input Parameter four is out of range

Dependencies

Side effects

Example

7.3.6 pm_smbc_get_batt_therm_gnd()

This function gets the ground connection for the battery thermistor.

Parameters

```
pm_err_flag_type pm_smbc_get_batt_therm_gnd
(
   int externalResourceIndex,
   pm_smbc_batt_temp_ref_gnd_type*  ref_gnd
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	ref_gnd	pm_smbc_batt_temp_ref_gnd_type: ▪ PM_SMBC_BATT_TEMP_REF_GND__PACK ▪ PM_SMBC_BATT_TEMP_REF_GND__GND

Description

This function gets the ground connection for the battery thermistor.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.7 pm_smbc_set_led_source ()

This function selects power source for the anode of the charge indicator LED.

Parameters

```
pm_err_flag_type pm_smbc_set_led_source
(
   int externalResourceIndex,
   pm_smbc_led_source_type source
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	source	pm_smbc_led_source_type: ▪ PM_SMBC_LED_SRC__GROUND ▪ PM_SMBC_LED_SRC__VPH_PWR ▪ PM_SMBC_LED_SRC__BOOST ▪ PM_SMBC_LED_SRC__INTERNAL

Description

This function selects power source for anode of the charge indicator LED.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.8 pm_smbc_get_battery_weak ()

This function checks if the battery is weak.

Parameters

```
pm_err_flag_type pm_smbc_get_battery_weak
(
   int externalResourceIndex,
   boolean *weak
)
```

→	externalResourceIndex	int – The external SMBC ID
→	*weak	Pointer of boolean: ▪ TRUE – The battery is weak (below the threshold) ▪ FALSE – The battery is normal (above the threshold)

Description

This function checks if the battery is weak by checking the battery voltage against the weak battery threshold; this threshold is set by pm_smbc_set_weak_vbatt_threshold().

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.9 pm_smbc_get_attached_charger ()

This function gets the attached charger type.

Parameters

```
pm_err_flag_type pm_smbc_get_attached_charger
(
   int externalResourceIndex,
   pm_smbc_attached_chgr_type *type
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	*type	pm_smbc_attached_chgr_type: ▪ PM_SMBC_ATTACHED_CHGR__NONE: no charger attached ▪ PM_SMBC_ATTACHED_CHGR__USB: USB device attached ▪ PM_SMBC_ATTACHED_CHGR__DCIN: DC wall charger attached ▪ PM_SMBC_ATTACHED_CHGR__BOTH

Description

This function gets the attached charger type.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.10 pm_smbc_get_battery_present ()

This function checks if the battery is present.

Parameters

```
pm_err_flag_type pm_smbc_get_battery_present
(
   int externalResourceIndex,
   boolean *present
)
```

→	externalResourceIndex	int – The external SMBC ID
→	*present	Pointer of boolean: ▪ TRUE – Battery is present ▪ FALSE – Battery is not present

Description

This function checks if the battery is present.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.11 pm_smbc_set_iusb_max ()

This function sets the maximum USB_IN current.

Parameters

```
pm_err_flag_type pm_smbc_set_iusb_max
(
   int externalResourceIndex,
   uint32 value_ma
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	value_ma	uint32 – The current value in mA

Description

This function sets the maximum USB_IN current. USB_IN current is the current going through the USB cable.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.12 pm_smbc_set_vdd_max ()

This function sets the maximum VDD.

Parameters

```
pm_err_flag_type pm_smbc_set_vdd_max
(
   int externalResourceIndex,
   uint32 value_mv
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	value_mv	uint32 – The VDD value in mV

Description

This function sets the maximum VDD. VDD is the power supply to the phone.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.13 pm_smbc_set_vbatdet ()

This function sets VBATDET.

Parameters

```
pm_err_flag_type pm_smbc_set_vdd_max
(
   int externalResourceIndex,
   uint32 value_mv
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	value_mv	uint32 – The VBATDETvalue in mV

Description

This function sets VBATDET (BATtery Voltage DETect). VBATDET comparator is used to restart charging when battery voltage is less than the VBATDET value. When the battery voltage is greater than the VBATDET threshold, charging stops.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.14 pm_smbc_set_ibat_max ()

This function sets the maximum fast charge battery current limit.

Parameters

```
pm_err_flag_type pm_smbc_set_ibat_max
(
   int externalResourceIndex,
   uint32 value_ma
)
```

→	externalResourceIndex	Int – The external SMBC ID
→	value_ma	uint32 – The maximum IBAT value in mA

Description

This function sets the maximum fast charge battery current limit.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.15 pm_smbc_set_vin_min ()

This function sets the minimum charger input voltage.

Parameters

```
pm_err_flag_type pm_smbc_set_vin_min
(
   int externalResourceIndex,
   uint32 value_mv
)
```

→	externalResourceIndex	int – – The external SMBC ID
→	value_mv	uint32 – The minimum input voltage regulation value in mV

Description

This function sets the minimum charger input voltage.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.16 pm_smbc_get_batt_sense_r ()

This function gets the battery current sense resistor value.

Parameters

```
pm_err_flag_type pm_smbc_get_batt_sense_r
(
   int externalResourceIndex,
   uint32* value_mOhm
)
```

→	externalResourceIndex	int – The external SMBC ID
→	*value_mOhm	Pointer of uint32 – value_mOhm, the battery current sense resistor value in mOHM

Description

This function gets the battery current sense resistor value.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER– Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.17 pm_smbc_set_weak_vbatt_threshold ()

This function sets the system weak battery threshold.

Parameters

```
pm_err_flag_type pm_smbc_set_weak_vbatt_threshold
(
   int externalResourceIndex,
   uint32 value_mv
)
```

→	externalResourceIndex	int – The external SMBC ID
→	value_mv	uint32 – The system weak battery threshold value in mV

Description

This function sets the system weak battery threshold. This is the battery threshold where high current trickle charging starts.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.18 pm_smbc_set_low_vbatt_threshold ()

This function sets the low battery threshold.

Parameters

```
pm_err_flag_type pm_smbc_set_low_vbatt_threshold
(
   int externalResourceIndex,
   uint32 value_mv
)
```

NOTE: If the value of the parameter value_mv is out of the valid range that the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds mV value parameter input.

→	`externalResourceIndex`	Int – The external SMBC ID
→	`value_mv`	uint32 –The system low battery threshold value in mV

Description

This function sets the system low battery threshold. This is the battery threshold at which low current automatic trickle charging (ATC) starts.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.19 pm_smbc_set_itrkl ()

This function sets the system trickle charging current.

Parameters

```
pm_err_flag_type pm_smbc_set_itrkl
(
   int externalResourceIndex,
   uint32 value_ma
)
```

NOTE: If the value of the parameter value_ma is out of the valid range that the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds mA value parameter input.

→	externalResourceIndex	Int – The external SMBC ID
→	value_ma	uint32 – The system trickle charging current value in mA

Description

This function sets the system trickle charging current.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.20 pm_smbc_set_iterm ()

This function sets the system end-of-charge current.

Parameters

```
pm_err_flag_type pm_smbc_set_iterm
(
   int externalResourceIndex,
   uint32 value_ma
)
```

NOTE: If the value of the parameter value_ma is out of the valid range that the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds mA value parameter input.

→	externalResourceIndex	Int – The external SMBC ID
→	value_ma	uint32 – The system end-of-charge current value in mA.

Description

This function sets system end-of-charge (EoC) current. This is the current used to generate the EoC interrupt and terminate charging.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.21 pm_smbc_set_ttrkl ()

This function sets the maximum time for trickle charging.

Parameters

```
pm_err_flag_type pm_smbc_set_tchg
(
   int externalResourceIndex,
   uint32 value_minutes
)
```

NOTE: If the value of the parameter value_minutes is out of the valid range that the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that was an out-of-bounds minute value parameter input.

→	externalResourceIndex	int – The external SMBC ID
→	value_minutes	uint32 – The maximum time value in minutes for SW initiated charging

Description

This function sets the maximum time for software-initiated charging, trickle, and fast phases.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.22 pm_smbc_set_tchg ()

This function sets the maximum time for software-initiated charging.

Parameters

```
pm_err_flag_type pm_smbc_set_ttrkl
(
   int externalResourceIndex,
   uint32 value_minutes
)
```

NOTE: If the value of the parameter value_minutes is out of the valid range the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds minute value parameter input.

→	externalResourceIndex	Int – The external SMBC ID
→	value_minutes	uint32 – The maximum time value in minutes for trickle charging

Description

This function sets the maximum time for initiated trickle charging, including software-initiated trickle charging as well as hardware-controlled auto-trickle charging (ATC).

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.23 pm_smbc_set_twdog ()

This function sets the delay and the timeout for the hardware WDOG timer in the PMIC SMBC module.

Parameters

```
pm_err_flag_type pm_smbc_set_twdog
(
   int externalResourceIndex,
   uint32 delay_seconds,
   uint32 timeout_seconds
)
```

NOTE: If the value of the parameter delay_seconds is out of the valid range the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds second value parameter input.

NOTE: If the value of the parameter timeout_seconds is out of the valid range that the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR3_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds third value parameter input.

→	`externalResourceIndex`	Int – The external SMBC ID
→	`delay_seconds`	uint32 – The delay in seconds between WDOG IRQ and stop charging
→	`timeout_seconds`	uint32 – The WDOG timer timeout value in seconds

Description

This function sets the delay between the hardware WDOG IRQ and stop charging and the timeout for the hardware WDOG timer in the PMIC SMBC module. The software-initiated charging will stop when this hardware timer expires. Use this API to set the timeout in order to restart the hardware WDOG timer (pet the dog). This timer can be used along with a software periodic timer, e.g., heartbeat, to ensure that charging will stop in time when the software crashes. The timeout of the software periodic timer should be set a little shorter than the timeout of the hardware timer in order to be able to restart the hardware WDOG timer before it times out.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct

- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE– Input Parameter three is out of range

Dependencies

Side effects

Example

7.3.24 pm_smbc_set_temp_threshold ()

This function sets charging temperature thresholds.

Parameters

```
pm_err_flag_type pm_smbc_set_temp_threshold
(
   int externalResourceIndex,
   uint32 stop_deg_c,
   uint32 resume_deg_c
)
```

NOTE: If the value of the parameter stop_deg_c is out of the valid range the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR2_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds second value parameter input.

NOTE: If the value of the parameter resume_deg_c is out of the valid range that the PMIC hardware can take, the value will be limited to the boundary and the PMIC hardware will be set to that value; at the same time, PM_ERR_FLAG__PAR3_OUT_OF_RANGE will be returned to indicate that there was an out-of-bounds third value parameter input.

→	`externalResourceIndex`	Int – The external SMBC ID
→	`stop_deg_c`	uint32 – stop-charging temperature value in Celsius
→	`resume_deg_c`	uint32 – Resume-charging temperature value in Celsius

Description

This function sets charging temperature thresholds. Charging will stop (pass device turns off) when the charging temperature is higher than the maximum charging temperature threshold; charging will resume (pass device turns on) when the charging temperature has fallen below the resume-charging temperature threshold.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct

- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE– Input Parameter three is out of range

Dependencies

Side effects

Example

7.3.25 pm_smbc_set_charge_path ()

This function sets the charging path.

Parameters

```
pm_err_flag_type pm_smbc_set_charge_path
(
   int externalResourceIndex,
   pm_smbc_chg_path_type path
)
```

NOTE: The default setting in PMIC hardware is device specific.

→	externalResourceIndex	Int – The external SMBC ID
→	path	pm_smbc_chg_path_type: ▪ PM_SMBC_CHGR_PATH__USB – USB device ▪ PM_SMBC_CHGR_PATH__DCIN – DC wall charger

Description

This function sets the charging path. The charging path will be either DC wall or USB, but not both at the same time.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.26 pm_smbc_get_charge_path ()

This function gets the charging path.

Parameters

```
pm_err_flag_type pm_smbc_set_charge_path
(
   int externalResourceIndex,
   pm_smbc_chg_path_type *path
)
```

NOTE: The default setting in PMIC hardware is device specific.

→	externalResourceIndex	Int – The external SMBC ID
→	*path	Pointer of pm_smbc_chg_path_type: ▪ PM_SMBC_CHGR_PATH__USB – USB device ▪ PM_SMBC_CHGR_PATH__DCIN – DC wall charger

Description

This function gets the charging path. This is the charging path that is set by pm_smbc_set_charge_path().

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER– Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.3.27 pm_smbc_set_charge_state ()

This function sets the FSM charge state.

Parameters

```
pm_err_flag_type pm_smbc_set_charge_state
(
   int externalResourceIndex,
   pm_smbc_fsm_state_name_type state
)
```

NOTE: The default setting in PMIC hardware is device-specific.

→	externalResourceIndex	Int – The external SMBC ID
→	state	pm_smbc_fsm_state_name_type: ▪ PM_SMBC_FSM_ST__OFF ▪ PM_SMBC_FSM_ST__PWR_ON_CHGR ▪ PM_SMBC_FSM_ST__ATC_LOW_I ▪ PM_SMBC_FSM_ST__PWR_ON_BATT ▪ PM_SMBC_FSM_ST__ATC_FAIL ▪ PM_SMBC_FSM_ST__PWR_ON_CHGR_AND_BATT ▪ PM_SMBC_FSM_ST__FAST_CHG ▪ PM_SMBC_FSM_ST__TRICKLE_CHG ▪ PM_SMBC_FSM_ST__CHG_FAIL ▪ PM_SMBC_FSM_ST__EOC ▪ PM_SMBC_FSM_ST__ATC_PAUSE ▪ PM_SMBC_FSM_ST__FAST_CHG_PAUSE ▪ PM_SMBC_FSM_ST__TRICKLE_CHG_PAUSE ▪ PM_SMBC_FSM_ST__ATC_HIGH_I ▪ PM_SMBC_FSM_ST__FLCB

Description

This function sets the FSM charge state.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.3.28 pm_smbc_get_charge_state ()

This function gets the FSM charge state.

Parameters

```
pm_err_flag_type pm_smbc_set_charge_state
(
   int externalResourceIndex,
   pm_smbc_fsm_state_name_type *state
)
```

NOTE: The default setting in PMIC hardware is device-specific.

→	externalResourceIndex	Int – The external SMBC ID
→	*state	Pointer of pm_smbc_fsm_state_name_type: ▪ PM_SMBC_FSM_ST__OFF ▪ PM_SMBC_FSM_ST__PWR_ON_CHGR ▪ PM_SMBC_FSM_ST__ATC_LOW_I ▪ PM_SMBC_FSM_ST__PWR_ON_BATT ▪ PM_SMBC_FSM_ST__ATC_FAIL ▪ PM_SMBC_FSM_ST__PWR_ON_CHGR_AND_BATT ▪ PM_SMBC_FSM_ST__FAST_CHG ▪ PM_SMBC_FSM_ST__TRICKLE_CHG ▪ PM_SMBC_FSM_ST__CHG_FAIL ▪ PM_SMBC_FSM_ST__EOC ▪ PM_SMBC_FSM_ST__ATC_PAUSE ▪ PM_SMBC_FSM_ST__FAST_CHG_PAUSE ▪ PM_SMBC_FSM_ST__TRICKLE_CHG_PAUSE ▪ PM_SMBC_FSM_ST__ATC_HIGH_I ▪ PM_SMBC_FSM_ST__FLCB

Description

This function gets the FSM charge state.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the SMBC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the SMBC module is not correct

- PM_ERR_FLAG__INVALID_SMBC_INDEXED – The internal resource ID for the SMBC module is not correct
- PM_ERR_FLAG__INVALID_POINTER– Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

7.4 Battery Temperature Management API

The Battery Temperature Management API provides a software interface for programming the Qualcomm PMIC BTM (Battery Temperature Management) module. This API enables the following PMIC hardware configurations.

7.4.1 pm_btm_set_mode ()

This function enables/disables the BTM module.

Parameters

```
pm_err_flag_type pm_btm_set_mode
(
    int externalResourceIndex,
    pm_btm_mode_type mode
)
```

→	externalResourceIndex	Int – The external BTM ID
→	mode	pm_btm_mode_type: ▪ PM_BTM_CMD__ENABLE ▪ PM_BTM_CMD__DISABLE

Description

This function enables/disables the BTM timing interval battery temperature measurements when the ADC arbiter module is enabled.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.2 pm_btm_set_conversion_request ()

This function sets the conversion request.

Parameters

```
pm_err_flag_type pm_btm_set_conversion_request
(
   int externalResourceIndex,
   pm_btm_conv_req_cmd_type cmd
)
```

→	externalResourceIndex	int – The external BTM ID
→	cmd	pm_btm_conv_req_cmd_type: ▪ PM_BTM_CMD__ENABLE ▪ PM_BTM_CMD__DISABLE

Description

This function sets the conversion request. After the request is sent, the ADC will start the BTM; the request is cleared when single measurement mode is selected and after the ADC conversion is completed.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE –The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct

Dependencies

Side effects

Example

7.4.3 pm_btm_get_conversion_status ()

This function gets the conversion status.

Parameters

```
pm_err_flag_type pm_btm_get_conversion_status
(
   int externalResourceIndex,
   pm_btm_xoadc_conversion_status_type *status
)
```

→	externalResourceIndex	int – The external BTM ID
→	*status	Pointer of pm_btm_xoadc_conversion_status_type: ▪ PM_BTM_CONVERSION_STATUS__INVALID ▪ PM_BTM_CONVERSION_STATUS__COMPLETE ▪ PM_BTM_CONVERSION_STATUS__OCCURRING ▪ PM_BTM_CONVERSION_STATUS__WAITING

Description

This function gets the conversion status based on the status of the conversion status strobe and the end-of-conversion status flag.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Invalid pointer passed in

Dependencies

Side effects

Example

7.4.4 pm_btm_set_measurement_mode ()

This function sets the measurement mode.

Parameters

```
pm_err_flag_type pm_btm_set_measurement_mode
(
   int externalResourceIndex,
   pm_btm_meas_mode_type mode
)
```

→	externalResourceIndex	int – The external BTM ID
→	mode	pm_btm_meas_mode_type: ▪ PM_XOADC_BTM_MEAS_MODE_SINGLE ▪ PM_XOADC_BTM_MEAS_MODE_CONTINUOUS

Description

This function sets the BTM mode.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.5 pm_btm_set_measurement_interval ()

This function sets the measurement interval.

Parameters

```
pm_err_flag_type pm_btm_set_measurement_interval
(
   int externalResourceIndex,
   uint32 second
)
```

→	externalResourceIndex	int – The external BTM ID
→	second	uint32 – Second, the interval time

Description

This function sets the BTM interval time.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.6 pm_btm_get_prescalar ()

This function gets the prescaler value for the specific channel.

Parameters

```
pm_err_flag_type pm_btm_get_prescalar
(
   int externalResourceIndex,
   uint8 channel,
   uint8 *numerator,
   uint8 *denominator
)
```

→	externalResourceIndex	int – The external BTM ID
→	channel	uint8 – The analog channel
→	*numerator	Pointer of uint8 – The numerator of the prescalar
→	*denominator	Pointer of uint8 – The denominator of the prescalar

Description

This function gets the prescaler value for the specific channel.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range
- PM_ERR_FLAG__INVALID_POINTER – Invalid pointer

Dependencies

Side effects

Example

7.4.7 pm_btm_set_xoadc_input ()

This function sets the XOADX input.

Parameters

```
pm_err_flag_type pm_btm_set_xoadc_input
(
   int externalResourceIndex,
   pm_btm_xoadc_input_select_type input
)
```

→	externalResourceIndex	int – The external BTM ID
→	input	pm_btm_xoadc_input_select_type: ▪ PM_BTM_XOADC_INPUT_PMIC – PMIC_IN ▪ PM_BTM_XOADC_INPUT_XO –- XO_IN ▪ PM_BTM_XOADC_INPUT_VREFP – vrefp

Description

This function sets the XOADX input that is used for BTM.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.8 pm_btm_set_xoadc_decimation_ratio ()

This function sets the XOADX decimation ratio.

Parameters

```
pm_err_flag_type pm_btm_set_xoadc_decimation_ratio
(
   int externalResourceIndex,
   pm_btm_xoadc_decimation_ratio_type ratio
)
```

→	externalResourceIndex	int –The external BTM ID
→	ratio	pm_btm_xoadc_decimation_ratio_type: ▪ PM_BTM_XOADC_DECIMATION_RATIO_512 ▪ PM_BTM_XOADC_DECIMATION_RATIO_1K ▪ PM_BTM_XOADC_DECIMATION_RATIO_2K ▪ PM_BTM_XOADC_DECIMATION_RATIO_4K

Description

This function sets the XOADX decimation ratio that is used for BTM.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.9 pm_btm_set_xoadc_conversion_rate ()

This function sets the XOADX clock rate.

Parameters

```
pm_err_flag_type pm_btm_set_xoadc_conversion_rate
(
   int externalResourceIndex,
   pm_btm_xoadc_conversion_rate_type rate
)
```

→	externalResourceIndex	int – The external BTM ID
→	rate	pm_btm_xoadc_decimation_ratio_type: ▪ PM_BTM_XOADC_CONVERSION_RATE_TCXO_DIV_8 ▪ PM_BTM_XOADC_CONVERSION_RATE_TCXO_DIV_4 ▪ PM_BTM_XOADC_CONVERSION_RATE_TCXO_DIV_2 ▪ PM_BTM_XOADC_CONVERSION_RATE_TCXO

Description

This function sets the XOADX clock rate that is used for BTM.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.10 pm_btm_get_xoadc_data ()

This function gets the XOADC conversion result.

Parameters

```
pm_err_flag_type pm_btm_get_xoadc_data
(
   int externalResourceIndex,
   uint32* data
)
```

→	externalResourceIndex	int –The external BTM ID
→	*data	Pointer of uint32 – This parameter is used to store the XOADC conversion result

Description

This function sets the XOADX conversion result.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Invalid pointer

Dependencies

Side effects

Example

7.4.11 pm_btm_set_batt_warm_threshold ()

This function sets the battery warm-temp threshold.

Parameters

```
pm_err_flag_type pm_btm_set_batt_warm_threshold
(
   int externalResourceIndex,
   uint32 warm_temp_thresh
)
```

→	externalResourceIndex	int –The external BTM ID
→	warm_temp_thresh	uint32 – This parameter is used to set the warm battery temp threshold

Description

This function sets the threshold for the battery to be considered warm.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE– Input Parameter two is out of range

Dependencies

Side effects

Example

7.4.12 pm_btm_set_batt_cool_threshold ()

This function sets the battery cool-temp threshold.

Parameters

```
pm_err_flag_type pm_btm_set_batt_cool_threshold
(
   int externalResourceIndex,
   uint32 cool_temp_thresh
)
```

→	externalResourceIndex	int – The external BTM ID
→	cool_temp_thresh	uint32 – This parameter is used to set the cool battery temp threshold

Description

This function sets the threshold for the battery to be considered cool.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the BTM module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the BTM module is not correct
- PM_ERR_FLAG__INVALID_BTM_INDEXED – The internal resource ID for the BTM module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.5 Over Voltage Protection API

The OVP module is used to protect the VCHG/VBUS pin from high voltages; it allows the connection of nonstandard charging devices (such as car kits) to the VBUS/VCHG pin for charging purposes. This module, along with the SMBC module, will majorly be used by the charger application for charging Li-Ion batteries with high current.

7.5.1 pm_ovp_set_mode ()

This function sets the OVP module.

Parameters

```
pm_err_flag_type pm_ovp_set_mode
(
   int externalResourceIndex,
   pm_ovp_mode_type mode
)
```

→	externalResourceIndex	int –The external OVP ID
→	mode	pm_ovp_mode_type: ▪ PM_OVP_MODE__MODULE_ON ▪ PM_OVP_MODE__MODULE_OFF

Description

This function sets certain modes on the OVP module, such as enabling the module, disabling the module, etc.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the OVP module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the OVP module is not correct
- PM_ERR_FLAG__INVALID_OVP_INDEXED – The internal resource ID for the OVP module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.5.2 pm_ovp_set_threshold ()

This function sets the overvoltage threshold that triggers the OVP.

Parameters

```
pm_err_flag_type pm_ovp_set_threshold
(
   int externalResourceIndex,
   uint32 value_mv
)
```

→	externalResourceIndex	int –The external OVP ID
→	value_mv	uint32 – The overvoltage threshold in mV

Description

This function sets the overvoltage threshold that triggers the OVP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the OVP module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the OVP module is not correct
- PM_ERR_FLAG__INVALID_OVP_INDEXED – The internal resource ID for the OVP module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.5.3 pm_ovp_set_vreg ()

This function sets the internal LDO regulator voltage.

Parameters

```
pm_err_flag_type pm_ovp_set_vreg
(
   int externalResourceIndex,
   uint32 value_mv
)
```

→	externalResourceIndex	int – The external OVP ID
→	value_mv	uint32 – The VREG value in mV

Description

This function sets the internal LDO regulator voltage (VREG).

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the OVP module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the OVP module is not correct
- PM_ERR_FLAG__INVALID_OVP_INDEXED – The internal resource ID for the OVP module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.5.4 pm_ovp_set_hysteresis ()

This function sets the time hysteresis for the voltage to exceed the threshold before it triggers the OVP.

Parameters

```
pm_err_flag_type pm_ovp_set_hysteresis
(
   int externalResourceIndex,
   uint32 value_us
)
```

→	externalResourceIndex	int –The external OVP ID
→	value_us	uint32 – Time hysteresis in microseconds for the voltage to exceed the threshold before it triggers the OVP

Description

This function sets the time hysteresis for the voltage to exceed the threshold before it triggers the OVP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – SUCCESS
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the OVP module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the OVP module is not correct
- PM_ERR_FLAG__INVALID_OVP_INDEXED – The internal resource ID for the OVP module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

7.5.5 pm_ovp_get_chg_in_valid ()

This function gets the charger-plugged-in status.

Parameters

```
pm_err_flag_type pm_ovp_get_chg_in_valid
(
   int externalResourceIndex,
   boolean *valid
)
```

→	externalResourceIndex	int – The external OVP ID
→	*valid	Pointer of boolean – A pointer to the charger_valid status

Description

This function gets the charger-plugged-in status for either the DC or USB charging device based on the input parameter externalResourceIndex.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the OVP module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the OVP module is not correct
- PM_ERR_FLAG__INVALID_OVP_INDEXED – The internal resource ID for the OVP module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Input Parameter two is a NULL pointer

Dependencies

Side effects

Example

8 Sudden Momentary Power Loss API

The PM8921 Sudden Momentary Power Loss (SMPL) feature initiates a power-on sequence without MSM intervention if the monitored phone voltage (VDD) drops out of range and then returns in-range within a programmable interval. When enabled by the software, SMPL achieves immediate and automatic PM8921 recovery from momentary power loss (such as a brief battery disconnect when the phone is jarred).

SMPL circuits run off an internal voltage net (dVDD) that is formed by diode O-ring voltages at the VBAT, ISNS_M (VDD), and VCOIN pins. Typically, this voltage also runs the crystal oscillator and real-time clock.

8.1 pm_set_smpld_timer()

This function configures the SMPL feature timeout value.

Parameters

```
pm_err_flag_type pm_set_smpld_timer
(
  pm_smpl_timer_type time
)
```

→	time	pm_smpl_timer_type: ▪ PM_SMPL_TIMER__0500msec ▪ PM_SMPL_TIMER__1000msec ▪ PM_SMPL_TIMER__1500msec ▪ PM_SMPL_TIMER__2500msec

Description

Use this function to configure the SMPL feature timeout value. If the battery or external power is reapplied before expiration of the timer, the PMIC initiates a power-on sequence.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the time parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while in this function.

Example

```
pm_set_smpld_timer(PM_SMPL_TIMER__0500msec);
```

8.2 pm_smpld_switch()

This function sets the SMPL detection on or off.

Parameters

```
pm_err_flag_type pm_smpld_switch
(
  pm_switch_cmd_type OnOff
)
```

→	OnOff	pm_switch_cmd_type: ▪ PM_ON_CMD ▪ PM_OFF_CMD

Description

Use this function to set the SMPL on or off. The SMPL must be set to off during normal power down.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

Before this function can be used:

- The pm_init() function must be invoked.
- The SMPL timeout value, pm_set_smpld_timer() (Section 8.1), must be configured.

Side effects

Interrupts are disabled while in this function.

Example

```
pm_smpld_switch(PM_ON_CMD);
```

9 Output Voltage Regulation

9.1 Name mapping

Table 9-1 provides the voltage regulator name mapping.

Table 9-1 Voltage regulator name mapping (software and hardware)

Hardware VREG	Software ID	Type	Default V_{out} (V)	Default state
S1	PM_SMPS_1	1.5 A Buck SMPS		On
S2	PM_SMPS_2	1.5 A Buck SMPS	1.300	Off
S3	PM_SMPS_3	1.5 A Buck SMPS	1.050	On
S4	PM_SMPS_4	1.5 A Buck SMPS	1.800	On
S5	PM_SMPS_5	2.0A Buck SMPS	1.050	Off
S6	PM_SMPS_6	2.0A Buck SMPS	1.050	Off
S7	PM_SMPS_7	1.5 A Buck SMPS	1.150	Off
S8	PM_SMPS_8	1.5 A Buck SMPS	2.200	Off
L1	PM_LDO_1	Linear n LDO	1.050	On
L2	PM_LDO_2	Linear n LDO	1.200	Off
L3	PM_LDO_3	Linear p LDO	3.075	On
L4	PM_LDO_4	Linear p LDO	1.800	On
L5	PM_LDO_5	Linear p LDO	2.950	On
L6	PM_LDO_6	Linear p LDO	2.950	On
L7	PM_LDO_7	Linear p LDO	2.950	On
L8	PM_LDO_8	Linear p LDO	2.800	Off
L9	PM_LDO_9	Linear p LDO	2.850	Off
L10	PM_LDO_10	Linear p LDO	2.900	Off
L11	PM_LDO_11	Linear p LDO	2.900	Off
L12	PM_LDO_12	Linear n LDO	1.200	Off
L14	PM_LDO_14	Linear p LDO	1.800	On
L15	PM_LDO_15	Linear p LDO	2.950	On
L16	PM_LDO_16	Linear p LDO	2.800	Off
L17	PM_LDO_17	Linear p LDO	2.950	Off
L18	PM_LDO_18	Linear n LDO	1.300	On
L21	PM_LDO_21	Linear p LDO	1.900	Off
L22	PM_LDO_22	Linear p LDO	2.600	Off
L23	PM_LDO_23	Linear p LDO	1.800	Off
L24	PM_LDO_24	Linear n LDO	1.050	On

Hardware VREG	Software ID	Type	Default V_{out} (V)	Default state
L25	PM_LDO_25	Linear n LDO	1.225	On
L26	PM_LDO_26	Linear n LDO	1.050	Off
L27	PM_LDO_27	Linear n LDO	1.050	Off
L28	PM_LDO_28	Linear n LDO	1.050	Off
LVS1	PM_VS_LVS_1	Low Voltage Switch	1.800	Off
LVS2	PM_VS_LVS_2	Low Voltage Switch	1.200	Off
LVS3	PM_VS_LVS_3	Low Voltage Switch	1.800	Off
LVS4	PM_VS_LVS_4	Low Voltage Switch	1.800	Off
LVS5	PM_VS_LVS_5	Low Voltage Switch	1.800	Off
LVS6	PM_VS_LVS_6	Low Voltage Switch	1.800	Off
LVS7	PM_VS_LVS_7	Low Voltage Switch	1.800	Off
5VS_OTG	PM_VS_OTG_1	5V switch	5.000	Off
5VS_HDMI	PM_VS_HDMI_1	5V switch	5.000	Off

NOTE: Unused Lxx designators are L13, L19, and L20.

9.2 Voltage regulator API

The PM8921 includes programmable VREGs to be used to power most of the hardware components within the handset. The two major types are on-chip, Switched-Mode Power Supplies (SMPS) and linear Low Dropout (LDO) Regulators.

NOTE: The number, type, and default values of the voltage regulators can vary. See the corresponding User Guide and Device Specification document for more details.

There are two types of SMPS voltage regulators:

- The Boost Voltage Regulators (BOOST VREG), which provide output DC voltages larger than their input DC voltage and are also known as step-up converters
- The Buck Voltage Regulators (BUCK VREGs), which provide output voltage smaller than their input voltage and are known as step-down converters

Linear Low Dropout Voltage Regulators (LDO VREGs) regulate the output voltage to match the scaled version of the reference voltage regardless (nearly) of the current delivered to the load.

9.2.1 pm_lp_mode_control()

This function enables/disables the LDO's Low Power mode.

Parameters

```
pm_err_flag_type pm_lp_mode_control
(
    pm_switch_cmd_type vreg_cmd,
    pm_vreg_id_type    which_vreg
)
```

→	vreg_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – Disable the VREG Low Power mode ▪ PM_ON_CMD – Enable the VREG Low Power mode
→	which_vreg	pm_vreg_id_type: ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28

Description

Use this function to configure the selected LDOs to Low Power mode.

Low Power mode is used to reduce the quiescent current. Placing a voltage regulator into Low Power mode maximizes standby time by decreasing the amount of current supplied by the voltage regulator while the handset is in Sleep mode.

Low Power mode does not become active until the TCXO clock is shut down.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg_cmd parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the which_vreg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable low power mode for VREG L16. Note: Low power mode will not become
active until TCXO is disabled. */
err = pm_lp_mode_control( PM_ON_CMD, PM_LDO_16)

/* "Find out if there was a PMIC API error.*/
if (err != PM_ERR_FLAG__SUCCESS)
{
     MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

9.2.2 pm_vreg_control()

This function enables/disables voltage regulators.

NOTE: Instead of this API< pm_vote_vreg_switch (), described inSection 9.2.26, is recommended.

Parameters

```
pm_err_flag_type pm_vreg_control
(
    pm_switch_cmd_type  vreg_cmd,
    pm_vreg_masked_type vregs
)
```

→	vreg_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – Disable the VREG ▪ PM_ON_CMD – Enable the VREG
→	vregs	pm_vreg_mask_type: ▪ uint64 (1 << (pm_vreg_id_type))

Description

Use this function to enable or disable each voltage regulator or a group of voltage regulators by ORing the pm_vreg_masked_type VREG parameter.

Return values

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg_cmd parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the vregs parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable the WLAN1 VREG: */
err |= pm_vreg_control(PM_ON_CMD, (1 << PM_VREG_WLAN1_ID));

/* "Find out if there was a PMIC API error.*/
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

9.2.3 pm_vreg_set_level()

This function programs the voltage levels for the selected voltage regulator.

Parameters

```
pm_err_flag_type  pm_vreg_set_level
(
    pm_vreg_id_type vreg,
    uint16 level
)
```

→	`vreg`	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8 ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28
→	`level`	Voltage setting in mV, e.g., 2700 = 2,700 mV

Description

Use this function to program the voltage levels for the specified voltage regulator.

NOTE: Voltage values between voltage steps are rounded up to the next setting.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set PM_LDO_2 to 1.2 V */
err |= pm_vreg_set_level(PM_LDO_2, 1200);

/* "Find out if there was a PMIC API error.*/
if (err != PM_ERR_FLAG__SUCCESS)
{
   MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

9.2.4 pm_vreg_smps_config()

This function configures the SMPS voltage regulators.

Parameters

```
pm_err_flag_type pm_vreg_smps_config
(
    pm_vreg_smps_id_type   vreg_smps,
    pm_vreg_smps_mode_type mode
)
```

→	vreg_smps	Selected SMPS voltage regulator that will be configured. pm_vreg_smps_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	mode	pm_vreg_smps_mode_type: ▪ PM_VREG_SMPS_MODE__CLK_D0 ▪ PM_VREG_SMPS_MODE__CLK_D1 ▪ PM_VREG_SMPS_MODE__CLK_A0 ▪ PM_VREG_SMPS_MODE__CLK_A1 ▪ PM_VREG_SMPS_MODE__CLK_A2 ▪ PM_VREG_SMPS_MODE__TCXO_EN ▪ PM_VREG_SMPS_MODE__PWM ▪ PM_VREG_SMPS_MODE__PFM ▪ PM_VREG_SMPS_MODE__AUTOMATIC ▪ PM_VREG_SMPS_MODE__HYSTERETIC ▪ PM_VREG_SMPS_MODE__CLK_D0_OFF ▪ PM_VREG_SMPS_MODE__CLK_D1_OFF ▪ PM_VREG_SMPS_MODE__CLK_A0_OFF ▪ PM_VREG_SMPS_MODE__CLK_A1_OFF ▪ PM_VREG_SMPS_MODE__CLK_D0_D1_A0_A1_OFF

Description

Use this function to configure the SMPS mode of the selected SMPS voltage regulators. The modes are:

- PFM – Forces the BUCK VREG into Pulse Frequency Modulation (similar to PBM); used for Low Power mode operation
- PWM – Forces the BUCK VREG into Pulse Width Modulation; used for normal operation
- TXCO_EN – Configures the BUCK VREG to switch automatically to PWM mode when TCXO is active (normal operation) and to PFM mode when TCXO is inactive (sleep)
- Automatic – Configures the BUCK VREG to switch automatically between PFM and PWM depending on the load

NOTE: PM_VREG_SMPS_MODE__TCXO_EN runs the SMPS in PWM mode when TCXO_EN is active. It runs in PFM mode when TCXO_EN is inactive.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg_smps parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the mode parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* initialize S4 for PFM mode */
err |= pm_vreg_smps_config(PM_SMPS_4, PM_VREG_SMPS_MODE__PFM);

/* "Find out if there was a PMIC API error.*/
if (err != PM_ERR_FLAG__SUCCESS)
{
   MSG_ERROR(" PMIC API ERROR(0x%x).",err,0,0);
}
```

9.2.5 pm_vreg_smps_clock_sel()

This function configures the SMPS VREG's clock source.

Parameters

```
pm_err_flag_type pm_vreg_smps_clock_sel
(
    pm_vreg_smps_clk_sel_type clk_sel
)
```

→	clk_sel	pm_vreg_smps_clk_sel_type: ▪ PM_VREG_SMPS_CLK_SEL__TCXO – Set SMPS VREG's clock source to TCXO ▪ PM_VREG_SMPS_CLK_SEL__RC – Default; set SMPS VREG's clock source to RC (Relaxation Crystal)

Description

Use this function to configure the SMPS VREG's clock source.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the clk_sel parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Also, the pm_vreg_smps_tcxo_div_sel() function must be invoked if TCXO is to be selected as the clock source.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Select RC as the SMPS VREGs clock source. */
err |= pm_vreg_smps_clock_sel(PM_VREG_SMPS_CLK_SEL__RC);


if (err != PM_ERR_FLAG__SUCCESS)
{
MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

9.2.6 pm_vreg_smps_tcxo_div_sel()

This function configures the SMPS VREG's TCXO clock source divider.

Parameters

```
pm_err_flag_type pm_vreg_smps_tcxo_div_sel
(
    pm_vreg_smps_tcxo_div_type div
)
```

→	div	pm_vreg_smps_tcxo_div_type div: ▪ PM_VREG_SMPS_TCXO_DIV__8 ▪ PM_VREG_SMPS_TCXO_DIV__10 ▪ PM_VREG_SMPS_TCXO_DIV__12 (default) ▪ PM_VREG_SMPS_TCXO_DIV__14 ▪ PM_VREG_SMPS_TCXO_DIV__16 ▪ PM_VREG_SMPS_TCXO_DIV__18

Description

Use this function to configure the SMPS VREG's TCXO clock source divider.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the div parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure the SMPS VREGs TCXO clock source divider to 16. */
err |= pm_vreg_smps_tcxo_div_sel(PM_VREG_SMPS_TCXO_DIV__16);
/* Select TCXO as the SMPS VREGs clock source. */
err |= pm_vreg_smps_clock_sel(PM_VREG_SMPS_CLK_SEL__TCXO);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

9.2.7 pm_vreg_ldo_bypass_set()

This function sets the Bypass mode to On for a given LDO.

Parameters

```
pm_err_flag_type pm_vreg_ldo_bypass_set
(
    pm_vreg_id_type  bypass_ldo
)
```

→	vreg_id	pm_vreg_id_type: ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28

Description

Use this function to set the Bypass mode to On for a given LDO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the bypass_ldo parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable bypass mode with LDO 11 */
pm_vreg_ldo_bypass_set(PM_LDO_11);
```

9.2.8 pm_vreg_ldo_bypass_clear()

This function clears the Bypass mode for a given LDO.

Parameters

```
pm_err_flag_type pm_vreg_ldo_bypass_clear
(
    pm_vreg_id_type  bypass_ldo
)
```

→	`vreg_id`	pm_vreg_id_type: ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28

Description

Use this function to clear the Bypass mode for a given LDO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the bypass_ldo parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Clear bypass mode with LDO 11 */
(void)pm_vreg_ldo_bypass_clear(PM_LDO_11);
```

9.2.9 pm_vreg_smps_switch_size_set()

This function sets the SMPS switch size. A smaller switch size can improve SMPS efficiency at low current loads, and a larger switch size is required for higher current loads.

Parameters

```
pm_err_flag_type pm_vreg_smps_switch_size_set
(
    pm_vreg_id_type                 vreg,
    pm_vreg_smps_switch_size_type   switch_size
)
```

→	vreg	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	switch_size	pm_vreg_smps_switch_size_type: ▪ PM_SMPS_SWITCH_SIZE_1_8 ▪ PM_8901_SMPS_SWITCH_SIZE_0_8 ▪ PM_SMPS_SWITCH_SIZE_1_4 ▪ PM_8901_SMPS_SWITCH_SIZE_1_8 ▪ PM_SMPS_SWITCH_SIZE_1_2 ▪ PM_SMPS_SWITCH_SIZE_1_1 ▪ PM_8901_SMPS_SWITCH_SIZE_2_8 ▪ PM_SMPS_SWITCH_SIZE_INVALID ▪ PM_8901_SMPS_SWITCH_SIZE_3_8 ▪ PM_8901_SMPS_SWITCH_SIZE_4_8 ▪ PM_8901_SMPS_SWITCH_SIZE_5_8 ▪ PM_8901_SMPS_SWITCH_SIZE_6_8 ▪ PM_8901_SMPS_SWITCH_SIZE_7_8 ▪ PM_8901_SMPS_SWITCH_SIZE_8_8 ▪ PM_8901_SMPS_SWITCH_SIZE_INVALID

Description

Use this function to set the SMPS switch size:

- A smaller switch size can improve SMPS efficiency at low current loads.
- A larger switch size is required for higher current loads.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the switch_size parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the SMPS switch size to 1/2 for S4 */
pm_vreg_smps_switch_size_set(   PM_SMPS_4,
                                PM_SMPS_SWITCH_SIZE_1_2);
```

9.2.10 pm_vreg_smps_pulse_skipping_enable()

This function enables/disables SMPS pulse skipping.

Parameters

```
pm_err_flag_type pm_vreg_smps_pulse_skipping_enable
(
    pm_vreg_id_type      vreg,
    boolean              enable
)
```

→	vreg	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	enable	▪ TRUE -- 1 ▪ FALSE – 0

Description

Use this function to enable or disable SMPS pulse skipping.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the enable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable S1 Pulse Skipping */
err |= pm_vreg_smps_pulse_skipping_enable(PM_SMPS_1, TRUE);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);

```

9.2.11 pm_vreg_pull_down_switch()

This function configures the voltage regulator active pull-down.

Parameters

```
pm_err_flag_type pm_vreg_pull_down_switch
(
    pm_switch_cmd_type vreg_cmd,
    pm_vreg_id_type    vreg
)
```

→	vreg_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD ▪ PM_ON_CMD

→	vreg	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8 ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28 ▪ PM_VS_LVS_1 ▪ PM_VS_LVS_2 ▪ PM_VS_LVS_3 ▪ PM_VS_LVS_4 ▪ PM_VS_LVS_5 ▪ PM_VS_LVS_6 ▪ PM_VS_LVS_7 ▪ PM_VS_OTG_1 ▪ PM_VS_HDMI_1

Description

Use this function to configure the voltage regulator active pull-down.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg_cmd parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the vreg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Configure L3 pull-down.

```
err |= pm_vreg_pull_down_switch(PM_ON_CMD, PM_LDO_3);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

9.2.12 pm_vreg_ncp_sample_comp_mode_enable()

This function enables the sample comparator mode for VREG_NCP.

Parameters

```
pm_err_flag_type pm_vreg_ncp_sample_comp_mode_enable
(
    pm_switch_cmd_type           OnOff,
    pm_ncp_switching_freq_type      freq
)
```

→	OnOff	pm_switch_cmd_type: ▪ PM_OFF_CMD – Sample-comparator mode OFF ▪ PM_ON_CMD – Sample-comparator mode ON
→	freq	pm_ncp_switching_freq_type: ▪ PM_NCP_FREQ_9_6 MHZ ▪ PM_NCP_FREQ_4_8 MHZ ▪ PM_NCP_FREQ_3_2 MHZ ▪ PM_NCP_FREQ_2_4 MHZ ▪ PM_NCP_FREQ_1_92 MHZ ▪ PM_NCP_FREQ_1_6 MHZ (default) ▪ PM_NCP_FREQ_1_37 MHZ ▪ PM_NCP_FREQ_1_2 MHZ

Description

Use this API to enable the sample comparator mode for VREG_NCP. This function will enable the comparator mode and allow us to set the switching frequency.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg_cmd parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the which_vreg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable NCP sample-comparator mode with 9.6 MHz switchign Frequency */
err = pm_vreg_ncp_sample_comp_mode_enable( PM_ON_CMD, PM_NCP_FREQ_9_6MHZ)
```

9.2.13 pm_ncp_control ()

This function enables the NCP functionality.

Parameters

```
pm_err_flag_type pm_ncp_control
(
   pm_switch_cmd_type on_off
)
```

→	on_off	pm_switch_cmd_type: ▪ PM_OFF_CMD – Command to turn off ▪ PM_ON_CMD – Command to turn on

Description

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – Processor does not have access to this resource
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – Resource is invalid; resource index was passed into router that is not defined in the build
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

Side effects

Example

9.2.14 pm_vreg_get_level()

This function returns the millivolt level of a regulator.

Parameters

```
pm_err_flag_type pm_vreg_get_level
(
    pm_vreg_id_type vreg,
    pm_vreg_level_type *vreg_level
)
```

→	vreg	pm_vreg_level_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8 ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28 ▪ PM_VS_LVS_1 ▪ PM_VS_LVS_2 ▪ PM_VS_LVS_3 ▪ PM_VS_LVS_4 ▪ PM_VS_LVS_5 ▪ PM_VS_LVS_6 ▪ PM_VS_LVS_7 ▪ PM_VS_OTG_1 ▪ PM_VS_HDMI_1
→	*vreg_level	pointer of pm_vreg_level_type – Voltage level of the regulator in milivolts

Description

Use this API to the voltage level of regulators in units of millivolts.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for vreg
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
int16 current_mv
if ( pm_vreg_get_level(PM_SMPS_1,
                       (pm_vreg_level_type*)(&current_mv))\!=
     PM_ERR_FLAG__SUCCESS )
{
    ERR_FATAL( "Error getting the current active voltage", 0, 0, 0 );
}
```

9.2.15 pm_lp_mode_control_get()

This function checks if LDO regulator low power mode is enabled.

Parameters

```
pm_err_flag_type pm_lp_mode_control_get
(
    pm_switch_cmd_type     *vreg_cmd,
    pm_vreg_id_type        which_vreg
)
```

→	*vreg_cmd	point of pm_switch_cmd_type: ▪ PM_OFF_CMD – LPM is not enabled ▪ PM_ON_CMD – LPM is enabled
→	which_vreg	pm_vreg_level_type: ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28

Description

Use this API to set the voltage level of regulators in units of millivolts.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for vreg_cmd variable
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for vreg
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_switch_cmd_type gp6_lpm;
err = pm_lp_mode_control_get(&gp6_lpm, PM_LDO_15);
```

9.2.16 pm_lp_force_lpm_control()

This function forces an LDO regulator to enter low power mode.

Parameters

```
pm_err_flag_type pm_lp_force_lpm_control
(
    pm_switch_cmd_type     vreg_cmd,
    pm_vreg_id_type        which_vreg
)
```

→	vreg_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – LDO enters LPM follow TCXO ▪ PM_ON_CMD – LDO enters LPM regardless TCXO
→	which_vreg	pm_vreg_level_type: ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28

Description

Use this API to force an LDO into low power mode, regardless of the state of TCXO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for which_vreg
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Force L15 always in LPM*/
err = pm_lp_force_lpm_control(PM_ON_CMD, PM_LDO_15);
```

9.2.17 pm_vreg_smps_config_get()

This function gets the SMPS mode.

Parameters

```
pm_err_flag_type  pm_vreg_smps_config_get
(
    pm_vreg_id_type        vreg_smps,
    pm_vreg_smps_mode_type *mode
)
```

→	vreg_smps	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	*mode	pointer of pm_vreg_smps_mode_type: ▪ PM_VREG_SMPS_MODE__CLK_D0 ▪ PM_VREG_SMPS_MODE__CLK_D1 ▪ PM_VREG_SMPS_MODE__CLK_A0 ▪ PM_VREG_SMPS_MODE__CLK_A1 ▪ PM_VREG_SMPS_MODE__CLK_A2 ▪ PM_VREG_SMPS_MODE__TCXO_EN ▪ PM_VREG_SMPS_MODE__PWM ▪ PM_VREG_SMPS_MODE__PFM ▪ PM_VREG_SMPS_MODE__AUTOMATIC ▪ PM_VREG_SMPS_MODE__HYSTERETIC ▪ PM_VREG_SMPS_MODE__CLK_D0_OFF ▪ PM_VREG_SMPS_MODE__CLK_D1_OFF ▪ PM_VREG_SMPS_MODE__CLK_A0_OFF ▪ PM_VREG_SMPS_MODE__CLK_A1_OFF ▪ PM_VREG_SMPS_MODE__CLK_D0_D1_A0_A1_OFF

Description

Use this API to get the SMPS mode, TCXO enabled, PWM, PFM, or Automatic.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for vreg
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for mode variable
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Get S4 mode*/
pm_vreg_smps_mode_type s4_mode;
err = pm_vreg_smps_config_get(PM_SMPS_4, &s4_mode);
```

9.2.18 pm_vreg_status_get()

This function returns the VREG on/off state.

Parameters

```
pm_vreg_mask_type pm_vreg_status_get
(
    void
)
```

Description

This function returns the real-time status of the voltage regulators. The return value is a bit-mapped variable where each bit represents the on (1) or off (0) state of the regulators enumerated in pm_vreg_mask_type

Return value

This function returns pm_vreg_mask_type, a bitmap regulator ON/OFF state.

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

None

Example

```
/*Check if S2 and S4 are ON*/
pm_vreg_mask_type vreg_st;
vreg_st = pm_vreg_status_get();

if(    ((vreg_st & 1<< PM_SMPS_2)!=0) &&
       ((vreg_st & 1<< PM_SMPS_4)!=0))
{
…
}
```

9.2.19 pm_vreg_ldo_current_limit_enable()

This function enables the current limit of an LDO regulator.

Parameters

```
pm_err_flag_type pm_vreg_ldo_current_limit_enable
(
    pm_switch_cmd_type     vreg_cmd,
    pm_vreg_id_type        vreg_id
)
```

→	vreg_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – Disable LDO current limit ▪ PM_ON_CMD – Enable LDO current limit
→	vreg_id	pm_vreg_level_type: ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28

Description

Use this API to enable the current limit of an LDO regulator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for which_vreg
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Enable L15 current limit*/
err = pm_vreg_ldo_current_limit_enable(PM_ON_CMD, PM_LDO_15);
```

9.2.20 pm_vreg_smps_switch_driver_size_set()

This function sets the SMPS switch size.

Parameters

```
pm_err_flag_type pm_vreg_smps_switch_driver_size_set
(
    pm_vreg_id_type                       vreg_id,
    pm_vreg_smps_switch_driver_size_type  switch_driver_size
);
```

→	vreg_id	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	switch_driver_size	pm_vreg_smps_switch_driver_size_type: ▪ PM_SMPS_SWITCH_DRIVER_SIZE_1_8 ▪ PM_SMPS_SWITCH_DRIVER_SIZE_1_4 ▪ PM_SMPS_SWITCH_DRIVER_SIZE_1_2 ▪ PM_SMPS_SWITCH_DRIVER_SIZE_1_1

Description

This API will set the SMPS switch size. A smaller switch size is used to improve SMPS efficiency at low current loads. A larger switch size is required.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for which_vreg
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Change the switch size of S4*/
err = pm_vreg_smps_switch_driver_size_set(
        PM_SMPS_4,
        PM_SMPS_SWITCH_DRIVER_SIZE_1_4);
```

9.2.21 pm_vreg_smps_set_stepper_config()

This function sets the SMPS stepper control configuration.

Parameters

```
pm_err_flag_type  pm_vreg_smps_set_stepper_config
(
    pm_vreg_id_type                  vreg_smps,
    pm_switch_cmd_type               vreg_cmd,
    pm_vreg_smps_stepper_delay_type  delay,
    pm_vreg_smps_step_size_type      step_size
)
```

→	vreg_smps	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	vreg_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – Disable stepper ▪ PM_ON_CMD – Enable stepper
→	delay	pm_vreg_smps_stepper_delay_type: ▪ Delay in milliseconds. The range for this parameter is 2.5 to 40 microseconds. (will be set to one of the following values: 2.5, 5, 7.5, 10, 12.5, 15, 20, 40)
→	step_size	pm_vreg_smps_step_size_type: ▪ Step size in millivolts. The range for this parameter is 12.5 to 100 mV (will be set to one of the following values: 12.5, 25, 50, 100)

Description

This API will configure the stepper control for one of the PMIC SMPS voltage regulators.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for which_vreg
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for delay
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for step_size
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure S0 Stepper */
errFlag = pm_vreg_smps_set_stepper_config(PM_VREG_MSMC1_ID,
                                          PM_ON_CMD,
                                          5,
                                          25);
…
Boolean s1_stepping_done = FALSE;
errFlag = pm_vreg_smps_is_stepping_done(  PM_SMPS_1,
                                          &s1_stepping_done);
If(!s1_stepping_done){
    …
}
```

9.2.22 pm_vreg_smps_is_stepping_done()

This function gets the SMPS voltage stepping done status.

Parameters

```
pm_err_flag_type  pm_vreg_smps_is_stepping_done
(
    pm_vreg_id_type                 vreg_smps,
    boolean                         *stepper_status
);
```

→	vreg_smps	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8
→	*stepper_status	pointer of boolean: ▪ TRUE – Stepping is complete ▪ FLASE – Stepping is not complete

Description

Use this API to get the SMPS voltage stepping done status.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__VREG_ID_OUT_OF_RANGE – Invalid value for which_vreg
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

See the example in Section 9.2.21.

9.2.23 pm_vreg_mode_status()

Parameters

```
pm_err_flag_type pm_vreg_mode_status
(
  pm_vreg_id_type     vreg ,
  pm_vreg_mode_type*  mode_status
)
```

→	vreg	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8 ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28
→	mode_status	pointer of pm_vreg_mode_type: ▪ PM_VREG_MODE__AUTO_SLEEP ▪ PM_VREG_MODE__SLEEP_B ▪ PM_VREG_MODE__LPM ▪ PM_VREG_MODE__HPM ▪ PM_VREG_MODE__PIN_CTRLED ▪ PM_VREG_MODE__INVALID

Description

Return value

This function returns pm_err_flag_type.

Dependencies

Side effects

Example

9.2.24 pm_vreg_smps_inductor_ilim()

This function changes the current limit depending on their inductor size (value).

Parameters

```
pm_err_flag_type pm_vreg_smps_inductor_ilim
(
  pm_vreg_smps_inductor_ilim_type ilim_level,
  pm_vreg_id_type                 externalResourceIndex
)
```

→	ilim_level	pm_vreg_smps_inductor_ilim_type: ▪ PM_SMPS_INDUCTOR_ILIM__300MA ▪ PM_SMPS_INDUCTOR_ILIM__700MA ▪ PM_SMPS_INDUCTOR_ILIM__1100MA ▪ PM_SMPS_INDUCTOR_ILIM__1500MA ▪ PM_SMPS_INDUCTOR_ILIM__1900MA ▪ PM_SMPS_INDUCTOR_ILIM__2300MA ▪ PM_SMPS_INDUCTOR_ILIM__2700MA ▪ PM_SMPS_INDUCTOR_ILIM__3100MA
→	externalResourceIndex	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8

Description

Return value

This function returns pm_err_flag_type.

Dependencies

Side effects

Example

9.2.25 pm_vreg_smps_inductor_ilim_status()

This function reads the current limit set (value).

Parameters

```
pm_err_flag_type pm_vreg_smps_inductor_ilim_status
(
  pm_vreg_smps_inductor_ilim_type* ilim_level,
  pm_vreg_id_type                  externalResourceIndex
)
```

→	ilim_level	Pointer of pm_vreg_smps_inductor_ilim_type: ▪ PM_SMPS_INDUCTOR_ILIM__300MA ▪ PM_SMPS_INDUCTOR_ILIM__700MA ▪ PM_SMPS_INDUCTOR_ILIM__1100MA ▪ PM_SMPS_INDUCTOR_ILIM__1500MA ▪ PM_SMPS_INDUCTOR_ILIM__1900MA ▪ PM_SMPS_INDUCTOR_ILIM__2300MA ▪ PM_SMPS_INDUCTOR_ILIM__2700MA ▪ PM_SMPS_INDUCTOR_ILIM__3100MA
→	externalResourceIndex	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8

Description

Return value

This function returns pm_err_flag_type.

Dependencies

Side effects

Example

9.2.26 pm_vote_vreg_switch ()

This function votes for a VREG to be turned on or off.

Parameters

```
void pm_vote_vreg_switch(
    pm_switch_cmd_type cmd,
    pm_vreg_id_type vreg_id,
    pm_vote_vreg_app_type app_mask
)
```

→	cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – Vote to Disable the VREG ▪ PM_ON_CMD – Vote to Enable the VREG
→	vreg_id	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8 ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28
→	app_mask	pm_vote_vreg_app_type – Application Mask

Description

Use this function to vote for a VREG to be turned on or off.

NOTE: This API is highly recommended when vreg is shared by multiple software clients.

Return values

None

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
static pm_vote_vreg_app_type swapp_vote;
…
boolean SWAppInitVreg()
{
   if(pm_vote_vreg_request_vote(PM_LDO_10, &swapp_vote) != TRUE)
   {
      return FALSE;
   }
   return TRUE;
}
…
void SWAppEnableVreg()
{
   pm_vote_vreg_switch(PM_ON_CMD, PM_LDO_10, swapp_vote);
}
…
void SWAppDisableVreg()
{
   pm_vote_vreg_switch(PM_OFF_CMD, PM_LDO_10, swapp_vote);
}
```

9.2.27 pm_vote_vreg_request_vote()

This function requests a vote for use with the PMIC VREG voting API.

Parameters

```
boolean pm_vote_vreg_request_vote
(
    pm_vreg_id_type vreg_id,
    pm_vote_vreg_app_type *vote
)
```

→	vreg_id	pm_vreg_id_type: ▪ PM_SMPS_1 ▪ PM_SMPS_2 ▪ PM_SMPS_3 ▪ PM_SMPS_4 ▪ PM_SMPS_5 ▪ PM_SMPS_6 ▪ PM_SMPS_7 ▪ PM_SMPS_8 ▪ PM_LDO_1 ▪ PM_LDO_2 ▪ PM_LDO_3 ▪ PM_LDO_4 ▪ PM_LDO_5 ▪ PM_LDO_6 ▪ PM_LDO_7 ▪ PM_LDO_8 ▪ PM_LDO_9 ▪ PM_LDO_10 ▪ PM_LDO_11 ▪ PM_LDO_12 ▪ PM_LDO_14 ▪ PM_LDO_15 ▪ PM_LDO_16 ▪ PM_LDO_17 ▪ PM_LDO_18 ▪ PM_LDO_21 ▪ PM_LDO_22 ▪ PM_LDO_23 ▪ PM_LDO_24 ▪ PM_LDO_25 ▪ PM_LDO_26 ▪ PM_LDO_27 ▪ PM_LDO_28
→	*vote	Pointer of pm_vote_vreg_app_type – If this function returns TRUE, vote will be set to the next available vote for vreg_id

Description

This function should be used to request a vote for use with the pm_vote_vreg_switch() API.

Return values

Boolean:

- TRUE – Vote was available
- FALSE – Vote was not available or if vreg_id was out of range

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

See the example in Section 9.2.26.

9.3 Master Bandgap Module API

9.3.1 pm_mbg_lpm_enable()

This function enables/disables MBG low power mode.

Parameters

```
pm_err_flag_type pm_mbg_lpm_enable
(
boolean lp_en,
boolean lp_tcxo_en
)
```

→	lp_en	boolean: ▪ TRUE ▪ FALSE
→	lp_tcxo_en	boolean: ▪ TRUE ▪ FALSE

Description

Use this function to enable/disable MBG low power mode.

The truth table for lp_en parameter and lp_tcxo_en parameter is shown in Table 9-2.

Table 9-2 MBG low power mode configuration

lp_en	lp_txco_en	Functional description
FALSE	(Don't Care)	▪ Low power bandgap is off ▪ Normal power bandgap is on ▪ Reference outputs from normal bandgap
TRUE	FALSE	▪ Low power bandgap is on ▪ Normal power bandgap is off ▪ Reference outputs from low power bandgap
TRUE	TRUE	▪ Low power bandgap is on ▪ Normal bandgap is on when tcxo_en is high, off otherwise ▪ Reference outputs from low power bandgap if tcxo_en is low; from normal bandgap otherwise

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Parameter 1 is out of range
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__SBI_OPT_ERR – SBI driver failed to communicate with PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Enable low power bandgap, normal power bandgap controlled by TCXO_EN*/
err = pm_mbg_lpm_enable(TRUE, TRUE);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
/*Trim lpm mbg*/
err = pm_mbg_lpm_trim();
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

9.3.2 pm_mbg_lpm_trim()

This function controls the trimming of the low power bandgap; it requires both lp_en and tcxo_en high.

Parameters

```
pm_err_flag_type pm_mbg_lpm_trim
(
    void
)
```

Description

Use this function to control the trimming of the low power bandgap; it requires both lp_en and tcxo_en high.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__SBI_OPT_ERR – SBI driver failed to communicate with PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Enable low power bandgap, normal power bandgap controlled by TCXO_EN*/
err = pm_mbg_lpm_enable(TRUE, TRUE);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
/*Trim lpm mbg*/
err = pm_mbg_lpm_trim();
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

9.3.3 pm_mbg_lpm_dec_ref_byp_r()

This function decreases/increases the series resistor with the reference bypass capacitor for MBG low power mode.

Parameters

```
pm_err_flag_type pm_mbg_lpm_dec_ref_byp_r
(
    boolean decrease_R
)
```

→	decrease_R	boolean: ▪ TRUE – Decreases the series resistor to 75 k Ohms ▪ FALSE – Increases the series resistor to 500 k Ohms

Description

Use this function to decrease/increase the series resistor with the reference bypass capacitor for MBG low power mode.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__SBI_OPT_ERR – SBI driver failed to communicate with PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* decreases the series resistor to 75k Ohms */
err = pm_mbg_lpm_dec_ref_byp_r(TRUE);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

9.3.4 pm_mbg_iref_enable()

This function enables or disables MBG iRef.

Parameters

```
pm_err_flag_type pm_mbg_iref_enable
(
    boolean enable
)
```

→	enable	boolean: ▪ TRUE – Enable MBG IRef ▪ FALSE – Disable MBG IRef

Description

Use this function to enable or disable MBG iRef.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC verion
- PM_ERR_FLAG__SBI_OPT_ERR – SBI driver failed to communicate with PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable mbg reference current */
err = pm_mbg_iref_enable(TRUE);
if (err != PM_ERR_FLAG__SUCCESS)
{
    MSG_ERROR("API ERROR(0x%x) DETECTED.",err,0,0);
}
```

10 General Housekeeping

The PM8921 IC includes many circuits that support handset-level housekeeping functions, various tasks that must be performed to keep the handset in order. Integration of these functions reduces the external parts count and the associated size and cost.

Housekeeping functions include an analog multiplexer with gain and offset adjustments, system clock circuits, a real-time clock for time and alarm functions, and over-temperature protection.

10.1 ADC arbiter API

10.1.1 pm_adc_set_mode ()

This function enables/disables the ADC Arbiter.

Parameters

```
pm_err_flag_type pm_adc_set_mode
(
   int externalResourceIndex,
   pm_adc_mode_type mode
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	mode	pm_adc_mode_type: ▪ PM_ADC_MODE__MODULE_OFF ▪ PM_ADC_MODE__MODULE_ON

Description

This function enables/disables the ADC Arbiter.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.2 pm_adc_set_conversion_request()

This function sets the ADC conversion request.

Parameters

```
pm_err_flag_type pm_adc_set_conversion_request
(
   int externalResourceIndex,
   pm_adc_conv_req_cmd_type cmd
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	cmd	pm_adc_conv_req_cmd_type: ▪ PM_ADC_CONVERSION_REQ_CMD__OFF ▪ PM_ADC_CONVERSION_REQ_CMD__ON

Description

This function sets the ADC conversion request. After the request was sent, the request will be stored in the conversion request queue in the PMIC hardware; the request is cleared when the ADC conversion is completed.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.3 pm_adc_get_conversion_status ()

This function gets the ADC conversion status.

Parameters

```
pm_err_flag_type pm_adc_get_conversion_status
(
   int externalResourceIndex,
   pm_adc_conversion_status_type *status
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	*status	Pointer of pm_adc_conversion_status_type: ▪ PM_ADC_CONVERSION_STATUS__COMPLETE ▪ PM_ARB_ADC_CONVERSION_STATUS__PENDING

Description

This function gets the ADC conversion status based on the status of the conversion status strobe and the end-of-conversion status flag.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Invalid pointer passed in

Dependencies

Side effects

Example

10.1.4 pm_adc_set_conv_sequencer ()

This function enables/disables the conversion sequencer.

NOTE: This API is available only on the modem processor.

Parameters

```
pm_err_flag_type pm_adc_set_conv_sequencer
(
   int externalResourceIndex,
   pm_adc_cmd_type cmd
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	cmd	pm_adc_cmd_type: ▪ PM_ARB_ADC_CMD__ENABLE ▪ PM_ARB_ADC_CMD__DISABLE

Description

This function enables/disables the conversion sequencer.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.5 pm_adc_set_conv_trig_condition ()

This function sets the conversion trigger condition.

NOTE: This API is available only on the modem processor.

Parameters

```
pm_err_flag_type pm_adc_set_conv_trig_condition
(
   int externalResourceIndex,
   pm_adc_conv_trig_type trig
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	trig	pm_adc_conv_trig_type: ▪ PM_ADC_CONV_TRIG__FE – Failing edge ▪ PM_ADC_CONV_TRIG__RE – Rising edge

Description

This function sets the conversion trigger condition that starts the ADC holdoff timer.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.6 pm_adc_set_holdoff_time ()

This function sets the holdoff time.

NOTE: This API is available only on the modem processor.

Parameters

```
pm_err_flag_type pm_adc_set_holdoff_time
(
   int externalResourceIndex,
   uint32 time_us
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	time_us	▪ uint32 – The holdoff time in microseconds

Description

This function sets the holdoff time which is the delay from conversion trigger transition to ADC enable.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.7 pm_adc_set_timeout_time ()

This function sets the timeout time.

NOTE: This API is available only on the modem processor.

Parameters

```
pm_err_flag_type pm_adc_set_timeout_time
(
   int externalResourceIndex,
   uint32 time_us
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	time_us	uint32 – Timeout time in microseconds

Description

This function sets the timeout time which is the delay from the SBI conversion request to the triggering conversion sequencer holdoff timer.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.8 pm_adc_get_conv_sequencer_status ()

This function gets the status of the conversion sequencer.

NOTE: This API is available only on the modem processor.

Parameters

```
pm_err_flag_type pm_adc_get_conv_sequencer_status
(   int externalResourceIndex,
    pm_adc_conv_seq_status_type *status
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	*status	Pointer of pm_adc_conv_seq_status_type: ▪ PM_ARB_ADC_CONVERSION_STATUS__INVALID ▪ PM_ARB_ADC_CONVERSION_STATUS__COMPLETE ▪ PM_ARB_ADC_CONVERSION_STATUS__OCCURRING ▪ PM_ARB_ADC_CONVERSION_STATUS__WAITING

Description

This function gets the status of the conversion sequencer, including the SBI conversion request timeout flag and status of FIFO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Invalid pointer passed in

Dependencies

Side effects

Example

10.1.9 pm_adc_set_amux ()

This function sets up the Amux.

Parameters

```
pm_err_flag_type pm_adc_set_amux
(   int externalResourceIndex,
    uint8 channel,
    pm_adc_mux_type mux
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	channel	uint8 – The analog channel
→	mux	pm_adc_mux_type: ▪ PM_ADC_AMUX_MAIN ▪ PM_ADC_PREMUX_TO_CH6 ▪ PM_ADC_PREMUX_TO_CH7 ▪ PM_ADC_RSV_DISABLED

Description

This function sets up the Amux, such as channel and premux selection.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Input Parameter three is out of range

Dependencies

Side effects

Example

10.1.10 pm_adc_get_resource_prescalar ()

This function gets the prescaler value for the specific channel.

Parameters

```
pm_err_flag_type pm_adc_get_resource_prescalar
(
   int externalResourceIndex,
   uint8 channel,
   uint8 *numerator,
   uint8 *denominator
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	channel	uint8 – The analog channel
→	*numerator	Pointer of uint8 – The numerator of the prescalar
→	*denominator	Pointer of uint8 – The denominator of the prescalar

Description

This function gets the prescaler value for the specific channel.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Invalid pointer passed in

Dependencies

Side effects

Example

10.1.11 pm_adc_set_xoadc_input ()

This function sets the ADX input.

Parameters

```
pm_err_flag_type pm_adc_set_xoadc_input
(
   int externalResourceIndex,
   pm_adc_xoadc_input_select_type input
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	input	pm_adc_xoadc_input_select_type: ▪ PM_ARB_ADC_ADC_INPUT_PMIC ▪ PM_ARB_ADC_ADC_INPUT_XO ▪ PM_ARB_ADC_ADC_INPUT_VREFP

Description

This function sets the XOADC input that is used for ARB_ADC.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.12 pm_adc_set_xoadc_decimation_ratio ()

This function sets the XOADC decimation ratio.

Parameters

```
pm_err_flag_type pm_adc_set_xoadc_decimation_ratio
(
   int externalResourceIndex,
   pm_adc_xoadc_decimation_ratio_type ratio
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	ratio	pm_adc_xoadc_decimation_ratio_type: ▪ PM_ARB_ADC_ADC_DECIMATION_RATIO_512 ▪ PM_ARB_ADC_ADC_DECIMATION_RATIO_1K ▪ PM_ARB_ADC_ADC_DECIMATION_RATIO_2K ▪ PM_ARB_ADC_ADC_DECIMATION_RATIO_4K

Description

This function sets the XOADC decimation ratio that is used for ARB_ADC.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.13 pm_adc_set_xoadc_conversion_rate ()

This function sets the XOADC clock rate.

Parameters

```
pm_err_flag_type pm_adc_set_xoadc_conversion_rate
(
   int externalResourceIndex,
   pm_adc_xoadc_conversion_rate_type rate
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	rate	pm_adc_xoadc_conversion_rate_type: ▪ PM_ARB_ADC_ADC_CONVERSION_RATE_TCXO_DIV_8 ▪ PM_ARB_ADC_ADC_CONVERSION_RATE_TCXO_DIV_4 ▪ PM_ARB_ADC_ADC_CONVERSION_RATE_TCXO_DIV_2 ▪ PM_ARB_ADC_ADC_CONVERSION_RATE_TCXO

Description

This function sets the XOADC clock rate that is used for ARB_ADC.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Input Parameter two is out of range

Dependencies

Side effects

Example

10.1.14 pm_adc_get_xoadc_data ()

This function gets the XOADC conversion data.

Parameters

```
pm_err_flag_type pm_adc_get_xoadc_data
(
   int externalResourceIndex,
   uint32* data
)
```

→	externalResourceIndex	int – The external ARB_ADC ID
→	data	Pointer of uint32 – This parameter is used to store the ADC conversion result

Description

This function gets the XOADC conversion data.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature not available on this PMIC version
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – The current processor does not have the access to the ARB_ADC module
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – The external resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_ARB_ADC_INDEXED – The internal resource ID for the ARB_ADC module is not correct
- PM_ERR_FLAG__INVALID_POINTER – Invalid input pointer

Dependencies

Side effects

Example

11 Multipurpose Pins API

The PM8921 IC includes 12 multipurpose pins (MPPs) having the following SBI-configurable options:

- Digital input – Digital inputs applied to the pin can be read via SBI, can trigger an interrupt, or can be routed to another MPP (making this pin the input side of a level translator or current sink controller). The logic level is programmable, providing compliance between I/Os that are running off different power supplies.
- Digital output – The output signal can be set through the SBI to logic LOW or HIGH, can come from this pin's complementary MPP (making this pin the output side of a level translator), or can be tri-stated for use as a switch. The logic level is programmable, providing compliance between I/Os that are running off different power supplies.
- Bidirectional I/O – The two MPPs making up a complementary pair can be jointly configured as a bidirectional, level-translating pair.
- Analog input – Inputs are routed to the analog multiplexer's switch network; if selected, that analog voltage is routed to the MSM HKADC for digitization.
- Analog output – A scaled version (1x) of the on-chip voltage reference (VREF) is buffered and provided as an output.
- Current sink – A programmable current sink is enabled via SBI or in response to its paired MPP input status; its uses include a general-purpose LED driver.

Each multipurpose pin (MPP) can be used independently or paired with its complement and used as a level translator or a bidirectional, level-translating path. In a level translator application, the odd side MPP (MPP1, MPP3, etc.) is configured as the input and the even side MPP (MPP2, MPP4, etc.) is configured as the output, each side is pulled-up to a selectable power supply, and the two are connected together on the internal side of the blocks using a switch at the even MPP block. In a bidirectional application, FET switches provide a path directly between the odd and even pins, the odd and even sides are each pulled up to a selectable power supply using programmable resistors, and the external drivers must be open-drain (always true when used as an RUIM level translator).

Depending upon which one of the seven possible configurations is selected, additional MPP characteristics are also programmed via SBI.

11.1 pm_mpp_config_digital_input()

This function configures the selected MPP as digital input pin.

Parameters

```
pm_err_flag_type pm_mpp_config_digital_input
(
    pm_mpp_which_type          mpp,
    pm_mpp_dlogic_lvl_type     level,
    pm_mpp_dlogic_in_dbus_type dbus
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	dbus	pm_mpp_dlogic_in_dbus_type: ▪ PM_MPP__DLOGIC_IN__DBUS_NONE ▪ PM_MPP__DLOGIC_IN__DBUS1 ▪ PM_MPP__DLOGIC_IN__DBUS2 ▪ PM_MPP__DLOGIC_IN__DBUS3

Description

Use this function to configure the digital voltage level and input Data Bus (DBUS) shift value of the selected MPP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the dbus parameter
- PM_ERR_FLAG__DBUS_IS_BUSY_MODE – DBUS is already in use by another MPP
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_mpp_config_digital_input(PM_MPP_5,
                            PM_MPP__DLOGIC__LVL_MSMP,
                            PM_MPP__DLOGIC_IN__DBUS_NONE);
```

11.2 pm_mpp_config_digital_output()

This function configures the selected MPP as digital output pin.

Parameters

```
pm_err_flag_type pm_mpp_config_digital_output
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type      level,
    pm_mpp_dlogic_out_ctrl_type output_ctrl
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	output_ctrl	pm_mpp_dlogic_out_ctrl_type: ▪ PM_MPP__DLOGIC_OUT__CTRL_LOW ▪ PM_MPP__DLOGIC_OUT__CTRL_HIGH ▪ PM_MPP__DLOGIC_OUT__CTRL_MPP ▪ PM_MPP__DLOGIC_OUT__CTRL_NOT_MPP

Description

Use this function to configure the digital voltage level and output control value of the selected MPP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the output_ctrl parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_mpp_config_digital_output(PM_MPP_10, PM_MPP__DLOGIC__LVL_RUIM,
PM_MPP__DLOGIC_OUT__CTRL_MPP);
```

11.3 pm_mpp_config_digital_inout()

This function configures the selected MPP as a digital bidirectional pin.

Parameters

```
pm_err_flag_type pm_mpp_config_digital_inout
(
    pm_mpp_which_type            mpp,
    pm_mpp_dlogic_lvl_type       level,
    pm_mpp_dlogic_inout_pup_type pull_up
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	pull_up	pm_mpp_dlogic_inout_pup_type: ▪ PM_MPP__DLOGIC_INOUT__PUP_1K ▪ PM_MPP__DLOGIC_INOUT__PUP_3K ▪ PM_MPP__DLOGIC_INOUT__PUP_10K ▪ PM_MPP__DLOGIC_INOUT__PUP_30K

Description

Use this function to configure the digital voltage level and bidirectional pull-up value of an MPP used in a bidirectional level translation pair. This function must be called twice, once each for the MPP pair (for example, once on MPP1 and once on MPP2).

The MPP pairings are summarized as follows:

- Pin 21, GP1_DRV_N (MPP1) ↔ Pin 80, REF_OUT (MPP2)
- Pin 26, RUIM_IO (MPP3) ↔ Pin 30, RUIM_M_IO (MPP4)
- Pin 34, RUIM_M_CLK (MPP5) ↔ Pin 38, RUIM_CLK (MPP6)
- Pin 40, RUIM_M_RST (MPP7) ↔ Pin 43, RUIM_RST (MPP8)
- Pin 66, CBL1PWR_N (MPP9) ↔ Pin 68, CBL0PWR_N (MPP10)
- Pin 70, AMUX_IN1 (MPP11) ↔ Pin 72, AMUX_IN2 (MPP12)

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the pull_up parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Configure MPP11 and MPP12 to be bidirectional. Set their logic voltage reference to RUIM VREG. Set up the pull-up resistor setting to 30K Ohms.

```
pm_mpp_config_digital_inout( PM_MPP_11,
                             PM_MPP__DLOGIC__LVL_RUIM,
                             PM_MPP__DLOGIC_INOUT__PUP_30K);
pm_mpp_config_digital_inout( PM_MPP_12,
                             PM_MPP__DLOGIC__LVL_RUIM,
                             PM_MPP__DLOGIC_INOUT__PUP_30K);
```

11.4 pm_mpp_config_analog_input()

This function configures the selected MPP as an analog input.

Parameters

```
pm_err_flag_type pm_mpp_config_analog_input

(
    pm_mpp_which_type  mpp,
    pm_mpp_ain_ch_type ch_select
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	ch_select	AMUX to which the selected MPP will be routed. pm_mpp_ain_ch_type: ▪ PM_MPP__AIN__CH_AMUX5 ▪ PM_MPP__AIN__CH_AMUX6 ▪ PM_MPP__AIN__CH_AMUX7 ▪ PM_MPP__AIN__CH_AMUX8 ▪ PM_MPP__AIN__CH_AMUX9 ▪ PM_MPP__AIN__CH_ABUS1 ▪ PM_MPP__AIN__CH_ABUS2 ▪ PM_MPP__AIN__CH_ABUS3

Description

Use this function to configure the AMUX input value of the selected MPP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the ch_select parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Route MPP6 to PM_MPP__AIN__CH_AMUX9 analog MUX */
errFlag = pm_mpp_config_analog_input(PM_MPP_6,PM_MPP__AIN__CH_AMUX9);
```

11.5 pm_mpp_config_analog_output()

This function configures the selected MPP as an analog output.

Parameters

```
pm_err_flag_type pm_mpp_config_analog_output
(
    pm_mpp_which_type       mpp,
    pm_mpp_aout_level_type  level,
    pm_mpp_aout_switch_type OnOff
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_aout_level_type: ▪ PM_MPP__AOUT__LEVEL_VREF_1p25_Volts ▪ PM_MPP__AOUT__LEVEL_VREF_0p625_Volts ▪ PM_MPP__AOUT__LEVEL_VREF_2p50_Volts,
→	OnOff	AMUX to which the selected MPP will be routed. pm_mpp_aout_switch_type ▪ PM_MPP__AOUT__SWITCH_OFF ▪ PM_MPP__AOUT__SWITCH_ON ▪ PM_MPP__AOUT__SWITCH_ON_IF_MPP_HIGH ▪ PM_MPP__AOUT__SWITCH_ON_IF_MPP_LOW

Description

Use this function to configure the analog output voltage and switch value of the selected MPP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP8 to be an analog out and turn it on */
err = pm_mpp_config_analog_output(PM_MPP_8,
                                  PM_MPP__AOUT__LEVEL_VREF_0p625_Volts,
                                  PM_MPP__AOUT__SWITCH_ON);
```

11.6 pm_mpp_config_i_sink()

This function configures the selected MPP as a current sink.

Parameters

```
pm_err_flag_type pm_mpp_config_i_sink
(
    pm_mpp_which_type         mpp,
    pm_mpp_i_sink_level_type  level,
    pm_mpp_i_sink_switch_type OnOff
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_i_sink_level_type: ▪ PM_MPP__I_SINK__LEVEL_5mA ▪ PM_MPP__I_SINK__LEVEL_10mA ▪ PM_MPP__I_SINK__LEVEL_15mA ▪ PM_MPP__I_SINK__LEVEL_20mA ▪ PM_MPP__I_SINK__LEVEL_25mA ▪ PM_MPP__I_SINK__LEVEL_30mA ▪ PM_MPP__I_SINK__LEVEL_35mA ▪ PM_MPP__I_SINK__LEVEL_40mA
→	OnOff	pm_mpp_i_sink_switch_type: ▪ PM_MPP__I_SINK__SWITCH_DIS ▪ PM_MPP__I_SINK__SWITCH_ENA ▪ PM_MPP__I_SINK__SWITCH_ENA_IF_MPP_HIGH ▪ PM_MPP__I_SINK__SWITCH_ENA_IF_MPP_LOW

Description

Use this function to configure the current sink level and enable or disable the current sink of the selected MPP.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP7 to be a current sink. Set the current sink to 25mA and
enable the current sink. */
err = pm_mpp_config_i_sink(PM_MPP_7,
                           PM_MPP__I_SINK__LEVEL_25mA,
                           PM_MPP__I_SINK__SWITCH_ENA);
```

11.7 pm_mpp_config_atest()

This function configures a selected MPP as an ATEST.

Parameters

```
pm_err_flag_type pm_mpp_config_atest
 (
   pm_mpp_which_type      mpp,
   pm_mpp_ch_atest_type   atest_select
 )
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	atest_select	pm_mpp_ch_atest_type: ▪ PM_MPP__CH_ATEST1 ▪ PM_MPP__CH_ATEST2 ▪ PM_MPP__CH_ATEST3

Description

Use this function to configure a selected MPP as an ATEST.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the atest_select parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP7 ATEST2. */
err = pm_mpp_config_atest(PM_MPP_7, PM_MPP__CH_ATEST2);
```

11.8 pm_secure_mpp_config_digital_output()

This function configures one of the PMIC MPPs as a digital output.

Parameters

```
pm_err_flag_type pm_secure_mpp_config_digital_output
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type      level,
    pm_mpp_dlogic_out_ctrl_type output_ctrl
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	output_ctrl	pm_mpp_dlogic_out_ctrl_type: ▪ PM_MPP__DLOGIC_OUT__CTRL_LOW ▪ PM_MPP__DLOGIC_OUT__CTRL_HIGH ▪ PM_MPP__DLOGIC_OUT__CTRL_MPP ▪ PM_MPP__DLOGIC_OUT__CTRL_NOT_MPP

Description

Use this function to configure one of the PMIC MPPs as a digital output. This function is intended to be used by the Application (Apps) processor through a Remote Procedure Call (RPC).

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the output_ctrl parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Configure MPP7 to be a digital output with MSMP level and active high. */
err = pm_secure_mpp_config_digital_output(PM_MPP_7,
                            PM_MPP__DLOGIC__LVL_MSMP,
                            PM_MPP__DLOGIC_OUT__CTRL_HIGH);
```

11.9 pm_config_secure_mpp_config_digital_output()

This function configures PMIC MPPs for secure access through RPC.

Parameters

```
pm_err_flag_type pm_config_secure_mpp_config_digital_output
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type      level,
    uint16                      numOfLevels,
    pm_mpp_dlogic_out_ctrl_type output_ctrl,
    uint16                      numOfCtrls
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	numOfLevels	Number of output levels
→	output_ctrl	pm_mpp_dlogic_out_ctrl_type: ▪ PM_MPP__DLOGIC_OUT__CTRL_LOW ▪ PM_MPP__DLOGIC_OUT__CTRL_HIGH ▪ PM_MPP__DLOGIC_OUT__CTRL_MPP ▪ PM_MPP__DLOGIC_OUT__CTRL_NOT_MPP
→	numOfCtrls	Number of output controls

Description

Use this function to configure PMIC MPPs for secure access through RPC. Call this function once for each MPP subsequent calls for the same MPP will overwrite old settings.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter

- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the numOfLevels parameter
- PM_ERR_FLAG__PAR4_OUT_OF_RANGE – Invalid value for the output_ctrl parameter
- PM_ERR_FLAG__PAR5_OUT_OF_RANGE – Invalid value for the numOfCtrls parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP7 to be a digital output with MSMP level and active high.*/
err = pm_config_secure_mpp_config_digital_output(PM_MPP_7,
                        PM_MPP__DLOGIC__LVL_MSMP,
                        PM_MPP__DLOGIC_OUT__CTRL_HIGH);
pm_mpp_dlogic_out_ctrl_type out_ctrl[3];
   pm_mpp_dlogic_lvl_type levels[3];

   out_ctrl[0] = xxx;
   out_ctrl[1] = xxx;
   out_ctrl[2] = xxx;
   levels[0] = yyy;
   levels[1] = yyy;
   levels[2] = yyy;
   err |= pm_config_secure_mpp_config_digital_output(PM_MPP_1,
                                                     levels,
                                                     3,
                                                     out_ctrl,
                                                     3);
```

11.10 pm_config_secure_mpp_config_i_sink()

This function configures I[sink] for PMIC MPPs for secure access through RPC.

Parameters

```
pm_err_flag_type pm_config_secure_mpp_config_i_sink
(
    pm_mpp_which_type         mpp,
    boolean                   i_sink_en_dis
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	i_sink_en_dis	▪ TRUE – Allow the pm_mpp_config_i_sink through an RPC version of the pm_secure_mpp_config_i_sink to be called by the Apps processor ▪ FALSE – Do not allow pm_mpp_config_i_sink to be called by the Apps processor

Description

Use this function to configure I[sink] for PMIC MPPs for secure access through RPC. Call this function once for each MPP called from Task Main Control (TMC) to configure security level subsequent calls for the same MPP will overwrite old settings.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the i_sink_en_dis parameter
- – Invalid value for the xxx parameter PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable pm_make_secure__mpp_config_i_sink for MPP1 but enable MPP2 to be
called by APPs processor.
*/
   err = pm_config_secure_mpp_config_i_sink ( PM_MPP_1 , DISABLE );
   err = pm_config_secure_mpp_config_i_sink ( PM_MPP_2 , ENABLE );
```

11.11 pm_secure_mpp_config_i_sink()

This secure function configures the selected MPP as a current sink.

Parameters

```
pm_err_flag_type pm_secure_mpp_config_i_sink
(
    pm_mpp_which_type         mpp,
    pm_mpp_i_sink_level_type  level ,
    pm_mpp_i_sink_switch_type OnOff
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_i_sink_level_type: ▪ PM_MPP__I_SINK__LEVEL_5mA ▪ PM_MPP__I_SINK__LEVEL_10mA ▪ PM_MPP__I_SINK__LEVEL_15mA ▪ PM_MPP__I_SINK__LEVEL_20mA ▪ PM_MPP__I_SINK__LEVEL_25mA ▪ PM_MPP__I_SINK__LEVEL_30mA ▪ PM_MPP__I_SINK__LEVEL_35mA ▪ PM_MPP__I_SINK__LEVEL_40mA
→	OnOff	pm_mpp_i_sink_switch_type: ▪ PM_MPP__I_SINK__SWITCH_DIS ▪ PM_MPP__I_SINK__SWITCH_ENA ▪ PM_MPP__I_SINK__SWITCH_ENA_IF_MPP_HIGH ▪ PM_MPP__I_SINK__SWITCH_ENA_IF_MPP_LOW

Description

This secure function configures the selected MPP as a current sink. This API should be remoted to the application processor.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable pm_make_secure__mpp_config_i_sink for MPP1 but enable MPP2 to be
called by APPs processor.
*/
   err = pm_secure_mpp_config_i_sink(PM_MPP_15,
                                     PM_MPP__I_SINK__LEVEL_25mA,
                                     PM_MPP__I_SINK__SWITCH_ENA);
```

11.12 pm_secure_mpp_config_digital_input()

This function configures one of the PMIC MPPs as a digital input.

Parameters

```
pm_err_flag_type pm_secure_mpp_config_digital_input
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type      level,
    pm_mpp_dlogic_in_dbus_type  dbus
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	dbus	pm_mpp_dlogic_in_dbus_type: ▪ PM_MPP__DLOGIC_IN__DBUS_NONE ▪ PM_MPP__DLOGIC_IN__DBUS1 ▪ PM_MPP__DLOGIC_IN__DBUS2 ▪ PM_MPP__DLOGIC_IN__DBUS3

Description

Use this function to configure one of the PMIC MPPs as a digital input. This function is to be used by the APPS processor through RPC.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the dbus parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP7 to be a digital input with MSMP level and to dbus2. */
err = pm_secure_mpp_config_digital_input( PM_MPP_7,
                                          PM_MPP__DLOGIC__LVL_MSMP,
                                          PM_MPP__DLOGIC_IN__DBUS2 );
```

11.13 pm_config_secure_mpp_config_digital_input()

This function configures PMIC MPPs for secure access through RPC.

Parameters

```
pm_err_flag_type pm_config_secure_mpp_config_digital_input
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type *    levels,
    uint16                      numOfLevels,
    pm_mpp_dlogic_in_dbus_type *data_line,
    uint16                      numOfDataLines
)
```

→	mpp	pm_mpp_which_type are: ▪ PM_MPP_1 through PM_MPP_16
→	levels	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL,
→	numOfLevels	Number of output levels
→	data_line	pm_mpp_dlogic_in_dbus_type: ▪ PM_MPP__DLOGIC_IN__DBUS_NONE ▪ PM_MPP__DLOGIC_IN__DBUS1 ▪ PM_MPP__DLOGIC_IN__DBUS2 ▪ PM_MPP__DLOGIC_IN__DBUS3
→	numOfDataLines	Number of output controls

Description

Use this function to configure PMIC MPPs for secure access through RPC. Call this function once for each MPP subsequent calls for the same MPP will overwrite old settings.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the levels parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the numOfLevels parameter
- PM_ERR_FLAG__PAR4_OUT_OF_RANGE – Invalid value for the data_line parameter
- PM_ERR_FLAG__PAR5_OUT_OF_RANGE – Invalid value for the numOfDataLines parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_mpp_dlogic_in_dbus_type data_line[3];
pm_mpp_dlogic_lvl_type levels[3];

data_line[0] = xxx;
data_line[1] = xxx;
data_line[2] = xxx;

levels[0] = yyy;
levels[1] = yyy;
levels[2] = yyy;
err = pm_config_secure_mpp_config_digital_input(PM_MPP_1,
                                                levels, 3,
                                                data_line, 3);
err = pm_config_secure_mpp_config_digital_input (PM_MPP_7,
                                                 PM_MPP__DLOGIC__LVL_MSMP,
                                                 PM_MPP__DLOGIC_OUT__CTRL_HIGH);
```

```
1         pm_mpp_dlogic_out_ctrl_type out_ctrl[3];
2         pm_mpp_dlogic_lvl_type levels[3];
3
4         out_ctrl[0] = xxx;
5         out_ctrl[1] = xxx;
6         out_ctrl[2] = xxx;
7         levels[0] = yyy;
8         levels[1] = yyy;
9         levels[2] = yyy;
10
11        err |= pm_config_secure_mpp_config_digital_output(PM_MPP_1,
12                                                          levels,
13                                                          3,
14                                                          out_ctrl,
15                                                          3);
```

11.14 pm_mpp_config_dtest_output()

This function configures the selected Multi Purpose pin (MPP) to be a digital test output pin

Parameters

```
pm_err_flag_type pm_mpp_config_dtest_output
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type      level,
    pm_mpp_dlogic_out_dbus_type  dbus
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL
→	dbus	pm_mpp_dlogic_out_dbus_type: ▪ PM_MPP__DLOGIC_OUT__DBUS1 ▪ PM_MPP__DLOGIC_OUT__DBUS2 ▪ PM_MPP__DLOGIC_OUT__DBUS3 ▪ PM_MPP__DLOGIC_OUT__DBUS4

Description

Use this function to configure the selected Multi Purpose pin (MPP) to be a digital test output pin.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the dbus parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP3 to be the output pin of dtest1. */
(void)pm_mpp_config_dtest_output(    PM_MPP_3,
                                     PM_MPP__DLOGIC__LVL_VDD,
                                     PM_MPP__DLOGIC_OUT__DBUS1);
```

11.15 pm_secure_mpp_config_dtest_output()

This function configures the selected Multi Purpose pin (MPP) to be a digital test output pin from Apps processor.

Parameters

```
pm_err_flag_type pm_secure_mpp_config_dtest_output
(
    pm_mpp_which_type           mpp,
    pm_mpp_dlogic_lvl_type      level,
    pm_mpp_dlogic_out_dbus_type dbus
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	level	pm_mpp_dlogic_lvl_type: ▪ PM_MPP__DLOGIC__LVL_USIM ▪ PM_MPP__DLOGIC__LVL_XO_OUT ▪ PM_MPP__DLOGIC__LVL_SDCC1 ▪ PM_MPP__DLOGIC__LVL_RFA ▪ PM_MPP__DLOGIC__LVL_WLAN1 ▪ PM_MPP__DLOGIC__LVL_VDD ▪ PM_MPP__DLOGIC__LVL_MSME ▪ PM_MPP__DLOGIC__LVL_MSMP ▪ PM_MPP__DLOGIC__LVL_RUIM ▪ PM_MPP__DLOGIC__LVL_MMC ▪ PM_MPP__DLOGIC__LVL_MPLL
→	dbus	pm_mpp_dlogic_out_dbus_type: ▪ PM_MPP__DLOGIC_OUT__DBUS1 ▪ PM_MPP__DLOGIC_OUT__DBUS2 ▪ PM_MPP__DLOGIC_OUT__DBUS3 ▪ PM_MPP__DLOGIC_OUT__DBUS4

Description

This function configures one of the PMIC MPPs as a dtest output. This function is intended to be used by the APPs processor through RPC.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the dbus parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure MPP3 to be the output pin of dtest1. */
(void) pm_secure_mpp_config_dtest_output (    PM_MPP_3,
                                              PM_MPP__DLOGIC__LVL_VDD,
                                              PM_MPP__DLOGIC_OUT__DBUS1);
```

11.16 pm_mpp_status_get()

This function returns the status of one of the PMIC MPPs.

Parameters

```
pm_err_flag_type pm_mpp_status_get
(
    pm_mpp_which_type mpp,
    pm_mpp_status_type *mpp_status
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	mpp_status	pm_mpp_status_type pointer: ▪ Return the status with pm_mpp_status_type struct

Description

Use this function to get the status of one of the PMIC MPPs.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mpp parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the level parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Get current configuration for MPP3*/
pm_mpp_status_type mpp3_st;
(void) pm_mpp_status_get(PM_MPP_3, &mpp3_st);
```

11.17 pm_get_mpp_with_shunt_cap_list_status_for_device()

Parameters

```
pm_err_flag_type  pm_get_mpp_with_shunt_cap_list_status_for_device
(
   pm_mpp_which_type  mpp,
   uint32*            shuntList
)
```

→	mpp	pm_mpp_which_type: ▪ PM_MPP_1 through PM_MPP_16
→	shuntList	uint32 pointer: ▪

Description

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – Processor does not have access to this resource
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – Resource is invalid. Resource index was pass into router that is not defined in the build
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

Side effects

Example

12 TCXO Controller API

The PMIC contains a TCXO controller to allow the main system reference (called the TCXO clock) regulator to warm up the TCXO module and to provide buffering for the TCXO reference. The TCXO controller operates in two modes: Automatic or Manual.

In Automatic mode, the TCXO controller uses a built-in, 11-bit programmable timer to drive the enable signal for the TCXO buffer. The timer is started by a transition on the TCXO_EN pin on the PMIC.

In Manual mode, the TCXO buffer as well as the TCXO voltage regulator is controlled via the software (for details, see Chapter 2.

This API allows you to:

- Set the TCXO_EN pin polarity
- Set the 11-bit programmable timer
- Set the TCXO regulator to Manual or Automatic mode and enable or disable the TCXO buffer (see pm_tcxo_cmd() in Section 12.3).

12.1 pm_config_tcxo_ctrl()

This function configures the operational characteristics of the TCXO controller.

Parameters

```
pm_err_flag_type pm_config_tcxo_ctrl
(
    pm_polarity_type polarity,
    uint16   delay_cycles
)
```

→	`polarity`	pm_polarity_type: ▪ PM_POLARITY_ACTIVE_HIGH ▪ PM_POLARITY_ACTIVE_LOW
→	`delay_cycles`	Warm-up period in multiples of sleep clock cycles before TCXO buffer is turned on. Values are: ▪ 0x003 through 0x802

Description

Use this function to configure the following:

- Polarity of TCXO_EN pin used for controlling TCXO buffer
- Warm-up period for the TCXO buffer when it is turned on

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the polarity parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the delay_cycles parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the TCXO buffer delay to (100 * sleep clock periods) */
errFlag = pm_config_tcxo_ctrl(PM_POLARITY_ACTIVE_HIGH, 100);
```

12.2 pm_tcxo_set_drive_strength()

This function sets the TCXO buffer output drive strength to either high or low.

Parameters

```
pm_err_flag_type pm_tcxo_set_drive_strength
(
   pm_tcxo_drive_strength_type drive_strength,
   pm_tcxo_buffer_type which_buffer
)
```

→	drive_strength	pm_tcxo_drive_strength_type: ▪ PM_TCXO_DRIVE_STRENGTH_LOW ▪ PM_TCXO_DRIVE_STRENGTH_MEDIUM ▪ PM_TCXO_DRIVE_STRENGTH_NORMAL ▪ PM_TCXO_DRIVE_STRENGTH_HIGH
→	which_buffer	pm_tcxo_buffer_type: ▪ PM_TCXO_BUFFER_D0 ▪ PM_TCXO_BUFFER_D1

Description

Use this function to set the TCXO buffer output drive strength to either high or low.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the drive_strength parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the which_buffer parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the D0 buffer output drive strength to LOW. */
errFlag = pm_tcxo_set_drive_strength(PM_TCXO_DRIVE_STRENGTH_LOW,
                                     PM_TCXO_BUFFER_D0);
```

12.3 pm_tcxo_cmd()

This function sets the TCXO controller in the following modes: manual, automatic, TCXO buffer on/off, and emulation of TCXO_EN active/inactive command.

Parameters

```
pm_err_flag_type pm_tcxo_cmd
(
    pm_tcxo_cmd_type cmd
)
```

→	cmd	pm_tcxo_cmd_type: ▪ PM_TCXO_MANUAL_MODE_CMD – Configure the controller to be operable in Manual mode wherein the switching is done in software using APIs ▪ PM_TCXO_AUTOMATIC_MODE_CMD – Configure the controller to be operable in Automatic mode wherein the switching is done in hardware using TCXO_EN pin ▪ PM_TCXO_BUFFER_ON_CMD – Enable TCXO buffer ▪ PM_TCXO_BUFFER_OFF_CMD – Disable TCXO buffer ▪ PM_TCXO_EMULATE_TCXO_EN_ACTIVE_CMD – This Manual mode command has the same effect as that of driving TCXO_EN pin active in Automatic mode (i.e. it turns TCXO LDO on and after the programmed delay, enables the TCXO buffer) ▪ PM_TCXO_EMULATE_TCXO_EN_INACTIVE_CMD – This Manual mode command has the same effect as that of driving TCXO_EN pin inactive in Automatic mode (i.e. it turns TCXO LDO off and disables the TCXO buffer) ▪ PM_TCXO_XO_GP2_OUT_ENABLE – Enable sine wave output ▪ PM_TCXO_XO_GP2_OUT_DISABLE – Disable sine wave output

Description

Use this function to set the TCXO controller in the following modes:

- Manual mode – User has control over turning TCXO buffer On/Off, enabling/disabling TCXO LDO, or emulating TCXO_EN pin control, which is available in Automatic mode.
- Automatic mode – TCXO is controlled by the TCXO_EN pin, which automatically controls TCXO LDO as well as the buffer.
- Turn TCXO buffer ON/OFF – If Manual mode has been selected, it is used to turn the TCXO buffer On/Off.
- Emulate hardware control of the TCXO_EN pin in the software when Manual mode has been selected.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the cmd parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* User has control over turning TCXO buffer ON/OFF */
   err |= pm_tcxo_cmd(PM_TCXO_BUFFER_ON_CMD);
/* User has control over enabling/disabling TCXO LDO */
   Err |= pm_ldo_control(PM_ON_CMD, PM_TCXO_LDO);
/* User can also choose to emulate TCXO_EN pin control */
/* available in AUTOMATIC mode */
   err |= pm_tcxo_cmd(PM_TCXO_EMULATE_TCXO_EN_ON_CMD);
```

12.4 pm_tcxo2_cmd()

This function sets the second TCXO controller in the following modes: manual, automatic, TCXO buffer on/off, and emulation of the TCXO_EN active/inactive command.

Parameters

```
pm_err_flag_type pm_tcxo2_cmd
(
    pm_tcxo_cmd_type cmd
)
```

→	cmd	pm_tcxo_cmd_type: ▪ PM_TCXO_MANUAL_MODE_CMD – Configure the controller to be operable in Manual mode wherein the switching is done in software using APIs ▪ PM_TCXO_AUTOMATIC_MODE_CMD – Configure the controller to be operable in Automatic mode wherein the switching is done in hardware using TCXO_EN pin ▪ PM_TCXO_BUFFER_ON_CMD – Enable TCXO buffer ▪ PM_TCXO_BUFFER_OFF_CMD – Disable TCXO buffer ▪ PM_TCXO_EMULATE_TCXO_EN_ACTIVE_CMD – This Manual mode command has the same effect as that of driving TCXO_EN pin active in Automatic mode (i.e. it turns TCXO LDO on and after the programmed delay, enables the TCXO buffer) ▪ PM_TCXO_EMULATE_TCXO_EN_INACTIVE_CMD – This Manual mode command has the same effect as that of driving TCXO_EN pin inactive in Automatic mode (i.e. it turns TCXO LDO off and disables the TCXO buffer) ▪ PM_TCXO_XO_GP2_OUT_ENABLE – Enable sine wave output ▪ PM_TCXO_XO_GP2_OUT_DISABLE – Disable sine wave output

Description

Use this function to set the second TCXO controller in the following modes:

- Manual mode – User has control over turning second TCXO buffer On/Off, enabling/disabling TCXO LDO, or emulating TCXO_EN pin control, which is available in Automatic mode.
- Automatic mode – TCXO is controlled by the TCXO_EN pin, which automatically controls TCXO LDO as well as the buffer.
- Turn TCXO buffer On/Off – If Manual mode has been selected, it is used to turn the TCXO buffer ON/OFF.
- Emulate hardware control of TCXO_EN pin in software when Manual mode has been selected.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the cmd parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* User has control over turning the second TCXO buffer ON/OFF */
   err |= pm_tcxo2_cmd(PM_TCXO_BUFFER_ON_CMD);
/* User has control over enabling/disabling TCXO LDO */
   Err |= pm_ldo_control(PM_ON_CMD, PM_TCXO_LDO);
/* User can also choose to emulate TCXO_EN pin control */
/* available in AUTOMATIC mode */
   err |= pm_tcxo2_cmd(PM_TCXO_EMULATE_TCXO_EN_ON_CMD);
```

12.5 pm_xo_buffer_A0_cmd()

This function allows the user to configure the XO buffers A0.

Parameters

```
pm_err_flag_type pm_xo_buffer_A0_cmd
(
    pm_xo_buffer_cmd_type cmd
)
```

→	cmd	pm_xo_buffer_cmd_type: ▪ PM_XO_BUFFER_MANUAL_MODE_CMD – TCXO Controller is controlled by software. ▪ PM_XO_BUFFER_AUTOMATIC_MODE_CMD – TCXO Controller is controlled by XO_EN3 pin. ▪ PM_XO_BUFFER_EN_SBI_ACTIVE_CMD – this MANUAL mode command has the same effect as that of driving TCXO_EN pin active in AUTOMATIC mode, ▪ i.e. it turns TCXO LDO on and after the programmed delay, enables the TCXO buffer ▪ PM_XO_BUFFER_EN_SBI_INACTIVE_CMD – this MANUAL mode command has the same effect as that of driving TCXO_EN pin inactive in AUTOMATIC mode, ▪ i.e. it turns TCXO LDO off and disables the TCXO buffer ▪ PM_XO_BUFFER_PULL_DOWN_REG_ENABLE – soft pull-down resistor enabled ▪ PM_XO_BUFFER_PULL_DOWN_REG_DISABLE – soft pull-down resistor disabled ▪ PM_XO_BUFFER_SET_POLARITY_ACTIVE_HIGH – Sets the logic polarity for the XO_EN3 pin to high ▪ PM_XO_BUFFER_SET_POLARITY_ACTIVE_LOW – Sets the logic polarity for the XO_EN3 pin to low

Description

Use this function to configure the XO buffers A0:

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the cmd parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init()must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* XO_A0 in SSBI mode, off */
(void)pm_xo_buffer_A0_cmd(PM_XO_BUFFER_MANUAL_MODE_CMD);
(void)pm_xo_buffer_A0_cmd(PM_XO_BUFFER_EN_SBI_INACTIVE_CMD);
```

12.6 pm_xo_buffer_A1_cmd()

This function allows the user to configure the XO buffers A1.

Parameters

```
pm_err_flag_type pm_xo_buffer_A1_cmd
(
    pm_xo_buffer_cmd_type cmd
)
```

→	cmd	pm_xo_buffer_cmd_type: ▪ PM_XO_BUFFER_MANUAL_MODE_CMD – TCXO Controller is controlled by software. ▪ PM_XO_BUFFER_AUTOMATIC_MODE_CMD – TCXO Controller is controlled by XO_EN4 pin. ▪ PM_XO_BUFFER_EN_SBI_ACTIVE_CMD – This MANUAL mode command has the same effect as that of driving TCXO_EN pin active in AUTOMATIC mode, ▪ That is, it turns TCXO LDO on and after the programmed delay, enables the TCXO buffer ▪ PM_XO_BUFFER_EN_SBI_INACTIVE_CMD – This MANUAL mode command has the same effect as that of driving TCXO_EN pin inactive in AUTOMATIC mode, ▪ That is, it turns TCXO LDO off and disables the TCXO buffer ▪ PM_XO_BUFFER_PULL_DOWN_REG_ENABLE – Soft pull-down resistor enabled ▪ PM_XO_BUFFER_PULL_DOWN_REG_DISABLE – Soft pull-down resistor disabled ▪ PM_XO_BUFFER_SET_POLARITY_ACTIVE_HIGH – Sets the logic polarity for the XO_EN4 pin to high ▪ PM_XO_BUFFER_SET_POLARITY_ACTIVE_LOW – Sets the logic polarity for the XO_EN4 pin to low

Description

Use this function to configure the XO buffers A1.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the cmd parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* XO_A0 in SSBI mode, off */
(void)pm_xo_buffer_A1_cmd(PM_XO_BUFFER_MANUAL_MODE_CMD);
(void)pm_xo_buffer_A1_cmd(PM_XO_BUFFER_EN_SBI_INACTIVE_CMD);
```

13 Crystal Oscillator

The crystal oscillator is the primary SLEEP clock source. This signal is typically generated by an external crystal (plus two shunt capacitors) supplemented by a PM8921 inverter and buffer. An external oscillator module can be used instead of a crystal by connecting the module output directly into XTAL_IN and using proper software control. In either case, XTAL_OUT is not a buffered output and is incapable of driving a load. The oscillator will be significantly disrupted if XTAL_OUT is loaded. When using an external oscillator module, the XTAL_OUT pin should be unconnected.

The PM8921 IC includes a circuit that continually monitors the crystal oscillator signal. If the crystal stops oscillating, the PM8921 automatically switches to the RC oscillator and sends an MSM device interrupt. Because the oscillator drives SMPL timers, it operates with supply voltages as low as 1.5 V.

13.1 Crystal Oscillator API

13.1.1 pm_xtal_sleep_osc_cmd_for_device()

This function enables/disables the external sleep crystal oscillator.

Parameters

```
pm_err_flag_type pm_xtal_sleep_osc_cmd_for_device
(
   int                externalResourceIndex,
   pm_switch_cmd_type cmd
)
```

→	externalResourceIndex	int – DEFAULT_XO1 = 0 = Turns on first xtal ▪ PM_XO_2 = 1 = Turns on second xtal
→	cmd	pm_switch_cmd_type: ▪ PM_ON_CMD ▪ PM_OFF_CMD

Description

Use this function to enable or disable the external sleep crystal oscillator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – Processor does not have access to this resource
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – Resource is invalid. Resource index was pass into router that is not defined in the build
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

Side effects

Example

13.1.2 pm_vote_clk_32k_for_device()

This function casts a vote to turn the 32 KHz FM radio clock on or off.

Parameters

```
pm_err_flag_type pm_vote_clk_32k_for_device
(
   int                          externalResourceIndex,
   pm_switch_cmd_type           cmd,
   pm_32k_clk_app_vote_type     vote
)
```

→	externalResourceIndex	Int: ▪ DEFAULT_XO1 = 0 = Turns on first xtal ▪ PM_XO_2 = 1 = Turns on second xtal
→	cmd	pm_switch_cmd_type: ▪ PM_ON_CMD ▪ PM_OFF_CMD
→	vote	pm_32k_clk_app_vote_type: ▪ PM_32K_CLK_BT_APP ▪ PM_32K_CLK_FM_APP ▪ PM_32K_CLK_RESERVED1_APP ▪ PM_32K_CLK_RESERVED2_APP ▪ PM_32K_CLK_INVALID_APP

Description

Use this function to cast a vote to turn the 32 KHz FM radio clock on or off, for a specified clock module in the system. If any application has voted to turn on, the clock will be on. Only when all applications that have voted on subsequently vote to turn off will the clock be off.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – Processor does not have access to this resource
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – Resource is invalid. Resource index was pass into router that is not defined in the build
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

Side effects

Example

```
FM radio application votes to turn the clock on:
 pm_vote_clk_32k(ON, PM_32K_CLK_FM_APP);

 Bluetooth application votes to turn the clock off:
 pm_vote_clk_32k(OFF, PM_32K_CLK_BT_APP);
```

13.1.3 pm_clk_select_rc_or_xo()

This function selects internal inaccurate RC or XO as a clock source.

Parameters

```
pm_err_flag_type pm_clk_select_rc_or_xo
(
   int      externalResourceIndex,
   boolean  xo
)
```

→	externalResourceIndex	int: ▪ DEFAULT_XO1 = 0 = Turns on first xtal ▪ PM_XO_2 = 1 = Turns on second xtal
→	xo	boolean: ▪ TCXO = 0, ▪ RC = 1

Description

Use this function to enable or disable the external sleep crystal oscillator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__TARGET_PROCESSOR_CAN_NOT_ACCESS_RESOURCE – Processor does not have access to this resource
- PM_ERR_FLAG__INVALID_RESOURCE_ACCESS_ATTEMPTED – Resource is invalid. Resource index was pass into router that is not defined in the build
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

Side effects

Example

13.1.4 pm_start_up_abort_timer_switch()

This function enables/disables the PMIC's start-up abort feature.

Parameters

```
pm_err_flag_type pm_start_up_abort_timer_switch
(
  pm_switch_cmd_type OnOff
)
```

→	OnOff	pm_switch_cmd_type: ▪ PM_ON_CMD ▪ PM_OFF_CMD

Description

Use this function to enable or disable the PMIC's start-up abort feature. The start-up abort feature allows the PMIC to give up trying to power on the phone if it is not able to do so (the MSM device will not bring PS_HOLD high) after one to two seconds.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the OnOff parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
err |= pm_start_up_abort_timer_switch(PM_ON_CMD);
```

13.2 XO API

13.2.1 pm_xo_enable()

This function enables/disables the 19.2 MHz crystal oscillator.

Parameters

```
pm_err_flag_type pm_xo_enable
(
  boolean xo_enable
)
```

→	xo_enable	▪ TRUE – Enable the 19.2 MHz crystal oscillator ▪ FALSE – Disable the 19.2 MHz crystal oscillator

Description

Use this function to enable or disable the 19.2 MHz crystal oscillator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the xo_enable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable the 19.2 MHz clock. */
err |= pm_xo_enable(FALSE);
```

13.2.2 pm_xo_sel_alt_sleep_clk_src()

This function selects an alternate sleep clock or uses the 19.2 MHz crystal oscillator to generate a sleep clock.

Parameters

```
pm_err_flag_type pm_xo_sel_alt_sleep_clk_src
(
  pm_xo_sleep_clock_src_type alt_clk_src
)
```

→	alt_clk_src	pm_xo_sleep_clock_src_type: ▪ PM_XO_32KHZ_SLEEP_CLOCK_XTAL_OSC ▪ PM_XO_19_2MHZ_XTAL_OSC

Description

Use this function to select an alternate sleep clock or use the 19.2 MHz crystal oscillator to generate a sleep clock.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the alt_clk_src parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Select the clock source 19.2 MHz to derive 32.768 sleep clock */
err |= pm_xo_sel_alt_sleep_clk_src(PM_XO_19_2MHZ_XTAL_OSC);
```

13.2.3 pm_xo_boost_enable()

This function enables/disables the boost of the XO core AGC gain.

Parameters

```
pm_err_flag_type pm_xo_boost_enable
(
   boolean boost_enable
)
```

→	boost_enable	▪ TRUE – Boost the XO core AGC gain ▪ FALSE – Do not boost the XO core AGC gain

Description

Use this function to enable or disable the boost of the XO core AGC gain.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the rfbuf_enable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* don’t boost the XO core AGC gain */
err |= pm_xo_boost_enable(FALSE);
```

13.2.4 pm_xo_set_xo_trim()

This function trims the 19.2 MHz XO load capacitances.

Parameters

```
pm_err_flag_type pm_xo_set_xo_trim
(
    uint8 trim_value
)
```

→	trim_value	Trim value to be written in the range of [0, 63]

Description

Use this function to trim the 19.2 MHz XO load capacitances.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the trim_value parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* trims the 19.2 MHz XO load capacitances */
err |= pm_xo_set_xo_trim(24);
```

13.2.5 pm_xo_get_xo_trim()

This function reads the raw trim value of the load capacitances for the 19.2 MHz XO.

Parameters

```
uint8 pm_xo_get_xo_trim
(
  void
)
```

Description

Use this function to get the raw trim value of the load capacitances for the 19.2 MHz XO.

Return value

This function returns the trim value (uint8):

- Valid value in the range of [0, 63]
- Value of 0xFF is an error code that indicates a read failure

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* get the load capacitances of the 19.2 MHz XO. */
uint 8  trim_value = 0;
trim_value = pm_xo_get_xo_trim();
```

13.2.6 pm_xo_get_cap_step_size()

This function gets the step size of the load capacitances in femto (10^-15) Farads for the 19.2 MHz XO.

Parameters

```
uint8 pm_xo_get_cap_step_size
(
  void
)
```

Description

Use this function to get the step size of the load capacitances in femto (10^-15) Farads for the 19.2 MHz XO.

Return value

This function returns the load capacitance step size (uint8):

Step size of load capacitance in femto (10^-15) Farads

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* get the load capacitances of the 19.2 MHz XO. */
uint 8  cap_step_size_value = 0;
cap_step_size_value = pm_xo_get_cap_step_size();
```

13.2.7 pm_xo_get_alt_sleep_clk_src()

This function returns the sleep clock source.

Parameters

```
pm_err_flag_type pm_xo_get_alt_sleep_clk_src
(
    pm_xo_sleep_clock_src_type *alt_clk_src
)
```

→	alt_clk_src	The sleep clock source is returned in this parameter ▪ PM_XO_32KHZ_SLEEP_CLOCK_XTAL_OSC – Sleep Clock Source is external 32K XTAL ▪ PM_XO_19_2MHZ_XTAL_OSC – Sleep Clock Source is 19.2MHz XO/586

Description

Use this API to get the sleep clock source. The sleep clock source may be selected by using the pm_xo_sel_alt_sleep_clk_src() API. This API returns the current PMIC setting.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Get sleep clock source configuration*/
pm_xo_sleep_clock_src_type sleep_clock_src = PM_XO_INVALID_SLEEP_SRC;

(void)pm_xo_get_alt_sleep_clk_src(&sleep_clock_src);

switch(sleep_clock_src)
{
    case PM_XO_32KHZ_SLEEP_CLOCK_XTAL_OSC:
        …
        break;
    case PM_XO_19_2MHZ_XTAL_OSC:
        …
        break;
    default:
        //Print error message
        …
}
```

13.2.8 pm_xo_force_xo_core_enable()

This function forces the XO core on.

Parameters

```
pm_err_flag_type pm_xo_force_xo_core_enable
(
    boolean xo_core_enable
)
```

→	xo_core_enable	boolean ▪ TRUE – Force the XO core on ▪ FALSE – Leave the XO core off

Description

Use this API to forces XO core always ON if pm_xo_enable(TRUE) has been called

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Don’t have XO core ON*/
(void) pm_xo_force_xo_core_enable(FALSE);
```

13.2.9 pm_xo_power_mode_set()

This function sets the XO buffer drive strength

Parameters

```
pm_err_flag_type pm_xo_power_mode_set
(
    pm_xo_power_mode_type xo_power_mode
)
```

→	xo_power_mode	pm_xo_power_mode_type ▪ PM_XO_POWER_MODE_LOW ▪ PM_XO_POWER_MODE_NORMAL ▪ PM_XO_POWER_MODE_MEDIUM ▪ PM_XO_POWER_MODE_HIGH

Description

Use this API to set the XO buffer drive strength.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid xo_power_mode parameter

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Set XO Strength to High*/
errFlag = pm_xo_power_mode_set(PM_XO_POWER_MODE_HIGH);
```

13.2.10 pm_xo_clk_div_set()

This function sets XO clock div.

Parameters

```
pm_err_flag_type pm_xo_clk_div_set
(
    pm_xo_clk_div_type xo_clk_div
)
```

→	xo_clk_div	pm_xo_clk_div_type ▪ PM_XO_NO_CLK ▪ PM_XO_CLKDIV_8 – Div = 8 ▪ PM_XO_CLKDIV_16 – Div = 16 ▪ PM_XO_CLK_TEST – Div = 1

Description

Use this API to set the XO clock div.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid xo_power_mode parameter

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Set XO Div = 16*/
errFlag = pm_xo_clk_div_set(PM_XO_CLKDIV_16);
```

13.3 XOADC API

13.3.1 pm_xoadc_reset_filter()

This function resets all the registers in the digital filter.

Parameters

```
pm_err_flag_type pm_xoadc_reset_filter
(
  void
)
```

Description

Use this function to reset all the registers in the digital filter.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* reset XOADC digital filter. */
err |= pm_xoadc_reset_filter();
```

13.3.2 pm_xoadc_set_input()

This function selects the input to use when performing the XO ADC conversion.

Parameters

```
pm_err_flag_type pm_xoadc_set_input
(
  pm_xoadc_input_select_type xoadcInput
)
```

→	xoadcInput	pm_xoadc_input_select_type: ▪ PM_XOADC_INPUT_PMIC – Selects PMIC_IN. Used for the PMIC AMUX channels. ▪ PM_XOADC_INPUT_XO – Selects XO_IN. Used for the XO THERM channel. ▪ PM_XOADC_INPUT_VREFP – Shorts the XOADC input to VREFP for gain error measurement. ▪ PM_XOADC_INPUT_VREFN – Shorts the XOADC input to VREFN for offset error measurement.

Description

Use this function to select the input to use when performing the XO ADC conversion.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the xoadcInput parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Select the XO therm for ADC  */
err |= pm_xoadc_set_input(PM_XOADC_INPUT_XO);
```

13.3.3 pm_xoadc_set_decimation_ratio()

This function sets the XOADC decimation ratio.

Parameters

```
pm_err_flag_type pm_xoadc_set_decimation_ratio
(
  pm_xoadc_decimation_ratio_type decimationRatio
)
```

→	decimationRatio	pm_xoadc_decimation_ratio_type: ▪ PM_XOADC_DECIMATION_RATIO_1K ▪ PM_XOADC_DECIMATION_RATIO_2K ▪ PM_XOADC_DECIMATION_RATIO_4K ▪ PM_XOADC_DECIMATION_RATIO_16K ▪ PM_XOADC_DECIMATION_RATIO_32K ▪ PM_XOADC_DECIMATION_RATIO_64K ▪ PM_XOADC_DECIMATION_RATIO_1M

Description

Use this function to set the XOADC decimation ratio.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the decimationRatio parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the XOADC decimation to 16K */
err |= pm_xoadc_set_decimation_ratio(PM_XOADC_DECIMATION_RATIO_16K);
```

13.3.4 pm_xoadc_set_conversion_rate()

This function sets the XOADC clock rate.

Parameters

```
pm_err_flag_type pm_xoadc_set_conversion_rate
(
  pm_xoadc_conversion_rate_type conversionRate
)
```

→	conversionRate	pm_xoadc_conversion_rate_type: ▪ PM_XOADC_CONVERSION_RATE_TCXO_DIV_8 ▪ PM_XOADC_CONVERSION_RATE_TCXO_DIV_4 ▪ PM_XOADC_CONVERSION_RATE_TCXO_DIV_2 ▪ PM_XOADC_CONVERSION_RATE_TCXO_NO_DIV

Description

Use this function to set the XOADC clock rate.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the conversionRate parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* set the XOADC clock conversion rate to divide by 4 */
err |= pm_xoadc_set_conversion_rate(PM_XOADC_CONVERSION_RATE_TCXO_DIV_4);
```

13.3.5 pm_xoadc_set_filter_config()

This function sets the XOADC decimation filter configuration.

Parameters

```
pm_err_flag_type pm_xoadc_set_filter_config
(
  pm_xoadc_filter_config_type filterConfig
)
```

→	filterConfig	pm_xoadc_filter_config_type: ▪ PM_XOADC_FILTER_DISABLE_SINC1_DISABLE_SINC2 ▪ PM_XOADC_FILTER_ENABLE_SINC1_DISABLE_SINC2 ▪ PM_XOADC_FILTER_ENABLE_SINC1_ENABLE_SINC2

Description

Use this function to set the XOADC decimation filter configuration.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the filterConfig parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* set the XOADC decimation filter configuration to: enable sync1 and sync2
disable*/
err |= pm_xoadc_set_filter_config(
                      PM_XOADC_FILTER_ENABLE_SINC1_DISABLE_SINC2
                     );
```

13.3.6 pm_xoadc_enable_modulator()

This function enables the modulator and starts the XOADC conversion.

Parameters

```
uint8 pm_xoadc_enable_modulator
(
  pm_xoadc_enable_type xoadcEnable
)
```

→	xoadcEnable	pm_xoadc_enable_type: ▪ PM_XOADC_DISABLE_CMD – Disable the modulator ▪ PM_XOADC_ENABLE_CMD – Enable the modulator

Description

Use this function to enable the modulator and start the XOADC conversion. Until the modulator is disabled, conversions continue to take place and the results accumulate in the data register.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the xoadcEnable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* disable the modulator. */
err |= pm_xoadc_enable_modulator(PM_XOADC_DISABLE_CMD);
```

13.3.7 pm_xoadc_read_data()

This function reads the PMIC XOADC data registers and returns the value.

Parameters

```
uint8 pm_xoadc_read_data
(
  uint32 *xoadc_data
)
```

→	*xoadc_data	uint32 range

Description

Use this function to read the PMIC XOADC data registers and returns the value (24-bit conversion value in registers XOADC_DATA0, XOADC_DATA1, and XOADC_DATA2).

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the xoadc_data parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* get the xoadc value */
err |= = pm_xoadc_read_data(&xoadc_data);
```

13.3.8 pm_xoadc_set_enable()

This function enables the XOADC device.

Parameters

```
pm_err_flag_type pm_xoadc_set_enable
(
    pm_xoadc_enable_type xoadcEnable
)
```

→	xoadcEnable	pm_xoadc_enable_type: ▪ PM_XOADC_DISABLE_CMD ▪ PM_XOADC_ENABLE_CMD

Description

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS function completed successfully
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED XOADC not available on this PMIC
- PM_ERR_FLAG__SBI_OPT_ERR error communicating with the PMIC

Dependencies

Side effects

Example

13.3.9 pm_xoadc_request_conversion()

This function sets the conversion request bit.

Parameters

```
pm_err_flag_type pm_xoadc_request_conversion
(
    void
)
```

Description

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Function completed successfully
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – XOADC not available on this PMIC
- PM_ERR_FLAG__SBI_OPT_ERR – Error communicating with the PMIC

Dependencies

Side effects

Example

13.3.10 pm_xoadc_get_conversion_status()

This function gets the end-of-conversion status.

Parameters

```
pm_err_flag_type pm_xoadc_get_conversion_status
(
    pm_xoadc_conversion_status_type *convStatus
)
```

→	*convStatus	Pointer of pm_xoadc_conversion_status_type: ▪ PM_XOADC_CONVERSION_STATUS_PENDING ▪ PM_XOADC_CONVERSION_STATUS_COMPLETE

Description

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Function completed successfully
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – XOADC not available on this PMIC
- PM_ERR_FLAG__SBI_OPT_ERR – Error communicating with the PMIC

Dependencies

Side effects

Example

13.3.11 pm_xoadc_config_mux()

This function configures the AMUX mux channel selection.

Parameters

```
pm_err_flag_type pm_xoadc_config_mux
(
   pm_adc_muxsel_type muxSel
)
```

→	muxSel	pm_adc_muxsel_type: ▪ PM_ADC_MUXSEL0 – uses 2/3 pre-scaler ▪ PM_ADC_MUXSEL1 -uses 1/2 pre-scaler ▪ PM_ADC_MUXSEL2 – uses 1/8 pre-scaler ▪ PM_ADC_MUXSEL3 – uses NO pre-scaler ▪ PM_ADC_MUXSEL4 – uses 1/2 pre-scaler ▪ PM_ADC_MUXSEL5 – uses NO pre-scaler ▪ PM_ADC_MUXSEL6 – uses NO pre-scaler ▪ PM_ADC_MUXSEL7 – uses NO pre-scaler ▪ PM_ADC_MUXSEL8 – uses NO pre-scaler ▪ PM_ADC_MUXSEL9 – uses NO pre-scaler ▪ PM_ADC_MUXSEL10 – uses 2/5 pre-scaler ▪ PM_ADC_MUXSEL11 – uses NO pre-scaler ▪ PM_ADC_MUXSEL12 – uses NO pre-scaler ▪ PM_ADC_MUXSEL13 – uses NO pre-scaler ▪ PM_ADC_MUXSEL14 – uses NO pre-scaler

Description

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Function completed successfully
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – XOADC not available on this PMIC
- PM_ERR_FLAG__SBI_OPT_ERR – Error communicating with the PMIC

Dependencies

Side effects

Example

14 Over-Temperature Protection API

The PM8921 IC provides over-temperature protection in four stages, depending on the level of urgency. If the die temperature exceeds a threshold, the IC sends an interrupt to the MSM ASIC without shutting down, shuts off nonessential functions (such as LED drivers, speaker drivers, etc.), or shuts off the phone completely. Temperature hysteresis is incorporated so that the die temperature must cool significantly before the device can be powered on again.

Temperature protection stages:

- Stage 0
  - Normal condition – $T \leq 105C$
  - No action
- Stage 1
  - Over-temperature condition – $105C < T \leq 120C$
  - Sends an over-temperature warning interrupt to modem IC without shutting down any PM circuits.
- Stage 2
  - Over-temperature condition – $120C < T \leq 140C$
  - An interrupt is sent to the modem IC and high-current drivers(LED drivers, backlight drivers, etc._ are shut down
- Stage 3
  - Over-temperature condition – $T > 140C$
  - An interrupt is sent to the modem IC and PM functions are completely shut down.

14.1 pm_itemp_get_stage()

This function returns the current PM8921 over-temperature protection stage.

Parameters

```
pm_err_flag_type pm_itemp_get_stage
(
  pm_item_stage_type *itemp_stage
)
```

←	*itemp_stage	pm_item_stage_type: ▪ PM_ITEMP_STAGE0 (T ≤ 105C) ▪ PM_ITEMP_STAGE1 (105C < T ≤ 120C) ▪ PM_ITEMP_STAGE2 (120C < T ≤ 140C) ▪ PM_ITEMP_STAGE3 (T > 140C)

Description

Use this function to return the current PM8921 over-temperature protection stage.

NOTE: A change of stage can be detected by enabling or pulling the Temperature Status Changed IRQ (refer to PM_T_STAT_CHANGED_IRQ_HDL and PM_T_STAT_CHANGED_RT_ST).

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the itemp_stage parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC device.

Example

```
pm_item_stage_type *itemp_stage
pm_err_flag_type errFlag = pm_itemp_get_stage(itemp_stage);
```

14.2 pm_itemp_stage_override()

This function overrides automatic shut down of the PM8921 over-temperature protection stage 2 (120C < T ≤ 140C).

Parameters

```
pm_err_flag_type pm_itemp_stage_override
(
  pm_switch_cmd_type  oride_cmd,
  pm_item_oride_type  oride_stage
)
```

→	oride_cmd	pm_switch_cmd_type: ▪ PM_OFF_CMD – Default; do not override automatic shutdown ▪ PM_ON_CMD – Override automatic shutdown
→	oride_stage	Identify the stage to override. pm_item_oride_type: ▪ PM_ITEMP_ORIDE_STAGE2 – Only stage 2 can be overridden

Description

Use this function to override the PM8921 over-temperature protection stage 2 (120C < T ≤ 140C), which shuts down high current drivers (such as speaker and LED drivers) automatically.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the oride_cmd parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the oride_stage parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_itemp_stage_override(PM_ON_CMD, PM_ITEMP_ORIDE_STAGE2);
```

14.3 pm_itemp_thresh_cntrl()

This function controls the temperature threshold.

Parameters

```
pm_err_flag_type pm_itemp_thresh_cntrl
(
   pm_itemp_threshold_type  thresh_value
)
```

→	thresh_value	pm_itemp_threshold_type: ▪ PM_TEMP_THRESH_CTRL0 ▪ PM_TEMP_THRESH_CTRL1 ▪ PM_TEMP_THRESH_CTRL2 ▪ PM_TEMP_THRESH_CTRL3

Description

Use this function to configure temperature threshold.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the thresh_value parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

This function should be called while or after executing the pm_init() function.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Use {105, 125, 145} as pmic temperature threshold */
(void)pm_itemp_thresh_cntrl(PM_TEMP_THRESH_CTRL0);
```

15 Real-Time Clock API

The Real-Time Clock (RTC) module contains a 32-bit counter for time-keeping that is clocked by an internal 1 Hz clock source. This 1 Hz clock source is derived from a frequency divider circuit that divides the 32768 Hz crystal oscillator by an ideal factor of 32768. The frequency divider can be adjusted in 3.05 ppm (parts per million) increments up to the limits of -192 ppm to +195 ppm to account for crystals that deviate slightly from the ideal value of 32768 Hz.

15.1 pm_hal_rtc_start()

This function starts the RTC with the indicated time as its current start-up time.

Parameters

```
pm_err_flag_type pm_hal_rtc_start
(
    pm_hal_rtc_time_type *start_time_ptr
)
```

→	*start_time_ptr	Pointer to a valid non-NULL RTC time structure that contains any 32-bit number indicating the number of seconds elapsed from a known point in the past

Description

The RTC must be started before it can be used for functions such as reading time, setting alarms, etc. The RTC keeps time using an unformatted 32-bit number that is incremented every second. Higher level logic must be used to interpret this 32-bit number as a specific moment in the Julian calendar.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – NULL pointer provided for the start_time_ptr parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_hal_rtc_time_type  start_time;
/*
Assume that 200 maps to some reference time in Julian calendar, say
00:00:00 on Jan 01, 2003.
*/
start_time.sec = 200;
if (PM_ERR_FLAG__SUCCESS != pm_hal_rtc_start(&start_time))
{
   MSG_ERROR("Could not start RTC",0,0,0);
}
```

15.2 pm_hal_rtc_stop()

This function stops the RTC.

Parameters

```
pm_err_flag_type pm_hal_rtc_stop(void)
```

Description

Use this function to halt the RTC if it was running at the time this function was invoked. It is safe to invoke this function even if the RTC is already halted.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Stop RTC */
if (PM_ERR_FLAG__SUCCESS != pm_hal_rtc_stop())
{
   MSG_ERROR("Could not halt RTC",0,0,0);
}
```

15.3 pm_hal_rtc_get_time()

This function returns the current RTC time.

Parameters

```
pm_err_flag_type pm_hal_rtc_get_time
(
    pm_hal_rtc_time_type *time_ptr
)
```

←	*time_ptr	Pointer to a valid non-NULL RTC time structure used for returning the current time of the real-time clock

Description

Use this function to return the current time (as an unformatted 32-bit number) of the RTC if it was running at the time this function was invoked. If the RTC was not running, this function returns an error.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__RTC_HALTED – RTC is currently halted
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – NULL pointer provided for the time_ptr parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- pm_hal_rtc_start()

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_hal_rtc_time_type  current_time;
/*Get current time from RTC */
if (PM_ERR_FLAG__SUCCESS != pm_hal_rtc_get_time(&current_time))
{
   MSG_ERROR("Could not get current time",0,0,0);
}
```

15.4 pm_hal_rtc_set_time_adjust()

This function compensates for an external sleep crystal oscillation frequency that is deviating slightly from the 32768 Hz frequency.

Parameters

```
pm_err_flag_type pm_hal_rtc_set_time_adjust
(
    uint8 time_adjust
)
```

→	time_adjust	Frequency adjustment factor and its corresponding parts per million (ppm). Valid inputs: ▪ 0 to 63 – Compensates for a slower crystal oscillator ▪ 64 – No compensation ▪ 65 to 127 – Compensates for a faster crystal oscillator

Description

Use this function to set the time adjustment correction factor for an RTC crystal oscillator that is slightly off the 32768 Hz frequency. Every tenth second, the frequency divider, which divides the external sleep crystal oscillation for generating 1 Hz pulses for the RTC, is switched from the nominal 32768 Hz to (32768 – 64 + time_adjust) Hz.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the time_adjust parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Compensate for a crystal that is slower than 32768 Hz by 6.10 ppm which
maps to value 62 for time_adjust */
if (PM_ERR_FLAG__SUCCESS != pm_hal_rtc_set_time_adjust(62))
{
   MSG_ERROR("Could not set sleep xtal compensation factor",0,0,0);
}
```

15.5 pm_hal_rtc_get_time_adjust()

This function returns the current external sleep crystal compensation factor in use.

Parameters

```
pm_err_flag_type pm_hal_rtc_get_time_adjust
(
    unit8 *time_adjust_ptr
)
```

←	*time_adjust_ptr	A non-NULL pointer to an 8-bit storage for returning the frequency adjustment factor. Valid inputs: ▪ 0 to 63 – Compensates for a slower crystal oscillator ▪ 64 – No compensation ▪ 65 to 127 – Compensates for a faster crystal oscillator

Description

Use this function to return the current frequency correction factor used by the RTC to compensate for a crystal oscillator that is slightly off the 32768 Hz frequency. Every tenth second, the frequency divider, which divides the external sleep crystal oscillation for generating 1 Hz pulses for RTC, is switched from the nominal 32768 to (32768 – 64 + value indicated by time_adjust_ptr).

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – NULL pointer provided for the time_adjust_ptr parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
uint8  current_freq_adjust;
/*Get the current frequency adjustment factor in use */
if (PM_ERR_FLAG__SUCCESS
       != pm_hal_rtc_get_time_adjust(&current_freq_adjust))
{
   MSG_ERROR("Couldn’t get current sleep xtal compensation factor",0,0,0);
}
```

15.6 pm_hal_rtc_enable_alarm()

This function starts a one-shot alarm.

Parameters

```
pm_err_flag_type pm_hal_rtc_enable_alarm
(
   pm_hal_rtc_alarm      what_alarm,
   pm_hal_rtc_time_type *trigger_time_ptr
)
```

→	what_alarm	Indicates the alarm to be turned on; currently supported alarm: ▪ PM_HAL_RTC_ALARM_1
→	*trigger_time_ptr	Pointer to the RTC time when the alarm should go off

Description

Use this function to enable an alarm to go off at the specified time. The RTC signals the expiry of the alarm by raising the PM_RTC_ALRM_IRQ_HDL IRQ with the interrupt manager, if there is an ISR currently associated with that IRQ.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Unknown alarm specified
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – NULL pointer provided for the trigger_time_ptr parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- pm_hal_rtc_start()

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*ISR for handling RTC alarm IRQ */
void rtc_alarm_isr(void)
{
   /*clear IRQ only after the alarm indication has been turned off */
   (void) pm_hal_rtc_disable_alarm(PM_HAL_RTC_ALARM_1);
   (void) pm_clear_irq(PM_RTC_ALRM_IRQ_HDL);

   MSG_HIGH("RTC alarm triggered!", 0, 0, 0);
}

/* Set the current time to 3600 secs (1 hr) from some known point in
history, say 00:00:00 on Jan 01, 2003. That is, the current time is
01:00:00 on Jan 01, 2003. */
pm_hal_rtc_time_type  start_time;
start_time.sec = 3600;
(void) pm_hal_rtc_start(&start_time);
/* Set the alarm to go off at ½ hr from now at 01:30:00 on Jan 01, 2003 */
pm_hal_rtc_time_type  alarm_time;
alarm_time.sec = 3600 + 1800;
(void) pm_hal_rtc_enable_alarm(PM_HAL_RTC_ALARM_1, &alarm_time);
(void) pm_set_irq_handle(PM_RTC_ALRM_IRQ_HDL, rtc_alarm_isr);

/* rtc_alarm_isr() will be invoked when RTC alarm expires */
```

15.7 pm_hal_rtc_disable_alarm()

This function disables the alarm.

Parameters

```
pm_err_flag_type pm_hal_rtc_disable_alarm
(
    pm_hal_rtc_alarm what_alarm
)
```

→	what_alarm	Indicates the alarm to be turned off; currently supported alarm: ▪ PM_HAL_RTC_ALARM_1

Description

Use this function to disable the specified alarm so that it does not go off in the future. If the alarm has already triggered by the time this function is called, it also clears the alarm trigger record so that the master RTC alarm PM_RTC_ALRM_IRQ_HDL IRQ can be cleared afterward.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Unknown alarm specified
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Referring to our example for pm_hal_rtc_enable_alarm(), let us assume
that some time before the alarm is set to trigger that we would like to
cancel it */
(void) pm_hal_rtc_disable_alarm(PM_HAL_RTC_ALARM_1);
```

15.8 pm_hal_rtc_get_alarm_time()

This function gets the alarm time.

Parameters

```
pm_err_flag_type pm_hal_rtc_get_alarm_time
(
   pm_hal_rtc_alarm       what_alarm,
   pm_hal_rtc_time_type   *alarm_time_ptr
)
```

→	what_alarm	Indicates the alarm to be read; currently supported alarm: ▪ PM_HAL_RTC_ALARM_1
←	*alarm_time_ptr	Pointer to the RTC time structure that is used by this function to return the time at which the specified alarm has been programmed to trigger

Description

Use this function to get the time at which the specified alarm has been programmed to go off.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Unknown alarm specified
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – NULL pointer provided for the alarm_time_ptr parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
pm_hal_rtc_time_type  current_alarm_time;
if (PM_ERR_FLAG__SUCCESS
     != pm_hal_rtc_get_alarm_time(PM_HAL_RTC_ALARM_1, &current_alarm_time))
{
   MSG_ERROR("Couldn’t get current alarm time",0,0,0);
}
```

15.9 pm_hal_rtc_get_alarm_status()

This function gets the alarm trigger status.

Parameters

```
pm_err_flag_type pm_hal_rtc_get_alarm_status
(
    uint8  *status_ptr
)
```

←	*status_ptr	Pointer to a valid uint8 bitmap used for returning the status information of the various alarms: ▪ Bit 0 (LSB) – PM_HAL_RTC_ALARM_1 ▪ Bits 1 to 7 – Not used

Description

Use this function to get the status of the alarms supported in the hardware. A value of 1 means that the alarm has triggered, and a value of 0 means that the alarm has not triggered.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – NULL pointer provided for the status_ptr parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Get the current status of RTC alarm #1 */
uint8  alarm_status;
if (PM_ERR_FLAG__SUCCESS != pm_hal_rtc_get_alarm_status(&alarm_status))
{
   MSG_ERROR("Could not get current RTC alarm status", 0, 0, 0);
}

if (alarm_status & 0x1)
{
   MSH_HIGH("RTC alarm #1 triggered!", 0, 0, 0);
}
```

16 General Purpose Input Output API

The General Purpose Input Output (GPIO) module is designed to function as a generic interface block between the internal digital block and external pins, which must support external devices.

Two of these blocks can be cascaded to provide level translations between two external devices. The GPIO is similar in functionality to the MPP. The difference between the two is that the MPP can provide analog buffering as well as digital buffering.

16.1 pm_gpio_init()

This function initializes the PMIC GPIO.

Parameters

```
pm_err_flag_type pm_gpio_init
(
    pm_model_type pmic_model
)
```

Description

Use this function to initialize the PMIC GPIO. This function must be called before attempting to use any of the GPIO APIs.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__INVALID_PMIC_MODEL– API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Initialize the GPIO */

pm_model_type pmic_model;
pmic_model = pm_get_pmic_model();
errFlag = pm_gpio_init(pmic_model);
```

16.2 pm_gpio_config_bias_voltage()

This function configures the bias voltage for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_config_bias_voltage
(
    pm_gpio_which_type                  gpio,
    pm_gpio_voltage_source_type         voltage_source
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37

		▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	`voltage_source`	pm_gpio_voltage_source_type: ▪ PM_GPIO_VIN0 – VREG_L6 ▪ PM_GPIO_VIN1 – VREG_GRU ▪ PM_GPIO_VIN2 – VREG_L16 ▪ PM_GPIO_VIN3 – VREG_L10 ▪ PM_GPIO_VIN4 – VREG_GR3 ▪ PM_GPIO_VIN5 – VREG_L8 ▪ PM_GPIO_VIN6 – VPH_PWR ▪ PM_GPIO_VIN7 – VPH_PWR

Description

Use this function to configure the bias voltage for a selected GPIO

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the voltage_source parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure GPIO2 to logic high referencing to an analog (quiet) supply */
errFlag = pm_gpio_config_bias_voltage(PM_GPIO_2, PM_GPIO_VIN4);
```

16.3 pm_gpio_config_digital_input()

This function configures a selected GPIO as a digital input.

Parameters

```
pm_err_flag_type pm_gpio_config_digital_input
(
    pm_gpio_which_type                  gpio,
    pm_gpio_current_source_pulls_type   i_source_pulls,
    pm_gpio_voltage_source_type         voltage_source,
    pm_gpio_out_buffer_drive_strength_type  out_buffer_strength,
    pm_gpio_source_config_type          source
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34

		▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	`i_source_pulls`	pm_gpio_current_source_pulls_type: ▪ PM_GPIO_I_SOURCE_PULL_UP_30uA ▪ PM_GPIO_I_SOURCE_PULL_UP_1_5uA ▪ PM_GPIO_I_SOURCE_PULL_UP_31_5uA ▪ PM_GPIO_I_SOURCE_PULL_UP_1_5uA_PLUS_30uA_BOOST ▪ PM_GPIO_I_SOURCE_PULL_DOWN_10uA ▪ PM_GPIO_I_SOURCE_PULL_NO_PULL
→	`voltage_source`	pm_gpio_voltage_source_type: ▪ PM_GPIO_VIN0 – VREG_L6 ▪ PM_GPIO_VIN1 – VREG_GRU ▪ PM_GPIO_VIN2 – VREG_L16 ▪ PM_GPIO_VIN3 – VREG_L10 ▪ PM_GPIO_VIN4 – VREG_GR3 ▪ PM_GPIO_VIN5 – VREG_L8 ▪ PM_GPIO_VIN6 – VPH_PWR ▪ PM_GPIO_VIN7 – VPH_PWR
→	`out_buffer_strength`	pm_gpio_out_buffer_drive_strength_type: ▪ PM_GPIO_OUT_BUFFER_OFF ▪ PM_GPIO_OUT_BUFFER_HIGH ▪ PM_GPIO_OUT_BUFFER_MEDIUM ▪ PM_GPIO_OUT_BUFFER_LOW
→	`source`	pm_gpio_source_config_type: ▪ PM_GPIO_SOURCE_GND ▪ PM_GPIO_SOURCE_PAIRED_GPIO ▪ PM_GPIO_SOURCE_SPECIAL_FUNCTION1 ▪ PM_GPIO_SOURCE_SPECIAL_FUNCTION2 ▪ PM_GPIO_SOURCE_DTEST1 ▪ PM_GPIO_SOURCE_DTEST2 ▪ PM_GPIO_SOURCE_DTEST3 ▪ PM_GPIO_SOURCE_DTEST4

Description

Use this function to configure selected GPIO as a digital input.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the i_source_pulls parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the voltage_source parameter
- PM_ERR_FLAG__PAR4_OUT_OF_RANGE – Invalid value for the out_buffer_strength parameter
- PM_ERR_FLAG__PAR5_OUT_OF_RANGE – Invalid value for the source parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Keypad scan module require GPIO to drive and sense the keypad.
 * Keypad drive signal should be configured as open_drain output with low
 * drive strength.
 * Keypad sense module should be configured as input with 1.5uA pull up
 * +30uA boost.
 * e.g Reference voltage VIN2 for both drive and sense lines.
 * Configure GPIO2 as a sense line.
 */
errFlag = pm_gpio_config_digital_input(PM_GPIO_2,
                  PM_GPIO_I_SOURCE_PULL_UP_1_5uA_PLUS_30uA_BOOST,
                                       PM_GPIO_VIN2,
                                       PM_GPIO_OUT_BUFFER_OFF,
                                       PM_GPIO_SOURCE_GND
                                      );
```

16.4 pm_gpio_config_digital_output()

This function configures a selected GPIO as a digital output.

Parameters

```
pm_err_flag_type pm_gpio_config_digital_output
(
    pm_gpio_which_type                  gpio,
    pm_gpio_out_buffer_config_type      out_buffer_config,
    pm_gpio_voltage_source_type         voltage_source,
    pm_gpio_source_config_type          source,
    pm_gpio_out_buffer_drive_strength_type out_buffer_strength,
    boolean                             out_inversion
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32

		▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	`out_buffer_config`	pm_gpio_out_buffer_config_type: ▪ PM_GPIO_I_SOURCE_PULL_UP_30uA ▪ PM_GPIO_I_SOURCE_PULL_UP_1_5uA ▪ PM_GPIO_I_SOURCE_PULL_UP_31_5uA ▪ PM_GPIO_I_SOURCE_PULL_UP_1_5uA_PLUS_30uA_BOOST ▪ PM_GPIO_I_SOURCE_PULL_DOWN_10uA ▪ PM_GPIO_I_SOURCE_PULL_NO_PULL
→	`voltage_source`	pm_gpio_voltage_source_type: ▪ PM_GPIO_VIN0 – VREG_L6 ▪ PM_GPIO_VIN1 – VREG_GRU ▪ PM_GPIO_VIN2 – VREG_L16 ▪ PM_GPIO_VIN3 – VREG_L10 ▪ PM_GPIO_VIN4 – VREG_GR3 ▪ PM_GPIO_VIN5 – VREG_L8 ▪ PM_GPIO_VIN6 – VPH_PWR ▪ PM_GPIO_VIN7 – VPH_PWR
→	`source`	pm_gpio_source_config_type: ▪ PM_GPIO_SOURCE_GND ▪ PM_GPIO_SOURCE_PAIRED_GPIO ▪ PM_GPIO_SOURCE_SPECIAL_FUNCTION1 ▪ PM_GPIO_SOURCE_SPECIAL_FUNCTION2 ▪ PM_GPIO_SOURCE_DTEST1 ▪ PM_GPIO_SOURCE_DTEST2 ▪ PM_GPIO_SOURCE_DTEST3 ▪ PM_GPIO_SOURCE_DTEST4
→	`out_buffer_strength`	pm_gpio_out_buffer_drive_strength_type: ▪ PM_GPIO_OUT_BUFFER_OFF ▪ PM_GPIO_OUT_BUFFER_HIGH ▪ PM_GPIO_OUT_BUFFER_MEDIUM ▪ PM_GPIO_OUT_BUFFER_LOW
→	`out_inversion`	Invert the output of Ext_Pi: ▪ TRUE – Invert the output ▪ FALSE – Do not to invert the output

Description

Use this function to configure a selected GPIO as a digital output.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the out_buffer_config parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the voltage_source parameter
- PM_ERR_FLAG__PAR4_OUT_OF_RANGE – Invalid value for the source parameter
- PM_ERR_FLAG__PAR5_OUT_OF_RANGE – Invalid value for the out_buffer_strength parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Keypad scan module require GPIO to drive and sense the keypad.
 * Keypad drive signal should be configured as open_drain output with low
 * drive strength.
 * Keypad sense module should be configured as input with 1.5uA pull up
 * +30uA boost.
 * e.g Reference voltage VIN2 for both drive and sense lines.
 * Configure GPIO2 as a sense line.
 */
errFlag = pm_gpio_config_digital_output(PM_GPIO_2,
                          PM_GPIO_OUT_BUFFER_CONFIG_OPEN_DRAIN,
                                    PM_GPIO_VIN2,
                                    PM_GPIO_SOURCE_SPECIAL_FUNCTION1,
                                    PM_GPIO_OUT_BUFFER_LOW,
                                    FALSE
                                   );
```

16.5 pm_gpio_set_voltage_source()

This function sets the voltage source for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_voltage_source
(
    pm_gpio_which_type                gpio,
    pm_gpio_voltage_source_type       voltage_source
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44

→	voltage_source	pm_gpio_voltage_source_type: ▪ PM_GPIO_VIN0 – VREG_L6 ▪ PM_GPIO_VIN1 – VREG_GRU ▪ PM_GPIO_VIN2 – VREG_L16 ▪ PM_GPIO_VIN3 – VREG_L10 ▪ PM_GPIO_VIN4 – VREG_GR3 ▪ PM_GPIO_VIN5 – VREG_L8 ▪ PM_GPIO_VIN6 – VPH_PWR ▪ PM_GPIO_VIN7 – VPH_PWR

Description

Use this function to set the voltage source for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the voltage_source parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set GPIO2 to an analog voltage source */
errFlag = pm_gpio_set_voltage_source(PM_GPIO_2, PM_GPIO_VIN2);
```

16.6 pm_gpio_config_mode_selection()

This function enables/disables the mode selection for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_config_mode_selection
(
    pm_gpio_which_type                gpio,
    boolean                           enable_disable
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	enable_disable	▪ TRUE – Enable ▪ FALSE – Disable

Description

Use this function to enable or disable the mode selection for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the enable_disable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable the mode selection for the GPIO2 */
errFlag = pm_gpio_config_mode_selection(PM_GPIO_2, FALSE);
```

16.7 pm_gpio_set_output_buffer_configuration()

This function sets the output buffer configuration for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_output_buffer_configuration
(
    pm_gpio_which_type                gpio,
    pm_gpio_out_buffer_config_type    out_buffer_config
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	out_buffer_config	pm_gpio_out_buffer_config_type: ▪ PM_GPIO_OUT_BUFFER_CONFIG_CMOS ▪ PM_GPIO_OUT_BUFFER_CONFIG_OPEN_DRAIN

Description

Use this function to set the output buffer configuration for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the out_buffer_config parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the out_buffer_config parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set output buffer configuration to CMOS in bias mode for GPIO2. */
errFlag = pm_gpio_set_output_buffer_configuration(PM_GPIO_2,
                                PM_GPIO_OUT_BUFFER_CONFIG_CMOS
                                                  );
```

16.8 pm_gpio_set_inversion_configuration()

This function sets inversion for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_inversion_configuration
(
    pm_gpio_which_type                gpio,
    boolean                           inversion
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	inversion	▪ TRUE – Inversion ▪ FALSE – No inversion

Description

Use this function to set the inversion for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the inversion parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set inversion for GPIO2 */
errFlag = pm_gpio_set_inversion_configuration(PM_GPIO_2, TRUE);
```

16.9 pm_gpio_set_current_source_pulls()

This function sets the current source pulls for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_current_source_pulls
(
    pm_gpio_which_type                gpio,
    pm_gpio_current_source_pulls_type  i_source_pulls
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44

→	i_source_pulls	pm_gpio_current_source_pulls_type: ▪ PM_GPIO_I_SOURCE_PULL_UP_30uA ▪ PM_GPIO_I_SOURCE_PULL_UP_1_5uA ▪ PM_GPIO_I_SOURCE_PULL_UP_31_5uA ▪ PM_GPIO_I_SOURCE_PULL_UP_1_5uA_PLUS_30uA_BOOST ▪ PM_GPIO_I_SOURCE_PULL_DOWN_10uA ▪ PM_GPIO_I_SOURCE_PULL_NO_PULL

Description

Use this function to set the current source pulls for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the i_source_pulls parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set current source pulls to 1_5uA PLUS 30uA_BOOST for GPIO2. */
errFlag = pm_gpio_set_current_source_pulls(PM_GPIO_2,
            PM_GPIO_I_SOURCE_PULL_UP_1_5uA_PLUS_30uA_BOOST);
```

16.10 pm_gpio_set_ext_pin_config()

This function sets the external pin configuration for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_ext_pin_config
(
    pm_gpio_which_type                gpio,
    pm_gpio_ext_pin_config_type       ext_pin_config
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	ext_pin_config	pm_gpio_ext_pin_config_type: ▪ PM_GPIO_EXT_PIN_ENABLE ▪ PM_GPIO_EXT_PIN_DISABLE

Description

Use this function to set the external pin configuration for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the ext_pin_config parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable EXT_IN for GPIO2 */
errFlag = pm_gpio_set_ext_pin_config(PM_GPIO_2, PM_GPIO_EXT_PIN_ENABLE);
```

16.11 pm_gpio_set_output_buffer_drive_strength()

This function sets the output buffer drive strength for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_output_buffer_drive_strength
(
   pm_gpio_which_type                        gpio,
   pm_gpio_out_buffer_drive_strength_type    out_buffer_strength
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15

		▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	`out_buffer_strength`	pm_gpio_out_buffer_drive_strength_type: ▪ PM_GPIO_OUT_BUFFER_OFF ▪ PM_GPIO_OUT_BUFFER_HIGH ▪ PM_GPIO_OUT_BUFFER_MEDIUM ▪ PM_GPIO_OUT_BUFFER_LOW

Description

Use this function to set the output buffer drive strength for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the out_buffer_strength parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set output buffer drive strength for GPIO2 to HIGH. */
errFlag = pm_gpio_set_output_buffer_drive_strength(PM_GPIO_2,
                                                   PM_GPIO_OUT_BUFFER_HIGH
                                                        );
```

16.12 pm_gpio_set_source_configuration()

This function sets the source configuration for a selected GPIO.

Parameters

```
pm_err_flag_type pm_gpio_set_source_configuration
(
    pm_gpio_which_type              gpio,
    pm_gpio_source_config_type      source
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44

→	source	pm_gpio_source_config_type: ▪ PM_GPIO_SOURCE_GND ▪ PM_GPIO_SOURCE_PAIRED_GPIO ▪ PM_GPIO_SOURCE_SPECIAL_FUNCTION1 ▪ PM_GPIO_SOURCE_SPECIAL_FUNCTION2 ▪ PM_GPIO_SOURCE_DTEST1 ▪ PM_GPIO_SOURCE_DTEST2 ▪ PM_GPIO_SOURCE_DTEST3 ▪ PM_GPIO_SOURCE_DTEST4

Description

Use this function to set the source configuration for a selected GPIO.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the source parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* To serve GPIO2 as an output in level translator mode,
 * select "Pair In" as the source.
 */
errFlag = pm_gpio_set_source_configuration(PM_GPIO_2,
                                           PM_GPIO_SOURCE_PAIRED_GPIO);
```

16.13 pm_gpio_set_mux_ctrl()

This function sets GPIO mux control.

Parameters

```
pm_err_flag_type pm_gpio_set_mux_ctrl
(
    pm_gpio_uart_path_type         uart_path
)
```

→	uart_path	pm_gpio_uart_path_type: ▪ PM_UART_MUX_NO ▪ PM_UART_MUX_1 ▪ PM_UART_MUX_2 ▪ PM_UART_MUX_3

Description

Use this function to set GPIO mux control.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the uart_path parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set GPIO MUX to MUX 2 */
errFlag = pm_gpio_set_mux_ctrl(PM_UART_MUX_2);
```

16.14 pm_gpio_status_get()

This function gets GPIO status.

Parameters

```
pm_err_flag_type pm_gpio_status_get
(
   pm_gpio_which_type gpio,
   pm_gpio_status_type *gpio_status
)
```

→	gpio	pm_gpio_which_type: ▪ PM_GPIO_1 ▪ PM_GPIO_2 ▪ PM_GPIO_3 ▪ PM_GPIO_4 ▪ PM_GPIO_5 ▪ PM_GPIO_6 ▪ PM_GPIO_7 ▪ PM_GPIO_8 ▪ PM_GPIO_9 ▪ PM_GPIO_10 ▪ PM_GPIO_11 ▪ PM_GPIO_12 ▪ PM_GPIO_13 ▪ PM_GPIO_14 ▪ PM_GPIO_15 ▪ PM_GPIO_16 ▪ PM_GPIO_17 ▪ PM_GPIO_18 ▪ PM_GPIO_19 ▪ PM_GPIO_20 ▪ PM_GPIO_21 ▪ PM_GPIO_22 ▪ PM_GPIO_23 ▪ PM_GPIO_24 ▪ PM_GPIO_25 ▪ PM_GPIO_26 ▪ PM_GPIO_27 ▪ PM_GPIO_28 ▪ PM_GPIO_29 ▪ PM_GPIO_30 ▪ PM_GPIO_31 ▪ PM_GPIO_32 ▪ PM_GPIO_33 ▪ PM_GPIO_34 ▪ PM_GPIO_35 ▪ PM_GPIO_36 ▪ PM_GPIO_37 ▪ PM_GPIO_38 ▪ PM_GPIO_39 ▪ PM_GPIO_40 ▪ PM_GPIO_41 ▪ PM_GPIO_42 ▪ PM_GPIO_43 ▪ PM_GPIO_44
→	*gpio_status	pm_gpio_status_type: Return a struct containing GPIO status

1

Description

Use this function to get GPIO status. The return structure is illustrated as follows:

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the gpio_status parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Get GPIO 40 status */
pm_gpio_status_type gpio40_status;
errFlag = pm_gpio_status_get(PM_GPIO_40, &gpio40_status);
```

17 User Interface API

In addition to housekeeping functions, the PM8921 IC also supports common handset-level user interfaces. Three dedicated current sinks are intended for driving the keypad backlight, the LCD backlight, and a camera flash. In addition, one MPP is expected to be used as a general-purpose LED driver, although more of the 12 MPPs can be allocated as LED drivers. All drivers provide programmable brightness (current) control.

Alerting the handset user of incoming calls is supported with vibration and audio driver circuits. A vibrator motor driver, compatible with 1.3 to 3.0 V devices (programmable), supports silent alarms. A speaker driver is available for audible alarms; it supports higher audio power applications, such as speakerphone or melody ringer as well. The speaker driver has programmable gain, turn-on time, and muting, and operates as a differential device delivering a volume-controlled 500 mW signal to an external 8 Ohm speaker.

17.1 HSED API

17.1.1 pm_hsed_enable()

This function enables/disables the signal control for the Once Touch Headset controller (HSED) module.

Parameters

```
pm_err_flag_type pm_hsed_enable
(
    pm_hsed_controller_type controller_select,
    pm_hsed_enable_type     enable_type
)
```

→	controller_select	pm_hsed_controller_type: ▪ PM_HSED_CONTROLLER_0 ▪ PM_HSED_CONTROLLER_1 ▪ PM_HSED_CONTROLLER_2
→	enable_type	pm_hsed_enable_type: ▪ PM_HSED_ENABLE_TCXO – Enabled if TCXO_EN is high ▪ PM_HSED_ENABLE_PWM_TCXO – Enabled if PWM signal is high or TCXO_EN is high ▪ PM_HSED_ENABLE_ALWAYS – Enabled always

Description

Use this function to enable or disable the signal control for the HSED module.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the controller_select parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the enable_type parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable headset controller 1 to follow with TCXO_EN (sleep -> OFF, and
 * wakeup from sleep -> ON.
 */
errFlag = pm_hsed_enable( PM_HSED_CONTROLLER_1,
                          PM_HSED_ENABLE_TCXO
                        );
```

17.1.2 pm_hsed_set_current_threshold()

This function configures the headset switch type and the corresponding current threshold for button-press detection.

Parameters

```
pm_err_flag_type pm_hsed_set_current_threshold
(
    pm_hsed_controller_type controller_select,
    pm_hsed_switch_type     switch_type,
    uint32                  current_threshold
)
```

→	controller_select	pm_hsed_ontroller_type: ▪ PM_HSED_CONTROLLER_0 ▪ PM_HSED_CONTROLLER_1 ▪ PM_HSED_CONTROLLER_2
→	switch_type	pm_hsed_switch_type: ▪ PM_HSED_SC_SWITCH_TYPE – Short circuit type ▪ PM_HSED_OC_SWITCH_TYPE – Open circuit type
→	current_threshold	Current threshold to detect a button press: ▪ Short circuit type – 200 uA to 1700 uA ▪ Open circuit type – 10 uA to 160 uA

Description

Use this function to configure the headset switch type and the corresponding current threshold for button-press detection.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the controller_select parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the switch_type parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the current_threshold parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure the headset controller 1 to be a short circuit switch type
 * the corresponding current threshold to 400uA for button press detection.
 */
errFlag = pm_hsed_set_current_threshold( PM_HSED_CONTROLLER_1,
                                         PM_HSED_SC_SWITCH_TYPE,
                                         400
                                        );
```

17.1.3 pm_hsed_set_hysteresis()

This function configures the hysteresis clock pre-divider and the hysteresis clock.

Parameters

```
pm_err_flag_type pm_hsed_set_hysteresis
(
    pm_hsed_controller_type    controller_select,
    pm_hsed_hyst_pre_div_type  hyst_pre_div,
    pm_hsed_hyst_time_type     hyst_time
)
```

→	controller_select	pm_hsed_ontroller_type: ▪ PM_HSED_CONTROLLER_0 ▪ PM_HSED_CONTROLLER_1 ▪ PM_HSED_CONTROLLER_2
→	hyst_pre_div	pm_hsed_hyst_pre_div_type: ▪ PM_HSED_HYST_PRE_DIV_1 – Divide the 1.024 kHz clock by 1 ▪ PM_HSED_HYST_PRE_DIV_2 – Divide the 1.024 kHz clock by 2 ▪ PM_HSED_HYST_PRE_DIV_4 – Divide the 1.024 kHz clock by 4 ▪ PM_HSED_HYST_PRE_DIV_8 – Divide the 1.024 kHz clock by 8 ▪ PM_HSED_HYST_PRE_DIV_16 – Divide the 1.024 kHz clock by 16 ▪ PM_HSED_HYST_PRE_DIV_32 – Divide the 1.024 kHz clock by 32 ▪ PM_HSED_HYST_PRE_DIV_64 – Divide the 1.024 kHz clock by 64 ▪ PM_HSED_HYST_PRE_DIV_128 – Divide the 1.024 kHz clock by 128
→	hyst_time	pm_hsed_hyst_time_type: ▪ PM_HSED_HYST_TIME_1_CLK_CYCLES ▪ PM_HSED_HYST_TIME_2_CLK_CYCLES ▪ PM_HSED_HYST_TIME_3_CLK_CYCLES ▪ PM_HSED_HYST_TIME_4_CLK_CYCLES ▪ PM_HSED_HYST_TIME_5_CLK_CYCLES ▪ PM_HSED_HYST_TIME_6_CLK_CYCLES ▪ PM_HSED_HYST_TIME_7_CLK_CYCLES ▪ PM_HSED_HYST_TIME_8_CLK_CYCLES

Description

Use this function to configure the hysteresis clock pre-divider and the hysteresis clock.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the controller_select parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the hyst_pre_div parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the hyst_time parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the hysteresis clock pre-divider to 4 and the hystersis clock to 7*/
errFlag = pm_hsed_set_hysteresis( PM_HSED_CONTROLLER_1,
                                  PM_HSED_HYST_PRE_DIV_4,
                                  PM_HSED_HYST_TIME_7_CLK_CYCLES
                                );
```

17.1.4 pm_hsed_set_period()

This function configures the period clock pre-divider and the period clock.

Parameters

```
pm_err_flag_type pm_hsed_set_period
(
    pm_hsed_controller_type      controller_select,
    pm_hsed_period_pre_div_type  period_pre_div,
    pm_hsed_period_time_type     period_time
)
```

→	controller_select	pm_hsed_controller_type: ▪ PM_HSED_CONTROLLER_0 ▪ PM_HSED_CONTROLLER_1 ▪ PM_HSED_CONTROLLER_2
→	period_pre_div	pm_hsed_period_pre_div_type: ▪ PM_HSED_PERIOD_PRE_DIV_2 – Divide the 1.024 kHz clock by 2 ▪ PM_HSED_PERIOD_PRE_DIV_4 – Divide the 1.024 kHz clock by 4 ▪ PM_HSED_PERIOD_PRE_DIV_8 – Divide the 1.024 kHz clock by 8 ▪ PM_HSED_PERIOD_PRE_DIV_16 – Divide the 1.024 kHz clock by 16 ▪ PM_HSED_PERIOD_PRE_DIV_32 – Divide the 1.024 kHz clock by 32 ▪ PM_HSED_PERIOD_PRE_DIV_64 – Divide the 1.024 kHz clock by 64 ▪ PM_HSED_PERIOD_PRE_DIV_128 – Divide the 1.024 kHz clock by 128 ▪ PM_HSED_PERIOD_PRE_DIV_256 – Divide the 1.024 kHz clock by 256
→	period_time	pm_hsed_period_time_type: ▪ PM_HSED_PERIOD_TIME_1_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_2_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_3_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_4_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_5_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_6_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_7_CLK_CYCLES ▪ PM_HSED_PERIOD_TIME_8_CLK_CYCLES

Description

Use this function to configure the period clock pre-divider and the period clock.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success

- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the controller_select parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the period_pre_div parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the period_time parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the period clock pre-divider to 16 and the period clock to 7 */
errFlag = pm pm_hsed_set_period( PM_HSED_CONTROLLER_1,
                                 PM_HSED_PERIOD_PRE_DIV_16,
                                 PM_HSED_PERIOD_TIME_7_CLK_CYCLES
                               );
```

17.2 Microphone API

17.2.1 pm_mic_en()

This function enables/disables the PMIC microphone.

Parameters

```
pm_err_flag_type pm_mic_en
(
    boolean enable_disable
)
```

→	enable_disable	▪ PM_OFF_CMD – Enable ▪ PM_ON_CMD – Disable

Description

Use this function to enable or disable the PMIC microphone.

Return values

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the enable_disable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable the PMIC microphone */
err |= pm_mic_en (PM_ON_CMD);
```

17.2.2 pm_mic_is_en()

This function checks if the PMIC microphone is enabled/disabled.

Parameters

```
pm_err_flag_type pm_mic_is_en
(
    boolean *enable_disable
)
```

→	*enable_disable	Pointer to the value for enabled/disabled

Description

Use this function to check if the PMIC microphone is enabled or disabled.

Return values

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the enable_disable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* This function checks if the PMIC microphone is enable */
err |= pm_mic_is_en (*enable_disable);
```

17.2.3 pm_mic_set_volt()

This function sets the PMIC microphone voltage.

Parameters

```
pm_err_flag_type pm_mic_set_volt
(
    pm_mic_volt_type voltage
)
```

→	voltage	pm_mic_volt_type: ▪ PM_MIC_VOLT_2_00V – 2.00 V ▪ PM_MIC_VOLT_1_93V – 1.93 ▪ M_MIC_VOLT_1_80V – 1.80 V ▪ PM_MIC_VOLT_1_73V – 1.73 V

Description

Use this function to set the PMIC microphone voltage.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the voltage parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* This function sets the PMIC microphone voltage to 1.93 V */
if (PM_ERR_FLAG__SUCCESS != pm_mic_set_volt ( PM_MIC_VOLT_1_93V ))
{
   MSG_ERROR("Could not set the microphone voltage to 1.93 V",0,0,0);
}
```

17.2.4 pm_mic_get_volt()

This function gets the PMIC microphone voltage.

Parameters

```
pm_err_flag_type pm_mic_get_volt
(
    pm_mic_volt_type *voltage
)
```

→	*voltage	Pointer to the microphone voltage value

Description

Use this function to get the PMIC microphone voltage.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the voltage parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The function pm_init() must be invoked before this function can be used:

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* This function sets the PMIC microphone voltage */
if (PM_ERR_FLAG__SUCCESS != pm_mic_get_volt (*voltage))
{
   MSG_ERROR("Could not get the PMIC microphone voltage",0,0,0);
}
```

17.3 LED driver API

17.3.1 pm_set_led_intensity()

This function sets the intensity of a selected LED driver signal.

Parameters

```
pm_err_flag_type pm_set_led_intensity
(
    pm_led_intensity_type led_driver,
    uint8 current_sink_setting
)
```

→	led_driver	pm_led_intensity_type: ▪ PM_LCD_LED – LED driver for the liquid crystal display ▪ PM_KBD_LED – LED driver for the keypad
→	current_sink_setting	Intensity value represents the current sink setting for the selected LED driver: ▪ 0 – 0 mA ▪ 1 – -10 mA ▪ 2 – -20 mA ▪ 3 – -30 mA ▪ 4 – -40 mA ▪ 5 – -50 mA ▪ 6 – -60 mA ▪ 7 – -70 mA ▪ 8 – -80 mA ▪ 9 – -90 mA ▪ 10 – -100 mA ▪ 11 – -110 mA ▪ 12 – -120 mA ▪ 13 – -130 mA ▪ 14 – -140 mA ▪ 15 – -150 mA

Description

Use this function to set the intensity of a selected LED driver signal.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_driver parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the current_sink_setting parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the LCD LED driver's current sink rating to –10 mA */
errFlag = pm_set_led_intensity(PM_LCD_LED, 1);
```

17.3.2 pm_get_led_intensity()

This function gets the intensity of the indicated LED driver.

Parameters

```
pm_err_flag_type pm_get_led_intensity
(
    pm_led_intensity_type led_driver,
    uint8 *ptr_to_current_sink_setting
)
```

→	led_driver	pm_led_intensity_type: ▪ PM_LCD_LED – LED driver for the liquid crystal display ▪ PM_KBD_LED – LED driver for the keypad
→	*ptr_to_current_sink_setting	Pointer to the intensity value: ▪ 0 – 0 mA ▪ 1 – -10 mA ▪ 2 – -20 mA ▪ 3 – -30 mA ▪ 4 – -40 mA ▪ 5 – -50 mA ▪ 6 – -60 mA ▪ 7 – -70 mA ▪ 8 – -80 mA ▪ 9 – -90 mA ▪ 10 – -100 mA ▪ 11 – -110 mA ▪ 12 – -120 mA ▪ 13 – -130 mA ▪ 14 – -140 mA ▪ 15 – -150 mA

Description

Use this function to get the intensity of the selected LED driver signal.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_driver parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the ptr_to_current_sink_setting parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Get the LCD LED driver's current sink rating */
uint8 current_sink_setting;
errFlag = pm_get_led_intensity(PM_LCD_LED, &current_sink_setting);
```

17.4 Flash LED API

17.4.1 pm_flash_led_set_current()

This function configures the current setting for the Flash LED.

Parameters

```
pm_err_flag_type pm_flash_led_set_current
(
    uint16 current
)
```

→	current	Current setting: ▪ 0 mA -> OFF ▪ 0 – 300 mA in 20 mA steps

Description

Use this function to configure the current setting for the Flash LED.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the current parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the Flash LED current to 40 mA */
err  = pm_flash_led_set_current(40);
```

17.4.2 pm_flash_led_set_mode()

Use this function to configure the Flash LED to Manual mode or Automatic mode.

- Manual mode:
  - Disable by setting the Flash LED current to 0 mA.
  - Enable by setting the Flash LED current to a setting within the valid current range.
- Automatic mode:
  - Enable, where the current = pm_flash_led_set_current() setting.
  - Disable depending on the logic state of the corresponding DBUS line; 0 mA regardless of the pm_flash_led_set_current() setting.

Parameters

```
pm_err_flag_type pm_flash_led_set_mode
(
    pm_flash_led_mode_type mode
)
```

→	mode	pm_flash_led_mode_type: ▪ PM_FLASH_LED_MODE__MANUAL ▪ PM_FLASH_LED_MODE__DBUS1 – Automatic mode, controlled by DBUS1 line ▪ PM_FLASH_LED_MODE__DBUS2 – Automatic mode, controlled by DBUS2 line ▪ PM_FLASH_LED_MODE__DBUS3 – Automatic mode, controlled by DBUS3 line

Description

Use this function to configure the Flash LED to Manual mode or one of the Automatic modes.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mode parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable the Flashed LED */
errFlag = pm_flash_led_set_current(40); Configure MPP5 to be a digital
input, set the voltage reference to MSMP VREG and drive data bus 2 (DBUS2)
err |= pm_mpp_config_digital_input(PM_MPP_5,
                                   PM_MPP__DLOGIC__LVL_MSMP,
                                   PM_MPP__DLOGIC_IN__DBUS2
                                   ); Configure Flash LED to automatic and
use DBUS2 as the control line
err |= pm_flash_led_set_mode(PM_FLASH_LED_MODE__DBUS2); Configure DBUS
polarity to active high
err |= pm_flash_led_set_polarity(PM_FLASH_LED_POL__ACTIVE_HIGH);  Configure
the Flash LED to 280 mA
err |= pm_flash_led_set_current(280); At this point, the Flash LED will
turn on (280 mA) if MPP5 is high and turn off if MPP5 is low.
```

17.4.3 pm_flash_led_set_polarity()

This function sets the Flash LED polarity.

Parameters

```
pm_err_flag_type pm_flash_led_set_polarity
(
    pm_flash_led_pol_type pol
)
```

→	pol	pm_flash_led_pol_type: ▪ PM_FLASH_LED_POL__ACTIVE_HIGH – Active high polarity ▪ PM_FLASH_LED_POL__ACTIVE_LOW – Active low polarity

Description

Use this function to set the Flash LED polarity.

If the Flash LED is in Automatic mode (see pm_flash_led_set_mode() in Section 17.4.2), this function configures the polarity of the DBUS line.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the pol parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable the Flashed LED */
      err  = pm_flash_led_set_current(0);
 – Configure MPP3 to be a digital input, set the voltage reference
 – to MSMP VREG and drive data bus 1 (DBUS1)
      err |= pm_mpp_config_digital_input(PM_MPP_3,
                                         PM_MPP__DLOGIC__LVL_MSMP,
                                         PM_MPP__DLOGIC_IN__DBUS1
                                        ); Configure Flash LED to automatic
and use DBUS1 as the control line
err |= pm_flash_led_set_mode(PM_FLASH_LED_MODE__DBUS1); Configure DBUS1
polarity to active low
err |= pm_flash_led_set_polarity(PM_FLASH_LED_POL__ACTIVE_LOW); Configure
the Flash LED to 280 mA
err |= pm_flash_led_set_current(280); At this point, the Flash LED will
turn on (280 mA) if MPP3 is low and turn off if MPP3 is high.
```

17.4.4 pm_high_current_led_set_current()

This function sets the current of high current LEDs.

Parameters

```
pm_err_flag_type pm_high_current_led_set_current
(
   pm_high_current_led_type led_type,
   uint16 current
)
```

→	led_type	pm_high_current_led_type: ▪ PM_FLASH_DRV0_LED ▪ PM_FLASH_DRV1_LED ▪ PM_KBD_DRV_LED
→	current	uint16: 0-300 mA in step of 20 mA

Description

Use this function to set the current of high current LEDs

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_type parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the current parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set FLASH_DRV0_LED to output 80 mA */
errFlag |= pm_high_current_led_set_current(PM_FLASH_DRV0_LED , 80);
```

17.4.5 pm_high_current_led_set_mode()

This function sets the mode of controlling high current LEDs.

Parameters

```
pm_err_flag_type pm_high_current_led_set_mode
(
   pm_high_current_led_type led_type,
   pm_flash_led_mode_type mode
)
```

→	led_type	pm_high_current_led_type: ▪ PM_FLASH_DRV0_LED ▪ PM_FLASH_DRV1_LED ▪ PM_KBD_DRV_LED
→	mode	pm_flash_led_mode_type: ▪ PM_FLASH_LED_MODE__MANUAL – Manual Mode ▪ PM_FLASH_LED_MODE__DBUS1 – Automatic Mode, Controlled by DBUS1 line ▪ PM_FLASH_LED_MODE__DBUS2 – Automatic Mode, Controlled by DBUS2 line ▪ PM_FLASH_LED_MODE__DBUS3 – Automatic Mode, Controlled by DBUS3 line

Description

Use this function to set the mode of controlling high current LEDs

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_type parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the mode parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set FLASH_DRV0_LED to manual mode */
errFlag |= pm_high_current_led_set_mode(  PM_FLASH_DRV0_LED,
PM_FLASH_LED_MODE__MANUAL);
```

17.4.6 pm_high_current_led_set_polarity()

This function sets the polarity of controlling current of high current LEDs.

Parameters

```
pm_err_flag_type pm_high_current_led_set_polarity
(
   pm_high_current_led_type led_type,
   pm_flash_led_pol_type pol
)
```

→	led_type	pm_high_current_led_type: ▪ PM_FLASH_DRV0_LED ▪ PM_FLASH_DRV1_LED ▪ PM_KBD_DRV_LED
→	pol	pm_flash_led_pol_type: ▪ PM_FLASH_LED_POL__ACTIVE_HIGH – Active high polarity ▪ PM_FLASH_LED_POL__ACTIVE_LOW – Active low polarity

Description

Use this function to set the polarity of controlling of high current LEDs

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_type parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the mode parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – API is not available on this PMIC

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set FLASH_DRV1_LED to be ON when control signal is high */
pm_high_current_led_set_polarity(    PM_FLASH_DRV1_LED,
                                     PM_FLASH_LED_POL__ACTIVE_HIGH);
```

17.5 Vibrator API

Alerting the handset user of incoming calls is supported with vibration and audio driver circuits. A vibrator motor driver, compatible with 1.3V to 3.0V devices (programmable), supports silent alarms.

17.5.1 pm_vib_mot_set_volt()

This function configures the voltage setting for the vibrator motor.

Parameters

```
pm_err_flag_type pm_vib_mot_set_volt
(
  uint16 vlt
)
```

→	vlt	Voltage setting: ▪ 0 mV (off, default) ▪ 1200 to 3100 mV, in 100 mV steps

Description

Use this function to configure the voltage setting for the vibrator motor.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vlt parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Disable the vibrator motor. */
err  = pm_vib_mot_set_volt(0);

/* This function configures MPP5 to be a digital input, set the voltage
reference to MSMP VREG and drive data bus 2 (DBUS2). */
err |= pm_mpp_config_digital_input(PM_MPP_5, PM_MPP__DLOGIC__LVL_MSMP,
PM_MPP__DLOGIC_IN__DBUS2);

/* This function configures vibrator motor to automatic and use DBUS2 as
the control line. */
err |= pm_vib_mot_set_mode(PM_VIB_MOT_MODE__DBUS2);

/* This function configures DBUS polarity to active high. */
err |= pm_vib_mot_set_polarity(PM_VIB_MOT_POL__ACTIVE_HIGH);

/* This function configures the vibrator motor to 3 Volts. */
err |= pm_vib_mot_set_volt(3000);

/* At this point, the Vibrator will turn on (3V) if MPP5 is high and turn
off if MPP5 is low. */
```

17.5.2 pm_vib_mot_set_polarity()

This function sets the polarity of the DBUS line when the vibrator motor is in Automatic mode.

Parameters

```
pm_err_flag_type pm_vib_mot_set_polarity
(
  pm_vib_mot_pol_type pol
)
```

→	pol	pm_vib_mot_pol_type: ▪ PM_VIB_MOT_POL__ACTIVE_HIGH ▪ PM_VIB_MOT_POL__ACTIVE_LOW

Description

Use this function to set the polarity of the DBUS line when the vibrator motor is in Automatic mode, as configured by the pm_vib_mot_set_mode() function.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the pol parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

See the example on page 318.

17.5.3 pm_vib_mot_set_mode()

This function configures the Vibrator Motor mode.

Parameters

```
pm_err_flag_type pm_vib_mot_set_mode
(
  pm_vib_mot_mode_type mode
)
```

→	mode	pm_vib_mot_mode_type: ▪ PM_VIB_MOT_MODE__MANUAL – Manual mode ▪ PM_VIB_MOT_MODE__DBUS1 – Automatic mode, controlled by DBUS1 line ▪ PM_VIB_MOT_MODE__DBUS2 – Automatic mode, controlled by DBUS2 line ▪ PM_VIB_MOT_MODE__DBUS3 – Automatic mode, controlled by DBUS3 line

Description

Use this function to configure the vibrator motor to Manual mode or one of the Automatic modes.

- Manual mode
  - Disable by setting the vibrator motor voltage to 0 mV
  - Enable by setting the vibrator motor voltage to a setting within the valid range

NOTE: See Section 17.5.1 for the vibrator motor voltage setting.

- Automatic mode
  - Vibrator motor is enabled/disabled, depending on the Logic state of the corresponding DBUS line.

NOTE: For the DBUS polarity setting, see Section 17.5.2.

NOTE: To configure an MPP to drive a DBUS line, see Section 11.1.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the mode parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The following functions must be invoked before this function can be used:

- pm_init(), pm_vib_mot_set_volt()
- pm_vib_mot_set_polarity()
- pm_mpp_config_digital_input()

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

See the example on page 318.

17.6 Pulse width modulator API

The PMIC supports silent incoming call alarms with its vibration motor driver (see **Error! Reference source not found.**) that has a dedicated output pin (VIB_DRV_N). The vibration driver is a programmable voltage output that is referenced to VDD. When off, its output voltage is VDD. The motor is connected between VDD and VIB_DRV_N. The voltage across the motor is Vm = VDD – V_{out}, where V_{out} is the voltage at the PMIC pin.

17.6.1 pm_pwm_generator_select_clock()

This function selects the input clock source for the PWM generator.

Parameters

```
pm_err_flag_type pm_pwm_generator_select_clock
(
    pm_pwm_generator_type generator,
    pm_pwm_generator_clock_type clock
)
```

→	generator	pm_pwm_generator_type: ■ PM_PWM_GENERATOR_1 ■ PM_PWM_GENERATOR_2 ■ PM_PWM_GENERATOR_3
→	clock	pm_pwm_generator_clock_type: ■ PM_PWM_GENERATOR_CLOCK_OFF ■ PM_PWM_GENERATOR_CLOCK_1kHz ■ PM_PWM_GENERATOR_CLOCK_32kHz ■ PM_PWM_GENERATOR_CLOCK_19_2 MHz

Description

Use this function to select the input clock source for the PWM generator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the generator parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the clock parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The `pm_init()` function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Select clock source from 32kHz for PWM Generator 1 */
err |= pm_pwm_generator_select_clock( PM_PWM_GENERATOR_1,
                                      PM_PWM_GENERATOR_CLOCK_32kHz
                                    );
```

17.6.2 pm_pwm_generator_select_pre_divider()

This function selects the clock pre-divider frequency for the PWM generator.

Parameters

```
pm_err_flag_type pm_pwm_generator_select_pre_divider
(
    pm_pwm_generator_type generator
    pm_pwm_generator_pre_div_type pre_div,
    pm_pwm_generator_pre_div_exp_type exp
)
```

→	generator	pm_pwm_generator_type: ■ PM_PWM_GENERATOR_1 ■ PM_PWM_GENERATOR_2 ■ PM_PWM_GENERATOR_3
→	pre_div	pm_pwm_generator_pre_div_type: ■ PM_PWM_GENERATOR_PRE_DIV_BY_2 ■ PM_PWM_GENERATOR_PRE_DIV_BY_3
→	exp	pm_pwm_generator_pre_div_exp_type: ■ PM_PWM_GENERATOR_PRE_DIV_EXP_0 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_1 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_2 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_3 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_4 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_5 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_6 ■ PM_PWM_GENERATOR_PRE_DIV_EXP_7

Description

Use this function to select the clock pre-divider frequency for the PWM generator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the generator parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the pre_div parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the exp parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The `pm_init()` function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Select clock divider by 2 for PWM Generator 1 */
err |= pm_pwm_generator_select_pre_divider( PM_PWM_GENERATOR_1,
                                            PM_PWM_GENERATOR_PRE_DIV_BY_2,
                                            PM_PWM_GENERATOR_PRE_DIV_EXP_4
                                          );
```

17.6.3 pm_pwm_generator_set()

This function sets the PWM size (6 bit or 9 bit) and its associated 6-bit or 9-bit value.

Parameters

```
pm_err_flag_type pm_pwm_generator_set
(
    pm_pwm_generator_type generator,
    pm_pwm_generator_size_type size,
    uint16 pwm_value
)
```

→	generator	pm_pwm_generator_type: ■ PM_PWM_GENERATOR_1 ■ PM_PWM_GENERATOR_2 ■ PM_PWM_GENERATOR_3
→	size	pm_pwm_generator_size_type: ■ PM_PWM_GENERATOR_SIZE_6_BIT ■ PM_PWM_GENERATOR_SIZE_9_BIT
→	pwm_val	uint16 range

Description

Use this function to set the PWM size (6 bit or 9 bit) and its associated 6-bit or 9-bit value.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the generator parameter
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the size parameter
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the pwm_val parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The following functions must be invoked before this function can be used:

- `pm_init()`
- `pm_vib_mot_set_volt()`
- `pm_vib_mot_set_polarity(`
- `pm_mpp_config_digital_input()`

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the PWM size of generator 2 to be 9 bits */
err  = pm_pwm_generator_set(PM_PWM_GENERATOR_2,
                            PM_PWM_GENERATOR_SIZE_9_BIT
                            5
                            );
```

17.7 USB overvoltage protection API

When this feature is enabled and supplemented by an external N-channel MOSFET, the PM8921 IC provides overvoltage protection between the external source and the PMIC input power pins (two VCHG pins). This safety mechanism protects the PMIC from a wall charger or USB car kit that is malfunctioning or has poor no-load regulation

17.7.1 pm_usb_ovp_enable()

This function enables/disables the USB Overvoltage Protection (OVP) functionality.

Parameters

```
pm_err_flag_type pm_usb_ovp_enable
(
    boolean enable
)
```

→	enable	▪ TRUE – 1; enable ▪ FALSE – 0; disable

Description

Use this function to enable or disable the USB OVP functionality.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the enable parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable USB OVP */
err |= pm_usb_ovp_enable(TRUE);
```

17.7.2 pm_usb_ovp_set_threshold()

This function configures the voltage threshold that triggers the USB OVP functionality.

Parameters

```
pm_err_flag_type pm_usb_ovp_set_threshold
(
    pm_usb_ovp_threshold_type threshold)
```

→	threshold	pm_usb_ovp_threshold_type: ▪ PM_USB_OVP_THRESHOLD_5V5 ▪ PM_USB_OVP_THRESHOLD_6V ▪ PM_USB_OVP_THRESHOLD_6V5 ▪ PM_USB_OVP_THRESHOLD_7V

Description

Use this function to configure the voltage threshold that triggers the USB OVP functionality.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the threshold parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* This function sets the USB OVP to 6.5V */
err |= pm_usb_ovp_set_threshold(PM_USB_OVP_THRESHOLD_6V5);
```

17.7.3 pm_usb_ovp_set_hystersis()

This function configures the hysteresis for the voltage to exceed the threshold before it triggers the USB OVP functionality.

Parameters

```
pm_err_flag_type pm_usb_ovp_set_hystersis
(
    pm_usb_ovp_hystersis_type hystersis
)
```

→	hystersis	pm_usb_ovp_hystersis_type: ▪ PM_USB_OVP_HYSTERSIS_0ms ▪ PM_USB_OVP_HYSTERSIS_20ms ▪ PM_USB_OVP_HYSTERSIS_40ms ▪ PM_USB_OVP_HYSTERSIS_80ms

Description

Use this function to configure the hysteresis for the voltage to exceed the threshold before it triggers the USB OVP functionality.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the hystersis parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The following functions must be invoked before this function can be used:

- pm_init()
- pm_vib_mot_set_volt()
- pm_vib_mot_set_polarity()
- pm_mpp_config_digital_input()

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Set the hystersis timer to 40 mSec. for the OVP to trigger */
err  = pm_usb_ovp_set_hystersis(PM_USB_OVP_HYSTERSIS_40ms);
```

17.8 UMTS subscriber identity API

17.8.1 pm_usim_aux_enable()

This function enables/disables the UMTS Subscriber Identity Module (USIM) AUX module.

Parameters

```
pm_err_flag_type pm_usim_aux_enable
(
    pm_switch_cmd_type on_off
)
```

→	on_off	pm_switch_cmd_type: ▪ PM_ON_CMD – Enable ▪ PM_OFF_CMD – Disable (default)

Description

Use this function to enable or disable the USIM AUX module.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the on_off parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable USIM Aux module */
err |= pm_usim_aux_enable(PM_ON_CMD);
```

17.8.2 pm_usim_aux_vreg_set()

This function sets the power supply (auxiliary voltage regulator) for the USIM

Parameters

```
pm_err_flag_type pm_usim_aux_vreg_set
(
    pm_usim_aux_cntrl_vreg_type vreg
)
```

→	vreg	pm_usim_aux_cntrl_vreg_type: ▪ PM_USIM_AUX_CTRL_VREG_MSME ▪ PM_USIM_AUX_CTRL_VREG_RUIM

Description

Use this function to set the power supply (auxiliary voltage regulator) for the USIM.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the vreg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* This function sets VREG_MSME as the power supply for USIM */
err |= pm_usim_aux_vreg_set(PM_USIM_AUX_CTRL_VREG_MSME);
```

17.9 Keypad

17.9.1 pm_kypd_set_keypad_size()

This function sets the keypad rows, cols.

Parameters

```
pm_err_flag_type pm_kypd_set_keypad_size
(
    pm_kypd_scan_rows kypd_rows,
    pm_kypd_scan_cols kypd_cols
)
```

→	kypd_rows	pm_kypd_scan_rows: ▪ PM_KYPD_5_ROWS ▪ PM_KYPD_6_ROWS ▪ PM_KYPD_7_ROWS ▪ PM_KYPD_8_ROWS ▪ PM_KYPD_10_ROWS ▪ PM_KYPD_12_ROWS ▪ PM_KYPD_15_ROWS ▪ PM_KYPD_18_ROWS
→	kypd_cols	pm_kypd_scan_cols: ▪ PM_KYPD_5_COLS ▪ PM_KYPD_6_COLS ▪ PM_KYPD_7_COLS ▪ PM_KYPD_8_COLS

Description

This API should be called by PMIC APP layer keypad driver to set the keypad rows, cols.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the kypd_rows parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the kypd_cols parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Configure a 5 by 5 keypad*/
if (PM_ERR_FLAG__SUCCESS != pm_kypd_set_keypad_size(    PM_KYPD_5_ROWS,
                                                        PM_KYPD_5_COLS))
{
    return FALSE;
}
if (PM_ERR_FLAG__SUCCESS != pm_kypd_set_keypad_scan_timers(
                                PM_KYPD_ROW_HOLD_31_25uS,
                                PM_KYPD_PAUSE_2uS,
                                PM_KYPD_DBOUNCE_2uS))
{
    return FALSE;
}


if (PM_ERR_FLAG__SUCCESS != pm_kypd_configure_gpios())
{
    return FALSE;
}
```

17.9.2 pm_kypd_set_keypad_scan_timers()

This function sets the timers for keypad.

Parameters

```
pm_err_flag_type pm_kypd_set_keypad_scan_timers
(
    pm_kypd_row_hold_time  kypd_hold_time,
    pm_kypd_pause_time     kypd_pause_time,
    pm_kypd_dbounce_time   kypd_dbounce_time
)
```

→	kypd_hold_time	pm_kypd_row_hold_time: ▪ PM_KYPD_ROW_HOLD_31_25uS ▪ PM_KYPD_ROW_HOLD_62_50uS ▪ PM_KYPD_ROW_HOLD_125_00uS ▪ PM_KYPD_ROW_HOLD_250_00uS
→	kypd_pause_time	pm_kypd_pause_time: ▪ PM_KYPD_PAUSE_1mS ▪ PM_KYPD_PAUSE_2mS ▪ PM_KYPD_PAUSE_4mS ▪ PM_KYPD_PAUSE_8mS ▪ PM_KYPD_PAUSE_16mS ▪ PM_KYPD_PAUSE_32mS ▪ PM_KYPD_PAUSE_64mS ▪ PM_KYPD_PAUSE_128mS
→	kypd_dbounce_time	pm_kypd_dbounce_time: ▪ PM_KYPD_DBOUNCE_1mS ▪ PM_KYPD_DBOUNCE_2mS ▪ PM_KYPD_DBOUNCE_4mS ▪ PM_KYPD_DBOUNCE_8mS

Description

This API should be called by PMIC APP layer keypad driver and specify the different delays.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the kypd_hold_time parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the kypd_pause_time parameter

- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the kypd_dbounce_time parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Rrefer to example in 17.9.1

17.9.3 pm_kypd_configure_gpios()

This function configures GPIOs for PMIC Keypad.

Parameters

```
pm_err_flag_type pm_kypd_configure_gpios
(
    void
)
```

Description

This API will be called from PMIC APP layer to configure the GPIOs required for PMIC KEYPAD.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.9.1

17.9.4 pm_kypd_get_queued_events()

This function returns number of queued events from PMIC Keypad module.

Parameters

```
pm_err_flag_type pm_kypd_get_queued_events
(
    uint8 *kypd_events
)
```

→	*kypd_events	uint8: Number of queue events, 0 – 3

Description

Use this API to get number of queued events from PMIC Keypad module

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the kypd_rows parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
uint8   kypd_events_queued = 0;
byte    *recent_data_ptr;
byte    *old_data_ptr = NULL;

/* check queued events before reading recent data */
errFlag |= pm_kypd_get_queued_events(&kypd_events_queued);
//read recent data
recent_data_ptr = pm_kypd_get_keypad_data(TRUE);

if((errFlag == PM_ERR_FLAG__SUCCESS) && (kypd_events_queued >= 2))
{
    //read old data
    old_data_ptr = pm_kypd_get_keypad_data(FALSE);

...
}
```

17.9.5 pm_kypd_get_keypad_data()

This function get data from PMIC keypad module.

Parameters

```
byte* pm_kypd_get_keypad_data
(
    boolean                     flag
)
```

→	flag	boolean: ▪ FALSE – old data ▪ TRUE – recent data

Description

Use this API get data from PMIC keypad module.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the kypd_rows parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.9.4

17.10 Low Current LED

17.10.1 pm_low_current_led_set_current()

This function sets the current of low current LEDs

Parameters

```
pm_err_flag_type pm_low_current_led_set_current
(
    pm_low_current_led_type led_type,
uint16 current
)
```

→	led_type	pm_low_current_led_type: ▪ PM_LOW_CURRENT_LED_DRV0 ▪ PM_LOW_CURRENT_LED_DRV1 ▪ PM_LOW_CURRENT_LED_DRV2
→	current	uint16 current: Valid range: 0 to 40 in unit of mA

Description

Use this API sets the cuurent of low current LEDs.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_type parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the current parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Configure LED DRV0*/
errFlag |= pm_low_current_led_set_current(PM_LOW_CURRENT_LED_DRV0 , 20);
```

17.10.2 pm_low_current_led_set_ext_signal()

This function sets the external signal source of low current LED

Parameters

```
pm_err_flag_type pm_low_current_led_set_ext_signal
(
    pm_low_current_led_type led_type,
pm_ext_signal_selection_type ext_signal
)
```

→	led_type	pm_low_current_led_type: ▪ PM_LOW_CURRENT_LED_DRV0 ▪ PM_LOW_CURRENT_LED_DRV1 ▪ PM_LOW_CURRENT_LED_DRV2
→	ext_signal	pm_low_current_led_type: ▪ PM_CURRENT_SINK_MANUAL_MODE ▪ PM_CURRENT_SINK_PWM1 ▪ PM_CURRENT_SINK_PWM2 ▪ PM_CURRENT_SINK_PWM3 ▪ PM_CURRENT_SINK_DTEST1 ▪ PM_CURRENT_SINK_DTEST2 ▪ PM_CURRENT_SINK_DTEST3 ▪ PM_CURRENT_SINK_DTEST4

Description

Use this API to set the external signal source of low current LED.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the led_type parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the ext_signal parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Configure LED DRV0 in manual mode*/
errFlag |= pm_low_current_led_set_ext_signal(
             PM_LOW_CURRENT_LED_DRV1,
             PM_CURRENT_SINK_MANUAL_MODE);
```

17.11 LPG

17.11.1 pm_lpg_interval_counter_set()

This function sets the time to wait before incrementing the interval counter.

Parameters

```
pm_err_flag_type pm_lpg_interval_counter_set
(
int which_lpg,
int counts
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	counts	uint16 count: counts of 1kHz clock to wait before incrementing the interval counter (time step between loading the next value from the LUT) the values range between 1 and 512. (1, 2, 3, 4, 6, 8, 16, 18, 24, 32, 36, 64, 128, 256, 512)

Description

Use this API to set the time to wait before incrementing the interval counter.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the counts parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/*Configure LPG Bank 4, 512 wait count*/
errFlag |= pm_lpg_interval_counter_set(4, 512);
```

17.11.2 pm_lpg_pwm_output_enable()

This function enables or disables the PWM.

Parameters

```
pm_err_flag_type pm_lpg_pwm_output_enable
(
    int which_lpg,
    boolean enable
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Enable PWM output ▪ FALSE – Disable PWM output (Tri-state)

Description

Use this API to enable or disable the PWM

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* select the PWM clock */
err_flag |= pm_lpg_pwm_clk_select(   backlight_lpg_channel,
                                     PM_LPG_CLK_SELECT_19P2M);

/* leave prediv and exponent at default */
err_flag |= pm_lpg_pwm_prediv_set(backlight_lpg_channel, 2);
err_flag |= pm_lpg_pwm_prediv_exponent_set(backlight_lpg_channel, 0);

/* enable PWM output */
err_flag |= pm_lpg_pwm_output_enable(backlight_lpg_channel, TRUE);

/* enable PWM */
err_flag |= pm_lpg_pwm_enable(backlight_lpg_channel, TRUE);

/* enable direct writing to the PWM register (bypass lookup table) */
err_flag |= pm_lpg_pwm_bypass_enable(backlight_lpg_channel, TRUE);

/* select 9-bit PWM mode */
err_flag |= pm_lpg_pwm_ninebit_enable(backlight_lpg_channel, TRUE);

/* enable LPG bank for the 8960  backlight */
err_flag |= pm_lpg_pwm_bank_enable(backlight_lpg_channel, TRUE);

/* connect the LPG module to the DTEST */
err_flag |= pm_lpg_dtest_config(backlight_lpg_channel, 1);

/* configure the high current led driver */
err_flag |= pm_high_current_led_set_current (PM_FLASH_DRV0_LED, 300);

err_flag |= pm_flash_led_set_mode (PM_FLASH_LED_MODE__DBUS3);

err_flag |= pm_flash_led_set_polarity (PM_FLASH_LED_POL__ACTIVE_HIGH);

if (err_flag != PM_ERR_FLAG__SUCCESS)
{
        return err_flag;
}
```

17.11.3 pm_lpg_pwm_enable()

This function enables or disables the LPG PWM module.

Parameters

```
pm_err_flag_type pm_lpg_pwm_enable
(
int which_lpg,
boolean enable
);
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Enable LPG PWM module ▪ FALSE – Disable LPG PWM module

Description

Use this API to enable or disable the LPG PWM module.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.4 pm_lpg_pwm_ramp_generator_enable()

This function enables or disables the PWM ramp generator.

Parameters

```
pm_err_flag_type pm_lpg_pwm_ramp_generator_enable
(
int which_lpg,
boolean enable
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Enable LPG ramp generator ▪ FALSE – Disable LPG ramp generator

Description

Use this API to enable or disable the PWM ramp generator.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable LPG Bank 0 ramp generator */
err_flag |= pm_lpg_pwm_ramp_generator_enable(0, TRUE);
```

17.11.5 pm_lpg_pwm_ramp_generator_start()

This function starts the ramp generator.

Parameters

```
pm_err_flag_type pm_lpg_pwm_ramp_generator_start
(
    int which_lpg,
    boolean start
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Start the LPG ramp function. ▪ FALSE – Stop the LPG ramp function.

Description

Use this API to start the LPG PWM ramp generator. The LPG module will step through the values that have been programmed into the PWM look up table.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Start LPG Bank 0 ramp generator */
err_flag |= pm_lpg_pwm_ramp_generator_start(0, TRUE);
```

17.11.6 pm_lpg_pwm_toggle_updown()

This function controls behavior at the end of the LUT sequence

Parameters

```
pm_err_flag_type pm_lpg_pwm_toggle_updown
(
    int which_lpg,
    boolean enable
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Toggle (reverse direction when end value is reached) ▪ FALSE – Do not toggle (return to start value when end value is reached).

Description

This API controls whether to return to the start value or reverse direction

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Enable LPG Bank 0 toggle feature */
err_flag |= pm_lpg_pwm_toggle_updown(0, TRUE);
```

17.11.7 pm_lpg_pwm_bypass_enable()

This function enables writing a PWM value directly

Parameters

```
pm_err_flag_type pm_lpg_pwm_bypass_enable
(
    int which_lpg,
    boolean bypass_enable
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	bypass_enable	boolean: ▪ TRUE – enable user writing in PWM value ▪ FALSE – disallow user writing in PWM value (read from LUT)

Description

This API allows direct writing of PWM values, bypassing the look up table

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.8 pm_lpg_pwm_lut_index_set()

This function sets the look up table index

Parameters

```
pm_err_flag_type pm_lpg_pwm_lut_index_set
(
    int which_lpg,
    int low_index,
    int high_index
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	low_index	▪ int low_index: Low look up table index to use with loop and ramp functions. Range is 0 to 63.
→	high_index	Int high_index: High look up table index to use with loop and ramp functions. Range is 0 to 63.

Description

This API sets the look up table index used for loop and ramp functions.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the low_index parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the high_index parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure LPG Bank 0 LUT */
errFlag = pm_lpg_pwm_lut_index_set(0, 0, 32);
```

17.11.9 pm_lpg_pwm_pause_set()

This function sets the pause time at the high and low index value

Parameters

```
pm_err_flag_type pm_lpg_pwm_pause_set
(
    int which_lpg,
    int high_index_pause,
    int low_index_pause
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	high_index_pause	▪ int high_index_pause: ▪ Pause (# interval counter steps) at high index value. Valid pause values range from 1 to 7000
→	low_index_pause	▪ int low_index_pause: ▪ Pause (# interval counter steps) at low index value. Valid pause values range from 1 to 7000

Description

This API sets the look up table index used for loop and ramp functions.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the high_index_pause parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the low_index_pause parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure LPG Bank 0 pause count */
errFlag = pm_lpg_pwm_pause_set(0, 100, 100);
```

17.11.10 pm_lpg_loop_enable()

This function enables or disables the loop function.

Parameters

```
pm_err_flag_type pm_lpg_loop_enable
(
    int which_lpg,
    boolean start
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	start	boolean: ▪ TRUE – start looping through the look up table ▪ FALSE – stop looping

Description

This API configures the LPG module to loop through the look up table.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Start LPG Bank 0 looping function */
errFlag = pm_lpg_loop_enable(0, TRUE);
```

17.11.11 pm_lpg_ramp_direction()

This function controls the direction of the LPG.

Parameters

```
pm_err_flag_type pm_lpg_ramp_direction
(
    int which_lpg,
    boolean ramp_direction
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	ramp_direction	boolean: ▪ TRUE – Ramp up ▪ FALSE – Ramp Down

Description

This API controls the order in which PWM values are read from the look up table

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure LPG Bank 0 to use LUT with ramp-up order */
errFlag = pm_lpg_ramp_direction(0, TRUE);
```

17.11.12 pm_lpg_pwm_value_set()

This function writes a PWM value.

Parameters

```
pm_err_flag_type pm_lpg_pwm_value_set
(
    int which_lpg,
    unsigned short value
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	value	unsigned short value: ▪ The PWM value to write. Valid Range is 0 to 0x1FF

Description

This API writes a PWM value to the LPG when in bypass mode.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the value parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.13 pm_lpg_pwm_clk_select()

This function sets the PWM clock source.

Parameters

```
pm_err_flag_type pm_lpg_pwm_clk_select
(
    int which_lpg,
    pm_lpg_pwm_clk_select_type clk
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	clk	pm_lpg_pwm_clk_select_type: ▪ PM_LPG_CLK_SELECT_OFF – OFF ▪ PM_LPG_CLK_SELECT_1K – 1KHz ▪ PM_LPG_CLK_SELECT_32K – 32KHz ▪ PM_LPG_CLK_SELECT_19P2M – 19.2MHz

Description

This API sets the PWM clock source to one of the values in pm_lpg_pwm_clk_select_type.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.14 pm_lpg_pwm_prediv_set()

This function sets the clock pre-divider.

Parameters

```
pm_err_flag_type pm_lpg_pwm_prediv_set
(
    int which_lpg,
    unsigned short pre_div_value
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	pre_div_value	unsigned short value: ▪ PWM clock pre-divider Valid settings are 2 (default), 3, 5, or 6

Description

This API sets the divider value used in generating the PWM clock.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.15 pm_lpg_pwm_prediv_exponent_set()

This function sets the clock pre-divider exponent.

Parameters

```
pm_err_flag_type pm_lpg_pwm_prediv_exponent_set
(
    int which_lpg,
    unsigned short pre_div_exponent
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	pre_div_exponent	unsigned short value: ▪ Predivider exponent, 0 to 7

Description

This API sets the divider exponent value used in generating the PWM clock.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the pre_div_exponent parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.16 pm_lpg_pwm_ninebit_enable()

This function sets nine or six bit PWM mode.

Parameters

```
pm_err_flag_type pm_lpg_pwm_ninebit_enable
(
    int which_lpg,
    boolean enable
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Enables 9 bit PWM ▪ FALSE – Enables 6 bit PWM

Description

This API sets the PWM to either 9-bit or 6-bit

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.17 pm_lpg_dtest_config()

This function connects the LPG output to one of the PMIC DTEST inputs.

Parameters

```
pm_err_flag_type pm_lpg_dtest_config
(
    int which_lpg,
    int dtest_config
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	dtest_config	int dtest_config: ▪ DTEST values 0 to 3

Description

This API connects the LPG output to one of the PMIC DTEST inputs.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the dtest_config parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.18 pm_lpg_pwm_bank_enable()

This function enables one of the LPG banks.

Parameters

```
pm_err_flag_type pm_lpg_pwm_bank_enable
(
    int which_lpg,
    boolean enable
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	enable	boolean: ▪ TRUE – Enable ▪ FALSE – Disable

Description

This API enables one of the LPG banks.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

Refer to example in 17.11.2

17.11.19 pm_lpg_lut_config_set()

This function writes a value to the look up table.

Parameters

```
pm_err_flag_type pm_lpg_lut_config_set
(
    int which_lpg,
    int index,
    int value
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	index	int index: ▪ which of the look up table entries to write (0 to 63)
→	value	int value: ▪ the 9-bit value to be written to the look up table

Description

This API writes one value to the LPG look up table.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the index parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the value parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Configure LPG Bank 0 LUT */
for (i = 0; i < PM_LPG_LUT_CFG_MAX; i++)
{
    (void) pm_lpg_lut_config_set(0, i, i << 3);
}
```

17.11.20 pm_lpg_lut_config_get()

API to read one value from the LPG look up table.

Parameters

```
pm_err_flag_type pm_lpg_lut_config_get
(
    int which_lpg,
    int index,
    int *value
)
```

→	which_lpg	▪ int which_lpg: LPG bank 0 to 7
→	index	int index: ▪ which of the look up table entries to write (0 to 63)
→	*value	int pointer value: ▪ The 9-bit value read from to the look up table

Description

This API writes one value to the LPG look up table.

Return value

This function returns pm_err_flag_type:

- PM_ERR_FLAG__SUCCESS – Success
- PM_ERR_FLAG__PAR1_OUT_OF_RANGE – Invalid value for the which_lpg parameter
- PM_ERR_FLAG__PAR2_OUT_OF_RANGE – Invalid value for the index parameter
- PM_ERR_FLAG__PAR3_OUT_OF_RANGE – Invalid value for the value parameter
- PM_ERR_FLAG__SBI_OPT_ERR – Failure to communicate with the PMIC device
- PM_ERR_FLAG__FEATURE_NOT_SUPPORTED – Feature is not available on this PMIC version

Dependencies

The pm_init() function must be invoked before this function can be used.

Side effects

Interrupts are disabled while communicating with the PMIC.

Example

```
/* Get LPG Bank 0 LUT */
int bank0_lut[PM_LPG_LUT_CFG_MAX],
int temp;
for (i = 0; i < PM_LPG_LUT_CFG_MAX; i++)
{
(void) pm_lpg_lut_config_set(0, i, &temp);
bank0_lut[i]=temp;
}
```

A Sleep Clock

A.1 Introduction

The PM8921 has three different clock sources:

- 3.2 MHz from an internal RC oscillator
- 32 kHz using an internal oscillator with an external crystal
- 19.2 MHz using an external TCXO module

The 32.768 kHz crystal oscillator (using an external crystal) is used as a general system clock to run internal state machines and various timers, run the real-time clock, and as a 32 kHz sleep clock to the MSM device through the SLEEP_CLK output pin.

The 19.2 MHz TCXO signal provides the 1.6 MHz clock for the SMPS regulators. It also provides a 19.2 MHz clock to the MSM device through the TCXO_OUT pin, as shown in Figure A-1.

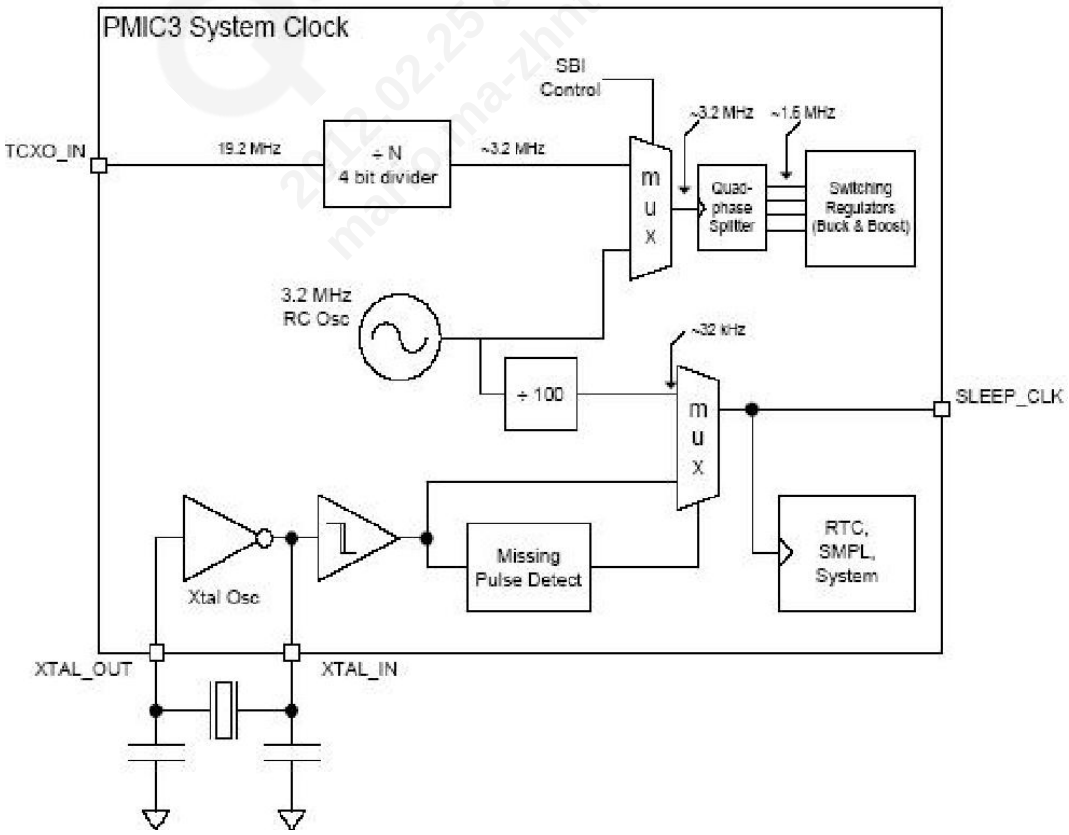

Figure A-1 PM8921 system clock

A.2 SLEEP_CLK as primary clock source

The 32.768 kHz crystal oscillator and the SLEEP_CLK output are enabled by default. However, during normal operation the MSM device will use the signal from the TCXO as the primary clock source. When the phone goes into Power Saving or Sleep mode, it will turn off the TCXO buffer. At this stage, the SLEEP_CLK will become the primary clock source for the MSM device to do time keeping.

A.3 Error handling and recovery

If the 32 kHz crystal oscillator stops for any reason, a missing pulse is detected by the PMIC and will automatically switch through the MUX to the RC oscillator as a backup source. However, the RC oscillator is less accurate for time keeping. Therefore, a recovery mechanism is needed to restart the 32 kHz crystal oscillator. The recovery process is illustrated in Figure A-2.

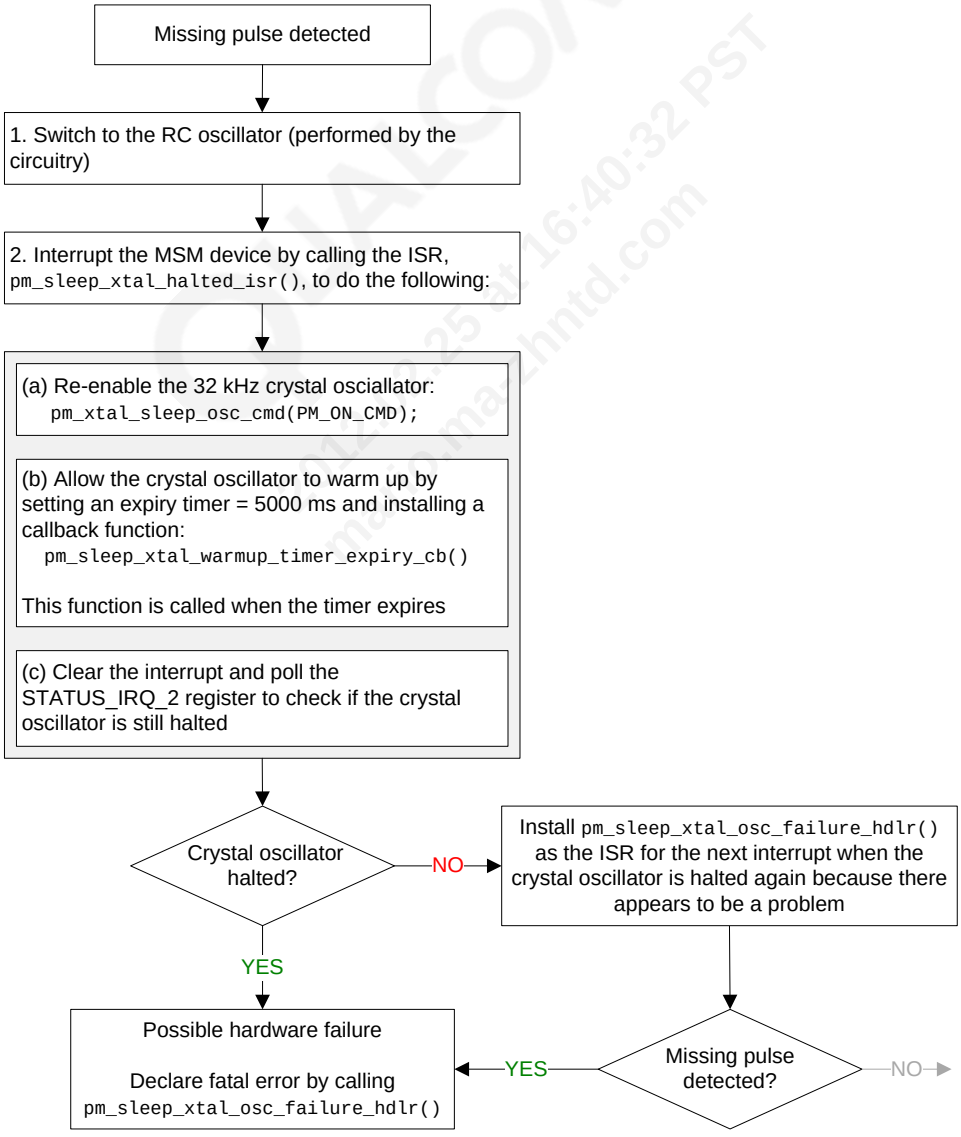

Figure A-2 32 kHz crystal oscillator flow diagram