

Intel® Quark™ Microcontroller Software Interface  
1.0

Generated by Doxygen 1.8.6

Sun Dec 6 2015 06:31:42

## Contents

<b>1</b>	<b>Licensing information</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>1</b>
2.1	Modules . . . . .	1
<b>3</b>	<b>Data Structure Index</b>	<b>2</b>
3.1	Data Structures . . . . .	2
<b>4</b>	<b>Module Documentation</b>	<b>3</b>
4.1	Always-on Counters . . . . .	3
4.1.1	Detailed Description . . . . .	3
4.1.2	Function Documentation . . . . .	3
4.2	Analog Comparator . . . . .	8
4.2.1	Detailed Description . . . . .	8
4.2.2	Function Documentation . . . . .	8
4.3	Flash . . . . .	9
4.3.1	Detailed Description . . . . .	9
4.3.2	Enumeration Type Documentation . . . . .	9
4.3.3	Function Documentation . . . . .	10
4.4	FPR . . . . .	13
4.4.1	Detailed Description . . . . .	13
4.4.2	Macro Definition Documentation . . . . .	13
4.4.3	Enumeration Type Documentation . . . . .	14
4.4.4	Function Documentation . . . . .	14
4.5	GPIO . . . . .	16
4.5.1	Detailed Description . . . . .	16
4.5.2	Function Documentation . . . . .	16
4.6	I2C . . . . .	19
4.6.1	Detailed Description . . . . .	19
4.6.2	Enumeration Type Documentation . . . . .	19
4.6.3	Function Documentation . . . . .	20
4.7	Identification . . . . .	23
4.7.1	Detailed Description . . . . .	23
4.7.2	Function Documentation . . . . .	23
4.8	Initialisation . . . . .	24
4.8.1	Detailed Description . . . . .	24
4.8.2	Enumeration Type Documentation . . . . .	24
4.8.3	Function Documentation . . . . .	24
4.9	Interrupt . . . . .	25

4.9.1	Detailed Description	25
4.9.2	Macro Definition Documentation	25
4.9.3	Typedef Documentation	26
4.9.4	Function Documentation	26
4.10	MPR	28
4.10.1	Detailed Description	28
4.10.2	Macro Definition Documentation	28
4.10.3	Function Documentation	28
4.11	PIC Timer	31
4.11.1	Detailed Description	31
4.11.2	Enumeration Type Documentation	31
4.11.3	Function Documentation	31
4.12	Pin Muxing setup	33
4.12.1	Detailed Description	33
4.12.2	Enumeration Type Documentation	33
4.12.3	Function Documentation	33
4.13	Power state	35
4.13.1	Detailed Description	35
4.13.2	Function Documentation	35
4.14	PWM / Timer	36
4.14.1	Detailed Description	36
4.14.2	Enumeration Type Documentation	36
4.14.3	Function Documentation	36
4.15	RTC	39
4.15.1	Detailed Description	39
4.15.2	Macro Definition Documentation	39
4.15.3	Function Documentation	40
4.16	System Control Subsystem	41
4.16.1	Detailed Description	41
4.16.2	Macro Definition Documentation	41
4.16.3	Enumeration Type Documentation	42
4.16.4	Function Documentation	42
4.17	SPI	46
4.17.1	Detailed Description	46
4.17.2	Enumeration Type Documentation	46
4.17.3	Function Documentation	47
4.18	UART	50
4.18.1	Detailed Description	51
4.18.2	Enumeration Type Documentation	51
4.18.3	Function Documentation	51

4.19	Version	57
4.19.1	Detailed Description	57
4.19.2	Macro Definition Documentation	57
4.20	WDT	58
4.20.1	Detailed Description	58
4.20.2	Enumeration Type Documentation	58
4.20.3	Function Documentation	58
<b>5</b>	<b>Data Structure Documentation</b>	<b>60</b>
5.1	qm_ac_config_t Struct Reference	60
5.1.1	Detailed Description	60
5.2	qm_aonpt_config_t Struct Reference	60
5.2.1	Detailed Description	60
5.3	qm_flash_config_t Struct Reference	60
5.3.1	Detailed Description	60
5.3.2	Field Documentation	60
5.4	qm_fpr_config_t Struct Reference	61
5.4.1	Detailed Description	61
5.4.2	Field Documentation	61
5.5	qm_gpio_port_config_t Struct Reference	62
5.5.1	Detailed Description	62
5.6	qm_i2c_config_t Struct Reference	62
5.6.1	Detailed Description	62
5.7	qm_i2c_transfer_t Struct Reference	62
5.7.1	Detailed Description	62
5.8	qm_mpr_config_t Struct Reference	62
5.8.1	Detailed Description	62
5.8.2	Field Documentation	63
5.9	qm_pic_timer_config_t Struct Reference	63
5.9.1	Detailed Description	63
5.9.2	Field Documentation	63
5.10	qm_pwm_config_t Struct Reference	64
5.10.1	Detailed Description	64
5.11	qm_rtc_config_t Struct Reference	64
5.11.1	Detailed Description	64
5.12	qm_spi_async_transfer_t Struct Reference	64
5.12.1	Detailed Description	64
5.13	qm_spi_config_t Struct Reference	64
5.13.1	Detailed Description	65
5.13.2	Field Documentation	65

5.14	<a href="#">qm_spi_transfer_t Struct Reference</a>	65
5.14.1	<a href="#">Detailed Description</a>	65
5.15	<a href="#">qm_uart_config_t Struct Reference</a>	65
5.15.1	<a href="#">Detailed Description</a>	66
5.16	<a href="#">qm_uart_transfer_t Struct Reference</a>	66
5.16.1	<a href="#">Detailed Description</a>	66
5.17	<a href="#">qm_wdt_config_t Struct Reference</a>	66
5.17.1	<a href="#">Detailed Description</a>	66
<b>Index</b>		<b>67</b>

## 1 Licensing information

Copyright (c) 2015, Intel Corporation All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the Intel Corporation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE INTEL CORPORATION OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 2 Module Index

### 2.1 Modules

Here is a list of all modules:

<b>Always-on Counters</b>	<b>3</b>
<b>Analog Comparator</b>	<b>8</b>
<b>Flash</b>	<b>9</b>
<b>FPR</b>	<b>13</b>
<b>GPIO</b>	<b>16</b>

I2C	19
Identification	23
Initialisation	24
Interrupt	25
MPR	28
PIC Timer	31
Pin Muxing setup	33
Power state	35
PWM / Timer	36
RTC	39
System Control Subsystem	41
SPI	46
UART	50
Version	57
WDT	58

## 3 Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">qm_ac_config_t</a>	60
<a href="#">qm_aonpt_config_t</a>	60
<a href="#">qm_flash_config_t</a>	60
<a href="#">qm_fpr_config_t</a>	61
<a href="#">qm_gpio_port_config_t</a>	62
<a href="#">qm_i2c_config_t</a>	62
<a href="#">qm_i2c_transfer_t</a>	62
<a href="#">qm_mpr_config_t</a>	62
<a href="#">qm_pic_timer_config_t</a>	63
<a href="#">qm_pwm_config_t</a>	64
<a href="#">qm_rtc_config_t</a>	64
<a href="#">qm_spi_async_transfer_t</a>	64
<a href="#">qm_spi_config_t</a>	64

<a href="#">qm_spi_transfer_t</a>	65
<a href="#">qm_uart_config_t</a>	65
<a href="#">qm_uart_transfer_t</a>	66
<a href="#">qm_wdt_config_t</a>	66

## 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 Microcontrollers

#### 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

<code>in</code>	<code>aonc</code>	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**

<i>in</i>	<i>aonc</i>	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**

<i>in</i>	<i>aonc</i>	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**

<i>in</i>	<i>aonc</i>	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 expires and an interrupt is triggered. The Periodic Timer is disabled if the counter is set to 0.

**Parameters**

<i>in</i>	<i>aonc</i>	Always-on counter to read.
<i>out</i>	<i>cfg</i>	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

<code>in</code>	<code>aonc</code>	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

<code>in</code>	<code>aonc</code>	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

<code>in</code>	<code>aonc</code>	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 expires and an interrupt is triggered. The Periodic Timer is disabled if the counter is set to 0.

**Parameters**

<code>in</code>	<code>aonc</code>	Always-on counter to read.
<code>in</code>	<code>cfg</code>	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

<i>in</i>	<i>config</i>	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

<i>in</i>	<i>config</i>	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	<i>flash</i>	Flash controller index.
out	<i>cfg</i>	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	<i>flash</i>	Flash controller index.
in	<i>include_rom</i>	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	<i>flash</i>	Flash controller index.
in	<i>region</i>	Flash region to address.
in	<i>page_num</i>	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	<i>flash</i>	Flash controller index.
in	<i>region</i>	Which Flash region to address.
in	<i>f_addr</i>	Address within Flash physical address space.
in	<i>page_buffer</i>	Page buffer to store page, must be at least QM_FLASH_PAGE_SIZE words big.
in	<i>data_buffer</i>	Data buffer to write.
in	<i>len</i>	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* )

Writes 2KB flash page.

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

**Parameters**

in	<i>flash</i>	Flash controller index.
in	<i>region</i>	Which Flash region to address.
in	<i>page_num</i>	Which page of flash to overwrite.
in	<i>data</i>	Data buffer to write.
in	<i>len</i>	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* )

Configures a Flash controller.

Configures 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	<i>flash</i>	Flash controller index.
in	<i>cfg</i>	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* )

Writes 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	<i>flash</i>	Flash controller index.
in	<i>region</i>	Flash region to address.
in	<i>f_addr</i>	Address within Flash physical address space.
in	<i>data</i>	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	<i>flash</i>	Which Flash to read the configuration of.
in	<i>id</i>	FPR identifier.
out	<i>cfg</i>	FPR configuration.
in	<i>region</i>	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	<i>flash</i>	Which Flash controller to configure.
in	<i>id</i>	FPR identifier.
in	<i>cfg</i>	FPR configuration.
in	<i>region</i>	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	<i>mode</i>	(generate interrupt, warm reset, enter probe mode).
in	<i>fpr_cb</i>	for interrupt mode (only). This cannot be null.
in	<i>flash</i>	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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	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	<i>gpio</i>	GPIO port index to read the configuration of.
out	<i>cfg</i>	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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	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	<i>gpio</i>	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	<i>gpio</i>	GPIO port index to configure.
in	<i>cfg</i>	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**

<i>in</i>	<i>gpio</i>	GPIO port index.
<i>in</i>	<i>pin</i>	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**

<i>in</i>	<i>gpio</i>	GPIO port index.
<i>in</i>	<i>val</i>	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

<code>in</code>	<code>i2c</code>	Which I2C to read the configuration of.
<code>out</code>	<code>cfg</code>	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

<code>in</code>	<code>i2c</code>	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	<i>i2c</i>	Which I2C to transfer from.
in	<i>xfer</i>	Transfer structure includes write / read data and length, write, read and error callback functions and a callback identifier.
in	<i>slave_addr</i>	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	<i>i2c</i>	Which I2C to read from.
in	<i>slave_addr</i>	Address of slave device to read from.
out	<i>data</i>	Pre-allocated buffer to populate with data.
in	<i>len</i>	length of data to read from slave.
in	<i>stop</i>	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	<i>i2c</i>	Which I2C to write to.
in	<i>slave_addr</i>	Address of slave to write to.
in	<i>data</i>	Pre-allocated buffer of data to write.
in	<i>len</i>	length of data to write.
in	<i>stop</i>	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	<i>i2c</i>	Which I2C to set the configuration of.
out	<i>cfg</i>	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	<i>i2c</i>	I2C index.
in	<i>speed</i>	Bus speed (Standard or Fast. Fast includes Fast+ mode)
in	<i>lo_cnt</i>	SCL low count.
in	<i>hi_cnt</i>	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	<i>i2c</i>	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

<i>in</i>	<i>reset_type</i>	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	<i>vector</i>	Vector number.
in	<i>isr</i>	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	<i>irq</i>	IRQ number. Must be of type <code>QM_IRQ_XXX</code> .
in	<i>isr</i>	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	<i>irq</i>	IRQ number. Must be of type <code>QM_IRQ_XXX</code> .
in	<i>register_offset</i>	Interrupt Mask Register offset on SCSS. Must be of type <code>QM_IRQ_XXX_MASK_OFFSET</code> .

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	<i>vector</i>	Interrupt Vector number.
in	<i>isr</i>	ISR to register to given vector. Must be a valid x86 ISR. If this can't be provided, <code>qm_irq_request()</code> or <code>qm_int_vector_request()</code> 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**

<i>in</i>	<i>irq</i>	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**

<i>in</i>	<i>irq</i>	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**

<i>in</i>	<i>id</i>	Which MPR to get configuration of.
<i>out</i>	<i>cfg</i>	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**

<i>in</i>	<i>id</i>	Which MPR to configure.
<i>in</i>	<i>cfg</i>	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**

<i>in</i>	<i>mode</i>	(generate interrupt, warm reset, enter probe mode).
<i>in</i>	<i>callback_fn</i>	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

---

<i>out</i>	<i>cfg</i>	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**

<i>in</i>	<i>count</i>	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**

<i>in</i>	<i>cfg</i>	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

<code>in</code>	<code>pin</code>	which pin to configure.
<code>in</code>	<code>enable</code>	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	<i>pin</i>	which pin to configure.
in	<i>enable</i>	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	<i>pin</i>	which pin to configure.
in	<i>fn</i>	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	<i>pin</i>	which pin to configure.
in	<i>slew</i>	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	<i>pwm</i>	Which PWM module to get the count of.
in	<i>id</i>	PWM channel id to read the values of.
out	<i>lo_count</i>	Num of cycles the output is driven low.



out	<i>hi_count</i>	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	<i>pwm</i>	Which PWM module to get the configuration of.
in	<i>id</i>	PWM channel id to get the configuration of.
out	<i>cfg</i>	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	<i>pwm</i>	Which PWM module to set the counts of.
in	<i>id</i>	PWM channel id to set.
in	<i>lo_count</i>	Num of cycles the output is driven low.
in	<i>hi_count</i>	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	<i>pwm</i>	Which PWM module to configure.
in	<i>id</i>	PWM channel id to configure.
in	<i>cfg</i>	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	<i>pwm</i>	Which PWM block the PWM is in.
in	<i>id</i>	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	<i>pwm</i>	Which PWM block the PWM is in.
in	<i>id</i>	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_rc_get_config ( const qm_rc_t rtc, qm_rc_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	<i>rtc</i>	RTC index.
out	<i>cfg</i>	New RTC configuration.

##### Returns

`qm_rc_t QM_RC_OK` on success, error code otherwise.

Definition at line 85 of file `qm_rc.c`.

#### 4.15.3.2 `void qm_rc_isr_0 ( void )`

RTC Interrupt Service Routine.

Definition at line 34 of file `qm_rc.c`.

#### 4.15.3.3 `qm_rc_t qm_rc_set_alarm ( const qm_rc_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_rc_set_config` function, has expired and a new alarm value is required.

##### Parameters

in	<i>rtc</i>	RTC index.
in	<i>alarm_val</i>	Value to set alarm to.

##### Returns

`qm_rc_t QM_RC_OK` on success, error code otherwise.

Definition at line 72 of file `qm_rc.c`.

Referenced by `qm_rc_set_config()`.

#### 4.15.3.4 `qm_rc_t qm_rc_set_config ( const qm_rc_t rtc, const qm_rc_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	<i>rtc</i>	RTC index.
in	<i>cfg</i>	New RTC configuration.

##### Returns

`qm_rc_t QM_RC_OK` on success, error code otherwise.

Definition at line 47 of file `qm_rc.c`.

References `clk_rc_set_div()`, and `qm_rc_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

<code>in</code>	<code>div</code>	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

<code>in</code>	<code>div</code>	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

<code>in</code>	<code>div</code>	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

<code>in</code>	<code>mode</code>	System clock source operating mode
<code>in</code>	<code>div</code>	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

<i>in</i>	<i>microseconds</i>	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	<i>spi</i>	Which SPI to transfer to / from.
in	<i>xfer</i>	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	<i>spi</i>	Which SPI module to configure.
in	<i>cfg</i>	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	<i>spi</i>	Which SPI module to configure.
----	------------	--------------------------------

<i>in</i>	<i>ss</i>	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**

<i>in</i>	<i>spi</i>	Which SPI to read/write on.
<i>in</i>	<i>xfer</i>	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**

<i>in</i>	<i>spi</i>	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 configuration.*
- [qm\\_rc\\_t qm\\_uart\\_get\\_config](#) (const [qm\\_uart\\_t](#) uart, [qm\\_uart\\_config\\_t](#) \*const cfg)  
*Get UART configuration.*
- [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 configuration.

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

##### Parameters

in	<i>uart</i>	Which UART module to read the configuration of.
in	<i>cfg</i>	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	<i>uart</i>	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	<i>uart</i>	UART register block pointer.
in	<i>xfer</i>	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**

<i>in</i>	<i>uart</i>	UART index.
<i>in</i>	<i>xfer</i>	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**

<i>in</i>	<i>uart</i>	UART index.
<i>out</i>	<i>data</i>	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**

<i>in</i>	<i>uart</i>	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**

<i>in</i>	<i>uart</i>	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 configuration.

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

**Parameters**

in	<i>uart</i>	Which UART module to configure.
in	<i>cfg</i>	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	<i>uart</i>	UART index.
in	<i>data</i>	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	<i>uart</i>	UART index.
in	<i>data</i>	Data to write to UART.
in	<i>len</i>	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	<i>uart</i>	UART index.
in	<i>data</i>	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

<code>in</code>	<code>uart</code>	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

<code>in</code>	<code>wdt</code>	WDT index.
<code>out</code>	<code>cfg</code>	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

- `_qm_irq_setup`
    - Interrupt, [26](#)
  - `_qm_register_isr`
    - Interrupt, [26](#)
- `agent_read_en_mask`
  - `qm_mpr_config_t`, [63](#)
- `agent_write_en_mask`
  - `qm_mpr_config_t`, [63](#)
- `allow_agents`
  - `qm_fpr_config_t`, [61](#)
- Always-on Counters, [3](#)
  - `qm_aonc_disable`, [3](#)
  - `qm_aonc_enable`, [3](#)
  - `qm_aonc_get_value`, [5](#)
  - `qm_aonpt_clear`, [5](#)
  - `qm_aonpt_get_config`, [5](#)
  - `qm_aonpt_get_status`, [5](#)
  - `qm_aonpt_get_value`, [6](#)
  - `qm_aonpt_isr_0`, [6](#)
  - `qm_aonpt_reset`, [6](#)
  - `qm_aonpt_set_config`, [6](#)
- Analog Comparator, [8](#)
  - `qm_ac_get_config`, [8](#)
  - `qm_ac_isr`, [8](#)
  - `qm_ac_set_config`, [8](#)
- `bus_mode`
  - `qm_spi_config_t`, [65](#)
- `callback`
  - `qm_pic_timer_config_t`, [63](#)
- `clk_adc_set_div`
  - System Control Subsystem, [42](#)
- `clk_divider`
  - `qm_spi_config_t`, [65](#)
- `clk_ext_div_t`
  - System Control Subsystem, [42](#)
- `clk_ext_set_div`
  - System Control Subsystem, [43](#)
- `clk_gpio_db_div_t`
  - System Control Subsystem, [42](#)
- `clk_gpio_db_set_div`
  - System Control Subsystem, [43](#)
- `clk_periph_disable`
  - System Control Subsystem, [43](#)
- `clk_periph_div_t`
  - System Control Subsystem, [42](#)
- `clk_periph_enable`
  - System Control Subsystem, [43](#)
- `clk_periph_set_div`
  - System Control Subsystem, [43](#)
- `clk_periph_t`
  - System Control Subsystem, [42](#)
- `clk_rtc_div_t`
  - System Control Subsystem, [42](#)
- `clk_rtc_set_div`
  - System Control Subsystem, [44](#)
- `clk_sys_div_t`
  - System Control Subsystem, [42](#)
- `clk_sys_get_ticks_per_us`
  - System Control Subsystem, [44](#)
- `clk_sys_mode_t`
  - System Control Subsystem, [42](#)
- `clk_sys_set_mode`
  - System Control Subsystem, [44](#)
- `clk_sys_udelay`
  - System Control Subsystem, [44](#)
- `cpu_halt`
  - Power state, [35](#)
- `en_lock_mask`
  - `qm_mpr_config_t`, [63](#)
- `en_mask`
  - `qm_fpr_config_t`, [61](#)
- FPR, [13](#)
  - `qm_fpr_get_config`, [14](#)
  - `qm_fpr_id_t`, [14](#)
  - `qm_fpr_isr_0`, [14](#)
  - `qm_fpr_set_config`, [14](#)
  - `qm_fpr_set_violation_policy`, [15](#)
- Flash, [9](#)
  - `qm_flash_disable_t`, [9](#)
  - `qm_flash_get_config`, [10](#)
  - `qm_flash_mass_erase`, [10](#)
  - `qm_flash_page_erase`, [10](#)
  - `qm_flash_page_update`, [10](#)
  - `qm_flash_page_write`, [11](#)
  - `qm_flash_region_t`, [9](#)
  - `qm_flash_set_config`, [11](#)
  - `qm_flash_word_write`, [11](#)
- `frame_size`
  - `qm_spi_config_t`, [65](#)
- GPIO, [16](#)
  - `qm_gpio_clear_pin`, [16](#)
  - `qm_gpio_get_config`, [16](#)
  - `qm_gpio_isr_0`, [16](#)
  - `qm_gpio_read_pin`, [17](#)
  - `qm_gpio_read_port`, [17](#)
  - `qm_gpio_set_config`, [17](#)
  - `qm_gpio_set_pin`, [17](#)
  - `qm_gpio_write_port`, [18](#)
- I2C, [19](#)
  - `qm_i2c_0_isr`, [20](#)
  - `qm_i2c_1_isr`, [20](#)
  - `qm_i2c_addr_t`, [19](#)
  - `qm_i2c_get_config`, [20](#)
  - `qm_i2c_get_status`, [20](#)
  - `qm_i2c_master_irq_transfer`, [20](#)

- qm\_i2c\_master\_read, 21
- qm\_i2c\_master\_write, 21
- qm\_i2c\_mode\_t, 19
- qm\_i2c\_set\_config, 21
- qm\_i2c\_set\_speed, 22
- qm\_i2c\_speed\_t, 19
- qm\_i2c\_status\_t, 20
- qm\_i2c\_transfer\_terminate, 22
- Identification, 23
  - qm\_soc\_id, 23
  - qm\_soc\_version, 23
- Initialisation, 24
  - qm\_soc\_reset, 24
  - qm\_soc\_reset\_t, 24
- int\_en
  - qm\_pic\_timer\_config\_t, 64
- Interrupt, 25
  - \_qm\_irq\_setup, 26
  - \_qm\_register\_isr, 26
  - qm\_int\_vector\_request, 25
  - qm\_irq\_disable, 26
  - qm\_irq\_enable, 27
  - qm\_irq\_mask, 27
  - qm\_irq\_request, 26
  - qm\_irq\_unmask, 27
  - qm\_isr\_t, 26
- low\_bound
  - qm\_fpr\_config\_t, 61
  - qm\_mpr\_config\_t, 63
- MPR, 28
  - qm\_mpr\_get\_config, 28
  - qm\_mpr\_isr, 30
  - qm\_mpr\_set\_config, 30
  - qm\_mpr\_set\_violation\_policy, 30
- mode
  - qm\_pic\_timer\_config\_t, 64
- PIC Timer, 31
  - qm\_pic\_timer\_get, 31
  - qm\_pic\_timer\_get\_config, 31
  - qm\_pic\_timer\_isr, 32
  - qm\_pic\_timer\_mode\_t, 31
  - qm\_pic\_timer\_set, 32
  - qm\_pic\_timer\_set\_config, 32
- PWM / Timer, 36
  - qm\_pwm\_get, 36
  - qm\_pwm\_get\_config, 37
  - qm\_pwm\_isr\_0, 37
  - qm\_pwm\_mode\_t, 36
  - qm\_pwm\_set, 37
  - qm\_pwm\_set\_config, 37
  - qm\_pwm\_start, 38
  - qm\_pwm\_stop, 38
- Pin Muxing setup, 33
  - qm\_pin\_id\_t, 33
  - qm\_pmux\_fn\_t, 33
  - qm\_pmux\_input\_en, 33
  - qm\_pmux\_pullup\_en, 33
  - qm\_pmux\_select, 34
  - qm\_pmux\_set\_slew, 34
  - qm\_pmux\_slew\_t, 33
- Power state, 35
  - cpu\_halt, 35
  - soc\_deep\_sleep, 35
  - soc\_sleep, 35
- QM\_SPI\_BMODE\_0
  - SPI, 46
- QM\_SPI\_BMODE\_1
  - SPI, 46
- QM\_SPI\_BMODE\_2
  - SPI, 46
- QM\_SPI\_BMODE\_3
  - SPI, 46
- QM\_SPI\_TMOD\_EEPROM\_READ
  - SPI, 47
- QM\_SPI\_TMOD\_RX
  - SPI, 47
- QM\_SPI\_TMOD\_TX
  - SPI, 47
- QM\_SPI\_TMOD\_TX\_RX
  - SPI, 47
- QM\_UART\_LC\_5E1
  - UART, 51
- QM\_UART\_LC\_5E1\_5
  - UART, 51
- QM\_UART\_LC\_5N1
  - UART, 51
- QM\_UART\_LC\_5N1\_5
  - UART, 51
- QM\_UART\_LC\_5O1
  - UART, 51
- QM\_UART\_LC\_5O1\_5
  - UART, 51
- QM\_UART\_LC\_6E1
  - UART, 51
- QM\_UART\_LC\_6E2
  - UART, 51
- QM\_UART\_LC\_6N1
  - UART, 51
- QM\_UART\_LC\_6N2
  - UART, 51
- QM\_UART\_LC\_6O1
  - UART, 51
- QM\_UART\_LC\_6O2
  - UART, 51
- QM\_UART\_LC\_7E1
  - UART, 51
- QM\_UART\_LC\_7E2
  - UART, 51
- QM\_UART\_LC\_7N1
  - UART, 51
- QM\_UART\_LC\_7N2
  - UART, 51
- QM\_UART\_LC\_7O1
  - UART, 51



QM\_UART\_LC\_7O2  
     UART, [51](#)  
 QM\_UART\_LC\_8E1  
     UART, [51](#)  
 QM\_UART\_LC\_8E2  
     UART, [51](#)  
 QM\_UART\_LC\_8N1  
     UART, [51](#)  
 QM\_UART\_LC\_8N2  
     UART, [51](#)  
 QM\_UART\_LC\_8O1  
     UART, [51](#)  
 QM\_UART\_LC\_8O2  
     UART, [51](#)  
 QM\_RTC\_ALARM\_DAY  
     RTC, [39](#)  
 QM\_RTC\_ALARM\_HOUR  
     RTC, [39](#)  
 QM\_VER\_API\_MAJOR  
     Version, [57](#)  
 QM\_VER\_API\_MINOR  
     Version, [57](#)  
 QM\_VER\_API\_PATCH  
     Version, [57](#)  
 QM\_VER\_API\_STRING  
     Version, [57](#)  
 QM\_VER\_API\_UINT  
     Version, [57](#)  
 qm\_ac\_config\_t, [60](#)  
 qm\_ac\_get\_config  
     Analog Comparator, [8](#)  
 qm\_ac\_isr  
     Analog Comparator, [8](#)  
 qm\_ac\_set\_config  
     Analog Comparator, [8](#)  
 qm\_aonc\_disable  
     Always-on Counters, [3](#)  
 qm\_aonc\_enable  
     Always-on Counters, [3](#)  
 qm\_aonc\_get\_value  
     Always-on Counters, [5](#)  
 qm\_aonpt\_clear  
     Always-on Counters, [5](#)  
 qm\_aonpt\_config\_t, [60](#)  
 qm\_aonpt\_get\_config  
     Always-on Counters, [5](#)  
 qm\_aonpt\_get\_status  
     Always-on Counters, [5](#)  
 qm\_aonpt\_get\_value  
     Always-on Counters, [6](#)  
 qm\_aonpt\_isr\_0  
     Always-on Counters, [6](#)  
 qm\_aonpt\_reset  
     Always-on Counters, [6](#)  
 qm\_aonpt\_set\_config  
     Always-on Counters, [6](#)  
 qm\_flash\_config\_t, [60](#)  
     us\_count, [60](#)  
     wait\_states, [60](#)  
     write\_disable, [60](#)  
 qm\_flash\_disable\_t  
     Flash, [9](#)  
 qm\_flash\_get\_config  
     Flash, [10](#)  
 qm\_flash\_mass\_erase  
     Flash, [10](#)  
 qm\_flash\_page\_erase  
     Flash, [10](#)  
 qm\_flash\_page\_update  
     Flash, [10](#)  
 qm\_flash\_page\_write  
     Flash, [11](#)  
 qm\_flash\_region\_t  
     Flash, [9](#)  
 qm\_flash\_set\_config  
     Flash, [11](#)  
 qm\_flash\_word\_write  
     Flash, [11](#)  
 qm\_fpr\_config\_t, [61](#)  
     allow\_agents, [61](#)  
     en\_mask, [61](#)  
     low\_bound, [61](#)  
     up\_bound, [61](#)  
 qm\_fpr\_get\_config  
     FPR, [14](#)  
 qm\_fpr\_id\_t  
     FPR, [14](#)  
 qm\_fpr\_isr\_0  
     FPR, [14](#)  
 qm\_fpr\_set\_config  
     FPR, [14](#)  
 qm\_fpr\_set\_violation\_policy  
     FPR, [15](#)  
 qm\_gpio\_clear\_pin  
     GPIO, [16](#)  
 qm\_gpio\_get\_config  
     GPIO, [16](#)  
 qm\_gpio\_isr\_0  
     GPIO, [16](#)  
 qm\_gpio\_port\_config\_t, [62](#)  
 qm\_gpio\_read\_pin  
     GPIO, [17](#)  
 qm\_gpio\_read\_port  
     GPIO, [17](#)  
 qm\_gpio\_set\_config  
     GPIO, [17](#)  
 qm\_gpio\_set\_pin  
     GPIO, [17](#)  
 qm\_gpio\_write\_port  
     GPIO, [18](#)  
 qm\_i2c\_0\_isr  
     I2C, [20](#)  
 qm\_i2c\_1\_isr  
     I2C, [20](#)  
 qm\_i2c\_addr\_t  
     I2C, [19](#)

- qm\_i2c\_config\_t, 62
- qm\_i2c\_get\_config
  - I2C, 20
- qm\_i2c\_get\_status
  - I2C, 20
- qm\_i2c\_master\_irq\_transfer
  - I2C, 20
- qm\_i2c\_master\_read
  - I2C, 21
- qm\_i2c\_master\_write
  - I2C, 21
- qm\_i2c\_mode\_t
  - I2C, 19
- qm\_i2c\_set\_config
  - I2C, 21
- qm\_i2c\_set\_speed
  - I2C, 22
- qm\_i2c\_speed\_t
  - I2C, 19
- qm\_i2c\_status\_t
  - I2C, 20
- qm\_i2c\_transfer\_t, 62
- qm\_i2c\_transfer\_terminate
  - I2C, 22
- qm\_int\_vector\_request
  - Interrupt, 25
- qm\_irq\_disable
  - Interrupt, 26
- qm\_irq\_enable
  - Interrupt, 27
- qm\_irq\_mask
  - Interrupt, 27
- qm\_irq\_request
  - Interrupt, 26
- qm\_irq\_unmask
  - Interrupt, 27
- qm\_isr\_t
  - Interrupt, 26
- qm\_mpr\_config\_t, 62
  - agent\_read\_en\_mask, 63
  - agent\_write\_en\_mask, 63
  - en\_lock\_mask, 63
  - low\_bound, 63
  - up\_bound, 63
- qm\_mpr\_get\_config
  - MPR, 28
- qm\_mpr\_isr
  - MPR, 30
- qm\_mpr\_set\_config
  - MPR, 30
- qm\_mpr\_set\_violation\_policy
  - MPR, 30
- qm\_pic\_timer\_config\_t, 63
  - callback, 63
  - int\_en, 64
  - mode, 64
- qm\_pic\_timer\_get
  - PIC Timer, 31
- qm\_pic\_timer\_get\_config
  - PIC Timer, 31
- qm\_pic\_timer\_isr
  - PIC Timer, 32
- qm\_pic\_timer\_mode\_t
  - PIC Timer, 31
- qm\_pic\_timer\_set
  - PIC Timer, 32
- qm\_pic\_timer\_set\_config
  - PIC Timer, 32
- qm\_pin\_id\_t
  - Pin Muxing setup, 33
- qm\_pmux\_fn\_t
  - Pin Muxing setup, 33
- qm\_pmux\_input\_en
  - Pin Muxing setup, 33
- qm\_pmux\_pullup\_en
  - Pin Muxing setup, 33
- qm\_pmux\_select
  - Pin Muxing setup, 34
- qm\_pmux\_set\_slew
  - Pin Muxing setup, 34
- qm\_pmux\_slew\_t
  - Pin Muxing setup, 33
- qm\_pwm\_config\_t, 64
- qm\_pwm\_get
  - PWM / Timer, 36
- qm\_pwm\_get\_config
  - PWM / Timer, 37
- qm\_pwm\_isr\_0
  - PWM / Timer, 37
- qm\_pwm\_mode\_t
  - PWM / Timer, 36
- qm\_pwm\_set
  - PWM / Timer, 37
- qm\_pwm\_set\_config
  - PWM / Timer, 37
- qm\_pwm\_start
  - PWM / Timer, 38
- qm\_pwm\_stop
  - PWM / Timer, 38
- qm\_rtc\_config\_t, 64
- qm\_rtc\_get\_config
  - RTC, 40
- qm\_rtc\_isr\_0
  - RTC, 40
- qm\_rtc\_set\_alarm
  - RTC, 40
- qm\_rtc\_set\_config
  - RTC, 40
- qm\_soc\_id
  - Identification, 23
- qm\_soc\_reset
  - Initialisation, 24
- qm\_soc\_reset\_t
  - Initialisation, 24
- qm\_soc\_version
  - Identification, 23

- qm\_spi\_async\_transfer\_t, 64
- qm\_spi\_bmode\_t
  - SPI, 46
- qm\_spi\_config\_t, 64
  - bus\_mode, 65
  - clk\_divider, 65
  - frame\_size, 65
  - transfer\_mode, 65
- qm\_spi\_frame\_size\_t
  - SPI, 46
- qm\_spi\_get\_config
  - SPI, 47
- qm\_spi\_get\_status
  - SPI, 47
- qm\_spi\_irq\_transfer
  - SPI, 47
- qm\_spi\_master\_0\_isr
  - SPI, 48
- qm\_spi\_set\_config
  - SPI, 48
- qm\_spi\_slave\_select
  - SPI, 48
- qm\_spi\_slave\_select\_t
  - SPI, 47
- qm\_spi\_tmode\_t
  - SPI, 47
- qm\_spi\_transfer
  - SPI, 49
- qm\_spi\_transfer\_t, 65
- qm\_spi\_transfer\_terminate
  - SPI, 49
- qm\_uart\_0\_isr
  - UART, 51
- qm\_uart\_1\_isr
  - UART, 51
- qm\_uart\_config\_t, 65
- qm\_uart\_get\_config
  - UART, 52
- qm\_uart\_get\_status
  - UART, 52
- qm\_uart\_irq\_read
  - UART, 52
- qm\_uart\_irq\_write
  - UART, 52
- qm\_uart\_lc\_t
  - UART, 51
- qm\_uart\_read
  - UART, 53
- qm\_uart\_read\_non\_block
  - UART, 53
- qm\_uart\_read\_terminate
  - UART, 53
- qm\_uart\_set\_config
  - UART, 53
- qm\_uart\_status\_t
  - UART, 51
- qm\_uart\_transfer\_t, 66
- qm\_uart\_write
  - UART, 55
- qm\_uart\_write\_buffer
  - UART, 55
- qm\_uart\_write\_non\_block
  - UART, 55
- qm\_uart\_write\_terminate
  - UART, 55
- qm\_wdt\_clock\_timeout\_cycles\_t
  - WDT, 58
- qm\_wdt\_config\_t, 66
- qm\_wdt\_get\_config
  - WDT, 58
- qm\_wdt\_isr\_0
  - WDT, 58
- qm\_wdt\_mode\_t
  - WDT, 58
- qm\_wdt\_reload
  - WDT, 59
- qm\_wdt\_set\_config
  - WDT, 59
- qm\_wdt\_start
  - WDT, 59
- RTC, 39
  - QM\_RTC\_ALARM\_DAY, 39
  - QM\_RTC\_ALARM\_HOUR, 39
  - qm\_rtc\_get\_config, 40
  - qm\_rtc\_isr\_0, 40
  - qm\_rtc\_set\_alarm, 40
  - qm\_rtc\_set\_config, 40
- SPI
  - QM\_SPI\_BMODE\_0, 46
  - QM\_SPI\_BMODE\_1, 46
  - QM\_SPI\_BMODE\_2, 46
  - QM\_SPI\_BMODE\_3, 46
  - QM\_SPI\_TMOD\_EEPROM\_READ, 47
  - QM\_SPI\_TMOD\_RX, 47
  - QM\_SPI\_TMOD\_TX, 47
  - QM\_SPI\_TMOD\_TX\_RX, 47
- SPI, 46
  - qm\_spi\_bmode\_t, 46
  - qm\_spi\_frame\_size\_t, 46
  - qm\_spi\_get\_config, 47
  - qm\_spi\_get\_status, 47
  - qm\_spi\_irq\_transfer, 47
  - qm\_spi\_master\_0\_isr, 48
  - qm\_spi\_set\_config, 48
  - qm\_spi\_slave\_select, 48
  - qm\_spi\_slave\_select\_t, 47
  - qm\_spi\_tmode\_t, 47
  - qm\_spi\_transfer, 49
  - qm\_spi\_transfer\_terminate, 49
- soc\_deep\_sleep
  - Power state, 35
- soc\_sleep
  - Power state, 35
- System Control Subsystem, 41
  - clk\_adc\_set\_div, 42

- clk\_ext\_div\_t, 42
- clk\_ext\_set\_div, 43
- clk\_gpio\_db\_div\_t, 42
- clk\_gpio\_db\_set\_div, 43
- clk\_periph\_disable, 43
- clk\_periph\_div\_t, 42
- clk\_periph\_enable, 43
- clk\_periph\_set\_div, 43
- clk\_periph\_t, 42
- clk\_rtc\_div\_t, 42
- clk\_rtc\_set\_div, 44
- clk\_sys\_div\_t, 42
- clk\_sys\_get\_ticks\_per\_us, 44
- clk\_sys\_mode\_t, 42
- clk\_sys\_set\_mode, 44
- clk\_sys\_udelay, 44
- transfer\_mode
  - qm\_spi\_config\_t, 65
- UART
  - QM\_UART\_LC\_5E1, 51
  - QM\_UART\_LC\_5E1\_5, 51
  - QM\_UART\_LC\_5N1, 51
  - QM\_UART\_LC\_5N1\_5, 51
  - QM\_UART\_LC\_5O1, 51
  - QM\_UART\_LC\_5O1\_5, 51
  - QM\_UART\_LC\_6E1, 51
  - QM\_UART\_LC\_6E2, 51
  - QM\_UART\_LC\_6N1, 51
  - QM\_UART\_LC\_6N2, 51
  - QM\_UART\_LC\_6O1, 51
  - QM\_UART\_LC\_6O2, 51
  - QM\_UART\_LC\_7E1, 51
  - QM\_UART\_LC\_7E2, 51
  - QM\_UART\_LC\_7N1, 51
  - QM\_UART\_LC\_7N2, 51
  - QM\_UART\_LC\_7O1, 51
  - QM\_UART\_LC\_7O2, 51
  - QM\_UART\_LC\_8E1, 51
  - QM\_UART\_LC\_8E2, 51
  - QM\_UART\_LC\_8N1, 51
  - QM\_UART\_LC\_8N2, 51
  - QM\_UART\_LC\_8O1, 51
  - QM\_UART\_LC\_8O2, 51
- UART, 50
  - qm\_uart\_0\_isr, 51
  - qm\_uart\_1\_isr, 51
  - qm\_uart\_get\_config, 52
  - qm\_uart\_get\_status, 52
  - qm\_uart\_irq\_read, 52
  - qm\_uart\_irq\_write, 52
  - qm\_uart\_lc\_t, 51
  - qm\_uart\_read, 53
  - qm\_uart\_read\_non\_block, 53
  - qm\_uart\_read\_terminate, 53
  - qm\_uart\_set\_config, 53
  - qm\_uart\_status\_t, 51
  - qm\_uart\_write, 55
  - qm\_uart\_write\_buffer, 55
  - qm\_uart\_write\_non\_block, 55
  - qm\_uart\_write\_terminate, 55
- up\_bound
  - qm\_fpr\_config\_t, 61
  - qm\_mpr\_config\_t, 63
- us\_count
  - qm\_flash\_config\_t, 60
- Version, 57
  - QM\_VER\_API\_MAJOR, 57
  - QM\_VER\_API\_MINOR, 57
  - QM\_VER\_API\_PATCH, 57
  - QM\_VER\_API\_STRING, 57
  - QM\_VER\_API\_UINT, 57
- WDT, 58
  - qm\_wdt\_clock\_timeout\_cycles\_t, 58
  - qm\_wdt\_get\_config, 58
  - qm\_wdt\_isr\_0, 58
  - qm\_wdt\_mode\_t, 58
  - qm\_wdt\_reload, 59
  - qm\_wdt\_set\_config, 59
  - qm\_wdt\_start, 59
- wait\_states
  - qm\_flash\_config\_t, 60
- write\_disable
  - qm\_flash\_config\_t, 60