Intel® Quark™ Microcontroller Software Interface

1.0

Generated by Doxygen 1.8.6

Sun Dec 6 2015 06:31:42

Contents

1 Licensing information

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 Module Documentation

4.1 Always-on Counters

Data Structures

- struct qm_aonpt_config_t

Functions

- void qm_aonpt_isr_0 (void)
- qm_rc_t qm_aonc_enable (const qm_scss_aon_t aonc)
- qm_rc_t qm_aonc_disable (const qm_scss_aon_t aonc)
- uint32_t qm_aonc_get_value (const qm_scss_aon_t aonc)

  Get the current value of the Always-on Counter.
- qm_rc_t qm_aonpt_set_config (const qm_scss_aon_t aonc, const qm_aonpt_config_t *const cfg)

  Set the Always-on Periodic Timer configuration.
- qm_rc_t qm_aonpt_get_config (const qm_scss_aon_t aonc, qm_aonpt_config_t *const cfg)

  Get the Always-on Periodic Timer configuration.
- uint32_t qm_aonpt_get_value (const qm_scss_aon_t aonc)

  Get the current value of the Always-on Periodic Timer.
- bool qm_aonpt_get_status (const qm_scss_aon_t aonc)

  Get the current status of the Always-on Periodic Timer.
- qm_rc_t qm_aonpt_clear (const qm_scss_aon_t aonc)

  Clear the status of the Always-on Periodic Timer.
- qm_rc_t qm_aonpt_reset (const qm_scss_aon_t aonc)

4.1.1 Detailed Description

Always-on Counters for Quark Mictocontrollers

4.1.2 Function Documentation

4.1.2.1 qm_rc_t qm_aonc_disable (const qm_scss_aon_t *aonc*)

Disable the Always-on Counter.

Parameters

in	*aonc*	Always-on counter to read.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 52 of file qm_aon_counters.c.

4.1.2.2 qm_rc_t qm_aonc_enable (const qm_scss_aon_t *aonc*)

Enable the Always-on Counter.

Parameters

in	*aonc*	Always-on counter to read.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 43 of file qm_aon_counters.c.

4.1.2.3 uint32_t qm_aonc_get_value (const qm_scss_aon_t *aonc*)

Get the current value of the Always-on Counter.

Get the current value of the Always-on Counter. Returns a 32-bit value which represents the number of clock cycles since the counter was first enabled.

Parameters

in	*aonc*	Always-on counter to read.

Returns

uint32_t Value of the counter.

Definition at line 61 of file qm_aon_counters.c.

4.1.2.4 qm_rc_t qm_aonpt_clear (const qm_scss_aon_t *aonc*)

Clear the status of the Always-on Periodic Timer.

Clear the status of the Always-on Periodic Timer. The status must be clear before the Always-on Periodic Timer can trigger another interrupt.

Parameters

in	*aonc*	Always-on counter to read.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 111 of file qm_aon_counters.c.

4.1.2.5 qm_rc_t qm_aonpt_get_config (const qm_scss_aon_t *aonc,* qm_aonpt_config_t *const *cfg*)

Get the Always-on Periodic Timer configuration.

Get the Always-on Periodic Timer configuration. This includes the initial value of the Always-on Periodic Timer, the interrupt enable and the callback function that will be run when the timer expiers and an interrupt is triggered. The Periodic Timer is disabled if the counter is set to 0.

Parameters

in	*aonc*	Always-on counter to read.
out	*cfg*	New configuration for the Always-on Periodic Timer.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 84 of file qm_aon_counters.c.

4.1.2.6 bool qm_aonpt_get_status (const qm_scss_aon_t *aonc*)

Get the current status of the Always-on Periodic Timer.

Get the current status of the Always-on Periodic Timer. Returns true if the timer has expired. This will continue to return true until it is cleared with qm_aonpt_clear().

Parameters

in	*aonc*	Always-on counter to read.

Returns

bool Status of the Always-on Periodic Timer.

Definition at line 106 of file qm_aon_counters.c.

4.1.2.7 uint32_t qm_aonpt_get_value (const qm_scss_aon_t *aonc*)

Get the current value of the Always-on Periodic Timer.

Get the current value of the Always-on Periodic Timer. Returns a 32-bit value which represents the number of clock cycles remaining before the timer fires. This is the initial configured number minus the number of cycles that have passed.

Parameters

in	*aonc*	Always-on counter to read.

Returns

uint32_t Value of the Always-on Periodic Timer.

Definition at line 101 of file qm_aon_counters.c.

4.1.2.8 void qm_aonpt_isr_0 (void)

Always-on Periodic Timer Interrupt Service Routine

Definition at line 34 of file qm_aon_counters.c.

4.1.2.9 qm_rc_t qm_aonpt_reset (const qm_scss_aon_t *aonc*)

Reset the Always-on Periodic Timer back to the configured value.

Parameters

in	*aonc*	Always-on counter to read.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 120 of file qm_aon_counters.c.

4.1.2.10 qm_rc_t qm_aonpt_set_config (const qm_scss_aon_t *aonc,* const qm_aonpt_config_t *const *cfg*)

Set the Always-on Periodic Timer configuration.

Set the Always-on Periodic Timer configuration. This includes the initial value of the Always-on Periodic Timer, the interrupt enable and the callback function that will be run when the timer expiers and an interrupt is triggered. The Periodic Timer is disabled if the counter is set to 0.

Parameters

in	*aonc*	Always-on counter to read.
in	*cfg*	New configuration for the Always-on Periodic Timer.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 66 of file qm_aon_counters.c.

4.2 Analog Comparator

Data Structures

- struct qm_ac_config_t

Functions

- void qm_ac_isr (void)
- qm_rc_t qm_ac_get_config (qm_ac_config_t *const config)
- qm_rc_t qm_ac_set_config (const qm_ac_config_t *const config)

4.2.1 Detailed Description

Analog comparator for Quark Microcontrollers.

4.2.2 Function Documentation

4.2.2.1 qm_rc_t qm_ac_get_config (qm_ac_config_t *const *config*)

Get Analog Comparator configuration.

Parameters

in	*config*	Analog Comparator configuration.

Returns

qm_rc_t QM_RC_OK on success, QM_RC_ERR otherwise.

Definition at line 64 of file qm_comparator.c.

4.2.2.2 void qm_ac_isr (void)

Analog Comparator Interrupt Service Routine

Definition at line 34 of file qm_comparator.c.

4.2.2.3 qm_rc_t qm_ac_set_config (const qm_ac_config_t *const *config*)

Set Analog Comparator configuration.

Parameters

in	*config*	Analog Comparator configuration.

Returns

qm_rc_t QM_RC_OK on success, QM_RC_ERR otherwise.

Definition at line 77 of file qm_comparator.c.

4.3 Flash

Flash Controller for QM.

Data Structures

- struct qm_flash_config_t

Enumerations

- enum qm_flash_region_t
- enum qm_flash_disable_t

Functions

- qm_rc_t qm_flash_set_config (const qm_flash_t flash, qm_flash_config_t *cfg)

  Configure a Flash controller.
- qm_rc_t qm_flash_get_config (const qm_flash_t flash, qm_flash_config_t *cfg)

  Get Flash controller configuration.
- qm_rc_t qm_flash_word_write (const qm_flash_t flash, qm_flash_region_t region, uint32_t f_addr, const uint32_t data)

  Write 4 bytes of data to Flash.
- qm_rc_t qm_flash_page_update (const qm_flash_t flash, qm_flash_region_t region, uint32_t f_addr, uint32_t *page_buffer, uint32_t *data_buffer, uint32_t len)

  Write multiple of 4 bytes of data to Flash.
- qm_rc_t qm_flash_page_write (const qm_flash_t flash, qm_flash_region_t region, uint32_t page_num, uint32_t *data, uint32_t len)

  Write s 2KB flash page.
- qm_rc_t qm_flash_page_erase (const qm_flash_t flash, qm_flash_region_t region, uint32_t page_num)

  Erase one page of Flash.
- qm_rc_t qm_flash_mass_erase (const qm_flash_t flash, uint8_t include_rom)

  Perform mass erase.

4.3.1 Detailed Description

Flash Controller for QM. Flash Controller for Quark Microcontrollers.

4.3.2 Enumeration Type Documentation

4.3.2.1 enum qm_flash_disable_t

Flash write disable / enable enum

Definition at line 83 of file qm_flash.h.

4.3.2.2 enum qm_flash_region_t

Flash region enum

Definition at line 71 of file qm_flash.h.

4.3.3 Function Documentation

4.3.3.1 qm_rc_t qm_flash_get_config (const qm_flash_t *flash*, qm_flash_config_t * *cfg*)

Get Flash controller configuration.

Retrieve Flash controller configuration. This will set the cfg parameter to match the current configuration of the given Flash controller.

Parameters

in	*flash*	Flash controller index.
out	*cfg*	Flash configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 53 of file qm_flash.c.

References qm_flash_config_t::us_count, qm_flash_config_t::wait_states, and qm_flash_config_t::write_disable.

4.3.3.2 qm_rc_t qm_flash_mass_erase (const qm_flash_t *flash*, uint8_t *include_rom*)

Perform mass erase.

Perform Flash mass erase. Check for brownout before initiating the erase. Performs mass erase on the Flash controller. The mass erase may include the ROM region, if present and unlocked. Note it is not possible to mass-erase the ROM portion separately.

Parameters

in	*flash*	Flash controller index.
in	*include_rom*	If set, it also erases the ROM region.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 309 of file qm_flash.c.

4.3.3.3 qm_rc_t qm_flash_page_erase (const qm_flash_t *flash*, qm_flash_region_t *region*, uint32_t *page_num*)

Erase one page of Flash.

Erase one page of Flash. Check for brownout before initiating the write.

Parameters

in	*flash*	Flash controller index.
in	*region*	Flash region to address.
in	*page_num*	Page within the Flash controller to erase.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 274 of file qm_flash.c.

4.3.3.4 qm_rc_t qm_flash_page_update (const qm_flash_t *flash*, qm_flash_region_t *region*, uint32_t *f_addr*, uint32_t * *page_buffer*, uint32_t * *data_buffer*, uint32_t *len*)

Write multiple of 4 bytes of data to Flash.

Write a multiple of 4 bytes of data to Flash. Check for brownout before initiating the write. The page is erased, and then written to.

Parameters

in	*flash*	Flash controller index.
in	*region*	Which Flash region to address.
in	*f_addr*	Address within Flash physical address space.
in	*page_buffer*	Page buffer to store page, must be at least QM_FLASH_PAGE_SIZE words big.
in	*data_buffer*	Data buffer to write.
in	*len*	Length of data to write.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 179 of file qm_flash.c.

4.3.3.5 qm_rc_t qm_flash_page_write (const qm_flash_t *flash*, qm_flash_region_t *region*, uint32_t *page_num*, uint32_t * *data*, uint32_t *len*)

Write s 2KB flash page.

Write a 2KB page of Flash. Check for brownout before initiating the write. The page is erased, and then written to.

Parameters

in	*flash*	Flash controller index.
in	*region*	Which Flash region to address.
in	*page_num*	Which page of flash to overwrite.
in	*data*	Data buffer to write.
in	*len*	Length of data to write.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 120 of file qm_flash.c.

4.3.3.6 qm_rc_t qm_flash_set_config (const qm_flash_t *flash*, qm_flash_config_t * *cfg*)

Configure a Flash controller.

Configure a Flash controller. This includes timing and behavioral settings. When switching SoC to a higher frequency, this register must be updated first to reflect settings associated with higher frequency BEFORE SoC frequency is changed. On the other hand, when switching SoC to a lower frequency, this register must be updated only 6 NOP instructions AFTER the SoC frequency has been updated. Otherwise, flash timings will be violated.

Parameters

in	*flash*	Flash controller index.
in	*cfg*	Flash configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 32 of file qm_flash.c.

References qm_flash_config_t::us_count, qm_flash_config_t::wait_states, and qm_flash_config_t::write_disable.

4.3.3.7 qm_rc_t qm_flash_word_write (const qm_flash_t *flash*, qm_flash_region_t *region*, uint32_t *f_addr*, const uint32_t *data*)

Write 4 bytes of data to Flash.

Write 4 bytes of data to Flash. Check for brownout before initiating the write. Note this function performs a write operation only; page erase may be needed if the page is already programmed.

Parameters

in	*flash*	Flash controller index.
in	*region*	Flash region to address.
in	*f_addr*	Address within Flash physical address space.
in	*data*	Data word to write.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 70 of file qm_flash.c.

4.4 FPR

Flash Protection Region for QM.

Data Structures

- struct qm_fpr_config_t

Macros

- #define QM_FPR_EN_MASK_ENABLE BIT(0)
- #define QM_FPR_EN_MASK_LOCK BIT(1)
- #define QM_FPR_AGENT_MASK_HOST BIT(0)
- #define QM_FPR_AGENT_MASK_SS BIT(1)
- #define QM_FPR_AGENT_MASK_DMA BIT(2)
- #define QM_FPR_AGENT_MASK_OTHER BIT(3)

Enumerations

- enum qm_fpr_id_t

Functions

- void qm_fpr_isr_0 (void)
- qm_rc_t qm_fpr_set_config (const qm_flash_t flash, const qm_fpr_id_t id, const qm_fpr_config_t *const cfg, const qm_flash_region_type_t region)
- qm_rc_t qm_fpr_get_config (const qm_flash_t flash, const qm_fpr_id_t id, qm_fpr_config_t *const cfg, const qm_flash_region_type_t region)

  Get Flash FPR configuration.
- qm_rc_t qm_fpr_set_violation_policy (const qm_fpr_viol_mode_t mode, const qm_flash_t flash, qm_fpr_callback_t fpr_cb)

4.4.1 Detailed Description

Flash Protection Region for QM. Flash Protection Region control for Quark Microcontrollers.

4.4.2 Macro Definition Documentation

4.4.2.1 #define QM_FPR_AGENT_MASK_DMA BIT(2)

FPR mask dma

Definition at line 119 of file qm_fpr.h.

4.4.2.2 #define QM_FPR_AGENT_MASK_HOST BIT(0)

FPR mask host

Definition at line 115 of file qm_fpr.h.

4.4.2.3 #define QM_FPR_AGENT_MASK_OTHER BIT(3)

FPR mask other agents

Definition at line 121 of file qm_fpr.h.

4.4.2.4 #define QM_FPR_AGENT_MASK_SS BIT(1)

FPR mask ss

Definition at line 117 of file qm_fpr.h.

4.4.2.5 #define QM_FPR_EN_MASK_ENABLE BIT(0)

FPR enable mask

Definition at line 111 of file qm_fpr.h.

4.4.2.6 #define QM_FPR_EN_MASK_LOCK BIT(1)

FPR mask lock

Definition at line 113 of file qm_fpr.h.

4.4.3 Enumeration Type Documentation

4.4.3.1 enum qm_fpr_id_t

FPR register map.

Definition at line 54 of file qm_fpr.h.

4.4.4 Function Documentation

4.4.4.1 qm_rc_t qm_fpr_get_config (const qm_flash_t *flash*, const qm_fpr_id_t *id*, qm_fpr_config_t *const *cfg*, const qm_flash_region_type_t *region*)

Get Flash FPR configuration.

Retrieve Flash controller's Flash Protection Region configuration. This will set the cfg parameter to match the current configuration of the given Flash controller's FPR.

Parameters

in	*flash*	Which Flash to read the configuration of.
in	*id*	FPR identifier.
out	*cfg*	FPR configuration.
in	*region*	The region of Flash configured.

Returns

RC_OK on success, error code otherwise.

Definition at line 82 of file qm_fpr.c.

References qm_fpr_config_t::allow_agents, qm_fpr_config_t::en_mask, qm_fpr_config_t::low_bound, and qm_fpr_config_t::up_bound.

4.4.4.2 void qm_fpr_isr_0 (void)

FPR Interrupt Service Routines

Definition at line 34 of file qm_fpr.c.

4.4.4.3 qm_rc_t qm_fpr_set_config (const qm_flash_t *flash*, const qm_fpr_id_t *id*, const qm_fpr_config_t *const *cfg*, const qm_flash_region_type_t *region*)

Configure a Flash controller's Flash Protection Region.

Parameters

in	*flash*	Which Flash controller to configure.
in	*id*	FPR identifier.
in	*cfg*	FPR configuration.
in	*region*	The region of Flash to be configured.

Returns

RC_OK on success, error code otherwise.

Definition at line 48 of file qm_fpr.c.

References qm_fpr_config_t::allow_agents, qm_fpr_config_t::en_mask, qm_fpr_config_t::low_bound, and qm_fpr_config_t::up_bound.

4.4.4.4 qm_rc_t qm_fpr_set_violation_policy (const qm_fpr_viol_mode_t *mode*, const qm_flash_t *flash*, qm_fpr_callback_t *fpr_cb*)

Configure FPR violation behaviour

Parameters

in	*mode*	(generate interrupt, warm reset, enter probe mode).
in	*fpr_cb*	for interrupt mode (only). This cannot be null.
in	*flash*	controller.

Returns

RC_OK on success, error code otherwise.

Definition at line 119 of file qm_fpr.c.

4.5 GPIO

Data Structures

- struct qm_gpio_port_config_t

Functions

- void qm_gpio_isr_0 (void)
- qm_rc_t qm_gpio_set_config (const qm_gpio_t gpio, const qm_gpio_port_config_t *const cfg)
  Set GPIO port configuration.
- qm_rc_t qm_gpio_get_config (const qm_gpio_t gpio, qm_gpio_port_config_t *const cfg)
  Get GPIO port configuration.
- bool qm_gpio_read_pin (const qm_gpio_t gpio, const uint8_t pin)
- qm_rc_t qm_gpio_set_pin (const qm_gpio_t gpio, const uint8_t pin)
- qm_rc_t qm_gpio_clear_pin (const qm_gpio_t gpio, const uint8_t pin)
- uint32_t qm_gpio_read_port (const qm_gpio_t gpio)
  Get GPIO port values.
- qm_rc_t qm_gpio_write_port (const qm_gpio_t gpio, const uint32_t val)
  Get GPIO port values.

4.5.1 Detailed Description

General Purpose IO for Quark Microcontrollers.

4.5.2 Function Documentation

4.5.2.1 qm_rc_t qm_gpio_clear_pin (const qm_gpio_t *gpio*, const uint8_t *pin*)

Clear a single pin on a given GPIO port.

Parameters

in	*gpio*	GPIO port index.
in	*pin*	Pin of GPIO port to clear.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 100 of file qm_gpio.c.

4.5.2.2 qm_rc_t qm_gpio_get_config (const qm_gpio_t *gpio*, qm_gpio_port_config_t *const *cfg*)

Get GPIO port configuration.

Get GPIO port configuration. This includes if interrupts are enabled or not, the level on which an interrupt is generated, the polarity of interrupts and if GPIO-debounce is enabled or not.

Parameters

in	*gpio*	GPIO port index to read the configuration of.
out	*cfg*	Current configuration for GPIO port.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 68 of file qm_gpio.c.

4.5.2.3 void qm_gpio_isr_0 (void)

GPIO Interrupt Service Routine

Definition at line 34 of file qm_gpio.c.

4.5.2.4 bool qm_gpio_read_pin (const qm_gpio_t *gpio*, const uint8_t *pin*)

Read the current value of a single pin on a given GPIO port.

Parameters

in	*gpio*	GPIO port index.
in	*pin*	Pin of GPIO port to read.

Returns

bool Value of the pin specified on GPIO port.

Definition at line 85 of file qm_gpio.c.

4.5.2.5 uint32_t qm_gpio_read_port (const qm_gpio_t *gpio*)

Get GPIO port values.

Read entire GPIO port. Each bit of the val parameter is set to the current value of each pin on the port. Maximum 32 pins per port.

Parameters

in	*gpio*	GPIO port index.

Returns

uint32_t Value of all pins on GPIO port.

Definition at line 110 of file qm_gpio.c.

4.5.2.6 qm_rc_t qm_gpio_set_config (const qm_gpio_t *gpio*, const qm_gpio_port_config_t *const *cfg*)

Set GPIO port configuration.

Set GPIO port configuration. This includes if interrupts are enabled or not, the level on which an interrupt is generated, the polarity of interrupts and if GPIO-debounce is enabled or not. If interrupts are enabled it also registers an ISR with the user defined callback function.

Parameters

in	*gpio*	GPIO port index to configure.
in	*cfg*	New configuration for GPIO port.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 46 of file qm_gpio.c.

4.5.2.7 qm_rc_t qm_gpio_set_pin (const qm_gpio_t *gpio*, const uint8_t *pin*)

Set a single pin on a given GPIO port.

Parameters

in	*gpio*	GPIO port index.
in	*pin*	Pin of GPIO port to set.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 90 of file qm_gpio.c.

4.5.2.8 qm_rc_t qm_gpio_write_port (const qm_gpio_t *gpio*, const uint32_t *val*)

Get GPIO port values.

Write entire GPIO port. Each pin on the GPIO port is set to the corresponding value set in the val parameter. Maximum 32 pins per port.

Parameters

in	*gpio*	GPIO port index.
in	*val*	Value of all pins on GPIO port.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 115 of file qm_gpio.c.

4.6 I2C

Data Structures

- struct qm_i2c_config_t
- struct qm_i2c_transfer_t

Enumerations

- enum qm_i2c_addr_t
- enum qm_i2c_mode_t
- enum qm_i2c_speed_t
- enum qm_i2c_status_t

Functions

- void qm_i2c_0_isr (void)
- void qm_i2c_1_isr (void)
- qm_rc_t qm_i2c_set_config (const qm_i2c_t i2c, const qm_i2c_config_t *const cfg)
- qm_rc_t qm_i2c_get_config (const qm_i2c_t i2c, qm_i2c_config_t *const cfg)
- qm_rc_t qm_i2c_set_speed (const qm_i2c_t i2c, qm_i2c_speed_t speed, uint16_t lo_cnt, uint16_t hi_cnt)

  Set I2C speed.
- qm_i2c_status_t qm_i2c_get_status (const qm_i2c_t i2c)
- qm_rc_t qm_i2c_master_write (const qm_i2c_t i2c, const uint16_t slave_addr, const uint8_t *const data, uint32_t len, bool stop)

  Master write on I2C.
- qm_rc_t qm_i2c_master_read (const qm_i2c_t i2c, const uint16_t slave_addr, uint8_t *const data, uint32_t len, bool stop)

  Master read of I2C.
- qm_rc_t qm_i2c_master_irq_transfer (const qm_i2c_t i2c, const qm_i2c_transfer_t *const xfer, const uint16_t slave_addr)

  Interrupt based master transfer on I2C.
- qm_rc_t qm_i2c_transfer_terminate (const qm_i2c_t i2c)

  Terminate I2C IRQ/DMA transfer.

4.6.1 Detailed Description

I2C driver for Quark Microcontrollers.

4.6.2 Enumeration Type Documentation

4.6.2.1 enum qm_i2c_addr_t

QM I2C addressing type.

Definition at line 56 of file qm_i2c.h.

4.6.2.2 enum qm_i2c_mode_t

QM I2C master / slave mode type.

Definition at line 61 of file qm_i2c.h.

4.6.2.3 enum qm_i2c_speed_t

QM I2C Speed Type.

Definition at line 66 of file qm_i2c.h.

4.6.2.4 enum qm_i2c_status_t

I2C status type.

Definition at line 75 of file qm_i2c.h.

4.6.3 Function Documentation

4.6.3.1 void qm_i2c_0_isr (void)

I2C 0 Interrupt Service Routine.

Definition at line 192 of file qm_i2c.c.

4.6.3.2 void qm_i2c_1_isr (void)

I2C 1 Interrupt Service Routine.

4.6.3.3 qm_rc_t qm_i2c_get_config (const qm_i2c_t *i2c*, qm_i2c_config_t *const *cfg*)

Retrieve I2C configuration.

Parameters

in	*i2c*	Which I2C to read the configuration of.
out	*cfg*	I2C configuration.

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Definition at line 339 of file qm_i2c.c.

4.6.3.4 qm_i2c_status_t qm_i2c_get_status (const qm_i2c_t *i2c*)

Retrieve I2C status.

Parameters

in	*i2c*	Which I2C to read the status of.

Returns

qm_i2c_status_t Returns Free of Busy.

Definition at line 400 of file qm_i2c.c.

4.6.3.5 qm_rc_t qm_i2c_master_irq_transfer (const qm_i2c_t *i2c*, const qm_i2c_transfer_t *const *xfer*, const uint16_t *slave_addr*)

Interrupt based master transfer on I2C.

Perform an interrupt based master transfer on the I2C bus. The function will replenish/empty TX/RX FIFOs on I2C empty/full interrupts.

Parameters

in	*i2c*	Which I2C to transfer from.
in	*xfer*	Transfer structure includes write / read data and length, write, read and error callback functions and a callback identifier.
in	*slave_addr*	Address of slave to transfer data with.

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Definition at line 554 of file qm_i2c.c.

4.6.3.6 qm_rc_t qm_i2c_master_read (const qm_i2c_t *i2c*, const uint16_t *slave_addr*, uint8_t *const *data*, uint32_t *len*, bool *stop*)

Master read of I2C.

Perform a single byte master read from the I2C. This is a blocking call.

Parameters

in	*i2c*	Which I2C to read from.
in	*slave_addr*	Address of slave device to read from.
out	*data*	Pre-allocated buffer to populate with data.
in	*len*	length of data to read from slave.
in	*stop*	Generate a STOP condition at the end of Tx

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Definition at line 461 of file qm_i2c.c.

4.6.3.7 qm_rc_t qm_i2c_master_write (const qm_i2c_t *i2c*, const uint16_t *slave_addr*, const uint8_t *const *data*, uint32_t *len*, bool *stop*)

Master write on I2C.

Perform a master write on the I2C bus. This is a blocking synchronous call.

Parameters

in	*i2c*	Which I2C to write to.
in	*slave_addr*	Address of slave to write to.
in	*data*	Pre-allocated buffer of data to write.
in	*len*	length of data to write.
in	*stop*	Generate a STOP condition at the end of Tx

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Definition at line 410 of file qm_i2c.c.

4.6.3.8 qm_rc_t qm_i2c_set_config (const qm_i2c_t *i2c*, const qm_i2c_config_t *const *cfg*)

Set I2C configuration.

Parameters

in	*i2c*	Which I2C to set the configuration of.
out	*cfg*	I2C configuration.

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Definition at line 233 of file qm_i2c.c.

4.6.3.9 qm_rc_t qm_i2c_set_speed (const qm_i2c_t *i2c*, qm_i2c_speed_t *speed*, uint16_t *lo_cnt*, uint16_t *hi_cnt*)

Set I2C speed.

Fine tune I2C clock speed. This will set the SCL low count and the SCL hi count cycles. To achieve any required speed.

Parameters

in	*i2c*	I2C index.
in	*speed*	Bus speed (Standard or Fast. Fast includes Fast+ mode)
in	*lo_cnt*	SCL low count.
in	*hi_cnt*	SCL high count.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 367 of file qm_i2c.c.

4.6.3.10 qm_rc_t qm_i2c_transfer_terminate (const qm_i2c_t *i2c*)

Terminate I2C IRQ/DMA transfer.

Terminate the current IRQ or DMA transfer on the I2C bus. This will cause the error callback to be called with status QM_I2C_TX_ABRT_USER_ABRT.

Parameters

in	*i2c*	I2C register block pointer.

Returns

qm_rc_t Returns QM_QM_RC_OK on success, error code otherwise.

Definition at line 624 of file qm_i2c.c.

4.7 Identification

Functions

- uint32_t qm_soc_id (void)
- uint32_t qm_soc_version (void)

4.7.1 Detailed Description

Identification functions for Quark Microcontrollers.

4.7.2 Function Documentation

4.7.2.1 uint32_t qm_soc_id (void)

Get Quark SoC identification number

Returns

uint32_t SoC identifier number.

Definition at line 35 of file qm_identification.c.

4.7.2.2 uint32_t qm_soc_version (void)

Get Quark SoC version number

Returns

uint32_t SoC version number.

Definition at line 46 of file qm_identification.c.

4.8 Initialisation

Enumerations

- enum qm_soc_reset_t

Functions

- void qm_soc_reset (qm_soc_reset_t reset_type)

4.8.1 Detailed Description

Initialisation and reset for Quark Microcontrollers.

4.8.2 Enumeration Type Documentation

4.8.2.1 enum qm_soc_reset_t

Reset Mode type.

Definition at line 46 of file qm_init.h.

4.8.3 Function Documentation

4.8.3.1 void qm_soc_reset (qm_soc_reset_t *reset_type*)

Reset the SoC. This can either be a cold reset or a warm reset.

Parameters

in	*reset_type*	Selects the type of reset to perform.

Definition at line 32 of file qm_init.c.

4.9 Interrupt

Interrupt for QM.

Macros

- #define qm_irq_request(irq, isr)

  Request an IRQ and attach an ISR to it.
- #define qm_int_vector_request(vector, isr)

  Request an interrupt vector and attach an ISR to it.

Typedefs

- typedef void(* qm_isr_t)(void)

Functions

- void qm_irq_enable (void)
- void qm_irq_disable (void)
- void qm_irq_unmask (uint32_t irq)
- void qm_irq_mask (uint32_t irq)
- void _qm_register_isr (uint32_t vector, qm_isr_t isr)
- void _qm_irq_setup (uint32_t irq, uint16_t register_offset)

4.9.1 Detailed Description

Interrupt for QM. Interrupt driver for Quark Microcontrollers.

4.9.2 Macro Definition Documentation

4.9.2.1 #define qm_int_vector_request(*vector, isr*)

Value:

```
do {                                                                \
                __asm__ __volatile__("push $1f\n\t"                 \
                                     "push %0\n\t"                  \
                                     "call %P1\n\t"                 \
                                     "add $8, %%esp\n\t"            \
                                     "jmp 2f\n\t"                   \
                                     ".align 4\n\t"                 \
                                     "1:\n\t"                       \
                                     "         pushal\n\t"          \
                                     "         call %P2\n\t"        \
                                     "         movl $0x0, (%P3)\n\t"\
                                     "         popal\n\t"           \
                                     "         iret\n\t"            \
                                     "2:\n\t" ::"g"(vector),        \
                                     "i"(_qm_register_isr), "i"(isr),\
                                     "i"(&(QM_INT_CONTROLLER->eoi.reg))\
                                     : "%eax", "%ecx", "%edx");     \
        } while (0)
```

Request an interrupt vector and attach an ISR to it.

Request an interrupt vector and register Interrupt Service Routine to it.

Parameters

in	*vector*	Vector number.
in	*isr*	ISR to register to given IRQ.

Definition at line 154 of file qm_interrupt.h.

4.9.2.2 #define qm_irq_request(*irq*, *isr*)

Value:

```
do {                                                                \
            qm_int_vector_request(irq##_VECTOR, isr);
      \
                                                                    \
            _qm_irq_setup(irq, irq##_MASK_OFFSET);                  \
      } while (0)
```

Request an IRQ and attach an ISR to it.

Request a given IRQ and register Interrupt Service Routine to interrupt vector.

Parameters

in	*irq*	IRQ number. Must be of type QM_IRQ_XXX.
in	*isr*	ISR to register to given IRQ.

Definition at line 114 of file qm_interrupt.h.

4.9.3 Typedef Documentation

4.9.3.1 typedef void(* qm_isr_t)(void)

Interrupt service routine type

Definition at line 58 of file qm_interrupt.h.

4.9.4 Function Documentation

4.9.4.1 void _qm_irq_setup (uint32_t *irq*, uint16_t *register_offset*)

Setup an IRQ and its routing on the Interrupt Controller.

Parameters

in	*irq*	IRQ number. Must be of type QM_IRQ_XXX.
in	*register_offset*	Interrupt Mask Register offset on SCSS. Must be of type QM_IRQ_XXX_MASK_OFFSET.

Definition at line 86 of file qm_interrupt.c.

4.9.4.2 void _qm_register_isr (uint32_t *vector*, qm_isr_t *isr*)

Register an Interrupt Service Routine to a given interrupt vector.

Parameters

in	*vector*	Interrupt Vector number.
in	*isr*	ISR to register to given vector. Must be a valid x86 ISR. If this can't be provided, qm_irq_request() or qm_int_vector_request() should be used instead.

Definition at line 128 of file qm_interrupt.c.

4.9.4.3 void qm_irq_disable (void)

Disable interrupt delivery for the SoC.

Definition at line 50 of file qm_interrupt.c.

4.9.4.4 void qm_irq_enable (void)

Enable interrupt delivery for the SoC.

Definition at line 55 of file qm_interrupt.c.

4.9.4.5 void qm_irq_mask (uint32_t *irq*)

Mask a given interrupt line.

Parameters

in	*irq*	Which IRQ to mask.

Definition at line 60 of file qm_interrupt.c.

4.9.4.6 void qm_irq_unmask (uint32_t *irq*)

Unmask a given interrupt line.

Parameters

in	*irq*	Which IRQ to unmask.

Definition at line 73 of file qm_interrupt.c.

4.10 MPR

Data Structures

- struct qm_mpr_config_t

Macros

- #define QM_SRAM_MPR_EN_MASK_ENABLE BIT(0)
- #define QM_SRAM_MPR_EN_MASK_LOCK BIT(1)
- #define QM_SRAM_MPR_AGENT_MASK_HOST BIT(0)
- #define QM_SRAM_MPR_AGENT_MASK_DMA BIT(2)

Functions

- void qm_mpr_isr (void)
- qm_rc_t qm_mpr_set_config (const qm_mpr_id_t id, const qm_mpr_config_t *const cfg)
- qm_rc_t qm_mpr_get_config (const qm_mpr_id_t id, qm_mpr_config_t *const cfg)

  Get MPR configuration.
- qm_rc_t qm_mpr_set_violation_policy (const qm_mpr_viol_mode_t mode, qm_mpr_callback_t callback_fn)

4.10.1 Detailed Description

Memory Protection Region control for Quark Microcontrollers.

4.10.2 Macro Definition Documentation

4.10.2.1 #define QM_SRAM_MPR_AGENT_MASK_DMA BIT(2)

MPR mask ss MPR mask dma

Definition at line 54 of file qm_mpr.h.

4.10.2.2 #define QM_SRAM_MPR_AGENT_MASK_HOST BIT(0)

MPR mask host

Definition at line 48 of file qm_mpr.h.

4.10.2.3 #define QM_SRAM_MPR_EN_MASK_ENABLE BIT(0)

MPR mask enable

Definition at line 44 of file qm_mpr.h.

4.10.2.4 #define QM_SRAM_MPR_EN_MASK_LOCK BIT(1)

MPR mask lock

Definition at line 46 of file qm_mpr.h.

4.10.3 Function Documentation

4.10.3.1 qm_rc_t qm_mpr_get_config (const qm_mpr_id_t *id,* qm_mpr_config_t *const *cfg*)

Get MPR configuration.

Retrieve SRAM controller's Memory Protection Region configuration. This will set the cfg parameter to match the current configuration of the SRAM controller's MPR.

Parameters

in	*id*	Which MPR to get configuration of.
out	*cfg*	MPR configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 66 of file qm_mpr.c.

References qm_mpr_config_t::agent_read_en_mask, qm_mpr_config_t::agent_write_en_mask, qm_mpr_config_t::en_lock_mask, qm_mpr_config_t::low_bound, and qm_mpr_config_t::up_bound.

4.10.3.2 void qm_mpr_isr (void)

MPR Interrupt Service Routine

Definition at line 38 of file qm_mpr.c.

4.10.3.3 qm_rc_t qm_mpr_set_config (const qm_mpr_id_t *id,* const qm_mpr_config_t *const *cfg*)

Configure SRAM controller's Memory Protection Region.

Parameters

in	*id*	Which MPR to configure.
in	*cfg*	MPR configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 44 of file qm_mpr.c.

References qm_mpr_config_t::agent_read_en_mask, qm_mpr_config_t::agent_write_en_mask, qm_mpr_config_t::en_lock_mask, qm_mpr_config_t::low_bound, and qm_mpr_config_t::up_bound.

4.10.3.4 qm_rc_t qm_mpr_set_violation_policy (const qm_mpr_viol_mode_t *mode,* qm_mpr_callback_t *callback_fn*)

Configure MPR violation behaviour

Parameters

in	*mode*	(generate interrupt, warm reset, enter probe mode).
in	*callback_fn*	for interrupt mode (only). This can not be null.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 87 of file qm_mpr.c.

4.11 PIC Timer

Data Structures

- struct qm_pic_timer_config_t

Enumerations

- enum qm_pic_timer_mode_t

Functions

- void qm_pic_timer_isr (void)
- qm_rc_t qm_pic_timer_set_config (const qm_pic_timer_config_t *const cfg)
  Set the PIC timer configuration.
- qm_rc_t qm_pic_timer_get_config (qm_pic_timer_config_t *const cfg)
  Get PIC timer configuration.
- qm_rc_t qm_pic_timer_set (const uint32_t count)
- uint32_t qm_pic_timer_get (void)

4.11.1 Detailed Description

PIC timer for Quark Microcontrollers.

4.11.2 Enumeration Type Documentation

4.11.2.1 enum qm_pic_timer_mode_t

PIC timer mode type.

Definition at line 46 of file qm_pic_timer.h.

4.11.3 Function Documentation

4.11.3.1 uint32_t qm_pic_timer_get (void)

Get the current count value of the PIC timer.

Returns

uint32_t Returns current PIC timer count value.

Definition at line 106 of file qm_pic_timer.c.

4.11.3.2 qm_rc_t qm_pic_timer_get_config (qm_pic_timer_config_t *const *cfg*)

Get PIC timer configuration.

Get PIC timer configuration. Populate the cfg parameter to match the current configuration of the PIC timer.

Parameters

out	*cfg*	PIC timer configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 86 of file qm_pic_timer.c.

References qm_pic_timer_config_t::callback, qm_pic_timer_config_t::int_en, and qm_pic_timer_config_t::mode.

4.11.3.3 void qm_pic_timer_isr (void)

PIC timer Interrupt Service Routine

Definition at line 52 of file qm_pic_timer.c.

4.11.3.4 qm_rc_t qm_pic_timer_set (const uint32_t *count*)

Set the current count value of the PIC timer. A value equal to 0 effectively stops the timer.

Parameters

in	*count*	Value to load the timer with.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 100 of file qm_pic_timer.c.

4.11.3.5 qm_rc_t qm_pic_timer_set_config (const qm_pic_timer_config_t *const *cfg*)

Set the PIC timer configuration.

Set the PIC timer configuration. This includes timer mode and if interrupts are enabled. If interrupts are enabled, it will configure the callback function.

Parameters

in	*cfg*	PIC timer configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 59 of file qm_pic_timer.c.

References qm_pic_timer_config_t::callback, qm_pic_timer_config_t::int_en, and qm_pic_timer_config_t::mode.

4.12 Pin Muxing setup

Enumerations

- enum qm_pmux_fn_t
- enum qm_pmux_slew_t
- enum qm_pin_id_t

Functions

- qm_rc_t qm_pmux_select (qm_pin_id_t pin, qm_pmux_fn_t fn)
- qm_rc_t qm_pmux_set_slew (qm_pin_id_t pin, qm_pmux_slew_t slew)
- qm_rc_t qm_pmux_input_en (qm_pin_id_t pin, bool enable)
- qm_rc_t qm_pmux_pullup_en (qm_pin_id_t pin, bool enable)

4.12.1 Detailed Description

Pin muxing configuration for Quark Microcontrollers.

4.12.2 Enumeration Type Documentation

4.12.2.1 enum qm_pin_id_t

External Pad pin identifiers

Definition at line 70 of file qm_pinmux.h.

4.12.2.2 enum qm_pmux_fn_t

Pin function type.

Definition at line 46 of file qm_pinmux.h.

4.12.2.3 enum qm_pmux_slew_t

Pin slew rate setting.

Definition at line 56 of file qm_pinmux.h.

4.12.3 Function Documentation

4.12.3.1 qm_rc_t qm_pmux_input_en (qm_pin_id_t *pin,* bool *enable*)

Enable input for a pin in the pin mux controller.

Parameters

in	*pin*	which pin to configure.
in	*enable*	set to true to enable input.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 74 of file qm_pinmux.c.

4.12.3.2 qm_rc_t qm_pmux_pullup_en (qm_pin_id_t *pin,* bool *enable*)

Enable pullup for a pin in the pin mux controller.

Parameters

in	*pin*	which pin to configure.
in	*enable*	set to true to enable pullup.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 89 of file qm_pinmux.c.

4.12.3.3 qm_rc_t qm_pmux_select (qm_pin_id_t *pin,* qm_pmux_fn_t *fn*)

Set up pin muxing for a SoC pin. Select one of the pin functions.

Parameters

in	*pin*	which pin to configure.
in	*fn*	the function to assign to the pin.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 46 of file qm_pinmux.c.

4.12.3.4 qm_rc_t qm_pmux_set_slew (qm_pin_id_t *pin,* qm_pmux_slew_t *slew*)

Set up pin's slew rate in the pin mux controller.

Parameters

in	*pin*	which pin to configure.
in	*slew*	the slew rate to assign to the pin.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 60 of file qm_pinmux.c.

4.13 Power state

Functions

- void cpu_halt (void)
- void soc_sleep ()

  Put SoC to sleep.
- void soc_deep_sleep ()

4.13.1 Detailed Description

Power mode control for Quark Microcontrollers.

4.13.2 Function Documentation

4.13.2.1 void cpu_halt (void)

Halts the CPU until next interrupt or reset.

Definition at line 42 of file qm_power.c.

4.13.2.2 void soc_deep_sleep ( )

Enter into deep sleep mode. All clocks are gated. The only way to return from this is to have an interrupt trigger on the low power comparators.

4.13.2.3 void soc_sleep ( )

Put SoC to sleep.

Enter into sleep mode. The hybrid oscillator is disabled, most peripherals are disabled and the voltage regulator is set into retention mode. The following peripherals are disabled in this mode:

- I2C
- SPI
- GPIO debouncing
- Watchdog timer
- PWM / Timers
- UART

The SoC operates from the 32 kHz clock source and the following peripherals may bring the SoC back into an active state:

- GPIO interrupts
- AON Timers
- RTC
- Low power comparators

4.14 PWM / Timer

Data Structures

- struct qm_pwm_config_t

Enumerations

- enum qm_pwm_mode_t

Functions

- void qm_pwm_isr_0 (void)
- qm_rc_t qm_pwm_set_config (const qm_pwm_t pwm, const qm_pwm_id_t id, const qm_pwm_config_t *const cfg)

  Set PWM channel configuration.
- qm_rc_t qm_pwm_get_config (const qm_pwm_t pwm, const qm_pwm_id_t id, qm_pwm_config_t *const cfg)

  Get PWM channel configuration.
- qm_rc_t qm_pwm_set (const qm_pwm_t pwm, const qm_pwm_id_t id, const uint32_t lo_count, const uint32_t hi_count)

  Set PWM period counts.
- qm_rc_t qm_pwm_get (const qm_pwm_t pwm, const qm_pwm_id_t id, uint32_t *const lo_count, uint32_t *const hi_count)
- qm_rc_t qm_pwm_start (const qm_pwm_t pwm, const qm_pwm_id_t id)
- qm_rc_t qm_pwm_stop (const qm_pwm_t pwm, const qm_pwm_id_t id)

4.14.1 Detailed Description

Pulse width modulation and Timer driver for Quark Microcontrollers.

4.14.2 Enumeration Type Documentation

4.14.2.1 enum qm_pwm_mode_t

PWM operating mode type.

Definition at line 50 of file qm_pwm.h.

4.14.3 Function Documentation

4.14.3.1 qm_rc_t qm_pwm_get (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*, uint32_t *const *lo_count*, uint32_t *const *hi_count*)

Get the current period values of a PWM channel.

Parameters

in	*pwm*	Which PWM module to get the count of.
in	*id*	PWM channel id to read the values of.
out	*lo_count*	Num of cycles the output is driven low.

out	*hi_count*	Num of cycles the output is driven high.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 128 of file qm_pwm.c.

4.14.3.2 qm_rc_t qm_pwm_get_config (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*, qm_pwm_config_t *const *cfg*)

Get PWM channel configuration.

Get the current configuration of a PWM channel. This includes low period load value, high period load value, interrupt enable/disable.

Parameters

in	*pwm*	Which PWM module to get the configuration of.
in	*id*	PWM channel id to get the configuration of.
out	*cfg*	Current configuration for PWM.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 89 of file qm_pwm.c.

4.14.3.3 void qm_pwm_isr_0 (void)

PWM Interrupt Service Routine

Definition at line 34 of file qm_pwm.c.

4.14.3.4 qm_rc_t qm_pwm_set (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*, const uint32_t *lo_count*, const uint32_t *hi_count*)

Set PWM period counts.

Set the next period values of a PWM channel. This includes low period count and high period count. When operating in PWM mode, 0% and 100% duty cycle is not available on Quark SE or Quark D2000. When operating in PWM mode, hi_count must be > 0. In timer mode, the value of high count is ignored.

Parameters

in	*pwm*	Which PWM module to set the counts of.
in	*id*	PWM channel id to set.
in	*lo_count*	Num of cycles the output is driven low.
in	*hi_count*	Num of cycles the output is driven high.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 109 of file qm_pwm.c.

4.14.3.5 qm_rc_t qm_pwm_set_config (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*, const qm_pwm_config_t *const *cfg*)

Set PWM channel configuration.

Change the configuration of a PWM channel. This includes low period load value, high period load value, interrupt enable/disable. If interrupts are enabled, registers an ISR with the given user callback function. When operating in PWM mode, 0% and 100% duty cycle is not available on Quark SE or Quark D2000. When setting the mode to PWM mode, hi_count must be > 0. In timer mode, the value of high count is ignored.

Parameters

in	*pwm*	Which PWM module to configure.
in	*id*	PWM channel id to configure.
in	*cfg*	New configuration for PWM.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 66 of file qm_pwm.c.

4.14.3.6 qm_rc_t qm_pwm_start (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*)

Start a PWM/timer channel.

Parameters

in	*pwm*	Which PWM block the PWM is in.
in	*id*	PWM channel id to start.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 46 of file qm_pwm.c.

4.14.3.7 qm_rc_t qm_pwm_stop (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*)

Stop a PWM/timer channel.

Parameters

in	*pwm*	Which PWM block the PWM is in.
in	*id*	PWM channel id to stop.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 56 of file qm_pwm.c.

4.15 RTC

Data Structures

- struct qm_rtc_config_t

Macros

- #define QM_RTC_ALARM_SECOND (32768 / BIT(QM_RTC_DIVIDER))
- #define QM_RTC_ALARM_MINUTE (QM_RTC_ALARM_SECOND ∗ 60)
- #define QM_RTC_ALARM_HOUR (QM_RTC_ALARM_MINUTE ∗ 60)
- #define QM_RTC_ALARM_DAY (QM_RTC_ALARM_HOUR ∗ 24)

Functions

- void qm_rtc_isr_0 (void)
- qm_rc_t qm_rtc_set_config (const qm_rtc_t rtc, const qm_rtc_config_t ∗const cfg)

  Set RTC configuration.
- qm_rc_t qm_rtc_set_alarm (const qm_rtc_t rtc, const uint32_t alarm_val)

  Set Alarm value.
- qm_rc_t qm_rtc_get_config (const qm_rtc_t rtc, qm_rtc_config_t ∗const cfg)

  Get RTC configuration.

4.15.1 Detailed Description

Real Time clock for Quark Microcontrollers.

4.15.2 Macro Definition Documentation

4.15.2.1 #define QM_RTC_ALARM_DAY (QM_RTC_ALARM_HOUR ∗ 24)

Number of RTC ticks in a day

Definition at line 57 of file qm_rtc.h.

4.15.2.2 #define QM_RTC_ALARM_HOUR (QM_RTC_ALARM_MINUTE ∗ 60)

Number of RTC ticks in an hour

Definition at line 55 of file qm_rtc.h.

4.15.2.3 #define QM_RTC_ALARM_MINUTE (QM_RTC_ALARM_SECOND ∗ 60)

Number of RTC ticks in a minute

Definition at line 53 of file qm_rtc.h.

4.15.2.4 #define QM_RTC_ALARM_SECOND (32768 / BIT(QM_RTC_DIVIDER))

Number of RTC ticks in a second

Definition at line 51 of file qm_rtc.h.

4.15.3 Function Documentation

4.15.3.1 qm_rc_t qm_rtc_get_config (const qm_rtc_t *rtc,* qm_rtc_config_t *const *cfg*)

Get RTC configuration.

Get current configuration of RTC module. This includes the initial value in RTC clock periods, if an alarm is required and the tick value of the alarm in RTC clock periods.

Parameters

in	*rtc*	RTC index.
out	*cfg*	New RTC configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 85 of file qm_rtc.c.

4.15.3.2 void qm_rtc_isr_0 (void)

RTC Interrupt Service Routine.

Definition at line 34 of file qm_rtc.c.

4.15.3.3 qm_rc_t qm_rtc_set_alarm (const qm_rtc_t *rtc,* const uint32_t *alarm_val*)

Set Alarm value.

Set a new RTC alarm value after an alarm, that has been set using the qm_rtc_set_config function, has expired and a new alarm value is required.

Parameters

in	*rtc*	RTC index.
in	*alarm_val*	Value to set alarm to.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 72 of file qm_rtc.c.

Referenced by qm_rtc_set_config().

4.15.3.4 qm_rc_t qm_rtc_set_config (const qm_rtc_t *rtc,* const qm_rtc_config_t *const *cfg*)

Set RTC configuration.

Set RTC module configuration. Including the initial value in RTC clock periods, and the alarm value if an alarm is required. If the alarm is enabled, register an ISR with the user defined callback function.

Parameters

in	*rtc*	RTC index.
in	*cfg*	New RTC configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 47 of file qm_rtc.c.

References clk_rtc_set_div(), and qm_rtc_set_alarm().

4.16 System Control Subsystem

Macros

- #define SYS_TICKS_PER_US_XTAL (32)

Enumerations

- enum clk_sys_div_t
- enum clk_sys_mode_t
- enum clk_periph_div_t
- enum clk_gpio_db_div_t
- enum clk_ext_div_t
- enum clk_rtc_div_t
- enum clk_periph_t

Functions

- qm_rc_t clk_sys_set_mode (const clk_sys_mode_t mode, const clk_sys_div_t div)
- qm_rc_t clk_adc_set_div (const uint16_t div)

  Change divider value of ADC clock.
- qm_rc_t clk_periph_set_div (const clk_periph_div_t div)

  Change divider value of peripheral clock.
- qm_rc_t clk_gpio_db_set_div (const clk_gpio_db_div_t div)

  Change divider value of GPIO debounce clock.
- qm_rc_t clk_ext_set_div (const clk_ext_div_t div)

  Change divider value of external clock.
- qm_rc_t clk_rtc_set_div (const clk_rtc_div_t div)

  Change divider value of RTC.
- qm_rc_t clk_periph_enable (const clk_periph_t clocks)
- qm_rc_t clk_periph_disable (const clk_periph_t clocks)
- uint32_t clk_sys_get_ticks_per_us (void)
- void clk_sys_udelay (uint32_t microseconds)

  Idle loop the processor for at least the value given in microseconds.

4.16.1 Detailed Description

System control subsystem for Quark Microcontrollers.

4.16.2 Macro Definition Documentation

4.16.2.1 #define SYS_TICKS_PER_US_XTAL (32)

When using an external crystal, this value must be set to the number of system ticks per micro second. The expected value is 32 ticks for a 32MHz crystal.

Definition at line 51 of file qm_scss.h.

Referenced by clk_sys_set_mode().

4.16.3 Enumeration Type Documentation

4.16.3.1 enum clk_ext_div_t

External crystal clock divider type.

Definition at line 110 of file qm_scss.h.

4.16.3.2 enum clk_gpio_db_div_t

GPIO clock debounce divider type.

Definition at line 96 of file qm_scss.h.

4.16.3.3 enum clk_periph_div_t

Peripheral clock divider type.

Definition at line 85 of file qm_scss.h.

4.16.3.4 enum clk_periph_t

SCSS peripheral clock register type.

Definition at line 142 of file qm_scss.h.

4.16.3.5 enum clk_rtc_div_t

RTC clock divider type.

Definition at line 120 of file qm_scss.h.

4.16.3.6 enum clk_sys_div_t

System clock divider type.

Definition at line 58 of file qm_scss.h.

4.16.3.7 enum clk_sys_mode_t

System clock mode type.

Definition at line 75 of file qm_scss.h.

4.16.4 Function Documentation

4.16.4.1 qm_rc_t clk_adc_set_div (const uint16_t *div*)

Change divider value of ADC clock.

Change ADC clock divider value. The new divider value is set to N, where N is the value set by the function and is between 1 and 1024.

Parameters

in	*div*	Divider value for the ADC clock.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 118 of file qm_scss.c.

4.16.4.2 qm_rc_t clk_ext_set_div (const clk_ext_div_t *div*)

Change divider value of external clock.

Change External clock divider value. The maximum divisor is /8.

Parameters

in	*div*	Divider value for the external clock.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 172 of file qm_scss.c.

4.16.4.3 qm_rc_t clk_gpio_db_set_div (const clk_gpio_db_div_t *div*)

Change divider value of GPIO debounce clock.

Change GPIO debounce clock divider value. The maximum divisor is /128.

Parameters

in	*div*	Divider value for the GPIO debounce clock.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 158 of file qm_scss.c.

4.16.4.4 qm_rc_t clk_periph_disable (const clk_periph_t *clocks*)

Disable clocks for peripherals / registers.

Parameters

in	*clocks*	Which peripheral and register clocks to disable.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 207 of file qm_scss.c.

4.16.4.5 qm_rc_t clk_periph_enable (const clk_periph_t *clocks*)

Enable clocks for peripherals / registers.

Parameters

in	*clocks*	Which peripheral and register clocks to enable.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 198 of file qm_scss.c.

4.16.4.6 qm_rc_t clk_periph_set_div (const clk_periph_div_t *div*)

Change divider value of peripheral clock.

Change Peripheral clock divider value. The maximum divisor is /8. These peripherals include GPIO Interrupt, SPI, I2C and ADC.

Parameters

in	*div*	Divider value for the peripheral clock.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 136 of file qm_scss.c.

4.16.4.7 qm_rc_t clk_rtc_set_div (const clk_rtc_div_t *div*)

Change divider value of RTC.

Change RTC divider value. The maximum divisor is /32768.

Parameters

in	*div*	Divider value for the RTC.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 185 of file qm_scss.c.

Referenced by qm_rtc_set_config().

4.16.4.8 uint32_t clk_sys_get_ticks_per_us (void)

Get number of system ticks per micro second.

Returns

uint32_t Number of system ticks per micro second.

Definition at line 216 of file qm_scss.c.

4.16.4.9 qm_rc_t clk_sys_set_mode (const clk_sys_mode_t *mode,* const clk_sys_div_t *div*)

Change the operating mode and clock divisor of the system clock source. Changing this clock speed affects all peripherals.

Parameters

in	*mode*	System clock source operating mode
in	*div*	System clock divisor.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 38 of file qm_scss.c.

References SYS_TICKS_PER_US_XTAL.

4.16.4.10 void clk_sys_udelay (uint32_t *microseconds*)

Idle loop the processor for at least the value given in microseconds.

This function will wait until at least the given number of microseconds has elapsed since calling this function. Note it is dependent on the system clock speed. The delay parameter does not include, calling the function, returning from it, calculation setup and while loops.

Parameters

in	*microseconds*	Minimum number of micro seconds to delay for.

Returns

void.

Definition at line 221 of file qm_scss.c.

4.17 SPI

Data Structures

- struct qm_spi_config_t
- struct qm_spi_async_transfer_t
- struct qm_spi_transfer_t

Enumerations

- enum qm_spi_frame_size_t
- enum qm_spi_tmode_t { QM_SPI_TMOD_TX_RX, QM_SPI_TMOD_TX, QM_SPI_TMOD_RX, QM_SPI_TMOD_EEPROM_READ }
- enum qm_spi_bmode_t { QM_SPI_BMODE_0, QM_SPI_BMODE_1, QM_SPI_BMODE_2, QM_SPI_BMODE_3 }
- enum qm_spi_slave_select_t

Functions

- qm_rc_t qm_spi_set_config (const qm_spi_t spi, const qm_spi_config_t *const cfg)

  Set SPI configuration.
- qm_rc_t qm_spi_get_config (const qm_spi_t spi, qm_spi_config_t *const cfg)

  Get SPI configuration.
- qm_rc_t qm_spi_slave_select (const qm_spi_t spi, const qm_spi_slave_select_t ss)
- qm_spi_status_t qm_spi_get_status (const qm_spi_t spi)

  Get SPI bus status.
- qm_rc_t qm_spi_transfer (const qm_spi_t spi, qm_spi_transfer_t *const xfer)

  Multi-frame read / write on SPI.
- qm_rc_t qm_spi_irq_transfer (const qm_spi_t spi, qm_spi_async_transfer_t *const xfer)

  Interrupt based transfer on SPI.
- void qm_spi_master_0_isr (void)
- qm_rc_t qm_spi_transfer_terminate (const qm_spi_t spi)

  Terminate SPI IRQ/DMA transfer.

4.17.1 Detailed Description

SPI peripheral driver for Quark Microcontrollers.

4.17.2 Enumeration Type Documentation

4.17.2.1 enum qm_spi_bmode_t

SPI bus mode type.

Enumerator

QM_SPI_BMODE_0	Clock Polarity = 0, Clock Phase = 0
QM_SPI_BMODE_1	Clock Polarity = 0, Clock Phase = 1
QM_SPI_BMODE_2	Clock Polarity = 1, Clock Phase = 0
QM_SPI_BMODE_3	Clock Polarity = 1, Clock Phase = 1

Definition at line 91 of file qm_spi.h.

4.17.2.2 enum qm_spi_frame_size_t

QM SPI frame size type.

Definition at line 46 of file qm_spi.h.

4.17.2.3 enum qm_spi_slave_select_t

SPI slave select type.

Definition at line 101 of file qm_spi.h.

4.17.2.4 enum qm_spi_tmode_t

SPI transfer mode type.

Enumerator

QM_SPI_TMOD_TX_RX Transmit & Receive

QM_SPI_TMOD_TX Transmit Only

QM_SPI_TMOD_RX Receive Only

QM_SPI_TMOD_EEPROM_READ EEPROM Read

Definition at line 81 of file qm_spi.h.

4.17.3 Function Documentation

4.17.3.1 qm_rc_t qm_spi_get_config (const qm_spi_t *spi*, qm_spi_config_t *const *cfg*)

Get SPI configuration.

Get the current configuration of a SPI module. This includes transfer mode, bus mode and clock divider.

Parameters

in	*spi*	Which SPI module to read the configuration of.
in	*cfg*	Current configuration of SPI.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 279 of file qm_spi.c.

References qm_spi_config_t::bus_mode, qm_spi_config_t::clk_divider, qm_spi_config_t::frame_size, and qm_spi_config_t::transfer_mode.

4.17.3.2 qm_spi_status_t qm_spi_get_status (const qm_spi_t *spi*)

Get SPI bus status.

Retrieve SPI bus status. Return QM_SPI_BUSY if transmitting data or data Tx FIFO not empty.

Parameters

in	*spi*	Which SPI to read the status of.

Returns

qm_spi_status_t Returns SPI specific return code.

Definition at line 309 of file qm_spi.c.

4.17.3.3 qm_rc_t qm_spi_irq_transfer (const qm_spi_t *spi*, qm_spi_async_transfer_t *const *xfer*)

Interrupt based transfer on SPI.

Perform an interrupt based transfer on the SPI bus. The function will replenish/empty TX/RX FIFOs on SPI empty/full interrupts. If transfer mode is full duplex (QM_SPI_TMOD_TX_RX), then tx_len and rx_len must be equal and both callbacks cannot be null. Similarly, for transmit-only transfers (QM_SPI_TMOD_TX) rx_len must be 0 and tx_-callback cannot be null, while for receive-only transfers (QM_SPI_TMOD_RX) tx_len must be 0 and rx_callback cannot be null.

Parameters

in	*spi*	Which SPI to transfer to / from.
in	*xfer*	Transfer structure includes write / read data and length; write, read and error callback functions and a callback identifier. This pointer must be kept valid until the transfer is complete. The error callback cannot be null.

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Enable SPI Device

Definition at line 416 of file qm_spi.c.

References QM_SPI_TMOD_RX, QM_SPI_TMOD_TX, and QM_SPI_TMOD_TX_RX.

4.17.3.4 void qm_spi_master_0_isr (void)

Interrupt service routine for the SPI masters

Definition at line 474 of file qm_spi.c.

4.17.3.5 qm_rc_t qm_spi_set_config (const qm_spi_t *spi*, const qm_spi_config_t *const *cfg*)

Set SPI configuration.

Change the configuration of a SPI module. This includes transfer mode, bus mode and clock divider.

Parameters

in	*spi*	Which SPI module to configure.
in	*cfg*	New configuration for SPI.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 247 of file qm_spi.c.

References qm_spi_config_t::bus_mode, qm_spi_config_t::clk_divider, qm_spi_config_t::frame_size, and qm_spi-_config_t::transfer_mode.

4.17.3.6 qm_rc_t qm_spi_slave_select (const qm_spi_t *spi*, const qm_spi_slave_select_t *ss*)

Select which slave to perform SPI transmissions on.

Parameters

in	*spi*	Which SPI module to configure.

in	*ss*	Which slave select line to enable when doing transmissions.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 297 of file qm_spi.c.

4.17.3.7 qm_rc_t qm_spi_transfer (const qm_spi_t *spi*, qm_spi_transfer_t *const *xfer*)

Multi-frame read / write on SPI.

Perform a multi-frame read/write on the SPI bus. This is a blocking synchronous call. If the SPI is currently in use, the function will wait until the SPI is free before beginning the transfer. If transfer mode is full duplex (QM_SPI_TMOD_TX_RX), then tx_len and rx_len must be equal. Similarly, for transmit-only transfers (QM_SPI_TMOD_TX) rx_len must be 0, while for receive-only transfers (QM_SPI_TMOD_RX) tx_len must be 0.

Parameters

in	*spi*	Which SPI to read/write on.
in	*xfer*	Structure containing pre-allocated write and read data buffers.

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Disable SPI Device

Definition at line 320 of file qm_spi.c.

References QM_SPI_TMOD_RX, QM_SPI_TMOD_TX, and QM_SPI_TMOD_TX_RX.

4.17.3.8 qm_rc_t qm_spi_transfer_terminate (const qm_spi_t *spi*)

Terminate SPI IRQ/DMA transfer.

Terminate the current IRQ or DMA transfer on the SPI bus. This will cause the relevant callbacks to be called.

Parameters

in	*spi*	Which SPI to cancel the current transfer.

Returns

qm_rc_t Returns QM_RC_OK on success, error code otherwise.

Disable SPI device

Definition at line 486 of file qm_spi.c.

References QM_SPI_TMOD_RX.

4.18 UART

Data Structures

- struct qm_uart_config_t
- struct qm_uart_transfer_t

Enumerations

- enum qm_uart_lc_t {
  QM_UART_LC_5N1 = 0x00, QM_UART_LC_5N1_5 = 0x04, QM_UART_LC_5E1 = 0x18, QM_UART_LC_5E1_5 = 0x1c,
  QM_UART_LC_5O1 = 0x08, QM_UART_LC_5O1_5 = 0x0c, QM_UART_LC_6N1 = 0x01, QM_UART_LC_6N2 = 0x05,
  QM_UART_LC_6E1 = 0x19, QM_UART_LC_6E2 = 0x1d, QM_UART_LC_6O1 = 0x09, QM_UART_LC_6O2 = 0x0d,
  QM_UART_LC_7N1 = 0x02, QM_UART_LC_7N2 = 0x06, QM_UART_LC_7E1 = 0x1a, QM_UART_LC_7E2 = 0x1e,
  QM_UART_LC_7O1 = 0x0a, QM_UART_LC_7O2 = 0x0e, QM_UART_LC_8N1 = 0x03, QM_UART_LC_8N2 = 0x07,
  QM_UART_LC_8E1 = 0x1b, QM_UART_LC_8E2 = 0x1f, QM_UART_LC_8O1 = 0x0b, QM_UART_LC_8O2 = 0x0f }
- enum qm_uart_status_t

Functions

- void qm_uart_0_isr (void)
- void qm_uart_1_isr (void)
- qm_rc_t qm_uart_set_config (const qm_uart_t uart, const qm_uart_config_t *const cfg)

  Set UART confguration.
- qm_rc_t qm_uart_get_config (const qm_uart_t uart, qm_uart_config_t *const cfg)

  Get UART confguration.
- qm_uart_status_t qm_uart_get_status (const qm_uart_t uart)

  Get UART bus status.
- qm_rc_t qm_uart_write (const qm_uart_t uart, const uint8_t data)

  UART character data write.
- qm_uart_status_t qm_uart_read (const qm_uart_t uart, uint8_t *data)

  UART character data read.
- qm_rc_t qm_uart_write_non_block (const qm_uart_t uart, const uint8_t data)

  UART character data write.
- uint8_t qm_uart_read_non_block (const qm_uart_t uart)

  UART character data read.
- qm_rc_t qm_uart_write_buffer (const qm_uart_t uart, const uint8_t *const data, uint32_t len)

  UART multi-byte data write.
- qm_uart_status_t qm_uart_irq_write (const qm_uart_t uart, const qm_uart_transfer_t *const xfer)

  Interrupt based TX on UART.
- qm_uart_status_t qm_uart_irq_read (const qm_uart_t uart, const qm_uart_transfer_t *const xfer)

  Interrupt based RX on UART.
- qm_rc_t qm_uart_write_terminate (const qm_uart_t uart)

  Terminate UART IRQ/DMA TX transfer.
- qm_rc_t qm_uart_read_terminate (const qm_uart_t uart)

  Terminate UART IRQ/DMA RX transfer.

4.18.1 Detailed Description

UART driver for Quark Microcontrollers.

4.18.2 Enumeration Type Documentation

4.18.2.1 enum qm_uart_lc_t

UART Line control.

Enumerator

QM_UART_LC_5N1 5 data bits, no parity, 1 stop bit
QM_UART_LC_5N1_5 5 data bits, no parity, 1.5 stop bits
QM_UART_LC_5E1 5 data bits, even parity, 1 stop bit
QM_UART_LC_5E1_5 5 data bits, even parity, 1.5 stop bits
QM_UART_LC_5O1 5 data bits, odd parity, 1 stop bit
QM_UART_LC_5O1_5 5 data bits, odd parity, 1.5 stop bits
QM_UART_LC_6N1 6 data bits, no parity, 1 stop bit
QM_UART_LC_6N2 6 data bits, no parity, 2 stop bits
QM_UART_LC_6E1 6 data bits, even parity, 1 stop bit
QM_UART_LC_6E2 6 data bits, even parity, 2 stop bits
QM_UART_LC_6O1 6 data bits, odd parity, 1 stop bit
QM_UART_LC_6O2 6 data bits, odd parity, 2 stop bits
QM_UART_LC_7N1 7 data bits, no parity, 1 stop bit
QM_UART_LC_7N2 7 data bits, no parity, 2 stop bits
QM_UART_LC_7E1 7 data bits, even parity, 1 stop bit
QM_UART_LC_7E2 7 data bits, even parity, 2 stop bits
QM_UART_LC_7O1 7 data bits, odd parity, 1 stop bit
QM_UART_LC_7O2 7 data bits, odd parity, 2 stop bits
QM_UART_LC_8N1 8 data bits, no parity, 1 stop bit
QM_UART_LC_8N2 8 data bits, no parity, 2 stop bits
QM_UART_LC_8E1 8 data bits, even parity, 1 stop bit
QM_UART_LC_8E2 8 data bits, even parity, 2 stop bits
QM_UART_LC_8O1 8 data bits, odd parity, 1 stop bit
QM_UART_LC_8O2 8 data bits, odd parity, 2 stop bits

Definition at line 80 of file qm_uart.h.

4.18.2.2 enum qm_uart_status_t

UART Status type.

Definition at line 130 of file qm_uart.h.

4.18.3 Function Documentation

4.18.3.1 void qm_uart_0_isr (void)

UART 0 Interrupt Service Routine.

Definition at line 127 of file qm_uart.c.

4.18.3.2 void qm_uart_1_isr (void)

UART 1 Interrupt Service Routine.

Definition at line 132 of file qm_uart.c.

4.18.3.3 qm_rc_t qm_uart_get_config (const qm_uart_t *uart*, qm_uart_config_t *const *cfg*)

Get UART confguration.

Read the current configuration of a UART module. This includes line control, baud rate and hardware flow control.

Parameters

in	*uart*	Which UART module to read the configuration of.
in	*cfg*	Current configuration for UART.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 170 of file qm_uart.c.

4.18.3.4 qm_uart_status_t qm_uart_get_status (const qm_uart_t *uart*)

Get UART bus status.

Retrieve UART interface status. Return QM_UART_BUSY if transmitting data; QM_UART_IDLE if available for transfer QM_UART_TX_ERROR if an error has occurred in transmission.

Parameters

in	*uart*	Which UART to read the status of.

Returns

qm_uart_status_t Returns UART specific return code.

Definition at line 189 of file qm_uart.c.

Referenced by qm_uart_irq_write().

4.18.3.5 qm_uart_status_t qm_uart_irq_read (const qm_uart_t *uart*, const qm_uart_transfer_t *const *xfer*)

Interrupt based RX on UART.

Perform an interrupt based RX transfer on the UART bus. The function will read back the RX FIFOs on UART empty interrupts.

Parameters

in	*uart*	UART register block pointer.
in	*xfer*	Structure containing pre-allocated read buffer and callback functions. The callbacks cannot be null.

Returns

qm_uart_status_t Returns UART specific return code.

Definition at line 315 of file qm_uart.c.

4.18.3.6 qm_uart_status_t qm_uart_irq_write (const qm_uart_t *uart*, const qm_uart_transfer_t *const *xfer*)

Interrupt based TX on UART.

Perform an interrupt based TX transfer on the UART bus. The function will replenish the TX FIFOs on UART empty interrupts.

Parameters

in	*uart*	UART index.
in	*xfer*	Structure containing pre-allocated write buffer and callback functions. The callbacks cannot be null.

Returns

qm_uart_status_t Returns UART specific return code.

Definition at line 284 of file qm_uart.c.

References qm_uart_get_status().

4.18.3.7 qm_uart_status_t qm_uart_read (const qm_uart_t *uart*, uint8_t * *data*)

UART character data read.

Perform a single character read from the UART interface. This is a blocking synchronous call.

Parameters

in	*uart*	UART index.
out	*data*	Data to read from UART.

Returns

qm_uart_status_t Returns UART specific return code.

Definition at line 228 of file qm_uart.c.

4.18.3.8 uint8_t qm_uart_read_non_block (const qm_uart_t *uart*)

UART character data read.

Perform a single character read from the UART interface. This is a non-blocking synchronous call.

Parameters

in	*uart*	UART index.

Returns

uint8_t Character read.

Definition at line 255 of file qm_uart.c.

4.18.3.9 qm_rc_t qm_uart_read_terminate (const qm_uart_t *uart*)

Terminate UART IRQ/DMA RX transfer.

Terminate the current IRQ or DMA RX transfer on the UART bus. This will cause the relevant callbacks to be called.

Parameters

in	*uart*	UART index.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 357 of file qm_uart.c.

4.18.3.10 qm_rc_t qm_uart_set_config (const qm_uart_t *uart*, const qm_uart_config_t * const *cfg*)

Set UART confguration.

Change the configuration of a UART module. This includes line control, baud rate and hardware flow control.

Parameters

in	*uart*	Which UART module to configure.
in	*cfg*	New configuration for UART.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 137 of file qm_uart.c.

4.18.3.11 qm_rc_t qm_uart_write (const qm_uart_t *uart*, const uint8_t *data*)

UART character data write.

Perform a single character write on the UART interface. This is a blocking synchronous call.

Parameters

in	*uart*	UART index.
in	*data*	Data to write to UART.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 214 of file qm_uart.c.

4.18.3.12 qm_rc_t qm_uart_write_buffer (const qm_uart_t *uart*, const uint8_t *const *data*, uint32_t *len*)

UART multi-byte data write.

Perform a write on the UART interface. This is a blocking synchronous call. The function will block until all data has been transferred.

Parameters

in	*uart*	UART index.
in	*data*	Data to write to UART.
in	*len*	Length of data to write to UART.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 260 of file qm_uart.c.

4.18.3.13 qm_rc_t qm_uart_write_non_block (const qm_uart_t *uart*, const uint8_t *data*)

UART character data write.

Perform a single character write on the UART interface. This is a non-blocking synchronous call.

Parameters

in	*uart*	UART index.
in	*data*	Data to write to UART.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 246 of file qm_uart.c.

4.18.3.14 qm_rc_t qm_uart_write_terminate (const qm_uart_t *uart*)

Terminate UART IRQ/DMA TX transfer.

Terminate the current IRQ or DMA TX transfer on the UART bus. This will cause the relevant callbacks to be called.

Parameters

in	*uart*	UART index.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 345 of file qm_uart.c.

4.19 Version

Macros

- #define QM_VER_API_MAJOR 1
- #define QM_VER_API_MINOR 0
- #define QM_VER_API_PATCH 0
- #define QM_VER_API_UINT
- #define QM_VER_API_STRING QM_VER_STRINGIFY(QM_VER_API_MAJOR, QM_VER_API_MINOR, QM_VER_API_PATCH)

4.19.1 Detailed Description

Version number functions for API.

4.19.2 Macro Definition Documentation

4.19.2.1 #define QM_VER_API_MAJOR 1

QM API major version number

Definition at line 45 of file qm_version.h.

4.19.2.2 #define QM_VER_API_MINOR 0

QM API minor version number

Definition at line 50 of file qm_version.h.

4.19.2.3 #define QM_VER_API_PATCH 0

QM API patch version number

Definition at line 55 of file qm_version.h.

4.19.2.4 #define QM_VER_API_STRING QM_VER_STRINGIFY(QM_VER_API_MAJOR, QM_VER_API_MINOR, QM_VER_API_PATCH)

Create a version number string from the major, minor and patch numbers

Definition at line 67 of file qm_version.h.

4.19.2.5 #define QM_VER_API_UINT

Value:

```
((QM_VER_API_MAJOR * 10000) + (QM_VER_API_MINOR * 100) +
      \
       QM_VER_API_PATCH)
```

Create a single version number from the major, minor and patch numbers

Definition at line 60 of file qm_version.h.

4.20 WDT

Data Structures

- struct qm_wdt_config_t

Enumerations

- enum qm_wdt_mode_t
- enum qm_wdt_clock_timeout_cycles_t

Functions

- qm_rc_t qm_wdt_start (const qm_wdt_t wdt)
- qm_rc_t qm_wdt_set_config (const qm_wdt_t wdt, const qm_wdt_config_t *const cfg)
- qm_rc_t qm_wdt_get_config (const qm_wdt_t wdt, qm_wdt_config_t *const cfg)
- qm_rc_t qm_wdt_reload (const qm_wdt_t wdt)
- void qm_wdt_isr_0 (void)

4.20.1 Detailed Description

Watchdog timer for Quark Microcontrollers.

4.20.2 Enumeration Type Documentation

4.20.2.1 enum qm_wdt_clock_timeout_cycles_t

WDT clock cycles for timeout type. This value is a power of 2.

Definition at line 57 of file qm_wdt.h.

4.20.2.2 enum qm_wdt_mode_t

WDT Mode type.

Definition at line 52 of file qm_wdt.h.

4.20.3 Function Documentation

4.20.3.1 qm_rc_t qm_wdt_get_config (const qm_wdt_t *wdt*, qm_wdt_config_t *const *cfg*)

Get the current configuration of WDT module. This includes the timeout period in PCLK cycles, the WDT mode of operation.

Parameters

in	*wdt*	WDT index.
out	*cfg*	Parameter to be set with the current WDT configuration.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 79 of file qm_wdt.c.

4.20.3.2 void qm_wdt_isr_0 (void)

WDT Interrupt Service Routine

Definition at line 36 of file qm_wdt.c.

4.20.3.3 qm_rc_t qm_wdt_reload (const qm_wdt_t *wdt*)

Reload the WDT counter with safety value, i.e. service the watchdog

Parameters

in	*wdt*	WDT index.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 91 of file qm_wdt.c.

Referenced by qm_wdt_set_config().

4.20.3.4 qm_rc_t qm_wdt_set_config (const qm_wdt_t *wdt,* const qm_wdt_config_t *const *cfg*)

Set configuration of WDT module. This includes the timeout period in PCLK cycles, the WDT mode of operation. It also registers an ISR to the user defined callback.

Parameters

in	*wdt*	WDT index.
in	*cfg*	New configuration for WDT.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 64 of file qm_wdt.c.

References qm_wdt_reload().

4.20.3.5 qm_rc_t qm_wdt_start (const qm_wdt_t *wdt*)

Start WDT. Once started, WDT can only be stopped by a SoC reset.

Parameters

in	*wdt*	WDT index.

Returns

qm_rc_t QM_RC_OK on success, error code otherwise.

Definition at line 42 of file qm_wdt.c.

5 Data Structure Documentation

5.1 qm_ac_config_t Struct Reference

`#include <qm_comparator.h>`

5.1.1 Detailed Description

Analog comparator configuration type. Each bit in the registers controls an Analog Comparator pin.

Definition at line 47 of file qm_comparator.h.

5.2 qm_aonpt_config_t Struct Reference

`#include <qm_aon_counters.h>`

5.2.1 Detailed Description

Always-on Periodic Timer configuration type.

Definition at line 46 of file qm_aon_counters.h.

5.3 qm_flash_config_t Struct Reference

`#include <qm_flash.h>`

Data Fields

- uint8_t wait_states
- uint8_t us_count
- qm_flash_disable_t write_disable

5.3.1 Detailed Description

Flash configuration structure

Definition at line 91 of file qm_flash.h.

5.3.2 Field Documentation

5.3.2.1 uint8_t qm_flash_config_t::us_count

Number of clocks in a microsecond

Definition at line 93 of file qm_flash.h.

Referenced by qm_flash_get_config(), and qm_flash_set_config().

5.3.2.2 uint8_t qm_flash_config_t::wait_states

Read wait state

Definition at line 92 of file qm_flash.h.

Referenced by qm_flash_get_config(), and qm_flash_set_config().

5.3.2.3 qm_flash_disable_t qm_flash_config_t::write_disable

Write disable

Definition at line 94 of file qm_flash.h.

Referenced by qm_flash_get_config(), and qm_flash_set_config().

5.4 qm_fpr_config_t Struct Reference

`#include <qm_fpr.h>`

Data Fields

- qm_fpr_en_t en_mask
- qm_fpr_read_allow_t allow_agents
- uint8_t up_bound
- uint8_t low_bound

5.4.1 Detailed Description

Flash Protection Region configuration structure

Definition at line 89 of file qm_fpr.h.

5.4.2 Field Documentation

5.4.2.1 qm_fpr_read_allow_t qm_fpr_config_t::allow_agents

Per-agent read enable bitmask

Definition at line 91 of file qm_fpr.h.

Referenced by qm_fpr_get_config(), and qm_fpr_set_config().

5.4.2.2 qm_fpr_en_t qm_fpr_config_t::en_mask

Enable/lock bitmask

Definition at line 90 of file qm_fpr.h.

Referenced by qm_fpr_get_config(), and qm_fpr_set_config().

5.4.2.3 uint8_t qm_fpr_config_t::low_bound

1KB-aligned lower Flash phys addr

Definition at line 93 of file qm_fpr.h.

Referenced by qm_fpr_get_config(), and qm_fpr_set_config().

5.4.2.4 uint8_t qm_fpr_config_t::up_bound

1KB-aligned upper Flash phys addr

Definition at line 92 of file qm_fpr.h.

Referenced by qm_fpr_get_config(), and qm_fpr_set_config().

5.5 qm_gpio_port_config_t Struct Reference

`#include <qm_gpio.h>`

5.5.1 Detailed Description

GPIO port configuration type. Each bit in the registers control a GPIO pin.

Definition at line 46 of file qm_gpio.h.

5.6 qm_i2c_config_t Struct Reference

`#include <qm_i2c.h>`

5.6.1 Detailed Description

I2C configuration type.

Definition at line 99 of file qm_i2c.h.

5.7 qm_i2c_transfer_t Struct Reference

`#include <qm_i2c.h>`

5.7.1 Detailed Description

I2C transfer type. Master mode:

- if Tx len is 0: perform receive-only transaction
- if Rx len is 0: perform transmit-only transaction
- both Tx and Rx len not 0: perform a transmit-then- receive combined transaction Slave mode:
- If read or write exceed the buffer, then wrap around.

Definition at line 116 of file qm_i2c.h.

5.8 qm_mpr_config_t Struct Reference

`#include <qm_mpr.h>`

Data Fields

- uint8_t en_lock_mask
- uint8_t agent_read_en_mask
- uint8_t agent_write_en_mask
- uint8_t up_bound
- uint8_t low_bound

5.8.1 Detailed Description

SRAM Memory Protection Region configuration type.

Definition at line 68 of file qm_mpr.h.

5.8.2 Field Documentation

5.8.2.1 uint8_t qm_mpr_config_t::agent_read_en_mask

Per-agent read enable bitmask

Definition at line 70 of file qm_mpr.h.

Referenced by qm_mpr_get_config(), and qm_mpr_set_config().

5.8.2.2 uint8_t qm_mpr_config_t::agent_write_en_mask

Per-agent write enable bitmask

Definition at line 71 of file qm_mpr.h.

Referenced by qm_mpr_get_config(), and qm_mpr_set_config().

5.8.2.3 uint8_t qm_mpr_config_t::en_lock_mask

Enable/lock bitmask

Definition at line 69 of file qm_mpr.h.

Referenced by qm_mpr_get_config(), and qm_mpr_set_config().

5.8.2.4 uint8_t qm_mpr_config_t::low_bound

1KB-aligned lower addr

Definition at line 73 of file qm_mpr.h.

Referenced by qm_mpr_get_config(), and qm_mpr_set_config().

5.8.2.5 uint8_t qm_mpr_config_t::up_bound

1KB-aligned upper addr

Definition at line 72 of file qm_mpr.h.

Referenced by qm_mpr_get_config(), and qm_mpr_set_config().

5.9 qm_pic_timer_config_t Struct Reference

`#include <qm_pic_timer.h>`

Data Fields

- qm_pic_timer_mode_t mode
- bool int_en
- void(∗ callback)(void)

5.9.1 Detailed Description

PIC timer configuration type.

Definition at line 54 of file qm_pic_timer.h.

5.9.2 Field Documentation

5.9.2.1 void(* qm_pic_timer_config_t::callback)(void)

Callback function

Definition at line 57 of file qm_pic_timer.h.

Referenced by qm_pic_timer_get_config(), and qm_pic_timer_set_config().

5.9.2.2 bool qm_pic_timer_config_t::int_en

Interrupt enable

Definition at line 56 of file qm_pic_timer.h.

Referenced by qm_pic_timer_get_config(), and qm_pic_timer_set_config().

5.9.2.3 qm_pic_timer_mode_t qm_pic_timer_config_t::mode

Operation mode

Definition at line 55 of file qm_pic_timer.h.

Referenced by qm_pic_timer_get_config(), and qm_pic_timer_set_config().

5.10 qm_pwm_config_t Struct Reference

`#include <qm_pwm.h>`

5.10.1 Detailed Description

PWM / Timer configuration type.

Definition at line 59 of file qm_pwm.h.

5.11 qm_rtc_config_t Struct Reference

`#include <qm_rtc.h>`

5.11.1 Detailed Description

RTC configuration type.

Definition at line 62 of file qm_rtc.h.

5.12 qm_spi_async_transfer_t Struct Reference

`#include <qm_spi.h>`

5.12.1 Detailed Description

SPI IRQ transfer type.

Definition at line 130 of file qm_spi.h.

5.13 qm_spi_config_t Struct Reference

`#include <qm_spi.h>`

Data Fields

- qm_spi_frame_size_t frame_size
- qm_spi_tmode_t transfer_mode
- qm_spi_bmode_t bus_mode
- uint16_t clk_divider

5.13.1 Detailed Description

SPI configuration type.

Definition at line 119 of file qm_spi.h.

5.13.2 Field Documentation

5.13.2.1 qm_spi_bmode_t qm_spi_config_t::bus_mode

Bus mode (enum)

Definition at line 122 of file qm_spi.h.

Referenced by qm_spi_get_config(), and qm_spi_set_config().

5.13.2.2 uint16_t qm_spi_config_t::clk_divider

SCK = SPI_clock/clk_divider. A value of 0 will disable SCK.

Definition at line 123 of file qm_spi.h.

Referenced by qm_spi_get_config(), and qm_spi_set_config().

5.13.2.3 qm_spi_frame_size_t qm_spi_config_t::frame_size

Frame Size

Definition at line 120 of file qm_spi.h.

Referenced by qm_spi_get_config(), and qm_spi_set_config().

5.13.2.4 qm_spi_tmode_t qm_spi_config_t::transfer_mode

Transfer mode (enum)

Definition at line 121 of file qm_spi.h.

Referenced by qm_spi_get_config(), and qm_spi_set_config().

5.14 qm_spi_transfer_t Struct Reference

`#include <qm_spi.h>`

5.14.1 Detailed Description

SPI transfer type.

Definition at line 147 of file qm_spi.h.

5.15 qm_uart_config_t Struct Reference

`#include <qm_uart.h>`

5.15.1 Detailed Description

UART configuration type.

Definition at line 147 of file qm_uart.h.

5.16 qm_uart_transfer_t Struct Reference

`#include <qm_uart.h>`

5.16.1 Detailed Description

UART IRQ transfer structure, holds pre-allocated write and read buffers. Also pointers to user defined callbacks for write, read and errors.

Definition at line 158 of file qm_uart.h.

5.17 qm_wdt_config_t Struct Reference

`#include <qm_wdt.h>`

5.17.1 Detailed Description

QM WDT configuration type.

Definition at line 79 of file qm_wdt.h.

Index