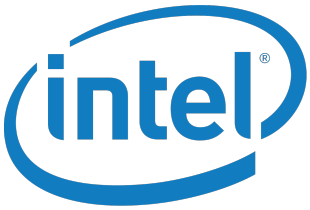

Intel® Quark™ Microcontroller Software Interface 1.1

July, 2016

Document Number: 333612-002EN

Contents

1 Licensing information

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 Module Documentation

4.1 Error Codes

This project uses <errno.h> for error codes.

The following return codes are used:

```
-EINVAL         /* Invalid parameters.                  */
-EIO            /* Operation failed.                    */
-ECANCELED      /* User terminated the transaction.     */
```

4.2 Clock Management

Clock Management.

Enumerations

- enum clk_sys_div_t {
  CLK_SYS_DIV_1, CLK_SYS_DIV_2, CLK_SYS_DIV_4, CLK_SYS_DIV_8,
  CLK_SYS_DIV_16, CLK_SYS_DIV_32, CLK_SYS_DIV_64, CLK_SYS_DIV_128 }

  System clock divider type.
- enum clk_sys_mode_t {
  CLK_SYS_HYB_OSC_32MHZ, CLK_SYS_HYB_OSC_16MHZ, CLK_SYS_HYB_OSC_8MHZ, CLK_SYS_-
  HYB_OSC_4MHZ,
  CLK_SYS_RTC_OSC, CLK_SYS_CRYSTAL_OSC }

  System clock mode type.
- enum clk_periph_div_t { CLK_PERIPH_DIV_1, CLK_PERIPH_DIV_2, CLK_PERIPH_DIV_4, CLK_PERIPH-
  _DIV_8 }

  Peripheral clock divider type.
- enum clk_gpio_db_div_t {
  CLK_GPIO_DB_DIV_1, CLK_GPIO_DB_DIV_2, CLK_GPIO_DB_DIV_4, CLK_GPIO_DB_DIV_8,
  CLK_GPIO_DB_DIV_16, CLK_GPIO_DB_DIV_32, CLK_GPIO_DB_DIV_64, CLK_GPIO_DB_DIV_128 }

  GPIO clock debounce divider type.
- enum clk_ext_div_t { CLK_EXT_DIV_1, CLK_EXT_DIV_2, CLK_EXT_DIV_4, CLK_EXT_DIV_8 }

  External crystal clock divider type.
- enum clk_rtc_div_t {
  CLK_RTC_DIV_1, CLK_RTC_DIV_2, CLK_RTC_DIV_4, CLK_RTC_DIV_8,
  CLK_RTC_DIV_16, CLK_RTC_DIV_32, CLK_RTC_DIV_64, CLK_RTC_DIV_128,
  CLK_RTC_DIV_256, CLK_RTC_DIV_512, CLK_RTC_DIV_1024, CLK_RTC_DIV_2048,
  CLK_RTC_DIV_4096, CLK_RTC_DIV_8192, CLK_RTC_DIV_16384, CLK_RTC_DIV_32768 }

  RTC clock divider type.

Functions

- int clk_sys_set_mode (const clk_sys_mode_t mode, const clk_sys_div_t div)

  Set clock mode and divisor.
- int clk_trim_read (uint32_t *const value)

  Read the silicon oscillator trim code for the current frequency.
- int clk_trim_apply (const uint32_t value)

  Apply silicon oscillator trim code.
- int clk_adc_set_div (const uint16_t div)

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

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

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

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

  Change divider value of RTC.
- int clk_periph_enable (const clk_periph_t clocks)

  Enable clocks for peripherals / registers.
- int clk_periph_disable (const clk_periph_t clocks)

Disable clocks for peripherals / registers.

- uint32_t clk_sys_get_ticks_per_us (void)

  Get number of system ticks per micro second.

- void clk_sys_udelay (uint32_t microseconds)

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

4.2.1 Detailed Description

Clock Management.

4.2.2 Enumeration Type Documentation

4.2.2.1 enum clk_ext_div_t

External crystal clock divider type.

Enumerator

CLK_EXT_DIV_1 External Crystal Clock Divider = 1.

CLK_EXT_DIV_2 External Crystal Clock Divider = 2.

CLK_EXT_DIV_4 External Crystal Clock Divider = 4.

CLK_EXT_DIV_8 External Crystal Clock Divider = 8.

Definition at line 92 of file clk.h.

4.2.2.2 enum clk_gpio_db_div_t

GPIO clock debounce divider type.

Enumerator

CLK_GPIO_DB_DIV_1 GPIO Clock Debounce Divider = 1.

CLK_GPIO_DB_DIV_2 GPIO Clock Debounce Divider = 2.

CLK_GPIO_DB_DIV_4 GPIO Clock Debounce Divider = 4.

CLK_GPIO_DB_DIV_8 GPIO Clock Debounce Divider = 8.

CLK_GPIO_DB_DIV_16 GPIO Clock Debounce Divider = 16.

CLK_GPIO_DB_DIV_32 GPIO Clock Debounce Divider = 32.

CLK_GPIO_DB_DIV_64 GPIO Clock Debounce Divider = 64.

CLK_GPIO_DB_DIV_128 GPIO Clock Debounce Divider = 128.

Definition at line 78 of file clk.h.

4.2.2.3 enum clk_periph_div_t

Peripheral clock divider type.

Enumerator

CLK_PERIPH_DIV_1 Peripheral Clock Divider = 1.

CLK_PERIPH_DIV_2 Peripheral Clock Divider = 2.

CLK_PERIPH_DIV_4 Peripheral Clock Divider = 4.

CLK_PERIPH_DIV_8 Peripheral Clock Divider = 8.

Definition at line 68 of file clk.h.

4.2.2.4 enum clk_rtc_div_t

RTC clock divider type.

Enumerator

CLK_RTC_DIV_1 Real Time Clock Divider = 1.
CLK_RTC_DIV_2 Real Time Clock Divider = 2.
CLK_RTC_DIV_4 Real Time Clock Divider = 4.
CLK_RTC_DIV_8 Real Time Clock Divider = 8.
CLK_RTC_DIV_16 Real Time Clock Divider = 16.
CLK_RTC_DIV_32 Real Time Clock Divider = 32.
CLK_RTC_DIV_64 Real Time Clock Divider = 64.
CLK_RTC_DIV_128 Real Time Clock Divider = 128.
CLK_RTC_DIV_256 Real Time Clock Divider = 256.
CLK_RTC_DIV_512 Real Time Clock Divider = 512.
CLK_RTC_DIV_1024 Real Time Clock Divider = 1024.
CLK_RTC_DIV_2048 Real Time Clock Divider = 2048.
CLK_RTC_DIV_4096 Real Time Clock Divider = 4096.
CLK_RTC_DIV_8192 Real Time Clock Divider = 8192.
CLK_RTC_DIV_16384 Real Time Clock Divider = 16384.
CLK_RTC_DIV_32768 Real Time Clock Divider = 32768.

Definition at line 102 of file clk.h.

4.2.2.5 enum clk_sys_div_t

System clock divider type.

Enumerator

CLK_SYS_DIV_1 Clock Divider = 1.
CLK_SYS_DIV_2 Clock Divider = 2.
CLK_SYS_DIV_4 Clock Divider = 4.
CLK_SYS_DIV_8 Clock Divider = 8.
CLK_SYS_DIV_16 Clock Divider = 16.
CLK_SYS_DIV_32 Clock Divider = 32.
CLK_SYS_DIV_64 Clock Divider = 64.
CLK_SYS_DIV_128 Clock Divider = 128.

Definition at line 39 of file clk.h.

4.2.2.6 enum clk_sys_mode_t

System clock mode type.

Enumerator

CLK_SYS_HYB_OSC_32MHZ 32MHz Hybrid Oscillator Clock.
CLK_SYS_HYB_OSC_16MHZ 16MHz Hybrid Oscillator Clock.
CLK_SYS_HYB_OSC_8MHZ 8MHz Hybrid Oscillator Clock.
CLK_SYS_HYB_OSC_4MHZ 4MHz Hybrid Oscillator Clock.
CLK_SYS_RTC_OSC Real Time Clock.
CLK_SYS_CRYSTAL_OSC Crystal Oscillator Clock.

Definition at line 56 of file clk.h.

4.2.3 Function Documentation

4.2.3.1 int 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. This function is only available on D2000.

Parameters

in	*div*	Divider value for the ADC clock.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 259 of file clk.c.

4.2.3.2 int 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

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 316 of file clk.c.

References CLK_EXT_DIV_8.

4.2.3.3 int 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

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 302 of file clk.c.

References CLK_GPIO_DB_DIV_128.

4.2.3.4 int clk_periph_disable (const clk_periph_t *clocks*)

Disable clocks for peripherals / registers.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 354 of file clk.c.

References CLK_PERIPH_ALL, SOCW_EVENT_REGISTER, and SOCW_REG_CCU_PERIPH_CLK_GATE_CTL.

Referenced by power_soc_deep_sleep(), and power_soc_sleep().

4.2.3.5 int clk_periph_enable (const clk_periph_t *clocks*)

Enable clocks for peripherals / registers.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 342 of file clk.c.

References CLK_PERIPH_ALL, SOCW_EVENT_REGISTER, and SOCW_REG_CCU_PERIPH_CLK_GATE_CTL.

Referenced by power_soc_deep_sleep().

4.2.3.6 int 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. Refer to the list of supported peripherals for your SoC.

Parameters

in	*div*	Divider value for the peripheral clock.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 280 of file clk.c.

References CLK_PERIPH_DIV_8.

4.2.3.7 int clk_rtc_set_div (const clk_rtc_div_t *div*)

Change divider value of RTC.

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

Parameters

in	*div*	Divider value for the RTC.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 329 of file clk.c.

References CLK_RTC_DIV_32768.

Referenced by qm_rtc_set_config().

4.2.3.8 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 366 of file clk.c.

Referenced by ss_clk_adc_set_div().

4.2.3.9 int clk_sys_set_mode (const clk_sys_mode_t *mode*, const clk_sys_div_t *div*)

Set clock mode and divisor.

Change the operating mode and clock divisor of the system clock source. Changing this clock speed affects all peripherals. This applies the correct trim code if available.

If trim code is not available, it is not computed and previous trim code is not modified.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 109 of file clk.c.

References CLK_SYS_CRYSTAL_OSC, CLK_SYS_HYB_OSC_16MHZ, CLK_SYS_HYB_OSC_32MHZ, CLK_SYS_HYB_OSC_4MHZ, CLK_SYS_HYB_OSC_8MHZ, CLK_SYS_RTC_OSC, clk_trim_apply(), SOCW_EVENT_REGISTER, SOCW_REG_CCU_SYS_CLK_CTL, and SOCW_REG_OSC0_CFG1.

Referenced by power_soc_deep_sleep(), and power_soc_sleep().

4.2.3.10 void clk_sys_udelay (uint32_t *microseconds*)

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

This function will wait until at least the given number of microseconds has elapsed since calling this function.

Note: It is dependent on the system clock speed. The delay parameter does not include, calling the function, returning from it, calculation setup and while loops.

Parameters

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

Definition at line 371 of file clk.c.

Referenced by clk_trim_apply().

4.2.3.11 int clk_trim_apply (const uint32_t *value*)

Apply silicon oscillator trim code.

Parameters

in	*value*	Trim code to apply.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 234 of file clk.c.

References clk_sys_udelay().

Referenced by clk_sys_set_mode().

4.2.3.12 int clk_trim_read (uint32_t *const *value*)

Read the silicon oscillator trim code for the current frequency.

Parameters

out	*value*	Pointer to store the trim code.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 224 of file clk.c.

4.3 Quark D2000 ADC

Analog to Digital Converter (ADC).

Data Structures

- struct qm_adc_config_t

  ADC configuration type.
- struct qm_adc_xfer_t

  ADC transfer type.

Typedefs

- typedef uint16_t qm_adc_sample_t

  ADC sample size type.
- typedef uint8_t qm_adc_calibration_t

  ADC calibration type.

Enumerations

- enum qm_adc_status_t { QM_ADC_IDLE, QM_ADC_COMPLETE, QM_ADC_OVERFLOW }
- enum qm_adc_resolution_t { QM_ADC_RES_6_BITS, QM_ADC_RES_8_BITS, QM_ADC_RES_10_BITS, QM_ADC_RES_12_BITS }

  ADC resolution type.
- enum qm_adc_mode_t {
  QM_ADC_MODE_DEEP_PWR_DOWN, QM_ADC_MODE_PWR_DOWN, QM_ADC_MODE_STDBY, QM_ADC_MODE_NORM_CAL,
  QM_ADC_MODE_NORM_NO_CAL }

  ADC operating mode type.
- enum qm_adc_channel_t {
  QM_ADC_CH_0, QM_ADC_CH_1, QM_ADC_CH_2, QM_ADC_CH_3,
  QM_ADC_CH_4, QM_ADC_CH_5, QM_ADC_CH_6, QM_ADC_CH_7,
  QM_ADC_CH_8, QM_ADC_CH_9, QM_ADC_CH_10, QM_ADC_CH_11,
  QM_ADC_CH_12, QM_ADC_CH_13, QM_ADC_CH_14, QM_ADC_CH_15,
  QM_ADC_CH_16, QM_ADC_CH_17, QM_ADC_CH_18 }

  ADC channels type.
- enum qm_adc_cb_source_t { QM_ADC_TRANSFER, QM_ADC_MODE_CHANGED, QM_ADC_CAL_COMPLETE }

  ADC interrupt callback source.

Functions

- int qm_adc_set_mode (const qm_adc_t adc, const qm_adc_mode_t mode)

  Switch operating mode of ADC.
- int qm_adc_irq_set_mode (const qm_adc_t adc, const qm_adc_mode_t mode, void(∗callback)(void ∗data, int error, qm_adc_status_t status, qm_adc_cb_source_t source), void ∗callback_data)

  Switch operating mode of ADC.
- int qm_adc_calibrate (const qm_adc_t adc)

  Calibrate the ADC.
- int qm_adc_irq_calibrate (const qm_adc_t adc, void(∗callback)(void ∗data, int error, qm_adc_status_t status, qm_adc_cb_source_t source), void ∗callback_data)

  Calibrate the ADC.

- int qm_adc_set_calibration (const qm_adc_t adc, const qm_adc_calibration_t cal)
  Set ADC calibration data.
- int qm_adc_get_calibration (const qm_adc_t adc, qm_adc_calibration_t *const cal)
  Get the current calibration data for an ADC.
- int qm_adc_set_config (const qm_adc_t adc, const qm_adc_config_t *const cfg)
  Set ADC configuration.
- int qm_adc_convert (const qm_adc_t adc, qm_adc_xfer_t *const xfer, qm_adc_status_t *const status)
  Synchronously read values from the ADC.
- int qm_adc_irq_convert (const qm_adc_t adc, qm_adc_xfer_t *const xfer)
  Asynchronously read values from the ADC.

4.3.1 Detailed Description

Analog to Digital Converter (ADC).

4.3.2 Enumeration Type Documentation

4.3.2.1 enum qm_adc_cb_source_t

ADC interrupt callback source.

Enumerator

QM_ADC_TRANSFER Transfer complete or error callback.

QM_ADC_MODE_CHANGED Mode change complete callback.

QM_ADC_CAL_COMPLETE Calibration complete callback.

Definition at line 85 of file qm_adc.h.

4.3.2.2 enum qm_adc_channel_t

ADC channels type.

Enumerator

QM_ADC_CH_0 ADC Channel 0.

QM_ADC_CH_1 ADC Channel 1.

QM_ADC_CH_2 ADC Channel 2.

QM_ADC_CH_3 ADC Channel 3.

QM_ADC_CH_4 ADC Channel 4.

QM_ADC_CH_5 ADC Channel 5.

QM_ADC_CH_6 ADC Channel 6.

QM_ADC_CH_7 ADC Channel 7.

QM_ADC_CH_8 ADC Channel 8.

QM_ADC_CH_9 ADC Channel 9.

QM_ADC_CH_10 ADC Channel 10.

QM_ADC_CH_11 ADC Channel 11.

QM_ADC_CH_12 ADC Channel 12.

QM_ADC_CH_13 ADC Channel 13.

QM_ADC_CH_14 ADC Channel 14.

QM_ADC_CH_15 ADC Channel 15.

QM_ADC_CH_16 ADC Channel 16.

QM_ADC_CH_17 ADC Channel 17.

QM_ADC_CH_18 ADC Channel 18.

Definition at line 60 of file qm_adc.h.

4.3.2.3 enum qm_adc_mode_t

ADC operating mode type.

Enumerator

QM_ADC_MODE_DEEP_PWR_DOWN Deep power down mode.

QM_ADC_MODE_PWR_DOWN Power down mode.

QM_ADC_MODE_STDBY Standby mode.

QM_ADC_MODE_NORM_CAL Normal mode, with calibration.

QM_ADC_MODE_NORM_NO_CAL Normal mode, no calibration.

Definition at line 49 of file qm_adc.h.

4.3.2.4 enum qm_adc_resolution_t

ADC resolution type.

Enumerator

QM_ADC_RES_6_BITS 6-bit mode.

QM_ADC_RES_8_BITS 8-bit mode.

QM_ADC_RES_10_BITS 10-bit mode.

QM_ADC_RES_12_BITS 12-bit mode.

Definition at line 39 of file qm_adc.h.

4.3.2.5 enum qm_adc_status_t

Enumerator

QM_ADC_IDLE ADC idle.

QM_ADC_COMPLETE ADC transfer complete.

QM_ADC_OVERFLOW ADC FIFO overflow error.

Definition at line 30 of file qm_adc.h.

4.3.3 Function Documentation

4.3.3.1 int qm_adc_calibrate (const qm_adc_t *adc*)

Calibrate the ADC.

It is necessary to calibrate if it is intended to use Normal Mode With Calibration. The calibration must be performed if the ADC is used for the first time or has been in deep power down mode. This call is blocking.

Parameters

in	*adc*	Which ADC to calibrate.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 206 of file qm_adc.c.

4.3.3.2 int qm_adc_convert (const qm_adc_t *adc*, qm_adc_xfer_t *const *xfer*, qm_adc_status_t *const *status*)

Synchronously read values from the ADC.

This blocking call can read 1-32 ADC values into the array provided.

Parameters

in	*adc*	Which ADC to read.
in,out	*xfer*	Channel and sample info. This must not be NULL.
out	*status*	Get status of the ADC device.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 347 of file qm_adc.c.

References qm_adc_xfer_t::ch, qm_adc_xfer_t::ch_len, QM_ADC_COMPLETE, qm_adc_xfer_t::samples, and qm_adc_xfer_t::samples_len.

4.3.3.3 int qm_adc_get_calibration (const qm_adc_t *adc*, qm_adc_calibration_t *const *cal*)

Get the current calibration data for an ADC.

Parameters

in	*adc*	Which ADC to get calibration for.
out	*cal*	Calibration data. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 263 of file qm_adc.c.

4.3.3.4 int qm_adc_irq_calibrate (const qm_adc_t *adc*, void(*)(void *data, int error, qm_adc_status_t status, qm_adc_cb_source_t source) *callback*, void * *callback_data*)

Calibrate the ADC.

It is necessary to calibrate if it is intended to use Normal Mode With Calibration. The calibration must be performed if the ADC is used for the first time or has been in deep power down mode. This call is non-blocking and will call the user callback on completion.

Parameters

in	*adc*	Which ADC to calibrate.
in	*callback*	Callback called on completion.
in	*callback_data*	The callback user data.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 222 of file qm_adc.c.

4.3.3.5 int qm_adc_irq_convert (const qm_adc_t *adc*, qm_adc_xfer_t *const *xfer*)

Asynchronously read values from the ADC.

This is a non-blocking call and will call the user provided callback after the requested number of samples have been converted.

Parameters

in	*adc*	Which ADC to read.
in,out	*xfer*	Channel sample and callback info. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 393 of file qm_adc.c.

References qm_adc_xfer_t::ch, qm_adc_xfer_t::ch_len, qm_adc_xfer_t::samples, and qm_adc_xfer_t::samples_len.

4.3.3.6 int qm_adc_irq_set_mode (const qm_adc_t *adc*, const qm_adc_mode_t *mode*, void(*)(void *data, int error, qm_adc_status_t status, qm_adc_cb_source_t source) *callback*, void * *callback_data*)

Switch operating mode of ADC.

This call is non-blocking and will call the user callback on completion.

Parameters

in	*adc*	Which ADC to enable.
in	*mode*	ADC operating mode.
in	*callback*	Callback called on completion.

in	*callback_data*	The callback user data.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 304 of file qm_adc.c.

References QM_ADC_MODE_NORM_CAL, and QM_ADC_MODE_NORM_NO_CAL.

4.3.3.7 int qm_adc_set_calibration (const qm_adc_t *adc*, const qm_adc_calibration_t *cal*)

Set ADC calibration data.

Parameters

in	*adc*	Which ADC to set calibration for.
in	*cal*	Calibration data.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 244 of file qm_adc.c.

4.3.3.8 int qm_adc_set_config (const qm_adc_t *adc*, const qm_adc_config_t *const *cfg*)

Set ADC configuration.

This sets the sample window and resolution.

Parameters

in	*adc*	Which ADC to configure.
in	*cfg*	ADC configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 331 of file qm_adc.c.

References QM_ADC_RES_12_BITS, qm_adc_config_t::resolution, and qm_adc_config_t::window.

4.3.3.9 int qm_adc_set_mode (const qm_adc_t *adc*, const qm_adc_mode_t *mode*)

Switch operating mode of ADC.

This call is blocking.

Parameters

in	*adc*	Which ADC to enable.
in	*mode*	ADC operating mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 273 of file qm_adc.c.

References QM_ADC_MODE_NORM_CAL, and QM_ADC_MODE_NORM_NO_CAL.

Referenced by power_soc_deep_sleep(), and power_soc_sleep().

4.4 Always-on Counters

Always-on Counters.

Data Structures

- struct qm_aonpt_config_t

  Always-on Periodic Timer configuration type.

Functions

- int qm_aonc_enable (const qm_scss_aon_t aonc)

  Enable the Always-on Counter.
- int qm_aonc_disable (const qm_scss_aon_t aonc)

  Disable the Always-on Counter.
- int qm_aonc_get_value (const qm_scss_aon_t aonc, uint32_t *const val)

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

  Set the Always-on Periodic Timer configuration.
- int qm_aonpt_get_value (const qm_scss_aon_t aonc, uint32_t *const val)

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

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

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

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

4.4.1 Detailed Description

Always-on Counters.

4.4.2 Function Documentation

4.4.2.1 int qm_aonc_disable (const qm_scss_aon_t *aonc*)

Disable the Always-on Counter.

Parameters

in	*aonc*	Always-on counter to read.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Negative	errno for possible error codes.

Definition at line 54 of file qm_aon_counters.c.

4.4.2.2 int qm_aonc_enable (const qm_scss_aon_t *aonc*)

Enable the Always-on Counter.

Parameters

in	*aonc*	Always-on counter to read.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 45 of file qm_aon_counters.c.

4.4.2.3 int qm_aonc_get_value (const qm_scss_aon_t *aonc*, uint32_t *const *const *val*)

Get the current value of the Always-on Counter.

Returns a 32-bit value which represents the number of clock cycles since the counter was first enabled.

Parameters

in	*aonc*	Always-on counter to read.
out	*val*	Value of the counter. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 63 of file qm_aon_counters.c.

4.4.2.4 int qm_aonpt_clear (const qm_scss_aon_t *aonc*)

Clear the status of the Always-on Periodic Timer.

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

Parameters

in	*aonc*	Always-on counter to read.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 109 of file qm_aon_counters.c.

4.4.2.5 int qm_aonpt_get_status (const qm_scss_aon_t *aonc*, bool *const *status*)

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

Returns true if the timer has expired. This will continue to return true until it is cleared with qm_aonpt_clear().

Parameters

in	*aonc*	Always-on counter to read.
out	*status*	Status of the Always-on Periodic Timer. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 100 of file qm_aon_counters.c.

4.4.2.6 int qm_aonpt_get_value (const qm_scss_aon_t *aonc*, uint32_t *const *val*)

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

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

Parameters

in	*aonc*	Always-on counter to read.
out	*val*	Value of the Always-on Periodic Timer. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 91 of file qm_aon_counters.c.

4.4.2.7 int qm_aonpt_reset (const qm_scss_aon_t *aonc*)

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

Parameters

in	*aonc*	Always-on counter to read.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 118 of file qm_aon_counters.c.

4.4.2.8 int qm_aonpt_set_config (const qm_scss_aon_t *aonc*, const qm_aonpt_config_t *const *cfg*)

Set the Always-on Periodic Timer configuration.

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

Parameters

in	*aonc*	Always-on counter to read.
in	*cfg*	New configuration for the Always-on Periodic Timer. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 72 of file qm_aon_counters.c.

References qm_aonpt_config_t::callback, qm_aonpt_config_t::callback_data, qm_aonpt_config_t::count, and qm_aonpt_config_t::int_en.

4.5 Analog Comparator

Analog Comparator.

Data Structures

- struct qm_ac_config_t

  Analog Comparator configuration type.

Functions

- int qm_ac_set_config (const qm_ac_config_t *const config)

  Set Analog Comparator configuration.

4.5.1 Detailed Description

Analog Comparator.

4.5.2 Function Documentation

4.5.2.1 int qm_ac_set_config (const qm_ac_config_t *const *config*)

Set Analog Comparator configuration.

Parameters

in	*config*	Analog Comparator configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 42 of file qm_comparator.c.

References qm_ac_config_t::callback, qm_ac_config_t::callback_data, qm_ac_config_t::int_en, qm_ac_config_t::polarity, qm_ac_config_t::power, and qm_ac_config_t::reference.

4.6 DMA

DMA Driver for Quark Microcontrollers.

Data Structures

- struct qm_dma_channel_config_t

  DMA channel configuration structure.
- struct qm_dma_transfer_t

  DMA transfer configuration structure.

Enumerations

- enum qm_dma_handshake_polarity_t { QM_DMA_HANDSHAKE_POLARITY_HIGH = 0x0, QM_DMA_HANDSHAKE_POLARITY_LOW = 0x1 }

  DMA Handshake Polarity.
- enum qm_dma_burst_length_t { QM_DMA_BURST_TRANS_LENGTH_1 = 0x0, QM_DMA_BURST_TRANS_LENGTH_4 = 0x1, QM_DMA_BURST_TRANS_LENGTH_8 = 0x2, QM_DMA_BURST_TRANS_LENGTH_16 = 0x3, QM_DMA_BURST_TRANS_LENGTH_32 = 0x4, QM_DMA_BURST_TRANS_LENGTH_64 = 0x5, QM_DMA_BURST_TRANS_LENGTH_128, QM_DMA_BURST_TRANS_LENGTH_256 = 0x7 }

  DMA Burst Transfer Length.
- enum qm_dma_transfer_width_t { QM_DMA_TRANS_WIDTH_8 = 0x0, QM_DMA_TRANS_WIDTH_16 = 0x1, QM_DMA_TRANS_WIDTH_32 = 0x2, QM_DMA_TRANS_WIDTH_64 = 0x3, QM_DMA_TRANS_WIDTH_128 = 0x4, QM_DMA_TRANS_WIDTH_256 = 0x5 }

  DMA Transfer Width.
- enum qm_dma_channel_direction_t { QM_DMA_MEMORY_TO_MEMORY = 0x0, QM_DMA_MEMORY_TO_PERIPHERAL, QM_DMA_PERIPHERAL_TO_MEMORY = 0x2 }

  DMA channel direction.

Functions

- int qm_dma_init (const qm_dma_t dma)

  Initialise the DMA controller.
- int qm_dma_channel_set_config (const qm_dma_t dma, const qm_dma_channel_id_t channel_id, qm_dma_channel_config_t *const channel_config)

  Setup a DMA channel configuration.
- int qm_dma_transfer_set_config (const qm_dma_t dma, const qm_dma_channel_id_t channel_id, qm_dma_transfer_t *const transfer_config)

  Setup a DMA channel transfer.
- int qm_dma_transfer_start (const qm_dma_t dma, const qm_dma_channel_id_t channel_id)

  Start a DMA transfer.
- int qm_dma_transfer_terminate (const qm_dma_t dma, const qm_dma_channel_id_t channel_id)

  Terminate a DMA transfer.
- int qm_dma_transfer_mem_to_mem (const qm_dma_t dma, const qm_dma_channel_id_t channel_id, qm_dma_transfer_t *const transfer_config)

  Setup and start memory to memory transfer.

4.6.1 Detailed Description

DMA Driver for Quark Microcontrollers.

4.6.2 Enumeration Type Documentation

4.6.2.1 enum qm_dma_burst_length_t

DMA Burst Transfer Length.

Enumerator

QM_DMA_BURST_TRANS_LENGTH_1 Burst length 1 data item.
QM_DMA_BURST_TRANS_LENGTH_4 Burst length 4 data items.
QM_DMA_BURST_TRANS_LENGTH_8 Burst length 8 data items.
QM_DMA_BURST_TRANS_LENGTH_16 Burst length 16 data items.
QM_DMA_BURST_TRANS_LENGTH_32 Burst length 32 data items.
QM_DMA_BURST_TRANS_LENGTH_64 Burst length 64 data items.
QM_DMA_BURST_TRANS_LENGTH_128 Burst length 128 data items.
QM_DMA_BURST_TRANS_LENGTH_256 Burst length 256 data items.

Definition at line 29 of file qm_dma.h.

4.6.2.2 enum qm_dma_channel_direction_t

DMA channel direction.

Enumerator

QM_DMA_MEMORY_TO_MEMORY Memory to memory transfer.
QM_DMA_MEMORY_TO_PERIPHERAL Memory to peripheral transfer.
QM_DMA_PERIPHERAL_TO_MEMORY Peripheral to memory transfer.

Definition at line 56 of file qm_dma.h.

4.6.2.3 enum qm_dma_handshake_polarity_t

DMA Handshake Polarity.

Enumerator

QM_DMA_HANDSHAKE_POLARITY_HIGH Set HS polarity high.
QM_DMA_HANDSHAKE_POLARITY_LOW Set HS polarity low.

Definition at line 21 of file qm_dma.h.

4.6.2.4 enum qm_dma_transfer_width_t

DMA Transfer Width.

Enumerator

QM_DMA_TRANS_WIDTH_8 Transfer width of 8 bits.
QM_DMA_TRANS_WIDTH_16 Transfer width of 16 bits.
QM_DMA_TRANS_WIDTH_32 Transfer width of 32 bits.
QM_DMA_TRANS_WIDTH_64 Transfer width of 64 bits.
QM_DMA_TRANS_WIDTH_128 Transfer width of 128 bits.
QM_DMA_TRANS_WIDTH_256 Transfer width of 256 bits.

Definition at line 44 of file qm_dma.h.

4.6.3 Function Documentation

4.6.3.1 int qm_dma_channel_set_config (const qm_dma_t *dma*, const qm_dma_channel_id_t *channel_id*, qm_dma_channel_config_t *const *channel_config*)

Setup a DMA channel configuration.

Configures the channel source width, burst size, channel direction, handshaking interface and registers the client callback and callback context. qm_dma_init() must first be called before configuring a channel. This function only needs to be called once unless a channel is being repurposed.

Parameters

in	*dma*	DMA instance.
in	*channel_id*	The channel to start.
in	*channel_config*	The DMA channel configuration as defined by the DMA client. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 189 of file qm_dma.c.

References qm_dma_channel_config_t::callback_context, qm_dma_channel_config_t::channel_direction, qm_dma_channel_config_t::client_callback, qm_dma_channel_config_t::destination_burst_length, qm_dma_channel_config_t::destination_transfer_width, qm_dma_channel_config_t::handshake_interface, qm_dma_channel_config_t::handshake_polarity, QM_DMA_CHANNEL_NUM, QM_DMA_MEMORY_TO_MEMORY, QM_DMA_MEMORY_TO_PERIPHERAL, QM_DMA_NUM, QM_DMA_PERIPHERAL_TO_MEMORY, qm_dma_channel_config_t::source_burst_length, and qm_dma_channel_config_t::source_transfer_width.

Referenced by qm_i2c_dma_channel_config(), qm_spi_dma_channel_config(), and qm_uart_dma_channel_config().

4.6.3.2 int qm_dma_init (const qm_dma_t *dma*)

Initialise the DMA controller.

The DMA controller and channels are first disabled. All DMA controller interrupts are masked using the controllers interrupt masking registers. The system DMA interrupts are then unmasked. Finally the DMA controller is enabled. This function must only be called once as it resets the DMA controller and interrupt masking.

Parameters

in	*dma*	DMA instance.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Negative	errno for possible error codes.

Definition at line 143 of file qm_dma.c.

References qm_dma_int_reg_t::clear_block_low, qm_dma_int_reg_t::clear_dst_trans_low, qm_dma_int_reg_t::clear_err_low, qm_dma_int_reg_t::clear_src_trans_low, qm_dma_int_reg_t::clear_tfr_low, qm_dma_int_reg_t::mask_block_low, qm_dma_int_reg_t::mask_dst_trans_low, qm_dma_int_reg_t::mask_err_low, qm_dma_int_reg_t::mask_src_trans_low, qm_dma_int_reg_t::mask_tfr_low, QM_DMA_CHANNEL_NUM, and QM_DMA_NUM.

4.6.3.3 int qm_dma_transfer_mem_to_mem (const qm_dma_t *dma*, const qm_dma_channel_id_t *channel_id*, qm_dma_transfer_t *const *const *transfer_config*)

Setup and start memory to memory transfer.

This function will setup a memory to memory transfer by calling qm_dma_transfer_setup() and will then start the transfer by calling qm_dma_transfer_start(). This is done for consistency across user applications.

Parameters

in	*dma*	DMA instance.
in	*channel_id*	The channel to start.
in	*transfer_config*	The transfer DMA configuration as defined by the dma client. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 346 of file qm_dma.c.

References qm_dma_transfer_t::block_size, qm_dma_transfer_t::destination_address, QM_DMA_CHANNEL_NUM, QM_DMA_NUM, qm_dma_transfer_set_config(), qm_dma_transfer_start(), and qm_dma_transfer_t::source_address.

4.6.3.4 int qm_dma_transfer_set_config (const qm_dma_t *dma*, const qm_dma_channel_id_t *channel_id*, qm_dma_transfer_t *const *const *transfer_config*)

Setup a DMA channel transfer.

Configure the source address,destination addresses and block size. qm_dma_channel_set_config() must first be called before configuring a transfer. qm_dma_transfer_set_config() must be called before starting every transfer, even if the addresses and block size remain unchanged.

Parameters

in	*dma*	DMA instance.
in	*channel_id*	The channel to start.
in	*transfer_config*	The transfer DMA configuration as defined by the dma client. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 267 of file qm_dma.c.

References qm_dma_transfer_t::block_size, qm_dma_transfer_t::destination_address, QM_DMA_CHANNEL_NUM, QM_DMA_NUM, and qm_dma_transfer_t::source_address.

Referenced by qm_dma_transfer_mem_to_mem(), qm_i2c_master_dma_transfer(), qm_spi_dma_transfer(), qm_uart_dma_read(), and qm_uart_dma_write().

4.6.3.5 int qm_dma_transfer_start (const qm_dma_t *dma*, const qm_dma_channel_id_t *channel_id*)

Start a DMA transfer.

qm_dma_transfer_set_config() mustfirst be called before starting a transfer.

Parameters

in	*dma*	DMA instance.
in	*channel_id*	The channel to start.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 293 of file qm_dma.c.

References qm_dma_int_reg_t::mask_err_low, qm_dma_int_reg_t::mask_tfr_low, QM_DMA_CHANNEL_NUM, and QM_DMA_NUM.

Referenced by qm_dma_transfer_mem_to_mem(), qm_i2c_master_dma_transfer(), qm_spi_dma_transfer(), qm_uart_dma_read(), and qm_uart_dma_write().

4.6.3.6 int qm_dma_transfer_terminate (const qm_dma_t *dma*, const qm_dma_channel_id_t *channel_id*)

Terminate a DMA transfer.

This function is only called if a transfer needs to be terminated manually. This may be require if an expected transfer complete callback has not been received. Terminating the transfer will trigger the transfer complete callback. The length returned by the callback is the transfer length at the time that the transfer was terminated.

Parameters

in	*dma*	DMA instance.
in	*channel_id*	The channel to stop.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Negative	errno for possible error codes.

Definition at line 312 of file qm_dma.c.

References qm_dma_int_reg_t::mask_err_low, qm_dma_int_reg_t::mask_tfr_low, QM_DMA_CHANNEL_NUM, and QM_DMA_NUM.

Referenced by qm_i2c_dma_transfer_terminate(), qm_spi_dma_transfer_terminate(), qm_uart_dma_read_terminate(), and qm_uart_dma_write_terminate().

4.7 Flash

Flash controller.

Data Structures

- struct qm_flash_config_t

  Flash configuration structure.

Enumerations

- enum qm_flash_region_t { QM_FLASH_REGION_OTP = 0, QM_FLASH_REGION_SYS, QM_FLASH_REGION_DATA, QM_FLASH_REGION_NUM }

  Flash region enum.
- enum qm_flash_disable_t { QM_FLASH_WRITE_ENABLE, QM_FLASH_WRITE_DISABLE }

  Flash write disable / enable enum.

Functions

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

  Configure a Flash controller.
- int qm_flash_word_write (const qm_flash_t flash, const qm_flash_region_t region, uint32_t f_addr, const uint32_t data)

  Write 4 bytes of data to Flash.
- int qm_flash_page_update (const qm_flash_t flash, const qm_flash_region_t reg, uint32_t f_addr, uint32_t *const page_buf, const uint32_t *const data, uint32_t len)

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

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

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

  Perform mass erase.

4.7.1 Detailed Description

Flash controller.

4.7.2 Enumeration Type Documentation

4.7.2.1 enum qm_flash_disable_t

Flash write disable / enable enum.

Enumerator

QM_FLASH_WRITE_ENABLE Flash write enable.

QM_FLASH_WRITE_DISABLE Flash write disable.

Definition at line 72 of file qm_flash.h.

4.7.2.2 enum qm_flash_region_t

Flash region enum.

Enumerator

QM_FLASH_REGION_OTP Flash OTP region.

QM_FLASH_REGION_SYS Flash System region.

QM_FLASH_REGION_DATA Flash Data region (Quark D2000 only).

QM_FLASH_REGION_NUM Total number of flash regions.

Definition at line 60 of file qm_flash.h.

4.7.3 Function Documentation

4.7.3.1 int qm_flash_mass_erase (const qm_flash_t *flash*, const uint8_t *include_rom*)

Perform mass erase.

Perform mass erase on the specified flash controller. Brownout check is performed before initiating the erase. The mass erase may include the ROM region, if present and unlocked. Note: it is not possible to mass-erase the ROM portion separately.

NOTE: Since this operation may take some time to complete, the caller is responsible for ensuring that the watchdog timer does not elapse in the meantime (e.g., by restarting it before calling this function).

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 287 of file qm_flash.c.

References qm_flash_reg_t::ctrl, and qm_flash_reg_t::flash_stts.

4.7.3.2 int qm_flash_page_erase (const qm_flash_t *flash*, const qm_flash_region_t *region*, uint32_t *page_num*)

Erase one page of Flash.

Brownout check is performed before initiating the write.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 250 of file qm_flash.c.

References qm_flash_reg_t::flash_stts, qm_flash_reg_t::flash_wr_ctrl, QM_FLASH_REGION_DATA, QM_FLASH_REGION_NUM, QM_FLASH_REGION_OTP, QM_FLASH_REGION_SYS, and qm_flash_reg_t::rom_wr_ctrl.

4.7.3.3 int qm_flash_page_update (const qm_flash_t *flash*, const qm_flash_region_t *reg*, uint32_t *f_addr*, uint32_t *const *const *page_buf*, const uint32_t *const *data*, uint32_t *len*)

Write multiple of 4 bytes of data to Flash.

Brownout check is performed before initiating the write. The page is erased, and then written to.

NOTE: Since this operation may take some time to complete, the caller is responsible for ensuring that the watchdog timer does not elapse in the meantime (e.g., by restarting it before calling this function).

Parameters

in	*flash*	Flash controller index.
in	*reg*	Which Flash region to address.
in	*f_addr*	Address within Flash physical address space.
in	*page_buf*	Page buffer to store page during update. Must be at least QM_FLASH_PAGE_SIZE words big and must not be NULL.
in	*data*	Data to write (array of words). This must not be NULL.
in	*len*	Length of data to write (number of words).

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 153 of file qm_flash.c.

References qm_flash_reg_t::flash_stts, qm_flash_reg_t::flash_wr_ctrl, qm_flash_reg_t::flash_wr_data, QM_FLASH_REGION_DATA, QM_FLASH_REGION_NUM, QM_FLASH_REGION_OTP, QM_FLASH_REGION_SYS, qm_flash_reg_t::rom_wr_ctrl, and qm_flash_reg_t::rom_wr_data.

4.7.3.4 int qm_flash_page_write (const qm_flash_t *flash*, const qm_flash_region_t *region*, uint32_t *page_num*, const uint32_t * *data*, uint32_t *len*)

Write a 2KB flash page.

Brownout check is performed before initiating the write. The page is erased, and then written to.

NOTE: Since this operation may take some time to complete, the caller is responsible for ensuring that the watchdog timer does not elapse in the meantime (e.g., by restarting it before calling this function).

Parameters

in	*flash*	Flash controller index.
in	*region*	Which Flash region to address.
in	*page_num*	Which page of flash to overwrite.

in	*data*	Data to write (array of words). This must not be NULL.
in	*len*	Length of data to write (number of words).

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 91 of file qm_flash.c.

References qm_flash_reg_t::flash_stts, qm_flash_reg_t::flash_wr_ctrl, qm_flash_reg_t::flash_wr_data, QM_FLASH_REGION_DATA, QM_FLASH_REGION_NUM, QM_FLASH_REGION_OTP, QM_FLASH_REGION_SYS, qm_flash_reg_t::rom_wr_ctrl, and qm_flash_reg_t::rom_wr_data.

4.7.3.5 int qm_flash_set_config (const qm_flash_t *flash*, const qm_flash_config_t *const *cfg)

Configure a Flash controller.

The configuration includes timing and behavioral settings.

Note: when switching SoC to a higher frequency, flash controllers must be reconfigured to reflect settings associated with higher frequency BEFORE SoC frequency is changed. On the other hand, when switching SoC to a lower frequency, flash controller must be reconfigured only 6 NOP instructions AFTER the SoC frequency has been updated. Otherwise, flash timings will be violated.

Parameters

in	*flash*	Flash controller index.
in	*cfg*	Flash configuration. It must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 16 of file qm_flash.c.

References qm_flash_reg_t::ctrl, QM_FLASH_WRITE_DISABLE, qm_flash_reg_t::tmg_ctrl, qm_flash_config_t::us_count, qm_flash_config_t::wait_states, and qm_flash_config_t::write_disable.

4.7.3.6 int qm_flash_word_write (const qm_flash_t *flash*, const qm_flash_region_t *region*, uint32_t *f_addr*, const uint32_t *data*)

Write 4 bytes of data to Flash.

Brownout check is performed before initiating the write.

Note: this function performs a write operation only; page erase may be needed if the page is already programmed.

Parameters

in	*flash*	Flash controller index.

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 39 of file qm_flash.c.

References qm_flash_reg_t::flash_stts, qm_flash_reg_t::flash_wr_ctrl, qm_flash_reg_t::flash_wr_data, QM_FLASH_REGION_DATA, QM_FLASH_REGION_NUM, QM_FLASH_REGION_OTP, QM_FLASH_REGION_SYS, qm_flash_reg_t::rom_wr_ctrl, and qm_flash_reg_t::rom_wr_data.

4.8 FPR

Flash Protection Region control.

Data Structures

- struct qm_fpr_config_t

  Flash Protection Region configuration structure.

Enumerations

- enum qm_fpr_id_t { QM_FPR_0, QM_FPR_1, QM_FPR_2, QM_FPR_3 }

  FPR register map.
- enum qm_fpr_en_t { QM_FPR_DISABLE, QM_FPR_ENABLE, QM_FPR_LOCK_DISABLE, QM_FPR_LOCK_ENABLE }

  FPR enable type.
- enum qm_fpr_viol_mode_t { FPR_VIOL_MODE_INTERRUPT = 0, FPR_VIOL_MODE_RESET, FPR_VIOL_MODE_PROBE }

  FPR vilation mode type.
- enum qm_flash_region_type_t { QM_MAIN_FLASH_SYSTEM = 0, QM_MAIN_FLASH_DATA, QM_MAIN_FLASH_NUM }

  FPR region type.
- enum qm_fpr_read_allow_t { QM_FPR_HOST_PROCESSOR, QM_FPR_SENSOR_SUBSYSTEM, QM_FPR_DMA = BIT(2), QM_FPR_OTHER_AGENTS }

  FPR read allow type.

Functions

- int 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.
- int qm_fpr_set_violation_policy (const qm_fpr_viol_mode_t mode, const qm_flash_t flash, qm_fpr_callback_t fpr_cb, void *data)

  Configure FPR violation behaviour.

4.8.1 Detailed Description

Flash Protection Region control.

4.8.2 Enumeration Type Documentation

4.8.2.1 enum qm_flash_region_type_t

FPR region type.

Enumerator

QM_MAIN_FLASH_SYSTEM System flash region.

QM_MAIN_FLASH_DATA Data flash region.

QM_MAIN_FLASH_NUM Number of flash regions.

Definition at line 52 of file qm_fpr.h.

4.8.2.2 enum qm_fpr_en_t

FPR enable type.

Enumerator

QM_FPR_DISABLE Disable FPR.

QM_FPR_ENABLE Enable FPR.

QM_FPR_LOCK_DISABLE Disable FPR lock.

QM_FPR_LOCK_ENABLE Enable FPR lock.

Definition at line 33 of file qm_fpr.h.

4.8.2.3 enum qm_fpr_id_t

FPR register map.

Enumerator

QM_FPR_0 FPR 0.

QM_FPR_1 FPR 1.

QM_FPR_2 FPR 2.

QM_FPR_3 FPR 3.

Definition at line 22 of file qm_fpr.h.

4.8.2.4 enum qm_fpr_read_allow_t

FPR read allow type.

Enumerator

QM_FPR_HOST_PROCESSOR Allow host processor to access flash region.

QM_FPR_SENSOR_SUBSYSTEM Allow sensor subsystem to access flash region.

QM_FPR_DMA Allow DMA to access flash region.

QM_FPR_OTHER_AGENTS Allow other agents to access flash region.

Definition at line 63 of file qm_fpr.h.

4.8.2.5 enum qm_fpr_viol_mode_t

FPR vilation mode type.

Enumerator

FPR_VIOL_MODE_INTERRUPT Generate interrupt on violation.

FPR_VIOL_MODE_RESET Reset SoC on violation.

FPR_VIOL_MODE_PROBE Enter probe mode on violation.

Definition at line 43 of file qm_fpr.h.

4.8.3 Function Documentation

4.8.3.1 int qm_fpr_set_config (const qm_flash_t *flash*, const qm_fpr_id_t *id*, const qm_fpr_config_t *const *cfg*, const qm_flash_region_type_t *region*)

Configure a Flash controller's Flash Protection Region.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 29 of file qm_fpr.c.

References qm_fpr_config_t::allow_agents, qm_fpr_config_t::en_mask, qm_flash_reg_t::fpr_rd_cfg, qm_fpr_config_t::low_bound, QM_MAIN_FLASH_DATA, QM_MAIN_FLASH_NUM, QM_MAIN_FLASH_SYSTEM, and qm_fpr_config_t::up_bound.

4.8.3.2 int qm_fpr_set_violation_policy (const qm_fpr_viol_mode_t *mode*, const qm_flash_t *flash*, qm_fpr_callback_t *fpr_cb*, void * *data*)

Configure FPR violation behaviour.

Parameters

in	*mode*	(generate interrupt, warm reset, enter probe mode).
in	*flash*	controller.
in	*fpr_cb*	for interrupt mode (only).
in	*data*	user callback data for interrupt mode (only).

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 116 of file qm_fpr.c.

References FPR_VIOL_MODE_INTERRUPT, FPR_VIOL_MODE_PROBE, qm_irq_mask(), and qm_irq_unmask().

4.9 GPIO

General Purpose IO.

Data Structures

- struct qm_gpio_port_config_t
  GPIO port configuration type.

Enumerations

- enum qm_gpio_state_t { QM_GPIO_LOW, QM_GPIO_HIGH, QM_GPIO_STATE_NUM }
  GPIO pin states.

Functions

- int qm_gpio_set_config (const qm_gpio_t gpio, const qm_gpio_port_config_t *const cfg)
  Set GPIO port configuration.
- int qm_gpio_read_pin (const qm_gpio_t gpio, const uint8_t pin, qm_gpio_state_t *const state)
  Read the current state of a single pin on a given GPIO port.
- int qm_gpio_set_pin (const qm_gpio_t gpio, const uint8_t pin)
  Set a single pin on a given GPIO port.
- int qm_gpio_clear_pin (const qm_gpio_t gpio, const uint8_t pin)
  Clear a single pin on a given GPIO port.
- int qm_gpio_set_pin_state (const qm_gpio_t gpio, const uint8_t pin, const qm_gpio_state_t state)
  Set or clear a single GPIO pin using a state variable.
- int qm_gpio_read_port (const qm_gpio_t gpio, uint32_t *const port)
  Read the value of every pin on a GPIO port.
- int qm_gpio_write_port (const qm_gpio_t gpio, const uint32_t val)
  Write a value to every pin on a GPIO port.

4.9.1 Detailed Description

General Purpose IO.

4.9.2 Enumeration Type Documentation

4.9.2.1 enum qm_gpio_state_t

GPIO pin states.

Enumerator

QM_GPIO_LOW GPIO low state.
QM_GPIO_HIGH GPIO high state.
QM_GPIO_STATE_NUM Number of GPIO states.

Definition at line 21 of file qm_gpio.h.

4.9.3 Function Documentation

4.9.3.1 int qm_gpio_clear_pin (const qm_gpio_t *gpio*, const uint8_t *pin*)

Clear a single pin on a given GPIO port.

Parameters

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

Returns

int 0 on success, error code otherwise.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 114 of file qm_gpio.c.

4.9.3.2 int qm_gpio_read_pin (const qm_gpio_t *gpio*, const uint8_t *pin*, qm_gpio_state_t *const *state*)

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

Parameters

in	*gpio*	GPIO port index.
in	*pin*	Pin of GPIO port to read.
out	*state*	Current state of the pin. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 92 of file qm_gpio.c.

4.9.3.3 int qm_gpio_read_port (const qm_gpio_t *gpio*, uint32_t *const *port*)

Read the value of every pin on a GPIO port.

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

Parameters

in	*gpio*	GPIO port index.
out	*port*	State of every pin in a GPIO port. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 138 of file qm_gpio.c.

4.9.3.4 int qm_gpio_set_config (const qm_gpio_t *gpio*, const qm_gpio_port_config_t *const *cfg*)

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 the user defined callback function.

Parameters

in	*gpio*	GPIO port index to configure.
in	*cfg*	New configuration for GPIO port. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 67 of file qm_gpio.c.

References qm_gpio_port_config_t::callback, qm_gpio_port_config_t::callback_data, qm_gpio_port_config_t::direction, qm_gpio_reg_t::gpio_debounce, qm_gpio_reg_t::gpio_int_bothedge, qm_gpio_reg_t::gpio_int_polarity, qm_gpio_reg_t::gpio_inten, qm_gpio_reg_t::gpio_intmask, qm_gpio_reg_t::gpio_inttype_level, qm_gpio_reg_t::gpio_swporta_ddr, qm_gpio_port_config_t::int_bothedge, qm_gpio_port_config_t::int_debounce, qm_gpio_port_config_t::int_en, qm_gpio_port_config_t::int_polarity, and qm_gpio_port_config_t::int_type.

4.9.3.5 int qm_gpio_set_pin (const qm_gpio_t *gpio*, const uint8_t *pin*)

Set a single pin on a given GPIO port.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 104 of file qm_gpio.c.

4.9.3.6 int qm_gpio_set_pin_state (const qm_gpio_t *gpio*, const uint8_t *pin*, const qm_gpio_state_t *state*)

Set or clear a single GPIO pin using a state variable.

Parameters

in	*gpio*	GPIO port index.
in	*pin*	Pin of GPIO port to update.
in	*state*	QM_GPIO_LOW for low or QM_GPIO_HIGH for high.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 124 of file qm_gpio.c.

References QM_GPIO_STATE_NUM.

4.9.3.7 int qm_gpio_write_port (const qm_gpio_t *gpio*, const uint32_t *val*)

Write a value to every pin on a GPIO port.

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 148 of file qm_gpio.c.

4.10 I2C

I2C.

Data Structures

- struct qm_i2c_config_t

  I2C configuration type.
- struct qm_i2c_transfer_t

  I2C transfer type.

Enumerations

- enum qm_i2c_addr_t { QM_I2C_7_BIT = 0, QM_I2C_10_BIT }

  QM I2C addressing type.
- enum qm_i2c_mode_t { QM_I2C_MASTER, QM_I2C_SLAVE }

  QM I2C master / slave mode type.
- enum qm_i2c_speed_t { QM_I2C_SPEED_STD = 1, QM_I2C_SPEED_FAST = 2, QM_I2C_SPEED_FAST_PLUS = 3 }

  QM I2C Speed Type.
- enum qm_i2c_status_t {
  QM_I2C_IDLE = 0, QM_I2C_TX_ABRT_7B_ADDR_NOACK = BIT(0), QM_I2C_TX_ABRT_10ADDR1_NOACK = BIT(1), QM_I2C_TX_ABRT_10ADDR2_NOACK = BIT(2),
  QM_I2C_TX_ABRT_TXDATA_NOACK = BIT(3), QM_I2C_TX_ABRT_GCALL_NOACK = BIT(4), QM_I2C_TX_ABRT_GCALL_READ = BIT(5), QM_I2C_TX_ABRT_HS_ACKDET = BIT(6),
  QM_I2C_TX_ABRT_SBYTE_ACKDET = BIT(7), QM_I2C_TX_ABRT_HS_NORSTRT = BIT(8), QM_I2C_TX_ABRT_10B_RD_NORSTRT = BIT(10), QM_I2C_TX_ABRT_MASTER_DIS = BIT(11),
  QM_I2C_TX_ARB_LOST = BIT(12), QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO = BIT(13), QM_I2C_TX_ABRT_SLV_ARBLOST = BIT(14), QM_I2C_TX_ABRT_SLVRD_INTX = BIT(15),
  QM_I2C_TX_ABRT_USER_ABRT = BIT(16), QM_I2C_BUSY = BIT(17) }

  I2C status type.

Functions

- int qm_i2c_set_config (const qm_i2c_t i2c, const qm_i2c_config_t *const cfg)

  Set I2C configuration.
- int qm_i2c_set_speed (const qm_i2c_t i2c, const qm_i2c_speed_t speed, const uint16_t lo_cnt, const uint16_t hi_cnt)

  Set I2C speed.
- int qm_i2c_get_status (const qm_i2c_t i2c, qm_i2c_status_t *const status)

  Retrieve I2C status.
- int qm_i2c_master_write (const qm_i2c_t i2c, const uint16_t slave_addr, const uint8_t *const data, uint32_t len, const bool stop, qm_i2c_status_t *const status)

  Master write on I2C.
- int qm_i2c_master_read (const qm_i2c_t i2c, const uint16_t slave_addr, uint8_t *const data, uint32_t len, const bool stop, qm_i2c_status_t *const status)

  Master read of I2C.
- int 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.
- int qm_i2c_irq_transfer_terminate (const qm_i2c_t i2c)

  Terminate I2C IRQ transfer.

- int qm_i2c_dma_channel_config (const qm_i2c_t i2c, const qm_dma_t dma_controller_id, const qm_dma_channel_id_t channel_id, const qm_dma_channel_direction_t direction)

  Configure a DMA channel with a specific transfer direction.
- int qm_i2c_master_dma_transfer (const qm_i2c_t i2c, qm_i2c_transfer_t *const xfer, const uint16_t slave_addr)

  Perform a DMA-based transfer on the I2C bus.
- int qm_i2c_dma_transfer_terminate (const qm_i2c_t i2c)

  Terminate any DMA transfer going on on the controller.

4.10.1 Detailed Description

I2C.

4.10.2 Enumeration Type Documentation

4.10.2.1 enum qm_i2c_addr_t

QM I2C addressing type.

Enumerator

QM_I2C_7_BIT 7-bit mode.

QM_I2C_10_BIT 10-bit mode.

Definition at line 36 of file qm_i2c.h.

4.10.2.2 enum qm_i2c_mode_t

QM I2C master / slave mode type.

Enumerator

QM_I2C_MASTER Master mode.

QM_I2C_SLAVE Slave mode.

Definition at line 44 of file qm_i2c.h.

4.10.2.3 enum qm_i2c_speed_t

QM I2C Speed Type.

Enumerator

QM_I2C_SPEED_STD Standard mode (100 Kbps).

QM_I2C_SPEED_FAST Fast mode (400 Kbps).

QM_I2C_SPEED_FAST_PLUS Fast plus mode (1 Mbps).

Definition at line 52 of file qm_i2c.h.

4.10.2.4 enum qm_i2c_status_t

I2C status type.

Enumerator

QM_I2C_IDLE Controller idle.

QM_I2C_TX_ABRT_7B_ADDR_NOACK 7-bit address noack.

QM_I2C_TX_ABRT_10ADDR1_NOACK 10-bit address noack.

QM_I2C_TX_ABRT_10ADDR2_NOACK 10-bit second address byte address noack.

QM_I2C_TX_ABRT_TXDATA_NOACK Tx data noack.

QM_I2C_TX_ABRT_GCALL_NOACK General call noack.

QM_I2C_TX_ABRT_GCALL_READ Read after general call.

QM_I2C_TX_ABRT_HS_ACKDET High Speed master ID ACK.

QM_I2C_TX_ABRT_SBYTE_ACKDET Start ACK.

QM_I2C_TX_ABRT_HS_NORSTRT High Speed with restart disabled.

QM_I2C_TX_ABRT_10B_RD_NORSTRT 10-bit address read and restart disabled.

QM_I2C_TX_ABRT_MASTER_DIS Master disabled.

QM_I2C_TX_ARB_LOST Master lost arbitration.

QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO Slave flush tx FIFO.

QM_I2C_TX_ABRT_SLV_ARBLOST Slave lost bus.

QM_I2C_TX_ABRT_SLVRD_INTX Slave read completion.

QM_I2C_TX_ABRT_USER_ABRT User abort.

QM_I2C_BUSY Controller busy.

Definition at line 61 of file qm_i2c.h.

4.10.3 Function Documentation

4.10.3.1 int qm_i2c_dma_channel_config (const qm_i2c_t *i2c*, const qm_dma_t *dma_controller_id*, const qm_dma_channel_id_t *channel_id*, const qm_dma_channel_direction_t *direction*)

Configure a DMA channel with a specific transfer direction.

Configure a DMA channel with a specific transfer direction. The user is responsible for managing the allocation of the pool of DMA channels provided by each DMA core to the different peripheral drivers that require them. Note that a I2C controller cannot use different DMA cores to manage transfers in different directions.

This function configures DMA channel parameters that are unlikely to change between transfers, like transaction width, burst size, and handshake interface parameters. The user will likely only call this function once for the lifetime of an application unless the channel needs to be repurposed.

Note that qm_dma_init() must first be called before configuring a channel.

Parameters

in	*i2c*	I2C controller identifier.
in	*dma_controller_id*	DMA controller identifier.
in	*channel_id*	DMA channel identifier.
in	*direction*	DMA channel direction, either QM_DMA_MEMORY_TO_PERIPHERAL (TX transfer) or QM_DMA_PERIPHERAL_TO_MEMORY (RX transfer).

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Configure a DMA channel with a specific transfer direction.

Definition at line 952 of file qm_i2c.c.

References qm_dma_channel_config_t::callback_context, qm_dma_channel_config_t::channel_direction, qm_dma_channel_config_t::client_callback, qm_dma_channel_config_t::destination_burst_length, qm_dma_channel_config_t::destination_transfer_width, qm_dma_channel_config_t::handshake_interface, qm_dma_channel_config_t::handshake_polarity, QM_DMA_BURST_TRANS_LENGTH_1, QM_DMA_CHANNEL_NUM, qm_dma_channel_set_config(), QM_DMA_HANDSHAKE_POLARITY_HIGH, QM_DMA_MEMORY_TO_PERIPHERAL, QM_DMA_NUM, QM_DMA_PERIPHERAL_TO_MEMORY, QM_DMA_TRANS_WIDTH_8, qm_dma_channel_config_t::source_burst_length, and qm_dma_channel_config_t::source_transfer_width.

4.10.3.2 int qm_i2c_dma_transfer_terminate (const qm_i2c_t *i2c*)

Terminate any DMA transfer going on on the controller.

Calls the DMA driver to stop any ongoing DMA transfer and calls qm_i2c_irq_transfer_terminate.

Parameters

in	*i2c*	Which I2C to terminate transfers from.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Terminate any DMA transfer going on on the controller.

Definition at line 718 of file qm_i2c.c.

References qm_dma_transfer_terminate().

4.10.3.3 int qm_i2c_get_status (const qm_i2c_t *i2c*, qm_i2c_status_t *const *status*)

Retrieve I2C status.

Parameters

in	*i2c*	Which I2C to read the status of.
out	*status*	Get i2c status. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 453 of file qm_i2c.c.

References qm_i2c_reg_t::ic_status, qm_i2c_reg_t::ic_tx_abrt_source, and QM_I2C_BUSY.

Referenced by qm_i2c_master_read(), and qm_i2c_master_write().

4.10.3.4 int qm_i2c_irq_transfer_terminate (const qm_i2c_t *i2c*)

Terminate I2C IRQ transfer.

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

Parameters

in	*i2c*	I2C register block pointer.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 697 of file qm_i2c.c.

4.10.3.5 int qm_i2c_master_dma_transfer (const qm_i2c_t *i2c*, qm_i2c_transfer_t *const *xfer*, const uint16_t *slave_addr*)

Perform a DMA-based transfer on the I2C bus.

Perform a DMA-based transfer on the I2C bus. If the transfer is TX only, it will enable DMA operation for the controller and start the transfer.

If it's an RX only transfer, it will require 2 channels, one for writting the READ commands and another one for reading the bytes from the bus. Both DMA operations will start in parallel.

If this is a combined transaction, both TX and RX operations will be set up, but only TX will be started. On TX finish (callback), the TX channel will be used for writing the READ commands and the RX operation will start.

Note that qm_i2c_dma_channel_config() must first be called in order to configure all DMA channels needed for a transfer.

Parameters

in	*i2c*	I2C controller identifier.
in	*xfer*	Structure containing pre-allocated write and read data buffers and callback functions. This pointer must be kept valid until the transfer is complete.
in	*slave_addr*	Slave address.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Perform a DMA-based transfer on the I2C bus.

In case of combined transaction, it sets up both operations and starts the write one; the read operation will be started in the read operation callback.

Definition at line 1002 of file qm_i2c.c.

References qm_dma_transfer_set_config(), qm_dma_transfer_start(), qm_i2c_transfer_t::rx, qm_i2c_transfer_t::rx_len, qm_i2c_transfer_t::stop, qm_i2c_transfer_t::tx, and qm_i2c_transfer_t::tx_len.

4.10.3.6 int qm_i2c_master_irq_transfer (const qm_i2c_t *i2c*, const qm_i2c_transfer_t *const *xfer*, const uint16_t *slave_addr*)

Interrupt based master transfer on I2C.

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

Parameters

in	*i2c*	Which I2C to transfer from.
in	*xfer*	Transfer structure includes write / read buffers, length, user callback function and the callback context. The structure must not be NULL and must be kept valid until the transfer is complete.
in	*slave_addr*	Address of slave to transfer data with.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 617 of file qm_i2c.c.

References qm_i2c_reg_t::ic_intr_mask, qm_i2c_reg_t::ic_rx_tl, qm_i2c_reg_t::ic_tar, qm_i2c_reg_t::ic_tx_tl, and qm_i2c_transfer_t::rx_len.

4.10.3.7 int qm_i2c_master_read (const qm_i2c_t *i2c*, const uint16_t *slave_addr*, uint8_t *const *data*, uint32_t *len*, const bool *stop*, qm_i2c_status_t *const *status*)

Master read of I2C.

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

Parameters

in	*i2c*	Which I2C to read from.
in	*slave_addr*	Address of slave device to read from.
out	*data*	Pre-allocated buffer to populate with data. This must not be NULL.
in	*len*	length of data to read from slave.
in	*stop*	Generate a STOP condition at the end of tx.
out	*status*	Get i2c status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 544 of file qm_i2c.c.

References qm_i2c_reg_t::ic_clr_tx_abrt, qm_i2c_reg_t::ic_data_cmd, qm_i2c_reg_t::ic_raw_intr_stat, qm_i2c_reg_t::ic_status, qm_i2c_reg_t::ic_tar, qm_i2c_reg_t::ic_tx_abrt_source, and qm_i2c_get_status().

4.10.3.8 int qm_i2c_master_write (const qm_i2c_t *i2c*, const uint16_t *slave_addr*, const uint8_t *const *data*, uint32_t *len*, const bool *stop*, qm_i2c_status_t *const *status*)

Master write on I2C.

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

Parameters

in	*i2c*	Which I2C to write to.
in	*slave_addr*	Address of slave to write to.
in	*data*	Pre-allocated buffer of data to write. This must not be NULL.
in	*len*	length of data to write.
in	*stop*	Generate a STOP condition at the end of tx.
out	*status*	Get i2c status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 474 of file qm_i2c.c.

References qm_i2c_reg_t::ic_clr_tx_abrt, qm_i2c_reg_t::ic_data_cmd, qm_i2c_reg_t::ic_status, qm_i2c_reg_t::ic_tar, qm_i2c_reg_t::ic_tx_abrt_source, and qm_i2c_get_status().

4.10.3.9 int qm_i2c_set_config (const qm_i2c_t *i2c*, const qm_i2c_config_t *const *cfg*)

Set I2C configuration.

Parameters

in	*i2c*	Which I2C to set the configuration of.
out	*cfg*	I2C configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 308 of file qm_i2c.c.

References qm_i2c_config_t::address_mode, qm_i2c_reg_t::ic_con, qm_i2c_reg_t::ic_fs_scl_hcnt, qm_i2c_reg_t::ic_fs_scl_lcnt, qm_i2c_reg_t::ic_fs_spklen, qm_i2c_reg_t::ic_intr_mask, qm_i2c_reg_t::ic_sar, qm_i2c_reg_t::ic_ss_scl_hcnt, qm_i2c_reg_t::ic_ss_scl_lcnt, qm_i2c_config_t::mode, QM_I2C_MASTER, QM_I2C_SLAVE, QM_I2C_SPEED_FAST, QM_I2C_SPEED_FAST_PLUS, QM_I2C_SPEED_STD, qm_i2c_config_t::slave_addr, and qm_i2c_config_t::speed.

4.10.3.10 int qm_i2c_set_speed (const qm_i2c_t *i2c*, const qm_i2c_speed_t *speed*, const uint16_t *lo_cnt*, const uint16_t *hi_cnt*)

Set I2C speed.

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 417 of file qm_i2c.c.

References qm_i2c_reg_t::ic_con, qm_i2c_reg_t::ic_fs_scl_hcnt, qm_i2c_reg_t::ic_fs_scl_lcnt, qm_i2c_reg_t::ic_fs_spklen, qm_i2c_reg_t::ic_ss_scl_hcnt, qm_i2c_reg_t::ic_ss_scl_lcnt, QM_I2C_SPEED_FAST, QM_I2C_SPEED_FAST_PLUS, and QM_I2C_SPEED_STD.

4.11 Identification

Identification functions for Quark Microcontrollers.

Functions

- uint32_t qm_soc_id (void)

  Get Quark SoC identification number.
- uint32_t qm_soc_version (void)

  Get Quark SoC version number.

4.11.1 Detailed Description

Identification functions for Quark Microcontrollers.

4.11.2 Function Documentation

4.11.2.1 uint32_t qm_soc_id (void)

Get Quark SoC identification number.

Returns

uint32_t SoC identifier number.

Definition at line 10 of file qm_identification.c.

4.11.2.2 uint32_t qm_soc_version (void)

Get Quark SoC version number.

Returns

uint32_t SoC version number.

Definition at line 21 of file qm_identification.c.

4.12 Initialisation

Initialisation and reset.

Enumerations

- enum qm_soc_reset_t { QM_WARM_RESET = BIT(1), QM_COLD_RESET = BIT(3) }
  Reset Mode type.

Functions

- void qm_soc_reset (qm_soc_reset_t reset_type)
  Reset the SoC.

4.12.1 Detailed Description

Initialisation and reset.

4.12.2 Enumeration Type Documentation

4.12.2.1 enum qm_soc_reset_t

Reset Mode type.

Enumerator

QM_WARM_RESET Warm reset.

QM_COLD_RESET Cold reset.

Definition at line 21 of file qm_init.h.

4.12.3 Function Documentation

4.12.3.1 void qm_soc_reset (qm_soc_reset_t *reset_type*)

Reset the SoC.

This can either be a cold reset or a warm reset.

Parameters

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

Definition at line 7 of file qm_init.c.

4.13 Interrupt

Interrupt driver.

Typedefs

- typedef void(∗ qm_isr_t)(struct interrupt_frame ∗frame)

  Interrupt service routine type.

Functions

- void qm_irq_enable (void)

  Enable interrupt delivery for the SoC.
- void qm_irq_disable (void)

  Disable interrupt delivery for the SoC.
- void qm_irq_unmask (uint32_t irq)

  Unmask a given interrupt line.
- void qm_irq_mask (uint32_t irq)

  Mask a given interrupt line.
- void qm_int_vector_request (uint32_t vector, qm_isr_t isr)

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

4.13.1 Detailed Description

Interrupt driver.

4.13.2 Function Documentation

4.13.2.1 static __inline__ void qm_int_vector_request (uint32_t *vector,* qm_isr_t *isr*)

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

Parameters

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

Definition at line 98 of file qm_interrupt.h.

4.13.2.2 void qm_irq_mask (uint32_t *irq*)

Mask a given interrupt line.

Parameters

in	*irq*	Which IRQ to mask.

Definition at line 52 of file qm_interrupt.c.

References qm_ss_irq_mask().

Referenced by qm_fpr_set_violation_policy(), qm_mbox_ch_set_config(), and qm_mpr_set_violation_policy().

4.13.2.3 void qm_irq_unmask (uint32_t *irq*)

Unmask a given interrupt line.

Parameters

in	*irq*	Which IRQ to unmask.

Definition at line 66 of file qm_interrupt.c.

References qm_ss_irq_unmask().

Referenced by qm_fpr_set_violation_policy(), qm_mbox_ch_set_config(), and qm_mpr_set_violation_policy().

4.14 ISR

Interrupt Service Routines.

Functions

- QM_ISR_DECLARE (qm_adc_0_isr)

  ISR for ADC 0 convert and calibration interrupt.
- QM_ISR_DECLARE (qm_adc_pwr_0_isr)

  ISR for ADC 0 change mode interrupt.
- QM_ISR_DECLARE (qm_aonpt_isr_0)

  ISR for Always-on Periodic Timer 0 interrupt.
- QM_ISR_DECLARE (qm_ac_isr)

  ISR for Analog Comparator 0 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_err)

  ISR for DMA error interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_0)

  ISR for DMA channel 0 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_1)

  ISR for DMA channel 1 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_2)

  ISR for DMA channel 2 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_3)

  ISR for DMA channel 3 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_4)

  ISR for DMA channel 4 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_5)

  ISR for DMA channel 5 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_6)

  ISR for DMA channel 6 interrupt.
- QM_ISR_DECLARE (qm_dma_0_isr_7)

  ISR for DMA 0 channel 7 interrupt.
- QM_ISR_DECLARE (qm_fpr_isr_0)

  ISR for FPR 0 interrupt.
- QM_ISR_DECLARE (qm_fpr_isr_1)

  ISR for FPR 1 interrupt.
- QM_ISR_DECLARE (qm_gpio_isr_0)

  ISR for GPIO 0 interrupt.
- QM_ISR_DECLARE (qm_aon_gpio_isr_0)

  ISR for AON GPIO 0 interrupt.
- QM_ISR_DECLARE (qm_i2c_0_isr)

  ISR for I2C 0 interrupt.
- QM_ISR_DECLARE (qm_i2c_1_isr)

  ISR for I2C 1 interrupt.
- QM_ISR_DECLARE (qm_mbox_isr)

  ISR for Mailbox interrupt.
- QM_ISR_DECLARE (qm_mpr_isr)

  ISR for Memory Protection Region interrupt.
- QM_ISR_DECLARE (qm_pic_timer_isr)

  ISR for PIC Timer interrupt.
- QM_ISR_DECLARE (qm_pwm_isr_0)

ISR for PWM 0 interrupt.

- QM_ISR_DECLARE (qm_rtc_isr_0)

  ISR for RTC 0 interrupt.

- QM_ISR_DECLARE (qm_spi_master_0_isr)

  ISR for SPI Master 0 interrupt.

- QM_ISR_DECLARE (qm_spi_master_1_isr)

  ISR for SPI Master 1 interrupt.

- QM_ISR_DECLARE (qm_uart_0_isr)

  ISR for UART 0 interrupt.

- QM_ISR_DECLARE (qm_uart_1_isr)

  ISR for UART 1 interrupt.

- QM_ISR_DECLARE (qm_wdt_isr_0)

  ISR for WDT 0 interrupt.

4.14.1 Detailed Description

Interrupt Service Routines.

4.14.2 Function Documentation

4.14.2.1 QM_ISR_DECLARE (qm_adc_0_isr)

ISR for ADC 0 convert and calibration interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_ADC_0, qm_adc_0_isr);
```

if IRQ based transfers are used.

Definition at line 168 of file qm_adc.c.

4.14.2.2 QM_ISR_DECLARE (qm_adc_pwr_0_isr)

ISR for ADC 0 change mode interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_ADC_PWR_0, qm_adc_pwr_0_isr);
```

if IRQ based transfers are used.

Definition at line 176 of file qm_adc.c.

4.14.2.3 QM_ISR_DECLARE (qm_aonpt_isr_0)

ISR for Always-on Periodic Timer 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_AONPT_0, qm_aonpt_isr_0);
```

if IRQ based transfers are used.

Definition at line 35 of file qm_aon_counters.c.

4.14.2.4 QM_ISR_DECLARE (qm_ac_isr)

ISR for Analog Comparator 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_AC, qm_ac_isr);
```

if IRQ based transfers are used.

Definition at line 10 of file qm_comparator.c.

4.14.2.5 QM_ISR_DECLARE (qm_dma_0_isr_err)

ISR for DMA error interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_ERR, qm_dma_0_isr_err);
```

if IRQ based transfers are used.

Definition at line 87 of file qm_dma.c.

References QM_DMA_0.

4.14.2.6 QM_ISR_DECLARE (qm_dma_0_isr_0)

ISR for DMA channel 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_0, qm_dma_0_isr_0);
```

if IRQ based transfers are used.

Definition at line 93 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_0.

4.14.2.7 QM_ISR_DECLARE (qm_dma_0_isr_1)

ISR for DMA channel 1 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_1, qm_dma_0_isr_1);
```

if IRQ based transfers are used.

Definition at line 99 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_1.

4.14.2.8 QM_ISR_DECLARE (qm_dma_0_isr_2)

ISR for DMA channel 2 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_2, qm_dma_0_isr_2);
```

if IRQ based transfers are used.

Definition at line 106 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_2.

4.14.2.9 QM_ISR_DECLARE (qm_dma_0_isr_3)

ISR for DMA channel 3 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_3, qm_dma_0_isr_3);
```

if IRQ based transfers are used.

Definition at line 112 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_3.

4.14.2.10 QM_ISR_DECLARE (qm_dma_0_isr_4)

ISR for DMA channel 4 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_4, qm_dma_0_isr_4);
```

if IRQ based transfers are used.

Definition at line 118 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_4.

4.14.2.11 QM_ISR_DECLARE (qm_dma_0_isr_5)

ISR for DMA channel 5 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_5, qm_dma_0_isr_5);
```

if IRQ based transfers are used.

Definition at line 124 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_5.

4.14.2.12 QM_ISR_DECLARE (qm_dma_0_isr_6)

ISR for DMA channel 6 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_6, qm_dma_0_isr_6);
```

if IRQ based transfers are used.

Definition at line 130 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_6.

4.14.2.13 QM_ISR_DECLARE (qm_dma_0_isr_7)

ISR for DMA 0 channel 7 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_DMA_7, qm_dma_0_isr_7);
```

if IRQ based transfers are used.

Definition at line 136 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_7.

4.14.2.14 QM_ISR_DECLARE (qm_fpr_isr_0)

ISR for FPR 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_FLASH_0, qm_fpr_isr_0);
```

if IRQ based transfers are used.

Definition at line 11 of file qm_fpr.c.

4.14.2.15 QM_ISR_DECLARE (qm_fpr_isr_1)

ISR for FPR 1 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_FLASH_1, qm_fpr_isr_1);
```

if IRQ based transfers are used.

Definition at line 20 of file qm_fpr.c.

4.14.2.16 QM_ISR_DECLARE (qm_gpio_isr_0)

ISR for GPIO 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_GPIO_0, qm_gpio_isr_0);
```

if IRQ based transfers are used.

Definition at line 53 of file qm_gpio.c.

4.14.2.17 QM_ISR_DECLARE (qm_aon_gpio_isr_0)

ISR for AON GPIO 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_AONGPIO_0, qm_aon_gpio_isr_0);
```

if IRQ based transfers are used.

Definition at line 60 of file qm_gpio.c.

4.14.2.18 QM_ISR_DECLARE (qm_i2c_0_isr)

ISR for I2C 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_I2C_0, qm_i2c_0_isr);
```

if IRQ based transfers are used.

Definition at line 265 of file qm_i2c.c.

4.14.2.19 QM_ISR_DECLARE (qm_i2c_1_isr)

ISR for I2C 1 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_I2C_1, qm_i2c_1_isr);
```

if IRQ based transfers are used.

Definition at line 272 of file qm_i2c.c.

4.14.2.20 QM_ISR_DECLARE (qm_mbox_isr)

ISR for Mailbox interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_MBOX, qm_mbox_isr);
```

if IRQ based transfers are used.

Definition at line 30 of file qm_mailbox.c.

References qm_mailbox_t::ch_sts, and QM_MBOX_CH_INT.

4.14.2.21 QM_ISR_DECLARE (qm_mpr_isr)

ISR for Memory Protection Region interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_SRAM, qm_mpr_isr);
```

if IRQ based transfers are used.

Definition at line 13 of file qm_mpr.c.

4.14.2.22 QM_ISR_DECLARE (qm_pic_timer_isr)

ISR for PIC Timer interrupt.

On Quark Microcontroller D2000 Development Platform, this function needs to be registered with:

```
qm_int_vector_request(QM_INT_VECTOR_PIC_TIMER, qm_pic_timer_isr);
```

if IRQ based transfers are used.

On Quark SE, this function needs to be registered with:

```
qm_irq_request(QM_IRQ_PIC_TIMER, qm_pic_timer_isr);
```

if IRQ based transfers are used.

Definition at line 27 of file qm_pic_timer.c.

4.14.2.23 QM_ISR_DECLARE (qm_pwm_isr_0)

ISR for PWM 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_PWM_0, qm_pwm_isr_0);
```

if IRQ based transfers are used.

Definition at line 11 of file qm_pwm.c.

4.14.2.24 QM_ISR_DECLARE (qm_rtc_isr_0)

ISR for RTC 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_RTC_0, qm_rtc_isr_0);
```

if IRQ based transfers are used.

Definition at line 10 of file qm_rtc.c.

4.14.2.25 QM_ISR_DECLARE (qm_spi_master_0_isr)

ISR for SPI Master 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_SPI_MASTER_0, qm_spi_master_0_isr);
```

if IRQ based transfers are used.

Definition at line 446 of file qm_spi.c.

4.14.2.26 QM_ISR_DECLARE (qm_spi_master_1_isr)

ISR for SPI Master 1 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_SPI_MASTER_1, qm_spi_master_1_isr);
```

if IRQ based transfers are used.

Definition at line 453 of file qm_spi.c.

4.14.2.27 QM_ISR_DECLARE (qm_uart_0_isr)

ISR for UART 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_UART_0, qm_uart_0_isr);
```

if IRQ based transfers are used.

Definition at line 203 of file qm_uart.c.

4.14.2.28 QM_ISR_DECLARE (qm_uart_1_isr)

ISR for UART 1 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_UART_1, qm_uart_1_isr);
```

if IRQ based transfers are used.

Definition at line 209 of file qm_uart.c.

4.14.2.29 QM_ISR_DECLARE (qm_wdt_isr_0)

ISR for WDT 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_WDT_0, qm_wdt_isr_0);
```

if IRQ based transfers are used.

Definition at line 13 of file qm_wdt.c.

4.15 Mailbox

Mailbox driver.

Data Structures

- struct qm_mbox_msg_t

  Definition of the mailbox message.

Typedefs

- typedef void(∗ qm_mbox_callback_t)(void ∗data)

  Definition of the mailbox callback function prototype.

Enumerations

- enum qm_mbox_ch_status_t { QM_MBOX_CH_IDLE = 0, QM_MBOX_CH_DATA = BIT(0), QM_MBOX_CH_INT = BIT(1), QM_MBOX_CH_STATUS_MASK = BIT(1) | BIT(0) }

  Mailbox status.
- enum qm_mbox_ch_t {
  QM_MBOX_CH_0 = 0, QM_MBOX_CH_1, QM_MBOX_CH_2, QM_MBOX_CH_3,
  QM_MBOX_CH_4, QM_MBOX_CH_5, QM_MBOX_CH_6, QM_MBOX_CH_7,
  QM_MBOX_CH_NUM }

  Mailbox channel.
- enum qm_mbox_payload_t {
  QM_MBOX_PAYLOAD_0 = 0, QM_MBOX_PAYLOAD_1, QM_MBOX_PAYLOAD_2, QM_MBOX_PAYLOAD_3,
  QM_MBOX_PAYLOAD_NUM }

  mailbox message pay-load index values.

Functions

- int qm_mbox_ch_set_config (const qm_mbox_ch_t mbox_ch, qm_mbox_callback_t mpr_cb, void ∗cb_data, const bool irq_en)

  Set the mailbox channel configuration.
- int qm_mbox_ch_write (const qm_mbox_ch_t mbox_ch, const qm_mbox_msg_t ∗const msg)

  Write to a specified mailbox channel.
- int qm_mbox_ch_read (const qm_mbox_ch_t mbox_ch, qm_mbox_msg_t ∗const msg)

  Read specified mailbox channel.
- int qm_mbox_ch_get_status (const qm_mbox_ch_t mbox_ch, qm_mbox_ch_status_t ∗const status)

  Retrieve the specified mailbox channel status.
- int qm_mbox_ch_data_ack (const qm_mbox_ch_t mbox_ch)

  Acknowledge the data arrival.

4.15.1 Detailed Description

Mailbox driver.

4.15.2 Typedef Documentation

4.15.2.1 typedef void(∗ qm_mbox_callback_t)(void ∗data)

Definition of the mailbox callback function prototype.

Parameters

in	*data*	The callback user data.

Definition at line 69 of file qm_mailbox.h.

4.15.3 Enumeration Type Documentation

4.15.3.1 enum qm_mbox_ch_status_t

Mailbox status.

Those values are tied to HW bit setting and made up of the bit 0 and bit 1 of the mailbox channel status register.

Enumerator

QM_MBOX_CH_IDLE No interrupt pending nor any data to consume.

QM_MBOX_CH_DATA Message has not been consumed.

QM_MBOX_CH_INT Channel interrupt pending.

QM_MBOX_CH_STATUS_MASK Status mask.

Definition at line 23 of file qm_mailbox.h.

4.15.3.2 enum qm_mbox_ch_t

Mailbox channel.

Enumerator

QM_MBOX_CH_0 Channel 0.

QM_MBOX_CH_1 Channel 1.

QM_MBOX_CH_2 Channel 2.

QM_MBOX_CH_3 Channel 3.

QM_MBOX_CH_4 Channel 4.

QM_MBOX_CH_5 Channel 5.

QM_MBOX_CH_6 Channel 6.

QM_MBOX_CH_7 Channel 7.

QM_MBOX_CH_NUM Mailbox number of channels.

Definition at line 34 of file qm_mailbox.h.

4.15.3.3 enum qm_mbox_payload_t

mailbox message pay-load index values.

Enumerator

QM_MBOX_PAYLOAD_0 Payload index value 0.

QM_MBOX_PAYLOAD_1 Payload index value 1.

QM_MBOX_PAYLOAD_2 Payload index value 2.

QM_MBOX_PAYLOAD_3 Payload index value 3.

QM_MBOX_PAYLOAD_NUM Numbers of payloads.

Definition at line 49 of file qm_mailbox.h.

4.15.4 Function Documentation

4.15.4.1 int qm_mbox_ch_data_ack (const qm_mbox_ch_t *mbox_ch*)

Acknowledge the data arrival.

Parameters

in	*mbox_ch*	Mailbox identifier to retrieve the status from.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 161 of file qm_mailbox.c.

References QM_MBOX_CH_DATA, and QM_MBOX_CH_NUM.

4.15.4.2 int qm_mbox_ch_get_status (const qm_mbox_ch_t *mbox_ch*, qm_mbox_ch_status_t *const *status*)

Retrieve the specified mailbox channel status.

Parameters

in	*mbox_ch*	Mailbox identifier to retrieve the status from.
out	*status*	Mailbox status. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 151 of file qm_mailbox.c.

References qm_mailbox_t::ch_sts, QM_MBOX_CH_NUM, and QM_MBOX_CH_STATUS_MASK.

4.15.4.3 int qm_mbox_ch_read (const qm_mbox_ch_t *mbox_ch*, qm_mbox_msg_t *const *msg*)

Read specified mailbox channel.

Parameters

in	*mbox_ch*	Mailbox channel identifier.
out	*msg*	Pointer to the data to read from the mailbox channel. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 128 of file qm_mailbox.c.

References qm_mailbox_t::ch_ctrl, qm_mailbox_t::ch_data, qm_mailbox_t::ch_sts, qm_mbox_msg_t::ctrl, qm_mbox_msg_t::data, QM_MBOX_CH_DATA, QM_MBOX_PAYLOAD_0, QM_MBOX_PAYLOAD_1, QM_MBOX_PAYLOAD_2, and QM_MBOX_PAYLOAD_3.

4.15.4.4 int qm_mbox_ch_set_config (const qm_mbox_ch_t *mbox_ch*, qm_mbox_callback_t *mpr_cb*, void * *cb_data*, const bool *irq_en*)

Set the mailbox channel configuration.

Configure a IRQ callback, enables or disables IRQ for the chosen mailbox channel.

Parameters

in	*mbox_ch*	Mailbox to enable.
in	*mpr_cb*	Callback function to call on read mailbox, NULL for a write to Mailbox).
in	*cb_data*	Callback function data to return via the callback. function. This must not be NULL.
in	*irq_en*	Flag to enable/disable IRQ for this channel.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 68 of file qm_mailbox.c.

References qm_irq_mask(), qm_irq_unmask(), QM_MBOX_CH_INT, and QM_MBOX_CH_NUM.

4.15.4.5 int qm_mbox_ch_write (const qm_mbox_ch_t *mbox_ch*, const qm_mbox_msg_t *const *msg*)

Write to a specified mailbox channel.

Parameters

in	*mbox_ch*	Mailbox channel identifier.
in	*msg*	Pointer to the data to write to the mailbox channel. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 106 of file qm_mailbox.c.

References qm_mailbox_t::ch_ctrl, qm_mailbox_t::ch_data, qm_mbox_msg_t::ctrl, qm_mbox_msg_t::data, QM_MBOX_PAYLOAD_0, QM_MBOX_PAYLOAD_1, QM_MBOX_PAYLOAD_2, and QM_MBOX_PAYLOAD_3.

4.16 MPR

Memory Protection Region control for Quark Microcontrollers.

Data Structures

- struct qm_mpr_config_t

  SRAM Memory Protection Region configuration type.

Functions

- int qm_mpr_set_config (const qm_mpr_id_t id, const qm_mpr_config_t *const cfg)

  Configure SRAM controller's Memory Protection Region.
- int qm_mpr_set_violation_policy (const qm_mpr_viol_mode_t mode, qm_mpr_callback_t callback_fn, void *data)

  Configure MPR violation behaviour.

4.16.1 Detailed Description

Memory Protection Region control for Quark Microcontrollers.

4.16.2 Function Documentation

4.16.2.1 int qm_mpr_set_config (const qm_mpr_id_t *id*, const qm_mpr_config_t *const *cfg*)

Configure SRAM controller's Memory Protection Region.

Parameters

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

Returns

int 0 on success, error code otherwise.

Definition at line 23 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.

4.16.2.2 int qm_mpr_set_violation_policy (const qm_mpr_viol_mode_t *mode*, qm_mpr_callback_t *callback_fn*, void * *data*)

Configure MPR violation behaviour.

Parameters

in	*mode*	(generate interrupt, warm reset, enter probe mode).
in	*callback_fn*	for interrupt mode (only).
in	*data*	user data for interrupt mode (only).

Returns

int 0 on success, error code otherwise.

Definition at line 91 of file qm_mpr.c.

References qm_irq_mask(), and qm_irq_unmask().

4.17 PIC Timer

PIC timer.

Data Structures

- struct qm_pic_timer_config_t
  PIC timer configuration type.

Enumerations

- enum qm_pic_timer_mode_t { QM_PIC_TIMER_MODE_ONE_SHOT, QM_PIC_TIMER_MODE_PERIODIC }
  PIC timer mode type.

Functions

- int qm_pic_timer_set_config (const qm_pic_timer_config_t *const cfg)
  Set the PIC timer configuration.
- int qm_pic_timer_set (const uint32_t count)
  Set the current count value of the PIC timer.
- int qm_pic_timer_get (uint32_t *const count)
  Get the current count value of the PIC timer.

4.17.1 Detailed Description

PIC timer.

4.17.2 Enumeration Type Documentation

4.17.2.1 enum qm_pic_timer_mode_t

PIC timer mode type.

Enumerator

QM_PIC_TIMER_MODE_ONE_SHOT One shot mode.

QM_PIC_TIMER_MODE_PERIODIC Periodic mode.

Definition at line 21 of file qm_pic_timer.h.

4.17.3 Function Documentation

4.17.3.1 int qm_pic_timer_get (uint32_t *const *count*)

Get the current count value of the PIC timer.

Parameters

out	*count*	Pointer to the store the timer count. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 76 of file qm_pic_timer.c.

4.17.3.2 int qm_pic_timer_set (const uint32_t *count*)

Set the current count value of the PIC timer.

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

Parameters

in	*count*	Value to load the timer with.

Returns

Standard errno return type for QMSI.

Return values

Always	returns 0.
Negative	errno for possible error codes.

Definition at line 69 of file qm_pic_timer.c.

4.17.3.3 int qm_pic_timer_set_config (const qm_pic_timer_config_t *const *cfg*)

Set the PIC timer configuration.

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

Parameters

in	*cfg*	PIC timer configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 41 of file qm_pic_timer.c.

References qm_pic_timer_config_t::callback, qm_pic_timer_config_t::callback_data, qm_pic_timer_config_t::int_-en, qm_pic_timer_config_t::mode, and QM_PIC_TIMER_MODE_PERIODIC.

4.18 Pin Muxing setup

Pin muxing configuration.

Enumerations

- enum qm_pmux_fn_t { QM_PMUX_FN_0, QM_PMUX_FN_1, QM_PMUX_FN_2, QM_PMUX_FN_3 }

  Pin function type.
- enum qm_pmux_slew_t { QM_PMUX_SLEW_2MA, QM_PMUX_SLEW_4MA, QM_PMUX_SLEW_12MA, QM_PMUX_SLEW_16MA, QM_PMUX_SLEW_NUM }

  Pin slew rate setting.
- enum qm_pin_id_t { QM_PIN_ID_0, QM_PIN_ID_1, QM_PIN_ID_2, QM_PIN_ID_3, QM_PIN_ID_4, QM_PIN_ID_5, QM_PIN_ID_6, QM_PIN_ID_7, QM_PIN_ID_8, QM_PIN_ID_9, QM_PIN_ID_10, QM_PIN_ID_11, QM_PIN_ID_12, QM_PIN_ID_13, QM_PIN_ID_14, QM_PIN_ID_15, QM_PIN_ID_16, QM_PIN_ID_17, QM_PIN_ID_18, QM_PIN_ID_19, QM_PIN_ID_20, QM_PIN_ID_21, QM_PIN_ID_22, QM_PIN_ID_23, QM_PIN_ID_24, QM_PIN_ID_25, QM_PIN_ID_26, QM_PIN_ID_27, QM_PIN_ID_28, QM_PIN_ID_29, QM_PIN_ID_30, QM_PIN_ID_31, QM_PIN_ID_32, QM_PIN_ID_33, QM_PIN_ID_34, QM_PIN_ID_35, QM_PIN_ID_36, QM_PIN_ID_37, QM_PIN_ID_38, QM_PIN_ID_39, QM_PIN_ID_40, QM_PIN_ID_41, QM_PIN_ID_42, QM_PIN_ID_43, QM_PIN_ID_44, QM_PIN_ID_45, QM_PIN_ID_46, QM_PIN_ID_47, QM_PIN_ID_48, QM_PIN_ID_49, QM_PIN_ID_50, QM_PIN_ID_51, QM_PIN_ID_52, QM_PIN_ID_53, QM_PIN_ID_54, QM_PIN_ID_55, QM_PIN_ID_56, QM_PIN_ID_57, QM_PIN_ID_58, QM_PIN_ID_59, QM_PIN_ID_60, QM_PIN_ID_61, QM_PIN_ID_62, QM_PIN_ID_63, QM_PIN_ID_64, QM_PIN_ID_65, QM_PIN_ID_66, QM_PIN_ID_67, QM_PIN_ID_68 }

  External Pad pin identifiers.

Functions

- int qm_pmux_select (const qm_pin_id_t pin, const qm_pmux_fn_t fn)

  Set up pin muxing for a SoC pin.
- int qm_pmux_set_slew (const qm_pin_id_t pin, const qm_pmux_slew_t slew)

  Set up pin's slew rate in the pin mux controller.
- int qm_pmux_input_en (const qm_pin_id_t pin, const bool enable)

  Enable input for a pin in the pin mux controller.
- int qm_pmux_pullup_en (const qm_pin_id_t pin, const bool enable)

  Enable pullup for a pin in the pin mux controller.

4.18.1 Detailed Description

Pin muxing configuration.

4.18.2 Enumeration Type Documentation

4.18.2.1 enum qm_pin_id_t

External Pad pin identifiers.

Enumerator

QM_PIN_ID_0 Pin id 0.

QM_PIN_ID_1 Pin id 1.

QM_PIN_ID_2 Pin id 2.

QM_PIN_ID_3 Pin id 3.

QM_PIN_ID_4 Pin id 4.

QM_PIN_ID_5 Pin id 5.

QM_PIN_ID_6 Pin id 6.

QM_PIN_ID_7 Pin id 7.

QM_PIN_ID_8 Pin id 8.

QM_PIN_ID_9 Pin id 9.

QM_PIN_ID_10 Pin id 10.

QM_PIN_ID_11 Pin id 11.

QM_PIN_ID_12 Pin id 12.

QM_PIN_ID_13 Pin id 13.

QM_PIN_ID_14 Pin id 14.

QM_PIN_ID_15 Pin id 15.

QM_PIN_ID_16 Pin id 16.

QM_PIN_ID_17 Pin id 17.

QM_PIN_ID_18 Pin id 18.

QM_PIN_ID_19 Pin id 19.

QM_PIN_ID_20 Pin id 20.

QM_PIN_ID_21 Pin id 21.

QM_PIN_ID_22 Pin id 22.

QM_PIN_ID_23 Pin id 23.

QM_PIN_ID_24 Pin id 24.

QM_PIN_ID_25 Pin id 25.

QM_PIN_ID_26 Pin id 26.

QM_PIN_ID_27 Pin id 27.

QM_PIN_ID_28 Pin id 28.

QM_PIN_ID_29 Pin id 29.

QM_PIN_ID_30 Pin id 30.

QM_PIN_ID_31 Pin id 31.

QM_PIN_ID_32 Pin id 32.

QM_PIN_ID_33 Pin id 33.

QM_PIN_ID_34 Pin id 34.

QM_PIN_ID_35 Pin id 35.

QM_PIN_ID_36 Pin id 36.

QM_PIN_ID_37 Pin id 37.

QM_PIN_ID_38 Pin id 38.

QM_PIN_ID_39 Pin id 39.

QM_PIN_ID_40 Pin id 40.

QM_PIN_ID_41 Pin id 41.

QM_PIN_ID_42 Pin id 42.

QM_PIN_ID_43 Pin id 43.

QM_PIN_ID_44 Pin id 44.
QM_PIN_ID_45 Pin id 45.
QM_PIN_ID_46 Pin id 46.
QM_PIN_ID_47 Pin id 47.
QM_PIN_ID_48 Pin id 48.
QM_PIN_ID_49 Pin id 49.
QM_PIN_ID_50 Pin id 50.
QM_PIN_ID_51 Pin id 51.
QM_PIN_ID_52 Pin id 52.
QM_PIN_ID_53 Pin id 53.
QM_PIN_ID_54 Pin id 54.
QM_PIN_ID_55 Pin id 55.
QM_PIN_ID_56 Pin id 56.
QM_PIN_ID_57 Pin id 57.
QM_PIN_ID_58 Pin id 58.
QM_PIN_ID_59 Pin id 59.
QM_PIN_ID_60 Pin id 60.
QM_PIN_ID_61 Pin id 61.
QM_PIN_ID_62 Pin id 62.
QM_PIN_ID_63 Pin id 63.
QM_PIN_ID_64 Pin id 64.
QM_PIN_ID_65 Pin id 65.
QM_PIN_ID_66 Pin id 66.
QM_PIN_ID_67 Pin id 67.
QM_PIN_ID_68 Pin id 68.

Definition at line 45 of file qm_pinmux.h.

4.18.2.2 enum qm_pmux_fn_t

Pin function type.

Enumerator

QM_PMUX_FN_0 Gpio function 0.
QM_PMUX_FN_1 Gpio function 0.
QM_PMUX_FN_2 Gpio function 0.
QM_PMUX_FN_3 Gpio function 0.

Definition at line 21 of file qm_pinmux.h.

4.18.2.3 enum qm_pmux_slew_t

Pin slew rate setting.

Enumerator

QM_PMUX_SLEW_2MA Set gpio slew rate to 2MA.
QM_PMUX_SLEW_4MA Set gpio slew rate to 4MA.
QM_PMUX_SLEW_12MA Set gpio slew rate to 12MA.
QM_PMUX_SLEW_16MA Set gpio slew rate to 16MA.
QM_PMUX_SLEW_NUM Max number of slew rate options.

Definition at line 31 of file qm_pinmux.h.

4.18.3 Function Documentation

4.18.3.1 int qm_pmux_input_en (const qm_pin_id_t *pin*, const bool *enable*)

Enable input for a pin in the pin mux controller.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 67 of file qm_pinmux.c.

4.18.3.2 int qm_pmux_pullup_en (const qm_pin_id_t *pin*, const bool *enable*)

Enable pullup for a pin in the pin mux controller.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 82 of file qm_pinmux.c.

4.18.3.3 int qm_pmux_select (const qm_pin_id_t *pin*, const qm_pmux_fn_t *fn*)

Set up pin muxing for a SoC pin.

Select one of the pin functions.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 37 of file qm_pinmux.c.

References QM_PMUX_FN_3.

4.18.3.4 int qm_pmux_set_slew (const qm_pin_id_t *pin*, const qm_pmux_slew_t *slew*)

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 51 of file qm_pinmux.c.

References QM_PMUX_SLEW_NUM.

4.19 PWM / Timer

Pulse width modulation and Timer driver.

Data Structures

- struct qm_pwm_config_t
  PWM / Timer configuration type.

Enumerations

- enum qm_pwm_mode_t { QM_PWM_MODE_TIMER_FREE_RUNNING = 0, QM_PWM_MODE_TIMER_COUNT = 2, QM_PWM_MODE_PWM = 10 }
  PWM operating mode type.

Functions

- int qm_pwm_set_config (const qm_pwm_t pwm, const qm_pwm_id_t id, const qm_pwm_config_t *const cfg)
  Change the configuration of a PWM channel.
- int 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 the next period values of a PWM channel.
- int 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.
- int qm_pwm_start (const qm_pwm_t pwm, const qm_pwm_id_t id)
  Start a PWM/timer channel.
- int qm_pwm_stop (const qm_pwm_t pwm, const qm_pwm_id_t id)
  Stop a PWM/timer channel.

4.19.1 Detailed Description

Pulse width modulation and Timer driver.

4.19.2 Enumeration Type Documentation

4.19.2.1 enum qm_pwm_mode_t

PWM operating mode type.

Enumerator

QM_PWM_MODE_TIMER_FREE_RUNNING Timer: free runnig mode.

QM_PWM_MODE_TIMER_COUNT Timer: Counter mode.

QM_PWM_MODE_PWM Pwm mode.

Definition at line 25 of file qm_pwm.h.

4.19.3 Function Documentation

4.19.3.1 int qm_pwm_get (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*, uint32_t *const *lo_count*, uint32_t *const *hi_count*)

Get the current period values of a PWM channel.

Parameters

in	*pwm*	Which PWM module to get the count of.
in	*id*	PWM channel id to read the values of.
out	*lo_count*	Num of cycles the output is driven low. This must not be NULL.
out	*hi_count*	Num of cycles the output is driven high. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 87 of file qm_pwm.c.

4.19.3.2 int 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 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.

Set PWM period counts.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 68 of file qm_pwm.c.

References QM_PWM_MODE_PWM.

4.19.3.3 int qm_pwm_set_config (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*, const qm_pwm_config_t *const *cfg*)

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.

Set PWM channel configuration.

Parameters

in	*pwm*	Which PWM module to configure.
in	*id*	PWM channel id to configure.
in	*cfg*	New configuration for PWM. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 44 of file qm_pwm.c.

References qm_pwm_config_t::callback, qm_pwm_config_t::callback_data, qm_pwm_config_t::hi_count, qm_pwm_config_t::lo_count, qm_pwm_config_t::mask_interrupt, qm_pwm_config_t::mode, and QM_PWM_MODE_PWM.

4.19.3.4 int qm_pwm_start (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*)

Start a PWM/timer channel.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 24 of file qm_pwm.c.

4.19.3.5 int qm_pwm_stop (const qm_pwm_t *pwm*, const qm_pwm_id_t *id*)

Stop a PWM/timer channel.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 34 of file qm_pwm.c.

4.20 RTC

Real Time clock.

Data Structures

- struct qm_rtc_config_t

  RTC configuration type.

Functions

- int qm_rtc_set_config (const qm_rtc_t rtc, const qm_rtc_config_t *const cfg)

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

  Set Alarm value.

4.20.1 Detailed Description

Real Time clock.

4.20.2 Function Documentation

4.20.2.1 int qm_rtc_set_alarm (const qm_rtc_t *rtc,* const uint32_t *alarm_val*)

Set Alarm value.

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

The RTC clock resides in a different clock domain to the system clock. It takes 3-4 RTC ticks for a system clock write to propagate to the RTC domain. If an entry to sleep is initiated without waiting for the transaction to complete the SOC will not wake from sleep.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 77 of file qm_rtc.c.

Referenced by qm_rtc_set_config().

4.20.2.2 int qm_rtc_set_config (const qm_rtc_t *rtc,* const qm_rtc_config_t *const *cfg*)

Set RTC configuration.

This includes 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.

The RTC clock resides in a different clock domain to the system clock. It takes 3-4 RTC ticks for a system clock write to propagate to the RTC domain. If an entry to sleep is initiated without waiting for the transaction to complete the SOC will not wake from sleep.

Parameters

in	*rtc*	RTC index.
in	*cfg*	New RTC configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 51 of file qm_rtc.c.

References qm_rtc_config_t::alarm_en, qm_rtc_config_t::alarm_val, qm_rtc_config_t::callback, qm_rtc_config_t::callback_data, clk_rtc_set_div(), qm_rtc_config_t::init_val, and qm_rtc_set_alarm().

4.21 SPI

SPI peripheral driver for Quark Microcontrollers.

Data Structures

- struct qm_spi_config_t

  SPI configuration type.
- struct qm_spi_async_transfer_t

  SPI IRQ transfer type.
- struct qm_spi_transfer_t

  SPI transfer type.

Enumerations

- enum qm_spi_frame_size_t {
  QM_SPI_FRAME_SIZE_4_BIT = 3, QM_SPI_FRAME_SIZE_5_BIT, QM_SPI_FRAME_SIZE_6_BIT, QM_SPI_FRAME_SIZE_7_BIT,
  QM_SPI_FRAME_SIZE_8_BIT, QM_SPI_FRAME_SIZE_9_BIT, QM_SPI_FRAME_SIZE_10_BIT, QM_SPI_FRAME_SIZE_11_BIT,
  QM_SPI_FRAME_SIZE_12_BIT, QM_SPI_FRAME_SIZE_13_BIT, QM_SPI_FRAME_SIZE_14_BIT, QM_SPI_FRAME_SIZE_15_BIT,
  QM_SPI_FRAME_SIZE_16_BIT, QM_SPI_FRAME_SIZE_17_BIT, QM_SPI_FRAME_SIZE_18_BIT, QM_SPI_FRAME_SIZE_19_BIT,
  QM_SPI_FRAME_SIZE_20_BIT, QM_SPI_FRAME_SIZE_21_BIT, QM_SPI_FRAME_SIZE_22_BIT, QM_SPI_FRAME_SIZE_23_BIT,
  QM_SPI_FRAME_SIZE_24_BIT, QM_SPI_FRAME_SIZE_25_BIT, QM_SPI_FRAME_SIZE_26_BIT, QM_SPI_FRAME_SIZE_27_BIT,
  QM_SPI_FRAME_SIZE_28_BIT, QM_SPI_FRAME_SIZE_29_BIT, QM_SPI_FRAME_SIZE_30_BIT, QM_SPI_FRAME_SIZE_31_BIT,
  QM_SPI_FRAME_SIZE_32_BIT }

  QM SPI frame size type.
- enum qm_spi_tmode_t { QM_SPI_TMOD_TX_RX, QM_SPI_TMOD_TX, QM_SPI_TMOD_RX, QM_SPI_TMOD_EEPROM_READ }

  SPI transfer mode type.
- enum qm_spi_bmode_t { QM_SPI_BMODE_0, QM_SPI_BMODE_1, QM_SPI_BMODE_2, QM_SPI_BMODE_3 }

  SPI bus mode type.
- enum qm_spi_slave_select_t {
  QM_SPI_SS_DISABLED = 0, QM_SPI_SS_0 = BIT(0), QM_SPI_SS_1 = BIT(1), QM_SPI_SS_2 = BIT(2),
  QM_SPI_SS_3 = BIT(3) }

  SPI slave select type.
- enum qm_spi_status_t { QM_SPI_IDLE, QM_SPI_BUSY, QM_SPI_RX_OVERFLOW }

  SPI status.

Functions

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

  Set SPI configuration.
- int qm_spi_slave_select (const qm_spi_t spi, const qm_spi_slave_select_t ss)

  Select which slave to perform SPI transmissions on.
- int qm_spi_get_status (const qm_spi_t spi, qm_spi_status_t *const status)

  Get SPI bus status.

- int qm_spi_transfer (const qm_spi_t spi, const qm_spi_transfer_t *const xfer, qm_spi_status_t *const status)
  Multi-frame read / write on SPI.
- int qm_spi_irq_transfer (const qm_spi_t spi, const qm_spi_async_transfer_t *const xfer)
  Interrupt based transfer on SPI.
- int qm_spi_dma_channel_config (const qm_spi_t spi, const qm_dma_t dma_ctrl_id, const qm_dma_channel_id_t dma_channel_id, const qm_dma_channel_direction_t dma_channel_direction)
  Configure a DMA channel with a specific transfer direction.
- int qm_spi_dma_transfer (const qm_spi_t spi, const qm_spi_async_transfer_t *const xfer)
  Perform a DMA-based transfer on the SPI bus.
- int qm_spi_irq_transfer_terminate (const qm_spi_t spi)
  Terminate SPI IRQ transfer.
- int qm_spi_dma_transfer_terminate (const qm_spi_t spi)
  Terminate the current DMA transfer on the SPI bus.

4.21.1 Detailed Description

SPI peripheral driver for Quark Microcontrollers.

4.21.2 Enumeration Type Documentation

4.21.2.1 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 67 of file qm_spi.h.

4.21.2.2 enum qm_spi_frame_size_t

QM SPI frame size type.

Enumerator

QM_SPI_FRAME_SIZE_4_BIT 4 bit frame.
QM_SPI_FRAME_SIZE_5_BIT 5 bit frame.
QM_SPI_FRAME_SIZE_6_BIT 6 bit frame.
QM_SPI_FRAME_SIZE_7_BIT 7 bit frame.
QM_SPI_FRAME_SIZE_8_BIT 8 bit frame.
QM_SPI_FRAME_SIZE_9_BIT 9 bit frame.
QM_SPI_FRAME_SIZE_10_BIT 10 bit frame.
QM_SPI_FRAME_SIZE_11_BIT 11 bit frame.
QM_SPI_FRAME_SIZE_12_BIT 12 bit frame.
QM_SPI_FRAME_SIZE_13_BIT 13 bit frame.
QM_SPI_FRAME_SIZE_14_BIT 14 bit frame.
QM_SPI_FRAME_SIZE_15_BIT 15 bit frame.

QM_SPI_FRAME_SIZE_16_BIT 16 bit frame.

QM_SPI_FRAME_SIZE_17_BIT 17 bit frame.

QM_SPI_FRAME_SIZE_18_BIT 18 bit frame.

QM_SPI_FRAME_SIZE_19_BIT 19 bit frame.

QM_SPI_FRAME_SIZE_20_BIT 20 bit frame.

QM_SPI_FRAME_SIZE_21_BIT 21 bit frame.

QM_SPI_FRAME_SIZE_22_BIT 22 bit frame.

QM_SPI_FRAME_SIZE_23_BIT 23 bit frame.

QM_SPI_FRAME_SIZE_24_BIT 24 bit frame.

QM_SPI_FRAME_SIZE_25_BIT 25 bit frame.

QM_SPI_FRAME_SIZE_26_BIT 26 bit frame.

QM_SPI_FRAME_SIZE_27_BIT 27 bit frame.

QM_SPI_FRAME_SIZE_28_BIT 28 bit frame.

QM_SPI_FRAME_SIZE_29_BIT 29 bit frame.

QM_SPI_FRAME_SIZE_30_BIT 30 bit frame.

QM_SPI_FRAME_SIZE_31_BIT 31 bit frame.

QM_SPI_FRAME_SIZE_32_BIT 32 bit frame.

Definition at line 22 of file qm_spi.h.

4.21.2.3 enum qm_spi_slave_select_t

SPI slave select type.

Slave selects can combined by logical OR if multiple slaves are selected during one transfer. Setting only QM_SPI-_SS_DISABLED prevents the controller from starting the transfer.

Enumerator

QM_SPI_SS_DISABLED Slave select disable.

QM_SPI_SS_0 Slave Select 0.

QM_SPI_SS_1 Slave Select 1.

QM_SPI_SS_2 Slave Select 2.

QM_SPI_SS_3 Slave Select 3.

Definition at line 81 of file qm_spi.h.

4.21.2.4 enum qm_spi_status_t

SPI status.

Enumerator

QM_SPI_IDLE SPI device is not in use.

QM_SPI_BUSY SPI device is busy.

QM_SPI_RX_OVERFLOW RX transfer has overflown.

Definition at line 92 of file qm_spi.h.

4.21.2.5 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 57 of file qm_spi.h.

4.21.3 Function Documentation

4.21.3.1 int qm_spi_dma_channel_config (const qm_spi_t *spi*, const qm_dma_t *dma_ctrl_id*, const qm_dma_channel_id_t *dma_channel_id*, const qm_dma_channel_direction_t *dma_channel_direction*)

Configure a DMA channel with a specific transfer direction.

The user is responsible for managing the allocation of the pool of DMA channels provided by each DMA core to the different peripheral drivers that require them.

Note that a SPI controller cannot use different DMA cores to manage transfers in different directions.

This function configures DMA channel parameters that are unlikely to change between transfers, like transaction width, burst size, and handshake interface parameters. The user will likely only call this function once for the lifetime of an application unless the channel needs to be repurposed.

Note that qm_dma_init() must first be called before configuring a channel.

Parameters

in	*spi*	SPI controller identifier.
in	*dma_ctrl_id*	DMA controller identifier.
in	*dma_channel_id*	DMA channel identifier.
in	*dma_channel_direction*	DMA channel direction, either QM_DMA_MEMORY_TO_PERIPHERAL (TX transfer) or QM_DMA_PERIPHERAL_TO_MEMORY (RX transfer).

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 565 of file qm_spi.c.

References qm_dma_channel_config_t::callback_context, qm_dma_channel_config_t::channel_direction, qm_dma_channel_config_t::client_callback, qm_dma_channel_config_t::destination_burst_length, qm_dma_channel_config_t::destination_transfer_width, DMA_HW_IF_SPI_MASTER_0_RX, DMA_HW_IF_SPI_MASTER_0_TX, DMA_HW_IF_SPI_MASTER_1_RX, DMA_HW_IF_SPI_MASTER_1_TX, qm_dma_channel_config_t::handshake_interface, qm_dma_channel_config_t::handshake_polarity, QM_DMA_CHANNEL_NUM, qm_dma_channel_set_config(), QM_DMA_HANDSHAKE_POLARITY_HIGH, QM_DMA_MEMORY_TO_PERIPHERAL, QM_DMA_NUM, QM_DMA_PERIPHERAL_TO_MEMORY, QM_DMA_TRANS_WIDTH_16, QM_DMA_TRANS_WIDTH_32, QM_DMA_TRANS_WIDTH_8, qm_dma_channel_config_t::source_burst_length, and qm_dma_channel_config_t::source_transfer_width.

4.21.3.2 int qm_spi_dma_transfer (const qm_spi_t *spi,* const qm_spi_async_transfer_t *const *xfer*)

Perform a DMA-based transfer on the SPI bus.

If transfer mode is full duplex (QM_SPI_TMOD_TX_RX), then tx_len and rx_len must be equal and neither of both callbacks can 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. Transfer length is limited to 4KB.

For starting a transfer, this controller demands at least one slave select line (SS) to be enabled. Thus, a call to qm_spi_slave_select() with one of the four SS valid lines is mandatory. This is true even if the native slave select line is not used (i.e. when a GPIO is used to drive the SS signal manually).

Note that qm_spi_dma_channel_config() must first be called in order to configure all DMA channels needed for a transfer.

Parameters

in	*spi*	SPI controller identifier.
in	*xfer*	Structure containing pre-allocated write and read data buffers and callback functions. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 664 of file qm_spi.c.

References qm_dma_transfer_t::block_size, qm_spi_reg_t::ctrlr1, qm_dma_transfer_t::destination_address, qm_spi_reg_t::dmacr, qm_spi_reg_t::dmardlr, qm_spi_reg_t::dmatdlr, qm_spi_reg_t::dr, qm_spi_reg_t::imr, QM_DMA_CHANNEL_NUM, QM_DMA_NUM, qm_dma_transfer_set_config(), qm_dma_transfer_start(), qm_spi_dma_transfer_terminate(), QM_SPI_TMOD_EEPROM_READ, QM_SPI_TMOD_RX, QM_SPI_TMOD_TX, QM_SPI_TMOD_TX_RX, qm_spi_async_transfer_t::rx, qm_spi_async_transfer_t::rx_len, qm_dma_transfer_t::source_address, qm_spi_reg_t::ssienr, qm_spi_async_transfer_t::tx, and qm_spi_async_transfer_t::tx_len.

4.21.3.3 int qm_spi_dma_transfer_terminate (const qm_spi_t *spi*)

Terminate the current DMA transfer on the SPI bus.

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

Parameters

in	*spi*	SPI controller identifier.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 797 of file qm_spi.c.

References QM_DMA_CHANNEL_NUM, and qm_dma_transfer_terminate().

Referenced by qm_spi_dma_transfer().

4.21.3.4 int qm_spi_get_status (const qm_spi_t *spi*, qm_spi_status_t *const *status*)

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.
out	*status*	Get spi status. This must not be null.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 290 of file qm_spi.c.

References QM_SPI_BUSY, QM_SPI_IDLE, QM_SPI_RX_OVERFLOW, qm_spi_reg_t::risr, and qm_spi_reg_t::sr.

4.21.3.5 int qm_spi_irq_transfer (const qm_spi_t *spi*, const 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. 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.

For starting a transfer, this controller demands at least one slave select line (SS) to be enabled. Thus, a call to qm_spi_slave_select() with one of the four SS valid lines is mandatory. This is true even if the native slave select line is not used (i.e. when a GPIO is used to drive the SS signal manually).

Parameters

in	*spi*	Which SPI to transfer to / from.
in	*xfer*	Transfer structure includes write / read buffers, length, user callback function and the callback context data. The structure must not be NULL and must be kept valid until the transfer is complete.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Enable SPI Device

Definition at line 390 of file qm_spi.c.

References qm_spi_reg_t::ctrlr1, qm_spi_reg_t::imr, QM_SPI_TMOD_EEPROM_READ, QM_SPI_TMOD_RX, QM_SPI_TMOD_TX, QM_SPI_TMOD_TX_RX, qm_spi_async_transfer_t::rx_len, qm_spi_reg_t::rxftlr, qm_spi_reg_t::ssienr, qm_spi_async_transfer_t::tx_len, and qm_spi_reg_t::txftlr.

4.21.3.6 int qm_spi_irq_transfer_terminate (const qm_spi_t *spi*)

Terminate SPI IRQ transfer.

Terminate the current IRQ transfer on the SPI bus. This will cause the user callback to be called with error code set to -ECANCELED.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Disable SPI device

Definition at line 460 of file qm_spi.c.

References qm_spi_async_transfer_t::callback, qm_spi_async_transfer_t::callback_data, qm_spi_reg_t::imr, QM_SPI_IDLE, QM_SPI_TMOD_TX, QM_SPI_TMOD_TX_RX, and qm_spi_reg_t::ssienr.

4.21.3.7 int qm_spi_set_config (const qm_spi_t *spi*, const qm_spi_config_t *const *cfg*)

Set SPI configuration.

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

Parameters

in	*spi*	Which SPI module to configure.
in	*cfg*	New configuration for SPI. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 249 of file qm_spi.c.

References qm_spi_reg_t::baudr, qm_spi_config_t::bus_mode, qm_spi_config_t::clk_divider, qm_spi_reg_t::ctrlr0, qm_spi_config_t::frame_size, and qm_spi_config_t::transfer_mode.

4.21.3.8 int qm_spi_slave_select (const qm_spi_t *spi*, const qm_spi_slave_select_t *ss*)

Select which slave to perform SPI transmissions on.

Parameters

in	*spi*	Which SPI module to configure.
in	*ss*	Which slave select line to enable when doing transmissions.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 278 of file qm_spi.c.

4.21.3.9 int qm_spi_transfer (const qm_spi_t *spi*, const qm_spi_transfer_t *const *xfer*, qm_spi_status_t *const *status*)

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.

For starting a transfer, this controller demands at least one slave select line (SS) to be enabled. Thus, a call to qm_spi_slave_select() with one of the four SS valid lines is mandatory. This is true even if the native slave select line is not used (i.e. when a GPIO is used to drive the SS signal manually).

Parameters

in	*spi*	Which SPI to read/write on.
in	*xfer*	Structure containing pre-allocated write and read data buffers. This must not be NULL.
out	*status*	Get spi status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 310 of file qm_spi.c.

References qm_spi_reg_t::ctrlr1, qm_spi_reg_t::imr, QM_SPI_RX_OVERFLOW, QM_SPI_TMOD_EEPROM_READ, QM_SPI_TMOD_RX, QM_SPI_TMOD_TX, QM_SPI_TMOD_TX_RX, qm_spi_reg_t::risr, qm_spi_transfer_t::rx, qm_spi_transfer_t::rx_len, qm_spi_reg_t::rxoicr, qm_spi_reg_t::sr, qm_spi_reg_t::ssienr, qm_spi_transfer_t::tx, and qm_spi_transfer_t::tx_len.

4.22 UART

UART peripheral driver.

Data Structures

- struct qm_uart_config_t

  UART configuration type.
- struct qm_uart_transfer_t

  UART asynchronous transfer structure.

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 }

  UART Line control.
- enum qm_uart_status_t {
  QM_UART_IDLE = 0, QM_UART_RX_OE = BIT(1), QM_UART_RX_PE = BIT(2), QM_UART_RX_FE = BIT(3),
  QM_UART_RX_BI = BIT(4), QM_UART_TX_BUSY = BIT(5), QM_UART_RX_BUSY = BIT(6), QM_UART_TX_NFULL = BIT(7),
  QM_UART_RX_NEMPTY = BIT(8) }

  UART Status type.

Functions

- int qm_uart_set_config (const qm_uart_t uart, const qm_uart_config_t *const cfg)

  Set UART configuration.
- int qm_uart_get_status (const qm_uart_t uart, qm_uart_status_t *const status)

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

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

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

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

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

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

  Interrupt based TX on UART.

- int qm_uart_irq_read (const qm_uart_t uart, const qm_uart_transfer_t *const xfer)

  Interrupt based RX on UART.
- int qm_uart_irq_write_terminate (const qm_uart_t uart)

  Terminate UART IRQ TX transfer.
- int qm_uart_irq_read_terminate (const qm_uart_t uart)

  Terminate UART IRQ RX transfer.
- int qm_uart_dma_channel_config (const qm_uart_t uart, const qm_dma_t dma_ctrl_id, const qm_dma_channel_id_t dma_channel_id, const qm_dma_channel_direction_t dma_channel_direction)

  Configure a DMA channel with a specific transfer direction.
- int qm_uart_dma_write (const qm_uart_t uart, const qm_uart_transfer_t *const xfer)

  Perform a DMA-based TX transfer on the UART bus.
- int qm_uart_dma_read (const qm_uart_t uart, const qm_uart_transfer_t *const xfer)

  Perform a DMA-based RX transfer on the UART bus.
- int qm_uart_dma_write_terminate (const qm_uart_t uart)

  Terminate the current DMA TX transfer on the UART bus.
- int qm_uart_dma_read_terminate (const qm_uart_t uart)

  Terminate the current DMA RX transfer on the UART bus.

4.22.1 Detailed Description

UART peripheral driver.

4.22.2 Enumeration Type Documentation

4.22.2.1 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 par., 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 116 of file qm_uart.h.

4.22.2.2 enum qm_uart_status_t

UART Status type.

Enumerator

QM_UART_IDLE IDLE.

QM_UART_RX_OE Receiver overrun.

QM_UART_RX_PE Parity error.

QM_UART_RX_FE Framing error.

QM_UART_RX_BI Break interrupt.

QM_UART_TX_BUSY TX Busy flag.

QM_UART_RX_BUSY RX Busy flag.

QM_UART_TX_NFULL TX FIFO not full.

QM_UART_RX_NEMPTY RX FIFO not empty.

Definition at line 146 of file qm_uart.h.

4.22.3 Function Documentation

4.22.3.1 int qm_uart_dma_channel_config (const qm_uart_t *uart*, const qm_dma_t *dma_ctrl_id*, const qm_dma_channel_id_t *dma_channel_id*, const qm_dma_channel_direction_t *dma_channel_direction*)

Configure a DMA channel with a specific transfer direction.

The user is responsible for managing the allocation of the pool of DMA channels provided by each DMA core to the different peripheral drivers that require them.

This function configures DMA channel parameters that are unlikely to change between transfers, like transaction width, burst size, and handshake interface parameters. The user will likely only call this function once for the lifetime of an application unless the channel needs to be repurposed.

Note that qm_dma_init() must first be called before configuring a channel.

Parameters

in	*uart*	UART index.
in	*dma_ctrl_id*	DMA controller identifier.
in	*dma_channel_id*	DMA channel identifier.
in	*dma_channel_direction*	DMA channel direction, either QM_DMA_MEMORY_TO_PERIPHERAL (write transfer) or QM_DMA_PERIPHERAL_TO_MEMORY (read transfer).

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 526 of file qm_uart.c.

References qm_dma_channel_config_t::callback_context, qm_dma_channel_config_t::channel_direction, qm_dma_channel_config_t::client_callback, qm_dma_channel_config_t::destination_burst_length, qm_dma_channel_config_t::destination_transfer_width, DMA_HW_IF_UART_A_RX, DMA_HW_IF_UART_A_TX, DMA_HW_IF_UART_B_RX, DMA_HW_IF_UART_B_TX, qm_dma_channel_config_t::handshake_interface, qm_dma_channel_config_t::handshake_polarity, QM_DMA_BURST_TRANS_LENGTH_8, QM_DMA_CHANNEL_NUM, qm_dma_channel_set_config(), QM_DMA_HANDSHAKE_POLARITY_LOW, QM_DMA_MEMORY_TO_PERIPHERAL, QM_DMA_NUM, QM_DMA_PERIPHERAL_TO_MEMORY, QM_DMA_TRANS_WIDTH_8, qm_dma_channel_config_t::source_burst_length, and qm_dma_channel_config_t::source_transfer_width.

4.22.3.2 int qm_uart_dma_read (const qm_uart_t *uart*, const qm_uart_transfer_t *const *xfer*)

Perform a DMA-based RX transfer on the UART bus.

In order for this call to succeed, previously the user must have configured a DMA channel with direction QM_DMA_PERIPHERAL_TO_MEMORY to be used on this UART, calling qm_uart_dma_channel_config(). The transfer length is limited to 4KB.

Parameters

in	*uart*	UART index.
in	*xfer*	Structure containing a pre-allocated read buffer and callback functions. This must not be NULL. Callback pointer must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 620 of file qm_uart.c.

References qm_dma_transfer_t::block_size, qm_uart_transfer_t::data, qm_uart_transfer_t::data_len, qm_dma_transfer_t::destination_address, qm_uart_reg_t::iir_fcr, qm_dma_transfer_set_config(), qm_dma_transfer_start(), qm_uart_reg_t::rbr_thr_dll, and qm_dma_transfer_t::source_address.

4.22.3.3 int qm_uart_dma_read_terminate (const qm_uart_t *uart*)

Terminate the current DMA RX transfer on the UART bus.

This will cause the relevant callbacks to be called.

Parameters

in	*uart*	UART index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 663 of file qm_uart.c.

References qm_dma_transfer_terminate().

4.22.3.4 int qm_uart_dma_write (const qm_uart_t *uart*, const qm_uart_transfer_t *const *xfer*)

Perform a DMA-based TX transfer on the UART bus.

In order for this call to succeed, previously the user must have configured a DMA channel with direction QM_DMA_MEMORY_TO_PERIPHERAL to be used on this UART, calling qm_uart_dma_channel_config(). The transfer length is limited to 4KB.

Note that this function uses the UART TX FIFO empty interrupt and therefore, in addition to the DMA interrupts, the ISR of the corresponding UART must be registered before using this function.

Parameters

in	*uart*	UART index.
in	*xfer*	Structure containing a pre-allocated write buffer and callback functions. This must not be NULL. Callback pointer must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 586 of file qm_uart.c.

References qm_dma_transfer_t::block_size, qm_uart_transfer_t::data, qm_uart_transfer_t::data_len, qm_dma_transfer_t::destination_address, qm_uart_reg_t::iir_fcr, qm_dma_transfer_set_config(), qm_dma_transfer_start(), qm_uart_reg_t::rbr_thr_dll, and qm_dma_transfer_t::source_address.

4.22.3.5 int qm_uart_dma_write_terminate (const qm_uart_t *uart*)

Terminate the current DMA TX transfer on the UART bus.

This will cause the relevant callbacks to be called.

Parameters

in	*uart*	UART index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 653 of file qm_uart.c.

References qm_dma_transfer_terminate().

4.22.3.6 int qm_uart_get_status (const qm_uart_t *uart*, qm_uart_status_t *const *status*)

Get UART bus status.

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

Parameters

in	*uart*	Which UART to read the status of.
out	*status*	UART specific status. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 249 of file qm_uart.c.

References qm_uart_reg_t::lsr, QM_UART_RX_BUSY, QM_UART_TX_BUSY, and qm_uart_reg_t::scr.

4.22.3.7 int qm_uart_irq_read (const qm_uart_t *uart,* const qm_uart_transfer_t *const *xfer*)

Interrupt based RX on UART.

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

Parameters

in	*uart*	UART index.
in	*xfer*	Structure containing pre-allocated read buffer and callback functions. The structure must not be NULL and must be kept valid until the transfer is complete.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 385 of file qm_uart.c.

References qm_uart_reg_t::ier_dlh, and qm_uart_reg_t::iir_fcr.

4.22.3.8 int qm_uart_irq_read_terminate (const qm_uart_t *uart*)

Terminate UART IRQ RX transfer.

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

Parameters

in	*uart*	UART index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 425 of file qm_uart.c.

References qm_uart_transfer_t::callback, qm_uart_transfer_t::callback_data, qm_uart_reg_t::ier_dlh, and QM_UART_IDLE.

4.22.3.9 int qm_uart_irq_write (const qm_uart_t *uart*, const qm_uart_transfer_t *const *xfer*)

Interrupt based TX on UART.

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

Parameters

in	*uart*	UART index.
in	*xfer*	Structure containing pre-allocated write buffer and callback functions. The structure must not be NULL and must be kept valid until the transfer is complete.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 364 of file qm_uart.c.

References qm_uart_reg_t::ier_dlh, and qm_uart_reg_t::iir_fcr.

4.22.3.10 int qm_uart_irq_write_terminate (const qm_uart_t *uart*)

Terminate UART IRQ TX transfer.

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

Parameters

in	*uart*	UART index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 408 of file qm_uart.c.

References qm_uart_transfer_t::callback, qm_uart_transfer_t::callback_data, qm_uart_reg_t::ier_dlh, and QM_UART_IDLE.

4.22.3.11 int qm_uart_read (const qm_uart_t *uart*, uint8_t *const *data*, qm_uart_status_t *const *status*)

UART character data read.

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

Parameters

in	*uart*	UART index.
out	*data*	Data to read from UART. This must not be NULL.
out	*status*	UART specific status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 292 of file qm_uart.c.

References qm_uart_reg_t::lsr, and qm_uart_reg_t::rbr_thr_dll.

4.22.3.12 int qm_uart_read_non_block (const qm_uart_t *uart*, uint8_t *const *const *data*)

UART character data read.

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

Parameters

in	*uart*	UART index.
out	*data*	Character read. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 327 of file qm_uart.c.

References qm_uart_reg_t::rbr_thr_dll.

4.22.3.13 int qm_uart_set_config (const qm_uart_t *uart*, const qm_uart_config_t *const *const *cfg*)

Set UART configuration.

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

Parameters

in	*uart*	Which UART module to configure.
in	*cfg*	New configuration for UART. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 215 of file qm_uart.c.

References qm_uart_config_t::baud_divisor, qm_uart_reg_t::dlf, qm_uart_config_t::hw_fc, qm_uart_reg_t::ier_dlh, qm_uart_reg_t::iir_fcr, qm_uart_reg_t::lcr, qm_uart_config_t::line_control, qm_uart_reg_t::mcr, and qm_uart_reg_t::rbr_thr_dll.

4.22.3.14 int qm_uart_write (const qm_uart_t *uart*, const uint8_t *data*)

UART character data write.

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 276 of file qm_uart.c.

References qm_uart_reg_t::lsr, and qm_uart_reg_t::rbr_thr_dll.

4.22.3.15 int qm_uart_write_buffer (const qm_uart_t *uart*, const uint8_t *const *data*, uint32_t *len*)

UART multi-byte data write.

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

Parameters

in	*uart*	UART index.
in	*data*	Data to write to UART. This must not be NULL.
in	*len*	Length of data to write to UART.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 338 of file qm_uart.c.

References qm_uart_reg_t::lsr, and qm_uart_reg_t::rbr_thr_dll.

4.22.3.16 int qm_uart_write_non_block (const qm_uart_t *uart*, const uint8_t *data*)

UART character data write.

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 316 of file qm_uart.c.

References qm_uart_reg_t::rbr_thr_dll.

4.23 Version

Version number functions for API.

Functions

- uint32_t qm_ver_rom (void)
  Get the ROM version number.

4.23.1 Detailed Description

Version number functions for API.

4.23.2 Function Documentation

4.23.2.1 uint32_t qm_ver_rom (void)

Get the ROM version number.

Reads the ROM version information from flash and returns it.

Returns

uint32_t ROM version.

Definition at line 7 of file qm_version.c.

4.24 WDT

Watchdog timer.

Data Structures

- struct qm_wdt_config_t

  QM WDT configuration type.

Enumerations

- enum qm_wdt_mode_t { QM_WDT_MODE_RESET, QM_WDT_MODE_INTERRUPT_RESET }

  WDT Mode type.
- enum qm_wdt_clock_timeout_cycles_t {
  QM_WDT_2_POW_16_CYCLES, QM_WDT_2_POW_17_CYCLES, QM_WDT_2_POW_18_CYCLES, QM_WDT_2_POW_19_CYCLES,
  QM_WDT_2_POW_20_CYCLES, QM_WDT_2_POW_21_CYCLES, QM_WDT_2_POW_22_CYCLES, QM_WDT_2_POW_23_CYCLES,
  QM_WDT_2_POW_24_CYCLES, QM_WDT_2_POW_25_CYCLES, QM_WDT_2_POW_26_CYCLES, QM_WDT_2_POW_27_CYCLES,
  QM_WDT_2_POW_28_CYCLES, QM_WDT_2_POW_29_CYCLES, QM_WDT_2_POW_30_CYCLES, QM_WDT_2_POW_31_CYCLES }

  WDT clock cycles for timeout type.

Functions

- int qm_wdt_start (const qm_wdt_t wdt)

  Start WDT.
- int qm_wdt_set_config (const qm_wdt_t wdt, const qm_wdt_config_t *const cfg)

  Set configuration of WDT module.
- int qm_wdt_reload (const qm_wdt_t wdt)

  Reload the WDT counter.

4.24.1 Detailed Description

Watchdog timer.

4.24.2 Enumeration Type Documentation

4.24.2.1 enum qm_wdt_clock_timeout_cycles_t

WDT clock cycles for timeout type.

This value is a power of 2.

Enumerator

QM_WDT_2_POW_16_CYCLES 16 clock cycles timeout.

QM_WDT_2_POW_17_CYCLES 17 clock cycles timeout.

QM_WDT_2_POW_18_CYCLES 18 clock cycles timeout.

QM_WDT_2_POW_19_CYCLES 19 clock cycles timeout.

QM_WDT_2_POW_20_CYCLES 20 clock cycles timeout.

QM_WDT_2_POW_21_CYCLES 21 clock cycles timeout.

QM_WDT_2_POW_22_CYCLES 22 clock cycles timeout.

QM_WDT_2_POW_23_CYCLES 23 clock cycles timeout.

QM_WDT_2_POW_24_CYCLES 24 clock cycles timeout.

QM_WDT_2_POW_25_CYCLES 25 clock cycles timeout.

QM_WDT_2_POW_26_CYCLES 26 clock cycles timeout.

QM_WDT_2_POW_27_CYCLES 27 clock cycles timeout.

QM_WDT_2_POW_28_CYCLES 28 clock cycles timeout.

QM_WDT_2_POW_29_CYCLES 29 clock cycles timeout.

QM_WDT_2_POW_30_CYCLES 30 clock cycles timeout.

QM_WDT_2_POW_31_CYCLES 31 clock cycles timeout.

Definition at line 49 of file qm_wdt.h.

4.24.2.2 enum qm_wdt_mode_t

WDT Mode type.

Enumerator

QM_WDT_MODE_RESET Watchdog Reset Response Mode. The watchdog will request a SoC Warm Reset on a timeout.

QM_WDT_MODE_INTERRUPT_RESET Watchdog Interrupt Reset Response Mode. The watchdog will generate an interrupt on first timeout. If interrupt has not been cleared by the second timeout the watchdog will then request a SoC Warm Reset.

Definition at line 30 of file qm_wdt.h.

4.24.3 Function Documentation

4.24.3.1 int qm_wdt_reload (const qm_wdt_t *wdt*)

Reload the WDT counter.

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

Parameters

in	*wdt*	WDT index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 64 of file qm_wdt.c.

Referenced by qm_wdt_set_config().

4.24.3.2 int 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. This must not be NULL. If QM_WDT_MODE_INTERRUPT_RESET mode is set, the 'callback' cannot be null.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 45 of file qm_wdt.c.

References qm_wdt_config_t::callback, qm_wdt_config_t::callback_data, qm_wdt_config_t::mode, QM_WDT_MODE_INTERRUPT_RESET, qm_wdt_reload(), and qm_wdt_config_t::timeout.

4.24.3.3 int 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

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 21 of file qm_wdt.c.

References SOCW_EVENT_REGISTER, and SOCW_REG_CCU_PERIPH_CLK_GATE_CTL.

4.25 SOC_WATCH

SoC Watch (Energy Analyzer).

Enumerations

- enum soc_watch_event_t {
  SOCW_EVENT_HALT = 0, SOCW_EVENT_INTERRUPT = 1, SOCW_EVENT_SLEEP = 2, SOCW_EVENT_REGISTER = 3,
  SOCW_EVENT_APP = 4, SOCW_EVENT_MAX = 5 }

  Power profiling events enumeration.
- enum soc_watch_reg_t {
  SOCW_REG_OSC0_CFG1 = 0, SOCW_REG_CCU_LP_CLK_CTL = 1, SOCW_REG_CCU_SYS_CLK_CTL = 2, SOCW_REG_CCU_PERIPH_CLK_GATE_CTL = 3,
  SOCW_REG_CCU_EXT_CLK_CTL = 4, SOCW_REG_CMP_PWR = 5, SOCW_REG_PMUX_PULLUP = 6, SOCW_REG_PMUX_SLEW = 7,
  SOCW_REG_PMUX_IN_EN = 8, SOCW_REG_MAX, SOCW_REG_OSC0_CFG1 = 0, SOCW_REG_CCU_LP_CLK_CTL = 1,
  SOCW_REG_CCU_SYS_CLK_CTL = 2, SOCW_REG_CCU_PERIPH_CLK_GATE_CTL = 3, SOCW_REG_CCU_SS_PERIPH_CLK_GATE_CTL = 4, SOCW_REG_CCU_EXT_CLK_CTL = 4,
  SOCW_REG_CMP_PWR = 5, SOCW_REG_SLP_CFG = 7, SOCW_REG_PMUX_PULLUP0 = 8, SOCW_REG_PMUX_PULLUP1 = 9,
  SOCW_REG_PMUX_PULLUP2 = 10, SOCW_REG_PMUX_PULLUP3 = 11, SOCW_REG_PMUX_SLEW0 = 12, SOCW_REG_PMUX_SLEW1 = 13,
  SOCW_REG_PMUX_SLEW2 = 14, SOCW_REG_PMUX_SLEW3 = 15, SOCW_REG_PMUX_IN_EN0 = 16, SOCW_REG_PMUX_IN_EN1 = 17,
  SOCW_REG_PMUX_IN_EN2 = 18, SOCW_REG_PMUX_IN_EN3 = 19, SOCW_REG_MAX }

  Register ID enumeration.
- enum soc_watch_reg_t {
  SOCW_REG_OSC0_CFG1 = 0, SOCW_REG_CCU_LP_CLK_CTL = 1, SOCW_REG_CCU_SYS_CLK_CTL = 2, SOCW_REG_CCU_PERIPH_CLK_GATE_CTL = 3,
  SOCW_REG_CCU_EXT_CLK_CTL = 4, SOCW_REG_CMP_PWR = 5, SOCW_REG_PMUX_PULLUP = 6, SOCW_REG_PMUX_SLEW = 7,
  SOCW_REG_PMUX_IN_EN = 8, SOCW_REG_MAX, SOCW_REG_OSC0_CFG1 = 0, SOCW_REG_CCU_LP_CLK_CTL = 1,
  SOCW_REG_CCU_SYS_CLK_CTL = 2, SOCW_REG_CCU_PERIPH_CLK_GATE_CTL = 3, SOCW_REG_CCU_SS_PERIPH_CLK_GATE_CTL = 4, SOCW_REG_CCU_EXT_CLK_CTL = 4,
  SOCW_REG_CMP_PWR = 5, SOCW_REG_SLP_CFG = 7, SOCW_REG_PMUX_PULLUP0 = 8, SOCW_REG_PMUX_PULLUP1 = 9,
  SOCW_REG_PMUX_PULLUP2 = 10, SOCW_REG_PMUX_PULLUP3 = 11, SOCW_REG_PMUX_SLEW0 = 12, SOCW_REG_PMUX_SLEW1 = 13,
  SOCW_REG_PMUX_SLEW2 = 14, SOCW_REG_PMUX_SLEW3 = 15, SOCW_REG_PMUX_IN_EN0 = 16, SOCW_REG_PMUX_IN_EN1 = 17,
  SOCW_REG_PMUX_IN_EN2 = 18, SOCW_REG_PMUX_IN_EN3 = 19, SOCW_REG_MAX }

Functions

- void soc_watch_log_event (soc_watch_event_t event_id, uintptr_t ev_data)

  Log a power profile event.
- void soc_watch_log_app_event (soc_watch_event_t event_id, uint8_t ev_subtype, uintptr_t ev_data)

  Log an application event via the power profile logger.

4.25.1 Detailed Description

SoC Watch (Energy Analyzer).

4.25.2 Enumeration Type Documentation

4.25.2.1 enum soc_watch_event_t

Power profiling events enumeration.

In order to maintain binary compatibility, only SOCW_EVENT_MAX should ever be altered: new events should be inserted before SOCW_EVENT_MAX, and SOCW_EVENT_MAX incremented. Add events, do not replace them.

Enumerator

SOCW_EVENT_HALT CPU Halt.

SOCW_EVENT_INTERRUPT CPU interrupt generated.

SOCW_EVENT_SLEEP Sleep mode entered.

SOCW_EVENT_REGISTER SOC register altered.

SOCW_EVENT_APP Application-defined event.

SOCW_EVENT_MAX End of events sentinel.

Definition at line 36 of file soc_watch.h.

4.25.2.2 enum soc_watch_reg_t

Register ID enumeration.

The Register Event stores a register ID enumeration instead of a register address in order to save space. Registers can be added, but they should not be deleted, in order to preserve compatibility with different versions of the post-processor.

Note that most of these names mirror the names used elsewhere in the QMSI code, although these are upper case, while the register pointer names are in lower case. That's one clue for identifying where logging calls should to be added: wherever you see one of the named registers below being written, you should consider that write may need a corresponding SoC Watch logging call.

Enumerator

SOCW_REG_OSC0_CFG1 0x000 OSC0_CFG1 register.

SOCW_REG_CCU_LP_CLK_CTL 0x02C Clock Control register.

SOCW_REG_CCU_SYS_CLK_CTL 0x038 System Clock Control.

SOCW_REG_CCU_PERIPH_CLK_GATE_CTL 0x018 Perip Clock Gate Ctl.

SOCW_REG_CCU_EXT_CLK_CTL 0x024 CCU Ext Clock Gate Ctl.

SOCW_REG_CMP_PWR 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.

SOCW_REG_PMUX_PULLUP 0x900 Pin Mux Pullup.

SOCW_REG_PMUX_SLEW 0x910 Pin Mux Slew.

SOCW_REG_PMUX_IN_EN 0x920 Pin Mux In Enable.

SOCW_REG_MAX Register enum sentinel.

SOCW_REG_OSC0_CFG1 0x000 OSC0_CFG1 register.

SOCW_REG_CCU_LP_CLK_CTL 0x02C Clock Control register.

SOCW_REG_CCU_SYS_CLK_CTL 0x038 System Clock Control.

SOCW_REG_CCU_PERIPH_CLK_GATE_CTL 0x018 Perip Clock Gate Ctl.

SOCW_REG_CCU_SS_PERIPH_CLK_GATE_CTL 0x0028 SS PCL Gate Ctl.

SOCW_REG_CCU_EXT_CLK_CTL 0x024 CCU Ext Clock Gate Ctl.

SOCW_REG_CMP_PWR 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.

SOCW_REG_SLP_CFG 0x550 Sleep Configuration.

SOCW_REG_PMUX_PULLUP0 0x900 Pin Mux Pullup.

SOCW_REG_PMUX_PULLUP1 0x904 Pin Mux Pullup.
SOCW_REG_PMUX_PULLUP2 0x908 Pin Mux Pullup.
SOCW_REG_PMUX_PULLUP3 0x90c Pin Mux Pullup.
SOCW_REG_PMUX_SLEW0 0x910 Pin Mux Slew.
SOCW_REG_PMUX_SLEW1 0x914 Pin Mux Slew.
SOCW_REG_PMUX_SLEW2 0x918 Pin Mux Slew.
SOCW_REG_PMUX_SLEW3 0x91c Pin Mux Slew.
SOCW_REG_PMUX_IN_EN0 0x920 Pin Mux In Enable.
SOCW_REG_PMUX_IN_EN1 0x924 Pin Mux In Enable.
SOCW_REG_PMUX_IN_EN2 0x928 Pin Mux In Enable.
SOCW_REG_PMUX_IN_EN3 0x92c Pin Mux In Enable.
SOCW_REG_MAX Register enum sentinel.

Definition at line 61 of file soc_watch.h.

4.25.2.3 enum soc_watch_reg_t

Enumerator

SOCW_REG_OSC0_CFG1 0x000 OSC0_CFG1 register.
SOCW_REG_CCU_LP_CLK_CTL 0x02C Clock Control register.
SOCW_REG_CCU_SYS_CLK_CTL 0x038 System Clock Control.
SOCW_REG_CCU_PERIPH_CLK_GATE_CTL 0x018 Perip Clock Gate Ctl.
SOCW_REG_CCU_EXT_CLK_CTL 0x024 CCU Ext Clock Gate Ctl.
SOCW_REG_CMP_PWR 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.
SOCW_REG_PMUX_PULLUP 0x900 Pin Mux Pullup.
SOCW_REG_PMUX_SLEW 0x910 Pin Mux Slew.
SOCW_REG_PMUX_IN_EN 0x920 Pin Mux In Enable.
SOCW_REG_MAX Register enum sentinel.
SOCW_REG_OSC0_CFG1 0x000 OSC0_CFG1 register.
SOCW_REG_CCU_LP_CLK_CTL 0x02C Clock Control register.
SOCW_REG_CCU_SYS_CLK_CTL 0x038 System Clock Control.
SOCW_REG_CCU_PERIPH_CLK_GATE_CTL 0x018 Perip Clock Gate Ctl.
SOCW_REG_CCU_SS_PERIPH_CLK_GATE_CTL 0x0028 SS PCL Gate Ctl.
SOCW_REG_CCU_EXT_CLK_CTL 0x024 CCU Ext Clock Gate Ctl.
SOCW_REG_CMP_PWR 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.
SOCW_REG_SLP_CFG 0x550 Sleep Configuration.
SOCW_REG_PMUX_PULLUP0 0x900 Pin Mux Pullup.
SOCW_REG_PMUX_PULLUP1 0x904 Pin Mux Pullup.
SOCW_REG_PMUX_PULLUP2 0x908 Pin Mux Pullup.
SOCW_REG_PMUX_PULLUP3 0x90c Pin Mux Pullup.
SOCW_REG_PMUX_SLEW0 0x910 Pin Mux Slew.
SOCW_REG_PMUX_SLEW1 0x914 Pin Mux Slew.
SOCW_REG_PMUX_SLEW2 0x918 Pin Mux Slew.
SOCW_REG_PMUX_SLEW3 0x91c Pin Mux Slew.
SOCW_REG_PMUX_IN_EN0 0x920 Pin Mux In Enable.
SOCW_REG_PMUX_IN_EN1 0x924 Pin Mux In Enable.
SOCW_REG_PMUX_IN_EN2 0x928 Pin Mux In Enable.
SOCW_REG_PMUX_IN_EN3 0x92c Pin Mux In Enable.
SOCW_REG_MAX Register enum sentinel.

Definition at line 78 of file soc_watch.h.

4.25.3 Function Documentation

4.25.3.1 void soc_watch_log_app_event (soc_watch_event_t *event_id,* uint8_t *ev_subtype,* uintptr_t *ev_data*)

Log an application event via the power profile logger.

This allows applications layered on top of QMSI to log their own events. The subtype identifies the type of data for the user, and 'data' is the actual information being logged.

Parameters

in	*event_id*	The Event ID of the profile event.
in	*ev_subtype*	A 1-byte user-defined event_id.
in	*ev_data*	A parameter to the event ID (if the event needs one).

Returns

Nothing.

Definition at line 227 of file soc_watch.c.

References qm_irq_disable(), qm_irq_enable(), SOCW_EVENT_HALT, SOCW_EVENT_INTERRUPT, SOCW_EVENT_MAX, and SOCW_EVENT_SLEEP.

Referenced by soc_watch_log_event().

4.25.3.2 void soc_watch_log_event (soc_watch_event_t *event_id,* uintptr_t *ev_data*)

Log a power profile event.

Log an event related to power management. This should be things like halts, or register reads which cause us to go to low power states, or register reads that affect the clock rate, or other clock gating.

Parameters

in	*event_id*	The Event ID of the profile event.
in	*ev_data*	A parameter to the event ID (if the event needs one).

Definition at line 217 of file soc_watch.c.

References soc_watch_log_app_event().

4.26 SS ADC

Analog to Digital Converter (ADC) for the Sensor Subsystem.

Data Structures

- struct qm_ss_adc_config_t

  SS ADC configuration type.
- struct qm_ss_adc_xfer_t

  SS ADC transfer type.

Typedefs

- typedef uint16_t qm_ss_adc_sample_t

  SS ADC sample size type.
- typedef uint8_t qm_ss_adc_calibration_t

  SS ADC calibration type.

Enumerations

- enum qm_ss_adc_status_t { QM_SS_ADC_IDLE = 0x0, QM_SS_ADC_COMPLETE = 0x1, QM_SS_ADC_OVERFLOW = 0x2, QM_SS_ADC_UNDERFLOW = 0x4, QM_SS_ADC_SEQERROR = 0x8 }

  SS ADC status.
- enum qm_ss_adc_resolution_t { QM_SS_ADC_RES_6_BITS = 0x5, QM_SS_ADC_RES_8_BITS = 0x7, QM_SS_ADC_RES_10_BITS = 0x9, QM_SS_ADC_RES_12_BITS = 0xB }

  SS ADC resolution type.
- enum qm_ss_adc_mode_t { QM_SS_ADC_MODE_DEEP_PWR_DOWN, QM_SS_ADC_MODE_PWR_DOWN, QM_SS_ADC_MODE_STDBY, QM_SS_ADC_MODE_NORM_CAL, QM_SS_ADC_MODE_NORM_NO_CAL }

  SS ADC operating mode type.
- enum qm_ss_adc_channel_t { QM_SS_ADC_CH_0, QM_SS_ADC_CH_1, QM_SS_ADC_CH_2, QM_SS_ADC_CH_3, QM_SS_ADC_CH_4, QM_SS_ADC_CH_5, QM_SS_ADC_CH_6, QM_SS_ADC_CH_7, QM_SS_ADC_CH_8, QM_SS_ADC_CH_9, QM_SS_ADC_CH_10, QM_SS_ADC_CH_11, QM_SS_ADC_CH_12, QM_SS_ADC_CH_13, QM_SS_ADC_CH_14, QM_SS_ADC_CH_15, QM_SS_ADC_CH_16, QM_SS_ADC_CH_17, QM_SS_ADC_CH_18 }

  SS ADC channels type.
- enum qm_ss_adc_cb_source_t { QM_SS_ADC_TRANSFER, QM_SS_ADC_MODE_CHANGED, QM_SS_ADC_CAL_COMPLETE }

  SS ADC interrupt callback source.

Functions

- int qm_ss_adc_set_mode (const qm_ss_adc_t adc, const qm_ss_adc_mode_t mode)

  Switch operating mode of SS ADC.
- int qm_ss_adc_irq_set_mode (const qm_ss_adc_t adc, const qm_ss_adc_mode_t mode, void(*callback)(void *data, int error, qm_ss_adc_status_t status, qm_ss_adc_cb_source_t source), void *callback_data)

  Switch operating mode of SS ADC.
- int qm_ss_adc_calibrate (const qm_ss_adc_t adc)

Calibrate the SS ADC.

- int qm_ss_adc_irq_calibrate (const qm_ss_adc_t adc, void(∗callback)(void ∗data, int error, qm_ss_adc_status_t status, qm_ss_adc_cb_source_t source), void ∗callback_data)

  Calibrate the SS ADC.
- int qm_ss_adc_set_calibration (const qm_ss_adc_t adc, const qm_ss_adc_calibration_t cal)

  Set SS ADC calibration data.
- int qm_ss_adc_get_calibration (const qm_ss_adc_t adc, qm_ss_adc_calibration_t ∗const cal)

  Get the current calibration data for an SS ADC.
- int qm_ss_adc_set_config (const qm_ss_adc_t adc, const qm_ss_adc_config_t ∗const cfg)

  Set SS ADC configuration.
- int qm_ss_adc_convert (const qm_ss_adc_t adc, qm_ss_adc_xfer_t ∗const xfer, qm_ss_adc_status_t ∗const status)

  Synchronously read values from the ADC.
- int qm_ss_adc_irq_convert (const qm_ss_adc_t adc, qm_ss_adc_xfer_t ∗const xfer)

  Asynchronously read values from the SS ADC.

4.26.1 Detailed Description

Analog to Digital Converter (ADC) for the Sensor Subsystem.

4.26.2 Enumeration Type Documentation

4.26.2.1 enum qm_ss_adc_cb_source_t

SS ADC interrupt callback source.

Enumerator

QM_SS_ADC_TRANSFER Transfer complete or error callback.

QM_SS_ADC_MODE_CHANGED Mode change complete callback.

QM_SS_ADC_CAL_COMPLETE Calibration complete callback.

Definition at line 89 of file qm_ss_adc.h.

4.26.2.2 enum qm_ss_adc_channel_t

SS ADC channels type.

Enumerator

QM_SS_ADC_CH_0 ADC Channel 0.

QM_SS_ADC_CH_1 ADC Channel 1.

QM_SS_ADC_CH_2 ADC Channel 2.

QM_SS_ADC_CH_3 ADC Channel 3.

QM_SS_ADC_CH_4 ADC Channel 4.

QM_SS_ADC_CH_5 ADC Channel 5.

QM_SS_ADC_CH_6 ADC Channel 6.

QM_SS_ADC_CH_7 ADC Channel 7.

QM_SS_ADC_CH_8 ADC Channel 8.

QM_SS_ADC_CH_9 ADC Channel 9.

QM_SS_ADC_CH_10 ADC Channel 10.

QM_SS_ADC_CH_11 ADC Channel 11.

QM_SS_ADC_CH_12 ADC Channel 12.
QM_SS_ADC_CH_13 ADC Channel 13.
QM_SS_ADC_CH_14 ADC Channel 14.
QM_SS_ADC_CH_15 ADC Channel 15.
QM_SS_ADC_CH_16 ADC Channel 16.
QM_SS_ADC_CH_17 ADC Channel 17.
QM_SS_ADC_CH_18 ADC Channel 18.

Definition at line 64 of file qm_ss_adc.h.

4.26.2.3 enum qm_ss_adc_mode_t

SS ADC operating mode type.

Enumerator

QM_SS_ADC_MODE_DEEP_PWR_DOWN Deep power down mode.
QM_SS_ADC_MODE_PWR_DOWN Power down mode.
QM_SS_ADC_MODE_STDBY Standby mode.
QM_SS_ADC_MODE_NORM_CAL Normal mode, with calibration.
QM_SS_ADC_MODE_NORM_NO_CAL Normal mode, no calibration.

Definition at line 53 of file qm_ss_adc.h.

4.26.2.4 enum qm_ss_adc_resolution_t

SS ADC resolution type.

Enumerator

QM_SS_ADC_RES_6_BITS 6-bit mode.
QM_SS_ADC_RES_8_BITS 8-bit mode.
QM_SS_ADC_RES_10_BITS 10-bit mode.
QM_SS_ADC_RES_12_BITS 12-bit mode.

Definition at line 43 of file qm_ss_adc.h.

4.26.2.5 enum qm_ss_adc_status_t

SS ADC status.

Enumerator

QM_SS_ADC_IDLE ADC idle.
QM_SS_ADC_COMPLETE ADC data available.
QM_SS_ADC_OVERFLOW ADC overflow error.
QM_SS_ADC_UNDERFLOW ADC underflow error.
QM_SS_ADC_SEQERROR ADC sequencer error.

Definition at line 32 of file qm_ss_adc.h.

4.26.3 Function Documentation

4.26.3.1 int qm_ss_adc_calibrate (const qm_ss_adc_t *adc*)

Calibrate the SS ADC.

It is necessary to calibrate if it is intended to use Normal Mode With Calibration. The calibration must be performed if the ADC is used for the first time or has been in deep power down mode. This call is blocking.

Parameters

in	*adc*	Which ADC to calibrate.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.26.3.2 int qm_ss_adc_convert (const qm_ss_adc_t *adc*, qm_ss_adc_xfer_t *const *xfer*, qm_ss_adc_status_t *const *status*)

Synchronously read values from the ADC.

This blocking call can read 1-32 ADC values into the array provided.

Parameters

in	*adc*	Which ADC to read.
in, out	*xfer*	Channel and sample info. This must not be NULL.
out	*status*	Get status of the adc device.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 553 of file qm_ss_adc.c.

References qm_ss_adc_xfer_t::ch, qm_ss_adc_xfer_t::ch_len, QM_SS_ADC_CTRL, QM_SS_ADC_INTSTAT, QM_SS_ADC_SAMPLE, QM_SS_ADC_SET, qm_ss_adc_xfer_t::samples, and qm_ss_adc_xfer_t::samples_len.

4.26.3.3 int qm_ss_adc_get_calibration (const qm_ss_adc_t *adc*, qm_ss_adc_calibration_t *const *cal*)

Get the current calibration data for an SS ADC.

Parameters

in	*adc*	Which ADC to get calibration for.
out	*cal*	Calibration data. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.26.3.4 int qm_ss_adc_irq_calibrate (const qm_ss_adc_t *adc*, void(*)(void *data, int error, qm_ss_adc_status_t status, qm_ss_adc_cb_source_t source) *callback*, void * *callback_data*)

Calibrate the SS ADC.

It is necessary to calibrate if it is intended to use Normal Mode With Calibration. The calibration must be performed if the ADC is used for the first time or has been in deep power down mode. This call is non-blocking and will call the user callback on completion.

Parameters

in	*adc*	Which ADC to calibrate.
in	*callback*	Callback called on completion.
in	*callback_data*	The callback user data.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 473 of file qm_ss_adc.c.

References QM_SS_IO_CREG_MST0_CTRL.

4.26.3.5 int qm_ss_adc_irq_convert (const qm_ss_adc_t *adc*, qm_ss_adc_xfer_t *const *xfer*)

Asynchronously read values from the SS ADC.

This is a non-blocking call and will call the user provided callback after the requested number of samples have been converted.

Parameters

in	*adc*	Which ADC to read.
in,out	*xfer*	Channel sample and callback info. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 627 of file qm_ss_adc.c.

References qm_ss_adc_xfer_t::ch, qm_ss_adc_xfer_t::ch_len, QM_SS_ADC_CTRL, QM_SS_ADC_SET, qm_ss_adc_xfer_t::samples, and qm_ss_adc_xfer_t::samples_len.

4.26.3.6 int qm_ss_adc_irq_set_mode (const qm_ss_adc_t *adc*, const qm_ss_adc_mode_t *mode*, void(*)(void *data, int error, qm_ss_adc_status_t status, qm_ss_adc_cb_source_t source) *callback*, void * *callback_data*)

Switch operating mode of SS ADC.

This call is non-blocking and will call the user callback on completion. An interrupt will not be generated if the user requests the same mode the ADC is currently in (default mode on boot is deep power down).

Parameters

in	*adc*	Which ADC to enable.

in	*mode*	ADC operating mode.
in	*callback*	Callback called on completion.
in	*callback_data*	The callback user data.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 403 of file qm_ss_adc.c.

References QM_SS_ADC_MODE_NORM_NO_CAL, and QM_SS_IO_CREG_MST0_CTRL.

4.26.3.7 int qm_ss_adc_set_calibration (const qm_ss_adc_t *adc*, const qm_ss_adc_calibration_t *cal*)

Set SS ADC calibration data.

Parameters

in	*adc*	Which ADC to set calibration for.
in	*cal*	Calibration data.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.26.3.8 int qm_ss_adc_set_config (const qm_ss_adc_t *adc*, const qm_ss_adc_config_t *const *cfg*)

Set SS ADC configuration.

This sets the sample window and resolution.

Parameters

in	*adc*	Which ADC to configure.
in	*cfg*	ADC configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 335 of file qm_ss_adc.c.

References QM_SS_ADC_RES_12_BITS, QM_SS_ADC_SET, qm_ss_adc_config_t::resolution, and qm_ss_adc_config_t::window.

4.26.3.9 int qm_ss_adc_set_mode (const qm_ss_adc_t *adc*, const qm_ss_adc_mode_t *mode*)

Switch operating mode of SS ADC.

This call is blocking.

Parameters

in	*adc*	Which ADC to enable.
in	*mode*	ADC operating mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 361 of file qm_ss_adc.c.

References QM_SS_ADC_MODE_NORM_CAL, QM_SS_ADC_MODE_NORM_NO_CAL, QM_SS_IO_CREG_MST0_CTRL, and QM_SS_IO_CREG_SLV0_OBSR.

4.27 SS GPIO

General Purpose IO for Sensor Subsystem.

Data Structures

- struct qm_ss_gpio_port_config_t
  GPIO port configuration type.

Enumerations

- enum qm_ss_gpio_state_t { QM_SS_GPIO_LOW, QM_SS_GPIO_HIGH }
  GPIO SS pin states.

Functions

- int qm_ss_gpio_set_config (const qm_ss_gpio_t gpio, const qm_ss_gpio_port_config_t *const cfg)
  Set SS GPIO port configuration.
- int qm_ss_gpio_read_pin (const qm_ss_gpio_t gpio, const uint8_t pin, qm_ss_gpio_state_t *const state)
  Read the current value of a single pin on a given SS GPIO port.
- int qm_ss_gpio_set_pin (const qm_ss_gpio_t gpio, const uint8_t pin)
  Set a single pin on a given SS GPIO port.
- int qm_ss_gpio_clear_pin (const qm_ss_gpio_t gpio, const uint8_t pin)
  Clear a single pin on a given SS GPIO port.
- int qm_ss_gpio_set_pin_state (const qm_ss_gpio_t gpio, const uint8_t pin, const qm_ss_gpio_state_t state)
  Set or clear a single SS GPIO pin using a state variable.
- int qm_ss_gpio_read_port (const qm_ss_gpio_t gpio, uint32_t *const port)
  Get SS GPIO port values.
- int qm_ss_gpio_write_port (const qm_ss_gpio_t gpio, const uint32_t val)
  Get SS GPIO port values.

4.27.1 Detailed Description

General Purpose IO for Sensor Subsystem.

4.27.2 Enumeration Type Documentation

4.27.2.1 enum qm_ss_gpio_state_t

GPIO SS pin states.

Enumerator

QM_SS_GPIO_LOW Pin level high.

QM_SS_GPIO_HIGH Pin level low.

Definition at line 21 of file qm_ss_gpio.h.

4.27.3 Function Documentation

4.27.3.1 int qm_ss_gpio_clear_pin (const qm_ss_gpio_t *gpio*, const uint8_t *pin*)

Clear a single pin on a given SS GPIO port.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 89 of file qm_ss_gpio.c.

4.27.3.2 int qm_ss_gpio_read_pin (const qm_ss_gpio_t *gpio*, const uint8_t *pin*, qm_ss_gpio_state_t *const *const *state*)

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

Parameters

in	*gpio*	SS GPIO port index.
in	*pin*	Pin of SS GPIO port to read.
out	*state*	QM_GPIO_LOW for low or QM_GPIO_HIGH for high. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 62 of file qm_ss_gpio.c.

4.27.3.3 int qm_ss_gpio_read_port (const qm_ss_gpio_t *gpio*, uint32_t *const *const *port*)

Get SS GPIO port values.

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

Parameters

in	*gpio*	SS GPIO port index.
out	*port*	Value of all pins on GPIO port. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 116 of file qm_ss_gpio.c.

4.27.3.4 int qm_ss_gpio_set_config (const qm_ss_gpio_t *gpio*, const qm_ss_gpio_port_config_t *const *const *cfg*)

Set SS GPIO port configuration.

This includes the direction of the pins, 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 the user defined callback function.

Parameters

in	*gpio*	SS GPIO port index to configure.
in	*cfg*	New configuration for SS GPIO port. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 37 of file qm_ss_gpio.c.

References qm_ss_gpio_port_config_t::callback, qm_ss_gpio_port_config_t::callback_data, qm_ss_gpio_port_config_t::direction, qm_ss_gpio_port_config_t::int_debounce, qm_ss_gpio_port_config_t::int_en, qm_ss_gpio_port_config_t::int_polarity, and qm_ss_gpio_port_config_t::int_type.

4.27.3.5 int qm_ss_gpio_set_pin (const qm_ss_gpio_t *gpio*, const uint8_t *pin*)

Set a single pin on a given SS GPIO port.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 76 of file qm_ss_gpio.c.

4.27.3.6 int qm_ss_gpio_set_pin_state (const qm_ss_gpio_t *gpio*, const uint8_t *pin*, const qm_ss_gpio_state_t *state*)

Set or clear a single SS GPIO pin using a state variable.

Parameters

in	*gpio*	GPIO port index.
in	*pin*	Pin of GPIO port to update.
in	*state*	QM_GPIO_LOW for low or QM_GPIO_HIGH for high.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 102 of file qm_ss_gpio.c.

4.27.3.7 int qm_ss_gpio_write_port (const qm_ss_gpio_t *gpio*, const uint32_t *val*)

Get SS GPIO port values.

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 126 of file qm_ss_gpio.c.

4.28 SS I2C

I2C driver for Sensor Subsystem.

Data Structures

- struct qm_ss_i2c_config_t

  QM SS I2C configuration type.
- struct qm_ss_i2c_transfer_t

  QM SS I2C transfer type.

Enumerations

- enum qm_ss_i2c_addr_t { QM_SS_I2C_7_BIT = 0, QM_SS_I2C_10_BIT }

  QM SS I2C addressing type.
- enum qm_ss_i2c_speed_t { QM_SS_I2C_SPEED_STD = 1, QM_SS_I2C_SPEED_FAST = 2 }

  QM SS I2C Speed Type.
- enum qm_ss_i2c_status_t {
  QM_I2C_IDLE = 0, QM_I2C_TX_ABRT_7B_ADDR_NOACK = BIT(0), QM_I2C_TX_ABRT_TXDATA_NOACK = BIT(3), QM_I2C_TX_ABRT_SBYTE_ACKDET = BIT(7),
  QM_I2C_TX_ABRT_MASTER_DIS = BIT(11), QM_I2C_TX_ARB_LOST = BIT(12), QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO = BIT(13), QM_I2C_TX_ABRT_SLV_ARBLOST = BIT(14),
  QM_I2C_TX_ABRT_SLVRD_INTX = BIT(15), QM_I2C_TX_ABRT_USER_ABRT = BIT(16), QM_I2C_BUSY = BIT(17) }

  QM SS I2C status type.

Functions

- int qm_ss_i2c_set_config (const qm_ss_i2c_t i2c, const qm_ss_i2c_config_t *const cfg)

  Set SS I2C configuration.
- int qm_ss_i2c_set_speed (const qm_ss_i2c_t i2c, const qm_ss_i2c_speed_t speed, const uint16_t lo_cnt, const uint16_t hi_cnt)

  Set I2C speed.
- int qm_ss_i2c_get_status (const qm_ss_i2c_t i2c, qm_ss_i2c_status_t *const status)

  Retrieve SS I2C status.
- int qm_ss_i2c_master_write (const qm_ss_i2c_t i2c, const uint16_t slave_addr, const uint8_t *const data, uint32_t len, const bool stop, qm_ss_i2c_status_t *const status)

  Master write on I2C.
- int qm_ss_i2c_master_read (const qm_ss_i2c_t i2c, const uint16_t slave_addr, uint8_t *const data, uint32_t len, const bool stop, qm_ss_i2c_status_t *const status)

  Master read of I2C.
- int qm_ss_i2c_master_irq_transfer (const qm_ss_i2c_t i2c, const qm_ss_i2c_transfer_t *const xfer, const uint16_t slave_addr)

  Interrupt based master transfer on I2C.
- int qm_ss_i2c_irq_transfer_terminate (const qm_ss_i2c_t i2c)

  Terminate I2C IRQ/DMA transfer.

4.28.1 Detailed Description

I2C driver for Sensor Subsystem.

4.28.2 Enumeration Type Documentation

4.28.2.1 enum qm_ss_i2c_addr_t

QM SS I2C addressing type.

Enumerator

QM_SS_I2C_7_BIT 7-bit mode.

QM_SS_I2C_10_BIT 10-bit mode.

Definition at line 37 of file qm_ss_i2c.h.

4.28.2.2 enum qm_ss_i2c_speed_t

QM SS I2C Speed Type.

Enumerator

QM_SS_I2C_SPEED_STD Standard mode (100 Kbps).

QM_SS_I2C_SPEED_FAST Fast mode (400 Kbps).

Definition at line 45 of file qm_ss_i2c.h.

4.28.2.3 enum qm_ss_i2c_status_t

QM SS I2C status type.

Enumerator

QM_I2C_IDLE Controller idle.

QM_I2C_TX_ABRT_7B_ADDR_NOACK 7-bit address noack.

QM_I2C_TX_ABRT_TXDATA_NOACK Tx data noack.

QM_I2C_TX_ABRT_SBYTE_ACKDET Start ACK.

QM_I2C_TX_ABRT_MASTER_DIS Master disabled.

QM_I2C_TX_ARB_LOST Master lost arbitration.

QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO Slave flush tx FIFO.

QM_I2C_TX_ABRT_SLV_ARBLOST Slave lost bus.

QM_I2C_TX_ABRT_SLVRD_INTX Slave read completion.

QM_I2C_TX_ABRT_USER_ABRT User abort.

QM_I2C_BUSY Controller busy.

Definition at line 53 of file qm_ss_i2c.h.

4.28.3 Function Documentation

4.28.3.1 int qm_ss_i2c_get_status (const qm_ss_i2c_t *i2c*, qm_ss_i2c_status_t *const *status*)

Retrieve SS I2C status.

Parameters

in	*i2c*	Which I2C to read the status of.
out	*status*	Get i2c status. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 408 of file qm_ss_i2c.c.

References QM_I2C_BUSY.

Referenced by qm_ss_i2c_master_read(), and qm_ss_i2c_master_write().

4.28.3.2 int qm_ss_i2c_irq_transfer_terminate (const qm_ss_i2c_t *i2c*)

Terminate I2C IRQ/DMA transfer.

Terminate the current IRQ transfer on the SS I2C bus. This will cause the user callback to be called with status QM_I2C_TX_ABRT_USER_ABRT.

Parameters

in	*i2c*	I2C register block pointer.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 680 of file qm_ss_i2c.c.

4.28.3.3 int qm_ss_i2c_master_irq_transfer (const qm_ss_i2c_t *i2c*, const qm_ss_i2c_transfer_t *const *xfer*, const uint16_t *slave_addr*)

Interrupt based master transfer on I2C.

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

Parameters

in	*i2c*	Which I2C to transfer from.
in	*xfer*	Transfer structure includes write / read data and length, user callback function and the callback context. The structure must not be NULL and must be kept valid until the transfer is complete.
in	*slave_addr*	Address of slave to transfer data with.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 589 of file qm_ss_i2c.c.

References qm_ss_i2c_transfer_t::rx_len.

4.28.3.4 int qm_ss_i2c_master_read (const qm_ss_i2c_t *i2c*, const uint16_t *slave_addr*, uint8_t *const *data*, uint32_t *len*, const bool *stop*, qm_ss_i2c_status_t *const *status*)

Master read of I2C.

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

Parameters

in	*i2c*	Which I2C to read from.
in	*slave_addr*	Address of slave device to read from.
out	*data*	Pre-allocated buffer to populate with data. This must not be NULL.
in	*len*	length of data to read from slave.
in	*stop*	Generate a STOP condition at the end of tx.
out	*status*	Get i2c status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 507 of file qm_ss_i2c.c.

References qm_ss_i2c_get_status().

4.28.3.5 int qm_ss_i2c_master_write (const qm_ss_i2c_t *i2c*, const uint16_t *slave_addr*, const uint8_t *const *data*, uint32_t *len*, const bool *stop*, qm_ss_i2c_status_t *const *status*)

Master write on I2C.

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

Parameters

in	*i2c*	Which I2C to write to.
in	*slave_addr*	Address of slave to write to.
in	*data*	Pre-allocated buffer of data to write. This must not be NULL.
in	*len*	length of data to write.
in	*stop*	Generate a STOP condition at the end of tx.
out	*status*	Get i2c status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 429 of file qm_ss_i2c.c.

References qm_ss_i2c_get_status().

4.28.3.6 int qm_ss_i2c_set_config (const qm_ss_i2c_t *i2c*, const qm_ss_i2c_config_t *const *cfg*)

Set SS I2C configuration.

Parameters

in	*i2c*	Which I2C to set the configuration of.
in	*cfg*	I2C configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 284 of file qm_ss_i2c.c.

References qm_ss_i2c_config_t::address_mode, QM_SS_I2C_SPEED_FAST, QM_SS_I2C_SPEED_STD, and qm_ss_i2c_config_t::speed.

4.28.3.7 int qm_ss_i2c_set_speed (const qm_ss_i2c_t *i2c*, const qm_ss_i2c_speed_t *speed*, const uint16_t *lo_cnt*, const uint16_t *hi_cnt*)

Set I2C speed.

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

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 370 of file qm_ss_i2c.c.

References QM_SS_I2C_SPEED_FAST, and QM_SS_I2C_SPEED_STD.

4.29 SS Interrupt

Interrupt driver for Sensor Subsystem.

Typedefs

- typedef void(∗ qm_ss_isr_t)(struct interrupt_frame ∗frame)
  Interrupt service routine type.

Functions

- void qm_ss_irq_enable (void)
  Enable interrupt delivery for the Sensor Subsystem.
- void qm_ss_irq_disable (void)
  Disable interrupt delivery for the Sensor Subsystem.
- void qm_ss_irq_unmask (uint32_t irq)
  Unmask a given interrupt line.
- void qm_ss_irq_mask (uint32_t irq)
  Mask a given interrupt line.
- void qm_ss_irq_request (uint32_t irq, qm_ss_isr_t isr)
  Request a given IRQ and register ISR to interrupt vector.
- void qm_ss_int_vector_request (uint32_t vector, qm_ss_isr_t isr)
  Register an Interrupt Service Routine to a given interrupt vector.

4.29.1 Detailed Description

Interrupt driver for Sensor Subsystem.

4.29.2 Function Documentation

4.29.2.1 void qm_ss_int_vector_request (uint32_t *vector,* qm_ss_isr_t *isr*)

Register an Interrupt Service Routine to a given interrupt vector.

Parameters

in	*vector*	Interrupt Vector number.
in	*isr*	ISR to register to given vector. Must be a valid Sensor Subsystem ISR.

Definition at line 42 of file qm_ss_interrupt.c.

Referenced by qm_ss_irq_request().

4.29.2.2 void qm_ss_irq_mask (uint32_t *irq*)

Mask a given interrupt line.

Parameters

in	*irq*	Which IRQ to mask.

Definition at line 30 of file qm_ss_interrupt.c.

Referenced by qm_irq_mask(), and qm_ss_irq_request().

4.29.2.3 void qm_ss_irq_request (uint32_t *irq,* qm_ss_isr_t *isr*)

Request a given IRQ and register ISR to interrupt vector.

Parameters

in	*irq*	IRQ number.
in	*isr*	ISR to register to given IRQ.

Definition at line 57 of file qm_ss_interrupt.c.

References qm_ss_int_vector_request(), qm_ss_irq_mask(), and qm_ss_irq_unmask().

4.29.2.4 void qm_ss_irq_unmask (uint32_t *irq*)

Unmask a given interrupt line.

Parameters

in	*irq*	Which IRQ to unmask.

Definition at line 36 of file qm_ss_interrupt.c.

Referenced by qm_irq_unmask(), and qm_ss_irq_request().

4.30 SS ISR

Sensor Subsystem Interrupt Service Routines.

Functions

- QM_ISR_DECLARE (qm_ss_adc_0_isr)
  ISR for ADC interrupt.
- QM_ISR_DECLARE (qm_ss_adc_0_err_isr)
  ISR for ADC error interrupt.
- QM_ISR_DECLARE (qm_ss_adc_0_cal_isr)
  ISR for SS ADC 0 calibration interrupt.
- QM_ISR_DECLARE (qm_ss_adc_0_pwr_isr)
  ISR for SS ADC 0 mode change interrupt.
- QM_ISR_DECLARE (qm_ss_gpio_isr_0)
  ISR for GPIO 0 error interrupt.
- QM_ISR_DECLARE (qm_ss_gpio_isr_1)
  ISR for GPIO 1 error interrupt.
- QM_ISR_DECLARE (qm_ss_i2c_isr_0)
  ISR for I2C 0 error interrupt.
- QM_ISR_DECLARE (qm_ss_i2c_isr_1)
  ISR for I2C 1 error interrupt.
- QM_ISR_DECLARE (qm_ss_spi_0_err_isr)
  ISR for SPI 0 error interrupt.
- QM_ISR_DECLARE (qm_ss_spi_1_err_isr)
  ISR for SPI 1 error interrupt.
- QM_ISR_DECLARE (qm_ss_spi_0_tx_isr)
  ISR for SPI 0 TX data requested interrupt.
- QM_ISR_DECLARE (qm_ss_spi_1_tx_isr)
  ISR for SPI 1 TX data requested interrupt.
- QM_ISR_DECLARE (qm_ss_spi_0_rx_isr)
  ISR for SPI 0 RX data available interrupt.
- QM_ISR_DECLARE (qm_ss_spi_1_rx_isr)
  ISR for SPI 1 data available interrupt.
- QM_ISR_DECLARE (qm_ss_timer_isr_0)
  ISR for SS Timer 0 interrupt.

4.30.1 Detailed Description

Sensor Subsystem Interrupt Service Routines.

4.30.2 Function Documentation

4.30.2.1 QM_ISR_DECLARE (qm_ss_adc_0_isr)

ISR for ADC interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_ADC_IRQ, qm_ss_adc_0_isr);
```

if IRQ based conversions are used.

Definition at line 197 of file qm_ss_adc.c.

References QM_SS_ADC_0.

4.30.2.2 QM_ISR_DECLARE (qm_ss_adc_0_err_isr)

ISR for ADC error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_ADC_ERR, qm_ss_adc_0_err_isr);
```

if IRQ based conversions are used.

Definition at line 203 of file qm_ss_adc.c.

References QM_SS_ADC_0.

4.30.2.3 QM_ISR_DECLARE (qm_ss_adc_0_cal_isr)

ISR for SS ADC 0 calibration interrupt.

This function needs to be registered with

```
qm_irq_request(QM_SS_IRQ_ADC_CAL, qm_ss_adc_0_cal_isr);
```

if IRQ based calibration is used.

Definition at line 215 of file qm_ss_adc.c.

References QM_SS_ADC_0.

4.30.2.4 QM_ISR_DECLARE (qm_ss_adc_0_pwr_isr)

ISR for SS ADC 0 mode change interrupt.

This function needs to be registered with

```
qm_irq_request(QM_SS_IRQ_ADC_PWR, qm_ss_adc_0_pwr_isr);
```

if IRQ based mode change is used.

Definition at line 209 of file qm_ss_adc.c.

References QM_SS_ADC_0.

4.30.2.5 QM_ISR_DECLARE (qm_ss_gpio_isr_0)

ISR for GPIO 0 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_GPIO_INTR_0, qm_ss_gpio_isr_0);
```

if IRQ based transfers are used.

Definition at line 27 of file qm_ss_gpio.c.

4.30.2.6 QM_ISR_DECLARE (qm_ss_gpio_isr_1)

ISR for GPIO 1 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_GPIO_INTR_1, qm_ss_gpio_isr_1);
```

if IRQ based transfers are used.

Definition at line 32 of file qm_ss_gpio.c.

4.30.2.7 QM_ISR_DECLARE (qm_ss_i2c_isr_0)

ISR for I2C 0 error interrupt.

This function needs to be registered with

```
 qm_ss_irq_request(QM_SS_IRQ_I2C_0_ERR, qm_ss_i2c_isr_0);
@code qm_ss_irq_request(QM_SS_IRQ_I2C_0_RX_AVAIL, qm_ss_i2c_isr_0);
@code qm_ss_irq_request(QM_SS_IRQ_I2C_0_TX_REQ, qm_ss_i2c_isr_0);
@code qm_ss_irq_request(QM_SS_IRQ_I2C_0_STOP_DET, qm_ss_i2c_isr_0);
```

if IRQ based transfers are used.

Definition at line 259 of file qm_ss_i2c.c.

4.30.2.8 QM_ISR_DECLARE (qm_ss_i2c_isr_1)

ISR for I2C 1 error interrupt.

This function needs to be registered with

```
 qm_ss_irq_request(QM_SS_IRQ_I2C_1_ERR, qm_ss_i2c_isr_1);
@code qm_ss_irq_request(QM_SS_IRQ_I2C_1_RX_AVAIL, qm_ss_i2c_isr_1);
@code qm_ss_irq_request(QM_SS_IRQ_I2C_1_TX_REQ, qm_ss_i2c_isr_1);
@code qm_ss_irq_request(QM_SS_IRQ_I2C_1_STOP_DET, qm_ss_i2c_isr_1);
```

if IRQ based transfers are used.

Definition at line 264 of file qm_ss_i2c.c.

4.30.2.9 QM_ISR_DECLARE (qm_ss_spi_0_err_isr)

ISR for SPI 0 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_0_ERR_INT, qm_ss_spi_0_err_isr);
```

if IRQ based transfers are used.

Definition at line 384 of file qm_ss_spi.c.

References QM_SS_SPI_0.

4.30.2.10 QM_ISR_DECLARE (qm_ss_spi_1_err_isr)

ISR for SPI 1 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_1_ERR_INT, qm_ss_spi_1_err_isr);
```

if IRQ based transfers are used.

Definition at line 388 of file qm_ss_spi.c.

References QM_SS_SPI_1.

4.30.2.11 QM_ISR_DECLARE (qm_ss_spi_0_tx_isr)

ISR for SPI 0 TX data requested interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_0_TX_REQ, qm_ss_spi_0_tx_isr);
```

if IRQ based transfers are used.

Definition at line 400 of file qm_ss_spi.c.

References QM_SS_SPI_0.

4.30.2.12 QM_ISR_DECLARE (qm_ss_spi_1_tx_isr)

ISR for SPI 1 TX data requested interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_1_TX_REQ, qm_ss_spi_1_tx_isr);
```

if IRQ based transfers are used.

Definition at line 404 of file qm_ss_spi.c.

References QM_SS_SPI_1.

4.30.2.13 QM_ISR_DECLARE (qm_ss_spi_0_rx_isr)

ISR for SPI 0 RX data available interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_0_RX_AVAIL, qm_ss_spi_0_rx_isr);
```

if IRQ based transfers are used.

Definition at line 392 of file qm_ss_spi.c.

References QM_SS_SPI_0.

4.30.2.14 QM_ISR_DECLARE (qm_ss_spi_1_rx_isr)

ISR for SPI 1 data available interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_1_RX_AVAIL, qm_ss_spi_1_rx_isr);
```

if IRQ based transfers are used.

Definition at line 396 of file qm_ss_spi.c.

References QM_SS_SPI_1.

4.30.2.15 QM_ISR_DECLARE (qm_ss_timer_isr_0)

ISR for SS Timer 0 interrupt.

This function needs to be registered with

```
qm_ss_int_vector_request(QM_SS_INT_TIMER_0, qm_ss_timer_isr_0);
```

Definition at line 24 of file qm_ss_timer.c.

4.31 SS SPI

SPI peripheral driver for Sensor Subsystem.

Data Structures

- struct qm_ss_spi_config_t

  SPI configuration type.
- struct qm_ss_spi_async_transfer_t

  SPI IRQ transfer type.
- struct qm_ss_spi_transfer_t

  SPI transfer type.

Enumerations

- enum qm_ss_spi_frame_size_t { QM_SS_SPI_FRAME_SIZE_4_BIT = 3, QM_SS_SPI_FRAME_SIZE_5_BIT, QM_SS_SPI_FRAME_SIZE_6_BIT, QM_SS_SPI_FRAME_SIZE_7_BIT, QM_SS_SPI_FRAME_SIZE_8_BIT, QM_SS_SPI_FRAME_SIZE_9_BIT, QM_SS_SPI_FRAME_SIZE_10_BIT, QM_SS_SPI_FRAME_SIZE_11_BIT, QM_SS_SPI_FRAME_SIZE_12_BIT, QM_SS_SPI_FRAME_SIZE_13_BIT, QM_SS_SPI_FRAME_SIZE_14_BIT, QM_SS_SPI_FRAME_SIZE_15_BIT, QM_SS_SPI_FRAME_SIZE_16_BIT }

  QM SPI frame size type.
- enum qm_ss_spi_tmode_t { QM_SS_SPI_TMOD_TX_RX, QM_SS_SPI_TMOD_TX, QM_SS_SPI_TMOD_RX, QM_SS_SPI_TMOD_EEPROM_READ }

  SPI transfer mode type.
- enum qm_ss_spi_bmode_t { QM_SS_SPI_BMODE_0, QM_SS_SPI_BMODE_1, QM_SS_SPI_BMODE_2, QM_SS_SPI_BMODE_3 }

  SPI bus mode type.
- enum qm_ss_spi_slave_select_t { QM_SS_SPI_SS_DISABLED = 0, QM_SS_SPI_SS_0 = BIT(0), QM_SS_SPI_SS_1 = BIT(1), QM_SS_SPI_SS_2 = BIT(2), QM_SS_SPI_SS_3 = BIT(3) }

  SPI slave select type.
- enum qm_ss_spi_status_t { QM_SS_SPI_IDLE, QM_SS_SPI_BUSY, QM_SS_SPI_RX_OVERFLOW }

  SPI status.

Functions

- int qm_ss_spi_set_config (const qm_ss_spi_t spi, const qm_ss_spi_config_t *const cfg)

  Set SPI configuration.
- int qm_ss_spi_slave_select (const qm_ss_spi_t spi, const qm_ss_spi_slave_select_t ss)

  Set Slave Select lines.
- int qm_ss_spi_get_status (const qm_ss_spi_t spi, qm_ss_spi_status_t *const status)

  Get SPI bus status.
- int qm_ss_spi_transfer (const qm_ss_spi_t spi, const qm_ss_spi_transfer_t *const xfer, qm_ss_spi_status_t *const status)

  Perform a blocking SPI transfer.
- int qm_ss_spi_irq_transfer (const qm_ss_spi_t spi, const qm_ss_spi_async_transfer_t *const xfer)

  Initiate a interrupt based SPI transfer.
- int qm_ss_spi_transfer_terminate (const qm_ss_spi_t spi)

  Terminate SPI IRQ transfer.

4.31.1 Detailed Description

SPI peripheral driver for Sensor Subsystem.

4.31.2 Enumeration Type Documentation

4.31.2.1 enum qm_ss_spi_bmode_t

SPI bus mode type.

Enumerator

QM_SS_SPI_BMODE_0 Clock Polarity = 0, Clock Phase = 0.
QM_SS_SPI_BMODE_1 Clock Polarity = 0, Clock Phase = 1.
QM_SS_SPI_BMODE_2 Clock Polarity = 1, Clock Phase = 0.
QM_SS_SPI_BMODE_3 Clock Polarity = 1, Clock Phase = 1.

Definition at line 78 of file qm_ss_spi.h.

4.31.2.2 enum qm_ss_spi_frame_size_t

QM SPI frame size type.

Enumerator

QM_SS_SPI_FRAME_SIZE_4_BIT 4 bit frame.
QM_SS_SPI_FRAME_SIZE_5_BIT 5 bit frame.
QM_SS_SPI_FRAME_SIZE_6_BIT 6 bit frame.
QM_SS_SPI_FRAME_SIZE_7_BIT 7 bit frame.
QM_SS_SPI_FRAME_SIZE_8_BIT 8 bit frame.
QM_SS_SPI_FRAME_SIZE_9_BIT 9 bit frame.
QM_SS_SPI_FRAME_SIZE_10_BIT 10 bit frame.
QM_SS_SPI_FRAME_SIZE_11_BIT 11 bit frame.
QM_SS_SPI_FRAME_SIZE_12_BIT 12 bit frame.
QM_SS_SPI_FRAME_SIZE_13_BIT 13 bit frame.
QM_SS_SPI_FRAME_SIZE_14_BIT 14 bit frame.
QM_SS_SPI_FRAME_SIZE_15_BIT 15 bit frame.
QM_SS_SPI_FRAME_SIZE_16_BIT 16 bit frame.

Definition at line 21 of file qm_ss_spi.h.

4.31.2.3 enum qm_ss_spi_slave_select_t

SPI slave select type.

Slave selects can combined by logical OR if multiple slaves are selected during one transfer. Setting only QM_SS-_SPI_SS_DISABLED prevents the controller from starting the transfer.

Enumerator

QM_SS_SPI_SS_DISABLED Slave select disable.
QM_SS_SPI_SS_0 Slave select 0.
QM_SS_SPI_SS_1 Slave select 1.
QM_SS_SPI_SS_2 Slave select 2.
QM_SS_SPI_SS_3 Slave select 3.

Definition at line 92 of file qm_ss_spi.h.

4.31.2.4 enum qm_ss_spi_status_t

SPI status.

Enumerator

QM_SS_SPI_IDLE SPI device is not in use.

QM_SS_SPI_BUSY SPI device is busy.

QM_SS_SPI_RX_OVERFLOW RX transfer has overflown.

Definition at line 103 of file qm_ss_spi.h.

4.31.2.5 enum qm_ss_spi_tmode_t

SPI transfer mode type.

Enumerator

QM_SS_SPI_TMOD_TX_RX Transmit & Receive mode. This mode synchronously receives and transmits data during the transfer. rx_len and tx_len buffer need to be the same length.

QM_SS_SPI_TMOD_TX Transmit-Only mode. This mode only transmits data. The rx buffer is not accessed and rx_len need to be set to 0.

QM_SS_SPI_TMOD_RX Receive-Only mode. This mode only receives data. The tx buffer is not accessed and tx_len need to be set to 0.

QM_SS_SPI_TMOD_EEPROM_READ EEPROM-Read Mode. This mode transmits the data stored in the tx buffer (EEPROM address). After the transmit is completed it populates the rx buffer (EEPROM data) with received data.

Definition at line 40 of file qm_ss_spi.h.

4.31.3 Function Documentation

4.31.3.1 int qm_ss_spi_get_status (const qm_ss_spi_t *spi*, qm_ss_spi_status_t *const *status*)

Get SPI bus status.

Parameters

in	*spi*	SPI module identifier.
out	*status*	Reference to the variable where to store the current SPI bus status (QM_SS_SPI_BUSY if a transfer is in progress or QM_SS_SPI_IDLE if SPI device is IDLE). This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 100 of file qm_ss_spi.c.

References QM_SS_SPI_BUSY, QM_SS_SPI_IDLE, and QM_SS_SPI_SR.

4.31.3.2 int qm_ss_spi_irq_transfer (const qm_ss_spi_t *spi,* const qm_ss_spi_async_transfer_t *const *xfer*)

Initiate a interrupt based SPI transfer.

Perform an interrupt based SPI transfer. If transfer mode is full duplex (QM_SS_SPI_TMOD_TX_RX), then tx_len and rx_len must be equal. Similarly, for transmit-only transfers (QM_SS_SPI_TMOD_TX) rx_len must be 0, while for receive-only transfers (QM_SS_SPI_TMOD_RX) tx_len must be 0. This function is non blocking.

Parameters

in	*spi*	SPI module identifier.
in	*xfer*	Structure containing transfer information. The structure must not be NULL and must be kept valid until the transfer is complete.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 197 of file qm_ss_spi.c.

References QM_SS_SPI_CTRL, QM_SS_SPI_FTLR, QM_SS_SPI_INTR_MASK, QM_SS_SPI_SPIEN, QM_SS_SPI_TMOD_EEPROM_READ, QM_SS_SPI_TMOD_RX, QM_SS_SPI_TMOD_TX_RX, qm_ss_spi_async_transfer_t::rx_len, and qm_ss_spi_async_transfer_t::tx_len.

4.31.3.3 int qm_ss_spi_set_config (const qm_ss_spi_t *spi*, const qm_ss_spi_config_t *const *cfg*)

Set SPI configuration.

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

This operation is permitted only when the SPI module is disabled.

Parameters

in	*spi*	SPI module identifier.
in	*cfg*	New configuration for SPI. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 60 of file qm_ss_spi.c.

References qm_ss_spi_config_t::bus_mode, qm_ss_spi_config_t::clk_divider, qm_ss_spi_config_t::frame_size, QM_SS_SPI_CTRL, QM_SS_SPI_SPIEN, QM_SS_SPI_TIMING, and qm_ss_spi_config_t::transfer_mode.

4.31.3.4 int qm_ss_spi_slave_select (const qm_ss_spi_t *spi*, const qm_ss_spi_slave_select_t *ss*)

Set Slave Select lines.

Select which slaves to perform SPI transmissions on. Select lines can be combined using the | operator. It is only suggested to use this functionality in TX only mode. This operation is permitted only when a SPI transfer is not already in progress; the caller should check that by retrieving the device status.

Parameters

in	*spi*	SPI module identifier.
in	*ss*	Select lines to enable when performing transfers.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 82 of file qm_ss_spi.c.

References QM_SS_SPI_SPIEN, and QM_SS_SPI_SR.

4.31.3.5 int qm_ss_spi_transfer (const qm_ss_spi_t *spi*, const qm_ss_spi_transfer_t *const *xfer*, qm_ss_spi_status_t *const *status*)

Perform a blocking SPI transfer.

This is a blocking synchronous call. If transfer mode is full duplex (QM_SS_SPI_TMOD_TX_RX) tx_len and rx_len must be equal. Similarly, for transmit-only transfers (QM_SS_SPI_TMOD_TX) rx_len must be 0, while for receive-only transfers (QM_SS_SPI_TMOD_RX) tx_len must be 0.

Parameters

in	*spi*	SPI module identifier.
in	*xfer*	Structure containing transfer information. This must not be NULL.
out	*status*	Reference to the variable where to store the SPI status at the end of the transfer.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 115 of file qm_ss_spi.c.

References QM_SS_SPI_CTRL, QM_SS_SPI_INTR_MASK, QM_SS_SPI_INTR_STAT, QM_SS_SPI_RX_OVERFLOW, QM_SS_SPI_SPIEN, QM_SS_SPI_SR, QM_SS_SPI_TMOD_EEPROM_READ, QM_SS_SPI_TMOD_RX, QM_SS_SPI_TMOD_TX, QM_SS_SPI_TMOD_TX_RX, qm_ss_spi_transfer_t::rx, qm_ss_spi_transfer_t::rx_len, qm_ss_spi_transfer_t::tx, and qm_ss_spi_transfer_t::tx_len.

4.31.3.6 int qm_ss_spi_transfer_terminate (const qm_ss_spi_t *spi*)

Terminate SPI IRQ transfer.

Terminate the current IRQ SPI transfer. This function will trigger complete callbacks even if the transfer is not completed.

Parameters

in	*spi*	SPI module identifier.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Negative	errno for possible error codes.

Definition at line 246 of file qm_ss_spi.c.

References qm_ss_spi_async_transfer_t::callback, qm_ss_spi_async_transfer_t::callback_data, QM_SS_SPI_CTRL, QM_SS_SPI_IDLE, QM_SS_SPI_TMOD_TX, QM_SS_SPI_TMOD_TX_RX, qm_ss_spi_async_transfer_t::rx_len, and qm_ss_spi_async_transfer_t::tx_len.

4.32 SS Timer

Timer driver for Sensor Subsystem.

Data Structures

- struct qm_ss_timer_config_t

  Sensor Subsystem Timer Configuration Type.

Functions

- int qm_ss_timer_set_config (const qm_ss_timer_t timer, const qm_ss_timer_config_t *const cfg)

  Set the SS timer configuration.
- int qm_ss_timer_set (const qm_ss_timer_t timer, const uint32_t count)

  Set SS timer count value.
- int qm_ss_timer_get (const qm_ss_timer_t timer, uint32_t *const count)

  Get SS timer count value.

4.32.1 Detailed Description

Timer driver for Sensor Subsystem.

4.32.2 Function Documentation

4.32.2.1 int qm_ss_timer_get (const qm_ss_timer_t *timer*, uint32_t *const *count*)

Get SS timer count value.

Get the current count value of the SS timer.

Parameters

in	*timer*	Which SS timer to get the count of.
out	*count*	Current value of timer. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 59 of file qm_ss_timer.c.

4.32.2.2 int qm_ss_timer_set (const qm_ss_timer_t *timer*, const uint32_t *count*)

Set SS timer count value.

Set the current count value of the SS timer.

Parameters

in	*timer*	Which SS timer to set the count of.
in	*count*	Value to load the timer with.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 50 of file qm_ss_timer.c.

4.32.2.3 int qm_ss_timer_set_config (const qm_ss_timer_t *timer*, const qm_ss_timer_config_t *const *cfg*)

Set the SS timer configuration.

This includes final count value, timer mode and if interrupts are enabled. If interrupts are enabled, it will configure the callback function.

Parameters

in	*timer*	Which SS timer to configure.
in	*cfg*	SS timer configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 29 of file qm_ss_timer.c.

References qm_ss_timer_config_t::callback, qm_ss_timer_config_t::callback_data, qm_ss_timer_config_t::count, qm_ss_timer_config_t::inc_run_only, qm_ss_timer_config_t::int_en, and qm_ss_timer_config_t::watchdog_mode.

4.33 SS Clock

Clock Management for Sensor Subsystem.

Enumerations

- enum ss_clk_periph_t {
  SS_CLK_PERIPH_ADC = BIT(31), SS_CLK_PERIPH_I2C_1 = BIT(30), SS_CLK_PERIPH_I2C_0 = BIT(29),
  SS_CLK_PERIPH_SPI_1 = BIT(28),
  SS_CLK_PERIPH_SPI_0 = BIT(27), SS_CLK_PERIPH_GPIO_1 = BIT(1), SS_CLK_PERIPH_GPIO_0 = BIT(0) }

  Peripheral clocks selection type.

Functions

- int ss_clk_gpio_enable (const qm_ss_gpio_t gpio)

  Enable clocking for SS GPIO peripheral.
- int ss_clk_gpio_disable (const qm_ss_gpio_t gpio)

  Disable clocking for SS GPIO peripheral.
- int ss_clk_spi_enable (const qm_ss_spi_t spi)

  Enable clocking for SS SPI peripheral.
- int ss_clk_spi_disable (const qm_ss_spi_t spi)

  Disable clocking for SS SPI peripheral.
- int ss_clk_i2c_enable (const qm_ss_i2c_t i2c)

  Enable clocking for SS I2C peripheral.
- int ss_clk_i2c_disable (const qm_ss_i2c_t i2c)

  Disable clocking for SS I2C peripheral.
- int ss_clk_adc_enable (void)

  Enable the SS ADC clock.
- int ss_clk_adc_disable (void)

  Disable the SS ADC clock.
- int ss_clk_adc_set_div (const uint32_t div)

  Set clock divisor for SS ADC.

4.33.1 Detailed Description

Clock Management for Sensor Subsystem. The clock distribution has three level of gating:

1. SE SoC gating through register CCU_PERIPH_CLK_GATE_CTL
2. SS Soc gating through register IO_CREG_MST0_CTRL (IO_CREG_MST0_CTRL)
3. SS peripheral clk gating Note: the first two are ungated by hardware power-on default (clock gating is done at peripheral level). Thus the only one level of control is enough (and implemented in ss_clk driver) to gate clock on or off to the particular peripheral.

4.33.2 Enumeration Type Documentation

4.33.2.1 enum ss_clk_periph_t

Peripheral clocks selection type.

Enumerator

SS_CLK_PERIPH_ADC ADC clock selector.

SS_CLK_PERIPH_I2C_1 I2C 1 clock selector.

SS_CLK_PERIPH_I2C_0 I2C 0 clock selector.

SS_CLK_PERIPH_SPI_1 SPI 1 clock selector.

SS_CLK_PERIPH_SPI_0 SPI 0 clock selector.

SS_CLK_PERIPH_GPIO_1 GPIO 1 clock selector. Special domain peripherals - these do not map onto the standard register.

SS_CLK_PERIPH_GPIO_0 GPIO 0 clock selector. Special domain peripherals - these do not map onto the standard register.

Definition at line 31 of file ss_clk.h.

4.33.3 Function Documentation

4.33.3.1 int ss_clk_adc_disable (void)

Disable the SS ADC clock.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Definition at line 68 of file ss_clk.c.

References QM_SS_ADC_CTRL.

4.33.3.2 int ss_clk_adc_enable (void)

Enable the SS ADC clock.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Definition at line 60 of file ss_clk.c.

References QM_SS_ADC_CTRL.

4.33.3.3 int ss_clk_adc_set_div (const uint32_t *div*)

Set clock divisor for SS ADC.

Note: If the system clock speed is changed, the divisor must be recalculated. The minimum supported speed for the SS ADC is 0.14 MHz. So for a system clock speed of 1 MHz, the max value of div is 7, and for 32 MHz, the max value is

1. System clock speeds of less than 1 MHz are not supported by this function.

Parameters

in	*div*	ADC clock divider value.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 76 of file ss_clk.c.

References clk_sys_get_ticks_per_us(), and QM_SS_ADC_DIVSEQSTAT.

4.33.3.4 int ss_clk_gpio_disable (const qm_ss_gpio_t *gpio*)

Disable clocking for SS GPIO peripheral.

Parameters

in	*gpio*	GPIO port index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 19 of file ss_clk.c.

4.33.3.5 int ss_clk_gpio_enable (const qm_ss_gpio_t *gpio*)

Enable clocking for SS GPIO peripheral.

Parameters

in	*gpio*	GPIO port index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 8 of file ss_clk.c.

4.33.3.6 int ss_clk_i2c_disable (const qm_ss_i2c_t *i2c*)

Disable clocking for SS I2C peripheral.

Parameters

in	*i2c*	I2C port index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 52 of file ss_clk.c.

4.33.3.7 int ss_clk_i2c_enable (const qm_ss_i2c_t *i2c*)

Enable clocking for SS I2C peripheral.

Parameters

in	*i2c*	I2C port index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 44 of file ss_clk.c.

4.33.3.8 int ss_clk_spi_disable (const qm_ss_spi_t *spi*)

Disable clocking for SS SPI peripheral.

Parameters

in	*spi*	SPI port index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 36 of file ss_clk.c.

References QM_SS_SPI_0, and QM_SS_SPI_CTRL.

4.33.3.9 int ss_clk_spi_enable (const qm_ss_spi_t *spi*)

Enable clocking for SS SPI peripheral.

Parameters

in	*spi*	SPI port index.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 28 of file ss_clk.c.

References QM_SS_SPI_0, and QM_SS_SPI_CTRL.

4.34 SS Power states

SS Power mode control for Quark SE Microcontrollers.

Enumerations

- enum ss_power_cpu_ss1_mode_t { SS_POWER_CPU_SS1_TIMER_OFF = 0, SS_POWER_CPU_SS1_TIMER_ON }

  Sensor Subsystem SS1 Timers mode type.

Functions

- void ss_power_soc_lpss_enable (void)

  Enable LPSS state entry.
- void ss_power_soc_lpss_disable (void)

  Disable LPSS state entry.
- void ss_power_cpu_ss1 (const ss_power_cpu_ss1_mode_t mode)

  Enter Sensor SS1 state.
- void ss_power_cpu_ss2 (void)

  Enter Sensor SS2 state or SoC LPSS state.

4.34.1 Detailed Description

SS Power mode control for Quark SE Microcontrollers.

4.34.2 Enumeration Type Documentation

4.34.2.1 enum ss_power_cpu_ss1_mode_t

Sensor Subsystem SS1 Timers mode type.

Enumerator

SS_POWER_CPU_SS1_TIMER_OFF Disable SS Timers in SS1.

SS_POWER_CPU_SS1_TIMER_ON Keep SS Timers enabled in SS1.

Definition at line 21 of file ss_power_states.h.

4.34.3 Function Documentation

4.34.3.1 void ss_power_cpu_ss1 (const ss_power_cpu_ss1_mode_t *mode*)

Enter Sensor SS1 state.

Put the Sensor Subsystem into SS1.

Processor Clock is gated in this state.

A wake event causes the Sensor Subsystem to transition to SS0.

A wake event is a sensor subsystem interrupt.

According to the mode selected, Sensor Subsystem Timers can be disabled.

Parameters

in	*mode*	Mode selection for SS1 state.

Definition at line 82 of file ss_power_states.c.

References SS_POWER_CPU_SS1_TIMER_OFF, and SS_POWER_CPU_SS1_TIMER_ON.

4.34.3.2 void ss_power_cpu_ss2 (void)

Enter Sensor SS2 state or SoC LPSS state.

Put the Sensor Subsystem into SS2.

Sensor Complex Clock is gated in this state.

Sensor Peripherals are gated in this state.

This enables entry in LPSS if:

- Sensor Subsystem is in SS2
- Lakemont is in C2 or C2LP
- LPSS entry is enabled

A wake event causes the Sensor Subsystem to transition to SS0.

There are two kinds of wake event depending on the Sensor Subsystem and SoC state:

- SS2: a wake event is a Sensor Subsystem interrupt
- LPSS: a wake event is a Sensor Subsystem interrupt or a Lakemont interrupt

LPSS wake events apply if LPSS is entered. If Host wakes the SoC from LPSS, Sensor also transitions back to SS0.

Definition at line 123 of file ss_power_states.c.

4.34.3.3 void ss_power_soc_lpss_disable (void)

Disable LPSS state entry.

Clear LPSS enable flag.

Disable Clock Gating of ADC, I2C0, I2C1, SPI0 and SPI1 sensor peripherals.

This will prevent entry in LPSS when cores are in C2/C2LP and SS2 states.

Definition at line 55 of file ss_power_states.c.

References SOCW_EVENT_REGISTER, and SOCW_REG_CCU_LP_CLK_CTL.

4.34.3.4 void ss_power_soc_lpss_enable (void)

Enable LPSS state entry.

Put the SoC into LPSS on next C2/C2LP and SS2 state combination.

This function needs to be called on the Sensor Core to Clock Gate ADC, I2C0, I2C1, SPI0 and SPI1 sensor peripherals.

Clock Gating sensor peripherals is a requirement to enter LPSS state.

After LPSS, ss_power_soc_lpss_disable needs to be called to restore clock gating.

This needs to be called before any transition to C2/C2LP and SS2 in order to enter LPSS.

SoC Hybrid Clock is gated in this state.

Core Well Clocks are gated.

RTC is the only clock running.

Possible SoC wake events are:

- Low Power Comparator Interrupt
- AON GPIO Interrupt
- AON Timer Interrupt
- RTC Interrupt

Definition at line 32 of file ss_power_states.c.

References SOCW_EVENT_REGISTER, and SOCW_REG_CCU_LP_CLK_CTL.

4.35 Quark D2000 Flash Layout

Flash Layout for Quark D2000 Microcontrollers.

4.35.1 Detailed Description

Flash Layout for Quark D2000 Microcontrollers.

4.36 Quark D2000 Power states

Power mode control for Quark D2000 Microcontrollers.

Enumerations

- enum power_wake_event_t { POWER_WAKE_FROM_GPIO_COMP, POWER_WAKE_FROM_RTC }
  Wake source for deep sleep mode type.

Functions

- void power_cpu_halt (void)
  Put CPU in halt state.
- void power_soc_sleep ()
  Put SoC to sleep.
- void power_soc_deep_sleep (const power_wake_event_t wake_event)
  Put SoC to deep sleep.

4.36.1 Detailed Description

Power mode control for Quark D2000 Microcontrollers.

4.36.2 Enumeration Type Documentation

4.36.2.1 enum power_wake_event_t

Wake source for deep sleep mode type.

Enumerator

POWER_WAKE_FROM_GPIO_COMP Use GPIO / Comparator as wake source.

POWER_WAKE_FROM_RTC Use RTC as wake source.

Definition at line 21 of file power_states.h.

4.36.3 Function Documentation

4.36.3.1 void power_cpu_halt (void)

Put CPU in halt state.

Halts the CPU until next interrupt or reset.

Definition at line 15 of file power_states.c.

References SOCW_EVENT_HALT.

Referenced by power_soc_deep_sleep(), and power_soc_sleep().

4.36.3.2 void power_soc_deep_sleep (const power_wake_event_t *wake_event*)

Put SoC to deep sleep.

Enter into deep sleep mode. All clocks are gated. The Wake source for this function depends on the input parameter, POWER_WAKE_FROM_GPIO_COMP will enable waking from GPIO or comparator pins and POWER_WAKE_FROM_RTC will enable waking from the RTC.

Parameters

in	*wake_event*	Select wake source for deep sleep mode.

Definition at line 149 of file power_states.c.

References CLK_PERIPH_CLK, clk_periph_disable(), clk_periph_enable(), CLK_PERIPH_REGISTER, CLK_SYS_DIV_1, CLK_SYS_DIV_128, CLK_SYS_HYB_OSC_4MHZ, CLK_SYS_RTC_OSC, clk_sys_set_mode(), power_cpu_halt(), POWER_WAKE_FROM_GPIO_COMP, POWER_WAKE_FROM_RTC, QM_ADC_MODE_DEEP_PWR_DOWN, qm_adc_set_mode(), RAR_NORMAL, RAR_RETENTION, rar_set_mode(), SOCW_EVENT_REGISTER, SOCW_REG_CCU_EXT_CLK_CTL, SOCW_REG_CCU_LP_CLK_CTL, SOCW_REG_CCU_PERIPH_CLK_GATE_CTL, SOCW_REG_CCU_SYS_CLK_CTL, SOCW_REG_OSC0_CFG1, and SOCW_REG_PMUX_SLEW.

4.36.3.3 void power_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

Definition at line 36 of file power_states.c.

4.37 SoC Registers (D2000)

Quark D2000 SoC Registers.

Data Structures

- struct qm_scss_ccu_reg_t
  System Core register map.
- struct qm_scss_gp_reg_t
  General Purpose register map.
- struct qm_scss_cmp_reg_t
  Comparator register map.
- struct qm_scss_int_reg_t
  Interrupt register map.
- struct qm_scss_pmu_reg_t
  Power Management register map.
- struct qm_scss_aon_reg_t
  Always-on Controller register map.
- struct qm_scss_peripheral_reg_t
  Peripheral Registers register map.
- struct qm_scss_pmux_reg_t
  Pin MUX register map.
- struct qm_scss_info_reg_t
  Information register map.
- struct qm_pwm_channel_t
  PWM / Timer channel register map.
- struct qm_pwm_reg_t
  PWM / Timer register map.
- struct qm_wdt_reg_t
  Watchdog timer register map.
- struct qm_uart_reg_t
  UART register map.
- struct qm_spi_reg_t
  SPI register map.
- struct qm_rtc_reg_t
  RTC register map.
- struct qm_i2c_reg_t
  I2C register map.
- struct qm_gpio_reg_t
  GPIO register map.
- struct qm_adc_reg_t
  ADC register map.
- struct qm_flash_reg_t
  Flash register map.
- struct qm_mpr_reg_t
  Memory Protection Region register map.
- struct pic_timer_reg_pad_t
  PIC timer register structure.
- struct qm_pic_timer_reg_t
  PIC timer register map.
- struct mvic_reg_pad_t

MVIC register structure.

- struct qm_mvic_reg_t

  MVIC register map.

- struct qm_dma_chan_reg_t

  DMA channel register map.

- struct qm_dma_int_reg_t

  DMA interrupt register map.

- struct qm_dma_misc_reg_t

  DMA miscellaneous register map.

System Core

- qm_scss_ccu_reg_t **test_scss_ccu**

General Purpose

- qm_scss_gp_reg_t **test_scss_gp**

Comparator

- qm_scss_cmp_reg_t **test_scss_cmp**

Interrupt

- qm_scss_int_reg_t **test_scss_int**

Power Management

- qm_scss_pmu_reg_t **test_scss_pmu**

Always-on controllers.

- enum qm_scss_aon_t

  Number of SCSS Always-on controllers.

- qm_scss_aon_reg_t **test_scss_aon**

Peripheral Registers

- qm_scss_peripheral_reg_t **test_scss_peripheral**

Pin MUX

- qm_scss_pmux_reg_t **test_scss_pmux**

ID

- qm_scss_info_reg_t **test_scss_info**

PWM / Timer

- enum qm_pwm_t
  Number of PWM / Timer controllers.
- enum qm_pwm_id_t
  PWM ID type.
- qm_pwm_reg_t **test_pwm_t**

WDT

- enum qm_wdt_t
  Number of WDT controllers.
- qm_wdt_reg_t **test_wdt**

UART

- enum qm_uart_t
  Number of UART controllers.
- qm_uart_reg_t **test_uart_instance**
- qm_uart_reg_t * **test_uart** [QM_UART_NUM]
- qm_uart_reg_t * **qm_uart** [QM_UART_NUM]

SPI

- enum qm_spi_t
  Number of SPI controllers.
- qm_spi_reg_t **test_spi**
- qm_spi_reg_t * **test_spi_controllers** [QM_SPI_NUM]
- qm_spi_reg_t * qm_spi_controllers [QM_SPI_NUM]
  Extern qm_spi_reg_t array declared at qm_soc_regs.h .*

RTC

- enum qm_rtc_t
  Number of RTC controllers.
- qm_rtc_reg_t **test_rtc**

I2C

- enum qm_i2c_t
  Number of I2C controllers.
- qm_i2c_reg_t **test_i2c_instance** [QM_I2C_NUM]
- qm_i2c_reg_t * **test_i2c** [QM_I2C_NUM]
- qm_i2c_reg_t * qm_i2c [QM_I2C_NUM]
  I2C register block.

GPIO

- enum qm_gpio_t
  Number of GPIO controllers.
- qm_gpio_reg_t **test_gpio_instance**
- qm_gpio_reg_t * **test_gpio** [QM_GPIO_NUM]
- qm_gpio_reg_t * **qm_gpio** [QM_GPIO_NUM]

ADC

- enum qm_adc_t

  Number of ADC controllers.
- qm_adc_reg_t **test_adc**

Flash

- enum qm_flash_t

  Number of Flash controllers.
- qm_flash_reg_t **test_flash_instance**
- qm_flash_reg_t * **test_flash** [QM_FLASH_NUM]
- uint8_t **test_flash_page** [0x800]
- qm_flash_reg_t * **qm_flash** [QM_FLASH_NUM]

Memory Protection Region

- qm_mpr_reg_t **test_mpr**

PIC

- qm_pic_timer_reg_t **test_pic_timer**

Peripheral Clock

- enum clk_periph_t {
  CLK_PERIPH_REGISTER = BIT(0), CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_PERIPH_SPI_S = BIT(4),
  CLK_PERIPH_SPI_M0 = BIT(5), CLK_PERIPH_GPIO_INTERRUPT = BIT(7), CLK_PERIPH_GPIO_DB = BIT(8), CLK_PERIPH_WDT_REGISTER = BIT(10),
  CLK_PERIPH_RTC_REGISTER = BIT(11), CLK_PERIPH_PWM_REGISTER = BIT(12), CLK_PERIPH_GPIO_REGISTER = BIT(13), CLK_PERIPH_SPI_M0_REGISTER,
  CLK_PERIPH_SPI_S_REGISTER, CLK_PERIPH_UARTA_REGISTER = BIT(17), CLK_PERIPH_UARTB_REGISTER = BIT(18), CLK_PERIPH_I2C_M0_REGISTER,
  CLK_PERIPH_ADC = BIT(22), CLK_PERIPH_ADC_REGISTER = BIT(23), CLK_PERIPH_ALL = 0xCFFFFF,
  CLK_PERIPH_REGISTER = BIT(0),
  CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_PERIPH_I2C_M1 = BIT(3), CLK_PERIPH_SPI_S = BIT(4),
  CLK_PERIPH_SPI_M0 = BIT(5), CLK_PERIPH_SPI_M1 = BIT(6), CLK_PERIPH_GPIO_INTERRUPT = BIT(7), CLK_PERIPH_GPIO_DB = BIT(8),
  CLK_PERIPH_I2S = BIT(9), CLK_PERIPH_WDT_REGISTER = BIT(10), CLK_PERIPH_RTC_REGISTER = BIT(11), CLK_PERIPH_PWM_REGISTER = BIT(12),
  CLK_PERIPH_GPIO_REGISTER = BIT(13), CLK_PERIPH_SPI_M0_REGISTER, CLK_PERIPH_SPI_M1_REGISTER, CLK_PERIPH_SPI_S_REGISTER,
  CLK_PERIPH_UARTA_REGISTER = BIT(17), CLK_PERIPH_UARTB_REGISTER = BIT(18), CLK_PERIPH_I2C_M0_REGISTER, CLK_PERIPH_I2C_M1_REGISTER,
  CLK_PERIPH_I2S_REGISTER = BIT(21), CLK_PERIPH_ALL = 0x3FFFFF }

  Peripheral clock register map.

MVIC

- qm_mvic_reg_t **test_mvic**
- qm_ioapic_reg_t **test_ioapic**

DMA

- enum qm_dma_t { QM_DMA_0, QM_DMA_NUM, QM_DMA_0, QM_DMA_NUM }

  DMA instances.
- enum qm_dma_channel_id_t {
  QM_DMA_CHANNEL_0 = 0, QM_DMA_CHANNEL_1, QM_DMA_CHANNEL_NUM, QM_DMA_CHANNEL_0 = 0,
  QM_DMA_CHANNEL_1, QM_DMA_CHANNEL_2, QM_DMA_CHANNEL_3, QM_DMA_CHANNEL_4,
  QM_DMA_CHANNEL_5, QM_DMA_CHANNEL_6, QM_DMA_CHANNEL_7, QM_DMA_CHANNEL_NUM }

  DMA channel IDs.
- enum qm_dma_handshake_interface_t {
  DMA_HW_IF_UART_A_TX = 0x0, DMA_HW_IF_UART_A_RX = 0x1, DMA_HW_IF_UART_B_TX = 0x2, DMA_HW_IF_UART_B_RX = 0x3,
  DMA_HW_IF_SPI_MASTER_0_TX = 0x4, DMA_HW_IF_SPI_MASTER_0_RX = 0x5, DMA_HW_IF_SPI_SLAVE_TX = 0x8, DMA_HW_IF_SPI_SLAVE_RX = 0x9,
  DMA_HW_IF_I2C_MASTER_0_TX = 0xc, DMA_HW_IF_I2C_MASTER_0_RX = 0xd, DMA_HW_IF_UART_A_TX = 0x0, DMA_HW_IF_UART_A_RX = 0x1,
  DMA_HW_IF_UART_B_TX = 0x2, DMA_HW_IF_UART_B_RX = 0x3, DMA_HW_IF_SPI_MASTER_0_TX = 0x4, DMA_HW_IF_SPI_MASTER_0_RX = 0x5,
  DMA_HW_IF_SPI_MASTER_1_TX = 0x6, DMA_HW_IF_SPI_MASTER_1_RX = 0x7, DMA_HW_IF_SPI_SLAVE_TX = 0x8, DMA_HW_IF_SPI_SLAVE_RX = 0x9,
  DMA_HW_IF_I2S_PLAYBACK = 0xa, DMA_HW_IF_I2S_CAPTURE = 0xb, DMA_HW_IF_I2C_MASTER_0_TX = 0xc, DMA_HW_IF_I2C_MASTER_0_RX = 0xd,
  DMA_HW_IF_I2C_MASTER_1_TX = 0xe, DMA_HW_IF_I2C_MASTER_1_RX = 0xf }

  DMA hardware handshake interfaces.
- qm_dma_reg_t **test_dma_instance** [QM_DMA_NUM]
- qm_dma_reg_t * **test_dma** [QM_DMA_NUM]
- qm_dma_reg_t * **qm_dma** [QM_DMA_NUM]

Versioning

- uint32_t **test_rom_version**

4.37.1 Detailed Description

Quark D2000 SoC Registers.

4.37.2 Enumeration Type Documentation

4.37.2.1 enum clk_periph_t

Peripheral clock register map.

Enumerator

CLK_PERIPH_REGISTER Peripheral Clock Gate Enable.

CLK_PERIPH_CLK Peripheral Clock Enable.

CLK_PERIPH_I2C_M0 I2C Master 0 Clock Enable.

CLK_PERIPH_SPI_S SPI Slave Clock Enable.

CLK_PERIPH_SPI_M0 SPI Master 0 Clock Enable.

CLK_PERIPH_GPIO_INTERRUPT GPIO Interrupt Clock Enable.

CLK_PERIPH_GPIO_DB GPIO Debounce Clock Enable.

CLK_PERIPH_WDT_REGISTER Watchdog Clock Enable.

CLK_PERIPH_RTC_REGISTER RTC Clock Gate Enable.

CLK_PERIPH_PWM_REGISTER PWM Clock Gate Enable.

CLK_PERIPH_GPIO_REGISTER GPIO Clock Gate Enable.

CLK_PERIPH_SPI_M0_REGISTER SPI Master 0 Clock Gate Enable.

CLK_PERIPH_SPI_S_REGISTER SPI Slave Clock Gate Enable.

CLK_PERIPH_UARTA_REGISTER UARTA Clock Gate Enable.

CLK_PERIPH_UARTB_REGISTER UARTB Clock Gate Enable.

CLK_PERIPH_I2C_M0_REGISTER I2C Master 0 Clock Gate Enable.

CLK_PERIPH_ADC ADC Clock Enable.

CLK_PERIPH_ADC_REGISTER ADC Clock Gate Enable.

CLK_PERIPH_ALL Quark D2000 peripherals Enable.

CLK_PERIPH_REGISTER Peripheral Clock Gate Enable.

CLK_PERIPH_CLK Peripheral Clock Enable.

CLK_PERIPH_I2C_M0 I2C Master 0 Clock Enable.

CLK_PERIPH_I2C_M1 I2C Master 1 Clock Enable.

CLK_PERIPH_SPI_S SPI Slave Clock Enable.

CLK_PERIPH_SPI_M0 SPI Master 0 Clock Enable.

CLK_PERIPH_SPI_M1 SPI Master 1 Clock Enable.

CLK_PERIPH_GPIO_INTERRUPT GPIO Interrupt Clock Enable.

CLK_PERIPH_GPIO_DB GPIO Debounce Clock Enable.

CLK_PERIPH_I2S I2S Clock Enable.

CLK_PERIPH_WDT_REGISTER Watchdog Clock Enable.

CLK_PERIPH_RTC_REGISTER RTC Clock Gate Enable.

CLK_PERIPH_PWM_REGISTER PWM Clock Gate Enable.

CLK_PERIPH_GPIO_REGISTER GPIO Clock Gate Enable.

CLK_PERIPH_SPI_M0_REGISTER SPI Master 0 Clock Gate Enable.

CLK_PERIPH_SPI_M1_REGISTER SPI Master 1 Clock Gate Enable.

CLK_PERIPH_SPI_S_REGISTER SPI Slave Clock Gate Enable.

CLK_PERIPH_UARTA_REGISTER UARTA Clock Gate Enable.

CLK_PERIPH_UARTB_REGISTER UARTB Clock Gate Enable.

CLK_PERIPH_I2C_M0_REGISTER I2C Master 0 Clock Gate Enable.

CLK_PERIPH_I2C_M1_REGISTER I2C Master 1 Clock Gate Enable.

CLK_PERIPH_I2S_REGISTER I2S Clock Gate Enable.

CLK_PERIPH_ALL Quark SE peripherals Mask.

Definition at line 1183 of file qm_soc_regs.h.

4.37.2.2 enum qm_adc_t

Number of ADC controllers.

Definition at line 976 of file qm_soc_regs.h.

4.37.2.3 enum qm_dma_channel_id_t

DMA channel IDs.

Enumerator

QM_DMA_CHANNEL_0 DMA channel id for channel 0.
QM_DMA_CHANNEL_1 DMA channel id for channel 1.
QM_DMA_CHANNEL_NUM Number of DMA channels.
QM_DMA_CHANNEL_0 DMA channel id for channel 0.
QM_DMA_CHANNEL_1 DMA channel id for channel 1.
QM_DMA_CHANNEL_2 DMA channel id for channel 2.
QM_DMA_CHANNEL_3 DMA channel id for channel 3.
QM_DMA_CHANNEL_4 DMA channel id for channel 4.
QM_DMA_CHANNEL_5 DMA channel id for channel 5.
QM_DMA_CHANNEL_6 DMA channel id for channel 6.
QM_DMA_CHANNEL_7 DMA channel id for channel 7.
QM_DMA_CHANNEL_NUM Number of DMA channels.

Definition at line 1299 of file qm_soc_regs.h.

4.37.2.4 enum qm_dma_handshake_interface_t

DMA hardware handshake interfaces.

Enumerator

DMA_HW_IF_UART_A_TX UART_A_TX.
DMA_HW_IF_UART_A_RX UART_A_RX.
DMA_HW_IF_UART_B_TX UART_B_TX.
DMA_HW_IF_UART_B_RX UART_B_RX.
DMA_HW_IF_SPI_MASTER_0_TX SPI_Master_0_TX.
DMA_HW_IF_SPI_MASTER_0_RX SPI_Master_0_RX.
DMA_HW_IF_SPI_SLAVE_TX SPI_Slave_TX.
DMA_HW_IF_SPI_SLAVE_RX SPI_Slave_RX.
DMA_HW_IF_I2C_MASTER_0_TX I2C_Master_0_TX.
DMA_HW_IF_I2C_MASTER_0_RX I2C_Master_0_RX.
DMA_HW_IF_UART_A_TX UART_A_TX.
DMA_HW_IF_UART_A_RX UART_A_RX.
DMA_HW_IF_UART_B_TX UART_B_TX.
DMA_HW_IF_UART_B_RX UART_B_RX.
DMA_HW_IF_SPI_MASTER_0_TX SPI_Master_0_TX.
DMA_HW_IF_SPI_MASTER_0_RX SPI_Master_0_RX.
DMA_HW_IF_SPI_MASTER_1_TX SPI_Master_1_TX.
DMA_HW_IF_SPI_MASTER_1_RX SPI_Master_1_RX.
DMA_HW_IF_SPI_SLAVE_TX SPI_Slave_TX.
DMA_HW_IF_SPI_SLAVE_RX SPI_Slave_RX.
DMA_HW_IF_I2S_PLAYBACK I2S_Playback channel.
DMA_HW_IF_I2S_CAPTURE I2S_Capture channel.
DMA_HW_IF_I2C_MASTER_0_TX I2C_Master_0_TX.
DMA_HW_IF_I2C_MASTER_0_RX I2C_Master_0_RX.
DMA_HW_IF_I2C_MASTER_1_TX I2C_Master_1_TX.
DMA_HW_IF_I2C_MASTER_1_RX I2C_Master_1_RX.

Definition at line 1306 of file qm_soc_regs.h.

4.37.2.5 enum qm_dma_t

DMA instances.

Enumerator

QM_DMA_0 DMA controller id.
QM_DMA_NUM Number of DMA controllers.
QM_DMA_0 DMA controller id.
QM_DMA_NUM Number of DMA controllers.

Definition at line 1293 of file qm_soc_regs.h.

4.37.2.6 enum qm_flash_t

Number of Flash controllers.

Definition at line 1048 of file qm_soc_regs.h.

4.37.2.7 enum qm_gpio_t

Number of GPIO controllers.

Definition at line 926 of file qm_soc_regs.h.

4.37.2.8 enum qm_i2c_t

Number of I2C controllers.

Definition at line 790 of file qm_soc_regs.h.

4.37.2.9 enum qm_pwm_id_t

PWM ID type.

Definition at line 515 of file qm_soc_regs.h.

4.37.2.10 enum qm_pwm_t

Number of PWM / Timer controllers.

Definition at line 512 of file qm_soc_regs.h.

4.37.2.11 enum qm_rtc_t

Number of RTC controllers.

Definition at line 756 of file qm_soc_regs.h.

4.37.2.12 enum qm_scss_aon_t

Number of SCSS Always-on controllers.

Definition at line 311 of file qm_soc_regs.h.

4.37.2.13 enum qm_spi_t

Number of SPI controllers.

Definition at line 650 of file qm_soc_regs.h.

4.37.2.14 enum qm_uart_t

Number of UART controllers.

Definition at line 598 of file qm_soc_regs.h.

4.37.2.15 enum qm_wdt_t

Number of WDT controllers.

Definition at line 558 of file qm_soc_regs.h.

4.37.3 Variable Documentation

4.37.3.1 qm_i2c_reg_t* qm_i2c[QM_I2C_NUM]

I2C register block.

Definition at line 20 of file qm_i2c.c.

4.38 RAR

Retention alternator regulator for Quark D2000.

Enumerations

- enum rar_state_t { RAR_NORMAL, RAR_RETENTION }
  RAR modes type.

Functions

- int rar_set_mode (const rar_state_t mode)
  Change operating mode of RAR.

4.38.1 Detailed Description

Retention alternator regulator for Quark D2000.

4.38.2 Enumeration Type Documentation

4.38.2.1 enum rar_state_t

RAR modes type.

Enumerator

RAR_NORMAL Normal mode = 50 mA.

RAR_RETENTION Retention mode = 300 uA.

Definition at line 22 of file rar.h.

4.38.3 Function Documentation

4.38.3.1 int rar_set_mode (const rar_state_t *mode*)

Change operating mode of RAR.

Normal mode is able to source up to 50 mA. Retention mode is able to source up to 300 uA. Care must be taken when entering into retention mode to ensure the overall system draw is less than 300 uA.

Parameters

in	*mode*	Operating mode of the RAR.

Returns

Standard errno return type for QMSI.

Return values

0	on success.

Negative	errno for possible error codes.

Definition at line 8 of file rar.c.

References RAR_NORMAL, and RAR_RETENTION.

Referenced by power_soc_deep_sleep(), and power_soc_sleep().

4.39 Quark SE Flash Layout

Flash Layout for Quark SE Microcontrollers.

4.39.1 Detailed Description

Flash Layout for Quark SE Microcontrollers.

4.40 Quark SE SoC Power states

SoC Power mode control for Quark SE Microcontrollers.

Functions

- void power_soc_sleep (void)

  Enter SoC sleep state.
- void power_soc_deep_sleep (void)

  Enter SoC deep sleep state.

4.40.1 Detailed Description

SoC Power mode control for Quark SE Microcontrollers.

4.40.2 Function Documentation

4.40.2.1 void power_soc_deep_sleep (void)

Enter SoC deep sleep state.

Put the SoC into deep sleep state until next SoC wake event.

- Core well is turned off
- Always on well is on
- Hybrid Clock is off
- RTC Clock is on

Possible SoC wake events are:

- Low Power Comparator Interrupt
- AON GPIO Interrupt
- AON Timer Interrupt
- RTC Interrupt

This function puts 1P8V regulators and 3P3V into Linear Mode.

Definition at line 33 of file power_states.c.

References SOCW_EVENT_REGISTER, SOCW_EVENT_SLEEP, SOCW_REG_SLP_CFG, vreg_plat1p8_set_mode(), and vreg_plat3p3_set_mode().

4.40.2.2 void power_soc_sleep (void)

Enter SoC sleep state.

Put the SoC into sleep state until next SoC wake event.

- Core well is turned off
- Always on well is on
- Hybrid Clock is off

- RTC Clock is on

Possible SoC wake events are:

- Low Power Comparator Interrupt
- AON GPIO Interrupt
- AON Timer Interrupt
- RTC Interrupt

Enter SoC sleep state.

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

Definition at line 36 of file power_states.c.

References clk_periph_disable(), CLK_PERIPH_GPIO_DB, CLK_PERIPH_GPIO_REGISTER, CLK_PERIPH_I2C_M0, CLK_PERIPH_I2C_M0_REGISTER, CLK_PERIPH_PWM_REGISTER, CLK_PERIPH_SPI_M0, CLK_PERIPH_SPI_M0_REGISTER, CLK_PERIPH_SPI_S, CLK_PERIPH_SPI_S_REGISTER, CLK_PERIPH_UARTA_REGISTER, CLK_PERIPH_UARTB_REGISTER, CLK_PERIPH_WDT_REGISTER, CLK_SYS_DIV_8, CLK_SYS_HYB_OSC_4MHZ, clk_sys_set_mode(), power_cpu_halt(), QM_ADC_MODE_PWR_DOWN, qm_adc_set_mode(), RAR_NORMAL, RAR_RETENTION, rar_set_mode(), SOCW_EVENT_REGISTER, SOCW_EVENT_SLEEP, SOCW_REG_CCU_LP_CLK_CTL, SOCW_REG_CCU_PERIPH_CLK_GATE_CTL, SOCW_REG_CCU_SYS_CLK_CTL, SOCW_REG_OSC0_CFG1, and SOCW_REG_SLP_CFG.

4.41 Quark SE Host Power states

Host Power mode control for Quark SE Microcontrollers.

Functions

- void power_cpu_c1 (void)
  Enter Host C1 state.
- void power_cpu_c2 (void)
  Enter Host C2 state or SoC LPSS state.
- void power_cpu_c2lp (void)
  Enter Host C2LP state or SoC LPSS state.

4.41.1 Detailed Description

Host Power mode control for Quark SE Microcontrollers.

These functions cannot be called from the Sensor Subsystem.

4.41.2 Function Documentation

4.41.2.1 void power_cpu_c1 (void)

Enter Host C1 state.

Put the Host into C1.

Processor Clock is gated in this state.

Nothing is turned off in this state.

A wake event causes the Host to transition to C0.

A wake event is a host interrupt.

Definition at line 61 of file power_states.c.

References SOCW_EVENT_HALT.

4.41.2.2 void power_cpu_c2 (void)

Enter Host C2 state or SoC LPSS state.

Put the Host into C2. Processor Clock is gated in this state. All rails are supplied.

This enables entry in LPSS if:

- Sensor Subsystem is in SS2.
- LPSS entry is enabled.

If C2 is entered:

- A wake event causes the Host to transition to C0.
- A wake event is a host interrupt.

If LPSS is entered:

- LPSS wake events applies.
- If the Sensor Subsystem wakes the SoC from LPSS, Host is back in C2.

Definition at line 67 of file power_states.c.

References SOCW_EVENT_REGISTER, SOCW_EVENT_SLEEP, and SOCW_REG_CCU_LP_CLK_CTL.

4.41.2.3 void power_cpu_c2lp (void)

Enter Host C2LP state or SoC LPSS state.

Put the Host into C2LP. Processor Complex Clock is gated in this state. All rails are supplied.

This enables entry in LPSS if:

- Sensor Subsystem is in SS2.
- LPSS is allowed.

If C2LP is entered:

- A wake event causes the Host to transition to C0.
- A wake event is a Host interrupt.

If LPSS is entered:

- LPSS wake events apply if LPSS is entered.
- If the Sensor Subsystem wakes the SoC from LPSS, Host transitions back to C2LP.

Definition at line 77 of file power_states.c.

References SOCW_EVENT_REGISTER, SOCW_EVENT_SLEEP, and SOCW_REG_CCU_LP_CLK_CTL.

4.42 SoC Registers (Sensor Subsystem)

Quark SE SoC Sensor Subsystem Registers.

SS Timer

- enum **qm_ss_timer_reg_t**
- enum qm_ss_timer_t

  Sensor Subsystem Timers.

SS GPIO

GPIO registers and definitions.

- enum qm_ss_gpio_reg_t

  Sensor Subsystem GPIO register block type.
- enum qm_ss_gpio_t

  Sensor Subsystem GPIO.

SS I2C

I2C registers and definitions.

- enum qm_ss_i2c_reg_t

  Sensor Subsystem I2C register block type.
- enum qm_ss_i2c_t

  Sensor Subsystem I2C.

- enum qm_ss_adc_reg_t {
  QM_SS_ADC_SET = 0, QM_SS_ADC_DIVSEQSTAT, QM_SS_ADC_SEQ, QM_SS_ADC_CTRL,
  QM_SS_ADC_INTSTAT, QM_SS_ADC_SAMPLE }

  Sensor Subsystem ADC.
- enum qm_ss_adc_t { QM_SS_ADC_0 = 0 }

  Sensor Subsystem ADC.

SS CREG

CREG Registers.

- enum qm_ss_creg_reg_t { QM_SS_IO_CREG_MST0_CTRL = 0x0, QM_SS_IO_CREG_SLV0_OBSR = 0x80, QM_SS_IO_CREG_SLV1_OBSR = 0x180 }

SS Interrupt

IRQs and interrupt vectors.

- enum **qm_ss_irq_priority_t**
- enum **qm_ss_irq_mask_t**
- enum **qm_ss_irq_trigger_t**

SS SPI

I2C registers and definitions.

- enum qm_ss_spi_reg_t { QM_SS_SPI_CTRL = 0, QM_SS_SPI_SPIEN = 2, QM_SS_SPI_TIMING = 4, QM_SS_SPI_FTLR, QM_SS_SPI_TXFLR = 7, QM_SS_SPI_RXFLR, QM_SS_SPI_SR, QM_SS_SPI_INTR_STAT, QM_SS_SPI_INTR_MASK, QM_SS_SPI_CLR_INTR, QM_SS_SPI_DR }

  Sensor Subsystem SPI register map.
- enum qm_ss_spi_t { QM_SS_SPI_0 = 0, QM_SS_SPI_1 }

  Sensor Subsystem SPI modules.

4.42.1 Detailed Description

Quark SE SoC Sensor Subsystem Registers. For detailed description please read the SOC datasheet.

4.42.2 Enumeration Type Documentation

4.42.2.1 enum qm_ss_adc_reg_t

Sensor Subsystem ADC.

Sensor Subsystem ADC registers

Enumerator

QM_SS_ADC_SET ADC and sequencer settings register.

QM_SS_ADC_DIVSEQSTAT ADC clock and sequencer status register.

QM_SS_ADC_SEQ ADC sequence entry register.

QM_SS_ADC_CTRL ADC control register.

QM_SS_ADC_INTSTAT ADC interrupt status register.

QM_SS_ADC_SAMPLE ADC sample register.

Definition at line 235 of file qm_sensor_regs.h.

4.42.2.2 enum qm_ss_adc_t

Sensor Subsystem ADC.

Enumerator

QM_SS_ADC_0 ADC first module.

Definition at line 245 of file qm_sensor_regs.h.

4.42.2.3 enum qm_ss_creg_reg_t

Enumerator

QM_SS_IO_CREG_MST0_CTRL Master control register.

QM_SS_IO_CREG_SLV0_OBSR Slave control register.

QM_SS_IO_CREG_SLV1_OBSR Slave control register.

Definition at line 307 of file qm_sensor_regs.h.

4.42.2.4 enum qm_ss_gpio_reg_t

Sensor Subsystem GPIO register block type.

Definition at line 108 of file qm_sensor_regs.h.

4.42.2.5 enum qm_ss_gpio_t

Sensor Subsystem GPIO.

Definition at line 127 of file qm_sensor_regs.h.

4.42.2.6 enum qm_ss_i2c_reg_t

Sensor Subsystem I2C register block type.

Definition at line 142 of file qm_sensor_regs.h.

4.42.2.7 enum qm_ss_spi_reg_t

Sensor Subsystem SPI register map.

Enumerator

QM_SS_SPI_CTRL SPI control register.

QM_SS_SPI_SPIEN SPI enable register.

QM_SS_SPI_TIMING SPI serial clock divider value.

QM_SS_SPI_FTLR Threshold value for TX/RX FIFO.

QM_SS_SPI_TXFLR Number of valid data entries in TX FIFO.

QM_SS_SPI_RXFLR Number of valid data entries in RX FIFO.

QM_SS_SPI_SR SPI status register.

QM_SS_SPI_INTR_STAT Interrupt status register.

QM_SS_SPI_INTR_MASK Interrupt mask register.

QM_SS_SPI_CLR_INTR Interrupt clear register.

QM_SS_SPI_DR RW buffer for FIFOs.

Definition at line 456 of file qm_sensor_regs.h.

4.42.2.8 enum qm_ss_spi_t

Sensor Subsystem SPI modules.

Enumerator

QM_SS_SPI_0 SPI module 0.

QM_SS_SPI_1 SPI module 1.

Definition at line 471 of file qm_sensor_regs.h.

4.43 SoC Registers (SE)

Quark SE SoC Registers.

Data Structures

- struct qm_scss_ccu_reg_t
  System Core register map.
- struct qm_scss_gp_reg_t
  General Purpose register map.
- struct qm_scss_mem_reg_t
  Memory Control register map.
- struct qm_scss_cmp_reg_t
  Comparator register map.
- struct qm_lapic_reg_t
  APIC register block type.
- struct int_ss_i2c_reg_t
  SS I2C Interrupt register map.
- struct int_ss_spi_reg_t
  SS SPI Interrupt register map.
- struct qm_scss_int_reg_t
  Interrupt register map.
- struct qm_scss_pmu_reg_t
  Power Management register map.
- struct qm_scss_ss_reg_t
  Sensor Subsystem register map.
- struct qm_scss_aon_reg_t
  Always-on Controller register map.
- struct qm_scss_peripheral_reg_t
  Peripheral Registers register map.
- struct qm_scss_pmux_reg_t
  Pin MUX register map.
- struct qm_scss_info_reg_t
  Information register map.
- struct qm_mailbox_t
  Mailbox register structure.
- struct qm_scss_mailbox_reg_t
  Mailbox register map.
- struct qm_pwm_channel_t
  PWM / Timer channel register map.
- struct qm_pwm_reg_t
  PWM / Timer register map.
- struct qm_wdt_reg_t
  Watchdog timer register map.
- struct qm_uart_reg_t
  UART register map.
- struct qm_spi_reg_t
  SPI register map.
- struct qm_rtc_reg_t
  RTC register map.
- struct qm_i2c_reg_t

Peripheral Registers

- qm_scss_peripheral_reg_t **test_scss_peripheral**

Pin MUX

- qm_scss_pmux_reg_t **test_scss_pmux**

ID

- qm_scss_info_reg_t **test_scss_info**

Mailbox

- qm_scss_mailbox_reg_t **test_scss_mailbox**

PWM / Timer

- enum qm_pwm_t

  Number of PWM / Timer controllers.
- enum qm_pwm_id_t

  PWM ID type.
- qm_pwm_reg_t **test_pwm_t**

WDT

- enum qm_wdt_t

  Number of WDT controllers.
- qm_wdt_reg_t **test_wdt**

UART

- enum qm_uart_t

  Number of UART controllers.
- qm_uart_reg_t **test_uart_instance**
- qm_uart_reg_t * **test_uart** [QM_UART_NUM]
- qm_uart_reg_t * **qm_uart** [QM_UART_NUM]

SPI

- enum qm_spi_t

  Number of SPI controllers.
- qm_spi_reg_t **test_spi**
- qm_spi_reg_t * **test_spi_controllers** [QM_SPI_NUM]
- qm_spi_reg_t * qm_spi_controllers [QM_SPI_NUM]

  Extern qm_spi_reg_t array declared at qm_soc_regs.h .*

RTC

- enum qm_rtc_t

  Number of RTC controllers.
- qm_rtc_reg_t **test_rtc**

I2C

- enum qm_i2c_t

  Number of I2C controllers.
- qm_i2c_reg_t **test_i2c_instance** [QM_I2C_NUM]
- qm_i2c_reg_t * **test_i2c** [QM_I2C_NUM]
- qm_i2c_reg_t * qm_i2c [QM_I2C_NUM]

  I2C register block.

GPIO

- enum qm_gpio_t

  Number of GPIO controllers.
- qm_gpio_reg_t **test_gpio_instance**
- qm_gpio_reg_t * **test_gpio** [QM_GPIO_NUM]
- qm_gpio_reg_t * qm_gpio [QM_GPIO_NUM]

  GPIO register block.

Flash

- enum qm_flash_t

  Number of Flash controllers.
- qm_flash_reg_t **test_flash_instance**
- qm_flash_reg_t * **test_flash** [QM_FLASH_NUM]
- uint8_t **test_flash_page** [0x800]
- qm_flash_reg_t * **qm_flash** [QM_FLASH_NUM]

Memory Protection Region

- qm_mpr_reg_t **test_mpr**

Peripheral Clock

- enum clk_periph_t {
  CLK_PERIPH_REGISTER = BIT(0), CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_PERIPH_SPI_S = BIT(4),
  CLK_PERIPH_SPI_M0 = BIT(5), CLK_PERIPH_GPIO_INTERRUPT = BIT(7), CLK_PERIPH_GPIO_DB = BIT(8), CLK_PERIPH_WDT_REGISTER = BIT(10),
  CLK_PERIPH_RTC_REGISTER = BIT(11), CLK_PERIPH_PWM_REGISTER = BIT(12), CLK_PERIPH_GPIO_REGISTER = BIT(13), CLK_PERIPH_SPI_M0_REGISTER,
  CLK_PERIPH_SPI_S_REGISTER, CLK_PERIPH_UARTA_REGISTER = BIT(17), CLK_PERIPH_UARTB_REGISTER = BIT(18), CLK_PERIPH_I2C_M0_REGISTER,
  CLK_PERIPH_ADC = BIT(22), CLK_PERIPH_ADC_REGISTER = BIT(23), CLK_PERIPH_ALL = 0xCFFFFF,
  CLK_PERIPH_REGISTER = BIT(0),
  CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_PERIPH_I2C_M1 = BIT(3), CLK_PERIPH_SPI_S = BIT(4),
  CLK_PERIPH_SPI_M0 = BIT(5), CLK_PERIPH_SPI_M1 = BIT(6), CLK_PERIPH_GPIO_INTERRUPT = BIT(7), CLK_PERIPH_GPIO_DB = BIT(8),
  CLK_PERIPH_I2S = BIT(9), CLK_PERIPH_WDT_REGISTER = BIT(10), CLK_PERIPH_RTC_REGISTER = BIT(11), CLK_PERIPH_PWM_REGISTER = BIT(12),
  CLK_PERIPH_GPIO_REGISTER = BIT(13), CLK_PERIPH_SPI_M0_REGISTER, CLK_PERIPH_SPI_M1_REGISTER, CLK_PERIPH_SPI_S_REGISTER,
  CLK_PERIPH_UARTA_REGISTER = BIT(17), CLK_PERIPH_UARTB_REGISTER = BIT(18), CLK_PERIPH_I2C_M0_REGISTER, CLK_PERIPH_I2C_M1_REGISTER,
  CLK_PERIPH_I2S_REGISTER = BIT(21), CLK_PERIPH_ALL = 0x3FFFFF }

Peripheral clock type.

DMA

- enum qm_dma_t { QM_DMA_0, QM_DMA_NUM, QM_DMA_0, QM_DMA_NUM }

  DMA instances.
- enum qm_dma_channel_id_t {
  QM_DMA_CHANNEL_0 = 0, QM_DMA_CHANNEL_1, QM_DMA_CHANNEL_NUM, QM_DMA_CHANNEL_0 = 0,
  QM_DMA_CHANNEL_1, QM_DMA_CHANNEL_2, QM_DMA_CHANNEL_3, QM_DMA_CHANNEL_4,
  QM_DMA_CHANNEL_5, QM_DMA_CHANNEL_6, QM_DMA_CHANNEL_7, QM_DMA_CHANNEL_NUM }

  DMA channel IDs.
- enum qm_dma_handshake_interface_t {
  DMA_HW_IF_UART_A_TX = 0x0, DMA_HW_IF_UART_A_RX = 0x1, DMA_HW_IF_UART_B_TX = 0x2, DMA_HW_IF_UART_B_RX = 0x3,
  DMA_HW_IF_SPI_MASTER_0_TX = 0x4, DMA_HW_IF_SPI_MASTER_0_RX = 0x5, DMA_HW_IF_SPI_SLAVE_TX = 0x8, DMA_HW_IF_SPI_SLAVE_RX = 0x9,
  DMA_HW_IF_I2C_MASTER_0_TX = 0xc, DMA_HW_IF_I2C_MASTER_0_RX = 0xd, DMA_HW_IF_UART_A_TX = 0x0, DMA_HW_IF_UART_A_RX = 0x1,
  DMA_HW_IF_UART_B_TX = 0x2, DMA_HW_IF_UART_B_RX = 0x3, DMA_HW_IF_SPI_MASTER_0_TX = 0x4, DMA_HW_IF_SPI_MASTER_0_RX = 0x5,
  DMA_HW_IF_SPI_MASTER_1_TX = 0x6, DMA_HW_IF_SPI_MASTER_1_RX = 0x7, DMA_HW_IF_SPI_SLAVE_TX = 0x8, DMA_HW_IF_SPI_SLAVE_RX = 0x9,
  DMA_HW_IF_I2S_PLAYBACK = 0xa, DMA_HW_IF_I2S_CAPTURE = 0xb, DMA_HW_IF_I2C_MASTER_0_TX = 0xc, DMA_HW_IF_I2C_MASTER_0_RX = 0xd,
  DMA_HW_IF_I2C_MASTER_1_TX = 0xe, DMA_HW_IF_I2C_MASTER_1_RX = 0xf }

  DMA hardware handshake interfaces.
- qm_dma_reg_t **test_dma_instance** [QM_DMA_NUM]
- qm_dma_reg_t * **test_dma** [QM_DMA_NUM]
- qm_dma_reg_t * **qm_dma** [QM_DMA_NUM]

Versioning

- uint32_t **test_rom_version**

4.43.1 Detailed Description

Quark SE SoC Registers.

4.43.2 Enumeration Type Documentation

4.43.2.1 enum clk_periph_t

Peripheral clock type.

Enumerator

CLK_PERIPH_REGISTER Peripheral Clock Gate Enable.

CLK_PERIPH_CLK Peripheral Clock Enable.

CLK_PERIPH_I2C_M0 I2C Master 0 Clock Enable.

CLK_PERIPH_SPI_S SPI Slave Clock Enable.

CLK_PERIPH_SPI_M0 SPI Master 0 Clock Enable.

CLK_PERIPH_GPIO_INTERRUPT GPIO Interrupt Clock Enable.

CLK_PERIPH_GPIO_DB GPIO Debounce Clock Enable.

CLK_PERIPH_WDT_REGISTER Watchdog Clock Enable.

CLK_PERIPH_RTC_REGISTER RTC Clock Gate Enable.

CLK_PERIPH_PWM_REGISTER PWM Clock Gate Enable.

CLK_PERIPH_GPIO_REGISTER GPIO Clock Gate Enable.

CLK_PERIPH_SPI_M0_REGISTER SPI Master 0 Clock Gate Enable.

CLK_PERIPH_SPI_S_REGISTER SPI Slave Clock Gate Enable.

CLK_PERIPH_UARTA_REGISTER UARTA Clock Gate Enable.

CLK_PERIPH_UARTB_REGISTER UARTB Clock Gate Enable.

CLK_PERIPH_I2C_M0_REGISTER I2C Master 0 Clock Gate Enable.

CLK_PERIPH_ADC ADC Clock Enable.

CLK_PERIPH_ADC_REGISTER ADC Clock Gate Enable.

CLK_PERIPH_ALL Quark D2000 peripherals Enable.

CLK_PERIPH_REGISTER Peripheral Clock Gate Enable.

CLK_PERIPH_CLK Peripheral Clock Enable.

CLK_PERIPH_I2C_M0 I2C Master 0 Clock Enable.

CLK_PERIPH_I2C_M1 I2C Master 1 Clock Enable.

CLK_PERIPH_SPI_S SPI Slave Clock Enable.

CLK_PERIPH_SPI_M0 SPI Master 0 Clock Enable.

CLK_PERIPH_SPI_M1 SPI Master 1 Clock Enable.

CLK_PERIPH_GPIO_INTERRUPT GPIO Interrupt Clock Enable.

CLK_PERIPH_GPIO_DB GPIO Debounce Clock Enable.

CLK_PERIPH_I2S I2S Clock Enable.

CLK_PERIPH_WDT_REGISTER Watchdog Clock Enable.

CLK_PERIPH_RTC_REGISTER RTC Clock Gate Enable.

CLK_PERIPH_PWM_REGISTER PWM Clock Gate Enable.

CLK_PERIPH_GPIO_REGISTER GPIO Clock Gate Enable.

CLK_PERIPH_SPI_M0_REGISTER SPI Master 0 Clock Gate Enable.

CLK_PERIPH_SPI_M1_REGISTER SPI Master 1 Clock Gate Enable.

CLK_PERIPH_SPI_S_REGISTER SPI Slave Clock Gate Enable.

CLK_PERIPH_UARTA_REGISTER UARTA Clock Gate Enable.

CLK_PERIPH_UARTB_REGISTER UARTB Clock Gate Enable.

CLK_PERIPH_I2C_M0_REGISTER I2C Master 0 Clock Gate Enable.

CLK_PERIPH_I2C_M1_REGISTER I2C Master 1 Clock Gate Enable.

CLK_PERIPH_I2S_REGISTER I2S Clock Gate Enable.

CLK_PERIPH_ALL Quark SE peripherals Mask.

Definition at line 1354 of file qm_soc_regs.h.

4.43.2.2 enum qm_dma_channel_id_t

DMA channel IDs.

Enumerator

QM_DMA_CHANNEL_0 DMA channel id for channel 0.

QM_DMA_CHANNEL_1 DMA channel id for channel 1.

QM_DMA_CHANNEL_NUM Number of DMA channels.

QM_DMA_CHANNEL_0 DMA channel id for channel 0.

QM_DMA_CHANNEL_1 DMA channel id for channel 1.

QM_DMA_CHANNEL_2 DMA channel id for channel 2.

QM_DMA_CHANNEL_3 DMA channel id for channel 3.

QM_DMA_CHANNEL_4 DMA channel id for channel 4.

QM_DMA_CHANNEL_5 DMA channel id for channel 5.

QM_DMA_CHANNEL_6 DMA channel id for channel 6.

QM_DMA_CHANNEL_7 DMA channel id for channel 7.

QM_DMA_CHANNEL_NUM Number of DMA channels.

Definition at line 1406 of file qm_soc_regs.h.

4.43.2.3 enum qm_dma_handshake_interface_t

DMA hardware handshake interfaces.

Enumerator

DMA_HW_IF_UART_A_TX UART_A_TX.

DMA_HW_IF_UART_A_RX UART_A_RX.

DMA_HW_IF_UART_B_TX UART_B_TX.

DMA_HW_IF_UART_B_RX UART_B_RX.

DMA_HW_IF_SPI_MASTER_0_TX SPI_Master_0_TX.

DMA_HW_IF_SPI_MASTER_0_RX SPI_Master_0_RX.

DMA_HW_IF_SPI_SLAVE_TX SPI_Slave_TX.

DMA_HW_IF_SPI_SLAVE_RX SPI_Slave_RX.

DMA_HW_IF_I2C_MASTER_0_TX I2C_Master_0_TX.

DMA_HW_IF_I2C_MASTER_0_RX I2C_Master_0_RX.

DMA_HW_IF_UART_A_TX UART_A_TX.

DMA_HW_IF_UART_A_RX UART_A_RX.

DMA_HW_IF_UART_B_TX UART_B_TX.

DMA_HW_IF_UART_B_RX UART_B_RX.

DMA_HW_IF_SPI_MASTER_0_TX SPI_Master_0_TX.

DMA_HW_IF_SPI_MASTER_0_RX SPI_Master_0_RX.

DMA_HW_IF_SPI_MASTER_1_TX SPI_Master_1_TX.

DMA_HW_IF_SPI_MASTER_1_RX SPI_Master_1_RX.

DMA_HW_IF_SPI_SLAVE_TX SPI_Slave_TX.

DMA_HW_IF_SPI_SLAVE_RX SPI_Slave_RX.

DMA_HW_IF_I2S_PLAYBACK I2S_Playback channel.

DMA_HW_IF_I2S_CAPTURE I2S_Capture channel.

DMA_HW_IF_I2C_MASTER_0_TX I2C_Master_0_TX.

DMA_HW_IF_I2C_MASTER_0_RX I2C_Master_0_RX.

DMA_HW_IF_I2C_MASTER_1_TX I2C_Master_1_TX.

DMA_HW_IF_I2C_MASTER_1_RX I2C_Master_1_RX.

Definition at line 1419 of file qm_soc_regs.h.

4.43.2.4 enum qm_dma_t

DMA instances.

Enumerator

QM_DMA_0 DMA controller id.

QM_DMA_NUM Number of DMA controllers.

QM_DMA_0 DMA controller id.

QM_DMA_NUM Number of DMA controllers.

Definition at line 1400 of file qm_soc_regs.h.

4.43.2.5 enum qm_flash_t

Number of Flash controllers.

Definition at line 1246 of file qm_soc_regs.h.

4.43.2.6 enum qm_gpio_t

Number of GPIO controllers.

Definition at line 1193 of file qm_soc_regs.h.

4.43.2.7 enum qm_i2c_t

Number of I2C controllers.

Definition at line 1057 of file qm_soc_regs.h.

4.43.2.8 enum qm_pwm_id_t

PWM ID type.

Definition at line 781 of file qm_soc_regs.h.

4.43.2.9 enum qm_pwm_t

Number of PWM / Timer controllers.

Definition at line 778 of file qm_soc_regs.h.

4.43.2.10 enum qm_rtc_t

Number of RTC controllers.

Definition at line 1023 of file qm_soc_regs.h.

4.43.2.11 enum qm_scss_aon_t

Number of SCSS Always-on controllers.

Definition at line 492 of file qm_soc_regs.h.

4.43.2.12 enum qm_spi_t

Number of SPI controllers.

Definition at line 914 of file qm_soc_regs.h.

4.43.2.13 enum qm_uart_t

Number of UART controllers.

Definition at line 870 of file qm_soc_regs.h.

4.43.2.14 enum qm_wdt_t

Number of WDT controllers.

Definition at line 830 of file qm_soc_regs.h.

4.43.3 Variable Documentation

4.43.3.1 qm_i2c_reg_t* qm_i2c[QM_I2C_NUM]

I2C register block.

Definition at line 20 of file qm_i2c.c.

4.44 Quark SE Voltage Regulators

Voltage Regulators Control.

Functions

- int vreg_aon_set_mode (const vreg_mode_t mode)
  Set AON Voltage Regulator mode.
- int vreg_plat3p3_set_mode (const vreg_mode_t mode)
  Set Platform 3P3 Voltage Regulator mode.
- int vreg_plat1p8_set_mode (const vreg_mode_t mode)
  Set Platform 1P8 Voltage Regulator mode.
- int vreg_host_set_mode (const vreg_mode_t mode)
  Set Host Voltage Regulator mode.

4.44.1 Detailed Description

Voltage Regulators Control.

4.44.2 Function Documentation

4.44.2.1 int vreg_aon_set_mode (const vreg_mode_t *mode*)

Set AON Voltage Regulator mode.

The AON Voltage Regulator is not a switching regulator and only acts as a linear regulator. VREG_SWITCHING_-MODE is not a value mode for the AON Voltage Regulator.

Parameters

in	*mode*	Voltage Regulator mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 45 of file vreg.c.

4.44.2.2 int vreg_host_set_mode (const vreg_mode_t *mode*)

Set Host Voltage Regulator mode.

Parameters

in	*mode*	Voltage Regulator mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 65 of file vreg.c.

4.44.2.3 int vreg_plat1p8_set_mode (const vreg_mode_t *mode*)

Set Platform 1P8 Voltage Regulator mode.

Parameters

in	*mode*	Voltage Regulator mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 59 of file vreg.c.

Referenced by power_soc_deep_sleep().

4.44.2.4 int vreg_plat3p3_set_mode (const vreg_mode_t *mode*)

Set Platform 3P3 Voltage Regulator mode.

Parameters

in	*mode*	Voltage Regulator mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 53 of file vreg.c.

Referenced by power_soc_deep_sleep().

4.45 Syscalls

newlib syscalls implementation

Functions

- int pico_printf (const char *format,...)

  This is an minimally useful subset of the POSIX printf() function.

4.45.1 Detailed Description

newlib syscalls implementation

4.45.2 Function Documentation

4.45.2.1 int pico_printf (const char * *format,* ...)

This is an minimally useful subset of the POSIX printf() function.

To reduce code size, this pico_printf() implementation only supports a few conversion specifiers:

- *'d'*, *'u'*: for signed and unsigned decimal numbers, respectively;
- *'x'*, *'X'*: for hexadecimal numbers, downcase and upcase;
- *'s'*: for NULL terminated strings.

Other limitations:

- No flag specifier is implemented;
- No field width specifier is implemented;
- The only supported length modifier is 'l', which is parsed and ignored, on supported archictetures 'int' and 'long int' are both 32 bits long.
- 32 digits maximum length for formatted numbers.

Definition at line 105 of file newlib-syscalls.c.

5 Data Structure Documentation

5.1 int_ss_i2c_reg_t Struct Reference

SS I2C Interrupt register map.

`#include <qm_soc_regs.h>`

5.1.1 Detailed Description

SS I2C Interrupt register map.

Definition at line 302 of file qm_soc_regs.h.

5.2 int_ss_spi_reg_t Struct Reference

SS SPI Interrupt register map.

`#include <qm_soc_regs.h>`

5.2.1 Detailed Description

SS SPI Interrupt register map.

Definition at line 310 of file qm_soc_regs.h.

5.3 mvic_reg_pad_t Struct Reference

MVIC register structure.

`#include <qm_soc_regs.h>`

5.3.1 Detailed Description

MVIC register structure.

Definition at line 1224 of file qm_soc_regs.h.

5.4 pic_timer_reg_pad_t Struct Reference

PIC timer register structure.

`#include <qm_soc_regs.h>`

5.4.1 Detailed Description

PIC timer register structure.

Definition at line 1152 of file qm_soc_regs.h.

5.5 qm_ac_config_t Struct Reference

Analog Comparator configuration type.

`#include <qm_comparator.h>`

Data Fields

- uint32_t int_en

  Interrupt enable.
- uint32_t reference

  Reference voltage, 1b: VREF; 0b: AR_PIN.
- uint32_t polarity

  0b: input>ref; 1b: input<ref
- uint32_t power

  1b: Normal mode; 0b:Power-down/Shutdown mode
- void(* callback)(void *data, uint32_t int_status)

  Transfer callback.
- void * callback_data

  Callback user data.

5.5.1 Detailed Description

Analog Comparator configuration type.

Each bit in the registers controls a single Analog Comparator pin.

Definition at line 23 of file qm_comparator.h.

5.5.2 Field Documentation

5.5.2.1 void(* qm_ac_config_t::callback)(void *data, uint32_t int_status)

Transfer callback.

Parameters

in	*data*	Callback user data.
in	*status*	Comparator interrupt status.

Definition at line 35 of file qm_comparator.h.

Referenced by qm_ac_set_config().

5.5.2.2 void* qm_ac_config_t::callback_data

Callback user data.

Definition at line 36 of file qm_comparator.h.

Referenced by qm_ac_set_config().

5.5.2.3 uint32_t qm_ac_config_t::int_en

Interrupt enable.

Definition at line 24 of file qm_comparator.h.

Referenced by qm_ac_set_config().

5.5.2.4 uint32_t qm_ac_config_t::reference

Reference voltage, 1b: VREF; 0b: AR_PIN.

Definition at line 25 of file qm_comparator.h.

Referenced by qm_ac_set_config().

5.6 qm_adc_config_t Struct Reference

ADC configuration type.

`#include <qm_adc.h>`

Data Fields

- uint8_t window

  Sample interval in ADC clock cycles, defines the period to wait between the start of each sample and can be in the range [(resolution+2) - 255].
- qm_adc_resolution_t resolution

  12, 10, 8, 6-bit resolution.

5.6.1 Detailed Description

ADC configuration type.

Definition at line 94 of file qm_adc.h.

5.6.2 Field Documentation

5.6.2.1 qm_adc_resolution_t qm_adc_config_t::resolution

12, 10, 8, 6-bit resolution.

Definition at line 101 of file qm_adc.h.

Referenced by qm_adc_set_config().

5.7 qm_adc_reg_t Struct Reference

ADC register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t adc_seq0

  ADC Channel Sequence Table Entry 0.
- QM_RW uint32_t adc_seq1

  ADC Channel Sequence Table Entry 1.
- QM_RW uint32_t adc_seq2

  ADC Channel Sequence Table Entry 2.
- QM_RW uint32_t adc_seq3

  ADC Channel Sequence Table Entry 3.
- QM_RW uint32_t adc_seq4

  ADC Channel Sequence Table Entry 4.
- QM_RW uint32_t adc_seq5

  ADC Channel Sequence Table Entry 5.
- QM_RW uint32_t adc_seq6

  ADC Channel Sequence Table Entry 6.
- QM_RW uint32_t adc_seq7

  ADC Channel Sequence Table Entry 7.

- QM_RW uint32_t adc_cmd
  ADC Command Register.
- QM_RW uint32_t adc_intr_status
  ADC Interrupt Status Register.
- QM_RW uint32_t adc_intr_enable
  ADC Interrupt Enable Register.
- QM_RW uint32_t adc_sample
  ADC Sample Register.
- QM_RW uint32_t adc_calibration
  ADC Calibration Data Register.
- QM_RW uint32_t adc_fifo_count
  ADC FIFO Count Register.
- QM_RW uint32_t adc_op_mode
  ADC Operating Mode Register.

5.7.1 Detailed Description

ADC register map.

Definition at line 979 of file qm_soc_regs.h.

5.8 qm_adc_xfer_t Struct Reference

ADC transfer type.

`#include <qm_adc.h>`

Data Fields

- qm_adc_channel_t * ch
  Channel sequence array (1-32 channels).
- uint8_t ch_len
  Number of channels in the above array.
- qm_adc_sample_t * samples
  Array to store samples.
- uint32_t samples_len
  Length of sample array.
- void(* callback)(void *data, int error, qm_adc_status_t status, qm_adc_cb_source_t source)
  Transfer callback.
- void * callback_data
  Callback user data.

5.8.1 Detailed Description

ADC transfer type.

Definition at line 107 of file qm_adc.h.

5.8.2 Field Documentation

5.8.2.1 void(* qm_adc_xfer_t::callback)(void *data, int error, qm_adc_status_t status, qm_adc_cb_source_t source)

Transfer callback.

Called when a conversion is performed or an error is detected.

Parameters

in	*data*	The callback user data.
in	*error*	0 on success. Negative errno for possible error codes.
in	*status*	ADC status.
in	*source*	Interrupt callback source.

Definition at line 124 of file qm_adc.h.

5.8.2.2 void* qm_adc_xfer_t::callback_data

Callback user data.

Definition at line 126 of file qm_adc.h.

5.8.2.3 qm_adc_channel_t* qm_adc_xfer_t::ch

Channel sequence array (1-32 channels).

Definition at line 108 of file qm_adc.h.

Referenced by qm_adc_convert(), and qm_adc_irq_convert().

5.8.2.4 uint8_t qm_adc_xfer_t::ch_len

Number of channels in the above array.

Definition at line 109 of file qm_adc.h.

Referenced by qm_adc_convert(), and qm_adc_irq_convert().

5.8.2.5 qm_adc_sample_t* qm_adc_xfer_t::samples

Array to store samples.

Definition at line 110 of file qm_adc.h.

Referenced by qm_adc_convert(), and qm_adc_irq_convert().

5.8.2.6 uint32_t qm_adc_xfer_t::samples_len

Length of sample array.

Definition at line 111 of file qm_adc.h.

Referenced by qm_adc_convert(), and qm_adc_irq_convert().

5.9 qm_aonpt_config_t Struct Reference

Always-on Periodic Timer configuration type.

`#include <qm_aon_counters.h>`

Data Fields

- uint32_t count
  Time to count down from in clock cycles.
- bool int_en
  Enable/disable the interrupts.
- void(* callback)(void *data)
  User callback.
- void * callback_data
  Callback data.

5.9.1 Detailed Description

Always-on Periodic Timer configuration type.

Definition at line 21 of file qm_aon_counters.h.

5.9.2 Field Documentation

5.9.2.1 void(∗ qm_aonpt_config_t::callback)(void ∗data)

User callback.

Parameters

in	*data*	User defined data.

Definition at line 30 of file qm_aon_counters.h.

Referenced by qm_aonpt_set_config().

5.9.2.2 void∗ qm_aonpt_config_t::callback_data

Callback data.

Definition at line 31 of file qm_aon_counters.h.

Referenced by qm_aonpt_set_config().

5.9.2.3 uint32_t qm_aonpt_config_t::count

Time to count down from in clock cycles.

Definition at line 22 of file qm_aon_counters.h.

Referenced by qm_aonpt_set_config().

5.9.2.4 bool qm_aonpt_config_t::int_en

Enable/disable the interrupts.

Definition at line 23 of file qm_aon_counters.h.

Referenced by qm_aonpt_set_config().

5.10 qm_dma_chan_reg_t Struct Reference

DMA channel register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t sar_low
  SAR.
- QM_RW uint32_t sar_high
  SAR.
- QM_RW uint32_t dar_low
  DAR.
- QM_RW uint32_t dar_high
  DAR.
- QM_RW uint32_t llp_low

LLP.

- QM_RW uint32_t llp_high

  LLP.

- QM_RW uint32_t ctrl_low

  CTL.

- QM_RW uint32_t ctrl_high

  CTL.

- QM_RW uint32_t src_stat_low

  SSTAT.

- QM_RW uint32_t src_stat_high

  SSTAT.

- QM_RW uint32_t dst_stat_low

  DSTAT.

- QM_RW uint32_t dst_stat_high

  DSTAT.

- QM_RW uint32_t src_stat_addr_low

  SSTATAR.

- QM_RW uint32_t src_stat_addr_high

  SSTATAR.

- QM_RW uint32_t dst_stat_addr_low

  DSTATAR.

- QM_RW uint32_t dst_stat_addr_high

  DSTATAR.

- QM_RW uint32_t cfg_low

  CFG.

- QM_RW uint32_t cfg_high

  CFG.

- QM_RW uint32_t src_sg_low

  SGR.

- QM_RW uint32_t src_sg_high

  SGR.

- QM_RW uint32_t dst_sg_low

  DSR.

- QM_RW uint32_t dst_sg_high

  DSR.

5.10.1 Detailed Description

DMA channel register map.

Definition at line 1320 of file qm_soc_regs.h.

5.11 qm_dma_channel_config_t Struct Reference

DMA channel configuration structure.

```
#include <qm_dma.h>
```

Data Fields

- qm_dma_handshake_interface_t handshake_interface
  DMA channel handshake interface ID.
- qm_dma_handshake_polarity_t handshake_polarity
  DMA channel handshake polarity.
- qm_dma_channel_direction_t channel_direction
  DMA channel direction.
- qm_dma_transfer_width_t source_transfer_width
  DMA source transfer width.
- qm_dma_transfer_width_t destination_transfer_width
  DMA destination transfer width.
- qm_dma_burst_length_t source_burst_length
  DMA source burst length.
- qm_dma_burst_length_t destination_burst_length
  DMA destination burst length.
- void(∗ client_callback)(void ∗callback_context, uint32_t len, int error_code)
  Client callback for DMA transfer ISR.
- void ∗ callback_context
  DMA client context passed to the callbacks.

5.11.1 Detailed Description

DMA channel configuration structure.

Definition at line 66 of file qm_dma.h.

5.11.2 Field Documentation

5.11.2.1 void(∗ qm_dma_channel_config_t::client_callback)(void ∗callback_context, uint32_t len, int error_code)

Client callback for DMA transfer ISR.

Parameters

in	*callback_context*	DMA client context.
in	*len*	Data length transferred.
in	*error*	Error code.

Definition at line 95 of file qm_dma.h.

Referenced by qm_dma_channel_set_config(), qm_i2c_dma_channel_config(), qm_spi_dma_channel_config(), and qm_uart_dma_channel_config().

5.12 qm_dma_int_reg_t Struct Reference

DMA interrupt register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t raw_tfr_low
  RawTfr.
- QM_RW uint32_t raw_tfr_high

RawTfr.

- QM_RW uint32_t raw_block_low

  RawBlock.
- QM_RW uint32_t raw_block_high

  RawBlock.
- QM_RW uint32_t raw_src_trans_low

  RawSrcTran.
- QM_RW uint32_t raw_src_trans_high

  RawSrcTran.
- QM_RW uint32_t raw_dst_trans_low

  RawDstTran.
- QM_RW uint32_t raw_dst_trans_high

  RawDstTran.
- QM_RW uint32_t raw_err_low

  RawErr.
- QM_RW uint32_t raw_err_high

  RawErr.
- QM_RW uint32_t status_tfr_low

  StatusTfr.
- QM_RW uint32_t status_tfr_high

  StatusTfr.
- QM_RW uint32_t status_block_low

  StatusBlock.
- QM_RW uint32_t status_block_high

  StatusBlock.
- QM_RW uint32_t status_src_trans_low

  StatusSrcTran.
- QM_RW uint32_t status_src_trans_high

  StatusSrcTran.
- QM_RW uint32_t status_dst_trans_low

  StatusDstTran.
- QM_RW uint32_t status_dst_trans_high

  StatusDstTran.
- QM_RW uint32_t status_err_low

  StatusErr.
- QM_RW uint32_t status_err_high

  StatusErr.
- QM_RW uint32_t mask_tfr_low

  MaskTfr.
- QM_RW uint32_t mask_tfr_high

  MaskTfr.
- QM_RW uint32_t mask_block_low

  MaskBlock.
- QM_RW uint32_t mask_block_high

  MaskBlock.
- QM_RW uint32_t mask_src_trans_low

  MaskSrcTran.
- QM_RW uint32_t mask_src_trans_high

  MaskSrcTran.
- QM_RW uint32_t mask_dst_trans_low

  MaskDstTran.

- QM_RW uint32_t mask_dst_trans_high
  MaskDstTran.
- QM_RW uint32_t mask_err_low
  MaskErr.
- QM_RW uint32_t mask_err_high
  MaskErr.
- QM_RW uint32_t clear_tfr_low
  ClearTfr.
- QM_RW uint32_t clear_tfr_high
  ClearTfr.
- QM_RW uint32_t clear_block_low
  ClearBlock.
- QM_RW uint32_t clear_block_high
  ClearBlock.
- QM_RW uint32_t clear_src_trans_low
  ClearSrcTran.
- QM_RW uint32_t clear_src_trans_high
  ClearSrcTran.
- QM_RW uint32_t clear_dst_trans_low
  ClearDstTran.
- QM_RW uint32_t clear_dst_trans_high
  ClearDstTran.
- QM_RW uint32_t clear_err_low
  ClearErr.
- QM_RW uint32_t clear_err_high
  ClearErr.
- QM_RW uint32_t status_int_low
  StatusInt.
- QM_RW uint32_t status_int_high
  StatusInt.

5.12.1 Detailed Description

DMA interrupt register map.

Definition at line 1387 of file qm_soc_regs.h.

5.13 qm_dma_misc_reg_t Struct Reference

DMA miscellaneous register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t cfg_low
  DmaCfgReg.
- QM_RW uint32_t cfg_high
  DmaCfgReg.
- QM_RW uint32_t chan_en_low
  ChEnReg.

- QM_RW uint32_t chan_en_high

  ChEnReg.
- QM_RW uint32_t id_low

  DmaIdReg.
- QM_RW uint32_t id_high

  DmaIdReg.
- QM_RW uint32_t test_low

  DmaTestReg.
- QM_RW uint32_t test_high

  DmaTestReg.
- QM_RW uint32_t reserved [4]

  Reserved.

5.13.1 Detailed Description

DMA miscellaneous register map.

Definition at line 1437 of file qm_soc_regs.h.

5.14 qm_dma_transfer_t Struct Reference

DMA transfer configuration structure.

`#include <qm_dma.h>`

Data Fields

- uint32_t block_size

  DMA block size, Min = 1, Max = 4095.
- uint32_t * source_address

  DMA source transfer address.
- uint32_t * destination_address

  DMA destination transfer address.

5.14.1 Detailed Description

DMA transfer configuration structure.

Definition at line 105 of file qm_dma.h.

5.14.2 Field Documentation

5.14.2.1 uint32_t qm_dma_transfer_t::block_size

DMA block size, Min = 1, Max = 4095.

Definition at line 106 of file qm_dma.h.

Referenced by qm_dma_transfer_mem_to_mem(), qm_dma_transfer_set_config(), qm_spi_dma_transfer(), qm_uart_dma_read(), and qm_uart_dma_write().

5.14.2.2 uint32_t* qm_dma_transfer_t::destination_address

DMA destination transfer address.

Definition at line 108 of file qm_dma.h.

Referenced by qm_dma_transfer_mem_to_mem(), qm_dma_transfer_set_config(), qm_spi_dma_transfer(), qm_uart_dma_read(), and qm_uart_dma_write().

5.14.2.3 uint32_t* qm_dma_transfer_t::source_address

DMA source transfer address.

Definition at line 107 of file qm_dma.h.

Referenced by qm_dma_transfer_mem_to_mem(), qm_dma_transfer_set_config(), qm_spi_dma_transfer(), qm_uart_dma_read(), and qm_uart_dma_write().

5.15 qm_flash_config_t Struct Reference

Flash configuration structure.

`#include <qm_flash.h>`

Data Fields

- uint8_t wait_states

  Read wait state.
- uint8_t us_count

  Number of clocks in a microsecond.
- qm_flash_disable_t write_disable

  Write disable.

5.15.1 Detailed Description

Flash configuration structure.

Definition at line 80 of file qm_flash.h.

5.15.2 Field Documentation

5.15.2.1 uint8_t qm_flash_config_t::us_count

Number of clocks in a microsecond.

Definition at line 82 of file qm_flash.h.

Referenced by qm_flash_set_config().

5.15.2.2 uint8_t qm_flash_config_t::wait_states

Read wait state.

Definition at line 81 of file qm_flash.h.

Referenced by qm_flash_set_config().

5.15.2.3 qm_flash_disable_t qm_flash_config_t::write_disable

Write disable.

Definition at line 83 of file qm_flash.h.

Referenced by qm_flash_set_config().

5.16 qm_flash_reg_t Struct Reference

Flash register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t tmg_ctrl
  TMG_CTRL.
- QM_RW uint32_t rom_wr_ctrl
  ROM_WR_CTRL.
- QM_RW uint32_t rom_wr_data
  ROM_WR_DATA.
- QM_RW uint32_t flash_wr_ctrl
  FLASH_WR_CTRL.
- QM_RW uint32_t flash_wr_data
  FLASH_WR_DATA.
- QM_RW uint32_t flash_stts
  FLASH_STTS.
- QM_RW uint32_t ctrl
  CTRL.
- QM_RW uint32_t fpr_rd_cfg [4]
  4 FPR_RD_CFG registers
- QM_RW uint32_t mpr_wr_cfg
  Flash Write Protection Control Register.
- QM_RW uint32_t mpr_vsts
  Protection Status Register.
- QM_RW uint32_t mpr_vdata
  MPR Violation Data Value Register.

5.16.1 Detailed Description

Flash register map.

Definition at line 1051 of file qm_soc_regs.h.

5.17 qm_fpr_config_t Struct Reference

Flash Protection Region configuration structure.

`#include <qm_fpr.h>`

Data Fields

- qm_fpr_en_t en_mask
  Enable/lock bitmask.
- qm_fpr_read_allow_t allow_agents

Per-agent read enable bitmask.

- uint8_t up_bound

  1KB-aligned upper Flash phys addr.

- uint8_t low_bound

  1KB-aligned lower Flash phys addr.

5.17.1 Detailed Description

Flash Protection Region configuration structure.

Definition at line 80 of file qm_fpr.h.

5.17.2 Field Documentation

5.17.2.1 qm_fpr_read_allow_t qm_fpr_config_t::allow_agents

Per-agent read enable bitmask.

Definition at line 82 of file qm_fpr.h.

Referenced by qm_fpr_set_config().

5.17.2.2 qm_fpr_en_t qm_fpr_config_t::en_mask

Enable/lock bitmask.

Definition at line 81 of file qm_fpr.h.

Referenced by qm_fpr_set_config().

5.17.2.3 uint8_t qm_fpr_config_t::low_bound

1KB-aligned lower Flash phys addr.

Definition at line 84 of file qm_fpr.h.

Referenced by qm_fpr_set_config().

5.17.2.4 uint8_t qm_fpr_config_t::up_bound

1KB-aligned upper Flash phys addr.

Definition at line 83 of file qm_fpr.h.

Referenced by qm_fpr_set_config().

5.18 qm_gpio_port_config_t Struct Reference

GPIO port configuration type.

`#include <qm_gpio.h>`

Data Fields

- uint32_t direction

  GPIO direction, 0b: input, 1b: output.

- uint32_t int_en

  Interrupt enable.

- uint32_t int_type

  Interrupt type, 0b: level; 1b: edge.

- uint32_t int_polarity

  Interrupt polarity, 0b: low, 1b: high.
- uint32_t int_debounce

  Interrupt debounce on/off.
- uint32_t int_bothedge

  Interrupt on rising and falling edges.
- void(∗ callback)(void ∗data, uint32_t int_status)

  Transfer callback.
- void ∗ callback_data

  Callback user data.

5.18.1 Detailed Description

GPIO port configuration type.

Each bit in the registers control a GPIO pin.

Definition at line 32 of file qm_gpio.h.

5.18.2 Field Documentation

5.18.2.1 void(∗ qm_gpio_port_config_t::callback)(void ∗data, uint32_t int_status)

Transfer callback.

Parameters

in	*data*	Callback user data.
in	*int_status*	GPIO interrupt status.

Definition at line 46 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.2 void∗ qm_gpio_port_config_t::callback_data

Callback user data.

Definition at line 47 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.3 uint32_t qm_gpio_port_config_t::direction

GPIO direction, 0b: input, 1b: output.

Definition at line 33 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.4 uint32_t qm_gpio_port_config_t::int_bothedge

Interrupt on rising and falling edges.

Definition at line 38 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.5 uint32_t qm_gpio_port_config_t::int_debounce

Interrupt debounce on/off.

Definition at line 37 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.6 uint32_t qm_gpio_port_config_t::int_en

Interrupt enable.

Definition at line 34 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.7 uint32_t qm_gpio_port_config_t::int_polarity

Interrupt polarity, 0b: low, 1b: high.

Definition at line 36 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.18.2.8 uint32_t qm_gpio_port_config_t::int_type

Interrupt type, 0b: level; 1b: edge.

Definition at line 35 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.19 qm_gpio_reg_t Struct Reference

GPIO register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t gpio_swporta_dr
  Port A Data.
- QM_RW uint32_t gpio_swporta_ddr
  Port A Data Direction.
- QM_RW uint32_t gpio_inten
  Interrupt Enable.
- QM_RW uint32_t gpio_intmask
  Interrupt Mask.
- QM_RW uint32_t gpio_inttype_level
  Interrupt Type.
- QM_RW uint32_t gpio_int_polarity
  Interrupt Polarity.
- QM_RW uint32_t gpio_intstatus
  Interrupt Status.
- QM_RW uint32_t gpio_raw_intstatus
  Raw Interrupt Status.
- QM_RW uint32_t gpio_debounce
  Debounce Enable.
- QM_RW uint32_t gpio_porta_eoi
  Clear Interrupt.
- QM_RW uint32_t gpio_ext_porta
  Port A External Port.
- QM_RW uint32_t gpio_ls_sync
  Synchronization Level.

- QM_RW uint32_t gpio_id_code

  GPIO ID code.
- QM_RW uint32_t gpio_int_bothedge

  Interrupt both edge type.
- QM_RW uint32_t gpio_ver_id_code

  GPIO Component Version.
- QM_RW uint32_t gpio_config_reg2

  GPIO Configuration Register 2.
- QM_RW uint32_t gpio_config_reg1

  GPIO Configuration Register 1.
- QM_RW uint32_t gpio_swporta_ctl

  Port A Data Source.

5.19.1 Detailed Description

GPIO register map.

Definition at line 929 of file qm_soc_regs.h.

5.19.2 Field Documentation

5.19.2.1 QM_RW uint32_t qm_gpio_reg_t::gpio_config_reg1

GPIO Configuration Register 1.

Definition at line 948 of file qm_soc_regs.h.

5.19.2.2 QM_RW uint32_t qm_gpio_reg_t::gpio_config_reg2

GPIO Configuration Register 2.

Definition at line 947 of file qm_soc_regs.h.

5.19.2.3 QM_RW uint32_t qm_gpio_reg_t::gpio_debounce

Debounce Enable.

Definition at line 939 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.4 QM_RW uint32_t qm_gpio_reg_t::gpio_ext_porta

Port A External Port.

Definition at line 941 of file qm_soc_regs.h.

5.19.2.5 QM_RW uint32_t qm_gpio_reg_t::gpio_id_code

GPIO ID code.

Definition at line 944 of file qm_soc_regs.h.

5.19.2.6 QM_RW uint32_t qm_gpio_reg_t::gpio_int_bothedge

Interrupt both edge type.

Definition at line 945 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.7 QM_RW uint32_t qm_gpio_reg_t::gpio_int_polarity

Interrupt Polarity.

Definition at line 936 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.8 QM_RW uint32_t qm_gpio_reg_t::gpio_inten

Interrupt Enable.

Definition at line 933 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.9 QM_RW uint32_t qm_gpio_reg_t::gpio_intmask

Interrupt Mask.

Definition at line 934 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.10 QM_RW uint32_t qm_gpio_reg_t::gpio_intstatus

Interrupt Status.

Definition at line 937 of file qm_soc_regs.h.

5.19.2.11 QM_RW uint32_t qm_gpio_reg_t::gpio_inttype_level

Interrupt Type.

Definition at line 935 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.12 QM_RW uint32_t qm_gpio_reg_t::gpio_ls_sync

Synchronization Level.

Definition at line 943 of file qm_soc_regs.h.

5.19.2.13 QM_RW uint32_t qm_gpio_reg_t::gpio_porta_eoi

Clear Interrupt.

Definition at line 940 of file qm_soc_regs.h.

5.19.2.14 QM_RW uint32_t qm_gpio_reg_t::gpio_raw_intstatus

Raw Interrupt Status.

Definition at line 938 of file qm_soc_regs.h.

5.19.2.15 QM_RW uint32_t qm_gpio_reg_t::gpio_swporta_ddr

Port A Data Direction.

Definition at line 931 of file qm_soc_regs.h.

Referenced by qm_gpio_set_config().

5.19.2.16 QM_RW uint32_t qm_gpio_reg_t::gpio_swporta_dr

Port A Data.

Definition at line 930 of file qm_soc_regs.h.

5.19.2.17 QM_RW uint32_t qm_gpio_reg_t::gpio_ver_id_code

GPIO Component Version.

Definition at line 946 of file qm_soc_regs.h.

5.20 qm_i2c_config_t Struct Reference

I2C configuration type.

`#include <qm_i2c.h>`

Data Fields

- qm_i2c_speed_t speed
  Standard, Fast Mode.
- qm_i2c_addr_t address_mode
  7 or 10 bit addressing.
- qm_i2c_mode_t mode
  Master or slave mode.
- uint16_t slave_addr
  I2C address when in slave mode.

5.20.1 Detailed Description

I2C configuration type.

Definition at line 88 of file qm_i2c.h.

5.20.2 Field Documentation

5.20.2.1 qm_i2c_addr_t qm_i2c_config_t::address_mode

7 or 10 bit addressing.

Definition at line 90 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.20.2.2 qm_i2c_mode_t qm_i2c_config_t::mode

Master or slave mode.

Definition at line 91 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.20.2.3 uint16_t qm_i2c_config_t::slave_addr

I2C address when in slave mode.

Definition at line 92 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.20.2.4 qm_i2c_speed_t qm_i2c_config_t::speed

Standard, Fast Mode.

Definition at line 89 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.21 qm_i2c_reg_t Struct Reference

I2C register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t ic_con
  Control Register.
- QM_RW uint32_t ic_tar
  Master Target Address.
- QM_RW uint32_t ic_sar
  Slave Address.
- QM_RW uint32_t ic_hs_maddr
  High Speed Master ID.
- QM_RW uint32_t ic_data_cmd
  Data Buffer and Command.
- QM_RW uint32_t ic_ss_scl_hcnt
  Standard Speed Clock SCL High Count.
- QM_RW uint32_t ic_ss_scl_lcnt
  Standard Speed Clock SCL Low Count.
- QM_RW uint32_t ic_fs_scl_hcnt
  Fast Speed Clock SCL High Count.
- QM_RW uint32_t ic_fs_scl_lcnt
  Fast Speed I2C Clock SCL Low Count.
- QM_RW uint32_t ic_hs_scl_hcnt
  High Speed I2C Clock SCL High Count.
- QM_RW uint32_t ic_hs_scl_lcnt
  High Speed I2C Clock SCL Low Count.
- QM_RW uint32_t ic_intr_stat
  Interrupt Status.
- QM_RW uint32_t ic_intr_mask
  Interrupt Mask.
- QM_RW uint32_t ic_raw_intr_stat
  Raw Interrupt Status.
- QM_RW uint32_t ic_rx_tl
  Receive FIFO Threshold Level.
- QM_RW uint32_t ic_tx_tl
  Transmit FIFO Threshold Level.
- QM_RW uint32_t ic_clr_intr
  Clear Combined and Individual Interrupt.
- QM_RW uint32_t ic_clr_rx_under
  Clear RX_UNDER Interrupt.
- QM_RW uint32_t ic_clr_rx_over
  Clear RX_OVER Interrupt.
- QM_RW uint32_t ic_clr_tx_over
  Clear TX_OVER Interrupt.
- QM_RW uint32_t ic_clr_rd_req
  Clear RD_REQ Interrupt.
- QM_RW uint32_t ic_clr_tx_abrt

Clear TX_ABRT Interrupt.

- QM_RW uint32_t ic_clr_rx_done

  Clear RX_DONE Interrupt.
- QM_RW uint32_t ic_clr_activity

  Clear ACTIVITY Interrupt.
- QM_RW uint32_t ic_clr_stop_det

  Clear STOP_DET Interrupt.
- QM_RW uint32_t ic_clr_start_det

  Clear START_DET Interrupt.
- QM_RW uint32_t ic_clr_gen_call

  Clear GEN_CALL Interrupt.
- QM_RW uint32_t ic_enable

  Enable.
- QM_RW uint32_t ic_status

  Status.
- QM_RW uint32_t ic_txflr

  Transmit FIFO Level.
- QM_RW uint32_t ic_rxflr

  Receive FIFO Level.
- QM_RW uint32_t ic_sda_hold

  SDA Hold.
- QM_RW uint32_t ic_tx_abrt_source

  Transmit Abort Source.
- QM_RW uint32_t ic_dma_cr

  SDA Setup.
- QM_RW uint32_t ic_dma_tdlr

  DMA Transmit Data Level Register.
- QM_RW uint32_t ic_dma_rdlr

  I2C Receive Data Level Register.
- QM_RW uint32_t ic_sda_setup

  SDA Setup.
- QM_RW uint32_t ic_ack_general_call

  General Call Ack.
- QM_RW uint32_t ic_enable_status

  Enable Status.
- QM_RW uint32_t ic_fs_spklen

  SS and FS Spike Suppression Limit.
- QM_RW uint32_t ic_hs_spklen

  HS spike suppression limit.
- QM_RW uint32_t ic_clr_restart_det

  clear the RESTART_DET interrupt.
- QM_RW uint32_t ic_comp_param_1

  Configuration Parameters.
- QM_RW uint32_t ic_comp_version

  Component Version.
- QM_RW uint32_t ic_comp_type

  Component Type.

5.21.1 Detailed Description

I2C register map.

Definition at line 793 of file qm_soc_regs.h.

5.21.2 Field Documentation

5.21.2.1 QM_RW uint32_t qm_i2c_reg_t::ic_ack_general_call

General Call Ack.

Definition at line 838 of file qm_soc_regs.h.

5.21.2.2 QM_RW uint32_t qm_i2c_reg_t::ic_clr_activity

Clear ACTIVITY Interrupt.

Definition at line 823 of file qm_soc_regs.h.

5.21.2.3 QM_RW uint32_t qm_i2c_reg_t::ic_clr_gen_call

Clear GEN_CALL Interrupt.

Definition at line 826 of file qm_soc_regs.h.

5.21.2.4 QM_RW uint32_t qm_i2c_reg_t::ic_clr_intr

Clear Combined and Individual Interrupt.

Definition at line 816 of file qm_soc_regs.h.

5.21.2.5 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rd_req

Clear RD_REQ Interrupt.

Definition at line 820 of file qm_soc_regs.h.

5.21.2.6 QM_RW uint32_t qm_i2c_reg_t::ic_clr_restart_det

clear the RESTART_DET interrupt.

Definition at line 843 of file qm_soc_regs.h.

5.21.2.7 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rx_done

Clear RX_DONE Interrupt.

Definition at line 822 of file qm_soc_regs.h.

5.21.2.8 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rx_over

Clear RX_OVER Interrupt.

Definition at line 818 of file qm_soc_regs.h.

5.21.2.9 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rx_under

Clear RX_UNDER Interrupt.

Definition at line 817 of file qm_soc_regs.h.

5.21.2.10 QM_RW uint32_t qm_i2c_reg_t::ic_clr_start_det

Clear START_DET Interrupt.

Definition at line 825 of file qm_soc_regs.h.

5.21.2.11 QM_RW uint32_t qm_i2c_reg_t::ic_clr_stop_det

Clear STOP_DET Interrupt.

Definition at line 824 of file qm_soc_regs.h.

5.21.2.12 QM_RW uint32_t qm_i2c_reg_t::ic_clr_tx_abrt

Clear TX_ABRT Interrupt.

Definition at line 821 of file qm_soc_regs.h.

Referenced by qm_i2c_master_read(), and qm_i2c_master_write().

5.21.2.13 QM_RW uint32_t qm_i2c_reg_t::ic_clr_tx_over

Clear TX_OVER Interrupt.

Definition at line 819 of file qm_soc_regs.h.

5.21.2.14 QM_RW uint32_t qm_i2c_reg_t::ic_comp_param_1

Configuration Parameters.

Definition at line 845 of file qm_soc_regs.h.

5.21.2.15 QM_RW uint32_t qm_i2c_reg_t::ic_comp_type

Component Type.

Definition at line 847 of file qm_soc_regs.h.

5.21.2.16 QM_RW uint32_t qm_i2c_reg_t::ic_comp_version

Component Version.

Definition at line 846 of file qm_soc_regs.h.

5.21.2.17 QM_RW uint32_t qm_i2c_reg_t::ic_con

Control Register.

Definition at line 794 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config(), and qm_i2c_set_speed().

5.21.2.18 QM_RW uint32_t qm_i2c_reg_t::ic_data_cmd

Data Buffer and Command.

Definition at line 798 of file qm_soc_regs.h.

Referenced by qm_i2c_master_read(), and qm_i2c_master_write().

5.21.2.19 QM_RW uint32_t qm_i2c_reg_t::ic_dma_cr

SDA Setup.

Definition at line 834 of file qm_soc_regs.h.

5.21.2.20 QM_RW uint32_t qm_i2c_reg_t::ic_dma_rdlr

I2C Receive Data Level Register.

Definition at line 836 of file qm_soc_regs.h.

5.21.2.21 QM_RW uint32_t qm_i2c_reg_t::ic_dma_tdlr

DMA Transmit Data Level Register.

Definition at line 835 of file qm_soc_regs.h.

5.21.2.22 QM_RW uint32_t qm_i2c_reg_t::ic_enable

Enable.

Definition at line 827 of file qm_soc_regs.h.

5.21.2.23 QM_RW uint32_t qm_i2c_reg_t::ic_enable_status

Enable Status.

Definition at line 839 of file qm_soc_regs.h.

5.21.2.24 QM_RW uint32_t qm_i2c_reg_t::ic_fs_scl_hcnt

Fast Speed Clock SCL High Count.

Definition at line 803 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config(), and qm_i2c_set_speed().

5.21.2.25 QM_RW uint32_t qm_i2c_reg_t::ic_fs_scl_lcnt

Fast Speed I2C Clock SCL Low Count.

Definition at line 805 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config(), and qm_i2c_set_speed().

5.21.2.26 QM_RW uint32_t qm_i2c_reg_t::ic_fs_spklen

SS and FS Spike Suppression Limit.

Definition at line 840 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config(), and qm_i2c_set_speed().

5.21.2.27 QM_RW uint32_t qm_i2c_reg_t::ic_hs_maddr

High Speed Master ID.

Definition at line 797 of file qm_soc_regs.h.

5.21.2.28 QM_RW uint32_t qm_i2c_reg_t::ic_hs_scl_hcnt

High Speed I2C Clock SCL High Count.

Definition at line 807 of file qm_soc_regs.h.

5.21.2.29 QM_RW uint32_t qm_i2c_reg_t::ic_hs_scl_lcnt

High Speed I2C Clock SCL Low Count.

Definition at line 809 of file qm_soc_regs.h.

5.21.2.30 QM_RW uint32_t qm_i2c_reg_t::ic_hs_spklen

HS spike suppression limit.

Definition at line 841 of file qm_soc_regs.h.

5.21.2.31 QM_RW uint32_t qm_i2c_reg_t::ic_intr_mask

Interrupt Mask.

Definition at line 811 of file qm_soc_regs.h.

Referenced by qm_i2c_master_irq_transfer(), and qm_i2c_set_config().

5.21.2.32 QM_RW uint32_t qm_i2c_reg_t::ic_intr_stat

Interrupt Status.

Definition at line 810 of file qm_soc_regs.h.

5.21.2.33 QM_RW uint32_t qm_i2c_reg_t::ic_raw_intr_stat

Raw Interrupt Status.

Definition at line 812 of file qm_soc_regs.h.

Referenced by qm_i2c_master_read().

5.21.2.34 QM_RW uint32_t qm_i2c_reg_t::ic_rx_tl

Receive FIFO Threshold Level.

Definition at line 813 of file qm_soc_regs.h.

Referenced by qm_i2c_master_irq_transfer().

5.21.2.35 QM_RW uint32_t qm_i2c_reg_t::ic_rxflr

Receive FIFO Level.

Definition at line 830 of file qm_soc_regs.h.

5.21.2.36 QM_RW uint32_t qm_i2c_reg_t::ic_sar

Slave Address.

Definition at line 796 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config().

5.21.2.37 QM_RW uint32_t qm_i2c_reg_t::ic_sda_hold

SDA Hold.

Definition at line 831 of file qm_soc_regs.h.

5.21.2.38 QM_RW uint32_t qm_i2c_reg_t::ic_sda_setup

SDA Setup.

Definition at line 837 of file qm_soc_regs.h.

5.21.2.39 QM_RW uint32_t qm_i2c_reg_t::ic_ss_scl_hcnt

Standard Speed Clock SCL High Count.

Definition at line 800 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config(), and qm_i2c_set_speed().

5.21.2.40 QM_RW uint32_t qm_i2c_reg_t::ic_ss_scl_lcnt

Standard Speed Clock SCL Low Count.

Definition at line 802 of file qm_soc_regs.h.

Referenced by qm_i2c_set_config(), and qm_i2c_set_speed().

5.21.2.41 QM_RW uint32_t qm_i2c_reg_t::ic_status

Status.

Definition at line 828 of file qm_soc_regs.h.

Referenced by qm_i2c_get_status(), qm_i2c_master_read(), and qm_i2c_master_write().

5.21.2.42 QM_RW uint32_t qm_i2c_reg_t::ic_tar

Master Target Address.

Definition at line 795 of file qm_soc_regs.h.

Referenced by qm_i2c_master_irq_transfer(), qm_i2c_master_read(), and qm_i2c_master_write().

5.21.2.43 QM_RW uint32_t qm_i2c_reg_t::ic_tx_abrt_source

Transmit Abort Source.

Definition at line 832 of file qm_soc_regs.h.

Referenced by qm_i2c_get_status(), qm_i2c_master_read(), and qm_i2c_master_write().

5.21.2.44 QM_RW uint32_t qm_i2c_reg_t::ic_tx_tl

Transmit FIFO Threshold Level.

Definition at line 814 of file qm_soc_regs.h.

Referenced by qm_i2c_master_irq_transfer().

5.21.2.45 QM_RW uint32_t qm_i2c_reg_t::ic_txflr

Transmit FIFO Level.

Definition at line 829 of file qm_soc_regs.h.

5.22 qm_i2c_transfer_t Struct Reference

I2C transfer type.

`#include <qm_i2c.h>`

Data Fields

- uint8_t * tx
  Write data.
- uint32_t tx_len
  Write data length.
- uint8_t * rx
  Read data.
- uint32_t rx_len
  Read buffer length.
- bool stop
  Generate master STOP.
- void(* callback)(void *data, int rc, qm_i2c_status_t status, uint32_t len)
  Callback.
- void * callback_data
  Callback identifier.

5.22.1 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 105 of file qm_i2c.h.

5.22.2 Field Documentation

5.22.2.1 void(∗ qm_i2c_transfer_t::callback)(void ∗data, int rc, qm_i2c_status_t status, uint32_t len)

Callback.

Definition at line 111 of file qm_i2c.h.

5.22.2.2 void∗ qm_i2c_transfer_t::callback_data

Callback identifier.

Definition at line 113 of file qm_i2c.h.

5.22.2.3 uint8_t∗ qm_i2c_transfer_t::rx

Read data.

Definition at line 108 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.22.2.4 uint32_t qm_i2c_transfer_t::rx_len

Read buffer length.

Definition at line 109 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer(), and qm_i2c_master_irq_transfer().

5.22.2.5 bool qm_i2c_transfer_t::stop

Generate master STOP.

Definition at line 110 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.22.2.6 uint8_t∗ qm_i2c_transfer_t::tx

Write data.

Definition at line 106 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.22.2.7 uint32_t qm_i2c_transfer_t::tx_len

Write data length.

Definition at line 107 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.23 qm_lapic_reg_t Struct Reference

APIC register block type.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW apic_reg_pad_t id
  LAPIC ID.
- QM_RW apic_reg_pad_t version
  LAPIC version.
- QM_RW apic_reg_pad_t tpr
  Task priority.
- QM_RW apic_reg_pad_t apr
  Arbitration priority.
- QM_RW apic_reg_pad_t ppr
  Processor priority.
- QM_RW apic_reg_pad_t eoi
  End of interrupt.
- QM_RW apic_reg_pad_t rrd
  Remote read.
- QM_RW apic_reg_pad_t ldr
  Logical destination.
- QM_RW apic_reg_pad_t dfr
  Destination format.
- QM_RW apic_reg_pad_t svr
  Spurious vector.
- QM_RW apic_reg_pad_t isr [8]
  In-service.
- QM_RW apic_reg_pad_t tmr [8]
  Trigger mode.
- QM_RW apic_reg_pad_t irr [8]
  Interrupt request.
- QM_RW apic_reg_pad_t esr
  Error status.
- QM_RW apic_reg_pad_t lvtcmci
  Corrected Machine Check vector.
- QM_RW apic_reg_pad_t icr [2]
  Interrupt command.
- QM_RW apic_reg_pad_t lvttimer
  Timer vector.
- QM_RW apic_reg_pad_t lvtts
  Thermal sensor vector.
- QM_RW apic_reg_pad_t lvtpmcr
  Perfmon counter vector.
- QM_RW apic_reg_pad_t lvtlint0
  Local interrupt 0 vector.
- QM_RW apic_reg_pad_t lvtlint1

Local interrupt 1 vector.

- QM_RW apic_reg_pad_t lvterr

  Error vector.

- QM_RW apic_reg_pad_t timer_icr

  Timer initial count.

- QM_RW apic_reg_pad_t timer_ccr

  Timer current count.

- QM_RW apic_reg_pad_t timer_dcr

  Timer divide configuration.

5.23.1 Detailed Description

APIC register block type.

Definition at line 215 of file qm_soc_regs.h.

5.24 qm_mailbox_t Struct Reference

Mailbox register structure.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t ch_ctrl

  Channel Control Word.

- QM_RW uint32_t ch_data [4]

  Channel Payload Data Word 0.

- QM_RW uint32_t ch_sts

  Channel status.

5.24.1 Detailed Description

Mailbox register structure.

Definition at line 606 of file qm_soc_regs.h.

5.25 qm_mbox_msg_t Struct Reference

Definition of the mailbox message.

`#include <qm_mailbox.h>`

Data Fields

- uint32_t ctrl

  Mailbox control element.

- uint32_t data [QM_MBOX_PAYLOAD_NUM]

  Mailbox data buffer.

5.25.1 Detailed Description

Definition of the mailbox message.

Definition at line 60 of file qm_mailbox.h.

5.25.2 Field Documentation

5.25.2.1 uint32_t qm_mbox_msg_t::ctrl

Mailbox control element.

Definition at line 61 of file qm_mailbox.h.

Referenced by qm_mbox_ch_read(), and qm_mbox_ch_write().

5.25.2.2 uint32_t qm_mbox_msg_t::data[QM_MBOX_PAYLOAD_NUM]

Mailbox data buffer.

Definition at line 62 of file qm_mailbox.h.

Referenced by qm_mbox_ch_read(), and qm_mbox_ch_write().

5.26 qm_mpr_config_t Struct Reference

SRAM Memory Protection Region configuration type.

`#include <qm_mpr.h>`

Data Fields

- uint8_t en_lock_mask

  Enable/lock bitmask.
- uint8_t agent_read_en_mask

  Per-agent read enable bitmask.
- uint8_t agent_write_en_mask

  Per-agent write enable bitmask.
- uint8_t up_bound

  1KB-aligned upper addr
- uint8_t low_bound

  1KB-aligned lower addr

5.26.1 Detailed Description

SRAM Memory Protection Region configuration type.

Definition at line 43 of file qm_mpr.h.

5.27 qm_mpr_reg_t Struct Reference

Memory Protection Region register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t mpr_cfg [4]
  MPR CFG.
- QM_RW uint32_t mpr_vdata
  MPR_VDATA.
- QM_RW uint32_t mpr_vsts
  MPR_VSTS.

5.27.1 Detailed Description

Memory Protection Region register map.

Definition at line 1117 of file qm_soc_regs.h.

5.28 qm_mvic_reg_t Struct Reference

MVIC register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW mvic_reg_pad_t tpr
  Task priority.
- QM_RW mvic_reg_pad_t ppr
  Processor priority.
- QM_RW mvic_reg_pad_t eoi
  End of interrupt.
- QM_RW mvic_reg_pad_t sivr
  Spurious vector.
- QM_RW mvic_reg_pad_t isr
  In-service.
- QM_RW mvic_reg_pad_t irr
  Interrupt request.
- QM_RW mvic_reg_pad_t lvttimer
  Timer vector.
- QM_RW mvic_reg_pad_t icr
  Timer initial count.
- QM_RW mvic_reg_pad_t ccr
  Timer current count.

5.28.1 Detailed Description

MVIC register map.

Definition at line 1230 of file qm_soc_regs.h.

5.28.2 Field Documentation

5.28.2.1 QM_RW mvic_reg_pad_t qm_mvic_reg_t::ccr

Timer current count.

Definition at line 1245 of file qm_soc_regs.h.

5.28.2.2 QM_RW mvic_reg_pad_t qm_mvic_reg_t::eoi

End of interrupt.

Definition at line 1234 of file qm_soc_regs.h.

5.28.2.3 QM_RW mvic_reg_pad_t qm_mvic_reg_t::icr

Timer initial count.

Definition at line 1244 of file qm_soc_regs.h.

5.28.2.4 QM_RW mvic_reg_pad_t qm_mvic_reg_t::irr

Interrupt request.

Definition at line 1240 of file qm_soc_regs.h.

5.28.2.5 QM_RW mvic_reg_pad_t qm_mvic_reg_t::isr

In-service.

Definition at line 1238 of file qm_soc_regs.h.

5.28.2.6 QM_RW mvic_reg_pad_t qm_mvic_reg_t::lvttimer

Timer vector.

Definition at line 1242 of file qm_soc_regs.h.

5.28.2.7 QM_RW mvic_reg_pad_t qm_mvic_reg_t::ppr

Processor priority.

Definition at line 1233 of file qm_soc_regs.h.

5.28.2.8 QM_RW mvic_reg_pad_t qm_mvic_reg_t::sivr

Spurious vector.

Definition at line 1236 of file qm_soc_regs.h.

5.28.2.9 QM_RW mvic_reg_pad_t qm_mvic_reg_t::tpr

Task priority.

Definition at line 1231 of file qm_soc_regs.h.

5.29 qm_pic_timer_config_t Struct Reference

PIC timer configuration type.

`#include <qm_pic_timer.h>`

Data Fields

- qm_pic_timer_mode_t mode
  Operation mode.
- bool int_en
  Interrupt enable.
- void(∗ callback)(void ∗data)
  User callback.
- void ∗ callback_data

Callback user data.

5.29.1 Detailed Description

PIC timer configuration type.

Definition at line 29 of file qm_pic_timer.h.

5.29.2 Field Documentation

5.29.2.1 void(∗ qm_pic_timer_config_t::callback)(void ∗data)

User callback.

Parameters

in	*data*	User defined data.

Definition at line 38 of file qm_pic_timer.h.

Referenced by qm_pic_timer_set_config().

5.29.2.2 void∗ qm_pic_timer_config_t::callback_data

Callback user data.

Definition at line 39 of file qm_pic_timer.h.

Referenced by qm_pic_timer_set_config().

5.29.2.3 bool qm_pic_timer_config_t::int_en

Interrupt enable.

Definition at line 31 of file qm_pic_timer.h.

Referenced by qm_pic_timer_set_config().

5.29.2.4 qm_pic_timer_mode_t qm_pic_timer_config_t::mode

Operation mode.

Definition at line 30 of file qm_pic_timer.h.

Referenced by qm_pic_timer_set_config().

5.30 qm_pic_timer_reg_t Struct Reference

PIC timer register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW pic_timer_reg_pad_t lvttimer

  Local Vector Table Timer.
- QM_RW pic_timer_reg_pad_t timer_icr

  Initial Count Register.
- QM_RW pic_timer_reg_pad_t timer_ccr

  Current Count Register.

5.30.1 Detailed Description

PIC timer register map.

Definition at line 1158 of file qm_soc_regs.h.

5.31 qm_pwm_channel_t Struct Reference

PWM / Timer channel register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t loadcount

  Load Coun.t.
- QM_RW uint32_t currentvalue

  Current Value.
- QM_RW uint32_t controlreg

  Control.
- QM_RW uint32_t eoi

  End Of Interrupt.
- QM_RW uint32_t intstatus

  Interrupt Status.

5.31.1 Detailed Description

PWM / Timer channel register map.

Definition at line 518 of file qm_soc_regs.h.

5.31.2 Field Documentation

5.31.2.1 QM_RW uint32_t qm_pwm_channel_t::controlreg

Control.

Definition at line 521 of file qm_soc_regs.h.

5.31.2.2 QM_RW uint32_t qm_pwm_channel_t::currentvalue

Current Value.

Definition at line 520 of file qm_soc_regs.h.

5.31.2.3 QM_RW uint32_t qm_pwm_channel_t::eoi

End Of Interrupt.

Definition at line 522 of file qm_soc_regs.h.

5.31.2.4 QM_RW uint32_t qm_pwm_channel_t::intstatus

Interrupt Status.

Definition at line 523 of file qm_soc_regs.h.

5.31.2.5 QM_RW uint32_t qm_pwm_channel_t::loadcount

Load Coun.t.

Load Count.

Definition at line 519 of file qm_soc_regs.h.

5.32 qm_pwm_config_t Struct Reference

PWM / Timer configuration type.

`#include <qm_pwm.h>`

Data Fields

- uint32_t lo_count
  Number of cycles the PWM output is driven low.
- uint32_t hi_count
  Number of cycles the PWM output is driven high.
- bool mask_interrupt
  Mask interrupt.
- qm_pwm_mode_t mode
  Pwm mode.
- void(∗ callback)(void ∗data, uint32_t int_status)
  User callback.
- void ∗ callback_data
  Callback user data.

5.32.1 Detailed Description

PWM / Timer configuration type.

Definition at line 34 of file qm_pwm.h.

5.32.2 Field Documentation

5.32.2.1 void(∗ qm_pwm_config_t::callback)(void ∗data, uint32_t int_status)

User callback.

Parameters

in	*data*	The callback user data.
in	*int_status*	The timer status.

Definition at line 49 of file qm_pwm.h.

Referenced by qm_pwm_set_config().

5.32.2.2 void∗ qm_pwm_config_t::callback_data

Callback user data.

Definition at line 50 of file qm_pwm.h.

Referenced by qm_pwm_set_config().

5.32.2.3 uint32_t qm_pwm_config_t::hi_count

Number of cycles the PWM output is driven high.

Not applicable in timer mode. Must be > 0.

Definition at line 39 of file qm_pwm.h.

Referenced by qm_pwm_set_config().

5.32.2.4 uint32_t qm_pwm_config_t::lo_count

Number of cycles the PWM output is driven low.

In timer mode, this is the timer load count. Must be

0.

Definition at line 36 of file qm_pwm.h.

Referenced by qm_pwm_set_config().

5.32.2.5 bool qm_pwm_config_t::mask_interrupt

Mask interrupt.

Definition at line 41 of file qm_pwm.h.

Referenced by qm_pwm_set_config().

5.32.2.6 qm_pwm_mode_t qm_pwm_config_t::mode

Pwm mode.

Definition at line 42 of file qm_pwm.h.

Referenced by qm_pwm_set_config().

5.33 qm_pwm_reg_t Struct Reference

PWM / Timer register map.

`#include <qm_soc_regs.h>`

Data Fields

- qm_pwm_channel_t timer [QM_PWM_ID_NUM]

  2 Timers.
- QM_RW uint32_t timersintstatus

  Timers Interrupt Status.
- QM_RW uint32_t timerseoi

  Timers End Of Interrupt.
- QM_RW uint32_t timersrawintstatus

  Timers Raw Interrupt Status.
- QM_RW uint32_t timerscompversion

  Timers Component Version.
- QM_RW uint32_t timer_loadcount2 [4]

  Timer Load Count 2.

5.33.1 Detailed Description

PWM / Timer register map.

Definition at line 527 of file qm_soc_regs.h.

5.33.2 Field Documentation

5.33.2.1 qm_pwm_channel_t qm_pwm_reg_t::timer

2 Timers.

4 Timers

Definition at line 528 of file qm_soc_regs.h.

5.34 qm_rtc_config_t Struct Reference

RTC configuration type.

`#include <qm_rtc.h>`

Data Fields

- uint32_t init_val
  Initial value in RTC clocks.
- bool alarm_en
  Alarm enable.
- uint32_t alarm_val
  Alarm value in RTC clocks.
- void(∗ callback)(void ∗data)
  User callback.
- void ∗ callback_data
  Callback user data.

5.34.1 Detailed Description

RTC configuration type.

Definition at line 33 of file qm_rtc.h.

5.34.2 Field Documentation

5.34.2.1 bool qm_rtc_config_t::alarm_en

Alarm enable.

Definition at line 35 of file qm_rtc.h.

Referenced by qm_rtc_set_config().

5.34.2.2 uint32_t qm_rtc_config_t::alarm_val

Alarm value in RTC clocks.

Definition at line 36 of file qm_rtc.h.

Referenced by qm_rtc_set_config().

5.34.2.3 void(∗ qm_rtc_config_t::callback)(void ∗data)

User callback.

Parameters

in	*data*	User defined data.

Definition at line 43 of file qm_rtc.h.

Referenced by qm_rtc_set_config().

5.34.2.4 void* qm_rtc_config_t::callback_data

Callback user data.

Definition at line 44 of file qm_rtc.h.

Referenced by qm_rtc_set_config().

5.34.2.5 uint32_t qm_rtc_config_t::init_val

Initial value in RTC clocks.

Definition at line 34 of file qm_rtc.h.

Referenced by qm_rtc_set_config().

5.35 qm_rtc_reg_t Struct Reference

RTC register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t rtc_ccvr

  Current Counter Value Register.
- QM_RW uint32_t rtc_cmr

  Current Match Register.
- QM_RW uint32_t rtc_clr

  Counter Load Register.
- QM_RW uint32_t rtc_ccr

  Counter Control Register.
- QM_RW uint32_t rtc_stat

  Interrupt Status Register.
- QM_RW uint32_t rtc_rstat

  Interrupt Raw Status Register.
- QM_RW uint32_t rtc_eoi

  End of Interrupt Register.
- QM_RW uint32_t rtc_comp_version

  End of Interrupt Register.

5.35.1 Detailed Description

RTC register map.

Definition at line 759 of file qm_soc_regs.h.

5.35.2 Field Documentation

5.35.2.1 QM_RW uint32_t qm_rtc_reg_t::rtc_ccr

Counter Control Register.

Definition at line 763 of file qm_soc_regs.h.

5.35.2.2 QM_RW uint32_t qm_rtc_reg_t::rtc_ccvr

Current Counter Value Register.

Definition at line 760 of file qm_soc_regs.h.

5.35.2.3 QM_RW uint32_t qm_rtc_reg_t::rtc_clr

Counter Load Register.

Definition at line 762 of file qm_soc_regs.h.

5.35.2.4 QM_RW uint32_t qm_rtc_reg_t::rtc_cmr

Current Match Register.

Definition at line 761 of file qm_soc_regs.h.

5.35.2.5 QM_RW uint32_t qm_rtc_reg_t::rtc_comp_version

End of Interrupt Register.

Definition at line 767 of file qm_soc_regs.h.

5.35.2.6 QM_RW uint32_t qm_rtc_reg_t::rtc_eoi

End of Interrupt Register.

Definition at line 766 of file qm_soc_regs.h.

5.35.2.7 QM_RW uint32_t qm_rtc_reg_t::rtc_rstat

Interrupt Raw Status Register.

Definition at line 765 of file qm_soc_regs.h.

5.35.2.8 QM_RW uint32_t qm_rtc_reg_t::rtc_stat

Interrupt Status Register.

Definition at line 764 of file qm_soc_regs.h.

5.36 qm_scss_aon_reg_t Struct Reference

Always-on Controller register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t aonc_cnt
  Always-on counter register.
- QM_RW uint32_t aonc_cfg
  Always-on counter enable.
- QM_RW uint32_t aonpt_cnt

Always-on periodic timer.

- QM_RW uint32_t aonpt_stat

  Always-on periodic timer status register.

- QM_RW uint32_t aonpt_ctrl

  Always-on periodic timer control.

- QM_RW uint32_t aonpt_cfg

  Always-on periodic timer configuration register.

5.36.1 Detailed Description

Always-on Controller register map.

Definition at line 314 of file qm_soc_regs.h.

5.36.2 Field Documentation

5.36.2.1 QM_RW uint32_t qm_scss_aon_reg_t::aonc_cfg

Always-on counter enable.

Definition at line 316 of file qm_soc_regs.h.

5.36.2.2 QM_RW uint32_t qm_scss_aon_reg_t::aonc_cnt

Always-on counter register.

Definition at line 315 of file qm_soc_regs.h.

5.36.2.3 QM_RW uint32_t qm_scss_aon_reg_t::aonpt_cfg

Always-on periodic timer configuration register.

Definition at line 322 of file qm_soc_regs.h.

5.36.2.4 QM_RW uint32_t qm_scss_aon_reg_t::aonpt_cnt

Always-on periodic timer.

Definition at line 317 of file qm_soc_regs.h.

5.36.2.5 QM_RW uint32_t qm_scss_aon_reg_t::aonpt_ctrl

Always-on periodic timer control.

Definition at line 320 of file qm_soc_regs.h.

5.36.2.6 QM_RW uint32_t qm_scss_aon_reg_t::aonpt_stat

Always-on periodic timer status register.

Definition at line 319 of file qm_soc_regs.h.

5.37 qm_scss_ccu_reg_t Struct Reference

System Core register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t osc0_cfg0

Hybrid Oscillator Configuration 0.

- QM_RW uint32_t osc0_stat1

  Hybrid Oscillator status 1.
- QM_RW uint32_t osc0_cfg1

  Hybrid Oscillator configuration 1.
- QM_RW uint32_t osc1_stat0

  RTC Oscillator status 0.
- QM_RW uint32_t osc1_cfg0

  RTC Oscillator Configuration 0.
- QM_RW uint32_t ccu_periph_clk_gate_ctl

  Peripheral Clock Gate Control.
- QM_RW uint32_t ccu_periph_clk_div_ctl0

  Peripheral Clock Divider Control 0.
- QM_RW uint32_t ccu_gpio_db_clk_ctl

  Peripheral Clock Divider Control 1.
- QM_RW uint32_t ccu_ext_clock_ctl

  External Clock Control Register.
- QM_RW uint32_t ccu_lp_clk_ctl

  System Low Power Clock Control.
- QM_RW uint32_t wake_mask

  Wake Mask register.
- QM_RW uint32_t ccu_mlayer_ahb_ctl

  AHB Control Register.
- QM_RW uint32_t ccu_sys_clk_ctl

  System Clock Control Register.
- QM_RW uint32_t osc_lock_0

  Clocks Lock Register.
- QM_RW uint32_t soc_ctrl

  SoC Control Register.
- QM_RW uint32_t soc_ctrl_lock

  SoC Control Register Lock.
- QM_RW uint32_t usb_pll_cfg0

  USB Phase lock look configuration.
- QM_RW uint32_t ccu_ss_periph_clk_gate_ctl

  Sensor Subsystem peripheral clock gate control.

5.37.1 Detailed Description

System Core register map.

Definition at line 27 of file qm_soc_regs.h.

5.37.2 Field Documentation

5.37.2.1 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_ext_clock_ctl

External Clock Control Register.

Definition at line 41 of file qm_soc_regs.h.

5.37.2.2 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_gpio_db_clk_ctl

Peripheral Clock Divider Control 1.

Definition at line 39 of file qm_soc_regs.h.

5.37.2.3 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_lp_clk_ctl

System Low Power Clock Control.

Definition at line 43 of file qm_soc_regs.h.

5.37.2.4 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_mlayer_ahb_ctl

AHB Control Register.

Definition at line 45 of file qm_soc_regs.h.

5.37.2.5 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_periph_clk_div_ctl0

Peripheral Clock Divider Control 0.

Peripheral Clock Divider Control.

0

Definition at line 37 of file qm_soc_regs.h.

5.37.2.6 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_periph_clk_gate_ctl

Peripheral Clock Gate Control.

Definition at line 35 of file qm_soc_regs.h.

5.37.2.7 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_ss_periph_clk_gate_ctl

Sensor Subsystem peripheral clock gate control.

Definition at line 48 of file qm_soc_regs.h.

5.37.2.8 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_sys_clk_ctl

System Clock Control Register.

Definition at line 46 of file qm_soc_regs.h.

5.37.2.9 QM_RW uint32_t qm_scss_ccu_reg_t::osc0_cfg0

Hybrid Oscillator Configuration 0.

Definition at line 28 of file qm_soc_regs.h.

5.37.2.10 QM_RW uint32_t qm_scss_ccu_reg_t::osc0_cfg1

Hybrid Oscillator configuration 1.

Definition at line 30 of file qm_soc_regs.h.

5.37.2.11 QM_RW uint32_t qm_scss_ccu_reg_t::osc0_stat1

Hybrid Oscillator status 1.

Definition at line 29 of file qm_soc_regs.h.

5.37.2.12 QM_RW uint32_t qm_scss_ccu_reg_t::osc1_cfg0

RTC Oscillator Configuration 0.

Definition at line 32 of file qm_soc_regs.h.

5.37.2.13 QM_RW uint32_t qm_scss_ccu_reg_t::osc1_stat0

RTC Oscillator status 0.

Definition at line 31 of file qm_soc_regs.h.

5.37.2.14 QM_RW uint32_t qm_scss_ccu_reg_t::osc_lock_0

Clocks Lock Register.

Definition at line 47 of file qm_soc_regs.h.

5.37.2.15 QM_RW uint32_t qm_scss_ccu_reg_t::soc_ctrl

SoC Control Register.

Definition at line 48 of file qm_soc_regs.h.

5.37.2.16 QM_RW uint32_t qm_scss_ccu_reg_t::soc_ctrl_lock

SoC Control Register Lock.

Definition at line 49 of file qm_soc_regs.h.

5.37.2.17 QM_RW uint32_t qm_scss_ccu_reg_t::usb_pll_cfg0

USB Phase lock look configuration.

Definition at line 38 of file qm_soc_regs.h.

5.37.2.18 QM_RW uint32_t qm_scss_ccu_reg_t::wake_mask

Wake Mask register.

Definition at line 44 of file qm_soc_regs.h.

5.38 qm_scss_cmp_reg_t Struct Reference

Comparator register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t cmp_en

  Comparator enable.
- QM_RW uint32_t cmp_ref_sel

  Comparator reference select.
- QM_RW uint32_t cmp_ref_pol

  Comparator reference polarity select register.
- QM_RW uint32_t cmp_pwr

  Comparator power enable register.
- QM_RW uint32_t cmp_stat_clr

  Comparator clear register.

5.38.1 Detailed Description

Comparator register map.

Definition at line 174 of file qm_soc_regs.h.

5.38.2 Field Documentation

5.38.2.1 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_en

Comparator enable.

Definition at line 175 of file qm_soc_regs.h.

5.38.2.2 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_pwr

Comparator power enable register.

Definition at line 179 of file qm_soc_regs.h.

5.38.2.3 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_ref_pol

Comparator reference polarity select register.

Definition at line 178 of file qm_soc_regs.h.

5.38.2.4 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_ref_sel

Comparator reference select.

Definition at line 176 of file qm_soc_regs.h.

5.38.2.5 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_stat_clr

Comparator clear register.

Definition at line 181 of file qm_soc_regs.h.

5.39 qm_scss_gp_reg_t Struct Reference

General Purpose register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t gps0
  General Purpose Sticky Register 0.
- QM_RW uint32_t gps1
  General Purpose Sticky Register 1.
- QM_RW uint32_t gps2
  General Purpose Sticky Register 2.
- QM_RW uint32_t gps3
  General Purpose Sticky Register 3.
- QM_RW uint32_t gp0
  General Purpose Scratchpad Register 0.
- QM_RW uint32_t gp1
  General Purpose Scratchpad Register 1.
- QM_RW uint32_t gp2
  General Purpose Scratchpad Register 2.
- QM_RW uint32_t gp3
  General Purpose Scratchpad Register 3.
- QM_RW uint32_t wo_sp
  Write-One-to-Set Scratchpad Register.

- QM_RW uint32_t wo_st

  Write-One-to-Set Sticky Scratchpad Register.
- QM_RW uint32_t id

  Identification Register.
- QM_RW uint32_t rev

  Revision Register.

5.39.1 Detailed Description

General Purpose register map.

Definition at line 138 of file qm_soc_regs.h.

5.39.2 Field Documentation

5.39.2.1 QM_RW uint32_t qm_scss_gp_reg_t::gp0

General Purpose Scratchpad Register 0.

Definition at line 144 of file qm_soc_regs.h.

5.39.2.2 QM_RW uint32_t qm_scss_gp_reg_t::gp1

General Purpose Scratchpad Register 1.

Definition at line 145 of file qm_soc_regs.h.

5.39.2.3 QM_RW uint32_t qm_scss_gp_reg_t::gp2

General Purpose Scratchpad Register 2.

Definition at line 146 of file qm_soc_regs.h.

5.39.2.4 QM_RW uint32_t qm_scss_gp_reg_t::gp3

General Purpose Scratchpad Register 3.

Definition at line 147 of file qm_soc_regs.h.

5.39.2.5 QM_RW uint32_t qm_scss_gp_reg_t::gps0

General Purpose Sticky Register 0.

Definition at line 139 of file qm_soc_regs.h.

5.39.2.6 QM_RW uint32_t qm_scss_gp_reg_t::gps1

General Purpose Sticky Register 1.

Definition at line 140 of file qm_soc_regs.h.

5.39.2.7 QM_RW uint32_t qm_scss_gp_reg_t::gps2

General Purpose Sticky Register 2.

Definition at line 141 of file qm_soc_regs.h.

5.39.2.8 QM_RW uint32_t qm_scss_gp_reg_t::gps3

General Purpose Sticky Register 3.

Definition at line 142 of file qm_soc_regs.h.

5.39.2.9 QM_RW uint32_t qm_scss_gp_reg_t::wo_sp

Write-One-to-Set Scratchpad Register.

Definition at line 149 of file qm_soc_regs.h.

5.39.2.10 QM_RW uint32_t qm_scss_gp_reg_t::wo_st

Write-One-to-Set Sticky Scratchpad Register.

Definition at line 151 of file qm_soc_regs.h.

5.40 qm_scss_info_reg_t Struct Reference

Information register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t id
  Identification Register.
- QM_RW uint32_t rev
  Revision Register.
- QM_RW uint32_t fs
  Flash Size Register.
- QM_RW uint32_t rs
  RAM Size Register.
- QM_RW uint32_t cotps
  Code OTP Size Register.
- QM_RW uint32_t dotps
  Data OTP Size Register.

5.40.1 Detailed Description

Information register map.

Definition at line 398 of file qm_soc_regs.h.

5.40.2 Field Documentation

5.40.2.1 QM_RW uint32_t qm_scss_info_reg_t::cotps

Code OTP Size Register.

Definition at line 403 of file qm_soc_regs.h.

5.40.2.2 QM_RW uint32_t qm_scss_info_reg_t::dotps

Data OTP Size Register.

Definition at line 404 of file qm_soc_regs.h.

5.40.2.3 QM_RW uint32_t qm_scss_info_reg_t::fs

Flash Size Register.

Definition at line 401 of file qm_soc_regs.h.

5.40.2.4 QM_RW uint32_t qm_scss_info_reg_t::id

Identification Register.

Definition at line 399 of file qm_soc_regs.h.

5.40.2.5 QM_RW uint32_t qm_scss_info_reg_t::rev

Revision Register.

Definition at line 400 of file qm_soc_regs.h.

5.40.2.6 QM_RW uint32_t qm_scss_info_reg_t::rs

RAM Size Register.

Definition at line 402 of file qm_soc_regs.h.

5.41 qm_scss_int_reg_t Struct Reference

Interrupt register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t int_i2c_mst_0_mask
  Interrupt Routing Mask 0.
- QM_RW uint32_t int_spi_mst_0_mask
  Interrupt Routing Mask 2.
- QM_RW uint32_t int_spi_slv_0_mask
  Interrupt Routing Mask 4.
- QM_RW uint32_t int_uart_0_mask
  Interrupt Routing Mask 5.
- QM_RW uint32_t int_uart_1_mask
  Interrupt Routing Mask 6.
- QM_RW uint32_t int_gpio_mask
  Interrupt Routing Mask 8.
- QM_RW uint32_t int_timer_mask
  Interrupt Routing Mask 9.
- QM_RW uint32_t int_rtc_mask
  Interrupt Routing Mask 11.
- QM_RW uint32_t int_watchdog_mask
  Interrupt Routing Mask 12.
- QM_RW uint32_t int_dma_channel_0_mask
  Interrupt Routing Mask 13.
- QM_RW uint32_t int_dma_channel_1_mask
  Interrupt Routing Mask 14.
- QM_RW uint32_t int_comparators_host_halt_mask
  Interrupt Routing Mask 23.
- QM_RW uint32_t int_comparators_host_mask
  Interrupt Routing Mask 25.
- QM_RW uint32_t int_host_bus_err_mask
  Interrupt Routing Mask 26.
- QM_RW uint32_t int_dma_error_mask

Interrupt Routing Mask 27.

- QM_RW uint32_t int_sram_controller_mask

  Interrupt Routing Mask 28.
- QM_RW uint32_t int_flash_controller_0_mask

  Interrupt Routing Mask 29.
- QM_RW uint32_t int_aon_timer_mask

  Interrupt Routing Mask 31.
- QM_RW uint32_t int_adc_pwr_mask

  Interrupt Routing Mask 32.
- QM_RW uint32_t int_adc_calib_mask

  Interrupt Routing Mask 33.
- QM_RW uint32_t lock_int_mask_reg

  Interrupt Mask Lock Register.
- QM_RW uint32_t int_flash_controller_1_mask

  Interrupt Routing Mask 30.

5.41.1 Detailed Description

Interrupt register map.

Definition at line 203 of file qm_soc_regs.h.

5.41.2 Field Documentation

5.41.2.1 QM_RW uint32_t qm_scss_int_reg_t::int_adc_calib_mask

Interrupt Routing Mask 33.

Definition at line 236 of file qm_soc_regs.h.

5.41.2.2 QM_RW uint32_t qm_scss_int_reg_t::int_adc_pwr_mask

Interrupt Routing Mask 32.

Definition at line 235 of file qm_soc_regs.h.

5.41.2.3 QM_RW uint32_t qm_scss_int_reg_t::int_aon_timer_mask

Interrupt Routing Mask 31.

Definition at line 234 of file qm_soc_regs.h.

5.41.2.4 QM_RW uint32_t qm_scss_int_reg_t::int_comparators_host_halt_mask

Interrupt Routing Mask 23.

Definition at line 223 of file qm_soc_regs.h.

5.41.2.5 QM_RW uint32_t qm_scss_int_reg_t::int_comparators_host_mask

Interrupt Routing Mask 25.

Definition at line 226 of file qm_soc_regs.h.

5.41.2.6 QM_RW uint32_t qm_scss_int_reg_t::int_dma_channel_0_mask

Interrupt Routing Mask 13.

Definition at line 218 of file qm_soc_regs.h.

5.41.2.7 QM_RW uint32_t qm_scss_int_reg_t::int_dma_channel_1_mask

Interrupt Routing Mask 14.

Definition at line 220 of file qm_soc_regs.h.

5.41.2.8 QM_RW uint32_t qm_scss_int_reg_t::int_dma_error_mask

Interrupt Routing Mask 27.

Definition at line 228 of file qm_soc_regs.h.

5.41.2.9 QM_RW uint32_t qm_scss_int_reg_t::int_flash_controller_0_mask

Interrupt Routing Mask 29.

Definition at line 232 of file qm_soc_regs.h.

5.41.2.10 QM_RW uint32_t qm_scss_int_reg_t::int_flash_controller_1_mask

Interrupt Routing Mask 30.

Definition at line 360 of file qm_soc_regs.h.

5.41.2.11 QM_RW uint32_t qm_scss_int_reg_t::int_gpio_mask

Interrupt Routing Mask 8.

Definition at line 212 of file qm_soc_regs.h.

5.41.2.12 QM_RW uint32_t qm_scss_int_reg_t::int_host_bus_err_mask

Interrupt Routing Mask 26.

Definition at line 227 of file qm_soc_regs.h.

5.41.2.13 QM_RW uint32_t qm_scss_int_reg_t::int_i2c_mst_0_mask

Interrupt Routing Mask 0.

Definition at line 204 of file qm_soc_regs.h.

5.41.2.14 QM_RW uint32_t qm_scss_int_reg_t::int_rtc_mask

Interrupt Routing Mask 11.

Definition at line 215 of file qm_soc_regs.h.

5.41.2.15 QM_RW uint32_t qm_scss_int_reg_t::int_spi_mst_0_mask

Interrupt Routing Mask 2.

Definition at line 206 of file qm_soc_regs.h.

5.41.2.16 QM_RW uint32_t qm_scss_int_reg_t::int_spi_slv_0_mask

Interrupt Routing Mask 4.

Definition at line 208 of file qm_soc_regs.h.

5.41.2.17 QM_RW uint32_t qm_scss_int_reg_t::int_sram_controller_mask

Interrupt Routing Mask 28.

Definition at line 230 of file qm_soc_regs.h.

5.41.2.18 QM_RW uint32_t qm_scss_int_reg_t::int_timer_mask

Interrupt Routing Mask 9.

Definition at line 213 of file qm_soc_regs.h.

5.41.2.19 QM_RW uint32_t qm_scss_int_reg_t::int_uart_0_mask

Interrupt Routing Mask 5.

Definition at line 209 of file qm_soc_regs.h.

5.41.2.20 QM_RW uint32_t qm_scss_int_reg_t::int_uart_1_mask

Interrupt Routing Mask 6.

Definition at line 210 of file qm_soc_regs.h.

5.41.2.21 QM_RW uint32_t qm_scss_int_reg_t::int_watchdog_mask

Interrupt Routing Mask 12.

Definition at line 216 of file qm_soc_regs.h.

5.41.2.22 QM_RW uint32_t qm_scss_int_reg_t::lock_int_mask_reg

Interrupt Mask Lock Register.

Definition at line 238 of file qm_soc_regs.h.

5.42 qm_scss_mailbox_reg_t Struct Reference

Mailbox register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- qm_mailbox_t mbox [8]
  8 Mailboxes
- QM_RW uint32_t mbox_chall_sts
  All channel status.

5.42.1 Detailed Description

Mailbox register map.

Definition at line 613 of file qm_soc_regs.h.

5.43 qm_scss_mem_reg_t Struct Reference

Memory Control register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t mem_ctrl
  Memory control.

5.43.1 Detailed Description

Memory Control register map.

Definition at line 160 of file qm_soc_regs.h.

5.44 qm_scss_peripheral_reg_t Struct Reference

Peripheral Registers register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t periph_cfg0
  Peripheral Configuration.
- QM_RW uint32_t cfg_lock
  Configuration Lock.
- QM_RW uint32_t usb_phy_cfg0
  USB Configuration.

5.44.1 Detailed Description

Peripheral Registers register map.

Definition at line 342 of file qm_soc_regs.h.

5.44.2 Field Documentation

5.44.2.1 QM_RW uint32_t qm_scss_peripheral_reg_t::cfg_lock

Configuration Lock.

Definition at line 345 of file qm_soc_regs.h.

5.44.2.2 QM_RW uint32_t qm_scss_peripheral_reg_t::periph_cfg0

Peripheral Configuration.

Definition at line 343 of file qm_soc_regs.h.

5.45 qm_scss_pmu_reg_t Struct Reference

Power Management register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t aon_vr
  AON Voltage Regulator.
- QM_RW uint32_t pm_wait
  Power Management Wait.
- QM_RW uint32_t p_sts
  Processor Status.

- QM_RW uint32_t rstc

  Reset Control.
- QM_RW uint32_t rsts

  Reset Status.
- QM_RW uint32_t pm_lock

  Power Management Lock.
- QM_RW uint32_t p_lvl2

  Processor level 2.
- QM_RW uint32_t pm1c

  Power management 1 control.
- QM_RW uint32_t plat3p3_vr

  Platform 3p3 voltage regulator.
- QM_RW uint32_t plat1p8_vr

  Platform 1p8 voltage regulator.
- QM_RW uint32_t host_vr

  Host Voltage Regulator.
- QM_RW uint32_t slp_cfg

  Sleeping Configuration.
- QM_RW uint32_t pmnetcs

  Power Management Network (PMNet) Control and Status.
- QM_RW uint32_t vr_lock

  Voltage regulator lock.

5.45.1 Detailed Description

Power Management register map.

Definition at line 271 of file qm_soc_regs.h.

5.45.2 Field Documentation

5.45.2.1 QM_RW uint32_t qm_scss_pmu_reg_t::aon_vr

AON Voltage Regulator.

Definition at line 272 of file qm_soc_regs.h.

5.45.2.2 QM_RW uint32_t qm_scss_pmu_reg_t::p_sts

Processor Status.

Definition at line 276 of file qm_soc_regs.h.

5.45.2.3 QM_RW uint32_t qm_scss_pmu_reg_t::pm_lock

Power Management Lock.

Definition at line 281 of file qm_soc_regs.h.

5.45.2.4 QM_RW uint32_t qm_scss_pmu_reg_t::pm_wait

Power Management Wait.

Definition at line 274 of file qm_soc_regs.h.

5.45.2.5 QM_RW uint32_t qm_scss_pmu_reg_t::rstc

Reset Control.

Definition at line 278 of file qm_soc_regs.h.

5.45.2.6 QM_RW uint32_t qm_scss_pmu_reg_t::rsts

Reset Status.

Definition at line 279 of file qm_soc_regs.h.

5.46 qm_scss_pmux_reg_t Struct Reference

Pin MUX register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t pmux_pullup [1]

  Pin Mux Pullup.
- QM_RW uint32_t pmux_slew [1]

  Pin Mux Slew Rate.
- QM_RW uint32_t pmux_in_en [1]

  Pin Mux Input Enable.
- QM_RW uint32_t pmux_sel [2]

  Pin Mux Select.
- QM_RW uint32_t pmux_pullup_lock

  Pin Mux Pullup Lock.
- QM_RW uint32_t pmux_slew_lock

  Pin Mux Slew Rate Lock.
- QM_RW uint32_t pmux_sel_0_lock

  Pin Mux Select Lock 0.
- QM_RW uint32_t pmux_in_en_lock

  Pin Mux Slew Rate Lock.
- QM_RW uint32_t pmux_sel_lock [3]

  Pin Mux Select Lock.

5.46.1 Detailed Description

Pin MUX register map.

Definition at line 365 of file qm_soc_regs.h.

5.46.2 Field Documentation

5.46.2.1 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_in_en

Pin Mux Input Enable.

Definition at line 370 of file qm_soc_regs.h.

5.46.2.2 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_in_en_lock

Pin Mux Slew Rate Lock.

Definition at line 378 of file qm_soc_regs.h.

5.46.2.3 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_pullup

Pin Mux Pullup.

Definition at line 366 of file qm_soc_regs.h.

5.46.2.4 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_pullup_lock

Pin Mux Pullup Lock.

Definition at line 374 of file qm_soc_regs.h.

5.46.2.5 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_sel

Pin Mux Select.

Definition at line 372 of file qm_soc_regs.h.

5.46.2.6 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_sel_0_lock

Pin Mux Select Lock 0.

Definition at line 376 of file qm_soc_regs.h.

5.46.2.7 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_slew

Pin Mux Slew Rate.

Definition at line 368 of file qm_soc_regs.h.

5.46.2.8 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_slew_lock

Pin Mux Slew Rate Lock.

Definition at line 375 of file qm_soc_regs.h.

5.47 qm_scss_ss_reg_t Struct Reference

Sensor Subsystem register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t ss_cfg

  Sensor Subsystem Configuration.
- QM_RW uint32_t ss_sts

  Sensor Subsystem status.

5.47.1 Detailed Description

Sensor Subsystem register map.

Definition at line 461 of file qm_soc_regs.h.

5.48 qm_spi_async_transfer_t Struct Reference

SPI IRQ transfer type.

`#include <qm_spi.h>`

Data Fields

- uint8_t * tx
  Write data.
- uint8_t * rx
  Read data.
- uint16_t tx_len
  Write data Length.
- uint16_t rx_len
  Read buffer length.
- void(* callback)(void *data, int error, qm_spi_status_t status, uint16_t len)
  Transfer callback.
- void * callback_data
  Callback user data.

5.48.1 Detailed Description

SPI IRQ transfer type.

Definition at line 117 of file qm_spi.h.

5.48.2 Field Documentation

5.48.2.1 void(* qm_spi_async_transfer_t::callback)(void *data, int error, qm_spi_status_t status, uint16_t len)

Transfer callback.

Called after all data is transmitted/received or if the driver detects an error during the SPI transfer.

Parameters

in	*data*	The callback user data.
in	*error*	0 on success. Negative errno for possible error codes.
in	*status*	SPI driver status.
in	*len*	Length of the SPI transfer if successful, 0 otherwise.

Definition at line 136 of file qm_spi.h.

Referenced by qm_spi_irq_transfer_terminate().

5.48.2.2 void* qm_spi_async_transfer_t::callback_data

Callback user data.

Definition at line 138 of file qm_spi.h.

Referenced by qm_spi_irq_transfer_terminate().

5.48.2.3 uint8_t* qm_spi_async_transfer_t::rx

Read data.

Definition at line 119 of file qm_spi.h.

Referenced by qm_spi_dma_transfer().

5.48.2.4 uint16_t qm_spi_async_transfer_t::rx_len

Read buffer length.

Definition at line 121 of file qm_spi.h.

Referenced by qm_spi_dma_transfer(), and qm_spi_irq_transfer().

5.48.2.5 uint8_t* qm_spi_async_transfer_t::tx

Write data.

Definition at line 118 of file qm_spi.h.

Referenced by qm_spi_dma_transfer().

5.48.2.6 uint16_t qm_spi_async_transfer_t::tx_len

Write data Length.

Definition at line 120 of file qm_spi.h.

Referenced by qm_spi_dma_transfer(), and qm_spi_irq_transfer().

5.49 qm_spi_config_t Struct Reference

SPI configuration type.

`#include <qm_spi.h>`

Data Fields

- qm_spi_frame_size_t frame_size

  Frame Size.
- qm_spi_tmode_t transfer_mode

  Transfer mode (enum).
- qm_spi_bmode_t bus_mode

  Bus mode (enum).
- uint16_t clk_divider

  SCK = SPI_clock/clk_divider.

5.49.1 Detailed Description

SPI configuration type.

Definition at line 101 of file qm_spi.h.

5.49.2 Field Documentation

5.49.2.1 qm_spi_bmode_t qm_spi_config_t::bus_mode

Bus mode (enum).

Definition at line 104 of file qm_spi.h.

Referenced by qm_spi_set_config().

5.49.2.2 uint16_t qm_spi_config_t::clk_divider

SCK = SPI_clock/clk_divider.

A value of 0 will disable SCK.

Definition at line 111 of file qm_spi.h.

Referenced by qm_spi_set_config().

5.49.2.3 qm_spi_frame_size_t qm_spi_config_t::frame_size

Frame Size.

Definition at line 102 of file qm_spi.h.

Referenced by qm_spi_set_config().

5.49.2.4 qm_spi_tmode_t qm_spi_config_t::transfer_mode

Transfer mode (enum).

Definition at line 103 of file qm_spi.h.

Referenced by qm_spi_set_config().

5.50 qm_spi_reg_t Struct Reference

SPI register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t ctrlr0
  Control Register 0.
- QM_RW uint32_t ctrlr1
  Control Register 1.
- QM_RW uint32_t ssienr
  SSI Enable Register.
- QM_RW uint32_t mwcr
  Microwire Control Register.
- QM_RW uint32_t ser
  Slave Enable Register.
- QM_RW uint32_t baudr
  Baud Rate Select.
- QM_RW uint32_t txftlr
  Transmit FIFO Threshold Level.
- QM_RW uint32_t rxftlr
  Receive FIFO Threshold Level.
- QM_RW uint32_t txflr
  Transmit FIFO Level Register.
- QM_RW uint32_t rxflr
  Receive FIFO Level Register.
- QM_RW uint32_t sr
  Status Register.
- QM_RW uint32_t imr
  Interrupt Mask Register.

- QM_RW uint32_t isr

  Interrupt Status Register.
- QM_RW uint32_t risr

  Raw Interrupt Status Register.
- QM_RW uint32_t txoicr

  Tx FIFO Overflow Interrupt Clear Register.
- QM_RW uint32_t rxoicr

  Rx FIFO Overflow Interrupt Clear Register.
- QM_RW uint32_t rxuicr

  Rx FIFO Underflow Interrupt Clear Register.
- QM_RW uint32_t msticr

  Multi-Master Interrupt Clear Register.
- QM_RW uint32_t icr

  Interrupt Clear Register.
- QM_RW uint32_t dmacr

  DMA Control Register.
- QM_RW uint32_t dmatdlr

  DMA Transmit Data Level.
- QM_RW uint32_t dmardlr

  DMA Receive Data Level.
- QM_RW uint32_t idr

  Identification Register.
- QM_RW uint32_t ssi_comp_version

  coreKit Version ID register.
- QM_RW uint32_t dr [36]

  Data Register.
- QM_RW uint32_t rx_sample_dly

  RX Sample Delay Register.

5.50.1 Detailed Description

SPI register map.

Definition at line 653 of file qm_soc_regs.h.

5.50.2 Field Documentation

5.50.2.1 QM_RW uint32_t qm_spi_reg_t::baudr

Baud Rate Select.

Definition at line 659 of file qm_soc_regs.h.

Referenced by qm_spi_set_config().

5.50.2.2 QM_RW uint32_t qm_spi_reg_t::ctrlr0

Control Register 0.

Definition at line 654 of file qm_soc_regs.h.

Referenced by qm_spi_set_config().

5.50.2.3 QM_RW uint32_t qm_spi_reg_t::ctrlr1

Control Register 1.

Definition at line 655 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer(), qm_spi_irq_transfer(), and qm_spi_transfer().

5.50.2.4 QM_RW uint32_t qm_spi_reg_t::dmacr

DMA Control Register.

Definition at line 676 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer().

5.50.2.5 QM_RW uint32_t qm_spi_reg_t::dmardlr

DMA Receive Data Level.

Definition at line 678 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer().

5.50.2.6 QM_RW uint32_t qm_spi_reg_t::dmatdlr

DMA Transmit Data Level.

Definition at line 677 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer().

5.50.2.7 QM_RW uint32_t qm_spi_reg_t::dr

Data Register.

Definition at line 681 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer().

5.50.2.8 QM_RW uint32_t qm_spi_reg_t::icr

Interrupt Clear Register.

Definition at line 675 of file qm_soc_regs.h.

5.50.2.9 QM_RW uint32_t qm_spi_reg_t::idr

Identification Register.

Definition at line 679 of file qm_soc_regs.h.

5.50.2.10 QM_RW uint32_t qm_spi_reg_t::imr

Interrupt Mask Register.

Definition at line 665 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer(), qm_spi_irq_transfer(), qm_spi_irq_transfer_terminate(), and qm_spi_-transfer().

5.50.2.11 QM_RW uint32_t qm_spi_reg_t::isr

Interrupt Status Register.

Definition at line 666 of file qm_soc_regs.h.

5.50.2.12 QM_RW uint32_t qm_spi_reg_t::msticr

Multi-Master Interrupt Clear Register.

Definition at line 674 of file qm_soc_regs.h.

5.50.2.13 QM_RW uint32_t qm_spi_reg_t::mwcr

Microwire Control Register.

Definition at line 657 of file qm_soc_regs.h.

5.50.2.14 QM_RW uint32_t qm_spi_reg_t::risr

Raw Interrupt Status Register.

Definition at line 667 of file qm_soc_regs.h.

Referenced by qm_spi_get_status(), and qm_spi_transfer().

5.50.2.15 QM_RW uint32_t qm_spi_reg_t::rx_sample_dly

RX Sample Delay Register.

Definition at line 682 of file qm_soc_regs.h.

5.50.2.16 QM_RW uint32_t qm_spi_reg_t::rxflr

Receive FIFO Level Register.

Definition at line 663 of file qm_soc_regs.h.

5.50.2.17 QM_RW uint32_t qm_spi_reg_t::rxftlr

Receive FIFO Threshold Level.

Definition at line 661 of file qm_soc_regs.h.

Referenced by qm_spi_irq_transfer().

5.50.2.18 QM_RW uint32_t qm_spi_reg_t::rxoicr

Rx FIFO Overflow Interrupt Clear Register.

Definition at line 671 of file qm_soc_regs.h.

Referenced by qm_spi_transfer().

5.50.2.19 QM_RW uint32_t qm_spi_reg_t::rxuicr

Rx FIFO Underflow Interrupt Clear Register.

Definition at line 673 of file qm_soc_regs.h.

5.50.2.20 QM_RW uint32_t qm_spi_reg_t::ser

Slave Enable Register.

Definition at line 658 of file qm_soc_regs.h.

5.50.2.21 QM_RW uint32_t qm_spi_reg_t::sr

Status Register.

Definition at line 664 of file qm_soc_regs.h.

Referenced by qm_spi_get_status(), and qm_spi_transfer().

5.50.2.22 QM_RW uint32_t qm_spi_reg_t::ssi_comp_version

coreKit Version ID register.

coreKit Version ID register

Definition at line 680 of file qm_soc_regs.h.

5.50.2.23 QM_RW uint32_t qm_spi_reg_t::ssienr

SSI Enable Register.

Definition at line 656 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer(), qm_spi_irq_transfer(), qm_spi_irq_transfer_terminate(), and qm_spi_-transfer().

5.50.2.24 QM_RW uint32_t qm_spi_reg_t::txflr

Transmit FIFO Level Register.

Definition at line 662 of file qm_soc_regs.h.

5.50.2.25 QM_RW uint32_t qm_spi_reg_t::txftlr

Transmit FIFO Threshold Level.

Definition at line 660 of file qm_soc_regs.h.

Referenced by qm_spi_irq_transfer().

5.50.2.26 QM_RW uint32_t qm_spi_reg_t::txoicr

Tx FIFO Overflow Interrupt Clear Register.

Definition at line 669 of file qm_soc_regs.h.

5.51 qm_spi_transfer_t Struct Reference

SPI transfer type.

`#include <qm_spi.h>`

Data Fields

- uint8_t * tx

  Write Data.
- uint8_t * rx

  Read Data.
- uint16_t tx_len

  Write Data Length.
- uint16_t rx_len

  Receive Data Length.

5.51.1 Detailed Description

SPI transfer type.

Definition at line 144 of file qm_spi.h.

5.51.2 Field Documentation

5.51.2.1 uint8_t* qm_spi_transfer_t::rx

Read Data.

Definition at line 146 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.51.2.2 uint16_t qm_spi_transfer_t::rx_len

Receive Data Length.

Definition at line 148 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.51.2.3 uint8_t* qm_spi_transfer_t::tx

Write Data.

Definition at line 145 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.51.2.4 uint16_t qm_spi_transfer_t::tx_len

Write Data Length.

Definition at line 147 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.52 qm_ss_adc_config_t Struct Reference

SS ADC configuration type.

`#include <qm_ss_adc.h>`

Data Fields

- uint8_t window

  Sample interval in ADC clock cycles, defines the period to wait between the start of each sample and can be in the range [(resolution+2) - 255].
- qm_ss_adc_resolution_t resolution

  12, 10, 8, 6-bit resolution.

5.52.1 Detailed Description

SS ADC configuration type.

Definition at line 98 of file qm_ss_adc.h.

5.52.2 Field Documentation

5.52.2.1 qm_ss_adc_resolution_t qm_ss_adc_config_t::resolution

12, 10, 8, 6-bit resolution.

Definition at line 105 of file qm_ss_adc.h.

Referenced by qm_ss_adc_set_config().

5.53 qm_ss_adc_xfer_t Struct Reference

SS ADC transfer type.

`#include <qm_ss_adc.h>`

Data Fields

- qm_ss_adc_channel_t * ch

  Channel sequence array (1-32 channels).
- uint8_t ch_len

  Number of channels in the above array.
- qm_ss_adc_sample_t * samples

  Array to store samples.
- uint32_t samples_len

  Length of sample array.
- void(* callback)(void *data, int error, qm_ss_adc_status_t status, qm_ss_adc_cb_source_t source)

  Transfer callback.
- void * callback_data

  Callback user data.

5.53.1 Detailed Description

SS ADC transfer type.

Definition at line 111 of file qm_ss_adc.h.

5.53.2 Field Documentation

5.53.2.1 void(* qm_ss_adc_xfer_t::callback)(void *data, int error, qm_ss_adc_status_t status, qm_ss_adc_cb_source_t source)

Transfer callback.

Called when a conversion is performed or an error is detected.

Parameters

in	*data*	The callback user data.
in	*error*	0 on success. Negative errno for possible error codes.
in	*status*	ADC status.
in	*source*	Interrupt callback source.

Definition at line 128 of file qm_ss_adc.h.

5.53.2.2 void* qm_ss_adc_xfer_t::callback_data

Callback user data.

Definition at line 130 of file qm_ss_adc.h.

5.53.2.3 qm_ss_adc_channel_t* qm_ss_adc_xfer_t::ch

Channel sequence array (1-32 channels).

Definition at line 112 of file qm_ss_adc.h.

Referenced by qm_ss_adc_convert(), and qm_ss_adc_irq_convert().

5.53.2.4 uint8_t qm_ss_adc_xfer_t::ch_len

Number of channels in the above array.

Definition at line 113 of file qm_ss_adc.h.

Referenced by qm_ss_adc_convert(), and qm_ss_adc_irq_convert().

5.53.2.5 qm_ss_adc_sample_t* qm_ss_adc_xfer_t::samples

Array to store samples.

Definition at line 114 of file qm_ss_adc.h.

Referenced by qm_ss_adc_convert(), and qm_ss_adc_irq_convert().

5.53.2.6 uint32_t qm_ss_adc_xfer_t::samples_len

Length of sample array.

Definition at line 115 of file qm_ss_adc.h.

Referenced by qm_ss_adc_convert(), and qm_ss_adc_irq_convert().

5.54 qm_ss_gpio_port_config_t Struct Reference

GPIO port configuration type.

`#include <qm_ss_gpio.h>`

Data Fields

- uint32_t direction
  SS GPIO direction, 0b: input, 1b: output.
- uint32_t int_en
  Interrupt enable.
- uint32_t int_type
  Interrupt type, 0b: level; 1b: edge.
- uint32_t int_polarity
  Interrupt polarity, 0b: low, 1b: high.
- uint32_t int_debounce
  Debounce on/off.
- void(* callback)(void *data, uint32_t int_status)
  User callback.
- void * callback_data
  Callback user data.

5.54.1 Detailed Description

GPIO port configuration type.

Each bit in the registers control a GPIO pin.

Definition at line 32 of file qm_ss_gpio.h.

5.54.2 Field Documentation

5.54.2.1 void(∗ qm_ss_gpio_port_config_t::callback)(void ∗data, uint32_t int_status)

User callback.

Called for any interrupt on the Sensor Subsystem GPIO.

Parameters

in	*data*	The callback user data.
in	*int_status*	Bitfield of triggered pins.

Definition at line 47 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.54.2.2 void∗ qm_ss_gpio_port_config_t::callback_data

Callback user data.

Definition at line 48 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.54.2.3 uint32_t qm_ss_gpio_port_config_t::direction

SS GPIO direction, 0b: input, 1b: output.

Definition at line 33 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.54.2.4 uint32_t qm_ss_gpio_port_config_t::int_debounce

Debounce on/off.

Definition at line 37 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.54.2.5 uint32_t qm_ss_gpio_port_config_t::int_en

Interrupt enable.

Definition at line 34 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.54.2.6 uint32_t qm_ss_gpio_port_config_t::int_polarity

Interrupt polarity, 0b: low, 1b: high.

Definition at line 36 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.54.2.7 uint32_t qm_ss_gpio_port_config_t::int_type

Interrupt type, 0b: level; 1b: edge.

Definition at line 35 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.55 qm_ss_i2c_config_t Struct Reference

QM SS I2C configuration type.

```
#include <qm_ss_i2c.h>
```

Data Fields

- qm_ss_i2c_speed_t speed
  Standard, Fast Mode.
- qm_ss_i2c_addr_t address_mode
  7 or 10 bit addressing.

5.55.1 Detailed Description

QM SS I2C configuration type.

Definition at line 70 of file qm_ss_i2c.h.

5.55.2 Field Documentation

5.55.2.1 qm_ss_i2c_addr_t qm_ss_i2c_config_t::address_mode

7 or 10 bit addressing.

Definition at line 72 of file qm_ss_i2c.h.

Referenced by qm_ss_i2c_set_config().

5.55.2.2 qm_ss_i2c_speed_t qm_ss_i2c_config_t::speed

Standard, Fast Mode.

Definition at line 71 of file qm_ss_i2c.h.

Referenced by qm_ss_i2c_set_config().

5.56 qm_ss_i2c_transfer_t Struct Reference

QM SS I2C transfer type.

```
#include <qm_ss_i2c.h>
```

Data Fields

- uint8_t * tx
  Write data.
- uint32_t tx_len
  Write data length.
- uint8_t * rx
  Read data.
- uint32_t rx_len
  Read buffer length.
- bool stop
  Generate master STOP.
- void(* callback)(void *data, int rc, qm_ss_i2c_status_t status, uint32_t len)
  User callback.
- void * callback_data
  User callback data.

5.56.1 Detailed Description

QM SS I2C transfer type.

- 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.

Definition at line 82 of file qm_ss_i2c.h.

5.56.2 Field Documentation

5.56.2.1 void(∗ qm_ss_i2c_transfer_t::callback)(void ∗data, int rc, qm_ss_i2c_status_t status, uint32_t len)

User callback.

Parameters

in	*data*	User defined data.
in	*rc*	0 on success. Negative errno for possible error codes.
in	*status*	I2C status.
in	*len*	Length of the transfer if succesfull, 0 otherwise.

Definition at line 98 of file qm_ss_i2c.h.

5.56.2.2 void∗ qm_ss_i2c_transfer_t::callback_data

User callback data.

Definition at line 100 of file qm_ss_i2c.h.

5.56.2.3 uint8_t∗ qm_ss_i2c_transfer_t::rx

Read data.

Definition at line 85 of file qm_ss_i2c.h.

5.56.2.4 uint32_t qm_ss_i2c_transfer_t::rx_len

Read buffer length.

Definition at line 86 of file qm_ss_i2c.h.

Referenced by qm_ss_i2c_master_irq_transfer().

5.56.2.5 bool qm_ss_i2c_transfer_t::stop

Generate master STOP.

Definition at line 87 of file qm_ss_i2c.h.

5.56.2.6 uint8_t∗ qm_ss_i2c_transfer_t::tx

Write data.

Definition at line 83 of file qm_ss_i2c.h.

5.56.2.7 uint32_t qm_ss_i2c_transfer_t::tx_len

Write data length.

Definition at line 84 of file qm_ss_i2c.h.

5.57 qm_ss_spi_async_transfer_t Struct Reference

SPI IRQ transfer type.

`#include <qm_ss_spi.h>`

Data Fields

- uint8_t * tx
  Write data buffer pointer.
- uint16_t tx_len
  Write data length.
- uint8_t * rx
  Read data buffer pointer.
- uint16_t rx_len
  Read buffer length.
- void(* callback)(void *data, int error, qm_ss_spi_status_t status, uint16_t len)
  Transfer callback.
- void * callback_data
  Callback user data.

5.57.1 Detailed Description

SPI IRQ transfer type.

Definition at line 129 of file qm_ss_spi.h.

5.57.2 Field Documentation

5.57.2.1 void(* qm_ss_spi_async_transfer_t::callback)(void *data, int error, qm_ss_spi_status_t status, uint16_t len)

Transfer callback.

Called after all data is transmitted/received or if the driver detects an error during the SPI transfer.

Parameters

in	*data*	The callback user data.
in	*error*	0 on success. Negative errno for possible error codes.
in	*status*	The SPI module status.
in	*len*	The amount of frames transmitted.

Definition at line 147 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer_terminate().

5.57.2.2 void* qm_ss_spi_async_transfer_t::callback_data

Callback user data.

Definition at line 149 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer_terminate().

5.57.2.3 uint8_t* qm_ss_spi_async_transfer_t::rx

Read data buffer pointer.

Definition at line 132 of file qm_ss_spi.h.

5.57.2.4 uint16_t qm_ss_spi_async_transfer_t::rx_len

Read buffer length.

Definition at line 133 of file qm_ss_spi.h.

Referenced by qm_ss_spi_irq_transfer(), and qm_ss_spi_transfer_terminate().

5.57.2.5 uint8_t* qm_ss_spi_async_transfer_t::tx

Write data buffer pointer.

Definition at line 130 of file qm_ss_spi.h.

5.57.2.6 uint16_t qm_ss_spi_async_transfer_t::tx_len

Write data length.

Definition at line 131 of file qm_ss_spi.h.

Referenced by qm_ss_spi_irq_transfer(), and qm_ss_spi_transfer_terminate().

5.58 qm_ss_spi_config_t Struct Reference

SPI configuration type.

`#include <qm_ss_spi.h>`

Data Fields

- qm_ss_spi_frame_size_t frame_size
  Frame Size.
- qm_ss_spi_tmode_t transfer_mode
  Transfer mode (enum).
- qm_ss_spi_bmode_t bus_mode
  Bus mode (enum).
- uint16_t clk_divider
  Clock divider.

5.58.1 Detailed Description

SPI configuration type.

Definition at line 112 of file qm_ss_spi.h.

5.58.2 Field Documentation

5.58.2.1 qm_ss_spi_bmode_t qm_ss_spi_config_t::bus_mode

Bus mode (enum).

Definition at line 115 of file qm_ss_spi.h.

Referenced by qm_ss_spi_set_config().

5.58.2.2 uint16_t qm_ss_spi_config_t::clk_divider

Clock divider.

The clock divider sets the SPI speed on the interface. A value of 0 will disable SCK. The LSB of this value is ignored.

Definition at line 123 of file qm_ss_spi.h.

Referenced by qm_ss_spi_set_config().

5.58.2.3 qm_ss_spi_frame_size_t qm_ss_spi_config_t::frame_size

Frame Size.

Definition at line 113 of file qm_ss_spi.h.

Referenced by qm_ss_spi_set_config().

5.58.2.4 qm_ss_spi_tmode_t qm_ss_spi_config_t::transfer_mode

Transfer mode (enum).

Definition at line 114 of file qm_ss_spi.h.

Referenced by qm_ss_spi_set_config().

5.59 qm_ss_spi_transfer_t Struct Reference

SPI transfer type.

`#include <qm_ss_spi.h>`

Data Fields

- uint8_t * tx

  Write data buffer pointer.
- uint16_t tx_len

  Write data length.
- uint8_t * rx

  Read data buffer pointer.
- uint16_t rx_len

  Read buffer length.

5.59.1 Detailed Description

SPI transfer type.

Definition at line 155 of file qm_ss_spi.h.

5.59.2 Field Documentation

5.59.2.1 uint8_t* qm_ss_spi_transfer_t::rx

Read data buffer pointer.

Definition at line 158 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.59.2.2 uint16_t qm_ss_spi_transfer_t::rx_len

Read buffer length.

Definition at line 159 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.59.2.3 uint8_t* qm_ss_spi_transfer_t::tx

Write data buffer pointer.

Definition at line 156 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.59.2.4 uint16_t qm_ss_spi_transfer_t::tx_len

Write data length.

Definition at line 157 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.60 qm_ss_timer_config_t Struct Reference

Sensor Subsystem Timer Configuration Type.

`#include <qm_ss_timer.h>`

Data Fields

- bool watchdog_mode

  Watchdog mode.
- bool inc_run_only

  Increments in run state only.
- bool int_en

  Interrupt enable.
- uint32_t count

  Final count value.
- void(* callback)(void *data)

  User callback.
- void * callback_data

  Callback user data.

5.60.1 Detailed Description

Sensor Subsystem Timer Configuration Type.

Definition at line 21 of file qm_ss_timer.h.

5.60.2 Field Documentation

5.60.2.1 void(* qm_ss_timer_config_t::callback)(void *data)

User callback.

Called for any interrupt on the Sensor Subsystem Timer.

Parameters

in	*data*	The callback user data.

Definition at line 43 of file qm_ss_timer.h.

Referenced by qm_ss_timer_set_config().

5.60.2.2 void* qm_ss_timer_config_t::callback_data

Callback user data.

Definition at line 44 of file qm_ss_timer.h.

Referenced by qm_ss_timer_set_config().

5.60.2.3 uint32_t qm_ss_timer_config_t::count

Final count value.

Definition at line 34 of file qm_ss_timer.h.

Referenced by qm_ss_timer_set_config().

5.60.2.4 bool qm_ss_timer_config_t::inc_run_only

Increments in run state only.

If this field is set to 0, the timer will count in both halt state and running state. When set to 1, this will only increment in running state.

Definition at line 32 of file qm_ss_timer.h.

Referenced by qm_ss_timer_set_config().

5.60.2.5 bool qm_ss_timer_config_t::int_en

Interrupt enable.

Definition at line 33 of file qm_ss_timer.h.

Referenced by qm_ss_timer_set_config().

5.60.2.6 bool qm_ss_timer_config_t::watchdog_mode

Watchdog mode.

Definition at line 22 of file qm_ss_timer.h.

Referenced by qm_ss_timer_set_config().

5.61 qm_uart_config_t Struct Reference

UART configuration type.

`#include <qm_uart.h>`

Data Fields

- qm_uart_lc_t line_control
  Line control (enum).
- uint32_t baud_divisor
  Baud Divisor.
- bool hw_fc
  Hardware Automatic Flow Control.
- bool int_en
  Interrupt enable.

5.61.1 Detailed Description

UART configuration type.

Definition at line 161 of file qm_uart.h.

5.61.2 Field Documentation

5.61.2.1 uint32_t qm_uart_config_t::baud_divisor

Baud Divisor.

Definition at line 163 of file qm_uart.h.

Referenced by qm_uart_set_config().

5.61.2.2 bool qm_uart_config_t::hw_fc

Hardware Automatic Flow Control.

Definition at line 164 of file qm_uart.h.

Referenced by qm_uart_set_config().

5.61.2.3 bool qm_uart_config_t::int_en

Interrupt enable.

Definition at line 165 of file qm_uart.h.

5.61.2.4 qm_uart_lc_t qm_uart_config_t::line_control

Line control (enum).

Definition at line 162 of file qm_uart.h.

Referenced by qm_uart_set_config().

5.62 qm_uart_reg_t Struct Reference

UART register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t rbr_thr_dll
  Rx Buffer/ Tx Holding/ Div Latch Low.
- QM_RW uint32_t ier_dlh
  Interrupt Enable / Divisor Latch High.
- QM_RW uint32_t iir_fcr
  Interrupt Identification / FIFO Control.
- QM_RW uint32_t lcr
  Line Control.
- QM_RW uint32_t mcr
  MODEM Control.
- QM_RW uint32_t lsr
  Line Status.
- QM_RW uint32_t msr
  MODEM Status.
- QM_RW uint32_t scr
  Scratchpad.
- QM_RW uint32_t usr

UART Status.

- QM_RW uint32_t htx

  Halt Transmission.
- QM_RW uint32_t dmasa

  DMA Software Acknowledge.
- QM_RW uint32_t tcr

  Transceiver Control Register.
- QM_RW uint32_t de_en

  Driver Output Enable Register.
- QM_RW uint32_t re_en

  Receiver Output Enable Register.
- QM_RW uint32_t det

  Driver Output Enable Timing Register.
- QM_RW uint32_t tat

  TurnAround Timing Register.
- QM_RW uint32_t dlf

  Divisor Latch Fraction.
- QM_RW uint32_t rar

  Receive Address Register.
- QM_RW uint32_t tar

  Transmit Address Register.
- QM_RW uint32_t lcr_ext

  Line Extended Control Register.

5.62.1 Detailed Description

UART register map.

Definition at line 601 of file qm_soc_regs.h.

5.62.2 Field Documentation

5.62.2.1 QM_RW uint32_t qm_uart_reg_t::de_en

Driver Output Enable Register.

Definition at line 617 of file qm_soc_regs.h.

5.62.2.2 QM_RW uint32_t qm_uart_reg_t::det

Driver Output Enable Timing Register.

Definition at line 619 of file qm_soc_regs.h.

5.62.2.3 QM_RW uint32_t qm_uart_reg_t::dlf

Divisor Latch Fraction.

Definition at line 621 of file qm_soc_regs.h.

Referenced by qm_uart_set_config().

5.62.2.4 QM_RW uint32_t qm_uart_reg_t::dmasa

DMA Software Acknowledge.

Definition at line 615 of file qm_soc_regs.h.

5.62.2.5 QM_RW uint32_t qm_uart_reg_t::htx

Halt Transmission.

Definition at line 614 of file qm_soc_regs.h.

5.62.2.6 QM_RW uint32_t qm_uart_reg_t::ier_dlh

Interrupt Enable / Divisor Latch High.

Definition at line 604 of file qm_soc_regs.h.

Referenced by qm_uart_irq_read(), qm_uart_irq_read_terminate(), qm_uart_irq_write(), qm_uart_irq_write_terminate(), and qm_uart_set_config().

5.62.2.7 QM_RW uint32_t qm_uart_reg_t::iir_fcr

Interrupt Identification / FIFO Control.

Definition at line 605 of file qm_soc_regs.h.

Referenced by qm_uart_dma_read(), qm_uart_dma_write(), qm_uart_irq_read(), qm_uart_irq_write(), and qm_uart_set_config().

5.62.2.8 QM_RW uint32_t qm_uart_reg_t::lcr

Line Control.

Definition at line 606 of file qm_soc_regs.h.

Referenced by qm_uart_set_config().

5.62.2.9 QM_RW uint32_t qm_uart_reg_t::lcr_ext

Line Extended Control Register.

Definition at line 624 of file qm_soc_regs.h.

5.62.2.10 QM_RW uint32_t qm_uart_reg_t::lsr

Line Status.

Definition at line 608 of file qm_soc_regs.h.

Referenced by qm_uart_get_status(), qm_uart_read(), qm_uart_write(), and qm_uart_write_buffer().

5.62.2.11 QM_RW uint32_t qm_uart_reg_t::mcr

MODEM Control.

Definition at line 607 of file qm_soc_regs.h.

Referenced by qm_uart_set_config().

5.62.2.12 QM_RW uint32_t qm_uart_reg_t::msr

MODEM Status.

Definition at line 609 of file qm_soc_regs.h.

5.62.2.13 QM_RW uint32_t qm_uart_reg_t::rar

Receive Address Register.

Definition at line 622 of file qm_soc_regs.h.

5.62.2.14 QM_RW uint32_t qm_uart_reg_t::rbr_thr_dll

Rx Buffer/ Tx Holding/ Div Latch Low.

Definition at line 603 of file qm_soc_regs.h.

Referenced by qm_uart_dma_read(), qm_uart_dma_write(), qm_uart_read(), qm_uart_read_non_block(), qm_uart_set_config(), qm_uart_write(), qm_uart_write_buffer(), and qm_uart_write_non_block().

5.62.2.15 QM_RW uint32_t qm_uart_reg_t::re_en

Receiver Output Enable Register.

Definition at line 618 of file qm_soc_regs.h.

5.62.2.16 QM_RW uint32_t qm_uart_reg_t::scr

Scratchpad.

Definition at line 610 of file qm_soc_regs.h.

Referenced by qm_uart_get_status().

5.62.2.17 QM_RW uint32_t qm_uart_reg_t::tar

Transmit Address Register.

Definition at line 623 of file qm_soc_regs.h.

5.62.2.18 QM_RW uint32_t qm_uart_reg_t::tat

TurnAround Timing Register.

Definition at line 620 of file qm_soc_regs.h.

5.62.2.19 QM_RW uint32_t qm_uart_reg_t::tcr

Transceiver Control Register.

Definition at line 616 of file qm_soc_regs.h.

5.62.2.20 QM_RW uint32_t qm_uart_reg_t::usr

UART Status.

Definition at line 612 of file qm_soc_regs.h.

5.63 qm_uart_transfer_t Struct Reference

UART asynchronous transfer structure.

`#include <qm_uart.h>`

Data Fields

- uint8_t * data
  Pre-allocated write or read buffer.
- uint32_t data_len
  Number of bytes to transfer.
- void(* callback)(void *data, int error, qm_uart_status_t status, uint32_t len)
  Transfer callback.
- void * callback_data
  Callback identifier.

5.63.1 Detailed Description

UART asynchronous transfer structure.

Definition at line 171 of file qm_uart.h.

5.63.2 Field Documentation

5.63.2.1 void(∗ qm_uart_transfer_t::callback)(void ∗data, int error, qm_uart_status_t status, uint32_t len)

Transfer callback.

Parameters

in	*data*	Callback user data.
in	*error*	0 on success. Negative errno for possible error codes.
in	*status*	UART module status
in	*len*	Length of the UART transfer if successful, 0 otherwise.

Definition at line 184 of file qm_uart.h.

Referenced by qm_uart_irq_read_terminate(), and qm_uart_irq_write_terminate().

5.63.2.2 void∗ qm_uart_transfer_t::callback_data

Callback identifier.

Definition at line 186 of file qm_uart.h.

Referenced by qm_uart_irq_read_terminate(), and qm_uart_irq_write_terminate().

5.63.2.3 uint8_t∗ qm_uart_transfer_t::data

Pre-allocated write or read buffer.

Definition at line 172 of file qm_uart.h.

Referenced by qm_uart_dma_read(), and qm_uart_dma_write().

5.63.2.4 uint32_t qm_uart_transfer_t::data_len

Number of bytes to transfer.

Definition at line 173 of file qm_uart.h.

Referenced by qm_uart_dma_read(), and qm_uart_dma_write().

5.64 qm_wdt_config_t Struct Reference

QM WDT configuration type.

`#include <qm_wdt.h>`

Data Fields

- qm_wdt_clock_timeout_cycles_t timeout
  Timeout in cycles.
- qm_wdt_mode_t mode
  Watchdog response mode.
- void(∗ callback)(void ∗data)
  User callback.
- void ∗ callback_data

Callback user data.

5.64.1 Detailed Description

QM WDT configuration type.

Definition at line 72 of file qm_wdt.h.

5.64.2 Field Documentation

5.64.2.1 void(∗ qm_wdt_config_t::callback)(void ∗data)

User callback.

param[in] data Callback user data.

Definition at line 81 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.64.2.2 void∗ qm_wdt_config_t::callback_data

Callback user data.

Definition at line 82 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.64.2.3 qm_wdt_mode_t qm_wdt_config_t::mode

Watchdog response mode.

Definition at line 74 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.64.2.4 qm_wdt_clock_timeout_cycles_t qm_wdt_config_t::timeout

Timeout in cycles.

Definition at line 73 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.65 qm_wdt_reg_t Struct Reference

Watchdog timer register map.

`#include <qm_soc_regs.h>`

Data Fields

- QM_RW uint32_t wdt_cr
  Control Register.
- QM_RW uint32_t wdt_torr
  Timeout Range Register.
- QM_RW uint32_t wdt_ccvr
  Current Counter Value Register.
- QM_RW uint32_t wdt_crr
  Current Restart Register.
- QM_RW uint32_t wdt_stat

Interrupt Status Register.

- QM_RW uint32_t wdt_eoi

  Interrupt Clear Register.
- QM_RW uint32_t wdt_comp_param_5

  Component Parameters.
- QM_RW uint32_t wdt_comp_param_4

  Component Parameters.
- QM_RW uint32_t wdt_comp_param_3

  Component Parameters.
- QM_RW uint32_t wdt_comp_param_2

  Component Parameters.
- QM_RW uint32_t wdt_comp_param_1

  Component Parameters Register 1.
- QM_RW uint32_t wdt_comp_version

  Component Version Register.
- QM_RW uint32_t wdt_comp_type

  Component Type Register.

5.65.1 Detailed Description

Watchdog timer register map.

Definition at line 561 of file qm_soc_regs.h.

5.65.2 Field Documentation

5.65.2.1 QM_RW uint32_t qm_wdt_reg_t::wdt_ccvr

Current Counter Value Register.

Definition at line 564 of file qm_soc_regs.h.

5.65.2.2 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_1

Component Parameters Register 1.

Definition at line 573 of file qm_soc_regs.h.

5.65.2.3 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_2

Component Parameters.

Definition at line 571 of file qm_soc_regs.h.

5.65.2.4 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_3

Component Parameters.

Definition at line 570 of file qm_soc_regs.h.

5.65.2.5 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_4

Component Parameters.

Definition at line 569 of file qm_soc_regs.h.

5.65.2.6 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_5

Component Parameters.

Definition at line 568 of file qm_soc_regs.h.

5.65.2.7 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_type

Component Type Register.

Definition at line 575 of file qm_soc_regs.h.

5.65.2.8 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_version

Component Version Register.

Definition at line 574 of file qm_soc_regs.h.

5.65.2.9 QM_RW uint32_t qm_wdt_reg_t::wdt_cr

Control Register.

Definition at line 562 of file qm_soc_regs.h.

5.65.2.10 QM_RW uint32_t qm_wdt_reg_t::wdt_crr

Current Restart Register.

Definition at line 565 of file qm_soc_regs.h.

5.65.2.11 QM_RW uint32_t qm_wdt_reg_t::wdt_eoi

Interrupt Clear Register.

Definition at line 567 of file qm_soc_regs.h.

5.65.2.12 QM_RW uint32_t qm_wdt_reg_t::wdt_stat

Interrupt Status Register.

Definition at line 566 of file qm_soc_regs.h.

5.65.2.13 QM_RW uint32_t qm_wdt_reg_t::wdt_torr

Timeout Range Register.

Definition at line 563 of file qm_soc_regs.h.

Index