



Intel® Quark™ Microcontroller Software Interface 1.4.0

March, 2017

Contents

1	Licensing information	2
2	Module Index	2
2.1	Modules	2
3	Data Structure Index	4
3.1	Data Structures	4
4	Module Documentation	8
4.1	Error Codes	8
4.2	Quark D2000 ADC	9
4.2.1	Detailed Description	10
4.2.2	Enumeration Type Documentation	10
4.2.3	Function Documentation	11
4.3	Always-on Counters	16
4.3.1	Detailed Description	16
4.3.2	Enumeration Type Documentation	17
4.3.3	Function Documentation	17
4.4	Analog Comparator	21
4.4.1	Detailed Description	21
4.4.2	Function Documentation	21
4.5	DMA	22
4.5.1	Detailed Description	23
4.5.2	Enumeration Type Documentation	23
4.5.3	Function Documentation	24
4.6	Flash	30
4.6.1	Detailed Description	30
4.6.2	Enumeration Type Documentation	30
4.6.3	Function Documentation	31
4.7	FPR	37
4.7.1	Detailed Description	37
4.7.2	Enumeration Type Documentation	37
4.7.3	Function Documentation	38
4.8	GPIO	42
4.8.1	Detailed Description	42
4.8.2	Enumeration Type Documentation	42
4.8.3	Function Documentation	43
4.9	I2C	47
4.9.1	Detailed Description	48

4.9.2	Enumeration Type Documentation	48
4.9.3	Function Documentation	50
4.10	I2S	58
4.10.1	Detailed Description	59
4.10.2	Enumeration Type Documentation	59
4.10.3	Function Documentation	61
4.11	Identification	64
4.11.1	Detailed Description	64
4.11.2	Function Documentation	64
4.12	Initialisation	65
4.12.1	Detailed Description	65
4.12.2	Enumeration Type Documentation	65
4.12.3	Function Documentation	65
4.13	Interrupt	66
4.13.1	Detailed Description	66
4.13.2	Function Documentation	66
4.14	ISR	68
4.14.1	Detailed Description	69
4.14.2	Function Documentation	69
4.15	Mailbox	78
4.15.1	Detailed Description	78
4.15.2	Typedef Documentation	78
4.15.3	Enumeration Type Documentation	79
4.15.4	Function Documentation	79
4.16	MPR	82
4.16.1	Detailed Description	82
4.16.2	Function Documentation	82
4.17	PIC Timer	85
4.17.1	Detailed Description	85
4.17.2	Enumeration Type Documentation	85
4.17.3	Function Documentation	85
4.18	Pin Muxing setup	89
4.18.1	Detailed Description	89
4.18.2	Enumeration Type Documentation	89
4.18.3	Function Documentation	92
4.19	PWM / Timer	94
4.19.1	Detailed Description	94
4.19.2	Enumeration Type Documentation	94
4.19.3	Function Documentation	95
4.20	RTC	99

4.20.1	Detailed Description	99
4.20.2	Function Documentation	99
4.21	SPI	102
4.21.1	Detailed Description	103
4.21.2	Enumeration Type Documentation	103
4.21.3	Function Documentation	105
4.22	SS ADC	112
4.22.1	Detailed Description	113
4.22.2	Enumeration Type Documentation	113
4.22.3	Function Documentation	115
4.23	SS GPIO	121
4.23.1	Detailed Description	121
4.23.2	Enumeration Type Documentation	121
4.23.3	Function Documentation	122
4.24	SS I2C	126
4.24.1	Detailed Description	127
4.24.2	Enumeration Type Documentation	127
4.24.3	Function Documentation	128
4.25	SS Interrupt	132
4.25.1	Detailed Description	132
4.25.2	Function Documentation	132
4.26	SS ISR	134
4.26.1	Detailed Description	135
4.26.2	Function Documentation	135
4.27	SS SPI	140
4.27.1	Detailed Description	141
4.27.2	Enumeration Type Documentation	141
4.27.3	Function Documentation	143
4.28	SS Timer	147
4.28.1	Detailed Description	147
4.28.2	Function Documentation	147
4.29	UART	149
4.29.1	Detailed Description	150
4.29.2	Enumeration Type Documentation	150
4.29.3	Function Documentation	151
4.30	USB	160
4.30.1	Detailed Description	161
4.30.2	Typedef Documentation	161
4.30.3	Enumeration Type Documentation	161
4.30.4	Function Documentation	162

4.31	Version	168
4.31.1	Detailed Description	168
4.31.2	Function Documentation	168
4.32	WDT	169
4.32.1	Detailed Description	169
4.32.2	Enumeration Type Documentation	169
4.32.3	Function Documentation	169
4.33	SOC_WATCH	172
4.33.1	Detailed Description	173
4.33.2	Enumeration Type Documentation	173
4.33.3	Function Documentation	175
4.34	SS Clock	176
4.34.1	Detailed Description	176
4.34.2	Enumeration Type Documentation	176
4.34.3	Function Documentation	177
4.35	Clock Management	181
4.35.1	Detailed Description	183
4.35.2	Enumeration Type Documentation	183
4.35.3	Function Documentation	189
4.36	Quark D2000 Flash Layout	195
4.36.1	Detailed Description	195
4.37	Quark D2000 Power states	196
4.37.1	Detailed Description	196
4.37.2	Enumeration Type Documentation	196
4.37.3	Function Documentation	196
4.38	Quark(TM) D2000 Retention Alternator Regulator (RAR)	199
4.38.1	Detailed Description	199
4.38.2	Enumeration Type Documentation	199
4.38.3	Function Documentation	199
4.39	SoC Interrupt Router (D2000)	201
4.39.1	Detailed Description	201
4.40	SoC Interrupts (D2000)	202
4.41	SoC Registers (D2000)	203
4.41.1	Detailed Description	208
4.41.2	Enumeration Type Documentation	208
4.41.3	Variable Documentation	212
4.42	Quark SE Flash Layout	213
4.42.1	Detailed Description	213
4.43	Quark SE SoC Power states	214
4.43.1	Detailed Description	214

4.43.2	Function Documentation	214
4.44	Quark SE Host Power states	217
4.44.1	Detailed Description	217
4.44.2	Function Documentation	217
4.45	SoC Interrupt Router (SE)	219
4.45.1	Detailed Description	219
4.46	Mailbox Definitions	220
4.46.1	Detailed Description	220
4.46.2	Enumeration Type Documentation	220
4.47	SoC Registers (Sensor Subsystem)	221
4.47.1	Detailed Description	222
4.47.2	Enumeration Type Documentation	222
4.48	SoC Interrupts (SE)	224
4.48.1	Detailed Description	224
4.49	SoC Registers (SE)	225
4.49.1	Detailed Description	231
4.49.2	Enumeration Type Documentation	231
4.49.3	Variable Documentation	235
4.50	Quark SE Sensor Subsystem Initialisation	236
4.50.1	Detailed Description	236
4.50.2	Function Documentation	236
4.51	SS Power states	237
4.51.1	Detailed Description	237
4.51.2	Enumeration Type Documentation	237
4.51.3	Function Documentation	238
4.52	Quark SE Voltage Regulators	241
4.52.1	Detailed Description	241
4.52.2	Function Documentation	241
4.53	Syscalls	243
4.53.1	Detailed Description	243
4.53.2	Function Documentation	243
5	Data Structure Documentation	244
5.1	int_ss_i2c_reg_t Struct Reference	244
5.1.1	Detailed Description	244
5.2	int_ss_spi_reg_t Struct Reference	244
5.2.1	Detailed Description	244
5.3	mvic_reg_pad_t Struct Reference	244
5.3.1	Detailed Description	244
5.4	pic_timer_reg_pad_t Struct Reference	244

5.4.1	Detailed Description	244
5.5	qm_ac_config_t Struct Reference	244
5.5.1	Detailed Description	245
5.5.2	Field Documentation	245
5.6	qm_adc_config_t Struct Reference	246
5.6.1	Detailed Description	246
5.6.2	Field Documentation	246
5.7	qm_adc_reg_t Struct Reference	246
5.7.1	Detailed Description	247
5.8	qm_adc_xfer_t Struct Reference	247
5.8.1	Detailed Description	247
5.8.2	Field Documentation	248
5.9	qm_aonc_reg_t Struct Reference	248
5.9.1	Detailed Description	249
5.9.2	Field Documentation	249
5.10	qm_aonpt_config_t Struct Reference	249
5.10.1	Detailed Description	250
5.10.2	Field Documentation	250
5.11	qm_dma_chan_reg_t Struct Reference	250
5.11.1	Detailed Description	251
5.12	qm_dma_channel_config_t Struct Reference	252
5.12.1	Detailed Description	252
5.12.2	Field Documentation	252
5.13	qm_dma_context_t Struct Reference	252
5.13.1	Detailed Description	253
5.13.2	Field Documentation	253
5.14	qm_dma_int_reg_t Struct Reference	254
5.14.1	Detailed Description	255
5.15	qm_dma_misc_reg_t Struct Reference	255
5.15.1	Detailed Description	256
5.16	qm_dma_multi_transfer_t Struct Reference	256
5.16.1	Detailed Description	256
5.16.2	Field Documentation	257
5.17	qm_dma_transfer_t Struct Reference	257
5.17.1	Detailed Description	257
5.17.2	Field Documentation	257
5.18	qm_flash_config_t Struct Reference	258
5.18.1	Detailed Description	258
5.18.2	Field Documentation	258
5.19	qm_flash_context_t Struct Reference	259

5.19.1 Detailed Description	259
5.19.2 Field Documentation	259
5.20 qm_flash_reg_t Struct Reference	259
5.20.1 Detailed Description	260
5.20.2 Field Documentation	260
5.21 qm_fpr_config_t Struct Reference	261
5.21.1 Detailed Description	262
5.21.2 Field Documentation	262
5.22 qm_fpr_context_t Struct Reference	262
5.22.1 Detailed Description	262
5.22.2 Field Documentation	263
5.23 qm_gpio_context_t Struct Reference	263
5.23.1 Detailed Description	263
5.23.2 Field Documentation	264
5.24 qm_gpio_port_config_t Struct Reference	265
5.24.1 Detailed Description	265
5.24.2 Field Documentation	265
5.25 qm_gpio_reg_t Struct Reference	266
5.25.1 Detailed Description	267
5.25.2 Field Documentation	267
5.26 qm_i2c_config_t Struct Reference	269
5.26.1 Detailed Description	270
5.26.2 Field Documentation	270
5.27 qm_i2c_context_t Struct Reference	270
5.27.1 Detailed Description	271
5.27.2 Field Documentation	271
5.28 qm_i2c_reg_t Struct Reference	272
5.28.1 Detailed Description	274
5.28.2 Field Documentation	274
5.29 qm_i2c_transfer_t Struct Reference	279
5.29.1 Detailed Description	279
5.29.2 Field Documentation	279
5.30 qm_i2s_buffer_link Struct Reference	280
5.30.1 Detailed Description	281
5.30.2 Field Documentation	281
5.31 qm_i2s_channel_cfg_data_t Struct Reference	281
5.31.1 Detailed Description	282
5.31.2 Field Documentation	282
5.32 qm_i2s_clock_cfg_data_t Struct Reference	283
5.32.1 Detailed Description	284

5.32.2 Field Documentation	284
5.33 qm_interrupt_router_reg_t Struct Reference	284
5.33.1 Detailed Description	286
5.33.2 Field Documentation	286
5.34 qm_irq_context_t Struct Reference	291
5.34.1 Detailed Description	291
5.34.2 Field Documentation	291
5.35 qm_lapic_reg_t Struct Reference	292
5.35.1 Detailed Description	293
5.36 qm_mailbox_reg_t Struct Reference	293
5.36.1 Detailed Description	293
5.37 qm_mailbox_t Struct Reference	293
5.37.1 Detailed Description	294
5.38 qm_mbox_config_t Struct Reference	294
5.38.1 Detailed Description	294
5.38.2 Field Documentation	294
5.39 qm_mbox_msg_t Struct Reference	295
5.39.1 Detailed Description	295
5.39.2 Field Documentation	295
5.40 qm_mpr_config_t Struct Reference	295
5.40.1 Detailed Description	296
5.41 qm_mpr_context_t Struct Reference	296
5.41.1 Detailed Description	296
5.41.2 Field Documentation	296
5.42 qm_mpr_reg_t Struct Reference	296
5.42.1 Detailed Description	296
5.43 qm_mvic_reg_t Struct Reference	297
5.43.1 Detailed Description	297
5.43.2 Field Documentation	297
5.44 qm_pic_timer_config_t Struct Reference	298
5.44.1 Detailed Description	298
5.44.2 Field Documentation	298
5.45 qm_pic_timer_context_t Struct Reference	299
5.45.1 Detailed Description	299
5.45.2 Field Documentation	299
5.46 qm_pic_timer_reg_t Struct Reference	300
5.46.1 Detailed Description	300
5.47 qm_pwm_channel_t Struct Reference	300
5.47.1 Detailed Description	301
5.47.2 Field Documentation	301

5.48	qm_pwm_config_t Struct Reference	301
5.48.1	Detailed Description	302
5.48.2	Field Documentation	302
5.49	qm_pwm_context_t Struct Reference	303
5.49.1	Detailed Description	303
5.49.2	Field Documentation	303
5.50	qm_pwm_reg_t Struct Reference	303
5.50.1	Detailed Description	304
5.50.2	Field Documentation	304
5.51	qm_rtc_config_t Struct Reference	304
5.51.1	Detailed Description	304
5.51.2	Field Documentation	305
5.52	qm_rtc_reg_t Struct Reference	305
5.52.1	Detailed Description	306
5.52.2	Field Documentation	306
5.53	qm_scss_ccu_reg_t Struct Reference	307
5.53.1	Detailed Description	308
5.53.2	Field Documentation	308
5.54	qm_scss_cmp_reg_t Struct Reference	309
5.54.1	Detailed Description	310
5.54.2	Field Documentation	310
5.55	qm_scss_gp_reg_t Struct Reference	310
5.55.1	Detailed Description	311
5.55.2	Field Documentation	311
5.56	qm_scss_info_reg_t Struct Reference	312
5.56.1	Detailed Description	312
5.56.2	Field Documentation	313
5.57	qm_scss_mem_reg_t Struct Reference	313
5.57.1	Detailed Description	313
5.58	qm_scss_peripheral_reg_t Struct Reference	313
5.58.1	Detailed Description	314
5.58.2	Field Documentation	314
5.59	qm_scss_pmu_reg_t Struct Reference	314
5.59.1	Detailed Description	315
5.59.2	Field Documentation	315
5.60	qm_scss_pmux_reg_t Struct Reference	315
5.60.1	Detailed Description	316
5.60.2	Field Documentation	316
5.61	qm_scss_ss_reg_t Struct Reference	317
5.61.1	Detailed Description	317

5.62	qm_spi_async_transfer_t Struct Reference	317
5.62.1	Detailed Description	318
5.62.2	Field Documentation	318
5.63	qm_spi_context_t Struct Reference	319
5.63.1	Detailed Description	319
5.63.2	Field Documentation	319
5.64	qm_spi_reg_t Struct Reference	320
5.64.1	Detailed Description	321
5.64.2	Field Documentation	321
5.65	qm_spi_transfer_t Struct Reference	324
5.65.1	Detailed Description	324
5.65.2	Field Documentation	324
5.66	qm_ss_adc_config_t Struct Reference	325
5.66.1	Detailed Description	325
5.66.2	Field Documentation	325
5.67	qm_ss_adc_context_t Struct Reference	325
5.67.1	Detailed Description	326
5.67.2	Field Documentation	326
5.68	qm_ss_adc_xfer_t Struct Reference	326
5.68.1	Detailed Description	327
5.68.2	Field Documentation	327
5.69	qm_ss_gpio_context_t Struct Reference	328
5.69.1	Detailed Description	328
5.69.2	Field Documentation	328
5.70	qm_ss_gpio_port_config_t Struct Reference	329
5.70.1	Detailed Description	330
5.70.2	Field Documentation	330
5.71	qm_ss_i2c_config_t Struct Reference	331
5.71.1	Detailed Description	331
5.71.2	Field Documentation	331
5.72	qm_ss_i2c_context_t Struct Reference	331
5.72.1	Detailed Description	332
5.73	qm_ss_i2c_transfer_t Struct Reference	332
5.73.1	Detailed Description	332
5.73.2	Field Documentation	332
5.74	qm_ss_spi_async_transfer_t Struct Reference	333
5.74.1	Detailed Description	334
5.74.2	Field Documentation	334
5.75	qm_ss_spi_config_t Struct Reference	335
5.75.1	Detailed Description	335

5.75.2	Field Documentation	335
5.76	qm_ss_spi_context_t Struct Reference	335
5.76.1	Detailed Description	336
5.76.2	Field Documentation	336
5.77	qm_ss_spi_transfer_t Struct Reference	336
5.77.1	Detailed Description	336
5.77.2	Field Documentation	337
5.78	qm_ss_timer_config_t Struct Reference	337
5.78.1	Detailed Description	337
5.78.2	Field Documentation	338
5.79	qm_uart_config_t Struct Reference	338
5.79.1	Detailed Description	339
5.79.2	Field Documentation	339
5.80	qm_uart_context_t Struct Reference	339
5.80.1	Detailed Description	340
5.80.2	Field Documentation	340
5.81	qm_uart_reg_t Struct Reference	341
5.81.1	Detailed Description	342
5.81.2	Field Documentation	342
5.82	qm_uart_transfer_t Struct Reference	344
5.82.1	Detailed Description	344
5.82.2	Field Documentation	344
5.83	qm_usb_ep_config_t Struct Reference	345
5.83.1	Detailed Description	345
5.83.2	Field Documentation	345
5.84	qm_usb_in_ep_reg_t Struct Reference	346
5.84.1	Detailed Description	346
5.85	qm_usb_out_ep_reg_t Struct Reference	346
5.85.1	Detailed Description	346
5.86	qm_usb_reg_t Struct Reference	347
5.86.1	Detailed Description	348
5.86.2	Field Documentation	348
5.87	qm_wdt_config_t Struct Reference	351
5.87.1	Detailed Description	351
5.87.2	Field Documentation	351
5.88	qm_wdt_reg_t Struct Reference	352
5.88.1	Detailed Description	352
5.88.2	Field Documentation	353

Index355

1 Licensing information

Copyright (c) 2016, Intel Corporation All rights reserved.Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

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

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

2 Module Index

2.1 Modules

Here is a list of all modules:

Error Codes	8
Quark D2000 ADC	9
Always-on Counters	16
Analog Comparator	21
DMA	22
Flash	30
FPR	37
GPIO	42
I2C	47
I2S	58
Identification	64
Initialisation	65
Interrupt	66
ISR	68
Mailbox	78

MPR	82
PIC Timer	85
Pin Muxing setup	89
PWM / Timer	94
RTC	99
SPI	102
SS ADC	112
SS GPIO	121
SS I2C	126
SS Interrupt	132
SS ISR	134
SS SPI	140
SS Timer	147
UART	149
USB	160
Version	168
WDT	169
SOC_WATCH	172
SS Clock	176
Clock Management	181
Quark D2000 Flash Layout	195
Quark D2000 Power states	196
Quark(TM) D2000 Retention Alternator Regulator (RAR).	199
SoC Interrupt Router (D2000)	201
SoC Interrupts (D2000)	202
SoC Registers (D2000)	203
Quark SE Flash Layout	213
Quark SE SoC Power states	214
Quark SE Host Power states	217
SoC Interrupt Router (SE)	219
Mailbox Definitions	220
SoC Registers (Sensor Subsystem)	221

SoC Interrupts (SE)	224
SoC Registers (SE)	225
Quark SE Sensor Subsystem Initialisation	236
SS Power states	237
Quark SE Voltage Regulators	241
Syscalls	243

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

int_ss_i2c_reg_t SS I2C Interrupt register map	244
int_ss_spi_reg_t SS SPI Interrupt register map	244
mvic_reg_pad_t MVIC register structure	244
pic_timer_reg_pad_t PIC timer register structure	244
qm_ac_config_t Analog Comparator configuration type	244
qm_adc_config_t ADC configuration type	246
qm_adc_reg_t ADC register map	246
qm_adc_xfer_t ADC transfer type	247
qm_aonc_reg_t Always-on Counter Controller register map	248
qm_aonpt_config_t QM Always-on Periodic Timer configuration type	249
qm_dma_chan_reg_t DMA channel register map	250
qm_dma_channel_config_t DMA channel configuration structure	252
qm_dma_context_t DMA context type	252
qm_dma_int_reg_t DMA interrupt register map	254

qm_dma_misc_reg_t	DMA miscellaneous register map	255
qm_dma_multi_transfer_t	DMA multiblock transfer configuration structure	256
qm_dma_transfer_t	DMA single block transfer configuration structure	257
qm_flash_config_t	Flash configuration structure	258
qm_flash_context_t	Flash context type	259
qm_flash_reg_t	Flash register map	259
qm_fpr_config_t	Flash Protection Region configuration structure	261
qm_fpr_context_t	FPR context type	262
qm_gpio_context_t	GPIO context type	263
qm_gpio_port_config_t	GPIO port configuration type	265
qm_gpio_reg_t	GPIO register map	266
qm_i2c_config_t	I2C configuration type	269
qm_i2c_context_t	I2C context to be saved between sleep/resume	270
qm_i2c_reg_t	I2C register map	272
qm_i2c_transfer_t	I2C transfer type	279
qm_i2s_buffer_link	Ring buffer link definition	280
qm_i2s_channel_cfg_data_t	I2S controller configuration	281
qm_i2s_clock_cfg_data_t	I2S Clock Configuration	283
qm_interrupt_router_reg_t	Interrupt register map	284
qm_irq_context_t	SS IRQ context type	291
qm_lapic_reg_t	APIC register block type	292

qm_mailbox_reg_t	
Mailbox register map	293
qm_mailbox_t	
Mailbox register structure	293
qm_mbox_config_t	
Mailbox Configuration Structure	294
qm_mbox_msg_t	
Definition of the mailbox message	295
qm_mpr_config_t	
SRAM Memory Protection Region configuration type	295
qm_mpr_context_t	
MPR context type	296
qm_mpr_reg_t	
Memory Protection Region register map	296
qm_mvic_reg_t	
MVIC register map	297
qm_pic_timer_config_t	
PIC timer configuration type	298
qm_pic_timer_context_t	
PIC TIMER context type	299
qm_pic_timer_reg_t	
PIC timer register map	300
qm_pwm_channel_t	
PWM / Timer channel register map	300
qm_pwm_config_t	
QM PWM / Timer configuration type	301
qm_pwm_context_t	
PWM context type	303
qm_pwm_reg_t	
PWM / Timer register map	303
qm_rtc_config_t	
QM RTC configuration type	304
qm_rtc_reg_t	
RTC register map	305
qm_scss_ccu_reg_t	
System Core register map	307
qm_scss_cmp_reg_t	
Comparator register map	309
qm_scss_gp_reg_t	
General Purpose register map	310
qm_scss_info_reg_t	
Information register map	312

qm_scss_mem_reg_t	Memory Control register map	313
qm_scss_peripheral_reg_t	Peripheral Registers register map	313
qm_scss_pmu_reg_t	Power Management register map	314
qm_scss_pmux_reg_t	Pin MUX register map	315
qm_scss_ss_reg_t	Sensor Subsystem register map	317
qm_spi_async_transfer_t	SPI asynchronous transfer type	317
qm_spi_context_t	SPI context type	319
qm_spi_reg_t	SPI register map	320
qm_spi_transfer_t	SPI synchronous transfer type	324
qm_ss_adc_config_t	SS ADC configuration type	325
qm_ss_adc_context_t	SS ADC context type	325
qm_ss_adc_xfer_t	SS ADC transfer type	326
qm_ss_gpio_context_t	SS GPIO context type	328
qm_ss_gpio_port_config_t	SS GPIO port configuration type	329
qm_ss_i2c_config_t	QM SS I2C configuration type	331
qm_ss_i2c_context_t	SS I2C context type	331
qm_ss_i2c_transfer_t	QM SS I2C transfer type	332
qm_ss_spi_async_transfer_t	SPI asynchronous transfer type	333
qm_ss_spi_config_t	SPI configuration type	335
qm_ss_spi_context_t	Sensor Subsystem SPI context type	335
qm_ss_spi_transfer_t	SPI synchronous transfer type	336

qm_ss_timer_config_t	Sensor Subsystem Timer Configuration Type	337
qm_uart_config_t	UART configuration structure type	338
qm_uart_context_t	UART context to be saved between sleep/resume	339
qm_uart_reg_t	UART register map	341
qm_uart_transfer_t	UART asynchronous transfer structure	344
qm_usb_ep_config_t	USB Endpoint Configuration	345
qm_usb_in_ep_reg_t	USB register map	346
qm_usb_out_ep_reg_t	OUT Endpoint Registers	346
qm_usb_reg_t	USB Register block type	347
qm_wdt_config_t	QM WDT configuration type	351
qm_wdt_reg_t	Watchdog timer register map	352

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 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.

4.2.1 Detailed Description

Analog to Digital Converter (ADC).

4.2.2 Enumeration Type Documentation

4.2.2.1 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](#).

4.2.2.2 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.

4.2.2.3 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.

4.2.2.4 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.

4.2.2.5 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.

4.2.3 Function Documentation

4.2.3.1 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

<i>in</i>	<i>adc</i>	Which ADC to calibrate.
-----------	------------	-------------------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 204 of file `qm_adc.c`.

4.2.3.2 `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

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

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 345 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`.

4.2.3.3 `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

<i>in</i>	<i>adc</i>	Which ADC to get calibration for.
<i>out</i>	<i>cal</i>	Calibration data. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 261 of file `qm_adc.c`.

4.2.3.4 `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	<i>adc</i>	Which ADC to calibrate.
in	<i>callback</i>	Callback called on completion.
in	<i>callback_data</i>	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 220 of file qm_adc.c.

4.2.3.5 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	<i>adc</i>	Which ADC to read.
in, out	<i>xfer</i>	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 391 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.

4.2.3.6 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	<i>adc</i>	Which ADC to enable.
in	<i>mode</i>	ADC operating mode.
in	<i>callback</i>	Callback called on completion.

<i>in</i>	<i>callback_data</i>	The callback user data.
-----------	----------------------	-------------------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 302 of file qm_adc.c.

References QM_ADC_MODE_NORM_CAL, and QM_ADC_MODE_NORM_NO_CAL.

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

Set ADC calibration data.

Parameters

<i>in</i>	<i>adc</i>	Which ADC to set calibration for.
<i>in</i>	<i>cal</i>	Calibration data.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 242 of file qm_adc.c.

4.2.3.8 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

<i>in</i>	<i>adc</i>	Which ADC to configure.
<i>in</i>	<i>cfg</i>	ADC configuration. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 329 of file qm_adc.c.

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

4.2.3.9 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	<i>adc</i>	Which ADC to enable.
in	<i>mode</i>	ADC operating mode.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 271 of file qm_adc.c.

References QM_ADC_MODE_NORM_CAL, and QM_ADC_MODE_NORM_NO_CAL.

Referenced by qm_power_soc_deep_sleep(), and qm_power_soc_sleep().

4.3 Always-on Counters

Always-on Counters.

Data Structures

- struct [qm_aonpt_config_t](#)
QM Always-on Periodic Timer configuration type.

Enumerations

- enum [qm_aonpt_status_t](#) { [QM_AONPT_READY](#) = 0, [QM_AONPT_EXPIRED](#) }
Always on counter status.

Functions

- int [qm_aonc_enable](#) (const [qm_aonc_t](#) aonc)
Enable the Always-on Counter.
- int [qm_aonc_disable](#) (const [qm_aonc_t](#) aonc)
Disable the Always-on Counter.
- int [qm_aonc_get_value](#) (const [qm_aonc_t](#) aonc, uint32_t *const val)
Get the current value of the Always-on Counter.
- int [qm_aonpt_set_config](#) (const [qm_aonc_t](#) aonc, const [qm_aonpt_config_t](#) *const cfg)
Set the Always-on Periodic Timer configuration.
- int [qm_aonpt_get_value](#) (const [qm_aonc_t](#) aonc, uint32_t *const val)
Get the current value of the Always-on Periodic Timer.
- int [qm_aonpt_get_status](#) (const [qm_aonc_t](#) aonc, [qm_aonpt_status_t](#) *const status)
Get the current status of an Always-on Periodic Timer.
- int [qm_aonpt_clear](#) (const [qm_aonc_t](#) aonc)
Clear the status of the Always-on Periodic Timer.
- int [qm_aonpt_reset](#) (const [qm_aonc_t](#) aonc)
Reset the Always-on Periodic Timer back to the configured value.
- int [qm_aonpt_save_context](#) (const [qm_aonc_t](#) aonc, [qm_aonc_context_t](#) *const ctx)
Save the Always-on Periodic Timer context.
- int [qm_aonpt_restore_context](#) (const [qm_aonc_t](#) aonc, const [qm_aonc_context_t](#) *const ctx)
Restore the Always-on Periodic Timer context.

4.3.1 Detailed Description

Always-on Counters.

Note

The always on counters are in the 32kHz clock domain. Some register operations take a minimum of a 32kHz clock cycle to complete. If the Always on timer interrupt is not configured to be edge triggered, multiple interrupts will occur.

4.3.2 Enumeration Type Documentation

4.3.2.1 enum `qm_aonpt_status_t`

Always on counter status.

Enumerator

`QM_AONPT_READY` Default Timer Status.

`QM_AONPT_EXPIRED` Timer expired. Status must be cleared with [qm_aonpt_clear\(\)](#).

Definition at line 26 of file `qm_aon_counters.h`.

4.3.3 Function Documentation

4.3.3.1 `int qm_aonc_disable (const qm_aonc_t aonc)`

Disable the Always-on Counter.

Parameters

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

Returns

Standard `errno` return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 140 of file `qm_aon_counters.c`.

4.3.3.2 `int qm_aonc_enable (const qm_aonc_t aonc)`

Enable the Always-on Counter.

Parameters

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

Returns

Standard `errno` return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 130 of file `qm_aon_counters.c`.

4.3.3.3 `int qm_aonc_get_value (const qm_aonc_t aonc, uint32_t *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	<i>aonc</i>	Always-on counter to read.
out	<i>val</i>	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 150 of file qm_aon_counters.c.

4.3.3.4 int qm_aonpt_clear (const qm_aonc_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	<i>aonc</i>	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 210 of file qm_aon_counters.c.

4.3.3.5 int qm_aonpt_get_status (const qm_aonc_t *aonc*, qm_aonpt_status_t *const *status*)

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

Parameters

in	<i>aonc</i>	Always-on counter to read.
out	<i>status</i>	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 195 of file qm_aon_counters.c.

References QM_AONPT_EXPIRED, and QM_AONPT_READY.

4.3.3.6 int qm_aonpt_get_value (const qm_aonc_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	<i>aonc</i>	Always-on counter to read.
out	<i>val</i>	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 184 of file `qm_aon_counters.c`.

4.3.3.7 `int qm_aonpt_reset (const qm_aonc_t aonc)`

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

Parameters

in	<i>aonc</i>	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 224 of file `qm_aon_counters.c`.

4.3.3.8 `int qm_aonpt_restore_context (const qm_aonc_t aonc, const qm_aonc_context_t *const ctx)`

Restore the Always-on Periodic Timer context.

Restore the configuration of the specified AONC peripheral after exiting sleep.

Parameters

in	<i>aonc</i>	AONC index.
in	<i>ctx</i>	AONC context structure. 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 246 of file `qm_aon_counters.c`.

4.3.3.9 `int qm_aonpt_save_context (const qm_aonc_t aonc, qm_aonc_context_t *const ctx)`

Save the Always-on Periodic Timer context.

Save the configuration of the specified AONC peripheral before entering sleep.

Parameters

in	<i>aonc</i>	AONC index.
out	<i>ctx</i>	AONC context structure. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 235 of file qm_aon_counters.c.

4.3.3.10 `int qm_aonpt_set_config (const qm_aonc_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 expires and an interrupt is triggered. The Periodic Timer is disabled if the counter is set to 0.

Parameters

in	<i>aonc</i>	Always-on counter to read.
in	<i>cfg</i>	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.
<i>Negative</i>	errno for possible error codes.

Definition at line 160 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`.

4.4 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.

4.4.1 Detailed Description

Analog Comparator.

4.4.2 Function Documentation

4.4.2.1 int qm_ac_set_config (const qm_ac_config_t *const config)

Set Analog Comparator configuration.

Parameters

<i>in</i>	<i>config</i>	Analog Comparator configuration. This must not be NULL.
-----------	---------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 36 of file qm_comparator.c.

References [qm_ac_config_t::callback](#), [qm_ac_config_t::callback_data](#), [qm_ac_config_t::cmp_en](#), [qm_ac_config_t::polarity](#), [qm_ac_config_t::power](#), and [qm_ac_config_t::reference](#).

4.5 DMA

DMA Driver for Quark Microcontrollers.

Data Structures

- struct [qm_dma_channel_config_t](#)
DMA channel configuration structure.
- struct [qm_dma_transfer_t](#)
DMA single block transfer configuration structure.
- struct [qm_dma_multi_transfer_t](#)
DMA multiblock 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.*
- enum [qm_dma_transfer_type_t](#) { [QM_DMA_TYPE_SINGLE](#), [QM_DMA_TYPE_MULTI_CONT](#), [QM_DMA_TYPE_MULTI_LL](#), [QM_DMA_TYPE_MULTI_LL_CIRCULAR](#) }

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 single block transfer.
- int [qm_dma_multi_transfer_set_config](#) (const [qm_dma_t](#) dma, const [qm_dma_channel_id_t](#) channel_id, [qm_dma_multi_transfer_t](#) *const multi_transfer_config)
Setup a DMA multiblock 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.
- int `qm_dma_save_context` (const `qm_dma_t` dma, `qm_dma_context_t` *const ctx)
Save DMA peripheral's context.
- int `qm_dma_restore_context` (const `qm_dma_t` dma, const `qm_dma_context_t` *const ctx)
Restore DMA peripheral's context.

4.5.1 Detailed Description

DMA Driver for Quark Microcontrollers.

4.5.2 Enumeration Type Documentation

4.5.2.1 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`.

4.5.2.2 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`.

4.5.2.3 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`.

4.5.2.4 enum qm_dma_transfer_type_t

Enumerator

QM_DMA_TYPE_SINGLE Single block mode.

QM_DMA_TYPE_MULTI_CONT Contiguous multiblock mode.

QM_DMA_TYPE_MULTI_LL Link list multiblock mode.

QM_DMA_TYPE_MULTI_LL_CIRCULAR Link list multiblock mode with cyclic operation.

Definition at line 66 of file qm_dma.h.

4.5.2.5 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.

4.5.3 Function Documentation

4.5.3.1 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, transfer type 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 or its transfer type is changed.

Parameters

in	<i>dma</i>	DMA instance.
in	<i>channel_id</i>	The channel to start.
in	<i>channel_config</i>	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 254 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_TYPE_MULTI_LL_CIRCULAR](#), [qm_dma_channel_config_t::source_burst_length](#), [qm_dma_channel_config_t::source_transfer_width](#), and [qm_dma_channel_config_t::transfer_type](#).

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

4.5.3.2 `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

<code>in</code>	<code>dma</code>	DMA instance.
-----------------	------------------	---------------

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 208 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`, `clk_dma_enable()`, `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`.

4.5.3.3 `int qm_dma_multi_transfer_set_config (const qm_dma_t dma, const qm_dma_channel_id_t channel_id, qm_dma_multi_transfer_t *const multi_transfer_config)`

Setup a DMA multiblock transfer.

If the DMA channel has been configured for contiguous multiblock transfers, this function sets the source address, destination address, block size and number of block parameters needed to perform a contiguous multiblock transfer. The `linked_list_first` parameter in the transfer struct is ignored.

If the DMA channel has been configured for linked-list multiblock transfers, this function populates a linked list in the client memory area pointed at by the `linked_list_first` parameter in the transfer struct, using the provided parameters source address, destination address, block size and number of blocks (equal to the number of LLIs). This function may be called repeatedly in order to add different client buffers for transmission/reception that are not contiguous in memory (scatter-gather) in a buffer chain fashion. When calling `qm_dma_transfer_start`, the DMA core sees a single linked list built using consecutive calls to this function. Furthermore, if the transfer type is linked-list cyclic, the `linked_list_address` parameter of the last LLI points at the first LLI. The client needs to allocate enough memory starting at `linked_list_first` for the whole set of LLIs to fit, i.e. `(num_blocks * sizeof(qm_dma_linked_list_item_t))`.

The DMA driver manages the block interrupts so that only when the last block of a buffer has been transferred, the client callback is invoked. Note that in linked-list mode, each buffer transfer results on a client callback, and all buffers need to contain the same number of blocks.

`qm_dma_multi_transfer_set_config()` must be called before starting every transfer, even if the addresses, block size and other configuration information remain unchanged.

Parameters

<code>in</code>	<code>dma</code>	DMA instance.
-----------------	------------------	---------------

in	<i>channel_id</i>	The channel to start.
in	<i>multi_transfer_ - config</i>	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 437 of file qm_dma.c.

References qm_dma_multi_transfer_t::block_size, qm_dma_chan_reg_t::ctrl_low, qm_dma_multi_transfer_t::destination_address, qm_dma_multi_transfer_t::linked_list_first, qm_dma_chan_reg_t::llp_low, qm_dma_multi_transfer_t::num_blocks, QM_DMA_CHANNEL_NUM, QM_DMA_NUM, QM_DMA_TYPE_MULTI_CONT, QM_DMA_TYPE_MULTI_LL, QM_DMA_TYPE_MULTI_LL_CIRCULAR, and qm_dma_multi_transfer_t::source_address.

4.5.3.4 int qm_dma_restore_context (const qm_dma_t dma, const qm_dma_context_t *const ctx)

Restore DMA peripheral's context.

Restore the configuration of the specified DMA peripheral after exiting sleep.

Parameters

in	<i>dma</i>	DMA device.
in	<i>ctx</i>	DMA context structure. 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 705 of file qm_dma.c.

References qm_dma_context_t::cfg_high, qm_dma_misc_reg_t::cfg_low, qm_dma_context_t::cfg_low, qm_dma_chan_reg_t::ctrl_low, qm_dma_context_t::ctrl_low, qm_dma_context_t::llp_low, qm_dma_context_t::misc_cfg_low, QM_DMA_CHANNEL_NUM, and QM_DMA_NUM.

4.5.3.5 int qm_dma_save_context (const qm_dma_t dma, qm_dma_context_t *const ctx)

Save DMA peripheral's context.

Saves the configuration of the specified DMA peripheral before entering sleep.

Parameters

in	<i>dma</i>	DMA device.
out	<i>ctx</i>	DMA context structure. 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 681 of file qm_dma.c.

References [qm_dma_context_t::cfg_high](#), [qm_dma_context_t::cfg_low](#), [qm_dma_context_t::ctrl_low](#), [qm_dma_context_t::llp_low](#), [qm_dma_context_t::misc_cfg_low](#), [QM_DMA_CHANNEL_NUM](#), and [QM_DMA_NUM](#).

4.5.3.6 `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.

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	<i>dma</i>	DMA instance.
in	<i>channel_id</i>	The channel to start.
in	<i>transfer_config</i>	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 656 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](#).

4.5.3.7 `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 single block 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	<i>dma</i>	DMA instance.
in	<i>channel_id</i>	The channel to start.
in	<i>transfer_config</i>	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 345 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()`.

4.5.3.8 `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\(\)](#) must first be called before starting a transfer.

Parameters

in	<i>dma</i>	DMA instance.
in	<i>channel_id</i>	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 582 of file qm_dma.c.

References `qm_dma_int_reg_t::clear_block_low`, `qm_dma_int_reg_t::clear_err_low`, `qm_dma_int_reg_t::clear_tfr_low`, `qm_dma_int_reg_t::mask_block_low`, `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()`.

4.5.3.9 `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 required 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	<i>dma</i>	DMA instance.
in	<i>channel_id</i>	The channel to stop.

Returns

Standard `errno` return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 618 of file qm_dma.c.

References `qm_dma_chan_reg_t::llp_low`, `qm_dma_int_reg_t::mask_block_low`, `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()`.

4.6 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 }
 - enum [qm_flash_disable_t](#) { QM_FLASH_WRITE_ENABLE = 0, QM_FLASH_WRITE_DISABLE }
- Flash region enum.*
- 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 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.
- int [qm_flash_save_context](#) (const [qm_flash_t](#) flash, [qm_flash_context_t](#) *const ctx)
Save flash context.
- int [qm_flash_restore_context](#) (const [qm_flash_t](#) flash, const [qm_flash_context_t](#) *const ctx)
Restore flash context.

4.6.1 Detailed Description

Flash controller.

4.6.2 Enumeration Type Documentation

4.6.2.1 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 33 of file qm_flash.h.

4.6.2.2 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 21 of file qm_flash.h.

4.6.3 Function Documentation

4.6.3.1 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. 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 314 of file qm_flash.c.

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

4.6.3.2 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.

Parameters

in	flash	Flash controller index.
----	-------	-------------------------

in	<i>region</i>	Flash region to address.
in	<i>page_num</i>	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 271 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`.

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

Write multiple of 4 bytes of data to Flash.

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	<i>flash</i>	Flash controller index.
in	<i>reg</i>	Which Flash region to address.
in	<i>f_addr</i>	Address within Flash physical address space.
in	<i>page_buf</i>	Page buffer to store page during update. Must be at least <code>QM_FLASH_PAGE_SIZE</code> words big and must not be NULL.
in	<i>data</i>	Data to write (array of words). This must not be NULL.
in	<i>len</i>	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 168 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`.

4.6.3.4 `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 flash page.

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	<i>flash</i>	Flash controller index.
in	<i>region</i>	Which Flash region to address.
in	<i>page_num</i>	Which page of flash to overwrite.
in	<i>data</i>	Data to write (array of words). This must not be NULL.
in	<i>len</i>	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 100 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.

4.6.3.5 int qm_flash_restore_context (const qm_flash_t flash, const qm_flash_context_t *const ctx)

Restore flash context.

Restore the configuration of the specified flash controller. If the system clock frequency is lowered, the flash timings need to be restored. Otherwise, reading from flash will not be optimal (there will be 2 wait states instead of 0 wait states.)

Parameters

in	<i>flash</i>	Flash controller index.
in	<i>ctx</i>	Flash context structure. 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 350 of file qm_flash.c.

References qm_flash_reg_t::ctrl, qm_flash_context_t::ctrl, qm_flash_reg_t::tmg_ctrl, and qm_flash_context_t::tmg_ctrl.

4.6.3.6 int qm_flash_save_context (const qm_flash_t flash, qm_flash_context_t *const ctx)

Save flash context.

Save the configuration of the specified flash controller.

Parameters

in	<i>flash</i>	Flash controller index.
----	--------------	-------------------------

out	ctx	Flash context structure. 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 337 of file qm_flash.c.

References qm_flash_reg_t::ctrl, qm_flash_context_t::ctrl, qm_flash_reg_t::tmg_ctrl, and qm_flash_context_t::tmg_ctrl.

4.6.3.7 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 22 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.

4.6.3.8 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.

Note

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

Parameters

in	<i>flash</i>	Flash controller index.
in	<i>region</i>	Flash region to address.
in	<i>f_addr</i>	Address within Flash physical address space.
in	<i>data</i>	Data word to write.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 43 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 FPR

Flash Protection Region control.

Data Structures

- struct [qm_fpr_config_t](#)
Flash Protection Region configuration structure.

Enumerations

- 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 violation 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.
- int [qm_fpr_save_context](#) (const [qm_flash_t](#) flash, [qm_fpr_context_t](#) *const ctx)
Save FPR context.
- int [qm_fpr_restore_context](#) (const [qm_flash_t](#) flash, const [qm_fpr_context_t](#) *const ctx)
Restore FPR context.

4.7.1 Detailed Description

Flash Protection Region control.

4.7.2 Enumeration Type Documentation

4.7.2.1 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 41 of file [qm_fpr.h](#).

4.7.2.2 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 22 of file qm_fpr.h.

4.7.2.3 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 52 of file qm_fpr.h.

4.7.2.4 enum qm_fpr_viol_mode_t

FPR violation 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 32 of file qm_fpr.h.

4.7.3 Function Documentation

4.7.3.1 int qm_fpr_restore_context (const qm_flash_t flash, const qm_fpr_context_t *const ctx)

Restore FPR context.

Restore the configuration of the specified FPR peripheral after exiting sleep. The Flash peripheral linked to the FPR restored needs to be restored as well by calling [qm_flash_restore_context\(\)](#).

FPR configuration is lost after sleep and can therefore be modified even if this configuration was locked before sleep. To support persistent configuration, the configuration must be restored when resuming as part of the bootloader.

Parameters

in	flash	Flash index.
in	ctx	FPR context structure. 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 197 of file `qm_fpr.c`.

References `qm_flash_reg_t::fpr_rd_cfg`, and `qm_fpr_context_t::fpr_rd_cfg`.

4.7.3.2 `int qm_fpr_save_context (const qm_flash_t flash, qm_fpr_context_t *const ctx)`

Save FPR context.

Save the configuration of the specified FPR peripheral before entering sleep. The Flash peripheral linked to the FPR saved needs to be saved as well by calling [qm_flash_save_context\(\)](#).

FPR configuration is lost after sleep and can therefore be modified even if this configuration was locked before sleep. To support persistent configuration, the configuration must be restored when resuming as part of the bootloader.

Parameters

in	<i>flash</i>	Flash index.
out	<i>ctx</i>	FPR context structure. 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 182 of file `qm_fpr.c`.

References `qm_flash_reg_t::fpr_rd_cfg`, and `qm_fpr_context_t::fpr_rd_cfg`.

4.7.3.3 `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	<i>flash</i>	Which Flash controller to configure.
in	<i>id</i>	FPR identifier.
in	<i>cfg</i>	FPR configuration.
in	<i>region</i>	The region of Flash to be configured.

Returns

Standard `errno` return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 34 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`.

4.7.3.4 `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	<i>mode</i>	(generate interrupt, warm reset, enter probe mode).
in	<i>flash</i>	controller.
in	<i>fpr_cb</i>	for interrupt mode (only).
in	<i>data</i>	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 70 of file qm_fpr.c.

References FPR_VIOL_MODE_INTERRUPT, and FPR_VIOL_MODE_PROBE.

4.8 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` = 0, `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.
- int `qm_gpio_save_context` (const `qm_gpio_t` gpio, `qm_gpio_context_t` *const ctx)
Save GPIO context.
- int `qm_gpio_restore_context` (const `qm_gpio_t` gpio, const `qm_gpio_context_t` *const ctx)
Restore GPIO context.

4.8.1 Detailed Description

General Purpose IO.

4.8.2 Enumeration Type Documentation

4.8.2.1 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`.

4.8.3 Function Documentation

4.8.3.1 `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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	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 107 of file qm_gpio.c.

4.8.3.2 `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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	Pin of GPIO port to read.
out	<i>state</i>	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 85 of file qm_gpio.c.

4.8.3.3 `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	<i>gpio</i>	GPIO port index.
out	<i>port</i>	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 131 of file qm_gpio.c.

4.8.3.4 int qm_gpio_restore_context (const qm_gpio_t gpio, const qm_gpio_context_t *const ctx)

Restore GPIO context.

Restore the configuration of the specified GPIO peripheral after exiting sleep.

Parameters

in	gpio	GPIO port index.
in	ctx	GPIO context structure. 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 174 of file qm_gpio.c.

References qm_gpio_reg_t::gpio_debounce, qm_gpio_context_t::gpio_debounce, qm_gpio_reg_t::gpio_int_bothedge, qm_gpio_context_t::gpio_int_bothedge, qm_gpio_reg_t::gpio_int_polarity, qm_gpio_context_t::gpio_int_polarity, qm_gpio_reg_t::gpio_inten, qm_gpio_context_t::gpio_inten, qm_gpio_reg_t::gpio_intmask, qm_gpio_context_t::gpio_intmask, qm_gpio_reg_t::gpio_inttype_level, qm_gpio_context_t::gpio_inttype_level, qm_gpio_reg_t::gpio_ls_sync, qm_gpio_context_t::gpio_ls_sync, qm_gpio_reg_t::gpio_swporta_ctl, qm_gpio_context_t::gpio_swporta_ctl, qm_gpio_reg_t::gpio_swporta_ddr, qm_gpio_context_t::gpio_swporta_ddr, qm_gpio_reg_t::gpio_swporta_dr, and qm_gpio_context_t::gpio_swporta_dr.

4.8.3.5 int qm_gpio_save_context (const qm_gpio_t gpio, qm_gpio_context_t *const ctx)

Save GPIO context.

Save the configuration of the specified GPIO peripheral before entering sleep.

Parameters

in	gpio	GPIO port index.
out	ctx	GPIO context structure. 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_gpio.c.

References qm_gpio_reg_t::gpio_debounce, qm_gpio_context_t::gpio_debounce, qm_gpio_reg_t::gpio_int_bothedge, qm_gpio_context_t::gpio_int_bothedge, qm_gpio_reg_t::gpio_int_polarity, qm_gpio_context_t::gpio_int_polarity, qm_gpio_reg_t::gpio_inten, qm_gpio_context_t::gpio_inten, qm_gpio_reg_t::gpio_intmask, qm_gpio_context_t::gpio_intmask, qm_gpio_reg_t::gpio_inttype_level, qm_gpio_context_t::gpio_inttype_level, qm_gpio_reg_t::gpio_ls_sync, qm_gpio_context_t::gpio_ls_sync, qm_gpio_reg_t::gpio_swporta_ctl, qm_gpio_context_t::gpio_swporta_ctl, qm_gpio_reg_t::gpio_swporta_ddr, qm_gpio_context_t::gpio_swporta_ddr, qm_gpio_reg_t::gpio_swporta_dr, and qm_gpio_context_t::gpio_swporta_dr.

4.8.3.6 `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	<i>gpio</i>	GPIO port index to configure.
in	<i>cfg</i>	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 59 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_ls_sync`, `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`.

4.8.3.7 `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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	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 97 of file `qm_gpio.c`.

4.8.3.8 `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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	Pin of GPIO port to update.
in	<i>state</i>	QM_GPIO_LOW for low or QM_GPIO_HIGH for high.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 117 of file qm_gpio.c.

References QM_GPIO_STATE_NUM.

4.8.3.9 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

<i>in</i>	<i>gpio</i>	GPIO port index.
<i>in</i>	<i>val</i>	Value of all pins on GPIO port.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 141 of file qm_gpio.c.

4.9 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` = 0, `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), `QM_I2C_TX_ABORT` = BIT(18), `QM_I2C_TX_OVER` = BIT(19),
`QM_I2C_RX_OVER` = BIT(20), `QM_I2C_RX_UNDER` = BIT(21), `QM_I2C_START_DETECTED` = BIT(22),
`QM_I2C_TX_EMPTY` = BIT(23),
`QM_I2C_RX_FULL` = BIT(24) , `QM_I2C_GEN_CALL_DETECTED` = BIT(26) }
- I2C status type.*
- enum `qm_i2c_slave_stop_t` { `QM_I2C_SLAVE_INTERRUPT_ALWAYS` = 0x0, `QM_I2C_SLAVE_INTERRUPT_WHEN_ADDRESSED` = 0x1 }
- QM I2C slave stop detect behaviour.*

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 bus 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_slave_irq_transfer](#) (const [qm_i2c_t](#) i2c, volatile const [qm_i2c_transfer_t](#) *const xfer)
- Interrupt based slave transfer on I2C.*
- int [qm_i2c_slave_irq_transfer_update](#) (const [qm_i2c_t](#) i2c, volatile const [qm_i2c_transfer_t](#) *const xfer)
- I2C interrupt based slave transfer buffer update.*
- 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 master transfer on the I2C bus.*
- int [qm_i2c_slave_dma_transfer](#) (const [qm_i2c_t](#) i2c, const [qm_i2c_transfer_t](#) *const xfer)
- Perform a DMA based slave 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.*
- int [qm_i2c_save_context](#) (const [qm_i2c_t](#) i2c, [qm_i2c_context_t](#) *const ctx)
- Save I2C context.*
- int [qm_i2c_restore_context](#) (const [qm_i2c_t](#) i2c, const [qm_i2c_context_t](#) *const ctx)
- Restore I2C context.*

4.9.1 Detailed Description

I2C.

4.9.2 Enumeration Type Documentation

4.9.2.1 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](#).

4.9.2.2 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](#).

4.9.2.3 enum qm_i2c_slave_stop_t

QM I2C slave stop detect behaviour.

Enumerator

QM_I2C_SLAVE_INTERRUPT_ALWAYS Interrupt regardless of whether this slave is addressed or not.

QM_I2C_SLAVE_INTERRUPT_WHEN_ADDRESSED Trigger interrupt only if this slave is being addressed.

Definition at line 100 of file qm_i2c.h.

4.9.2.4 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.

4.9.2.5 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_ABORT User abort.

QM_I2C_BUSY Controller busy.

QM_I2C_TX_ABORT Tx abort.

QM_I2C_TX_OVER Tx overflow.

QM_I2C_RX_OVER Rx overflow.

QM_I2C_RX_UNDER Rx underflow.

QM_I2C_START_DETECTED Start or restart detected.

QM_I2C_TX_EMPTY TX buffer empty.

QM_I2C_RX_FULL RX buffer full.

QM_I2C_GEN_CALL_DETECTED Stop detected. General call detected.

Definition at line 61 of file qm_i2c.h.

4.9.3 Function Documentation

4.9.3.1 `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	<i>i2c</i>	I2C controller identifier.
in	<i>dma_controller_id</i>	DMA controller identifier.
in	<i>channel_id</i>	DMA channel identifier.
in	<i>direction</i>	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 1490 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_4`, `QM_DMA_BURST_TRANS_LENGTH_8`, `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_TYPE_SINGLE`, `qm_dma_channel_config_t::source_burst_length`, `qm_dma_channel_config_t::source_transfer_width`, and `qm_dma_channel_config_t::transfer_type`.

4.9.3.2 `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	<i>i2c</i>	Which I2C to terminate transfers from.
----	------------	--

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 1235 of file qm_i2c.c.

References qm_dma_transfer_terminate().

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

Retrieve I2C bus status.

Parameters

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

The user may call this function before performing an I2C transfer in order to guarantee that the I2C interface is available.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 903 of file qm_i2c.c.

References qm_i2c_reg_t::ic_status, qm_i2c_reg_t::ic_tx_abrt_source, QM_I2C_BUSY, and QM_I2C_IDLE.

Referenced by qm_i2c_master_read(), and qm_i2c_master_write().

4.9.3.4 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 controller identifier.
----	-----	----------------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 1217 of file qm_i2c.c.

4.9.3.5 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 master transfer on the I2C bus.

Perform a DMA based master 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 writing 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	<i>i2c</i>	I2C controller identifier.
in	<i>xfer</i>	Structure containing pre-allocated write and read data buffers and callback functions. This must not be NULL and must be kept valid until the transfer is complete.
in	<i>slave_addr</i>	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 1542 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`.

4.9.3.6 `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	<i>i2c</i>	Which I2C to transfer from.
in	<i>xfer</i>	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	<i>slave_addr</i>	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 1073 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`.

4.9.3.7 `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	<i>i2c</i>	Which I2C to read from.
in	<i>slave_addr</i>	Address of slave device to read from.
out	<i>data</i>	Pre-allocated buffer to populate with data. This must not be NULL.
in	<i>len</i>	Length of data to read from slave.
in	<i>stop</i>	Generate a STOP condition at the end of rx.
out	<i>status</i>	Get I2C status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 999 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().

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 924 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().

4.9.3.9 int qm_i2c_restore_context (const qm_i2c_t *i2c*, const qm_i2c_context_t *const *ctx*)

Restore I2C context.

Restore the configuration of the specified I2C peripheral after exiting sleep.

Parameters

in	<i>i2c</i>	I2C port index.
in	<i>ctx</i>	I2C context structure. 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 1690 of file qm_i2c.c.

References qm_i2c_context_t::con, qm_i2c_context_t::enable, qm_i2c_context_t::fs_scl_hcnt, qm_i2c_context_t::fs_scl_lcmt, qm_i2c_context_t::fs_spklen, qm_i2c_reg_t::ic_con, qm_i2c_reg_t::ic_enable, qm_i2c_reg_t::ic_fs_scl_hcnt, qm_i2c_reg_t::ic_fs_scl_lcmt, qm_i2c_reg_t::ic_fs_spklen, qm_i2c_reg_t::ic_intr_mask, qm_i2c_context_t::ic_intr_mask, qm_i2c_reg_t::ic_rx_tl, qm_i2c_reg_t::ic_sar, qm_i2c_reg_t::ic_ss_scl_hcnt, qm_i2c_reg_t::ic_ss_scl_lcmt, qm_i2c_reg_t::ic_tx_tl, qm_i2c_context_t::sar, qm_i2c_context_t::ss_scl_hcnt, qm_i2c_context_t::ss_scl_lcmt, and qm_i2c_context_t::tx_tl.

4.9.3.10 int qm_i2c_save_context (const qm_i2c_t *i2c*, qm_i2c_context_t *const *ctx*)

Save I2C context.

Saves the configuration of the specified I2C peripheral before entering sleep. The slave operations need to be disabled before being able to save the context as otherwise we could be interrupted by an I2C transfer while saving registers.

Parameters

in	<i>i2c</i>	I2C port index.
out	<i>ctx</i>	I2C context structure. 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 1668 of file qm_i2c.c.

References qm_i2c_context_t::con, qm_i2c_context_t::enable, qm_i2c_context_t::fs_scl_hcnt, qm_i2c_context_t::fs_scl_lcmt, qm_i2c_context_t::fs_spklen, qm_i2c_reg_t::ic_con, qm_i2c_reg_t::ic_enable, qm_i2c_reg_t::ic_fs_scl_hcnt, qm_i2c_reg_t::ic_fs_scl_lcmt, qm_i2c_reg_t::ic_fs_spklen, qm_i2c_reg_t::ic_intr_mask, qm_i2c_context_t::ic_intr_mask, qm_i2c_reg_t::ic_rx_tl, qm_i2c_reg_t::ic_sar, qm_i2c_reg_t::ic_ss_scl_hcnt, qm_i2c_reg_t::ic_ss_scl_lcmt, qm_i2c_reg_t::ic_tx_tl, qm_i2c_context_t::sar, qm_i2c_context_t::ss_scl_hcnt, qm_i2c_context_t::ss_scl_lcmt, and qm_i2c_context_t::tx_tl.

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

Set I2C configuration.

Parameters

in	<i>i2c</i>	Which I2C to set the configuration of.
out	<i>cfg</i>	I2C configuration. 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 749 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_lcmt, 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_lcmt, qm_i2c_config_t::mode, QM_I2C_MASTER, QM_I2C_SLAVE, QM_I2C_SLAVE_INTERRUPT_WHEN_ADDRESSED, QM_I2C_SPEED_FAST, QM_I2C_SPEED_FAST_PLUS, QM_I2C_SPEED_STD, qm_i2c_config_t::slave_addr, qm_i2c_config_t::speed, and qm_i2c_config_t::stop_detect - behaviour.

4.9.3.12 `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	<i>i2c</i>	I2C index.
in	<i>speed</i>	Bus speed (Standard or Fast. Fast includes Fast+ mode).
in	<i>lo_cnt</i>	SCL low count.
in	<i>hi_cnt</i>	SCL high count.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 867 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_lcmt, qm_i2c_reg_t::ic_fs_spklen, qm_i2c_reg_t::ic_ss_scl_hcnt, qm_i2c_reg_t::ic_ss_scl_lcmt, QM_I2C_SPEED_FAST, QM_I2C_SPEED_FAST_PLUS, and QM_I2C_SPEED_STD.

4.9.3.13 `int qm_i2c_slave_dma_transfer (const qm_i2c_t i2c, const qm_i2c_transfer_t *const xfer)`

Perform a DMA based slave transfer on the I2C bus.

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

Parameters

in	<i>i2c</i>	I2C controller identifier.
in	<i>xfer</i>	Structure containing pre-allocated write and read data buffers and callback functions. This pointer 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.

4.9.3.14 int qm_i2c_slave_irq_transfer (const qm_i2c_t *i2c*, volatile const qm_i2c_transfer_t *const *xfer*)

Interrupt based slave transfer on I2C.

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

Parameters

in	<i>i2c</i>	Which I2C to transfer from.
in	<i>xfer</i>	Transfer structure includes write / read buffers, length, user callback function and the callback context. 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 1123 of file qm_i2c.c.

References qm_i2c_reg_t::ic_intr_mask, qm_i2c_reg_t::ic_rx_tl, and qm_i2c_reg_t::ic_tx_tl.

4.9.3.15 int qm_i2c_slave_irq_transfer_update (const qm_i2c_t *i2c*, volatile const qm_i2c_transfer_t *const *xfer*)

I2C interrupt based slave transfer buffer update.

Update transfer buffers location and size. The function will replenish/empty TX/RX FIFOs on I2C empty/full interrupts. This function must be called from callback function to update transfer buffers when requested by ISR.

It is strongly recommended to use this function for slave-based applications only, as slave controllers usually do not know how many frames an external master will send or request before starting the communication. Master controllers should not use this function as it will most likely corrupt the transaction.

Parameters

in	<i>i2c</i>	Which I2C to transfer from.
in	<i>xfer</i>	Transfer structure includes write / read buffers, length, user callback function and the callback context. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 1164 of file qm_i2c.c.

4.10 I2S

I2S.

Data Structures

- struct [qm_i2s_buffer_link](#)
Ring buffer link definition.
- struct [qm_i2s_channel_cfg_data_t](#)
I2S controller configuration.
- struct [qm_i2s_clock_cfg_data_t](#)
I2S Clock Configuration.

Typedefs

- typedef struct [qm_i2s_buffer_link](#) [qm_i2s_buffer_link_t](#)
Ring buffer link definition.

Enumerations

- enum [qm_i2s_master_slave_t](#) { [QM_I2S_MASTER](#) = 0, [QM_I2S_SLAVE](#) }
I2S master/slave configuration.
- enum [qm_i2s_audio_stream_t](#) { [QM_I2S_AUDIO_STREAM_TRANSMIT](#) = 0, [QM_I2S_AUDIO_STREAM_RECEIVE](#) }
I2S Audio stream identifiers.
- enum [qm_i2s_audio_format_t](#) { [QM_I2S_AUDIO_FORMAT_I2S_MODE](#) = 0, [QM_I2S_AUDIO_FORMAT_RIGHT_J](#), [QM_I2S_AUDIO_FORMAT_LEFT_J](#), [QM_I2S_AUDIO_FORMAT_DSP_MODE](#) }
I2S audio formats.
- enum [qm_i2s_sample_resolution_t](#) {
[QM_I2S_12_BIT_SAMPLE_RESOLUTION](#) = 12, [QM_I2S_13_BIT_SAMPLE_RESOLUTION](#) = 13, [QM_I2S_14_BIT_SAMPLE_RESOLUTION](#) = 14, [QM_I2S_15_BIT_SAMPLE_RESOLUTION](#) = 15,
[QM_I2S_16_BIT_SAMPLE_RESOLUTION](#) = 16, [QM_I2S_17_BIT_SAMPLE_RESOLUTION](#) = 17, [QM_I2S_18_BIT_SAMPLE_RESOLUTION](#) = 18, [QM_I2S_19_BIT_SAMPLE_RESOLUTION](#) = 19,
[QM_I2S_20_BIT_SAMPLE_RESOLUTION](#) = 20, [QM_I2S_21_BIT_SAMPLE_RESOLUTION](#) = 21, [QM_I2S_22_BIT_SAMPLE_RESOLUTION](#) = 22, [QM_I2S_23_BIT_SAMPLE_RESOLUTION](#) = 23,
[QM_I2S_24_BIT_SAMPLE_RESOLUTION](#) = 24, [QM_I2S_25_BIT_SAMPLE_RESOLUTION](#) = 25, [QM_I2S_26_BIT_SAMPLE_RESOLUTION](#) = 26, [QM_I2S_27_BIT_SAMPLE_RESOLUTION](#) = 27,
[QM_I2S_28_BIT_SAMPLE_RESOLUTION](#) = 28, [QM_I2S_29_BIT_SAMPLE_RESOLUTION](#) = 29, [QM_I2S_30_BIT_SAMPLE_RESOLUTION](#) = 30, [QM_I2S_31_BIT_SAMPLE_RESOLUTION](#) = 31,
[QM_I2S_32_BIT_SAMPLE_RESOLUTION](#) = 32 }
I2S Audio sample resolution.
- enum [qm_i2s_audio_rate_t](#) {
[QM_I2S_RATE_4000KHZ](#) = 0, [QM_I2S_RATE_8000KHZ](#), [QM_I2S_RATE_11025KHZ](#), [QM_I2S_RATE_16000KHZ](#),
[QM_I2S_RATE_22050KHZ](#), [QM_I2S_RATE_32000KHZ](#), [QM_I2S_RATE_44100KHZ](#), [QM_I2S_RATE_48000KHZ](#) }
I2S Audio rates available.
- enum [qm_i2s_buffer_mode_t](#) { [QM_I2S_TERMINATED_BUFFER](#) = 0, [QM_I2S_RING_BUFFER](#) }
I2S Audio buffer termination.
- enum [qm_i2s_clk_src_t](#) { [QM_I2S_INT_CLK](#) = 0, [QM_I2S_MCLK](#) }
I2S Reference Clock Source Select.

- enum `qm_i2s_status_t` {
`QM_I2S_FIFO_OVERFLOW` = BIT(0), `QM_I2S_FIFO_UNDERRUN` = BIT(1), `QM_I2S_TX_FIFO_FULL` = BIT(2), `QM_I2S_TX_FIFO_ALMOST_FULL` = BIT(3),
`QM_I2S_TX_FIFO_ALMOST_EMPTY` = BIT(4), `QM_I2S_TX_FIFO_EMPTY` = BIT(5), `QM_I2S_RX_FIFO_FULL` = BIT(6), `QM_I2S_RX_FIFO_ALMOST_FULL` = BIT(7),
`QM_I2S_RX_FIFO_ALMOST_EMPTY` = BIT(8), `QM_I2S_RX_FIFO_EMPTY` = BIT(9), `QM_I2S_DMA_ERROR` = BIT(10), `QM_I2S_DMA_COMPLETE` = BIT(11) }

I2S status type.

Functions

- int `qm_i2s_dma_channel_config` (const `qm_i2s_t` i2s, const `qm_i2s_audio_stream_t` audio_stream, const `qm_i2s_channel_cfg_data_t` *const config)
Function to configure specified I2S controller stream Configuration parameters must be valid or an error is returned - see return values below.
- int `qm_i2s_channel_disable` (const `qm_i2s_t` i2s, const `qm_i2s_audio_stream_t` audio_stream)
Function to place I2S controller into a disabled state.
- int `qm_i2s_set_clock_config` (const `qm_i2s_t` i2s, const `qm_i2s_clock_cfg_data_t` *const i2s_clk_cfg)
Function to configure and enable I2S clocks Configuration parameters must be valid or an error is returned - see return values below.
- int `qm_i2s_dma_transfer` (const `qm_i2s_t` i2s, const `qm_i2s_audio_stream_t` audio_stream)
Function to transmit a block of data to the specified I2S channel.
- int `qm_i2s_dma_transfer_terminate` (const `qm_i2s_t` i2s, `qm_i2s_channel_cfg_data_t` *const config)
Terminate the current DMA transfer on the given I2S peripheral channel.

4.10.1 Detailed Description

I2S.

4.10.2 Enumeration Type Documentation

4.10.2.1 enum `qm_i2s_audio_format_t`

I2S audio formats.

Enumerator

`QM_I2S_AUDIO_FORMAT_I2S_MODE` I2S Mode audio format.
`QM_I2S_AUDIO_FORMAT_RIGHT_J` Right justified audio format.
`QM_I2S_AUDIO_FORMAT_LEFT_J` Left justified audio format.
`QM_I2S_AUDIO_FORMAT_DSP_MODE` DSP mode audio format.

Definition at line 38 of file `qm_i2s.h`.

4.10.2.2 enum `qm_i2s_audio_rate_t`

I2S Audio rates available.

Enumerator

`QM_I2S_RATE_4000KHZ` 4kHz audio rate
`QM_I2S_RATE_8000KHZ` 8kHz audio rate
`QM_I2S_RATE_11025KHZ` 11.025kHz audio rate
`QM_I2S_RATE_16000KHZ` 16kHz audio rate

QM_I2S_RATE_22050KHZ 22.05kHz audio rate

QM_I2S_RATE_32000KHZ 32kHz audio rate

QM_I2S_RATE_44100KHZ 44.1kHz audio rate

QM_I2S_RATE_48000KHZ 48kHz audio rate

Definition at line 96 of file qm_i2s.h.

4.10.2.3 enum qm_i2s_audio_stream_t

I2S Audio stream identifiers.

Enumerator

QM_I2S_AUDIO_STREAM_TRANSMIT Playback audio stream.

QM_I2S_AUDIO_STREAM_RECEIVE Capture audio stream.

Definition at line 30 of file qm_i2s.h.

4.10.2.4 enum qm_i2s_buffer_mode_t

I2S Audio buffer termination.

Enumerator

QM_I2S_TERMINATED_BUFFER Used for single audio playback/recording.

QM_I2S_RING_BUFFER Used for continuous audio playback/recording.

Definition at line 110 of file qm_i2s.h.

4.10.2.5 enum qm_i2s_clk_src_t

I2S Reference Clock Source Select.

Enumerator

QM_I2S_INT_CLK Use internal clock.

QM_I2S_MCLK Take in clock from external source.

Definition at line 119 of file qm_i2s.h.

4.10.2.6 enum qm_i2s_master_slave_t

I2S master/slave configuration.

Enumerator

QM_I2S_MASTER Master configuration selection.

QM_I2S_SLAVE Slave configuration selection.

Definition at line 22 of file qm_i2s.h.

4.10.2.7 enum qm_i2s_sample_resolution_t

I2S Audio sample resolution.

Resolution ranges between 12 bits and 32 bits.

Enumerator

QM_I2S_12_BIT_SAMPLE_RESOLUTION 12 bits audio sample resolution

<i>QM_I2S_13_BIT_SAMPLE_RESOLUTION</i>	13 bits audio sample resolution
<i>QM_I2S_14_BIT_SAMPLE_RESOLUTION</i>	14 bits audio sample resolution
<i>QM_I2S_15_BIT_SAMPLE_RESOLUTION</i>	15 bits audio sample resolution
<i>QM_I2S_16_BIT_SAMPLE_RESOLUTION</i>	16 bits audio sample resolution
<i>QM_I2S_17_BIT_SAMPLE_RESOLUTION</i>	17 bits audio sample resolution
<i>QM_I2S_18_BIT_SAMPLE_RESOLUTION</i>	18 bits audio sample resolution
<i>QM_I2S_19_BIT_SAMPLE_RESOLUTION</i>	19 bits audio sample resolution
<i>QM_I2S_20_BIT_SAMPLE_RESOLUTION</i>	20 bits audio sample resolution
<i>QM_I2S_21_BIT_SAMPLE_RESOLUTION</i>	21 bits audio sample resolution
<i>QM_I2S_22_BIT_SAMPLE_RESOLUTION</i>	22 bits audio sample resolution
<i>QM_I2S_23_BIT_SAMPLE_RESOLUTION</i>	23 bits audio sample resolution
<i>QM_I2S_24_BIT_SAMPLE_RESOLUTION</i>	24 bits audio sample resolution
<i>QM_I2S_25_BIT_SAMPLE_RESOLUTION</i>	25 bits audio sample resolution
<i>QM_I2S_26_BIT_SAMPLE_RESOLUTION</i>	26 bits audio sample resolution
<i>QM_I2S_27_BIT_SAMPLE_RESOLUTION</i>	27 bits audio sample resolution
<i>QM_I2S_28_BIT_SAMPLE_RESOLUTION</i>	28 bits audio sample resolution
<i>QM_I2S_29_BIT_SAMPLE_RESOLUTION</i>	29 bits audio sample resolution
<i>QM_I2S_30_BIT_SAMPLE_RESOLUTION</i>	30 bits audio sample resolution
<i>QM_I2S_31_BIT_SAMPLE_RESOLUTION</i>	31 bits audio sample resolution
<i>QM_I2S_32_BIT_SAMPLE_RESOLUTION</i>	32 bits audio sample resolution

Definition at line 48 of file qm_i2s.h.

4.10.2.8 enum qm_i2s_status_t

I2S status type.

Enumerator

<i>QM_I2S_FIFO_OVERFLOW</i>	Indicates that a FIFO overflow error was detected.
<i>QM_I2S_FIFO_UNDERRUN</i>	Indicates that a FIFO underrun error was detected.
<i>QM_I2S_TX_FIFO_FULL</i>	Indicates TX FIFO full interrupt occurred.
<i>QM_I2S_TX_FIFO_ALMOST_FULL</i>	Indicates TX FIFO almost full interrupt occurred.
<i>QM_I2S_TX_FIFO_ALMOST_EMPTY</i>	Indicates TX FIFO almost empty interrupt occurred.
<i>QM_I2S_TX_FIFO_EMPTY</i>	Indicates TX FIFO empty interrupt occurred.
<i>QM_I2S_RX_FIFO_FULL</i>	Indicates RX FIFO full interrupt occurred.
<i>QM_I2S_RX_FIFO_ALMOST_FULL</i>	Indicates RX FIFO almost full interrupt occurred.
<i>QM_I2S_RX_FIFO_ALMOST_EMPTY</i>	Indicates RX FIFO almost empty interrupt occurred.
<i>QM_I2S_RX_FIFO_EMPTY</i>	Indicates RX FIFO empty interrupt occurred.
<i>QM_I2S_DMA_ERROR</i>	Indicates that a DMA error was detected.
<i>QM_I2S_DMA_COMPLETE</i>	Indicates that DMA operation was completed.

Definition at line 127 of file qm_i2s.h.

4.10.3 Function Documentation

4.10.3.1 int qm_i2s_channel_disable (const qm_i2s_t i2s, const qm_i2s_audio_stream_t audio_stream)

Function to place I2S controller into a disabled state.

This function assumes that there is no pending transaction on the I2S interface in question. It is the responsibility of the calling application to do so.

Parameters

in	<i>i2s</i>	I2S controller identifier
in	<i>audio_stream</i>	I2S audio_stream identifier

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.10.3.2 int qm_i2s_dma_channel_config (const qm_i2s_t *i2s*, const qm_i2s_audio_stream_t *audio_stream*, const qm_i2s_channel_cfg_data_t *const *config*)

Function to configure specified I2S controller stream Configuration parameters must be valid or an error is returned - see return values below.

Parameters

in	<i>i2s</i>	I2S controller identifier
in	<i>audio_stream</i>	I2S audio_stream identifier
in	<i>config</i>	Pointer to configuration structure

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.10.3.3 int qm_i2s_dma_transfer (const qm_i2s_t *i2s*, const qm_i2s_audio_stream_t *audio_stream*)

Function to transmit a block of data to the specified I2S channel.

Parameters

in	<i>i2s</i>	I2S controller identifier
in	<i>audio_stream</i>	I2S audio_stream identifier

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.10.3.4 int qm_i2s_dma_transfer_terminate (const qm_i2s_t *i2s*, qm_i2s_channel_cfg_data_t *const *config*)

Terminate the current DMA transfer on the given I2S peripheral channel.

Terminate the current DMA transfer on the I2S channel. This will cause the relevant callbacks to be invoked.

In the case the user is doing a continuous transfer using a circular linked list, it might be useful to fine control at which point in the linked list that we stop the transfer. To do this, set the relevant linked list item's "next" item to NULL.

Parameters

in	<i>i2s</i>	I2S controller identifier
in	<i>config</i>	Pointer to configuration structure

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

4.10.3.5 int qm_i2s_set_clock_config (const qm_i2s_t i2s, const qm_i2s_clock_cfg_data_t *const i2s_clk_cfg)

Function to configure and enable I2S clocks Configuration parameters must be valid or an error is returned - see return values below.

Parameters

in	<i>i2s</i>	I2S controller identifier
in	<i>i2s_clk_cfg</i>	I2S Clock configuration

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

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

<code>in</code>	<code>reset_type</code>	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)
Unconditionally enable interrupt delivery on the CPU.
- void [qm_irq_disable](#) (void)
Unconditionally disable interrupt delivery on the CPU.
- unsigned int [qm_irq_lock](#) (void)
Save interrupt state and disable all interrupts on the CPU.
- void [qm_irq_unlock](#) (unsigned int key)
Restore previous interrupt state on the CPU saved via [qm_irq_lock\(\)](#).
- 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	<i>vector</i>	Vector number.
in	<i>isr</i>	ISR to register to given IRQ.

Definition at line 163 of file qm_interrupt.h.

4.13.2.2 unsigned int qm_irq_lock (void)

Save interrupt state and disable all interrupts on the CPU.

This routine disables interrupts. It can be called from either interrupt or non-interrupt context. This routine returns an architecture-dependent lock-out key representing the "interrupt disable state" prior to the call; this key can be passed to [qm_irq_unlock\(\)](#) to re-enable interrupts.

This function can be called recursively: it will return a key to return the state of interrupt locking to the previous level.

Returns

An architecture-dependent lock-out key representing the "interrupt disable state" prior to the call.

Definition at line 74 of file qm_interrupt.c.

4.13.2.3 void qm_irq_mask (uint32_t irq)

Mask a given interrupt line.

Parameters

<i>in</i>	<i>irq</i>	Which IRQ to mask.
-----------	------------	--------------------

Definition at line 111 of file qm_interrupt.c.

References qm_ss_irq_mask().

4.13.2.4 void qm_irq_unlock (unsigned int key)

Restore previous interrupt state on the CPU saved via [qm_irq_lock\(\)](#).

Parameters

<i>in</i>	<i>key</i>	architecture-dependent lock-out key returned by a previous invocation of qm_irq_lock() .
-----------	------------	--

Definition at line 92 of file qm_interrupt.c.

4.13.2.5 void qm_irq_unmask (uint32_t irq)

Unmask a given interrupt line.

Parameters

<i>in</i>	<i>irq</i>	Which IRQ to unmask.
-----------	------------	----------------------

Definition at line 125 of file qm_interrupt.c.

References qm_ss_irq_unmask().

4.14 ISR

Interrupt Service Routines.

Functions

- [QM_ISR_DECLARE](#) (qm_adc_0_cal_isr)
ISR for ADC 0 convert and calibration interrupt.
- [QM_ISR_DECLARE](#) (qm_adc_0_pwr_isr)
ISR for ADC 0 change mode interrupt.
- [QM_ISR_DECLARE](#) (qm_aonpt_0_isr)
ISR for Always-on Periodic Timer 0 interrupt.
- [QM_ISR_DECLARE](#) (qm_comparator_0_isr)
ISR for Analog Comparator 0 interrupt.
- [QM_ISR_DECLARE](#) (qm_dma_0_error_isr)
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_flash_mpr_0_isr)
ISR for FPR 0 interrupt.
- [QM_ISR_DECLARE](#) (qm_flash_mpr_1_isr)
ISR for FPR 1 interrupt.
- [QM_ISR_DECLARE](#) (qm_gpio_0_isr)
ISR for GPIO 0 interrupt.
- [QM_ISR_DECLARE](#) (qm_aon_gpio_0_isr)
ISR for AON GPIO 0 interrupt.
- [QM_ISR_DECLARE](#) (qm_i2c_0_irq_isr)
ISR for I2C 0 irq mode transfer interrupt.
- [QM_ISR_DECLARE](#) (qm_i2c_1_irq_isr)
ISR for I2C 1 irq mode transfer interrupt.
- [QM_ISR_DECLARE](#) (qm_i2c_0_dma_isr)
ISR for I2C 0 dma mode transfer interrupt.
- [QM_ISR_DECLARE](#) (qm_i2c_1_dma_isr)
ISR for I2C 1 dma mode transfer interrupt.
- [QM_ISR_DECLARE](#) (qm_mailbox_0_isr)
ISR for Mailbox interrupt.
- [QM_ISR_DECLARE](#) (qm_sram_mpr_0_isr)

ISR for Memory Protection Region interrupt.

- [QM_ISR_DECLARE](#) (qm_pic_timer_0_isr)

ISR for PIC Timer interrupt.

- [QM_ISR_DECLARE](#) (qm_pwm_0_isr_0)

ISR for PWM 0 Channel 0 interrupt.

- [QM_ISR_DECLARE](#) (qm_pwm_0_isr_1)

ISR for PWM 0 channel 1 interrupt.

- [QM_ISR_DECLARE](#) (qm_pwm_0_isr_2)

ISR for PWM 0 channel 2 interrupt.

- [QM_ISR_DECLARE](#) (qm_pwm_0_isr_3)

ISR for PWM 0 channel 3 interrupt.

- [QM_ISR_DECLARE](#) (qm_rtc_0_isr)

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_spi_slave_0_isr)

ISR for SPI Slave 0 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_0_isr)

ISR for WDT 0 interrupt.

- [QM_ISR_DECLARE](#) (qm_wdt_1_isr)

ISR for WDT 1 interrupt.

- [QM_ISR_DECLARE](#) (qm_usb_0_isr)

ISR for USB 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_cal_isr)

ISR for ADC 0 convert and calibration interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_ADC_0_CAL_INT, qm_adc_0_cal_isr);
```

if IRQ based transfers are used.

Definition at line 166 of file qm_adc.c.

4.14.2.2 QM_ISR_DECLARE (qm_adc_0_pwr_isr)

ISR for ADC 0 change mode interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_ADC_0_PWR_INT, qm_adc_0_pwr_isr);
```

if IRQ based transfers are used.

Definition at line 174 of file qm_adc.c.

4.14.2.3 QM_ISR_DECLARE (qm_aonpt_0_isr)

ISR for Always-on Periodic Timer 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_AONPT_0_INT, qm_aonpt_0_isr);
```

if IRQ based transfers are used.

Definition at line 103 of file qm_aon_counters.c.

References qm_power_soc_restore().

4.14.2.4 QM_ISR_DECLARE (qm_comparator_0_isr)

ISR for Analog Comparator 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_COMPARATOR_0_INT, qm_comparator_0_isr);
```

if IRQ based transfers are used.

Definition at line 17 of file qm_comparator.c.

References qm_power_soc_restore().

4.14.2.5 QM_ISR_DECLARE (qm_dma_0_error_isr)

ISR for DMA error interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_DMA_0_ERROR_INT, qm_dma_0_error_isr);
```

if IRQ based transfers are used.

Definition at line 152 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_INT_0, qm_dma_0_isr_0);
```

if IRQ based transfers are used.

Definition at line 158 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_0_INT_1, qm_dma_0_isr_1);
```

if IRQ based transfers are used.

Definition at line 164 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_0_INT_2, qm_dma_0_isr_2);
```

if IRQ based transfers are used.

Definition at line 171 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_0_INT_3, qm_dma_0_isr_3);
```

if IRQ based transfers are used.

Definition at line 177 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_0_INT_4, qm_dma_0_isr_4);
```

if IRQ based transfers are used.

Definition at line 183 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_0_INT_5, qm_dma_0_isr_5);
```

if IRQ based transfers are used.

Definition at line 189 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_0_INT_6, qm_dma_0_isr_6);
```

if IRQ based transfers are used.

Definition at line 195 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_0_INT_7, qm_dma_0_isr_7);
```

if IRQ based transfers are used.

Definition at line 201 of file qm_dma.c.

References QM_DMA_0, and QM_DMA_CHANNEL_7.

4.14.2.14 QM_ISR_DECLARE (qm_flash_mpr_0_isr)

ISR for FPR 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_FLASH_MPR_0_INT, qm_flash_mpr_0_isr);
```

if IRQ based transfers are used.

Definition at line 12 of file qm_fpr.c.

4.14.2.15 QM_ISR_DECLARE (qm_flash_mpr_1_isr)

ISR for FPR 1 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_FLASH_MPR_1_INT, qm_flash_mpr_1_isr);
```

if IRQ based transfers are used.

Definition at line 23 of file qm_fpr.c.

4.14.2.16 QM_ISR_DECLARE (qm_gpio_0_isr)

ISR for GPIO 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_GPIO_0_INT, qm_gpio_0_isr);
```

if IRQ based transfers are used.

Definition at line 45 of file qm_gpio.c.

4.14.2.17 QM_ISR_DECLARE (qm_aon_gpio_0_isr)

ISR for AON GPIO 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_AON_GPIO_0_INT, qm_aon_gpio_0_isr);
```

if IRQ based transfers are used.

Definition at line 52 of file qm_gpio.c.

4.14.2.18 QM_ISR_DECLARE (qm_i2c_0_irq_isr)

ISR for I2C 0 irq mode transfer interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_I2C_0_INT, qm_i2c_0_irq_isr);
```

if IRQ based transfers are used.

Definition at line 711 of file qm_i2c.c.

4.14.2.19 QM_ISR_DECLARE (qm_i2c_1_irq_isr)

ISR for I2C 1 irq mode transfer interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_I2C_1_INT, qm_i2c_1_irq_isr);
```

if IRQ based transfers are used.

Definition at line 724 of file qm_i2c.c.

4.14.2.20 QM_ISR_DECLARE (qm_i2c_0_dma_isr)

ISR for I2C 0 dma mode transfer interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_I2C_0_INT, qm_i2c_0_dma_isr);
```

if DMA based transfers are used.

Definition at line 717 of file qm_i2c.c.

4.14.2.21 QM_ISR_DECLARE (qm_i2c_1_dma_isr)

ISR for I2C 1 dma mode transfer interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_I2C_1_INT, qm_i2c_1_dma_isr);
```

if DMA based transfers are used.

Definition at line 730 of file qm_i2c.c.

4.14.2.22 QM_ISR_DECLARE (qm_mailbox_0_isr)

ISR for Mailbox interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_MAILBOX_0_INT, qm_mailbox_0_isr);
```

if IRQ based transfers are used.

Definition at line 66 of file qm_mailbox_se.c.

4.14.2.23 QM_ISR_DECLARE (qm_sram_mpr_0_isr)

ISR for Memory Protection Region interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_SRAM_MPR_0_INT, qm_sram_mpr_0_isr);
```

if IRQ based transfers are used.

Definition at line 14 of file qm_mpr.c.

4.14.2.24 QM_ISR_DECLARE (qm_pic_timer_0_isr)

ISR for PIC Timer interrupt.

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

```
qm_int_vector_request(QM_X86_PIC_TIMER_INT_VECTOR,
qm_pic_timer_0_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_0_isr);
```

if IRQ based transfers are used.

Definition at line 27 of file qm_pic_timer.c.

4.14.2.25 QM_ISR_DECLARE (qm_pwm_0_isr_0)

ISR for PWM 0 Channel 0 interrupt.

If there is only one interrupt per controller this ISR handles all channel interrupts.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_PWM_0_INT, qm_pwm_0_isr);
```

if IRQ based transfers are used.

Definition at line 62 of file qm_pwm.c.

References qm_pwm_channel_t::eoi, qm_pwm_reg_t::timer, and qm_pwm_reg_t::timersintstatus.

4.14.2.26 QM_ISR_DECLARE (qm_pwm_0_isr_1)

ISR for PWM 0 channel 1 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_PWM_1, qm_pwm_0_isr_1);
```

if IRQ based transfers are used.

Definition at line 46 of file qm_pwm.c.

4.14.2.27 QM_ISR_DECLARE (qm_pwm_0_isr_2)

ISR for PWM 0 channel 2 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_PWM_2, qm_pwm_0_isr_2);
```

if IRQ based transfers are used.

Definition at line 51 of file qm_pwm.c.

4.14.2.28 QM_ISR_DECLARE (qm_pwm_0_isr_3)

ISR for PWM 0 channel 3 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_PWM_3, qm_pwm_0_isr_3);
```

if IRQ based transfers are used.

Definition at line 56 of file qm_pwm.c.

4.14.2.29 QM_ISR_DECLARE (qm_rtc_0_isr)

ISR for RTC 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_RTC_0_INT, qm_rtc_0_isr);
```

if IRQ based transfers are used.

Definition at line 18 of file qm_rtc.c.

References qm_power_soc_restore().

4.14.2.30 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_INT, qm_spi_master_0_isr);
```

if IRQ based transfers are used.

Definition at line 716 of file qm_spi.c.

4.14.2.31 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_INT, qm_spi_master_1_isr);
```

if IRQ based transfers are used.

Definition at line 723 of file qm_spi.c.

4.14.2.32 QM_ISR_DECLARE (qm_spi_slave_0_isr)

ISR for SPI Slave 0 interrupt.

This function needs to be registered with

```
qm_irq_request(QM_IRQ_SPI_SLAVE_0_INT, qm_spi_slave_0_isr);
```

if IRQ based transfers are used.

Definition at line 730 of file qm_spi.c.

4.14.2.33 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_INT, qm_uart_0_isr);
```

if IRQ based transfers are used.

Definition at line 203 of file qm_uart.c.

4.14.2.34 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_INT, qm_uart_1_isr);
```

if IRQ based transfers are used.

Definition at line 209 of file qm_uart.c.

4.14.2.35 QM_ISR_DECLARE (qm_wdt_0_isr)

ISR for WDT 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_WDT_0_INT, qm_wdt_0_isr);
```

if IRQ based transfers are used.

Definition at line 20 of file qm_wdt.c.

4.14.2.36 QM_ISR_DECLARE (qm_wdt_1_isr)

ISR for WDT 1 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_WDT_1_INT, qm_wdt_1_isr);
```

if IRQ based transfers are used.

Definition at line 32 of file qm_wdt.c.

4.14.2.37 QM_ISR_DECLARE (qm_usb_0_isr)

ISR for USB 0 interrupt.

This function needs to be registered with

```
QM_IRQ_REQUEST(QM_IRQ_USB_0_INT, qm_usb_0_isr);
```

if IRQ based transfers are used.

Definition at line 410 of file qm_usb.c.

4.15 Mailbox

Mailbox driver.

Data Structures

- struct [qm_mbox_msg_t](#)
Definition of the mailbox message.
- struct [qm_mbox_config_t](#)
Mailbox Configuration Structure.

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_PEND](#) = [QM_MBOX_CH_STS](#), [QM_MBOX_CH_INT_AND_DATA_PEND](#) }
- Mailbox channel status return codes.*
- 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 payload index values.*
- enum [qm_mbox_mode_t](#) { [QM_MBOX_INTERRUPT_MODE](#) = 0, [QM_MBOX_POLLING_MODE](#) }
- Definition of the mailbox mode of operation, interrupt mode or polling mode.*

Functions

- int [qm_mbox_ch_set_config](#) (const [qm_mbox_ch_t](#) mbox_ch, const [qm_mbox_config_t](#) *const config)
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.

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

<i>in</i>	<i>data</i>	The callback user data.
-----------	-------------	-------------------------

Definition at line 78 of file qm_mailbox.h.

4.15.3 Enumeration Type Documentation

4.15.3.1 enum qm_mbox_ch_status_t

Mailbox channel status return codes.

Enumerator

QM_MBOX_CH_IDLE No interrupt pending nor any data to consume.

QM_MBOX_CH_DATA_PEND Receiver has serviced the interrupt and data has not been consumed.

QM_MBOX_CH_INT_AND_DATA_PEND Receiver hasn't serviced the interrupt and data has not been consumed.

Definition at line 23 of file qm_mailbox.h.

4.15.3.2 enum qm_mbox_mode_t

Definition of the mailbox mode of operation, interrupt mode or polling mode.

Enumerator

QM_MBOX_INTERRUPT_MODE Mailbox channel operates in interrupt mode.

QM_MBOX_POLLING_MODE Mailbox channel operates in polling mode.

Definition at line 54 of file qm_mailbox.h.

4.15.3.3 enum qm_mbox_payload_t

Mailbox message payload 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 Number of payloads.

Definition at line 43 of file qm_mailbox.h.

4.15.4 Function Documentation

4.15.4.1 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	<i>mbox_ch</i>	Mailbox identifier to retrieve the status from.
out	<i>status</i>	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 248 of file qm_mailbox_se.c.

References QM_MBOX_CH_0.

4.15.4.2 `int qm_mbox_ch_read (const qm_mbox_ch_t mbox_ch, qm_mbox_msg_t *const msg)`

Read specified mailbox channel.

Parameters

in	<i>mbox_ch</i>	Mailbox channel identifier.
out	<i>msg</i>	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 198 of file qm_mailbox_se.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, and QM_MBOX_CH_0.

4.15.4.3 `int qm_mbox_ch_set_config (const qm_mbox_ch_t mbox_ch, const qm_mbox_config_t *const config)`

Set the mailbox channel configuration.

The function registers the interrupt vector to the mailbox ISR handler when at least one mailbox channel is enabled, configured in interrupt mode and the ISR is not handled by the application.

Parameters

in	<i>mbox_ch</i>	Mailbox channel identifier.
in	<i>config</i>	Mailbox configuration.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
---	-------------

<i>Negative</i>	errno for possible error codes.
-----------------	---

Definition at line 99 of file qm_mailbox_se.c.

References qm_mbox_config_t::callback, qm_mbox_config_t::dest, qm_mbox_config_t::mode, QM_MBOX_CH_0, QM_MBOX_INTERRUPT_MODE, and QM_MBOX_POLLING_MODE.

4.15.4.4 `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

<i>in</i>	<i>mbox_ch</i>	Mailbox channel identifier.
<i>in</i>	<i>msg</i>	Pointer to the data to write to the mailbox channel. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 171 of file qm_mailbox_se.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_0, 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.
- int [qm_mpr_save_context](#) ([qm_mpr_context_t](#) *const ctx)
Save MPR context.
- int [qm_mpr_restore_context](#) (const [qm_mpr_context_t](#) *const ctx)
Restore MPR context.

4.16.1 Detailed Description

Memory Protection Region control for Quark Microcontrollers.

4.16.2 Function Documentation

4.16.2.1 int [qm_mpr_restore_context](#) (const [qm_mpr_context_t](#) *const ctx)

Restore MPR context.

Restore the configuration of the specified MPR peripheral after exiting sleep.

MPR configuration is lost after sleep and can therefore be modified even if this configuration was locked before sleep. To support persistent configuration, the configuration must be restored when resuming as part of the bootloader.

Parameters

in	ctx	MPR context structure. 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 167 of file [qm_mpr.c](#).

References [qm_mpr_reg_t::mpr_cfg](#), [qm_mpr_context_t::mpr_cfg](#), and [QM_MPR_NUM](#).

4.16.2.2 `int qm_mpr_save_context (qm_mpr_context_t *const ctx)`

Save MPR context.

Save the configuration of the specified MPR peripheral before entering sleep.

MPR configuration is lost after sleep and can therefore be modified even if this configuration was locked before sleep. To support persistent configuration, the configuration must be restored when resuming as part of the bootloader.

Parameters

<i>out</i>	<i>ctx</i>	MPR context structure. This must not be NULL.
------------	------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 160 of file `qm_mpr.c`.

References `qm_mpr_reg_t::mpr_cfg`, `qm_mpr_context_t::mpr_cfg`, and `QM_MPR_NUM`.

4.16.2.3 `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

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

Returns

int 0 on success, error code otherwise.

Definition at line 24 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`, `QM_MPR_NUM`, and `qm_mpr_config_t::up_bound`.

4.16.2.4 `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

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

Returns

int 0 on success, error code otherwise.

Definition at line 89 of file `qm_mpr.c`.

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.
- int `qm_pic_timer_save_context` (`qm_pic_timer_context_t` *const `ctx`)
Save PIC Timer peripheral's context.
- int `qm_pic_timer_restore_context` (const `qm_pic_timer_context_t` *const `ctx`)
Restore PIC Timer peripheral's context.

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

<i>out</i>	<i>count</i>	Pointer to the store the timer count. This must not be NULL.
------------	--------------	--

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 76 of file qm_pic_timer.c.

4.17.3.2 int qm_pic_timer_restore_context (const qm_pic_timer_context_t *const ctx)

Restore PIC Timer peripheral's context.

Restore the configuration of the specified PIC Timer peripheral after exiting sleep. The timer is restored to the count saved before sleep.

Parameters

<i>in</i>	<i>ctx</i>	PIC Timer context structure. This must not be NULL.
-----------	------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 116 of file qm_pic_timer.c.

References qm_pic_timer_context_t::lvttimer, qm_pic_timer_context_t::timer_dcr, and qm_pic_timer_context_t::timer_icr.

4.17.3.3 int qm_pic_timer_save_context (qm_pic_timer_context_t *const ctx)

Save PIC Timer peripheral's context.

Saves the configuration of the specified PIC Timer peripheral before entering sleep.

Parameters

<i>out</i>	<i>ctx</i>	PIC Timer context structure. This must not be NULL.
------------	------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 109 of file qm_pic_timer.c.

References qm_pic_timer_context_t::lvttimer, qm_pic_timer_context_t::timer_dcr, and qm_pic_timer_context_t::timer_icr.

4.17.3.4 `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

<i>in</i>	<i>count</i>	Value to load the timer with.
-----------	--------------	-------------------------------

Returns

Standard errno return type for QMSI.

Return values

<i>Always</i>	returns 0.
<i>Negative</i>	errno for possible error codes.

Definition at line 69 of file qm_pic_timer.c.

4.17.3.5 `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

<i>in</i>	<i>cfg</i>	PIC timer configuration. This must not be NULL.
-----------	------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	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

<code>in</code>	<code>pin</code>	which pin to configure.
<code>in</code>	<code>enable</code>	set to true to enable input.

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	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

<code>in</code>	<code>pin</code>	which pin to configure.
<code>in</code>	<code>enable</code>	set to true to enable pullup.

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	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

<code>in</code>	<code>pin</code>	which pin to configure.
<code>in</code>	<code>fn</code>	the function to assign to the pin.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	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_pmx_set_slew (const qm_pin_id_t pin, const qm_pmx_slew_t slew)`

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

Parameters

<i>in</i>	<i>pin</i>	which pin to configure.
<i>in</i>	<i>slew</i>	the slew rate to assign to the pin.

Returns

Standard `errno` return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	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](#)
QM PWM / Timer configuration type.

Enumerations

- enum [qm_pwm_mode_t](#) { [QM_PWM_MODE_TIMER_FREE_RUNNING](#) = QM_PWM_MODE_TIMER_FREE_RUNNING_VALUE, [QM_PWM_MODE_TIMER_COUNT](#) = QM_PWM_MODE_TIMER_COUNT_VALUE }
- QM PWM 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.
- int [qm_pwm_save_context](#) (const [qm_pwm_t](#) pwm, [qm_pwm_context_t](#) *const ctx)
Save PWM peripheral's context.
- int [qm_pwm_restore_context](#) (const [qm_pwm_t](#) pwm, const [qm_pwm_context_t](#) *const ctx)
Restore PWM peripheral's context.

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](#)

QM PWM mode type.

Enumerator

QM_PWM_MODE_TIMER_FREE_RUNNING Timer: Free running mode. Timer: Counter mode.

QM_PWM_MODE_TIMER_COUNT PWM mode.

Definition at line 21 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	<i>pwm</i>	Which PWM module to get the count of.
in	<i>id</i>	PWM channel id to read the values of.
out	<i>lo_count</i>	Num of cycles the output is driven low. This must not be NULL.
out	<i>hi_count</i>	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 151 of file qm_pwm.c.

References qm_pwm_channel_t::loadcount, qm_pwm_reg_t::timer, and qm_pwm_reg_t::timer_loadcount2.

4.19.3.2 int qm_pwm_restore_context (const qm_pwm_t *pwm*, const qm_pwm_context_t *const *ctx*)

Restore PWM peripheral's context.

Restore the configuration of the specified PWM peripheral after exiting sleep.

Parameters

in	<i>pwm</i>	PWM device.
in	<i>ctx</i>	PWM context structure. 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 210 of file qm_pwm.c.

References qm_pwm_channel_t::controlreg, qm_pwm_context_t::controlreg, qm_pwm_channel_t::loadcount, qm_pwm_context_t::loadcount, qm_pwm_context_t::loadcount2, qm_pwm_reg_t::timer, and qm_pwm_reg_t::timer_loadcount2.

4.19.3.3 int qm_pwm_save_context (const qm_pwm_t *pwm*, qm_pwm_context_t *const *ctx*)

Save PWM peripheral's context.

Saves the configuration of the specified PWM peripheral before entering sleep.

Parameters

in	<i>pwm</i>	PWM device.
out	<i>ctx</i>	PWM context structure. 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 202 of file qm_pwm.c.

References qm_pwm_channel_t::controlreg, qm_pwm_context_t::controlreg, qm_pwm_channel_t::loadcount, qm_pwm_context_t::loadcount, qm_pwm_context_t::loadcount2, qm_pwm_reg_t::timer, and qm_pwm_reg_t::timer_loadcount2.

4.19.3.4 `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	<i>pwm</i>	Which PWM module to set the counts of.
in	<i>id</i>	PWM channel id to set.
in	<i>lo_count</i>	Num of cycles the output is driven low.
in	<i>hi_count</i>	Num of cycles the output is driven high.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 131 of file qm_pwm.c.

References qm_pwm_channel_t::loadcount, qm_pwm_reg_t::timer, and qm_pwm_reg_t::timer_loadcount2.

4.19.3.5 `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	<i>pwm</i>	Which PWM module to configure.
in	<i>id</i>	PWM channel id to configure.
in	<i>cfg</i>	New configuration for PWM. 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_pwm.c.

References qm_pwm_config_t::callback, qm_pwm_config_t::callback_data, qm_pwm_channel_t::controlreg, qm_pwm_config_t::hi_count, qm_pwm_config_t::lo_count, qm_pwm_channel_t::loadcount, qm_pwm_config_t::mask_interrupt, qm_pwm_config_t::mode, qm_pwm_reg_t::timer, and qm_pwm_reg_t::timer_loadcount2.

4.19.3.6 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 83 of file qm_pwm.c.

References qm_pwm_channel_t::controlreg, and qm_pwm_reg_t::timer.

4.19.3.7 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 94 of file qm_pwm.c.

References qm_pwm_channel_t::controlreg, and qm_pwm_reg_t::timer.

4.20 RTC

Real Time Clock.

Data Structures

- struct [qm_rtc_config_t](#)
QM 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.
- int [qm_rtc_read](#) (const [qm_rtc_t](#) rtc, uint32_t *const value)
Read the RTC register value.
- int [qm_rtc_save_context](#) (const [qm_rtc_t](#) rtc, [qm_rtc_context_t](#) *const ctx)
Save RTC context.
- int [qm_rtc_restore_context](#) (const [qm_rtc_t](#) rtc, const [qm_rtc_context_t](#) *const ctx)
Restore RTC context.

4.20.1 Detailed Description

Real Time Clock.

Warning

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 a transaction to complete the SOC will not wake from sleep.

4.20.2 Function Documentation

4.20.2.1 int qm_rtc_read (const [qm_rtc_t](#) rtc, uint32_t *const value)

Read the RTC register value.

Parameters

in	<i>rtc</i>	RTC index.
out	<i>value</i>	Location to store RTC value. This must not be NULL.

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.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 101 of file qm_rtc.c.

4.20.2.2 `int qm_rtc_restore_context (const qm_rtc_t rtc, const qm_rtc_context_t *const ctx)`

Restore RTC context.

Restore the configuration of the specified RTC peripheral after exiting sleep.

Parameters

in	rtc	RTC index.
in	ctx	RTC context structure. 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 158 of file qm_rtc.c.

4.20.2.3 `int qm_rtc_save_context (const qm_rtc_t rtc, qm_rtc_context_t *const ctx)`

Save RTC context.

Save the configuration of the specified RTC peripheral before entering sleep.

Parameters

in	rtc	RTC index.
out	ctx	RTC context structure. 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 150 of file qm_rtc.c.

4.20.2.4 `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	<i>rtc</i>	RTC index.
in	<i>alarm_val</i>	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 85 of file qm_rtc.c.

Referenced by qm_rtc_set_config().

4.20.2.5 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.

Parameters

in	<i>rtc</i>	RTC index.
in	<i>cfg</i>	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 37 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, qm_rtc_config_t::prescaler, and qm_rtc_set_alarm().

4.21 SPI

SPI peripheral driver for Quark Microcontrollers.

Data Structures

- struct [qm_spi_async_transfer_t](#)
SPI asynchronous transfer type.
- struct [qm_spi_transfer_t](#)
SPI synchronous 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, QM_SPI_RX_FULL,
QM_SPI_TX_EMPTY }
SPI status.
- enum [qm_spi_frame_format_t](#) { QM_SPI_FRAME_FORMAT_STANDARD = 0x0 }
QM SPI Frame Format.

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, volatile const `qm_spi_async_transfer_t` *const xfer)
Interrupt based transfer on SPI.
- int `qm_spi_irq_update` (const `qm_spi_t` spi, volatile const `qm_spi_async_transfer_t` *const xfer, const `qm_spi_update_t` update)
Update parameters of 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.
- int `qm_spi_save_context` (const `qm_spi_t` spi, `qm_spi_context_t` *const ctx)
Save SPI context.
- int `qm_spi_restore_context` (const `qm_spi_t` spi, const `qm_spi_context_t` *const ctx)
Restore SPI context.

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_format_t`

QM SPI Frame Format.

Enumerator

`QM_SPI_FRAME_FORMAT_STANDARD` Standard SPI mode.

Definition at line 103 of file `qm_spi.h`.

4.21.2.3 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.

4.21.2.4 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.

4.21.2.5 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 overflowed.
- QM_SPI_RX_FULL** Appl. Rx buffer full (slave only).
- QM_SPI_TX_EMPTY** Appl. Tx buffer empty (slave only) .

Definition at line 92 of file qm_spi.h.

4.21.2.6 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.

This function configures the DMA source transfer width according to the currently set SPI frame size. Therefore, whenever the SPI frame is updated (using qm_spi_set_config) this function needs to be called again as the previous source transfer width configuration is no longer valid. Note that if the current frame size lies between 17 and 24 bits, this function fails (returning -EINVAL) as the DMA core cannot handle 3-byte source width transfers with buffers containing 1 padding byte between consecutive frames.

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	<i>dma_channel_id</i>	DMA channel identifier.
in	<i>dma_channel_direction</i>	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 851 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_TYPE_SINGLE, qm_dma_channel_config_t::source_burst_length, qm_dma_channel_config_t::source_transfer_width, and qm_dma_channel_config_t::transfer_type.

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. 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. 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	<i>spi</i>	SPI controller identifier.
in	<i>xfer</i>	Structure containing pre-allocated write and read data buffers and callback functions. This 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 955 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 1094 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 SPI TX FIFO not empty; QM_SPI_IDLE is available for transfer; QM_SPI_RX_OVERFLOW if an RX overflow has occurred.

The user may call this function before performing an SPI transfer in order to guarantee that the SPI interface is available.

Parameters

in	spi	Which SPI to read the status of.
out	status	Current 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 479 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, volatile 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	<i>spi</i>	Which SPI to transfer to / from.
in	<i>xfer</i>	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.

Definition at line 639 of file qm_spi.c.

References qm_spi_reg_t::ctrlr0, qm_spi_reg_t::ctrlr1, QM_SPI_FRAME_FORMAT_STANDARD, qm_spi_irq_update(), 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::rxflr, qm_spi_reg_t::ssienr, qm_spi_async_transfer_t::tx_len, and qm_spi_reg_t::txflr.

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	<i>spi</i>	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.

Definition at line 736 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_reg_t::ssienr, and qm_spi_reg_t::txflr.

4.21.3.7 int qm_spi_irq_update (const qm_spi_t spi, volatile const qm_spi_async_transfer_t *const xfer, const qm_spi_update_t update)

Update parameters of Interrupt based transfer on SPI.

Allow the application to transmit and/or receive more data over the current SPI communication. The application is supposed to call this function only inside the registered callback, once notified from the driver. It is strongly recommended to use this function for slave-based applications only, as slave controllers usually do not know how many frames an external master will send or request before starting the communication. Master controllers should not use this function as it will most likely corrupt the transaction.

Parameters

in	<i>spi</i>	Which SPI to transfer to / from.
in	<i>xfer</i>	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.
in	<i>update</i>	Specify if only RX has to be updated, or only TX or both.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 584 of file qm_spi.c.

References qm_spi_reg_t::imr, QM_SPI_FRAME_FORMAT_STANDARD, QM_SPI_TMOD_EEPROM_READ, QM_SPI_TMOD_RX, QM_SPI_TMOD_TX, qm_spi_async_transfer_t::rx_len, and qm_spi_async_transfer_t::tx_len.

Referenced by qm_spi_irq_transfer().

4.21.3.8 int qm_spi_restore_context (const qm_spi_t *spi*, const qm_spi_context_t *const *ctx*)

Restore SPI context.

Restore the configuration of the specified SPI peripheral after exiting sleep.

Parameters

in	<i>spi</i>	SPI controller identifier.
in	<i>ctx</i>	SPI context structure. 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 1165 of file qm_spi.c.

References qm_spi_reg_t::baudr, qm_spi_context_t::baudr, qm_spi_reg_t::ctrlr0, qm_spi_context_t::ctrlr0, qm_spi_reg_t::ser, and qm_spi_context_t::ser.

4.21.3.9 int qm_spi_save_context (const qm_spi_t *spi*, qm_spi_context_t *const *ctx*)

Save SPI context.

Saves the configuration of the specified SPI peripheral before entering sleep.

Parameters

in	<i>spi</i>	SPI controller identifier.
out	<i>ctx</i>	SPI context structure. 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 1157 of file qm_spi.c.

References qm_spi_reg_t::baudr, qm_spi_context_t::baudr, qm_spi_reg_t::ctrlr0, qm_spi_context_t::ctrlr0, qm_spi_reg_t::ser, and qm_spi_context_t::ser.

4.21.3.10 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, clock divider and data frame size.

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 423 of file qm_spi.c.

References qm_spi_reg_t::baudr, qm_spi_reg_t::ctrlr0, and QM_SPI_TMOD_EEPROM_READ.

4.21.3.11 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 465 of file qm_spi.c.

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 499 of file qm_spi.c.

References [qm_spi_reg_t::ctrlr1](#), [qm_spi_reg_t::imr](#), [QM_SPI_FRAME_FORMAT_STANDARD](#), [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 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.

- int `qm_ss_adc_save_context` (const `qm_ss_adc_t` adc, `qm_ss_adc_context_t` *const ctx)

Save SS ADC context.

- int `qm_ss_adc_restore_context` (const `qm_ss_adc_t` adc, const `qm_ss_adc_context_t` *const ctx)

Restore SS ADC context.

4.22.1 Detailed Description

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

4.22.2 Enumeration Type Documentation

4.22.2.1 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`.

4.22.2.2 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.

4.22.2.3 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.

4.22.2.4 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.

4.22.2.5 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.

4.22.3 Function Documentation

4.22.3.1 `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

<code>in</code>	<code>adc</code>	Which ADC to calibrate.
-----------------	------------------	-------------------------

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<code>Negative</code>	errno for possible error codes.

4.22.3.2 `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

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

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<code>Negative</code>	errno for possible error codes.

Definition at line 564 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`.

4.22.3.3 `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

<code>in</code>	<code>adc</code>	Which ADC to get calibration for.
<code>out</code>	<code>cal</code>	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.

4.22.3.4 `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 479 of file qm_ss_adc.c.

References QM_SS_IO_CREG_MST0_CTRL.

4.22.3.5 `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 638 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.

4.22.3.6 `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	<i>adc</i>	Which ADC to enable.
in	<i>mode</i>	ADC operating mode.
in	<i>callback</i>	Callback called on completion.
in	<i>callback_data</i>	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 405 of file `qm_ss_adc.c`.

References `QM_SS_ADC_MODE_NORM_NO_CAL`, and `QM_SS_IO_CREG_MST0_CTRL`.

4.22.3.7 `int qm_ss_adc_restore_context (const qm_ss_adc_t adc, const qm_ss_adc_context_t *const ctx)`

Restore SS ADC context.

Restore the configuration of the specified ADC peripheral after exiting sleep.

Note: Previous calibration data is not restored with this function, the user may need to recalibrate the ADC. The user will need to set the `ADC_ENA` bit in the ADC Control register as it is initialized to 0.

Parameters

in	<i>adc</i>	SS ADC port index.
in	<i>ctx</i>	SS ADC context structure. 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 712 of file `qm_ss_adc.c`.

References `qm_ss_adc_context_t::adc_ctrl`, `qm_ss_adc_context_t::adc_divseqstat`, `qm_ss_adc_context_t::adc_seq`, `qm_ss_adc_context_t::adc_set`, `QM_SS_ADC_CTRL`, `QM_SS_ADC_DIVSEQSTAT`, `QM_SS_ADC_SEQ`, and `QM_SS_ADC_SET`.

4.22.3.8 `int qm_ss_adc_save_context (const qm_ss_adc_t adc, qm_ss_adc_context_t *const ctx)`

Save SS ADC context.

Save the configuration of the specified ADC peripheral before entering sleep.

Note: Calibration data is not saved with this function. The value of the `ADC_ENA` bit in the ADC Control register is also not saved with this function.

Parameters

in	<i>adc</i>	SS ADC port index.
----	------------	--------------------

out	ctx	SS ADC context structure. 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 685 of file qm_ss_adc.c.

References qm_ss_adc_context_t::adc_ctrl, qm_ss_adc_context_t::adc_divseqstat, qm_ss_adc_context_t::adc_seq, qm_ss_adc_context_t::adc_set, QM_SS_ADC_CTRL, QM_SS_ADC_DIVSEQSTAT, QM_SS_ADC_SEQ, and QM_SS_ADC_SET.

4.22.3.9 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.

4.22.3.10 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.

4.22.3.11 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

<i>in</i>	<i>adc</i>	Which ADC to enable.
<i>in</i>	<i>mode</i>	ADC operating mode.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	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.

4.23 SS GPIO

General Purpose IO for Sensor Subsystem.

Data Structures

- struct `qm_ss_gpio_port_config_t`
SS GPIO port configuration type.

Enumerations

- enum `qm_ss_gpio_state_t` { `QM_SS_GPIO_LOW` = 0, `QM_SS_GPIO_HIGH`, `QM_SS_GPIO_STATE_NUM` }
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.
- int `qm_ss_gpio_save_context` (const `qm_ss_gpio_t` gpio, `qm_ss_gpio_context_t` *const ctx)
Save SS GPIO context.
- int `qm_ss_gpio_restore_context` (const `qm_ss_gpio_t` gpio, const `qm_ss_gpio_context_t` *const ctx)
Restore SS GPIO context.

4.23.1 Detailed Description

General Purpose IO for Sensor Subsystem.

4.23.2 Enumeration Type Documentation

4.23.2.1 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.
- `QM_SS_GPIO_STATE_NUM`** Number of states.

Definition at line 21 of file `qm_ss_gpio.h`.

4.23.3 Function Documentation

4.23.3.1 `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	<i>gpio</i>	SS GPIO port index.
in	<i>pin</i>	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 111 of file `qm_ss_gpio.c`.

4.23.3.2 `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.

Parameters

in	<i>gpio</i>	SS GPIO port index.
in	<i>pin</i>	Pin of SS GPIO port to read.
out	<i>state</i>	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 84 of file `qm_ss_gpio.c`.

4.23.3.3 `int qm_ss_gpio_read_port (const qm_ss_gpio_t gpio, uint32_t *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	<i>gpio</i>	SS GPIO port index.
out	<i>port</i>	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 138 of file qm_ss_gpio.c.

4.23.3.4 `int qm_ss_gpio_restore_context (const qm_ss_gpio_t gpio, const qm_ss_gpio_context_t *const ctx)`

Restore SS GPIO context.

Restore the configuration of the specified GPIO peripheral after exiting sleep.

Parameters

in	<i>gpio</i>	SS GPIO port index.
in	<i>ctx</i>	SS GPIO context structure. 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 184 of file qm_ss_gpio.c.

References `qm_ss_gpio_context_t::gpio_debounce`, `qm_ss_gpio_context_t::gpio_int_polarity`, `qm_ss_gpio_context_t::gpio_inten`, `qm_ss_gpio_context_t::gpio_intmask`, `qm_ss_gpio_context_t::gpio_inttype_level`, `qm_ss_gpio_context_t::gpio_ls_sync`, `qm_ss_gpio_context_t::gpio_swporta_ddr`, and `qm_ss_gpio_context_t::gpio_swporta_dr`.

4.23.3.5 `int qm_ss_gpio_save_context (const qm_ss_gpio_t gpio, qm_ss_gpio_context_t *const ctx)`

Save SS GPIO context.

Save the configuration of the specified GPIO peripheral before entering sleep.

Parameters

in	<i>gpio</i>	SS GPIO port index.
out	<i>ctx</i>	SS GPIO context structure. 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 158 of file qm_ss_gpio.c.

References `qm_ss_gpio_context_t::gpio_debounce`, `qm_ss_gpio_context_t::gpio_int_polarity`, `qm_ss_gpio_context_t::gpio_inten`, `qm_ss_gpio_context_t::gpio_intmask`, `qm_ss_gpio_context_t::gpio_inttype_level`, `qm_ss_gpio_context_t::gpio_ls_sync`, `qm_ss_gpio_context_t::gpio_swporta_ddr`, and `qm_ss_gpio_context_t::gpio_swporta_dr`.

4.23.3.6 `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.

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	<i>gpio</i>	SS GPIO port index to configure.
in	<i>cfg</i>	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_bothedge*, *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*.

4.23.3.7 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	<i>gpio</i>	SS GPIO port index.
in	<i>pin</i>	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 98 of file *qm_ss_gpio.c*.

4.23.3.8 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	<i>gpio</i>	GPIO port index.
in	<i>pin</i>	Pin of GPIO port to update.
in	<i>state</i>	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_ss_gpio.c.

References QM_SS_GPIO_STATE_NUM.

4.23.3.9 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 148 of file qm_ss_gpio.c.

4.24 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_FAST_PLUS](#) }
- QM SS I2C speed type.*
- enum [qm_ss_i2c_status_t](#) {
[QM_SS_I2C_IDLE](#) = 0, [QM_SS_I2C_TX_ABRT_7B_ADDR_NOACK](#) = BIT(0), [QM_SS_I2C_TX_ABRT_10-ADDR1_NOACK](#) = BIT(1), [QM_SS_I2C_TX_ABRT_10ADDR2_NOACK](#) = BIT(2),
[QM_SS_I2C_TX_ABRT_TXDATA_NOACK](#) = BIT(3), [QM_SS_I2C_TX_ABRT_GCALL_NOACK](#) = BIT(4) ,
[QM_SS_I2C_TX_ABRT_SBYTE_ACKDET](#) = BIT(7), [QM_SS_I2C_TX_ABRT_NORSTRT](#) = BIT(9),
[QM_SS_I2C_TX_ABRT_10B_RD_NORSTRT](#) = BIT(10), [QM_SS_I2C_TX_ABRT_MASTER_DIS](#) = BIT(11),
[QM_SS_I2C_TX_ARB_LOST](#) = BIT(12) , [QM_SS_I2C_TX_ABRT_SLV_ARBLOST](#) = BIT(14),
[QM_SS_I2C_TX_ABRT_SLVRD_INTX](#) = BIT(15), [QM_SS_I2C_TX_ABRT_USER_ABRT](#) = BIT(16), [QM_S-I2C_BUSY](#) = BIT(17), [QM_SS_I2C_TX_ABORT](#) = BIT(18),
[QM_SS_I2C_TX_OVER](#) = BIT(19), [QM_SS_I2C_RX_OVER](#) = BIT(20), [QM_SS_I2C_RX_UNDER](#) = BIT(21)
 }
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.
- int [qm_ss_i2c_save_context](#) (const [qm_ss_i2c_t](#) i2c, [qm_ss_i2c_context_t](#) *const ctx)

Save SS I2C context.

- int `qm_ss_i2c_restore_context` (const `qm_ss_i2c_t` i2c, const `qm_ss_i2c_context_t` *const ctx)

Restore SS I2C context.

4.24.1 Detailed Description

I2C driver for Sensor Subsystem.

4.24.2 Enumeration Type Documentation

4.24.2.1 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`.

4.24.2.2 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).

`QM_SS_I2C_SPEED_FAST_PLUS` Fast plus mode (1 Mbps).

Definition at line 45 of file `qm_ss_i2c.h`.

4.24.2.3 enum `qm_ss_i2c_status_t`

QM SS I2C status type.

Enumerator

`QM_SS_I2C_IDLE` Controller idle.

`QM_SS_I2C_TX_ABRT_7B_ADDR_NOACK` 7-bit address noack.

`QM_SS_I2C_TX_ABRT_10ADDR1_NOACK` 10-bit address noack.

`QM_SS_I2C_TX_ABRT_10ADDR2_NOACK` 10-bit address noack.

`QM_SS_I2C_TX_ABRT_TXDATA_NOACK` Tx data noack.

`QM_SS_I2C_TX_ABRT_GCALL_NOACK` General call noack. General call configured as read.

`QM_SS_I2C_TX_ABRT_SBYTE_ACKDET` Start ACK.

`QM_SS_I2C_TX_ABRT_NORSTR` Restart disabled.

`QM_SS_I2C_TX_ABRT_10B_RD_NORSTR` Restart disabled.

`QM_SS_I2C_TX_ABRT_MASTER_DIS` Master disabled.

`QM_SS_I2C_TX_ARB_LOST` Master lost arbitration. Slave flush tx FIFO.

`QM_SS_I2C_TX_ABRT_SLV_ARBLOST` Slave lost bus.

`QM_SS_I2C_TX_ABRT_SLVRD_INTX` Slave read completion.

`QM_SS_I2C_TX_ABRT_USER_ABRT` User abort.

QM_SS_I2C_BUSY Controller busy.
QM_SS_I2C_TX_ABORT Tx abort.
QM_SS_I2C_TX_OVER Tx overflow.
QM_SS_I2C_RX_OVER Rx overflow.
QM_SS_I2C_RX_UNDER Rx underflow.

Definition at line 56 of file qm_ss_i2c.h.

4.24.3 Function Documentation

4.24.3.1 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 505 of file qm_ss_i2c.c.

References QM_SS_I2C_BUSY, and QM_SS_I2C_IDLE.

Referenced by qm_ss_i2c_master_read(), and qm_ss_i2c_master_write().

4.24.3.2 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_SS_I2C_TX_ABORT_USER_ABORT.

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 761 of file qm_ss_i2c.c.

4.24.3.3 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	<i>i2c</i>	Which I2C to transfer from.
in	<i>xfer</i>	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	<i>slave_addr</i>	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 675 of file qm_ss_i2c.c.

References qm_ss_i2c_transfer_t::rx_len.

4.24.3.4 `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	<i>i2c</i>	Which I2C to read from.
in	<i>slave_addr</i>	Address of slave device to read from.
out	<i>data</i>	Pre-allocated buffer to populate with data. This must not be NULL.
in	<i>len</i>	Length of data to read from slave.
in	<i>stop</i>	Generate a STOP condition at the end of rx.
out	<i>status</i>	Get I2C status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 598 of file qm_ss_i2c.c.

References qm_ss_i2c_get_status().

4.24.3.5 `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	<i>i2c</i>	Which I2C to write to.
----	------------	------------------------

in	<i>slave_addr</i>	Address of slave to write to.
in	<i>data</i>	Pre-allocated buffer of data to write. This must not be NULL.
in	<i>len</i>	Length of data to write.
in	<i>stop</i>	Generate a STOP condition at the end of tx.
out	<i>status</i>	Get I2C status.

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_ss_i2c.c`.

References `qm_ss_i2c_get_status()`.

4.24.3.6 `int qm_ss_i2c_restore_context (const qm_ss_i2c_t i2c, const qm_ss_i2c_context_t *const ctx)`

Restore SS I2C context.

Restore the configuration of the specified SS I2C peripheral after exiting sleep.

Parameters

in	<i>i2c</i>	I2C port index.
in	<i>ctx</i>	I2C context structure. 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 799 of file `qm_ss_i2c.c`.

4.24.3.7 `int qm_ss_i2c_save_context (const qm_ss_i2c_t i2c, qm_ss_i2c_context_t *const ctx)`

Save SS I2C context.

Saves the configuration of the specified SS I2C peripheral before entering sleep. The slave operations need to be disabled before being able to save the context as otherwise we could be interrupted by an I2C transfer while saving registers.

Parameters

in	<i>i2c</i>	I2C port index.
out	<i>ctx</i>	I2C context structure. 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 782 of file qm_ss_i2c.c.

4.24.3.8 `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	<i>i2c</i>	Which I2C to set the configuration of.
in	<i>cfg</i>	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 347 of file qm_ss_i2c.c.

References `qm_ss_i2c_config_t::address_mode`, `QM_SS_I2C_SPEED_FAST`, `QM_SS_I2C_SPEED_FAST_PLUS`, `QM_SS_I2C_SPEED_STD`, and `qm_ss_i2c_config_t::speed`.

4.24.3.9 `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	<i>i2c</i>	I2C index.
in	<i>speed</i>	Bus speed (Standard or Fast. Fast includes Fast+ mode).
in	<i>lo_cnt</i>	SCL low count.
in	<i>hi_cnt</i>	SCL high count.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 450 of file qm_ss_i2c.c.

References `QM_SS_I2C_SPEED_FAST`, `QM_SS_I2C_SPEED_FAST_PLUS`, and `QM_SS_I2C_SPEED_STD`.

4.25 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.

4.25.1 Detailed Description

Interrupt driver for Sensor Subsystem.

4.25.2 Function Documentation

4.25.2.1 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	<i>vector</i>	Interrupt Vector number.
in	<i>isr</i>	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\(\)](#).

4.25.2.2 void [qm_ss_irq_mask](#) (uint32_t irq)

Mask a given interrupt line.

Parameters

in	<i>irq</i>	Which IRQ to mask.
----	------------	--------------------

Definition at line 30 of file [qm_ss_interrupt.c](#).

Referenced by [qm_irq_mask\(\)](#), and [qm_ss_irq_request\(\)](#).

4.25.2.3 void [qm_ss_irq_request](#) (uint32_t irq, [qm_ss_isr_t](#) isr)

Request a given IRQ and register ISR to interrupt vector.

Parameters

<i>in</i>	<i>irq</i>	IRQ number.
<i>in</i>	<i>isr</i>	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()`.

4.25.2.4 void qm_ss_irq_unmask (uint32_t *irq*)

Unmask a given interrupt line.

Parameters

<i>in</i>	<i>irq</i>	Which IRQ to unmask.
-----------	------------	----------------------

Definition at line 36 of file `qm_ss_interrupt.c`.

Referenced by `qm_irq_unmask()`, and `qm_ss_irq_request()`.

4.26 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_error_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_0_isr)
ISR for GPIO 0 error interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_gpio_1_isr)
ISR for GPIO 1 error interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_0_error_isr)
ISR for I2C 0 error interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_0_rx_avail_isr)
ISR for I2C 0 RX data available interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_0_tx_req_isr)
ISR for I2C 0 TX data requested interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_0_stop_det_isr)
ISR for I2C 0 STOP detected interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_1_error_isr)
ISR for I2C 1 error interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_1_rx_avail_isr)
ISR for I2C 1 RX data available interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_1_tx_req_isr)
ISR for I2C 1 TX data requested interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_i2c_1_stop_det_isr)
ISR for I2C 1 STOP detected interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_spi_0_error_isr)
ISR for SPI 0 error interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_spi_1_error_isr)
ISR for SPI 1 error interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_spi_0_tx_req_isr)
ISR for SPI 0 TX data requested interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_spi_1_tx_req_isr)
ISR for SPI 1 TX data requested interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_spi_0_rx_avail_isr)
ISR for SPI 0 RX data available interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_spi_1_rx_avail_isr)
ISR for SPI 1 data available interrupt.
- [QM_ISR_DECLARE](#) (qm_ss_timer_0_isr)
ISR for SS Timer 0 interrupt.

4.26.1 Detailed Description

Sensor Subsystem Interrupt Service Routines.

4.26.2 Function Documentation

4.26.2.1 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_0_INT, 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.

4.26.2.2 QM_ISR_DECLARE (qm_ss_adc_0_error_isr)

ISR for ADC error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_ADC_0_ERROR_INT,  
qm_ss_adc_0_error_isr);
```

if IRQ based conversions are used.

Definition at line 203 of file qm_ss_adc.c.

References QM_SS_ADC_0.

4.26.2.3 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_0_CAL_INT, 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.

4.26.2.4 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_0_PWR_INT, 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.

4.26.2.5 QM_ISR_DECLARE (qm_ss_gpio_0_isr)

ISR for GPIO 0 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_GPIO_0_INT, qm_ss_gpio_0_isr);
```

if IRQ based transfers are used.

Definition at line 27 of file qm_ss_gpio.c.

4.26.2.6 QM_ISR_DECLARE (qm_ss_gpio_1_isr)

ISR for GPIO 1 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_GPIO_1_INT, qm_ss_gpio_1_isr);
```

if IRQ based transfers are used.

Definition at line 32 of file qm_ss_gpio.c.

4.26.2.7 QM_ISR_DECLARE (qm_ss_i2c_0_error_isr)

ISR for I2C 0 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_0_ERROR_INT, qm_ss_i2c_0_error_isr);
```

if IRQ based transfers are used.

Definition at line 294 of file qm_ss_i2c.c.

4.26.2.8 QM_ISR_DECLARE (qm_ss_i2c_0_rx_avail_isr)

ISR for I2C 0 RX data available interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_0_RX_AVAIL_INT, qm_ss_i2c_0_rx_avail_isr);
```

if IRQ based transfers are used.

Definition at line 299 of file qm_ss_i2c.c.

4.26.2.9 QM_ISR_DECLARE (qm_ss_i2c_0_tx_req_isr)

ISR for I2C 0 TX data requested interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_0_TX_REQ_INT, qm_ss_i2c_0_tx_req_isr);
```

if IRQ based transfers are used.

Definition at line 304 of file qm_ss_i2c.c.

4.26.2.10 QM_ISR_DECLARE (qm_ss_i2c_0_stop_det_isr)

ISR for I2C 0 STOP detected interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_0_STOP_DET_INT,
qm_ss_i2c_0_stop_det_isr);
```

if IRQ based transfers are used.

Definition at line 309 of file qm_ss_i2c.c.

4.26.2.11 QM_ISR_DECLARE (qm_ss_i2c_1_error_isr)

ISR for I2C 1 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_1_ERROR_INT, qm_ss_i2c_1_error_isr);
```

if IRQ based transfers are used.

Definition at line 314 of file qm_ss_i2c.c.

4.26.2.12 QM_ISR_DECLARE (qm_ss_i2c_1_rx_avail_isr)

ISR for I2C 1 RX data available interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_1_RX_AVAIL_INT,
qm_ss_i2c_1_rx_avail_isr);
```

if IRQ based transfers are used.

Definition at line 319 of file qm_ss_i2c.c.

4.26.2.13 QM_ISR_DECLARE (qm_ss_i2c_1_tx_req_isr)

ISR for I2C 1 TX data requested interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_1_TX_REQ_INT, qm_ss_i2c_1_tx_req_isr);
```

if IRQ based transfers are used.

Definition at line 324 of file qm_ss_i2c.c.

4.26.2.14 QM_ISR_DECLARE (qm_ss_i2c_1_stop_det_isr)

ISR for I2C 1 STOP detected interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_I2C_1_STOP_DET_INT,
qm_ss_i2c_1_stop_det_isr);
```

if IRQ based transfers are used.

Definition at line 329 of file qm_ss_i2c.c.

4.26.2.15 QM_ISR_DECLARE (qm_ss_spi_0_error_isr)

ISR for SPI 0 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_0_ERROR_INT,
qm_ss_spi_0_error_isr);
```

if IRQ based transfers are used.

Definition at line 413 of file `qm_ss_spi.c`.

References `QM_SS_SPI_0`.

4.26.2.16 `QM_ISR_DECLARE (qm_ss_spi_1_error_isr)`

ISR for SPI 1 error interrupt.

This function needs to be registered with

```
qm_ss_irq_request(QM_SS_IRQ_SPI_1_ERROR_INT,
qm_ss_spi_1_error_isr);
```

if IRQ based transfers are used.

Definition at line 417 of file `qm_ss_spi.c`.

References `QM_SS_SPI_1`.

4.26.2.17 `QM_ISR_DECLARE (qm_ss_spi_0_tx_req_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_INT,
qm_ss_spi_0_tx_req_isr);
```

if IRQ based transfers are used.

Definition at line 429 of file `qm_ss_spi.c`.

References `QM_SS_SPI_0`.

4.26.2.18 `QM_ISR_DECLARE (qm_ss_spi_1_tx_req_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_INT,
qm_ss_spi_1_tx_req_isr);
```

if IRQ based transfers are used.

Definition at line 433 of file `qm_ss_spi.c`.

References `QM_SS_SPI_1`.

4.26.2.19 `QM_ISR_DECLARE (qm_ss_spi_0_rx_avail_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_INT,
qm_ss_spi_0_rx_avail_isr);
```

if IRQ based transfers are used.

Definition at line 421 of file `qm_ss_spi.c`.

References `QM_SS_SPI_0`.

4.26.2.20 QM_ISR_DECLARE (qm_ss_spi_1_rx_avail_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_INT,  
qm_ss_spi_1_rx_avail_isr);
```

if IRQ based transfers are used.

Definition at line 425 of file qm_ss_spi.c.

References QM_SS_SPI_1.

4.26.2.21 QM_ISR_DECLARE (qm_ss_timer_0_isr)

ISR for SS Timer 0 interrupt.

This function needs to be registered with

```
qm_ss_int_vector_request(QM_ARC_TIMER_0_INT, qm_ss_timer_0_isr);
```

Definition at line 24 of file qm_ss_timer.c.

4.27 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 asynchronous transfer type.
- struct [qm_ss_spi_transfer_t](#)
SPI synchronous 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_SS_SPI_FRAME_SIZE_17_BIT, QM_SS_SPI_FRAME_SIZE_18_BIT, QM_SS_SPI_FRAME_SIZE_19_BIT,
QM_SS_SPI_FRAME_SIZE_20_BIT, QM_SS_SPI_FRAME_SIZE_21_BIT, QM_SS_SPI_FRAME_SIZE_22_BIT, QM_SS_SPI_FRAME_SIZE_23_BIT,
QM_SS_SPI_FRAME_SIZE_24_BIT, QM_SS_SPI_FRAME_SIZE_25_BIT, QM_SS_SPI_FRAME_SIZE_26_BIT, QM_SS_SPI_FRAME_SIZE_27_BIT,
QM_SS_SPI_FRAME_SIZE_28_BIT, QM_SS_SPI_FRAME_SIZE_29_BIT, QM_SS_SPI_FRAME_SIZE_30_BIT, QM_SS_SPI_FRAME_SIZE_31_BIT,
QM_SS_SPI_FRAME_SIZE_32_BIT }
QM Sensor 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 = 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, QM_SS_SPI_TX_OVERFLOW, QM_SS_SPI_RX_UNDERFLOW }
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 an interrupt based SPI transfer.
- int `qm_ss_spi_irq_transfer_terminate` (const `qm_ss_spi_t` spi)
Terminate SPI IRQ transfer.
- int `qm_ss_spi_save_context` (const `qm_ss_spi_t` spi, `qm_ss_spi_context_t` *const ctx)
Save SS SPI context.
- int `qm_ss_spi_restore_context` (const `qm_ss_spi_t` spi, const `qm_ss_spi_context_t` *const ctx)
Restore SS SPI context.

4.27.1 Detailed Description

SPI peripheral driver for Sensor Subsystem.

4.27.2 Enumeration Type Documentation

4.27.2.1 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 95 of file `qm_ss_spi.h`.

4.27.2.2 enum `qm_ss_spi_frame_size_t`

QM Sensor 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.
QM_SS_SPI_FRAME_SIZE_17_BIT 17 bit frame.
QM_SS_SPI_FRAME_SIZE_18_BIT 18 bit frame.
QM_SS_SPI_FRAME_SIZE_19_BIT 19 bit frame.
QM_SS_SPI_FRAME_SIZE_20_BIT 20 bit frame.
QM_SS_SPI_FRAME_SIZE_21_BIT 21 bit frame.
QM_SS_SPI_FRAME_SIZE_22_BIT 22 bit frame.
QM_SS_SPI_FRAME_SIZE_23_BIT 23 bit frame.
QM_SS_SPI_FRAME_SIZE_24_BIT 24 bit frame.
QM_SS_SPI_FRAME_SIZE_25_BIT 25 bit frame.
QM_SS_SPI_FRAME_SIZE_26_BIT 26 bit frame.
QM_SS_SPI_FRAME_SIZE_27_BIT 27 bit frame.
QM_SS_SPI_FRAME_SIZE_28_BIT 28 bit frame.
QM_SS_SPI_FRAME_SIZE_29_BIT 29 bit frame.
QM_SS_SPI_FRAME_SIZE_30_BIT 30 bit frame.
QM_SS_SPI_FRAME_SIZE_31_BIT 31 bit frame.
QM_SS_SPI_FRAME_SIZE_32_BIT 32 bit frame.

Definition at line 20 of file `qm_ss_spi.h`.

4.27.2.3 enum `qm_ss_spi_slave_select_t`

SPI Slave select type.

Slave selects can be combined by logical OR.

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 107 of file `qm_ss_spi.h`.

4.27.2.4 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 overflowed.
QM_SS_SPI_TX_OVERFLOW TX transfer has overflowed.
QM_SS_SPI_RX_UNDERFLOW RX transfer has underflowed.

Definition at line 118 of file `qm_ss_spi.h`.

4.27.2.5 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 57 of file qm_ss_spi.h.

4.27.3 Function Documentation

4.27.3.1 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 111 of file qm_ss_spi.c.

References QM_SS_SPI_BUSY, QM_SS_SPI_IDLE, and QM_SS_SPI_SR.

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

Initiate an 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 206 of file `qm_ss_spi.c`.

References `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`.

4.27.3.3 int qm_ss_spi_irq_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 257 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_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`.

4.27.3.4 int qm_ss_spi_restore_context (const qm_ss_spi_t spi, const qm_ss_spi_context_t *const ctx)

Restore SS SPI context.

Restore the configuration of the specified SS SPI peripheral after exiting sleep.

Parameters

in	spi	SPI controller identifier.
in	ctx	SPI context structure. 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 478 of file `qm_ss_spi.c`.

References `QM_SS_SPI_CTRL`, `QM_SS_SPI_SPIEN`, `QM_SS_SPI_TIMING`, `qm_ss_spi_context_t::spi_ctrl`, `qm_ss_spi_context_t::spi_spien`, and `qm_ss_spi_context_t::spi_timing`.

4.27.3.5 int qm_ss_spi_save_context (const qm_ss_spi_t spi, qm_ss_spi_context_t *const ctx)

Save SS SPI context.

Saves the configuration of the specified SS SPI peripheral before entering sleep.

Parameters

in	<i>spi</i>	SPI controller identifier.
out	<i>ctx</i>	SPI context structure. 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 469 of file `qm_ss_spi.c`.

References `QM_SS_SPI_CTRL`, `QM_SS_SPI_SPIEN`, `QM_SS_SPI_TIMING`, `qm_ss_spi_context_t::spi_ctrl`, `qm_ss_spi_context_t::spi_spien`, and `qm_ss_spi_context_t::spi_timing`.

4.27.3.6 `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, clock divider and data frame size.

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

Parameters

in	<i>spi</i>	SPI module identifier.
in	<i>cfg</i>	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 65 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_SPIEN`, and `qm_ss_spi_config_t::transfer_mode`.

4.27.3.7 `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	<i>spi</i>	SPI module identifier.
in	<i>ss</i>	Select lines to enable when performing transfers.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 96 of file qm_ss_spi.c.

References QM_SS_SPI_SR.

4.27.3.8 `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	<i>spi</i>	SPI module identifier.
in	<i>xfer</i>	Structure containing transfer information. This must not be NULL.
out	<i>status</i>	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

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 126 of file qm_ss_spi.c.

References 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.

4.28 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.

4.28.1 Detailed Description

Timer driver for Sensor Subsystem.

4.28.2 Function Documentation

4.28.2.1 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	<i>timer</i>	Which SS timer to get the count of.
out	<i>count</i>	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](#).

4.28.2.2 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	<i>timer</i>	Which SS timer to set the count of.
in	<i>count</i>	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.

4.28.2.3 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	<i>timer</i>	Which SS timer to configure.
in	<i>cfg</i>	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.

4.29 UART

UART peripheral driver.

Data Structures

- struct `qm_uart_config_t`
UART configuration structure 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, const 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.
- int [qm_uart_save_context](#) (const [qm_uart_t](#) uart, [qm_uart_context_t](#) *const ctx)
Save UART context.
- int [qm_uart_restore_context](#) (const [qm_uart_t](#) uart, const [qm_uart_context_t](#) *const ctx)
Restore UART context.

4.29.1 Detailed Description

UART peripheral driver.

4.29.2 Enumeration Type Documentation

4.29.2.1 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 22 of file qm_uart.h.

4.29.2.2 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 52 of file qm_uart.h.

4.29.3 Function Documentation

4.29.3.1 `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	<i>uart</i>	UART identifier.
in	<i>dma_ctrl_id</i>	DMA controller identifier.
in	<i>dma_channel_id</i>	DMA channel identifier.
in	<i>dma_channel_direction</i>	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 530 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_TYPE_SINGLE`, `qm_dma_channel_config_t::source_burst_length`, `qm_dma_channel_config_t::source_transfer_width`, and `qm_dma_channel_config_t::transfer_type`.

4.29.3.2 `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	<i>uart</i>	UART identifier.
in	<i>xfer</i>	Structure containing a pre-allocated read buffer 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 625 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`.

4.29.3.3 `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	<i>uart</i>	UART identifier.
----	-------------	------------------

Returns

Standard `errno` return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 668 of file `qm_uart.c`.

References `qm_dma_transfer_terminate()`.

4.29.3.4 `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	<i>uart</i>	UART identifier.
in	<i>xfer</i>	Structure containing a pre-allocated write buffer 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 591 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`.

4.29.3.5 `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	<i>uart</i>	UART identifier.
----	-------------	------------------

Returns

Standard `errno` return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 658 of file `qm_uart.c`.

References `qm_dma_transfer_terminate()`.

4.29.3.6 `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.

The user may call this function before performing an UART transfer in order to guarantee that the UART interface is available.

Parameters

in	<i>uart</i>	Which UART to read the status of.
out	<i>status</i>	Current UART 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 253 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`.

4.29.3.7 `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	<i>uart</i>	UART identifier.
in	<i>xfer</i>	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 389 of file `qm_uart.c`.

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

4.29.3.8 `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	<i>uart</i>	UART identifier.
----	-------------	------------------

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_uart.c.

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

4.29.3.9 `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 identifier.
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 368 of file qm_uart.c.

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

4.29.3.10 `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 identifier.
----	------	------------------

Returns

Standard `errno` return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 412 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`.

4.29.3.11 `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	<i>uart</i>	UART identifier.
out	<i>data</i>	Data to read from UART. This must not be NULL.
out	<i>status</i>	UART specific status.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 296 of file qm_uart.c.

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

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

UART character data read.

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

Parameters

in	<i>uart</i>	UART identifier.
out	<i>data</i>	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 331 of file qm_uart.c.

References qm_uart_reg_t::rbr_thr_dll.

4.29.3.13 int qm_uart_restore_context (const qm_uart_t *uart*, const qm_uart_context_t *const *ctx*)

Restore UART context.

Restore the configuration of the specified UART peripheral after exiting sleep.

FIFO control register cannot be read back, the default configuration is applied for this register. Application will need to restore its own parameters.

Parameters

in	<i>uart</i>	UART port index.
in	<i>ctx</i>	UART context structure. 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 742 of file `qm_uart.c`.

References `qm_uart_reg_t::dlf`, `qm_uart_context_t::dlf`, `qm_uart_context_t::dlh`, `qm_uart_context_t::dll`, `qm_uart_reg_t::htx`, `qm_uart_context_t::htx`, `qm_uart_context_t::ier`, `qm_uart_reg_t::ier_dlh`, `qm_uart_reg_t::iir_fcr`, `qm_uart_reg_t::lcr`, `qm_uart_context_t::lcr`, `qm_uart_reg_t::mcr`, `qm_uart_context_t::mcr`, `qm_uart_reg_t::rbr_thr_dll`, `qm_uart_reg_t::scr`, and `qm_uart_context_t::scr`.

4.29.3.14 `int qm_uart_save_context (const qm_uart_t uart, qm_uart_context_t *const ctx)`

Save UART context.

Saves the configuration of the specified UART peripheral before entering sleep.

Parameters

in	<i>uart</i>	UART port index.
out	<i>ctx</i>	UART context structure. 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 734 of file `qm_uart.c`.

References `qm_uart_reg_t::dlf`, `qm_uart_context_t::dlf`, `qm_uart_context_t::dlh`, `qm_uart_context_t::dll`, `qm_uart_reg_t::htx`, `qm_uart_context_t::htx`, `qm_uart_context_t::ier`, `qm_uart_reg_t::ier_dlh`, `qm_uart_reg_t::lcr`, `qm_uart_context_t::lcr`, `qm_uart_reg_t::mcr`, `qm_uart_context_t::mcr`, `qm_uart_reg_t::rbr_thr_dll`, `qm_uart_reg_t::scr`, and `qm_uart_context_t::scr`.

4.29.3.15 `int qm_uart_set_config (const qm_uart_t uart, const qm_uart_config_t *const cfg)`

Set UART configuration.

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

Parameters

in	<i>uart</i>	Which UART module to configure.
in	<i>cfg</i>	New configuration for UART. 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::lsr`, `qm_uart_reg_t::mcr`, and `qm_uart_reg_t::rbr_thr_dll`.

4.29.3.16 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 identifier.
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 280 of file qm_uart.c.

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

4.29.3.17 int qm_uart_write_buffer (const qm_uart_t uart, const uint8_t *const data, const 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 controller identifier
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 342 of file qm_uart.c.

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

4.29.3.18 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 identifier.
in	data	Data to write to UART.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 320 of file `qm_uart.c`.

References `qm_uart_reg_t::rbr_thr_dll`.

4.30 USB

USB peripheral driver for Quark Microcontrollers.

Data Structures

- struct `qm_usb_ep_config_t`
USB Endpoint Configuration.

Typedefs

- typedef void(* `qm_usb_status_callback_t`)(void *data, int error, `qm_usb_status_t` status)
Callback function signature for the device status.

Enumerations

- enum `qm_usb_status_t` {
`QM_USB_RESET`, `QM_USB_CONNECTED`, `QM_USB_CONFIGURED`, `QM_USB_DISCONNECTED`,
`QM_USB_SUSPEND`, `QM_USB_RESUME` }
USB Driver Status Codes.
- enum `qm_usb_ep_status_t` { `QM_USB_EP_SETUP`, `QM_USB_EP_DATA_OUT`, `QM_USB_EP_DATA_IN` }
USB Endpoint Callback Status Codes.
- enum `qm_usb_ep_type_t` { `QM_USB_EP_CONTROL` = 0, `QM_USB_EP_BULK` = 2, `QM_USB_EP_INTERRUPT` = 3 }
USB Endpoint type.

Functions

- int `qm_usb_attach` (const `qm_usb_t` usb)
Attach the USB device.
- int `qm_usb_detach` (const `qm_usb_t` usb)
Detach the USB device.
- int `qm_usb_reset` (const `qm_usb_t` usb)
Reset the USB device controller back to it's initial state.
- int `qm_usb_set_address` (const `qm_usb_t` usb, const uint8_t addr)
Set USB device address.
- int `qm_usb_set_status_callback` (const `qm_usb_t` usb, const `qm_usb_status_callback_t` cb)
Set USB device controller status callback.
- int `qm_usb_ep_set_config` (const `qm_usb_t` usb, const `qm_usb_ep_config_t` *const cfg)
Configure endpoint.
- int `qm_usb_ep_set_stall_state` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep, const bool is_stalled)
Set / Clear stall condition for the selected endpoint.
- int `qm_usb_ep_is_stalled` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep, bool *const stalled)
Check stall condition for the selected endpoint.
- int `qm_usb_ep_halt` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep)
Halt the selected endpoint.
- int `qm_usb_ep_enable` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep)
Enable the selected endpoint.
- int `qm_usb_ep_disable` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep)
Disable the selected endpoint.
- int `qm_usb_ep_flush` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep)

Flush the selected IN endpoint TX FIFO.

- int `qm_usb_ep_write` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep, const `uint8_t *const data`, const `uint32_t len`, `uint32_t *const ret_bytes`)

Write data to the specified IN endpoint.

- int `qm_usb_ep_read` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep, `uint8_t *const data`, const `uint32_t max_len`, `uint32_t *const read_bytes`)

Read data from OUT endpoint.

- int `qm_usb_ep_get_bytes_read` (const `qm_usb_t` usb, const `qm_usb_ep_idx_t` ep, `uint32_t *const read_bytes`)

Check how many bytes are available on OUT endpoint.

4.30.1 Detailed Description

USB peripheral driver for Quark Microcontrollers.

4.30.2 Typedef Documentation

4.30.2.1 typedef void(* qm_usb_status_callback_t)(void *data, int error, qm_usb_status_t status)

Callback function signature for the device status.

Parameters

in	<i>data</i>	The callback user data.
in	<i>error</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	USB Controller Driver status.

Definition at line 82 of file `qm_usb.h`.

4.30.3 Enumeration Type Documentation

4.30.3.1 enum qm_usb_ep_status_t

USB Endpoint Callback Status Codes.

Enumerator

QM_USB_EP_SETUP SETUP received.

QM_USB_EP_DATA_OUT Out transaction on this EP.

QM_USB_EP_DATA_IN In transaction on this EP.

Definition at line 33 of file `qm_usb.h`.

4.30.3.2 enum qm_usb_ep_type_t

USB Endpoint type.

Enumerator

QM_USB_EP_CONTROL Control endpoint.

QM_USB_EP_BULK Bulk type endpoint.

QM_USB_EP_INTERRUPT Interrupt type endpoint.

Definition at line 42 of file `qm_usb.h`.

4.30.3.3 enum qm_usb_status_t

USB Driver Status Codes.

Enumerator

QM_USB_RESET USB reset.

QM_USB_CONNECTED USB connection ready and enumeration done.

QM_USB_CONFIGURED USB configuration completed.

QM_USB_DISCONNECTED USB connection lost.

QM_USB_SUSPEND USB connection suspended by the HOST.

QM_USB_RESUME USB connection resumed by the HOST.

Definition at line 21 of file qm_usb.h.

4.30.4 Function Documentation

4.30.4.1 int qm_usb_attach (const qm_usb_t usb)

Attach the USB device.

Upon success, the USB PLL is enabled, and the USB device is now capable of transmitting and receiving on the USB bus and of generating interrupts.

Parameters

in	usb	Which USB module to perform action.
----	-----	-------------------------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 416 of file qm_usb.c.

References `clk_sys_usb_enable()`.

4.30.4.2 int qm_usb_detach (const qm_usb_t usb)

Detach the USB device.

Upon success, the USB hardware PLL is powered down and USB communication is disabled.

Parameters

in	usb	Which USB module to perform action.
----	-----	-------------------------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 458 of file qm_usb.c.

References `clk_sys_usb_disable()`.

4.30.4.3 `int qm_usb_ep_disable (const qm_usb_t usb, const qm_usb_ep_idx_t ep)`

Disable the selected endpoint.

Parameters

in	<i>usb</i>	Which USB module to perform action.
in	<i>ep</i>	Endpoint index corresponding to the one listed in the device configuration table.

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_usb.c.

4.30.4.4 `int qm_usb_ep_enable (const qm_usb_t usb, const qm_usb_ep_idx_t ep)`

Enable the selected endpoint.

Parameters

in	<i>usb</i>	Which USB module to perform action.
in	<i>ep</i>	Endpoint index corresponding to the one listed in the device configuration table.

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_usb.c.

4.30.4.5 `int qm_usb_ep_flush (const qm_usb_t usb, const qm_usb_ep_idx_t ep)`

Flush the selected IN endpoint TX FIFO.

RX FIFO is global and cannot be flushed per endpoint. Thus, this function only applies to endpoints of direction IN.

Parameters

in	<i>usb</i>	Which USB module to perform action.
----	------------	-------------------------------------

in	ep	Endpoint index corresponding to the one listed in the device configuration table. Must be of IN direction.
----	----	--

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 727 of file qm_usb.c.

References `clk_sys_udelay()`.

4.30.4.6 `int qm_usb_ep_get_bytes_read (const qm_usb_t usb, const qm_usb_ep_idx_t ep, uint32_t *const read_bytes)`

Check how many bytes are available on OUT endpoint.

Parameters

in	usb	Which USB module to perform action.
in	ep	Endpoint index corresponding to the one listed in the device configuration table. Must be of OUT direction.
out	read_bytes	Number of bytes read. 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 857 of file qm_usb.c.

4.30.4.7 `int qm_usb_ep_halt (const qm_usb_t usb, const qm_usb_ep_idx_t ep)`

Halt the selected endpoint.

Parameters

in	usb	Which USB module to perform action.
in	ep	Endpoint index corresponding to the one listed in the device configuration table.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 609 of file qm_usb.c.

References `qm_usb_ep_set_stall_state()`.

4.30.4.8 `int qm_usb_ep_is_stalled (const qm_usb_t usb, const qm_usb_ep_idx_t ep, bool *const stalled)`

Check stall condition for the selected endpoint.

Parameters

in	<i>usb</i>	Which USB module to perform action.
in	<i>ep</i>	Endpoint index corresponding to the one listed in the device configuration table.
out	<i>stalled</i>	Endpoint stall state. 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 640 of file qm_usb.c.

4.30.4.9 int qm_usb_ep_read (const qm_usb_t *usb*, const qm_usb_ep_idx_t *ep*, uint8_t *const *data*, const uint32_t *max_len*, uint32_t *const *read_bytes*)

Read data from OUT endpoint.

This function is called by the Endpoint handler function, after an OUT interrupt has been received for that EP.

Parameters

in	<i>usb</i>	Which USB module to perform action.
in	<i>ep</i>	Endpoint index corresponding to the one listed in the device configuration table. Must be of OUT direction.
in	<i>data</i>	Pointer to data to read to. Must not be null.
in	<i>max_len</i>	Length of data requested to be read.
out	<i>read_bytes</i>	Number of bytes read.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 791 of file qm_usb.c.

4.30.4.10 int qm_usb_ep_set_config (const qm_usb_t *usb*, const qm_usb_ep_config_t *const *cfg*)

Configure endpoint.

Parameters

in	<i>usb</i>	Which USB module to perform action.
in	<i>cfg</i>	Endpoint configuration.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 495 of file qm_usb.c.

References qm_usb_ep_config_t::max_packet_size, QM_USB_EP_BULK, QM_USB_EP_CONTROL, QM_USB_EP_INTERRUPT, and qm_usb_ep_config_t::type.

4.30.4.11 int qm_usb_ep_set_stall_state (const qm_usb_t usb, const qm_usb_ep_idx_t ep, const bool is_stalled)

Set / Clear stall condition for the selected endpoint.

Parameters

in	usb	Which USB module to perform action.
in	ep	Endpoint index corresponding to the one listed in the device configuration table.
in	is_stalled	Endpoint stall state to be set.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 578 of file qm_usb.c.

Referenced by qm_usb_ep_halt().

4.30.4.12 int qm_usb_ep_write (const qm_usb_t usb, const qm_usb_ep_idx_t ep, const uint8_t *const data, const uint32_t len, uint32_t *const ret_bytes)

Write data to the specified IN endpoint.

Parameters

in	usb	Which USB module to perform action.
in	ep	Endpoint index corresponding to the one listed in the device configuration table. Must be of IN direction.
in	data	Pointer to data to write.
in	len	Length of data requested to write. This may be zero for a zero length status packet.
out	ret_bytes	Bytes scheduled for transmission. This value may be NULL if the application expects all bytes to be written.

Returns

Standard errno return type for QMSI.

Return values

0	on success.
Negative	errno for possible error codes.

Definition at line 761 of file qm_usb.c.

4.30.4.13 int qm_usb_reset (const qm_usb_t usb)

Reset the USB device controller back to it's initial state.

Performs the Core Soft Reset from the USB controller. This means that all internal state machines are reset to the IDLE state, all FIFOs are flushed and all ongoing transactions are terminated.

Note

This function does NOT disable the USB clock PLL.

Parameters

<i>in</i>	<i>usb</i>	Which USB module to perform action.
-----------	------------	-------------------------------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 476 of file qm_usb.c.

4.30.4.14 `int qm_usb_set_address (const qm_usb_t usb, const uint8_t addr)`

Set USB device address.

Parameters

<i>in</i>	<i>usb</i>	Which USB module to perform action.
<i>in</i>	<i>addr</i>	USB Device Address.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 482 of file qm_usb.c.

4.30.4.15 `int qm_usb_set_status_callback (const qm_usb_t usb, const qm_usb_status_callback_t cb)`

Set USB device controller status callback.

Parameters

<i>in</i>	<i>usb</i>	Which USB module to perform action.
<i>in</i>	<i>cb</i>	USB Device Status callback.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
----------	-------------

<i>Negative</i>	errno for possible error codes.
-----------------	---

Definition at line 848 of file qm_usb.c.

4.31 Version

Version number functions for API.

Functions

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

4.31.1 Detailed Description

Version number functions for API.

4.31.2 Function Documentation

4.31.2.1 `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`.

4.32 WDT

Watchdog timer.

Data Structures

- struct [qm_wdt_config_t](#)
QM WDT configuration type.

Enumerations

- enum [qm_wdt_mode_t](#) { [QM_WDT_MODE_RESET](#) = 0, [QM_WDT_MODE_INTERRUPT_RESET](#) }
WDT Mode 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.
- int [qm_wdt_save_context](#) (const [qm_wdt_t](#) wdt, [qm_wdt_context_t](#) *const ctx)
Save watchdog context.
- int [qm_wdt_restore_context](#) (const [qm_wdt_t](#) wdt, const [qm_wdt_context_t](#) *const ctx)
Restore watchdog context.

4.32.1 Detailed Description

Watchdog timer.

4.32.2 Enumeration Type Documentation

4.32.2.1 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 21 of file [qm_wdt.h](#).

4.32.3 Function Documentation

4.32.3.1 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

<i>in</i>	<i>wdt</i>	WDT index.
-----------	------------	------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 98 of file `qm_wdt.c`.

Referenced by `qm_wdt_set_config()`, and `qm_wdt_start()`.

4.32.3.2 `int qm_wdt_restore_context (const qm_wdt_t wdt, const qm_wdt_context_t *const ctx)`

Restore watchdog context.

Restore the configuration of the watchdog after exiting sleep.

Parameters

<i>in</i>	<i>wdt</i>	WDT index.
<i>in</i>	<i>ctx</i>	WDT context structure. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 152 of file `qm_wdt.c`.

4.32.3.3 `int qm_wdt_save_context (const qm_wdt_t wdt, qm_wdt_context_t *const ctx)`

Save watchdog context.

Save the configuration of the watchdog before entering sleep.

Parameters

<i>in</i>	<i>wdt</i>	WDT index.
<i>out</i>	<i>ctx</i>	WDT context structure. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 144 of file `qm_wdt.c`.

4.32.3.4 `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

<i>in</i>	<i>wdt</i>	WDT index.
<i>in</i>	<i>cfg</i>	New configuration for WDT. This must not be NULL.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 60 of file `qm_wdt.c`.

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

4.32.3.5 `int qm_wdt_start (const qm_wdt_t wdt)`

Start WDT.

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

Parameters

<i>in</i>	<i>wdt</i>	WDT index.
-----------	------------	------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 44 of file `qm_wdt.c`.

References `CLK_PERIPH_CLK`, `clk_periph_enable()`, `CLK_PERIPH_WDT_REGISTER`, and `qm_wdt_reload()`.

4.33 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_FREQ` = 5, `SOCW_EVENT_MAX` = 6 }

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.
- void `soc_watch_trigger_flush` ()
Trigger a buffer flush via watchpoint.

4.33.1 Detailed Description

SoC Watch (Energy Analyzer). SoC Pins definition.

4.33.2 Enumeration Type Documentation

4.33.2.1 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_FREQ Frequency altered.
SOCW_EVENT_MAX End of events sentinel.

Definition at line 36 of file soc_watch.h.

4.33.2.2 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 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 71 of file soc_watch.h.

4.33.2.3 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 88 of file soc_watch.h.

4.33.3 Function Documentation

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

Returns

Nothing.

Definition at line 344 of file soc_watch.c.

References SOCW_EVENT_HALT, SOCW_EVENT_INTERRUPT, SOCW_EVENT_MAX, and SOCW_EVENT_SLEEP.

Referenced by soc_watch_log_event().

4.33.3.2 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	<i>event_id</i>	The Event ID of the profile event.
in	<i>ev_data</i>	A parameter to the event ID (if the event needs one).

Definition at line 334 of file soc_watch.c.

References soc_watch_log_app_event().

4.33.3.3 void soc_watch_trigger_flush ()

Trigger a buffer flush via watchpoint.

This allows applications layered on top of QMSI to trigger the transfer of profiler information to the host whenever it requires.

Definition at line 467 of file soc_watch.c.

4.34 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.

4.34.1 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.

4.34.2 Enumeration Type Documentation

4.34.2.1 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.

4.34.3 Function Documentation

4.34.3.1 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.

4.34.3.2 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.

4.34.3.3 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

<i>in</i>	<i>div</i>	ADC clock divider value.
-----------	------------	--------------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	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`.

4.34.3.4 `int ss_clk_gpio_disable (const qm_ss_gpio_t gpio)`

Disable clocking for SS GPIO peripheral.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 19 of file ss_clk.c.

4.34.3.5 `int ss_clk_gpio_enable (const qm_ss_gpio_t gpio)`

Enable clocking for SS GPIO peripheral.

Parameters

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

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 8 of file ss_clk.c.

4.34.3.6 `int ss_clk_i2c_disable (const qm_ss_i2c_t i2c)`

Disable clocking for SS I2C peripheral.

Parameters

<i>in</i>	<i>i2c</i>	I2C port index.
-----------	------------	-----------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 52 of file ss_clk.c.

4.34.3.7 `int ss_clk_i2c_enable (const qm_ss_i2c_t i2c)`

Enable clocking for SS I2C peripheral.

Parameters

<i>in</i>	<i>i2c</i>	I2C port index.
-----------	------------	-----------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 44 of file ss_clk.c.

4.34.3.8 `int ss_clk_spi_disable (const qm_ss_spi_t spi)`

Disable clocking for SS SPI peripheral.

Parameters

<i>in</i>	<i>spi</i>	SPI port index.
-----------	------------	-----------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 36 of file ss_clk.c.

References QM_SS_SPI_0, and QM_SS_SPI_CTRL.

4.34.3.9 `int ss_clk_spi_enable (const qm_ss_spi_t spi)`

Enable clocking for SS SPI peripheral.

Parameters

<i>in</i>	<i>spi</i>	SPI port index.
-----------	------------	-----------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 28 of file `ss_clk.c`.

References `QM_SS_SPI_0`, and `QM_SS_SPI_CTRL`.

4.35 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` ,
`CLK_SYS_DIV_1`, `CLK_SYS_DIV_2`, `CLK_SYS_DIV_4`, `CLK_SYS_DIV_8` }
System clock divider type.
- enum `clk_sys_mode_t` {
`CLK_SYS_HYB_OSC_32MHZ` = 0, `CLK_SYS_HYB_OSC_16MHZ` = 1, `CLK_SYS_HYB_OSC_8MHZ` = 2,
`CLK_SYS_HYB_OSC_4MHZ` = 3,
`CLK_SYS_RTC_OSC` = 4, `CLK_SYS_CRYSTAL_OSC` = 5, `CLK_SYS_HYB_OSC_32MHZ` = 0, `CLK_SYS-`
`_HYB_OSC_16MHZ` = 1,
`CLK_SYS_HYB_OSC_8MHZ` = 2, `CLK_SYS_HYB_OSC_4MHZ` = 3, `CLK_SYS_RTC_OSC` = 4, `CLK_SYS-`
`_CRYSTAL_OSC` = 5 }
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`,
`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`,
`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`,
`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`,
`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.
- 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` ,
`CLK_SYS_DIV_1`, `CLK_SYS_DIV_2`, `CLK_SYS_DIV_4`, `CLK_SYS_DIV_8` }
System clock divider type.
- enum `clk_sys_mode_t` {
`CLK_SYS_HYB_OSC_32MHZ` = 0, `CLK_SYS_HYB_OSC_16MHZ` = 1, `CLK_SYS_HYB_OSC_8MHZ` = 2,
`CLK_SYS_HYB_OSC_4MHZ` = 3,
`CLK_SYS_RTC_OSC` = 4, `CLK_SYS_CRYSTAL_OSC` = 5, `CLK_SYS_HYB_OSC_32MHZ` = 0, `CLK_SYS-`
`_HYB_OSC_16MHZ` = 1,
`CLK_SYS_HYB_OSC_8MHZ` = 2, `CLK_SYS_HYB_OSC_4MHZ` = 3, `CLK_SYS_RTC_OSC` = 4, `CLK_SYS-`
`_CRYSTAL_OSC` = 5 }
System clock mode type.

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`,
`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`,
`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`,
`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`,
`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.
- int `clk_dma_enable` (void)

Enable the DMA clock.

- int [clk_dma_disable](#) (void)

Disable the DMA clock.

- uint32_t [get_i2c_clk_freq_in_mhz](#) (void)

Get I2C clock frequency in MHz.

- int [clk_sys_usb_enable](#) (void)

Enable the USB Clock mode.

- int [clk_sys_usb_disable](#) (void)

Disable the USB Clock mode.

4.35.1 Detailed Description

Clock Management.

4.35.2 Enumeration Type Documentation

4.35.2.1 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.

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 86 of file [clk.h](#).

4.35.2.2 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.

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 87 of file [clk.h](#).

4.35.2.3 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.
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 72 of file clk.h.

4.35.2.4 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.
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 73 of file clk.h.

4.35.2.5 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.
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 62 of file clk.h.

4.35.2.6 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.
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 63 of file clk.h.

4.35.2.7 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.
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 96 of file clk.h.

4.35.2.8 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.
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 97 of file clk.h.

4.35.2.9 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.
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.

Definition at line 36 of file clk.h.

4.35.2.10 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.

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.

Definition at line 39 of file clk.h.

4.35.2.11 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.

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 50 of file clk.h.

4.35.2.12 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.

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 51 of file clk.h.

4.35.3 Function Documentation

4.35.3.1 `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.

Parameters

<code>in</code>	<code>div</code>	Divider value for the ADC clock.
-----------------	------------------	----------------------------------

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 249 of file clk.c.

4.35.3.2 `int clk_dma_disable (void)`

Disable the DMA clock.

Disable the DMA clock by clearing the corresponding bit in the AHB Control register.

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 369 of file clk.c.

4.35.3.3 `int clk_dma_enable (void)`

Enable the DMA clock.

Enable the DMA clock by setting the corresponding bit in the AHB Control register. By default the DMA clock is disabled.

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 362 of file clk.c.

Referenced by `qm_dma_init()`.

4.35.3.4 `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

<i>in</i>	<i>div</i>	Divider value for the external clock.
-----------	------------	---------------------------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 293 of file clk.c.

References CLK_EXT_DIV_8.

4.35.3.5 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

<i>in</i>	<i>div</i>	Divider value for the GPIO debounce clock.
-----------	------------	--

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 279 of file clk.c.

References CLK_GPIO_DB_DIV_128.

4.35.3.6 int clk_periph_disable (const clk_periph_t clocks)

Disable clocks for peripherals / registers.

Parameters

<i>in</i>	<i>clocks</i>	Which peripheral and register clocks to disable.
-----------	---------------	--

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 333 of file clk.c.

References CLK_PERIPH_ALL, SOCW_EVENT_REGISTER, and SOCW_REG_CCU_PERIPH_CLK_GATE_CTL.

Referenced by qm_power_soc_deep_sleep(), and qm_power_soc_sleep().

4.35.3.7 int clk_periph_enable (const clk_periph_t clocks)

Enable clocks for peripherals / registers.

Parameters

<i>in</i>	<i>clocks</i>	Which peripheral and register clocks to enable.
-----------	---------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 319 of file clk.c.

References CLK_PERIPH_ALL, SOCW_EVENT_REGISTER, and SOCW_REG_CCU_PERIPH_CLK_GATE_CTL.

Referenced by qm_power_soc_deep_sleep_restore(), and qm_wdt_start().

4.35.3.8 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

<i>in</i>	<i>div</i>	Divider value for the peripheral clock.
-----------	------------	---

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 265 of file clk.c.

References CLK_PERIPH_DIV_8.

4.35.3.9 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

<i>in</i>	<i>div</i>	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 306 of file clk.c.

References CLK_RTC_DIV_32768.

Referenced by qm_rtc_set_config().

4.35.3.10 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 347 of file clk.c.

Referenced by get_i2c_clk_freq_in_mhz(), and ss_clk_adc_set_div().

4.35.3.11 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 100 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(), and SOCW_EVENT_FREQ.

Referenced by clk_sys_usb_enable(), qm_power_soc_deep_sleep(), and qm_power_soc_sleep().

4.35.3.12 void clk_sys_udelay (uint32_t microseconds)

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

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

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

Parameters

<i>in</i>	<i>microseconds</i>	Minimum number of micro seconds to delay for.
-----------	---------------------	---

Definition at line 352 of file clk.c.

Referenced by `clk_sys_usb_disable()`, `clk_sys_usb_enable()`, `clk_trim_apply()`, and `qm_usb_ep_flush()`.

4.35.3.13 `int clk_sys_usb_disable (void)`

Disable the USB Clock mode.

This function will disable the USB Clock and PLL. The system clock will remain as `CLK_SYS_CRYSTAL_OSC` mode and `CLK_SYS_DIV_1` divider.

Note: If the application must restore the original system clock mode / divisor it must explicitly call `clk_sys_set_mode()` restoring the previous config. This can only be done after the USB clock mode is disabled.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 386 of file clk.c.

References `clk_sys_udelay()`.

Referenced by `qm_usb_detach()`.

4.35.3.14 `int clk_sys_usb_enable (void)`

Enable the USB Clock mode.

For Quark SE, this function will set the system clock to `CLK_SYS_CRYSTAL_OSC` mode with `CLK_SYS_DIV_1` divisor. It then will enable the USB Clock and PLL, blocking execution until the USB PLL lock is ready.

Note: Application must retain the original system clock mode / divisor in case it will need to restore it after disabling the usb clock.

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 356 of file clk.c.

References `CLK_SYS_CRYSTAL_OSC`, `CLK_SYS_DIV_1`, `clk_sys_set_mode()`, and `clk_sys_udelay()`.

Referenced by `qm_usb_attach()`.

4.35.3.15 `int clk_trim_apply (const uint32_t value)`

Apply silicon oscillator trim code.

Parameters

<i>in</i>	<i>value</i>	Trim code to apply.
-----------	--------------	---------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 224 of file clk.c.

References `clk_sys_udelay()`.

Referenced by `clk_sys_set_mode()`.

4.35.3.16 int clk_trim_read (uint32_t *const *value*)

Read the silicon oscillator trim code for the current frequency.

Parameters

<i>out</i>	<i>value</i>	Pointer to store the trim code.
------------	--------------	---------------------------------

Returns

Standard errno return type for QMSI.

Return values

0	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 214 of file clk.c.

4.35.3.17 uint32_t get_i2c_clk_freq_in_mhz (void)

Get I2C clock frequency in MHz.

Returns

[uint32_t] I2C freq_in_mhz.

Definition at line 381 of file clk.c.

References `clk_sys_get_ticks_per_us()`.

4.36 Quark D2000 Flash Layout

Flash Layout for Quark D2000 Microcontrollers.

4.36.1 Detailed Description

Flash Layout for Quark D2000 Microcontrollers.

4.37 Quark D2000 Power states

Power mode control for Quark D2000 Microcontrollers.

Modules

- [Quark\(TM\) D2000 Retention Alternator Regulator \(RAR\)](#).

Retention alternator regulator for Quark D2000.

Enumerations

- enum [qm_power_wake_event_t](#) { [QM_POWER_WAKE_FROM_GPIO_COMP](#), [QM_POWER_WAKE_FROM_RTC](#) }

Wake source for deep sleep mode type.

Functions

- void [qm_power_cpu_halt](#) (void)
Put CPU in halt state.
- void [qm_power_soc_sleep](#) ()
Put SoC to sleep.
- void [qm_power_soc_deep_sleep](#) (const [qm_power_wake_event_t](#) wake_event)
Put SoC to deep sleep.
- void [qm_power_soc_restore](#) (void)
Restore system state after sleep or deep sleep.

4.37.1 Detailed Description

Power mode control for Quark D2000 Microcontrollers.

4.37.2 Enumeration Type Documentation

4.37.2.1 enum [qm_power_wake_event_t](#)

Wake source for deep sleep mode type.

Enumerator

[QM_POWER_WAKE_FROM_GPIO_COMP](#) Use GPIO/Comparator as wake source.

[QM_POWER_WAKE_FROM_RTC](#) Use RTC as wake source.

Definition at line 21 of file power_states.h.

4.37.3 Function Documentation

4.37.3.1 void [qm_power_cpu_halt](#) (void)

Put CPU in halt state.

Halts the CPU until next interrupt or reset.

This function can be called with interrupts disabled. Interrupts will be enabled before triggering the transition.

Definition at line 29 of file power_states.c.

References SOCW_EVENT_HALT.

Referenced by `qm_power_soc_deep_sleep()`, and `qm_power_soc_sleep()`.

4.37.3.2 `void qm_power_soc_deep_sleep (const qm_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, QM_POWER_WAKE_FROM_GPIO_COMP will enable waking from GPIO or comparator pins and QM_POWER_WAKE_FROM_RTC will enable waking from the RTC.

Parameters

<code>in</code>	<code>wake_event</code>	Select wake source for deep sleep mode.
-----------------	-------------------------	---

Definition at line 180 of file `power_states.c`.

References CLK_PERIPH_CLK, `clk_periph_disable()`, CLK_PERIPH_REGISTER, CLK_SYS_DIV_1, CLK_SYS_DIV_128, CLK_SYS_HYB_OSC_4MHZ, CLK_SYS_RTC_OSC, `clk_sys_set_mode()`, QM_ADC_MODE_DEEP_POWER_DOWN, `qm_adc_set_mode()`, `qm_power_cpu_halt()`, QM_POWER_WAKE_FROM_GPIO_COMP, QM_POWER_WAKE_FROM_RTC, QM_RAR_RETENTION, `qm_rar_set_mode()`, SOCW_EVENT_REGISTER, SOCW_REG_CCU_EXT_CLK_CTL, SOCW_REG_CCU_LP_CLK_CTL, and SOCW_REG_PMUX_SLEW.

Referenced by `qm_power_soc_deep_sleep_restore()`, and `qm_ss_power_soc_deep_sleep_restore()`.

4.37.3.3 `void qm_power_soc_restore (void)`

Restore system state after sleep or deep sleep.

On wakeup, the system is restored to the previous state before `qm_power_soc_sleep()` or `qm_power_soc_deep_sleep()` was called.

Definition at line 402 of file `power_states.c`.

References `qm_power_soc_deep_sleep_restore()`, and `qm_power_soc_sleep_restore()`.

Referenced by QM_ISR_DECLARE().

4.37.3.4 `void qm_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 59 of file power_states.c.

4.38 Quark(TM) D2000 Retention Alternator Regulator (RAR).

Retention alternator regulator for Quark D2000.

Enumerations

- enum `qm_rar_state_t` { `QM_RAR_NORMAL`, `QM_RAR_RETENTION` }
RAR modes type.

Functions

- int `qm_rar_set_mode` (const `qm_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 `qm_rar_state_t`

RAR modes type.

Enumerator

`QM_RAR_NORMAL` Normal mode = 50 mA.

`QM_RAR_RETENTION` Retention mode = 300 uA.

Definition at line 90 of file `power_states.h`.

4.38.3 Function Documentation

4.38.3.1 int `qm_rar_set_mode` (const `qm_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

<code>in</code>	<code>mode</code>	Operating mode of the RAR.
-----------------	-------------------	----------------------------

Returns

Standard errno return type for QMSI.

Return values

<code>0</code>	on success.
----------------	-------------

<i>Negative</i>	errno for possible error codes.
-----------------	---

Definition at line 415 of file power_states.c.

References QM_RAR_NORMAL, and QM_RAR_RETENTION.

Referenced by qm_power_soc_deep_sleep(), qm_power_soc_deep_sleep_restore(), qm_power_soc_sleep(), and qm_power_soc_sleep_restore().

4.39 SoC Interrupt Router (D2000)

Quark D2000 SoC Interrupt Router.

Data Structures

- struct [qm_interrupt_router_reg_t](#)
Interrupt register map.

4.39.1 Detailed Description

Quark D2000 SoC Interrupt Router. Quark D2000 SoC Interrupt Router registers.

4.40 SoC Interrupts (D2000)

Quark D2000 SoC Interrupts.

Quark D2000 SoC Interrupts.

4.41 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_pmu_reg_t](#)
Power Management register map.
- struct [qm_aonc_reg_t](#)
Always-on Counter 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](#)

Power Management

- [qm_scss_pmu_reg_t](#) [test_scss_pmu](#)

Always-on Counters.

- enum [qm_aonc_t](#)
Number of Always-on counter controllers.
- [qm_aonc_reg_t](#) [test_aonc_instance](#) [QM_AONC_NUM]
- [qm_aonc_reg_t](#) * [test_aonc](#) [QM_AONC_NUM]
- [qm_aonc_reg_t](#) * [qm_aonc](#) [QM_AONC_NUM]

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_instance` [QM_PWM_NUM]
- `qm_pwm_reg_t * test_pwm` [QM_PWM_NUM]
- `qm_pwm_reg_t * qm_pwm` [QM_PWM_NUM]

WDT

- enum `qm_wdt_t`
Number of WDT controllers.
- `qm_wdt_reg_t test_wdt_instance` [QM_WDT_NUM]
- `qm_wdt_reg_t * test_wdt` [QM_WDT_NUM]
- `qm_wdt_reg_t * qm_wdt` [QM_WDT_NUM]

UART

WDT timeout table (in clock cycles): Each table entry corresponds with the value loaded into the WDT at the time of a WDT reload for the corresponding timeout range register value.

TORR | Timeout (Clock Cycles) 0. | 2^{16} (65536)

1. | 2^{17} (131072)
2. | 2^{18} (262144)
3. | 2^{19} (524288)
4. | 2^{20} (1048576)
5. | 2^{21} (2097152)
6. | 2^{22} (4194304)
7. | 2^{23} (8388608)
8. | 2^{24} (16777216)
9. | 2^{25} (33554432)
10. | 2^{26} (67108864)
11. | 2^{27} (134217728)
12. | 2^{28} (268435456)
13. | 2^{29} (536870912)
14. | 2^{30} (1073741824)
15. | 2^{31} (2147483648)

- 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_instance** [QM_RTC_NUM]
- [qm_rtc_reg_t](#) * **test_rtc** [QM_RTC_NUM]
- [qm_rtc_reg_t](#) * **qm_rtc** [QM_RTC_NUM]

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]

Flash Protection Region

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

FPR register map.

Memory Protection Region

- enum `qm_mpr_id_t` {
`QM_MPR_0 = 0, QM_MPR_1, QM_MPR_2, QM_MPR_3,`
`QM_MPR_NUM, QM_MPR_0 = 0, QM_MPR_1, QM_MPR_2,`
`QM_MPR_3, QM_MPR_NUM` }
- `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_P-`
`ERIPH_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_G-`
`PIO_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 = 0xCFFFFFF,`
`CLK_PERIPH_REGISTER = BIT(0),`
`CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_PERIPH_I2C_M1 = BIT(3), CLK_PERI-`
`PH_SPI_S = BIT(4),`
`CLK_PERIPH_SPI_M0 = BIT(5), CLK_PERIPH_SPI_M1 = BIT(6), CLK_PERIPH_GPIO_INTERRUPT = BI-`
`T(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_PERIP-`
`H_I2C_M0_REGISTER, CLK_PERIPH_I2C_M1_REGISTER,`
`CLK_PERIPH_I2S_REGISTER = BIT(21), CLK_PERIPH_ALL = 0x3FFFFFF }`

Peripheral clock register map.

MVIC

- `qm_mvica_reg_t test_mvica`
- `qm_mvica_reg_t test_mvica`

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`

4.41.1 Detailed Description

Quark D2000 SoC Registers.

4.41.2 Enumeration Type Documentation

4.41.2.1 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 1368 of file qm_soc_regs.h.

4.41.2.2 enum qm_adc_t

Number of ADC controllers.

Definition at line 1066 of file qm_soc_regs.h.

4.41.2.3 enum qm_aonc_t

Number of Always-on counter controllers.

Definition at line 255 of file qm_soc_regs.h.

4.41.2.4 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 1486 of file qm_soc_regs.h.

4.41.2.5 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 1493 of file qm_soc_regs.h.

4.41.2.6 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 1480 of file qm_soc_regs.h.

4.41.2.7 enum qm_flash_t

Number of Flash controllers.

Definition at line 1141 of file qm_soc_regs.h.

4.41.2.8 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.

QM_FPR_0 FPR 0.

QM_FPR_1 FPR 1.

QM_FPR_2 FPR 2.

QM_FPR_3 FPR 3.

Definition at line 1265 of file qm_soc_regs.h.

4.41.2.9 enum qm_gpio_t

Number of GPIO controllers.

Definition at line 1014 of file qm_soc_regs.h.

4.41.2.10 enum qm_i2c_t

Number of I2C controllers.

Definition at line 861 of file qm_soc_regs.h.

4.41.2.11 enum qm_mpr_id_t

Enumerator

QM_MPR_0 Memory Protection Region 0.

QM_MPR_1 Memory Protection Region 1.

QM_MPR_2 Memory Protection Region 2.

QM_MPR_3 Memory Protection Region 3.

QM_MPR_NUM Number of Memory Protection Regions.

QM_MPR_0 Memory Protection Region 0.

QM_MPR_1 Memory Protection Region 1.
QM_MPR_2 Memory Protection Region 2.
QM_MPR_3 Memory Protection Region 3.
QM_MPR_NUM Number of Memory Protection Regions.

Definition at line 1286 of file qm_soc_regs.h.

4.41.2.12 enum qm_pwm_id_t

PWM ID type.

Definition at line 386 of file qm_soc_regs.h.

4.41.2.13 enum qm_pwm_t

Number of PWM / Timer controllers.

Definition at line 383 of file qm_soc_regs.h.

4.41.2.14 enum qm_rtc_t

Number of RTC controllers.

Definition at line 818 of file qm_soc_regs.h.

4.41.2.15 enum qm_spi_t

Number of SPI controllers.

Definition at line 709 of file qm_soc_regs.h.

4.41.2.16 enum qm_uart_t

Number of UART controllers.

Definition at line 655 of file qm_soc_regs.h.

4.41.2.17 enum qm_wdt_t

Number of WDT controllers.

Definition at line 467 of file qm_soc_regs.h.

4.41.3 Variable Documentation

4.41.3.1 qm_i2c_reg_t* qm_i2c[QM_I2C_NUM]

I2C register block.

Definition at line 20 of file qm_i2c.c.

4.42 Quark SE Flash Layout

Flash Layout for Quark SE Microcontrollers.

4.42.1 Detailed Description

Flash Layout for Quark SE Microcontrollers.

4.43 Quark SE SoC Power states

SoC Power mode control for Quark SE Microcontrollers.

Functions

- void [qm_power_soc_sleep](#) (void)
Enter SoC sleep state.
- void [qm_power_soc_deep_sleep](#) (void)
Enter SoC deep sleep state.
- void [qm_power_soc_sleep_restore](#) (void)
Enter SoC sleep state and restore after wake up.
- void [qm_power_soc_deep_sleep_restore](#) (void)
Enter SoC deep sleep state and restore after wake up.
- void [qm_power_sleep_wait](#) (void)
Save context, enter x86 C2 power save state and restore after wake up.
- void [qm_power_soc_set_x86_restore_flag](#) (void)
Enable the x86 startup restore flag, see GPS0 #define in qm_soc_regs.h.

4.43.1 Detailed Description

SoC Power mode control for Quark SE Microcontrollers. Available SoC states are:

- Low Power Sensing Standby (LPSS)
- Sleep

LPSS can only be enabled from the Sensor core, refer to [qm_ss_power_soc_lpss_enable](#) for further details.

4.43.2 Function Documentation

4.43.2.1 void qm_power_sleep_wait (void)

Save context, enter x86 C2 power save state and restore after wake up.

This routine is same as [qm_power_soc_sleep_restore\(\)](#), just instead of going to sleep it will go to C2 power save state. Note: this function has a while(1) which will spin until we enter (and exit) sleep while the power state change will be managed by the other core.

Definition at line 108 of file power_states.c.

References [qm_power_cpu_c2\(\)](#), and [qm_power_soc_set_x86_restore_flag\(\)](#).

4.43.2.2 void qm_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 23 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()`.

4.43.2.3 void `qm_power_soc_deep_sleep_restore` (void)

Enter SoC deep sleep state and restore after wake up.

Put the SoC into deep sleep state until next SoC wake event and continue execution after wake up where the application stopped.

If the library is built with `ENABLE_RESTORE_CONTEXT=1`, then this function will use the common RAM `__x86_restore_info[0]` to save the necessary context to bring back the CPU to the point where this function was called. This means that applications should refrain from using them.

This function calls `qm_x86_save_context` and `qm_x86_restore_context` in order to restore execution where it stopped. All power management transitions are done by [qm_power_soc_deep_sleep\(\)](#).

Definition at line 304 of file `power_states.c`.

References `clk_periph_enable()`, `CLK_PERIPH_REGISTER`, `qm_power_soc_deep_sleep()`, `qm_power_soc_set_x86_restore_flag()`, `QM_RAR_NORMAL`, `qm_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`.

Referenced by `qm_power_soc_restore()`.

4.43.2.4 void `qm_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 59 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()`, `QM_ADC_MODE_PWR_DOWN`, `qm_adc_set_mode()`, `qm_power_cpu_halt()`, `QM_RAR_RETENTION`, `qm_rar_set_mode()`, `SOCW_EVENT_REGISTER`, `SOCW_EVENT_SLEEP`, `SOCW_REG_CCU_LP_CLK_CTL`, and `SOCW_REG_SLP_CFG`.

Referenced by `qm_power_soc_sleep_restore()`, and `qm_ss_power_soc_sleep_restore()`.

4.43.2.5 void qm_power_soc_sleep_restore (void)

Enter SoC sleep state and restore after wake up.

Put the SoC into sleep state until next SoC wake event and continue execution after wake up where the application stopped.

If the library is built with `ENABLE_RESTORE_CONTEXT=1`, then this function will use the common RAM `__x86_restore_info[0]` to save the necessary context to bring back the CPU to the point where this function was called. This means that applications should refrain from using them.

This function calls `qm_x86_save_context` and `qm_x86_restore_context` in order to restore execution where it stopped. All power management transitions are done by [qm_power_soc_sleep\(\)](#).

Definition at line 135 of file `power_states.c`.

References `qm_power_soc_set_x86_restore_flag()`, `qm_power_soc_sleep()`, `QM_RAR_NORMAL`, `qm_rar_set_mode()`, `SOCW_EVENT_REGISTER`, `SOCW_REG_CCU_PERIPH_CLK_GATE_CTL`, `SOCW_REG_CCU_SYS_CLK_CTL`, and `SOCW_REG_OSC0_CFG1`.

Referenced by `qm_power_soc_restore()`.

4.44 Quark SE Host Power states

Host Power mode control for Quark SE Microcontrollers.

Functions

- void [qm_power_cpu_c1](#) (void)
Enter Host C1 state.
- void [qm_power_cpu_c2](#) (void)
Enter Host C2 state or SoC LPSS state.
- void [qm_power_cpu_c2lp](#) (void)
Enter Host C2LP state or SoC LPSS state.

4.44.1 Detailed Description

Host Power mode control for Quark SE Microcontrollers.

These functions cannot be called from the Sensor Subsystem.

4.44.2 Function Documentation

4.44.2.1 void [qm_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.

This function can be called with interrupts disabled. Interrupts will be enabled before triggering the transition.

A wake event causes the Host to transition to C0.

A wake event is a host interrupt.

Definition at line 145 of file `power_states.c`.

References `SOCW_EVENT_HALT`.

4.44.2.2 void [qm_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 160 of file `power_states.c`.

References `SOCW_EVENT_REGISTER`, `SOCW_EVENT_SLEEP`, and `SOCW_REG_CCU_LP_CLK_CTL`.

Referenced by `qm_power_sleep_wait()`.

4.44.2.3 `void qm_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 170 of file `power_states.c`.

References `SOCW_EVENT_REGISTER`, `SOCW_EVENT_SLEEP`, and `SOCW_REG_CCU_LP_CLK_CTL`.

4.45 SoC Interrupt Router (SE)

Quark SE SoC Interrupt Router registers.

Data Structures

- struct [int_ss_i2c_reg_t](#)
SS I2C Interrupt register map.
- struct [int_ss_spi_reg_t](#)
SS SPI Interrupt register map.
- struct [qm_interrupt_router_reg_t](#)
Interrupt register map.

4.45.1 Detailed Description

Quark SE SoC Interrupt Router registers.

4.46 Mailbox Definitions

Mailbox definitions.

Enumerations

- 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` }
Mailbox channel identifiers.
- enum `qm_mbox_destination_t` { , `QM_MBOX_TO_LMT`, `QM_MBOX_TO_SS` }
Definition of the mailbox direction of operation The direction of communication for each channel is configurable by the user.

4.46.1 Detailed Description

Mailbox definitions.

4.46.2 Enumeration Type Documentation

4.46.2.1 enum `qm_mbox_ch_t`

Mailbox channel identifiers.

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.

Definition at line 18 of file `qm_mailbox_defs.h`.

4.46.2.2 enum `qm_mbox_destination_t`

Definition of the mailbox direction of operation The direction of communication for each channel is configurable by the user.

The list below describes the possible communication directions for each channel.

Enumerator

`QM_MBOX_TO_LMT` Lakemont core as destination.
`QM_MBOX_TO_SS` Sensor Sub-System core as destination.

Definition at line 35 of file `qm_mailbox_defs.h`.

4.47 SoC Registers (Sensor Subsystem)

Quark SE SoC Sensor Subsystem Registers.

Data Structures

- struct [qm_irq_context_t](#)
SS IRQ context type.
- struct [qm_ss_gpio_context_t](#)
SS GPIO context type.
- struct [qm_ss_i2c_context_t](#)
SS I2C context type.
- struct [qm_ss_adc_context_t](#)
SS ADC context type.
- struct [qm_ss_spi_context_t](#)
Sensor Subsystem SPI context type.

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 SPI

SPI 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.

4.47.1 Detailed Description

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

4.47.2 Enumeration Type Documentation

4.47.2.1 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 541 of file `qm_sensor_regs.h`.

4.47.2.2 enum `qm_ss_adc_t`

Sensor Subsystem ADC.

Enumerator

`QM_SS_ADC_0` ADC first module.

Definition at line 551 of file `qm_sensor_regs.h`.

4.47.2.3 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 628 of file qm_sensor_regs.h.

4.47.2.4 enum qm_ss_gpio_reg_t

Sensor Subsystem GPIO register block type.

Definition at line 159 of file qm_sensor_regs.h.

4.47.2.5 enum qm_ss_gpio_t

Sensor Subsystem GPIO.

Definition at line 196 of file qm_sensor_regs.h.

4.47.2.6 enum qm_ss_i2c_reg_t

Sensor Subsystem I2C register block type.

Definition at line 211 of file qm_sensor_regs.h.

4.47.2.7 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 676 of file qm_sensor_regs.h.

4.47.2.8 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 704 of file qm_sensor_regs.h.

4.48 SoC Interrupts (SE)

Quark SE SoC Interrupts.

SS Interrupt

Sensor Sub-System Specific IRQs and interrupt vectors.

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

4.48.1 Detailed Description

Quark SE SoC Interrupts.

4.49 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 [qm_irq_context_t](#)
SS IRQ context type.
- struct [qm_pic_timer_context_t](#)
PIC TIMER context type.
- struct [qm_scss_pmu_reg_t](#)
Power Management register map.
- struct [qm_scss_ss_reg_t](#)
Sensor Subsystem register map.
- struct [qm_aonc_reg_t](#)
Always-on Counter 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_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_pwm_context_t](#)
PWM context type.
- struct [qm_wdt_reg_t](#)
Watchdog timer register map.
- struct [qm_uart_reg_t](#)
UART register map.
- struct [qm_uart_context_t](#)
UART context to be saved between sleep/resume.
- struct [qm_spi_reg_t](#)
SPI register map.
- struct [qm_spi_context_t](#)

- SPI context type.*
 - struct [qm_rtc_reg_t](#)
 - RTC register map.*
 - struct [qm_i2c_reg_t](#)
 - I2C register map.*
 - struct [qm_i2c_context_t](#)
 - I2C context to be saved between sleep/resume.*
 - struct [qm_gpio_reg_t](#)
 - GPIO register map.*
 - struct [qm_gpio_context_t](#)
 - GPIO context type.*
 - struct [qm_flash_reg_t](#)
 - Flash register map.*
 - struct [qm_flash_context_t](#)
 - Flash context type.*
 - struct [qm_fpr_context_t](#)
 - FPR context type.*
 - struct [qm_mpr_reg_t](#)
 - Memory Protection Region register map.*
 - struct [qm_mpr_context_t](#)
 - MPR context type.*
 - 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.*
 - struct [qm_dma_context_t](#)
 - DMA context type.*
 - struct [qm_usb_in_ep_reg_t](#)
 - USB register map.*
 - struct [qm_usb_out_ep_reg_t](#)
 - OUT Endpoint Registers.*
 - struct [qm_usb_reg_t](#)
 - USB Register block type.*

System Core

- [qm_scss_ccu_reg_t](#) **test_scss_ccu**

General Purpose

- [qm_scss_gp_reg_t](#) **test_scss_gp**

Memory Control

- [qm_scss_mem_reg_t](#) **test_scss_mem**

Comparator

- [qm_scss_cmp_reg_t](#) **test_scss_cmp**

APIC

- [qm_lapic_reg_t](#) **test_lapic**
- [qm_ioapic_reg_t](#) **test_ioapic**

Power Management

- [qm_scss_pmu_reg_t](#) **test_scss_pmu**

Sensor Subsystem

- [qm_scss_ss_reg_t](#) **test_scss_ss**

Always-on Counters.

- enum [qm_aonc_t](#)
Number of Always-on counter controllers.
- [qm_aonc_reg_t](#) **test_aonc_instance** [QM_AONC_NUM]
- [qm_aonc_reg_t](#) * **test_aonc** [QM_AONC_NUM]
- [qm_aonc_reg_t](#) * **qm_aonc** [QM_AONC_NUM]

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_mailbox_reg_t](#) **test_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_instance** [QM_PWM_NUM]
- [qm_pwm_reg_t](#) * **test_pwm** [QM_PWM_NUM]
- [qm_pwm_reg_t](#) * **qm_pwm** [QM_PWM_NUM]

WDT

- enum `qm_wdt_t`
Number of WDT controllers.
- `qm_wdt_reg_t test_wdt_instance` [QM_WDT_NUM]
- `qm_wdt_reg_t * test_wdt` [QM_WDT_NUM]
- `qm_wdt_reg_t * qm_wdt` [QM_WDT_NUM]

UART

WDT timeout table (in clock cycles): Each table entry corresponds with the value loaded into the WDT at the time of a WDT reload for the corresponding timeout range register value.

TORR | Timeout (Clock Cycles) 0. | 2^{16} (65536)

1. | 2^{17} (131072)
2. | 2^{18} (262144)
3. | 2^{19} (524288)
4. | 2^{20} (1048576)
5. | 2^{21} (2097152)
6. | 2^{22} (4194304)
7. | 2^{23} (8388608)
8. | 2^{24} (16777216)
9. | 2^{25} (33554432)
10. | 2^{26} (67108864)
11. | 2^{27} (134217728)
12. | 2^{28} (268435456)
13. | 2^{29} (536870912)
14. | 2^{30} (1073741824)
15. | 2^{31} (2147483648)

- 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_instance** [QM_RTC_NUM]
- [qm_rtc_reg_t](#) * **test_rtc** [QM_RTC_NUM]
- [qm_rtc_reg_t](#) * **qm_rtc** [QM_RTC_NUM]

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]

Flash Protection Region

- enum [qm_fpr_id_t](#) {
 QM_FPR_0, QM_FPR_1, QM_FPR_2, QM_FPR_3 ,
 QM_FPR_0, QM_FPR_1, QM_FPR_2, QM_FPR_3 }
FPR register map.

Memory Protection Region

- enum [qm_mpr_id_t](#) {
 QM_MPR_0 = 0, QM_MPR_1, QM_MPR_2, QM_MPR_3,
 QM_MPR_NUM, QM_MPR_0 = 0, QM_MPR_1, QM_MPR_2,
 QM_MPR_3, QM_MPR_NUM }
• [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` = 0xCFFFFFFF,
`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` = 0x3FFFFFFF }

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`]

USB

- enum [qm_usb_t](#)
Number of USB controllers.
- enum **qm_usb_ep_idx_t**
- [qm_usb_reg_t](#) **test_usb**
- uint32_t **test_usb_pll**

Versioning

- uint32_t **test_rom_version**

4.49.1 Detailed Description

Quark SE SoC Registers.

4.49.2 Enumeration Type Documentation

4.49.2.1 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 1669 of file qm_soc_regs.h.

4.49.2.2 enum qm_aonc_t

Number of Always-on counter controllers.

Definition at line 432 of file qm_soc_regs.h.

4.49.2.3 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 1721 of file qm_soc_regs.h.

4.49.2.4 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 1734 of file qm_soc_regs.h.

4.49.2.5 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 1715 of file qm_soc_regs.h.

4.49.2.6 enum qm_flash_t

Number of Flash controllers.

Definition at line 1442 of file qm_soc_regs.h.

4.49.2.7 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.

QM_FPR_0 FPR 0.

QM_FPR_1 FPR 1.

QM_FPR_2 FPR 2.

QM_FPR_3 FPR 3.

Definition at line 1576 of file qm_soc_regs.h.

4.49.2.8 enum qm_gpio_t

Number of GPIO controllers.

Definition at line 1366 of file qm_soc_regs.h.

4.49.2.9 enum qm_i2c_t

Number of I2C controllers.

Definition at line 1193 of file qm_soc_regs.h.

4.49.2.10 enum qm_mpr_id_t

Enumerator

QM_MPR_0 Memory Protection Region 0.

QM_MPR_1 Memory Protection Region 1.

QM_MPR_2 Memory Protection Region 2.

QM_MPR_3 Memory Protection Region 3.

QM_MPR_NUM Number of Memory Protection Regions.

QM_MPR_0 Memory Protection Region 0.

QM_MPR_1 Memory Protection Region 1.

QM_MPR_2 Memory Protection Region 2.

QM_MPR_3 Memory Protection Region 3.

QM_MPR_NUM Number of Memory Protection Regions.

Definition at line 1607 of file qm_soc_regs.h.

4.49.2.11 enum qm_pwm_id_t

PWM ID type.

Definition at line 666 of file qm_soc_regs.h.

4.49.2.12 enum qm_pwm_t

Number of PWM / Timer controllers.

Definition at line 663 of file qm_soc_regs.h.

4.49.2.13 enum `qm_rtc_t`

Number of RTC controllers.

Definition at line 1149 of file `qm_soc_regs.h`.

4.49.2.14 enum `qm_spi_t`

Number of SPI controllers.

Definition at line 1026 of file `qm_soc_regs.h`.

4.49.2.15 enum `qm_uart_t`

Number of UART controllers.

Definition at line 964 of file `qm_soc_regs.h`.

4.49.2.16 enum `qm_usb_t`

Number of USB controllers.

Definition at line 1944 of file `qm_soc_regs.h`.

4.49.2.17 enum `qm_wdt_t`

Number of WDT controllers.

Definition at line 766 of file `qm_soc_regs.h`.

4.49.3 Variable Documentation

4.49.3.1 `qm_i2c_reg_t`* `qm_i2c[QM_I2C_NUM]`

I2C register block.

Definition at line 20 of file `qm_i2c.c`.

4.50 Quark SE Sensor Subsystem Initialisation

Sensor Subsystem initialisation.

Functions

- void `sensor_activation` (void)
Initialise the sensor subsystem.

4.50.1 Detailed Description

Sensor Subsystem initialisation.

4.50.2 Function Documentation

4.50.2.1 void `sensor_activation` (void)

Initialise the sensor subsystem.

Set the ARC reset vector with the sensor subsystem application entry point address. Then, instruct the ARC to start running. Note: This function is only intended to be called from the Lakemont core.

Returns

void.

Definition at line 37 of file `ss_init.c`.

4.51 SS Power states

SS Power mode control for Quark SE Microcontrollers.

Enumerations

- enum `qm_ss_power_cpu_ss1_mode_t` { `QM_SS_POWER_CPU_SS1_TIMER_OFF` = 0, `QM_SS_POWER_CPU_SS1_TIMER_ON` }

Sensor Subsystem SS1 Timers mode type.

Functions

- void `qm_ss_power_soc_lpss_enable` (void)
Enable LPSS state entry.
- void `qm_ss_power_soc_sleep_restore` (void)
Enter SoC sleep state and restore after wake up.
- void `qm_ss_power_soc_deep_sleep_restore` (void)
Enter SoC sleep state and restore after wake up.
- void `qm_ss_power_sleep_wait` (void)
Save context, enter ARC SS1 power save state and restore after wake up.
- void `qm_power_soc_set_ss_restore_flag` (void)
Enable the SENSOR startup restore flag.
- void `qm_ss_power_soc_lpss_disable` (void)
Disable LPSS state entry.
- void `qm_ss_power_cpu_ss1` (const `qm_ss_power_cpu_ss1_mode_t` mode)
Enter Sensor SS1 state.
- void `qm_ss_power_cpu_ss2` (void)
Enter Sensor SS2 state or SoC LPSS state.

4.51.1 Detailed Description

SS Power mode control for Quark SE Microcontrollers.

4.51.2 Enumeration Type Documentation

4.51.2.1 enum `qm_ss_power_cpu_ss1_mode_t`

Sensor Subsystem SS1 Timers mode type.

Enumerator

`QM_SS_POWER_CPU_SS1_TIMER_OFF` Disable SS Timers in SS1.

`QM_SS_POWER_CPU_SS1_TIMER_ON` Keep SS Timers enabled in SS1.

Definition at line 21 of file `ss_power_states.h`.

4.51.3 Function Documentation

4.51.3.1 void qm_ss_power_cpu_ss1 (const qm_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

<i>in</i>	<i>mode</i>	Mode selection for SS1 state.
-----------	-------------	-------------------------------

Definition at line 91 of file ss_power_states.c.

References QM_SS_POWER_CPU_SS1_TIMER_OFF, and QM_SS_POWER_CPU_SS1_TIMER_ON.

Referenced by qm_ss_power_sleep_wait().

4.51.3.2 void qm_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 122 of file ss_power_states.c.

4.51.3.3 void qm_ss_power_sleep_wait (void)

Save context, enter ARC SS1 power save state and restore after wake up.

This routine is same as [qm_ss_power_soc_sleep_restore\(\)](#), just instead of going to sleep it will go to SS1 power save state. Note: this function has a while(1) which will spin until we enter (and exit) sleep and the power state change will be managed by the other core.

Definition at line 196 of file ss_power_states.c.

References qm_power_soc_set_ss_restore_flag(), qm_ss_power_cpu_ss1(), and QM_SS_POWER_CPU_SS1_TIMER_ON.

4.51.3.4 void qm_ss_power_soc_deep_sleep_restore (void)

Enter SoC sleep state and restore after wake up.

Put the ARC core into sleep state until next SoC wake event and continue execution after wake up where the application stopped.

If the library is built with ENABLE_RESTORE_CONTEXT=1, then this function will use the arc_restore_addr to save restore trap address which brings back the ARC CPU to the point where this function was called. This means that applications should refrain from using them.

This function calls qm_ss_save_context and qm_ss_restore_context in order to restore execution where it stopped. All power management transitions are done by power_qm_soc_deep_sleep().

Definition at line 168 of file ss_power_states.c.

References qm_power_soc_deep_sleep(), and qm_power_soc_set_ss_restore_flag().

4.51.3.5 void qm_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 64 of file ss_power_states.c.

References SOCW_EVENT_REGISTER, and SOCW_REG_CCU_LP_CLK_CTL.

4.51.3.6 void qm_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, qm_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 41 of file ss_power_states.c.

References SOCW_EVENT_REGISTER, and SOCW_REG_CCU_LP_CLK_CTL.

4.51.3.7 void qm_ss_power_soc_sleep_restore (void)

Enter SoC sleep state and restore after wake up.

Put the ARC core into sleep state until next SoC wake event and continue execution after wake up where the application stopped.

If the library is built with `ENABLE_RESTORE_CONTEXT=1`, then this function will use the `arc_restore_addr` to save restore trap address which brings back the ARC CPU to the point where this function was called. This means that applications should refrain from using them.

This function calls `qm_ss_save_context` and `qm_ss_restore_context` in order to restore execution where it stopped. All power management transitions are done by [qm_power_soc_sleep\(\)](#).

Definition at line 141 of file `ss_power_states.c`.

References `qm_power_soc_set_ss_restore_flag()`, and `qm_power_soc_sleep()`.

4.52 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.

4.52.1 Detailed Description

Voltage Regulators Control.

4.52.2 Function Documentation

4.52.2.1 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

<i>in</i>	<i>mode</i>	Voltage Regulator mode.
-----------	-------------	-------------------------

Returns

Standard errno return type for QMSI.

Return values

<i>0</i>	on success.
<i>Negative</i>	errno for possible error codes.

Definition at line 45 of file vreg.c.

4.52.2.2 int vreg_host_set_mode (const vreg_mode_t mode)

Set Host Voltage Regulator mode.

Parameters

<i>in</i>	<i>mode</i>	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.

4.52.2.3 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 qm_power_soc_deep_sleep().

4.52.2.4 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 qm_power_soc_deep_sleep().

4.53 Syscalls

newlib syscalls implementation

Functions

- int [pico_printf](#) (const char *format,...)
This is an minimally useful subset of the POSIX printf() function.

4.53.1 Detailed Description

newlib syscalls implementation

4.53.2 Function Documentation

4.53.2.1 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 architectures '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_interrupt_router_regs.h>
```

5.1.1 Detailed Description

SS I2C Interrupt register map.

Definition at line 223 of file `qm_interrupt_router_regs.h`.

5.2 `int_ss_spi_reg_t` Struct Reference

SS SPI Interrupt register map.

```
#include <qm_interrupt_router_regs.h>
```

5.2.1 Detailed Description

SS SPI Interrupt register map.

Definition at line 231 of file `qm_interrupt_router_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 1409 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 1335 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 [cmp_en](#)
Comparator 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.

Note

There is no way to control comparator interrupts using this configuration struct: when a comparator is enabled and powered-up, it starts generating interrupts when proper input conditions are met; however, comparator interrupts can be masked at interrupt routing level.

Definition at line 28 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

<i>in</i>	<i>data</i>	Callback user data.
<i>in</i>	<i>status</i>	Comparator interrupt status.

Definition at line 40 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 41 of file qm_comparator.h.

Referenced by qm_ac_set_config().

5.5.2.3 uint32_t qm_ac_config_t::cmp_en

Comparator enable.

Definition at line 29 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 30 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 1069 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	<i>data</i>	The callback user data.
in	<i>error</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	ADC status.
in	<i>source</i>	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_aonc_reg_t Struct Reference

Always-on Counter 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.

5.9.1 Detailed Description

Always-on Counter Controller register map.

Definition at line 258 of file `qm_soc_regs.h`.

5.9.2 Field Documentation

5.9.2.1 QM_RW uint32_t qm_aonc_reg_t::aonc_cfg

Always-on counter enable.

Definition at line 260 of file `qm_soc_regs.h`.

5.9.2.2 QM_RW uint32_t qm_aonc_reg_t::aonc_cnt

Always-on counter register.

Definition at line 259 of file `qm_soc_regs.h`.

5.9.2.3 QM_RW uint32_t qm_aonc_reg_t::aonpt_cfg

Always-on periodic timer configuration register.

Definition at line 266 of file `qm_soc_regs.h`.

5.9.2.4 QM_RW uint32_t qm_aonc_reg_t::aonpt_cnt

Always-on periodic timer.

Definition at line 261 of file `qm_soc_regs.h`.

5.9.2.5 QM_RW uint32_t qm_aonc_reg_t::aonpt_ctrl

Always-on periodic timer control.

Definition at line 264 of file `qm_soc_regs.h`.

5.9.2.6 QM_RW uint32_t qm_aonc_reg_t::aonpt_stat

Always-on periodic timer status register.

Definition at line 263 of file `qm_soc_regs.h`.

5.10 qm_aonpt_config_t Struct Reference

QM 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.

5.10.1 Detailed Description

QM Always-on Periodic Timer configuration type.

Definition at line 40 of file qm_aon_counters.h.

5.10.2 Field Documentation

5.10.2.1 void(* qm_aonpt_config_t::callback)(void *data)

User callback.

Parameters

in	data	User defined data.
--------------------	----------------------	--------------------

Definition at line 49 of file qm_aon_counters.h.

Referenced by [qm_aonpt_set_config\(\)](#).

5.10.2.2 void* qm_aonpt_config_t::callback_data

Callback data.

Definition at line 50 of file qm_aon_counters.h.

Referenced by [qm_aonpt_set_config\(\)](#).

5.10.2.3 uint32_t qm_aonpt_config_t::count

Time to count down from in clock cycles.

Definition at line 41 of file qm_aon_counters.h.

Referenced by [qm_aonpt_set_config\(\)](#).

5.10.2.4 bool qm_aonpt_config_t::int_en

Enable/disable the interrupts.

Definition at line 42 of file qm_aon_counters.h.

Referenced by [qm_aonpt_set_config\(\)](#).

5.11 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.

5.11.1 Detailed Description

DMA channel register map.

Definition at line 1507 of file qm_soc_regs.h.

5.12 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.
- [qm_dma_transfer_type_t transfer_type](#)
DMA transfer type.
- 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.

5.12.1 Detailed Description

DMA channel configuration structure.

Definition at line 77 of file qm_dma.h.

5.12.2 Field Documentation

5.12.2.1 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	<i>callback_context</i>	DMA client context.
in	<i>len</i>	Data length transferred.
in	<i>error</i>	Error code.

Definition at line 109 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\(\)](#).

5.13 qm_dma_context_t Struct Reference

DMA context type.

```
#include <qm_soc_regs.h>
```


Data Fields

- uint32_t [misc_cfg_low](#)
DMA Configuration.
- uint32_t [ctrl_low](#)
Channel Control Lower.
- uint32_t [cfg_low](#)
Channel Configuration Lower.
- uint32_t [cfg_high](#)
Channel Configuration Upper.
- uint32_t [llp_low](#)
Channel Linked List Pointer.

5.13.1 Detailed Description

DMA context type.

Applications should not modify the content. This structure is only intended to be used by the `qm_dma_save_context` and `qm_dma_restore_context` functions.

Definition at line 1911 of file `qm_soc_regs.h`.

5.13.2 Field Documentation

5.13.2.1 uint32_t qm_dma_context_t::cfg_high

Channel Configuration Upper.

Definition at line 1915 of file `qm_soc_regs.h`.

Referenced by `qm_dma_restore_context()`, and `qm_dma_save_context()`.

5.13.2.2 uint32_t qm_dma_context_t::cfg_low

Channel Configuration Lower.

Definition at line 1914 of file `qm_soc_regs.h`.

Referenced by `qm_dma_restore_context()`, and `qm_dma_save_context()`.

5.13.2.3 uint32_t qm_dma_context_t::ctrl_low

Channel Control Lower.

Definition at line 1913 of file `qm_soc_regs.h`.

Referenced by `qm_dma_restore_context()`, and `qm_dma_save_context()`.

5.13.2.4 uint32_t qm_dma_context_t::llp_low

Channel Linked List Pointer.

Definition at line 1916 of file `qm_soc_regs.h`.

Referenced by `qm_dma_restore_context()`, and `qm_dma_save_context()`.

5.13.2.5 uint32_t qm_dma_context_t::misc_cfg_low

DMA Configuration.

Definition at line 1918 of file `qm_soc_regs.h`.

Referenced by `qm_dma_restore_context()`, and `qm_dma_save_context()`.

5.14 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.*

5.14.1 Detailed Description

DMA interrupt register map.

Definition at line 1583 of file `qm_soc_regs.h`.

5.15 `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.

5.15.1 Detailed Description

DMA miscellaneous register map.

Definition at line 1634 of file qm_soc_regs.h.

5.16 qm_dma_multi_transfer_t Struct Reference

DMA multiblock transfer configuration structure.

```
#include <qm_dma.h>
```

Data Fields

- uint32_t * [source_address](#)
First block source address.
- uint32_t * [destination_address](#)
First block destination address.
- uint16_t [block_size](#)
DMA block size, Min = 1, Max = 4095.
- uint16_t [num_blocks](#)
Number of contiguous blocks to be transfered.
- qm_dma_linked_list_item_t * [linked_list_first](#)
First block LLI descriptor or NULL (contiguous mode)

5.16.1 Detailed Description

DMA multiblock transfer configuration structure.

Definition at line 143 of file qm_dma.h.

5.16.2 Field Documentation

5.16.2.1 uint16_t qm_dma_multi_transfer_t::block_size

DMA block size, Min = 1, Max = 4095.

Definition at line 146 of file qm_dma.h.

Referenced by qm_dma_multi_transfer_set_config().

5.16.2.2 uint32_t* qm_dma_multi_transfer_t::destination_address

First block destination address.

Definition at line 145 of file qm_dma.h.

Referenced by qm_dma_multi_transfer_set_config().

5.16.2.3 uint16_t qm_dma_multi_transfer_t::num_blocks

Number of contiguous blocks to be transfered.

Definition at line 148 of file qm_dma.h.

Referenced by qm_dma_multi_transfer_set_config().

5.16.2.4 uint32_t* qm_dma_multi_transfer_t::source_address

First block source address.

Definition at line 144 of file qm_dma.h.

Referenced by qm_dma_multi_transfer_set_config().

5.17 qm_dma_transfer_t Struct Reference

DMA single block 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.

5.17.1 Detailed Description

DMA single block transfer configuration structure.

Definition at line 133 of file qm_dma.h.

5.17.2 Field Documentation

5.17.2.1 uint32_t qm_dma_transfer_t::block_size

DMA block size, Min = 1, Max = 4095.

Definition at line 134 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().

5.17.2.2 uint32_t* qm_dma_transfer_t::destination_address

DMA destination transfer address.

Definition at line 136 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().

5.17.2.3 uint32_t* qm_dma_transfer_t::source_address

DMA source transfer address.

Definition at line 135 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().

5.18 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.

5.18.1 Detailed Description

Flash configuration structure.

Definition at line 41 of file qm_flash.h.

5.18.2 Field Documentation

5.18.2.1 uint8_t qm_flash_config_t::us_count

Number of clocks in a microsecond.

Needed for program/erase operations.

Definition at line 48 of file qm_flash.h.

Referenced by qm_flash_set_config().

5.18.2.2 uint8_t qm_flash_config_t::wait_states

Read wait state.

Definition at line 42 of file qm_flash.h.

Referenced by `qm_flash_set_config()`.

5.18.2.3 `qm_flash_disable_t qm_flash_config_t::write_disable`

Write Disable.

When this is set, only a reset will enable writes to Flash again.

Definition at line 55 of file `qm_flash.h`.

Referenced by `qm_flash_set_config()`.

5.19 `qm_flash_context_t` Struct Reference

Flash context type.

```
#include <qm_soc_regs.h>
```

Data Fields

- `uint32_t tmg_ctrl`
Flash Timing Control Register.
- `uint32_t ctrl`
Control Register.

5.19.1 Detailed Description

Flash context type.

Applications should not modify the content. This structure is only intended to be used by the `qm_flash_save_context` and `qm_flash_restore_context` functions.

Definition at line 1466 of file `qm_soc_regs.h`.

5.19.2 Field Documentation

5.19.2.1 `uint32_t qm_flash_context_t::ctrl`

Control Register.

Definition at line 1470 of file `qm_soc_regs.h`.

Referenced by `qm_flash_restore_context()`, and `qm_flash_save_context()`.

5.19.2.2 `uint32_t qm_flash_context_t::tmg_ctrl`

Flash Timing Control Register.

Definition at line 1468 of file `qm_soc_regs.h`.

Referenced by `qm_flash_restore_context()`, and `qm_flash_save_context()`.

5.20 `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.

5.20.1 Detailed Description

Flash register map.

Definition at line 1144 of file `qm_soc_regs.h`.

5.20.2 Field Documentation

5.20.2.1 QM_RW uint32_t qm_flash_reg_t::ctrl

CTRL.

Definition at line 1151 of file `qm_soc_regs.h`.

Referenced by `qm_flash_mass_erase()`, `qm_flash_restore_context()`, `qm_flash_save_context()`, and `qm_flash_set_config()`.

5.20.2.2 QM_RW uint32_t qm_flash_reg_t::flash_stts

FLASH_STTS.

Definition at line 1150 of file `qm_soc_regs.h`.

Referenced by `qm_flash_mass_erase()`, `qm_flash_page_erase()`, `qm_flash_page_update()`, `qm_flash_page_write()`, and `qm_flash_word_write()`.

5.20.2.3 QM_RW uint32_t qm_flash_reg_t::flash_wr_ctrl

FLASH_WR_CTRL.

Definition at line 1148 of file `qm_soc_regs.h`.

Referenced by `qm_flash_page_erase()`, `qm_flash_page_update()`, `qm_flash_page_write()`, and `qm_flash_word_write()`.

5.20.2.4 QM_RW uint32_t qm_flash_reg_t::flash_wr_data

FLASH_WR_DATA.

Definition at line 1149 of file qm_soc_regs.h.

Referenced by qm_flash_page_update(), qm_flash_page_write(), and qm_flash_word_write().

5.20.2.5 QM_RW uint32_t qm_flash_reg_t::fpr_rd_cfg

4 FPR_RD_CFG registers

4 FPR_RD_CFG registers.

Definition at line 1152 of file qm_soc_regs.h.

Referenced by qm_fpr_restore_context(), qm_fpr_save_context(), and qm_fpr_set_config().

5.20.2.6 QM_RW uint32_t qm_flash_reg_t::mpr_vsts

Protection Status Register.

Definition at line 1155 of file qm_soc_regs.h.

5.20.2.7 QM_RW uint32_t qm_flash_reg_t::mpr_wr_cfg

Flash Write Protection Control Register.

Definition at line 1154 of file qm_soc_regs.h.

5.20.2.8 QM_RW uint32_t qm_flash_reg_t::rom_wr_ctrl

ROM_WR_CTRL.

Definition at line 1146 of file qm_soc_regs.h.

Referenced by qm_flash_page_erase(), qm_flash_page_update(), qm_flash_page_write(), and qm_flash_word_write().

5.20.2.9 QM_RW uint32_t qm_flash_reg_t::rom_wr_data

ROM_WR_DATA.

Definition at line 1147 of file qm_soc_regs.h.

Referenced by qm_flash_page_update(), qm_flash_page_write(), and qm_flash_word_write().

5.20.2.10 QM_RW uint32_t qm_flash_reg_t::tmg_ctrl

TMG_CTRL.

Definition at line 1145 of file qm_soc_regs.h.

Referenced by qm_flash_restore_context(), qm_flash_save_context(), and qm_flash_set_config().

5.21 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.

5.21.1 Detailed Description

Flash Protection Region configuration structure.

Definition at line 69 of file qm_fpr.h.

5.21.2 Field Documentation

5.21.2.1 qm_fpr_read_allow_t qm_fpr_config_t::allow_agents

Per-agent read enable bitmask.

Definition at line 71 of file qm_fpr.h.

Referenced by [qm_fpr_set_config\(\)](#).

5.21.2.2 qm_fpr_en_t qm_fpr_config_t::en_mask

Enable/lock bitmask.

Definition at line 70 of file qm_fpr.h.

Referenced by [qm_fpr_set_config\(\)](#).

5.21.2.3 uint8_t qm_fpr_config_t::low_bound

1KB-aligned lower Flash phys addr.

Definition at line 73 of file qm_fpr.h.

Referenced by [qm_fpr_set_config\(\)](#).

5.21.2.4 uint8_t qm_fpr_config_t::up_bound

1KB-aligned upper Flash phys addr.

Definition at line 72 of file qm_fpr.h.

Referenced by [qm_fpr_set_config\(\)](#).

5.22 qm_fpr_context_t Struct Reference

FPR context type.

```
#include <qm_soc_regs.h>
```

Data Fields

- uint32_t [fpr_rd_cfg](#) [QM_FPR_NUM]
Flash Protection Region Read Control Register.

5.22.1 Detailed Description

FPR context type.

Applications should not modify the content. This structure is only intended to be used by the `qm_fpr_save_context` and `qm_fpr_restore_context` functions.

Definition at line 1591 of file `qm_soc_regs.h`.

5.22.2 Field Documentation

5.22.2.1 `uint32_t qm_fpr_context_t::fpr_rd_cfg[QM_FPR_NUM]`

Flash Protection Region Read Control Register.

Definition at line 1593 of file `qm_soc_regs.h`.

Referenced by `qm_fpr_restore_context()`, and `qm_fpr_save_context()`.

5.23 `qm_gpio_context_t` Struct Reference

GPIO context type.

```
#include <qm_soc_regs.h>
```

Data Fields

- `uint32_t gpio_swporta_dr`
Port A Data.
- `uint32_t gpio_swporta_ddr`
Port A Data Direction.
- `uint32_t gpio_swporta_ctl`
Port A Data Source.
- `uint32_t gpio_inten`
Interrupt Enable.
- `uint32_t gpio_intmask`
Interrupt Mask.
- `uint32_t gpio_inttype_level`
Interrupt Type.
- `uint32_t gpio_int_polarity`
Interrupt Polarity.
- `uint32_t gpio_debounce`
Debounce Enable.
- `uint32_t gpio_ls_sync`
Synchronization Level.
- `uint32_t gpio_int_bothedge`
Interrupt both edge type.

5.23.1 Detailed Description

GPIO context type.

Application should not modify the content. This structure is only intended to be used by the `qm_gpio_save_context` and `qm_gpio_restore_context` functions.

Definition at line 1399 of file `qm_soc_regs.h`.

5.23.2 Field Documentation

5.23.2.1 uint32_t qm_gpio_context_t::gpio_debounce

Debounce Enable.

Definition at line 1407 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.2 uint32_t qm_gpio_context_t::gpio_int_bothedge

Interrupt both edge type.

Definition at line 1409 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.3 uint32_t qm_gpio_context_t::gpio_int_polarity

Interrupt Polarity.

Definition at line 1406 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.4 uint32_t qm_gpio_context_t::gpio_inten

Interrupt Enable.

Definition at line 1403 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.5 uint32_t qm_gpio_context_t::gpio_intmask

Interrupt Mask.

Definition at line 1404 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.6 uint32_t qm_gpio_context_t::gpio_inttype_level

Interrupt Type.

Definition at line 1405 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.7 uint32_t qm_gpio_context_t::gpio_ls_sync

Synchronization Level.

Definition at line 1408 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.8 uint32_t qm_gpio_context_t::gpio_swporta_ctl

Port A Data Source.

Definition at line 1402 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.9 uint32_t qm_gpio_context_t::gpio_swporta_ddr

Port A Data Direction.

Definition at line 1401 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.23.2.10 uint32_t qm_gpio_context_t::gpio_swporta_dr

Port A Data.

Definition at line 1400 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), and qm_gpio_save_context().

5.24 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.

5.24.1 Detailed Description

GPIO port configuration type.

Each bit in the registers control a GPIO pin.

Definition at line 32 of file qm_gpio.h.

5.24.2 Field Documentation

5.24.2.1 void(* qm_gpio_port_config_t::callback)(void *data, uint32_t int_status)

Transfer callback.

Parameters

in	<i>data</i>	Callback user data.
in	<i>int_status</i>	GPIO interrupt status.

Definition at line 46 of file qm_gpio.h.

Referenced by qm_gpio_set_config().

5.24.2.2 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().

5.24.2.3 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().

5.24.2.4 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().

5.24.2.5 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().

5.24.2.6 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().

5.24.2.7 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().

5.24.2.8 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().

5.25 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.

5.25.1 Detailed Description

GPIO register map.

Definition at line 1017 of file `qm_soc_regs.h`.

5.25.2 Field Documentation

5.25.2.1 QM_RW uint32_t qm_gpio_reg_t::gpio_config_reg1

GPIO Configuration Register 1.

Definition at line 1036 of file `qm_soc_regs.h`.

5.25.2.2 QM_RW uint32_t qm_gpio_reg_t::gpio_config_reg2

GPIO Configuration Register 2.

Definition at line 1035 of file qm_soc_regs.h.

5.25.2.3 QM_RW uint32_t qm_gpio_reg_t::gpio_debounce

Debounce Enable.

Definition at line 1027 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), qm_gpio_save_context(), and qm_gpio_set_config().

5.25.2.4 QM_RW uint32_t qm_gpio_reg_t::gpio_ext_porta

Port A External Port.

Definition at line 1029 of file qm_soc_regs.h.

5.25.2.5 QM_RW uint32_t qm_gpio_reg_t::gpio_id_code

GPIO ID code.

Definition at line 1032 of file qm_soc_regs.h.

5.25.2.6 QM_RW uint32_t qm_gpio_reg_t::gpio_int_bothedge

Interrupt both edge type.

Definition at line 1033 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), qm_gpio_save_context(), and qm_gpio_set_config().

5.25.2.7 QM_RW uint32_t qm_gpio_reg_t::gpio_int_polarity

Interrupt Polarity.

Definition at line 1024 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), qm_gpio_save_context(), and qm_gpio_set_config().

5.25.2.8 QM_RW uint32_t qm_gpio_reg_t::gpio_inten

Interrupt Enable.

Definition at line 1021 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), qm_gpio_save_context(), and qm_gpio_set_config().

5.25.2.9 QM_RW uint32_t qm_gpio_reg_t::gpio_intmask

Interrupt Mask.

Definition at line 1022 of file qm_soc_regs.h.

Referenced by qm_gpio_restore_context(), qm_gpio_save_context(), and qm_gpio_set_config().

5.25.2.10 QM_RW uint32_t qm_gpio_reg_t::gpio_intstatus

Interrupt Status.

Definition at line 1025 of file qm_soc_regs.h.

5.25.2.11 QM_RW uint32_t qm_gpio_reg_t::gpio_inttype_level

Interrupt Type.

Definition at line 1023 of file qm_soc_regs.h.

Referenced by `qm_gpio_restore_context()`, `qm_gpio_save_context()`, and `qm_gpio_set_config()`.

5.25.2.12 QM_RW uint32_t qm_gpio_reg_t::gpio_ls_sync

Synchronization Level.

Definition at line 1031 of file `qm_soc_regs.h`.

Referenced by `qm_gpio_restore_context()`, `qm_gpio_save_context()`, and `qm_gpio_set_config()`.

5.25.2.13 QM_RW uint32_t qm_gpio_reg_t::gpio_porta_eoi

Clear Interrupt.

Definition at line 1028 of file `qm_soc_regs.h`.

5.25.2.14 QM_RW uint32_t qm_gpio_reg_t::gpio_raw_intstatus

Raw Interrupt Status.

Definition at line 1026 of file `qm_soc_regs.h`.

5.25.2.15 QM_RW uint32_t qm_gpio_reg_t::gpio_swporta_ddr

Port A Data Direction.

Definition at line 1019 of file `qm_soc_regs.h`.

Referenced by `qm_gpio_restore_context()`, `qm_gpio_save_context()`, and `qm_gpio_set_config()`.

5.25.2.16 QM_RW uint32_t qm_gpio_reg_t::gpio_swporta_dr

Port A Data.

Definition at line 1018 of file `qm_soc_regs.h`.

Referenced by `qm_gpio_restore_context()`, and `qm_gpio_save_context()`.

5.25.2.17 QM_RW uint32_t qm_gpio_reg_t::gpio_ver_id_code

GPIO Component Version.

Definition at line 1034 of file `qm_soc_regs.h`.

5.26 qm_i2c_config_t Struct Reference

I2C configuration type.

```
#include <qm_i2c.h>
```

Data Fields

- [qm_i2c_speed_t speed](#)
Standard, fast or fast plus mode.
- [qm_i2c_addr_t address_mode](#)
7 bit or 10 bit addressing.
- [qm_i2c_mode_t mode](#)
Master or slave mode.
- [uint16_t slave_addr](#)
I2C address when in slave mode.
- [qm_i2c_slave_stop_t stop_detect_behaviour](#)
Slave stop detect behaviour.

5.26.1 Detailed Description

I2C configuration type.

Definition at line 111 of file qm_i2c.h.

5.26.2 Field Documentation

5.26.2.1 qm_i2c_addr_t qm_i2c_config_t::address_mode

7 bit or 10 bit addressing.

Definition at line 113 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.26.2.2 qm_i2c_mode_t qm_i2c_config_t::mode

Master or slave mode.

Definition at line 114 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.26.2.3 uint16_t qm_i2c_config_t::slave_addr

I2C address when in slave mode.

Definition at line 115 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.26.2.4 qm_i2c_speed_t qm_i2c_config_t::speed

Standard, fast or fast plus mode.

Definition at line 112 of file qm_i2c.h.

Referenced by qm_i2c_set_config().

5.27 qm_i2c_context_t Struct Reference

I2C context to be saved between sleep/resume.

```
#include <qm_soc_regs.h>
```

Data Fields

- uint32_t [con](#)
Control Register.
- uint32_t [sar](#)
Slave Address.
- uint32_t [ss_scl_hcnt](#)
Standard Speed Clock SCL High Count.
- uint32_t [ss_scl_lcnt](#)
Standard Speed Clock SCL Low Count.
- uint32_t [fs_scl_hcnt](#)
Fast Speed Clock SCL High Count.
- uint32_t [fs_scl_lcnt](#)
Fast Speed I2C Clock SCL Low Count.

- uint32_t [enable](#)
Enable.
- uint32_t [fs_spklen](#)
SS and FS Spike Suppression Limit.
- uint32_t [ic_intr_mask](#)
I2C Interrupt Mask.
- uint32_t [tx_tl](#)
Receive FIFO threshold register.

5.27.1 Detailed Description

I2C context to be saved between sleep/resume.

Application should not modify the content. This structure is only intended to be used by the `qm_i2c_save_context` and `qm_i2c_restore_context` functions.

Definition at line 1260 of file `qm_soc_regs.h`.

5.27.2 Field Documentation

5.27.2.1 uint32_t qm_i2c_context_t::con

Control Register.

Definition at line 1261 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_restore_context()`, and `qm_i2c_save_context()`.

5.27.2.2 uint32_t qm_i2c_context_t::enable

Enable.

Definition at line 1267 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_restore_context()`, and `qm_i2c_save_context()`.

5.27.2.3 uint32_t qm_i2c_context_t::fs_scl_hcnt

Fast Speed Clock SCL High Count.

Definition at line 1265 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_restore_context()`, and `qm_i2c_save_context()`.

5.27.2.4 uint32_t qm_i2c_context_t::fs_scl_lcnt

Fast Speed I2C Clock SCL Low Count.

Definition at line 1266 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_restore_context()`, and `qm_i2c_save_context()`.

5.27.2.5 uint32_t qm_i2c_context_t::fs_spklen

SS and FS Spike Suppression Limit.

Definition at line 1268 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_restore_context()`, and `qm_i2c_save_context()`.

5.27.2.6 uint32_t qm_i2c_context_t::ic_intr_mask

I2C Interrupt Mask.

Definition at line 1269 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), and qm_i2c_save_context().

5.27.2.7 uint32_t qm_i2c_context_t::sar

Slave Address.

Definition at line 1262 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), and qm_i2c_save_context().

5.27.2.8 uint32_t qm_i2c_context_t::ss_scl_hcnt

Standard Speed Clock SCL High Count.

Definition at line 1263 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), and qm_i2c_save_context().

5.27.2.9 uint32_t qm_i2c_context_t::ss_scl_lcnt

Standard Speed Clock SCL Low Count.

Definition at line 1264 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), and qm_i2c_save_context().

5.27.2.10 uint32_t qm_i2c_context_t::tx_tl

Receive FIFO threshold register.

Definition at line 1271 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), and qm_i2c_save_context().

5.28 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.

5.28.1 Detailed Description

I2C register map.

Definition at line 864 of file qm_soc_regs.h.

5.28.2 Field Documentation

5.28.2.1 QM_RW uint32_t qm_i2c_reg_t::ic_ack_general_call

General Call Ack.

Definition at line 909 of file qm_soc_regs.h.

5.28.2.2 QM_RW uint32_t qm_i2c_reg_t::ic_clr_activity

Clear ACTIVITY Interrupt.

Definition at line 894 of file qm_soc_regs.h.

5.28.2.3 QM_RW uint32_t qm_i2c_reg_t::ic_clr_gen_call

Clear GEN_CALL Interrupt.

Definition at line 897 of file qm_soc_regs.h.

5.28.2.4 QM_RW uint32_t qm_i2c_reg_t::ic_clr_intr

Clear Combined and Individual Interrupt.

Definition at line 887 of file qm_soc_regs.h.

5.28.2.5 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rd_req

Clear RD_REQ Interrupt.

Definition at line 891 of file qm_soc_regs.h.

5.28.2.6 QM_RW uint32_t qm_i2c_reg_t::ic_clr_restart_det

clear the RESTART_DET interrupt.

Definition at line 914 of file qm_soc_regs.h.

5.28.2.7 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rx_done

Clear RX_DONE Interrupt.

Definition at line 893 of file qm_soc_regs.h.

5.28.2.8 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rx_over

Clear RX_OVER Interrupt.

Definition at line 889 of file qm_soc_regs.h.

5.28.2.9 QM_RW uint32_t qm_i2c_reg_t::ic_clr_rx_under

Clear RX_UNDER Interrupt.

Definition at line 888 of file qm_soc_regs.h.

5.28.2.10 QM_RW uint32_t qm_i2c_reg_t::ic_clr_start_det

Clear START_DET Interrupt.

Definition at line 896 of file qm_soc_regs.h.

5.28.2.11 QM_RW uint32_t qm_i2c_reg_t::ic_clr_stop_det

Clear STOP_DET Interrupt.

Definition at line 895 of file qm_soc_regs.h.

5.28.2.12 QM_RW uint32_t qm_i2c_reg_t::ic_clr_tx_abrt

Clear TX_ABRT Interrupt.

Definition at line 892 of file qm_soc_regs.h.

Referenced by qm_i2c_master_read(), and qm_i2c_master_write().

5.28.2.13 QM_RW uint32_t qm_i2c_reg_t::ic_clr_tx_over

Clear TX_OVER Interrupt.

Definition at line 890 of file qm_soc_regs.h.

5.28.2.14 QM_RW uint32_t qm_i2c_reg_t::ic_comp_param_1

Configuration Parameters.

Definition at line 916 of file qm_soc_regs.h.

5.28.2.15 QM_RW uint32_t qm_i2c_reg_t::ic_comp_type

Component Type.

Definition at line 918 of file qm_soc_regs.h.

5.28.2.16 QM_RW uint32_t qm_i2c_reg_t::ic_comp_version

Component Version.

Definition at line 917 of file qm_soc_regs.h.

5.28.2.17 QM_RW uint32_t qm_i2c_reg_t::ic_con

Control Register.

Definition at line 865 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), qm_i2c_save_context(), qm_i2c_set_config(), and qm_i2c_set_speed().

5.28.2.18 QM_RW uint32_t qm_i2c_reg_t::ic_data_cmd

Data Buffer and Command.

Definition at line 869 of file qm_soc_regs.h.

Referenced by qm_i2c_master_read(), and qm_i2c_master_write().

5.28.2.19 QM_RW uint32_t qm_i2c_reg_t::ic_dma_cr

SDA Setup.

DMA Control Register for Tx and Rx Handshaking Interface.

Definition at line 905 of file qm_soc_regs.h.

5.28.2.20 QM_RW uint32_t qm_i2c_reg_t::ic_dma_rdlr

I2C Receive Data Level Register.

Definition at line 907 of file qm_soc_regs.h.

5.28.2.21 QM_RW uint32_t qm_i2c_reg_t::ic_dma_tdlr

DMA Transmit Data Level Register.

Definition at line 906 of file qm_soc_regs.h.

5.28.2.22 QM_RW uint32_t qm_i2c_reg_t::ic_enable

Enable.

Definition at line 898 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), and qm_i2c_save_context().

5.28.2.23 QM_RW uint32_t qm_i2c_reg_t::ic_enable_status

Enable Status.

Definition at line 910 of file qm_soc_regs.h.

5.28.2.24 QM_RW uint32_t qm_i2c_reg_t::ic_fs_scl_hcnt

Fast Speed Clock SCL High Count.

Definition at line 874 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), qm_i2c_save_context(), qm_i2c_set_config(), and qm_i2c_set_speed().

5.28.2.25 QM_RW uint32_t qm_i2c_reg_t::ic_fs_scl_lcnt

Fast Speed I2C Clock SCL Low Count.

Definition at line 876 of file qm_soc_regs.h.

Referenced by `qm_i2c_restore_context()`, `qm_i2c_save_context()`, `qm_i2c_set_config()`, and `qm_i2c_set_speed()`.

5.28.2.26 QM_RW uint32_t qm_i2c_reg_t::ic_fs_spklen

SS and FS Spike Suppression Limit.

Definition at line 911 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_restore_context()`, `qm_i2c_save_context()`, `qm_i2c_set_config()`, and `qm_i2c_set_speed()`.

5.28.2.27 QM_RW uint32_t qm_i2c_reg_t::ic_hs_maddr

High Speed Master ID.

Definition at line 868 of file `qm_soc_regs.h`.

5.28.2.28 QM_RW uint32_t qm_i2c_reg_t::ic_hs_scl_hcnt

High Speed I2C Clock SCL High Count.

Definition at line 878 of file `qm_soc_regs.h`.

5.28.2.29 QM_RW uint32_t qm_i2c_reg_t::ic_hs_scl_lcmt

High Speed I2C Clock SCL Low Count.

Definition at line 880 of file `qm_soc_regs.h`.

5.28.2.30 QM_RW uint32_t qm_i2c_reg_t::ic_hs_spklen

HS spike suppression limit.

Definition at line 912 of file `qm_soc_regs.h`.

5.28.2.31 QM_RW uint32_t qm_i2c_reg_t::ic_intr_mask

Interrupt Mask.

Definition at line 882 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_master_irq_transfer()`, `qm_i2c_restore_context()`, `qm_i2c_save_context()`, `qm_i2c_set_config()`, and `qm_i2c_slave_irq_transfer()`.

5.28.2.32 QM_RW uint32_t qm_i2c_reg_t::ic_intr_stat

Interrupt Status.

Definition at line 881 of file `qm_soc_regs.h`.

5.28.2.33 QM_RW uint32_t qm_i2c_reg_t::ic_raw_intr_stat

Raw Interrupt Status.

Definition at line 883 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_master_read()`.

5.28.2.34 QM_RW uint32_t qm_i2c_reg_t::ic_rx_tl

Receive FIFO Threshold Level.

Definition at line 884 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_master_irq_transfer()`, `qm_i2c_restore_context()`, `qm_i2c_save_context()`, and `qm_i2c_slave_irq_transfer()`.

5.28.2.35 QM_RW uint32_t qm_i2c_reg_t::ic_rxflr

Receive FIFO Level.

Definition at line 901 of file qm_soc_regs.h.

5.28.2.36 QM_RW uint32_t qm_i2c_reg_t::ic_sar

Slave Address.

Definition at line 867 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), qm_i2c_save_context(), and qm_i2c_set_config().

5.28.2.37 QM_RW uint32_t qm_i2c_reg_t::ic_sda_hold

SDA Hold.

Definition at line 902 of file qm_soc_regs.h.

5.28.2.38 QM_RW uint32_t qm_i2c_reg_t::ic_sda_setup

SDA Setup.

Definition at line 908 of file qm_soc_regs.h.

5.28.2.39 QM_RW uint32_t qm_i2c_reg_t::ic_ss_scl_hcnt

Standard Speed Clock SCL High Count.

Definition at line 871 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), qm_i2c_save_context(), qm_i2c_set_config(), and qm_i2c_set_speed().

5.28.2.40 QM_RW uint32_t qm_i2c_reg_t::ic_ss_scl_lcnt

Standard Speed Clock SCL Low Count.

Definition at line 873 of file qm_soc_regs.h.

Referenced by qm_i2c_restore_context(), qm_i2c_save_context(), qm_i2c_set_config(), and qm_i2c_set_speed().

5.28.2.41 QM_RW uint32_t qm_i2c_reg_t::ic_status

Status.

Definition at line 899 of file qm_soc_regs.h.

Referenced by qm_i2c_get_status(), qm_i2c_master_read(), and qm_i2c_master_write().

5.28.2.42 QM_RW uint32_t qm_i2c_reg_t::ic_tar

Master Target Address.

Definition at line 866 of file qm_soc_regs.h.

Referenced by qm_i2c_master_irq_transfer(), qm_i2c_master_read(), and qm_i2c_master_write().

5.28.2.43 QM_RW uint32_t qm_i2c_reg_t::ic_tx_abrt_source

Transmit Abort Source.

Definition at line 903 of file qm_soc_regs.h.

Referenced by qm_i2c_get_status(), qm_i2c_master_read(), and qm_i2c_master_write().

5.28.2.44 QM_RW uint32_t qm_i2c_reg_t::ic_tx_tl

Transmit FIFO Threshold Level.

Definition at line 885 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_master_irq_transfer()`, `qm_i2c_restore_context()`, `qm_i2c_save_context()`, and `qm_i2c_slave_irq_transfer()`.

5.28.2.45 QM_RW uint32_t qm_i2c_reg_t::ic_txflr

Transmit FIFO Level.

Definition at line 900 of file `qm_soc_regs.h`.

5.29 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`
Master: Generate STOP.
- `void(* callback)(void *data, int rc, qm_i2c_status_t status, uint32_t len)`
Transfer callback.
- `void * callback_data`
User callback data.

5.29.1 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.

Definition at line 130 of file `qm_i2c.h`.

5.29.2 Field Documentation

5.29.2.1 void(* qm_i2c_transfer_t::callback)(void *data, int rc, qm_i2c_status_t status, uint32_t len)

Transfer callback.

Called after all data is transmitted/received or if the driver detects an error during the I2C transfer.

In slave mode, `qm_i2c_slave_irq_transfer_update` shall be called from this callback to update transfer buffers when receiving a `QM_I2C_RX_FULL` or `QM_I2C_TX_EMPTY` status. If the update function is not called with these statuses, the driver will drop every new data received or send dummy data (0x00) for each byte until next bus start.

Parameters

in	<i>data</i>	User defined data.
in	<i>rc</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	I2C status.
in	<i>len</i>	Length of the transfer if successful, 0 otherwise.

Definition at line 161 of file qm_i2c.h.

5.29.2.2 void* qm_i2c_transfer_t::callback_data

User callback data.

Definition at line 163 of file qm_i2c.h.

5.29.2.3 uint8_t* qm_i2c_transfer_t::rx

Read data.

Definition at line 133 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.29.2.4 uint32_t qm_i2c_transfer_t::rx_len

Read buffer length.

Definition at line 134 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer(), and qm_i2c_master_irq_transfer().

5.29.2.5 bool qm_i2c_transfer_t::stop

Master: Generate STOP.

Slave: stop at the end of transaction.

Definition at line 140 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.29.2.6 uint8_t* qm_i2c_transfer_t::tx

Write data.

Definition at line 131 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.29.2.7 uint32_t qm_i2c_transfer_t::tx_len

Write data length.

Definition at line 132 of file qm_i2c.h.

Referenced by qm_i2c_master_dma_transfer().

5.30 qm_i2s_buffer_link Struct Reference

Ring buffer link definition.

```
#include <qm_i2s.h>
```

Data Fields

- uint32_t * [buff](#)

Pointer to the buffer base address.

- `qm_dma_llt_item_t * dma_link_head`

Pointer to the head of the DMA link list for the buffer link.

- `struct qm_i2s_buffer_link * next`

Link list item's pointer to the next link.

5.30.1 Detailed Description

Ring buffer link definition.

Definition at line 157 of file `qm_i2s.h`.

5.30.2 Field Documentation

5.30.2.1 `uint32_t* qm_i2s_buffer_link::buff`

Pointer to the buffer base address.

Definition at line 158 of file `qm_i2s.h`.

5.30.2.2 `qm_dma_llt_item_t* qm_i2s_buffer_link::dma_link_head`

Pointer to the head of the DMA link list for the buffer link.

Definition at line 160 of file `qm_i2s.h`.

5.30.2.3 `struct qm_i2s_buffer_link* qm_i2s_buffer_link::next`

Link list item's pointer to the next link.

Definition at line 162 of file `qm_i2s.h`.

5.31 `qm_i2s_channel_cfg_data_t` Struct Reference

I2S controller configuration.

```
#include <qm_i2s.h>
```

Data Fields

- `qm_dma_channel_id_t dma_channel`
DMA controller Channel ID.
- `qm_i2s_audio_stream_t audio_stream`
I2S audio stream identifier.
- `qm_i2s_audio_format_t audio_format`
I2S audio format configuration.
- `qm_i2s_master_slave_t master_slave`
I2S Master or Slave configuration.
- `qm_i2s_sample_resolution_t sample_resolution`
Sampling resolution.
- `qm_i2s_audio_rate_t audio_rate`
Audio rate.
- `qm_i2s_buffer_mode_t audio_buff_cfg`
Terminated link list or ring buffer configuration.
- `uint32_t block_size`

- DMA block size configuration: The source and destination transfer width is 32-bits for I2S.*
- uint32_t [num_dma_links](#)
Total number of DMA links.
 - qm_dma_llt_item_t * [dma_link](#)
Pointer to the DMA link list.
 - uint32_t [num_buffer_links](#)
Number of buffer links in the ring buffer.
 - [qm_i2s_buffer_link_t](#) * [buffer_link](#)
Ring buffer head pointer.
 - uint32_t [num_dma_link_per_buffer](#)
Number of DMA links per audio buffer.
 - uint32_t [buffer_len](#)
Length of a buffer in the ring buffer.
 - uint32_t [afull_thresh](#)
TX/RX almost full threshold.
 - uint32_t [aempty_thresh](#)
TX/RX almost empty threshold.
 - void(* [callback](#))(void *data, [qm_i2s_status_t](#) i2s_status)
Transfer callback.
 - void * [callback_data](#)
Callback user data.

5.31.1 Detailed Description

I2S controller configuration.

Driver instantiates one of these with given parameters for each I2S channel configured using the "qm_i2s_set_channel_config" function.

Definition at line 170 of file qm_i2s.h.

5.31.2 Field Documentation

5.31.2.1 uint32_t qm_i2s_channel_cfg_data_t::aempty_thresh

TX/RX almost empty threshold.

Definition at line 197 of file qm_i2s.h.

5.31.2.2 uint32_t qm_i2s_channel_cfg_data_t::afull_thresh

TX/RX almost full threshold.

Definition at line 196 of file qm_i2s.h.

5.31.2.3 uint32_t qm_i2s_channel_cfg_data_t::block_size

DMA block size configuration: The source and destination transfer width is 32-bits for I2S.

Min = 1, Max = see DMA configuration

Definition at line 187 of file qm_i2s.h.

5.31.2.4 uint32_t qm_i2s_channel_cfg_data_t::buffer_len

Length of a buffer in the ring buffer.

Definition at line 195 of file qm_i2s.h.

5.31.2.5 `qm_i2s_buffer_link_t* qm_i2s_channel_cfg_data_t::buffer_link`

Ring buffer head pointer.

Definition at line 192 of file `qm_i2s.h`.

5.31.2.6 `void(* qm_i2s_channel_cfg_data_t::callback)(void *data, qm_i2s_status_t i2s_status)`

Transfer callback.

Parameters

<code>in</code>	<code>data</code>	Callback user data
<code>in</code>	<code>i2s_status</code>	I2S status

Definition at line 205 of file `qm_i2s.h`.

5.31.2.7 `void* qm_i2s_channel_cfg_data_t::callback_data`

Callback user data.

Definition at line 206 of file `qm_i2s.h`.

5.31.2.8 `uint32_t qm_i2s_channel_cfg_data_t::num_buffer_links`

Number of buffer links in the ring buffer.

Definition at line 191 of file `qm_i2s.h`.

5.31.2.9 `uint32_t qm_i2s_channel_cfg_data_t::num_dma_link_per_buffer`

Number of DMA links per audio buffer.

Definition at line 194 of file `qm_i2s.h`.

5.31.2.10 `uint32_t qm_i2s_channel_cfg_data_t::num_dma_links`

Total number of DMA links.

Definition at line 188 of file `qm_i2s.h`.

5.32 `qm_i2s_clock_cfg_data_t` Struct Reference

I2S Clock Configuration.

```
#include <qm_i2s.h>
```

Data Fields

- [`qm_i2s_clk_src_t i2s_clock_sel`](#)
Select Internal NCO or external reference clock for I2S block.
- `uint32_t i2s_clk_freq`
Used by the driver when configuring I2S reference clock from the NCO.
- `bool i2s_mclk_output_en`
Enable MCLK output.
- `uint32_t i2s_mclk_divisor`
MCLK output is generated from the Internal NCO output, normally @24.576Mhz, using a divisor.
- `uint32_t i2s_ext_clk_freq`
Informs the driver of frequency of external clock when it is enabled.

5.32.1 Detailed Description

I2S Clock Configuration.

Definition at line 212 of file qm_i2s.h.

5.32.2 Field Documentation

5.32.2.1 uint32_t qm_i2s_clock_cfg_data_t::i2s_mclk_divisor

MCLK output is generated from the Internal NCO output, normally @24.576Mhz, using a divisor.

Valid divisor range is 0 to 4095. E.g. with a 48kHz data rate, and NCO = 24.576Mhz, then required MCLK = $256 * \text{data rate} = 12.288\text{MHz}$ MCLK divisor = $2 (24.576\text{Mhz}/2 = 12.288\text{MHz})$

Definition at line 230 of file qm_i2s.h.

5.32.2.2 bool qm_i2s_clock_cfg_data_t::i2s_mclk_output_en

Enable MCLK output.

Definition at line 221 of file qm_i2s.h.

5.33 qm_interrupt_router_reg_t Struct Reference

Interrupt register map.

```
#include <qm_interrupt_router_regs.h>
```

Data Fields

- QM_RW uint32_t [i2c_master_0_int_mask](#)
I2C Master 0, Mask 0.
- QM_RW uint32_t [spi_master_0_int_mask](#)
SPI Master 0, Mask 3.
- QM_RW uint32_t [spi_slave_0_int_mask](#)
SPI Slave 0, Mask 5.
- QM_RW uint32_t [uart_0_int_mask](#)
UART 0, Mask 6.
- QM_RW uint32_t [uart_1_int_mask](#)
UART 1, Mask 7.
- QM_RW uint32_t [gpio_0_int_mask](#)
GPIO 0, Mask 9.
- QM_RW uint32_t [timer_0_int_mask](#)
Timer 0, Mask 10.
- QM_RW uint32_t [rtc_0_int_mask](#)
RTC 0, Mask 12.
- QM_RW uint32_t [wdt_0_int_mask](#)
WDT 0, Mask 13.
- QM_RW uint32_t [dma_0_int_0_mask](#)
DMA 0 int 0, Mask 14.
- QM_RW uint32_t [dma_0_int_1_mask](#)
DMA 0 int 1, Mask 15.
- QM_RW uint32_t [comparator_0_host_halt_int_mask](#)
Comparator 0 Host halt, Mask 24.

- QM_RW uint32_t [comparator_0_host_int_mask](#)
Comparator 0 Host, Mask 26.
- QM_RW uint32_t [host_bus_error_int_mask](#)
Host bus error, Mask 27.
- QM_RW uint32_t [dma_0_error_int_mask](#)
DMA 0 Error, Mask 28.
- QM_RW uint32_t [sram_mpr_0_int_mask](#)
SRAM MPR 0, Mask 29.
- QM_RW uint32_t [flash_mpr_0_int_mask](#)
Flash MPR 0, Mask 30.
- QM_RW uint32_t [aonpt_0_int_mask](#)
AONPT 0, Mask 32.
- QM_RW uint32_t [adc_0_pwr_int_mask](#)
ADC 0 PWR, Mask 33.
- QM_RW uint32_t [adc_0_cal_int_mask](#)
ADC 0 CAL, Mask 34.
- QM_RW uint32_t [lock_int_mask_reg](#)
Interrupt Mask Lock Register.
- QM_RW uint32_t [ss_adc_0_error_int_mask](#)
Sensor ADC 0 Error.
- QM_RW uint32_t [ss_adc_0_int_mask](#)
Sensor ADC 0.
- QM_RW uint32_t [ss_gpio_0_int_mask](#)
Sensor GPIO 0.
- QM_RW uint32_t [ss_gpio_1_int_mask](#)
Sensor GPIO 1.
- [int_ss_i2c_reg_t ss_i2c_0_int](#)
Sensor I2C 0 Masks.
- [int_ss_i2c_reg_t ss_i2c_1_int](#)
Sensor I2C 1 Masks.
- [int_ss_spi_reg_t ss_spi_0_int](#)
Sensor SPI 0 Masks.
- [int_ss_spi_reg_t ss_spi_1_int](#)
Sensor SPI 1 Masks.
- QM_RW uint32_t [i2c_master_1_int_mask](#)
I2C Master 1.
- QM_RW uint32_t [spi_master_1_int_mask](#)
SPI Master 1.
- QM_RW uint32_t [i2s_0_int_mask](#)
I2S 0.
- QM_RW uint32_t [pwm_0_int_mask](#)
PWM 0.
- QM_RW uint32_t [usb_0_int_mask](#)
USB 0.
- QM_RW uint32_t [dma_0_int_2_mask](#)
DMA 0 Ch 2.
- QM_RW uint32_t [dma_0_int_3_mask](#)
DMA 0 Ch 3.
- QM_RW uint32_t [dma_0_int_4_mask](#)
DMA 0 Ch 4.
- QM_RW uint32_t [dma_0_int_5_mask](#)

- DMA 0 Ch 5.*
- QM_RW uint32_t [dma_0_int_6_mask](#)
- DMA 0 Ch 6.*
- QM_RW uint32_t [dma_0_int_7_mask](#)
- DMA 0 Ch 7.*
- QM_RW uint32_t [mailbox_0_int_mask](#)
- Mailbox 0 Combined 8 Channel Host and Sensor Masks.*
- QM_RW uint32_t [comparator_0_ss_halt_int_mask](#)
- Comparator Sensor Halt Mask.*
- QM_RW uint32_t [comparator_0_ss_int_mask](#)
- Comparator Sensor Mask.*
- QM_RW uint32_t [flash_mpr_1_int_mask](#)
- Flash MPR 1.*
- QM_RW uint32_t [aon_gpio_0_int_mask](#)
- AON GPIO 0.*

5.33.1 Detailed Description

Interrupt register map.

Definition at line 77 of file `qm_interrupt_router_regs.h`.

5.33.2 Field Documentation

5.33.2.1 QM_RW uint32_t qm_interrupt_router_reg_t::adc_0_cal_int_mask

ADC 0 CAL, Mask 34.

ADC 0 CAL.

Definition at line 106 of file `qm_interrupt_router_regs.h`.

5.33.2.2 QM_RW uint32_t qm_interrupt_router_reg_t::adc_0_pwr_int_mask

ADC 0 PWR, Mask 33.

ADC 0 PWR.

Definition at line 105 of file `qm_interrupt_router_regs.h`.

5.33.2.3 QM_RW uint32_t qm_interrupt_router_reg_t::aon_gpio_0_int_mask

AON GPIO 0.

Definition at line 287 of file `qm_interrupt_router_regs.h`.

5.33.2.4 QM_RW uint32_t qm_interrupt_router_reg_t::aonpt_0_int_mask

AONPT 0, Mask 32.

AONPT 0.

Definition at line 104 of file `qm_interrupt_router_regs.h`.

5.33.2.5 QM_RW uint32_t qm_interrupt_router_reg_t::comparator_0_host_halt_int_mask

Comparator 0 Host halt, Mask 24.

Comparator Host Halt Mask.

Definition at line 95 of file `qm_interrupt_router_regs.h`.

5.33.2.6 QM_RW uint32_t qm_interrupt_router_reg_t::comparator_0_host_int_mask

Comparator 0 Host, Mask 26.

Comparator Host Mask.

Definition at line 98 of file qm_interrupt_router_regs.h.

5.33.2.7 QM_RW uint32_t qm_interrupt_router_reg_t::comparator_0_ss_halt_int_mask

Comparator Sensor Halt Mask.

Definition at line 272 of file qm_interrupt_router_regs.h.

5.33.2.8 QM_RW uint32_t qm_interrupt_router_reg_t::comparator_0_ss_int_mask

Comparator Sensor Mask.

Definition at line 276 of file qm_interrupt_router_regs.h.

5.33.2.9 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_error_int_mask

DMA 0 Error, Mask 28.

DMA 0 Error.

Definition at line 100 of file qm_interrupt_router_regs.h.

5.33.2.10 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_0_mask

DMA 0 int 0, Mask 14.

DMA 0 Ch 0.

Definition at line 91 of file qm_interrupt_router_regs.h.

5.33.2.11 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_1_mask

DMA 0 int 1, Mask 15.

DMA 0 Ch 1.

Definition at line 92 of file qm_interrupt_router_regs.h.

5.33.2.12 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_2_mask

DMA 0 Ch 2.

Definition at line 263 of file qm_interrupt_router_regs.h.

5.33.2.13 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_3_mask

DMA 0 Ch 3.

Definition at line 264 of file qm_interrupt_router_regs.h.

5.33.2.14 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_4_mask

DMA 0 Ch 4.

Definition at line 265 of file qm_interrupt_router_regs.h.

5.33.2.15 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_5_mask

DMA 0 Ch 5.

Definition at line 266 of file qm_interrupt_router_regs.h.

5.33.2.16 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_6_mask

DMA 0 Ch 6.

Definition at line 267 of file qm_interrupt_router_regs.h.

5.33.2.17 QM_RW uint32_t qm_interrupt_router_reg_t::dma_0_int_7_mask

DMA 0 Ch 7.

Definition at line 268 of file qm_interrupt_router_regs.h.

5.33.2.18 QM_RW uint32_t qm_interrupt_router_reg_t::flash_mpr_0_int_mask

Flash MPR 0, Mask 30.

Flash MPR 0.

Definition at line 102 of file qm_interrupt_router_regs.h.

5.33.2.19 QM_RW uint32_t qm_interrupt_router_reg_t::flash_mpr_1_int_mask

Flash MPR 1.

Definition at line 283 of file qm_interrupt_router_regs.h.

5.33.2.20 QM_RW uint32_t qm_interrupt_router_reg_t::gpio_0_int_mask

GPIO 0, Mask 9.

GPIO 0.

Definition at line 86 of file qm_interrupt_router_regs.h.

5.33.2.21 QM_RW uint32_t qm_interrupt_router_reg_t::host_bus_error_int_mask

Host bus error, Mask 27.

Host bus error.

Definition at line 99 of file qm_interrupt_router_regs.h.

5.33.2.22 QM_RW uint32_t qm_interrupt_router_reg_t::i2c_master_0_int_mask

I2C Master 0, Mask 0.

I2C Master 0.

Definition at line 78 of file qm_interrupt_router_regs.h.

5.33.2.23 QM_RW uint32_t qm_interrupt_router_reg_t::i2c_master_1_int_mask

I2C Master 1.

Definition at line 248 of file qm_interrupt_router_regs.h.

5.33.2.24 QM_RW uint32_t qm_interrupt_router_reg_t::i2s_0_int_mask

I2S 0.

Definition at line 255 of file qm_interrupt_router_regs.h.

5.33.2.25 QM_RW uint32_t qm_interrupt_router_reg_t::lock_int_mask_reg

Interrupt Mask Lock Register.

Definition at line 108 of file qm_interrupt_router_regs.h.

5.33.2.26 QM_RW uint32_t qm_interrupt_router_reg_t::mailbox_0_int_mask

Mailbox 0 Combined 8 Channel Host and Sensor Masks.

Definition at line 270 of file qm_interrupt_router_regs.h.

5.33.2.27 QM_RW uint32_t qm_interrupt_router_reg_t::pwm_0_int_mask

PWM 0.

Definition at line 257 of file qm_interrupt_router_regs.h.

5.33.2.28 QM_RW uint32_t qm_interrupt_router_reg_t::rtc_0_int_mask

RTC 0, Mask 12.

RTC 0.

Definition at line 89 of file qm_interrupt_router_regs.h.

5.33.2.29 QM_RW uint32_t qm_interrupt_router_reg_t::spi_master_0_int_mask

SPI Master 0, Mask 3.

SPI Master 0.

Definition at line 80 of file qm_interrupt_router_regs.h.

5.33.2.30 QM_RW uint32_t qm_interrupt_router_reg_t::spi_master_1_int_mask

SPI Master 1.

Definition at line 251 of file qm_interrupt_router_regs.h.

5.33.2.31 QM_RW uint32_t qm_interrupt_router_reg_t::spi_slave_0_int_mask

SPI Slave 0, Mask 5.

SPI Slave 0.

Definition at line 82 of file qm_interrupt_router_regs.h.

5.33.2.32 QM_RW uint32_t qm_interrupt_router_reg_t::sram_mpr_0_int_mask

SRAM MPR 0, Mask 29.

SRAM MPR 0.

Definition at line 101 of file qm_interrupt_router_regs.h.

5.33.2.33 QM_RW uint32_t qm_interrupt_router_reg_t::ss_adc_0_error_int_mask

Sensor ADC 0 Error.

Definition at line 239 of file qm_interrupt_router_regs.h.

5.33.2.34 QM_RW uint32_t qm_interrupt_router_reg_t::ss_adc_0_int_mask

Sensor ADC 0.

Definition at line 240 of file qm_interrupt_router_regs.h.

5.33.2.35 QM_RW uint32_t qm_interrupt_router_reg_t::ss_gpio_0_int_mask

Sensor GPIO 0.

Definition at line 241 of file qm_interrupt_router_regs.h.

5.33.2.36 QM_RW uint32_t qm_interrupt_router_reg_t::ss_gpio_1_int_mask

Sensor GPIO 1.

Definition at line 242 of file qm_interrupt_router_regs.h.

5.33.2.37 int_ss_i2c_reg_t qm_interrupt_router_reg_t::ss_i2c_0_int

Sensor I2C 0 Masks.

Definition at line 243 of file qm_interrupt_router_regs.h.

5.33.2.38 int_ss_i2c_reg_t qm_interrupt_router_reg_t::ss_i2c_1_int

Sensor I2C 1 Masks.

Definition at line 244 of file qm_interrupt_router_regs.h.

5.33.2.39 int_ss_spi_reg_t qm_interrupt_router_reg_t::ss_spi_0_int

Sensor SPI 0 Masks.

Definition at line 245 of file qm_interrupt_router_regs.h.

5.33.2.40 int_ss_spi_reg_t qm_interrupt_router_reg_t::ss_spi_1_int

Sensor SPI 1 Masks.

Definition at line 246 of file qm_interrupt_router_regs.h.

5.33.2.41 QM_RW uint32_t qm_interrupt_router_reg_t::timer_0_int_mask

Timer 0, Mask 10.

Definition at line 87 of file qm_interrupt_router_regs.h.

5.33.2.42 QM_RW uint32_t qm_interrupt_router_reg_t::uart_0_int_mask

UART 0, Mask 6.

UART 0.

Definition at line 83 of file qm_interrupt_router_regs.h.

5.33.2.43 QM_RW uint32_t qm_interrupt_router_reg_t::uart_1_int_mask

UART 1, Mask 7.

UART 1.

Definition at line 84 of file qm_interrupt_router_regs.h.

5.33.2.44 QM_RW uint32_t qm_interrupt_router_reg_t::usb_0_int_mask

USB 0.

Definition at line 258 of file qm_interrupt_router_regs.h.

5.33.2.45 QM_RW uint32_t qm_interrupt_router_reg_t::wdt_0_int_mask

WDT 0, Mask 13.

WDT 0.

Definition at line 90 of file qm_interrupt_router_regs.h.

5.34 qm_irq_context_t Struct Reference

SS IRQ context type.

```
#include <qm_sensor_regs.h>
```

Data Fields

- uint32_t [status32_irq_threshold](#)
STATUS32 Interrupt Threshold.
- uint32_t [status32_irq_enable](#)
STATUS32 Interrupt Enable.
- uint32_t [irq_ctrl](#)
Interrupt Context Saving Control Register.
- uint8_t [irq_config](#) [QM_SS_INT_VECTOR_NUM-QM_SS_EXCEPTION_NUM]
IRQ configuration:
- uint32_t [redtbl_entries](#) [QM_IOAPIC_NUM_RTES]
Redirection Table Entries.

5.34.1 Detailed Description

SS IRQ context type.

IRQ context type.

Applications should not modify the content. This structure is only intended to be used by `qm_irq_save_context` and `qm_irq_restore_context` functions.

Definition at line 96 of file `qm_sensor_regs.h`.

5.34.2 Field Documentation

5.34.2.1 uint8_t qm_irq_context_t::irq_config[QM_SS_INT_VECTOR_NUM-QM_SS_EXCEPTION_NUM]

IRQ configuration:

- IRQ Priority:BIT(6):BIT(2)
- IRQ Trigger:BIT(1)
- IRQ Enable:BIT(0)

Definition at line 107 of file `qm_sensor_regs.h`.

5.34.2.2 uint32_t qm_irq_context_t::irq_ctrl

Interrupt Context Saving Control Register.

Definition at line 99 of file `qm_sensor_regs.h`.

5.34.2.3 uint32_t qm_irq_context_t::redtbl_entries[QM_IOAPIC_NUM_RTES]

Redirection Table Entries.

Definition at line 272 of file `qm_soc_regs.h`.

5.34.2.4 uint32_t qm_irq_context_t::status32_irq_enable

STATUS32 Interrupt Enable.

Definition at line 98 of file `qm_sensor_regs.h`.

5.34.2.5 uint32_t qm_irq_context_t::status32_irq_threshold

STATUS32 Interrupt Threshold.

Definition at line 97 of file qm_sensor_regs.h.

5.35 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 [lvtimer](#)
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.

5.35.1 Detailed Description

APIC register block type.

Definition at line 224 of file `qm_soc_regs.h`.

5.36 qm_mailbox_reg_t Struct Reference

Mailbox register map.

```
#include <qm_soc_regs.h>
```

Data Fields

- [qm_mailbox_t mbox](#) [NUM_MAILBOXES]
8 Mailboxes
- QM_RW uint32_t [mbox_chall_sts](#)
All channel status.

5.36.1 Detailed Description

Mailbox register map.

Definition at line 641 of file `qm_soc_regs.h`.

5.37 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.

5.37.1 Detailed Description

Mailbox register structure.

Definition at line 634 of file qm_soc_regs.h.

5.38 qm_mbox_config_t Struct Reference

Mailbox Configuration Structure.

```
#include <qm_mailbox.h>
```

Data Fields

- [qm_mbox_destination_t dest](#)
 < Mailbox Destination
- [qm_mbox_mode_t mode](#)
 Message callback.
- [qm_mbox_callback_t callback](#)
 Callback function data to return via the callback function.

5.38.1 Detailed Description

Mailbox Configuration Structure.

Definition at line 83 of file qm_mailbox.h.

5.38.2 Field Documentation

5.38.2.1 qm_mbox_callback_t qm_mbox_config_t::callback

Callback function data to return via the callback function.

Definition at line 99 of file qm_mailbox.h.

Referenced by qm_mbox_ch_set_config().

5.38.2.2 qm_mbox_destination_t qm_mbox_config_t::dest

< Mailbox Destination

Mode of operation

Definition at line 85 of file qm_mailbox.h.

Referenced by qm_mbox_ch_set_config().

5.38.2.3 qm_mbox_mode_t qm_mbox_config_t::mode

Message callback.

Called after a message is received on the channel and the channel is configured in interrupt mode.

Note

NULL for a write Mailbox.

Parameters

<code>in</code>	<code>data</code>	User defined data.
-----------------	-------------------	--------------------

Definition at line 87 of file `qm_mailbox.h`.

Referenced by `qm_mbox_ch_set_config()`.

5.39 qm_mbox_msg_t Struct Reference

Definition of the mailbox message.

```
#include <qm_mailbox.h>
```

Data Fields

- `uint32_t ctrl`
Control word - bits 30 to 0 used as data/message id, bit 31 triggers channel interrupt when set by the driver.
- `uint32_t data [QM_MBOX_PAYLOAD_NUM]`
Mailbox data buffer.

5.39.1 Detailed Description

Definition of the mailbox message.

Definition at line 64 of file `qm_mailbox.h`.

5.39.2 Field Documentation**5.39.2.1 uint32_t qm_mbox_msg_t::data[QM_MBOX_PAYLOAD_NUM]**

Mailbox data buffer.

Definition at line 70 of file `qm_mailbox.h`.

Referenced by `qm_mbox_ch_read()`, and `qm_mbox_ch_write()`.

5.40 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

5.40.1 Detailed Description

SRAM Memory Protection Region configuration type.

Definition at line 34 of file qm_mpr.h.

5.41 qm_mpr_context_t Struct Reference

MPR context type.

```
#include <qm_soc_regs.h>
```

Data Fields

- uint32_t [mpr_cfg](#) [[QM_MPR_NUM](#)]
MPR Configuration Register.

5.41.1 Detailed Description

MPR context type.

Application should not modify the content. This structure is only intended to be used by the qm_mpr_save_context and qm_mpr_restore_context functions.

Definition at line 1629 of file qm_soc_regs.h.

5.41.2 Field Documentation

5.41.2.1 uint32_t qm_mpr_context_t::mpr_cfg[QM_MPR_NUM]

MPR Configuration Register.

Definition at line 1630 of file qm_soc_regs.h.

Referenced by qm_mpr_restore_context(), and qm_mpr_save_context().

5.42 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.

5.42.1 Detailed Description

Memory Protection Region register map.

Definition at line 1295 of file qm_soc_regs.h.

5.43 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 lvtimer](#)
Timer vector.
- QM_RW [mvic_reg_pad_t icr](#)
Timer initial count.
- QM_RW [mvic_reg_pad_t ccr](#)
Timer current count.

5.43.1 Detailed Description

MVIC register map.

Definition at line 1415 of file qm_soc_regs.h.

5.43.2 Field Documentation

5.43.2.1 QM_RW mvic_reg_pad_t qm_mvic_reg_t::ccr

Timer current count.

Definition at line 1430 of file qm_soc_regs.h.

5.43.2.2 QM_RW mvic_reg_pad_t qm_mvic_reg_t::eoi

End of interrupt.

Definition at line 1419 of file qm_soc_regs.h.

5.43.2.3 QM_RW mvic_reg_pad_t qm_mvic_reg_t::icr

Timer initial count.

Definition at line 1429 of file qm_soc_regs.h.

5.43.2.4 QM_RW mvic_reg_pad_t qm_mvic_reg_t::irr

Interrupt request.

Definition at line 1425 of file qm_soc_regs.h.

5.43.2.5 QM_RW mvic_reg_pad_t qm_mvic_reg_t::isr

In-service.

Definition at line 1423 of file qm_soc_regs.h.

5.43.2.6 QM_RW mvic_reg_pad_t qm_mvic_reg_t::lvtimer

Timer vector.

Definition at line 1427 of file qm_soc_regs.h.

5.43.2.7 QM_RW mvic_reg_pad_t qm_mvic_reg_t::ppr

Processor priority.

Definition at line 1418 of file qm_soc_regs.h.

5.43.2.8 QM_RW mvic_reg_pad_t qm_mvic_reg_t::sivr

Spurious vector.

Definition at line 1421 of file qm_soc_regs.h.

5.43.2.9 QM_RW mvic_reg_pad_t qm_mvic_reg_t::tpr

Task priority.

Definition at line 1416 of file qm_soc_regs.h.

5.44 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.

5.44.1 Detailed Description

PIC timer configuration type.

Definition at line 29 of file qm_pic_timer.h.

5.44.2 Field Documentation

5.44.2.1 void(* qm_pic_timer_config_t::callback)(void *data)

User callback.

Parameters

<i>in</i>	<i>data</i>	User defined data.
-----------	-------------	--------------------

Definition at line 38 of file `qm_pic_timer.h`.

Referenced by `qm_pic_timer_set_config()`.

5.44.2.2 `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()`.

5.44.2.3 `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()`.

5.44.2.4 `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()`.

5.45 `qm_pic_timer_context_t` Struct Reference

PIC TIMER context type.

```
#include <qm_soc_regs.h>
```

Data Fields

- `uint32_t timer_icr`
Initial Count Register.
- `uint32_t timer_dcr`
Divide Configuration Register.
- `uint32_t lvtimer`
Timer Entry in Local Vector Table.

5.45.1 Detailed Description

PIC TIMER context type.

Applications should not modify the content. This structure is only intended to be used by the `qm_pic_timer_save_context` and `qm_pic_timer_restore_context` functions.

Definition at line 283 of file `qm_soc_regs.h`.

5.45.2 Field Documentation

5.45.2.1 `uint32_t qm_pic_timer_context_t::lvtimer`

Timer Entry in Local Vector Table.

Definition at line 286 of file qm_soc_regs.h.

Referenced by qm_pic_timer_restore_context(), and qm_pic_timer_save_context().

5.45.2.2 uint32_t qm_pic_timer_context_t::timer_dcr

Divide Configuration Register.

Definition at line 285 of file qm_soc_regs.h.

Referenced by qm_pic_timer_restore_context(), and qm_pic_timer_save_context().

5.45.2.3 uint32_t qm_pic_timer_context_t::timer_icr

Initial Count Register.

Definition at line 284 of file qm_soc_regs.h.

Referenced by qm_pic_timer_restore_context(), and qm_pic_timer_save_context().

5.46 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 lvtimer](#)
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.

5.46.1 Detailed Description

PIC timer register map.

Definition at line 1341 of file qm_soc_regs.h.

5.47 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.

5.47.1 Detailed Description

PWM / Timer channel register map.

Definition at line 389 of file `qm_soc_regs.h`.

5.47.2 Field Documentation

5.47.2.1 QM_RW uint32_t `qm_pwm_channel_t::controlreg`

Control.

Definition at line 392 of file `qm_soc_regs.h`.

Referenced by `qm_pwm_restore_context()`, `qm_pwm_save_context()`, `qm_pwm_set_config()`, `qm_pwm_start()`, and `qm_pwm_stop()`.

5.47.2.2 QM_RW uint32_t `qm_pwm_channel_t::currentvalue`

Current Value.

Definition at line 391 of file `qm_soc_regs.h`.

5.47.2.3 QM_RW uint32_t `qm_pwm_channel_t::eoi`

End Of Interrupt.

Definition at line 393 of file `qm_soc_regs.h`.

Referenced by `QM_ISR_DECLARE()`.

5.47.2.4 QM_RW uint32_t `qm_pwm_channel_t::intstatus`

Interrupt Status.

Definition at line 394 of file `qm_soc_regs.h`.

5.47.2.5 QM_RW uint32_t `qm_pwm_channel_t::loadcount`

Load Coun.t.

Load Count.

Definition at line 390 of file `qm_soc_regs.h`.

Referenced by `qm_pwm_get()`, `qm_pwm_restore_context()`, `qm_pwm_save_context()`, `qm_pwm_set()`, and `qm_pwm_set_config()`.

5.48 `qm_pwm_config_t` Struct Reference

QM 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.

5.48.1 Detailed Description

QM PWM / Timer configuration type.

Definition at line 33 of file qm_pwm.h.

5.48.2 Field Documentation

5.48.2.1 void(* qm_pwm_config_t::callback)(void *data, uint32_t int_status)

User callback.

Parameters

in	<i>data</i>	The callback user data.
in	<i>int_status</i>	The timer status.

Definition at line 53 of file qm_pwm.h.

Referenced by [qm_pwm_set_config\(\)](#).

5.48.2.2 void* qm_pwm_config_t::callback_data

Callback user data.

Definition at line 54 of file qm_pwm.h.

Referenced by [qm_pwm_set_config\(\)](#).

5.48.2.3 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 43 of file qm_pwm.h.

Referenced by [qm_pwm_set_config\(\)](#).

5.48.2.4 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 38 of file qm_pwm.h.

Referenced by [qm_pwm_set_config\(\)](#).

5.48.2.5 bool qm_pwm_config_t::mask_interrupt

Mask interrupt.

Definition at line 44 of file qm_pwm.h.

Referenced by `qm_pwm_set_config()`.

5.48.2.6 `qm_pwm_mode_t` `qm_pwm_config_t::mode`

Pwm mode.

Definition at line 45 of file `qm_pwm.h`.

Referenced by `qm_pwm_set_config()`.

5.49 `qm_pwm_context_t` Struct Reference

PWM context type.

```
#include <qm_soc_regs.h>
```

5.49.1 Detailed Description

PWM context type.

Applications should not modify the content. This structure is only intended to be used by the `qm_pwm_save_context` and `qm_pwm_restore_context` functions.

Definition at line 702 of file `qm_soc_regs.h`.

5.49.2 Field Documentation

5.49.2.1 `uint32_t` `qm_pwm_context_t::controlreg`

Control Register.

Definition at line 706 of file `qm_soc_regs.h`.

Referenced by `qm_pwm_restore_context()`, and `qm_pwm_save_context()`.

5.49.2.2 `uint32_t` `qm_pwm_context_t::loadcount`

Load Count 1.

Definition at line 704 of file `qm_soc_regs.h`.

Referenced by `qm_pwm_restore_context()`, and `qm_pwm_save_context()`.

5.49.2.3 `uint32_t` `qm_pwm_context_t::loadcount2`

Load Count 2.

Definition at line 705 of file `qm_soc_regs.h`.

Referenced by `qm_pwm_restore_context()`, and `qm_pwm_save_context()`.

5.50 `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](#) [QM_PWM_ID_NUM]
Timer Load Count 2.

5.50.1 Detailed Description

PWM / Timer register map.

Definition at line 398 of file qm_soc_regs.h.

5.50.2 Field Documentation

5.50.2.1 qm_pwm_channel_t qm_pwm_reg_t::timer

2 Timers.

4 Timers

Definition at line 399 of file qm_soc_regs.h.

Referenced by QM_ISR_DECLARE(), qm_pwm_get(), qm_pwm_restore_context(), qm_pwm_save_context(), qm_pwm_set(), qm_pwm_set_config(), qm_pwm_start(), and qm_pwm_stop().

5.51 qm_rtc_config_t Struct Reference

QM 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.
- clk_rtc_div_t [prescaler](#)
RTC Clock prescaler.
- void(* [callback](#))(void *data)
User callback.
- void * [callback_data](#)
Callback user data.

5.51.1 Detailed Description

QM RTC configuration type.

Definition at line 39 of file qm_rtc.h.

5.51.2 Field Documentation

5.51.2.1 `bool qm_rtc_config_t::alarm_en`

Alarm enable.

Definition at line 41 of file `qm_rtc.h`.

Referenced by `qm_rtc_set_config()`.

5.51.2.2 `uint32_t qm_rtc_config_t::alarm_val`

Alarm value in RTC clocks.

Definition at line 42 of file `qm_rtc.h`.

Referenced by `qm_rtc_set_config()`.

5.51.2.3 `void(* qm_rtc_config_t::callback)(void *data)`

User callback.

Parameters

<code>in</code>	<code>data</code>	User defined data.
-----------------	-------------------	--------------------

Definition at line 57 of file `qm_rtc.h`.

Referenced by `qm_rtc_set_config()`.

5.51.2.4 `void* qm_rtc_config_t::callback_data`

Callback user data.

Definition at line 58 of file `qm_rtc.h`.

Referenced by `qm_rtc_set_config()`.

5.51.2.5 `uint32_t qm_rtc_config_t::init_val`

Initial value in RTC clocks.

Definition at line 40 of file `qm_rtc.h`.

Referenced by `qm_rtc_set_config()`.

5.51.2.6 `clk_rtc_div_t qm_rtc_config_t::prescaler`

RTC Clock prescaler.

Used to divide the clock frequency of the RTC.

Definition at line 50 of file `qm_rtc.h`.

Referenced by `qm_rtc_set_config()`.

5.52 `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.

5.52.1 Detailed Description

RTC register map.

Definition at line 821 of file qm_soc_regs.h.

5.52.2 Field Documentation

5.52.2.1 QM_RW uint32_t qm_rtc_reg_t::rtc_ccr

Counter Control Register.

Definition at line 825 of file qm_soc_regs.h.

5.52.2.2 QM_RW uint32_t qm_rtc_reg_t::rtc_ccvr

Current Counter Value Register.

Definition at line 822 of file qm_soc_regs.h.

5.52.2.3 QM_RW uint32_t qm_rtc_reg_t::rtc_clr

Counter Load Register.

Definition at line 824 of file qm_soc_regs.h.

5.52.2.4 QM_RW uint32_t qm_rtc_reg_t::rtc_cmr

Current Match Register.

Definition at line 823 of file qm_soc_regs.h.

5.52.2.5 QM_RW uint32_t qm_rtc_reg_t::rtc_comp_version

End of Interrupt Register.

Definition at line 829 of file qm_soc_regs.h.

5.52.2.6 QM_RW uint32_t qm_rtc_reg_t::rtc_eoi

End of Interrupt Register.

Definition at line 828 of file qm_soc_regs.h.

5.52.2.7 QM_RW uint32_t qm_rtc_reg_t::rtc_rstat

Interrupt Raw Status Register.

Definition at line 827 of file qm_soc_regs.h.

5.52.2.8 QM_RW uint32_t qm_rtc_reg_t::rtc_stat

Interrupt Status Register.

Definition at line 826 of file qm_soc_regs.h.

5.53 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 lock configuration.
- QM_RW uint32_t [ccu_ss_periph_clk_gate_ctl](#)
Sensor Subsystem peripheral clock gate control.

5.53.1 Detailed Description

System Core register map.

Definition at line 30 of file qm_soc_regs.h.

5.53.2 Field Documentation

5.53.2.1 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_ext_clock_ctl

External Clock Control Register.

Definition at line 44 of file qm_soc_regs.h.

5.53.2.2 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_gpio_db_clk_ctl

Peripheral Clock Divider Control 1.

Definition at line 42 of file qm_soc_regs.h.

5.53.2.3 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_lp_clk_ctl

System Low Power Clock Control.

Definition at line 46 of file qm_soc_regs.h.

5.53.2.4 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_mlayer_ahb_ctl

AHB Control Register.

Definition at line 48 of file qm_soc_regs.h.

5.53.2.5 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 40 of file qm_soc_regs.h.

5.53.2.6 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_periph_clk_gate_ctl

Peripheral Clock Gate Control.

Definition at line 38 of file qm_soc_regs.h.

5.53.2.7 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_ss_periph_clk_gate_ctl

Sensor Subsystem peripheral clock gate control.

Definition at line 52 of file qm_soc_regs.h.

5.53.2.8 QM_RW uint32_t qm_scss_ccu_reg_t::ccu_sys_clk_ctl

System Clock Control Register.

Definition at line 49 of file qm_soc_regs.h.

5.53.2.9 QM_RW uint32_t qm_scss_ccu_reg_t::osc0_cfg0

Hybrid Oscillator Configuration 0.

Definition at line 31 of file qm_soc_regs.h.

5.53.2.10 QM_RW uint32_t qm_scss_ccu_reg_t::osc0_cfg1

Hybrid Oscillator configuration 1.

Definition at line 33 of file qm_soc_regs.h.

5.53.2.11 QM_RW uint32_t qm_scss_ccu_reg_t::osc0_stat1

Hybrid Oscillator status 1.

Definition at line 32 of file qm_soc_regs.h.

5.53.2.12 QM_RW uint32_t qm_scss_ccu_reg_t::osc1_cfg0

RTC Oscillator Configuration 0.

Definition at line 35 of file qm_soc_regs.h.

5.53.2.13 QM_RW uint32_t qm_scss_ccu_reg_t::osc1_stat0

RTC Oscillator status 0.

Definition at line 34 of file qm_soc_regs.h.

5.53.2.14 QM_RW uint32_t qm_scss_ccu_reg_t::osc_lock_0

Clocks Lock Register.

Definition at line 50 of file qm_soc_regs.h.

5.53.2.15 QM_RW uint32_t qm_scss_ccu_reg_t::soc_ctrl

SoC Control Register.

Definition at line 51 of file qm_soc_regs.h.

5.53.2.16 QM_RW uint32_t qm_scss_ccu_reg_t::soc_ctrl_lock

SoC Control Register Lock.

Definition at line 52 of file qm_soc_regs.h.

5.53.2.17 QM_RW uint32_t qm_scss_ccu_reg_t::usb_pll_cfg0

USB Phase lock look configuration.

Definition at line 42 of file qm_soc_regs.h.

5.53.2.18 QM_RW uint32_t qm_scss_ccu_reg_t::wake_mask

Wake Mask register.

Definition at line 47 of file qm_soc_regs.h.

5.54 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.

5.54.1 Detailed Description

Comparator register map.

Definition at line 181 of file qm_soc_regs.h.

5.54.2 Field Documentation

5.54.2.1 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_en

Comparator enable.

Definition at line 182 of file qm_soc_regs.h.

5.54.2.2 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_pwr

Comparator power enable register.

Definition at line 186 of file qm_soc_regs.h.

5.54.2.3 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_ref_pol

Comparator reference polarity select register.

Definition at line 185 of file qm_soc_regs.h.

5.54.2.4 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_ref_sel

Comparator reference select.

Definition at line 183 of file qm_soc_regs.h.

5.54.2.5 QM_RW uint32_t qm_scss_cmp_reg_t::cmp_stat_clr

Comparator clear register.

Definition at line 188 of file qm_soc_regs.h.

5.55 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.*

5.55.1 Detailed Description

General Purpose register map.

Definition at line 144 of file qm_soc_regs.h.

5.55.2 Field Documentation

5.55.2.1 QM_RW uint32_t qm_scss_gp_reg_t::gp0

General Purpose Scratchpad Register 0.

Definition at line 150 of file qm_soc_regs.h.

5.55.2.2 QM_RW uint32_t qm_scss_gp_reg_t::gp1

General Purpose Scratchpad Register 1.

Definition at line 151 of file qm_soc_regs.h.

5.55.2.3 QM_RW uint32_t qm_scss_gp_reg_t::gp2

General Purpose Scratchpad Register 2.

Definition at line 152 of file qm_soc_regs.h.

5.55.2.4 QM_RW uint32_t qm_scss_gp_reg_t::gp3

General Purpose Scratchpad Register 3.

Definition at line 153 of file qm_soc_regs.h.

5.55.2.5 QM_RW uint32_t qm_scss_gp_reg_t::gps0

General Purpose Sticky Register 0.

Definition at line 145 of file qm_soc_regs.h.

5.55.2.6 QM_RW uint32_t qm_scss_gp_reg_t::gps1

General Purpose Sticky Register 1.

Definition at line 146 of file qm_soc_regs.h.

5.55.2.7 QM_RW uint32_t qm_scss_gp_reg_t::gps2

General Purpose Sticky Register 2.

Definition at line 147 of file qm_soc_regs.h.

5.55.2.8 QM_RW uint32_t qm_scss_gp_reg_t::gps3

General Purpose Sticky Register 3.

Definition at line 148 of file qm_soc_regs.h.

5.55.2.9 QM_RW uint32_t qm_scss_gp_reg_t::wo_sp

Write-One-to-Set Scratchpad Register.

Definition at line 155 of file qm_soc_regs.h.

5.55.2.10 QM_RW uint32_t qm_scss_gp_reg_t::wo_st

Write-One-to-Set Sticky Scratchpad Register.

Definition at line 157 of file qm_soc_regs.h.

5.56 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.

5.56.1 Detailed Description

Information register map.

Definition at line 357 of file qm_soc_regs.h.

5.56.2 Field Documentation

5.56.2.1 QM_RW uint32_t qm_scss_info_reg_t::cotps

Code OTP Size Register.

Definition at line 362 of file qm_soc_regs.h.

5.56.2.2 QM_RW uint32_t qm_scss_info_reg_t::dotps

Data OTP Size Register.

Definition at line 363 of file qm_soc_regs.h.

5.56.2.3 QM_RW uint32_t qm_scss_info_reg_t::fs

Flash Size Register.

Definition at line 360 of file qm_soc_regs.h.

5.56.2.4 QM_RW uint32_t qm_scss_info_reg_t::id

Identification Register.

Definition at line 358 of file qm_soc_regs.h.

5.56.2.5 QM_RW uint32_t qm_scss_info_reg_t::rev

Revision Register.

Definition at line 359 of file qm_soc_regs.h.

5.56.2.6 QM_RW uint32_t qm_scss_info_reg_t::rs

RAM Size Register.

Definition at line 361 of file qm_soc_regs.h.

5.57 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.

5.57.1 Detailed Description

Memory Control register map.

Definition at line 169 of file qm_soc_regs.h.

5.58 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.

5.58.1 Detailed Description

Peripheral Registers register map.

Definition at line 301 of file qm_soc_regs.h.

5.58.2 Field Documentation

5.58.2.1 QM_RW uint32_t qm_scss_peripheral_reg_t::cfg_lock

Configuration Lock.

Definition at line 304 of file qm_soc_regs.h.

5.58.2.2 QM_RW uint32_t qm_scss_peripheral_reg_t::periph_cfg0

Peripheral Configuration.

Definition at line 302 of file qm_soc_regs.h.

5.59 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_lv2](#)
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.

5.59.1 Detailed Description

Power Management register map.

Definition at line 210 of file `qm_soc_regs.h`.

5.59.2 Field Documentation

5.59.2.1 QM_RW uint32_t qm_scss_pmu_reg_t::aon_vr

AON Voltage Regulator.

Definition at line 211 of file `qm_soc_regs.h`.

5.59.2.2 QM_RW uint32_t qm_scss_pmu_reg_t::p_sts

Processor Status.

Definition at line 215 of file `qm_soc_regs.h`.

5.59.2.3 QM_RW uint32_t qm_scss_pmu_reg_t::pm_lock

Power Management Lock.

Definition at line 220 of file `qm_soc_regs.h`.

5.59.2.4 QM_RW uint32_t qm_scss_pmu_reg_t::pm_wait

Power Management Wait.

Definition at line 213 of file `qm_soc_regs.h`.

5.59.2.5 QM_RW uint32_t qm_scss_pmu_reg_t::rstc

Reset Control.

Definition at line 217 of file `qm_soc_regs.h`.

5.59.2.6 QM_RW uint32_t qm_scss_pmu_reg_t::rsts

Reset Status.

Definition at line 218 of file `qm_soc_regs.h`.

5.60 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.

5.60.1 Detailed Description

Pin MUX register map.

Definition at line 324 of file `qm_soc_regs.h`.

5.60.2 Field Documentation

5.60.2.1 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_in_en

Pin Mux Input Enable.

Definition at line 329 of file `qm_soc_regs.h`.

5.60.2.2 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_in_en_lock

Pin Mux Slew Rate Lock.

Definition at line 337 of file `qm_soc_regs.h`.

5.60.2.3 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_pullup

Pin Mux Pullup.

Definition at line 325 of file `qm_soc_regs.h`.

5.60.2.4 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_pullup_lock

Pin Mux Pullup Lock.

Definition at line 333 of file `qm_soc_regs.h`.

5.60.2.5 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_sel

Pin Mux Select.

Definition at line 331 of file `qm_soc_regs.h`.

5.60.2.6 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_sel_0_lock

Pin Mux Select Lock 0.

Definition at line 335 of file qm_soc_regs.h.

5.60.2.7 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_slew

Pin Mux Slew Rate.

Definition at line 327 of file qm_soc_regs.h.

5.60.2.8 QM_RW uint32_t qm_scss_pmux_reg_t::pmux_slew_lock

Pin Mux Slew Rate Lock.

Definition at line 334 of file qm_soc_regs.h.

5.61 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.

5.61.1 Detailed Description

Sensor Subsystem register map.

Definition at line 408 of file qm_soc_regs.h.

5.62 qm_spi_async_transfer_t Struct Reference

SPI asynchronous transfer type.

```
#include <qm_spi.h>
```

Data Fields

- void * [tx](#)
Write data.
- void * [rx](#)
Read data.
- uint16_t [tx_len](#)
Number of data frames to write.
- uint16_t [rx_len](#)
Number of data frames to read.
- bool [keep_enabled](#)
Keep device on once transfer is done.
- void(* [callback](#))(void *data, int error, [qm_spi_status_t](#) status, uint16_t len)

Transfer callback.

- void * [callback_data](#)

Callback user data.

5.62.1 Detailed Description

SPI asynchronous transfer type.

If the frame size is 8 bits or less, 1 byte is needed per data frame. If the frame size is 9-16 bits, 2 bytes are needed per data frame and frames of more than 16 bits require 4 bytes. In each case, the least significant bits are sent while the extra bits are discarded. The most significant bits of the frame are sent first.

Definition at line 145 of file qm_spi.h.

5.62.2 Field Documentation

5.62.2.1 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. For slave device it also allows the application to update transfer information by calling the qm_spi_irq_update function.

Parameters

in	<i>data</i>	The callback user data.
in	<i>error</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	SPI driver status.
in	<i>len</i>	Length of the SPI transfer if successful, 0 otherwise.

Definition at line 167 of file qm_spi.h.

Referenced by qm_spi_irq_transfer_terminate().

5.62.2.2 void* qm_spi_async_transfer_t::callback_data

Callback user data.

Definition at line 169 of file qm_spi.h.

Referenced by qm_spi_irq_transfer_terminate().

5.62.2.3 bool qm_spi_async_transfer_t::keep_enabled

Keep device on once transfer is done.

Definition at line 150 of file qm_spi.h.

5.62.2.4 void* qm_spi_async_transfer_t::rx

Read data.

Definition at line 147 of file qm_spi.h.

Referenced by qm_spi_dma_transfer().

5.62.2.5 uint16_t qm_spi_async_transfer_t::rx_len

Number of data frames to read.

Definition at line 149 of file qm_spi.h.

Referenced by qm_spi_dma_transfer(), qm_spi_irq_transfer(), and qm_spi_irq_update().

5.62.2.6 void* qm_spi_async_transfer_t::tx

Write data.

Definition at line 146 of file qm_spi.h.

Referenced by qm_spi_dma_transfer().

5.62.2.7 uint16_t qm_spi_async_transfer_t::tx_len

Number of data frames to write.

Definition at line 148 of file qm_spi.h.

Referenced by qm_spi_dma_transfer(), qm_spi_irq_transfer(), and qm_spi_irq_update().

5.63 qm_spi_context_t Struct Reference

SPI context type.

```
#include <qm_soc_regs.h>
```

Data Fields

- uint32_t [ctrlr0](#)
Control Register 0.
- uint32_t [ser](#)
Slave Enable Register.
- uint32_t [baudr](#)
Baud Rate Select.

5.63.1 Detailed Description

SPI context type.

Applications should not modify the content. This structure is only intended to be used by the qm_spi_save_context and qm_spi_restore_context functions.

Definition at line 1071 of file qm_soc_regs.h.

5.63.2 Field Documentation

5.63.2.1 uint32_t qm_spi_context_t::baudr

Baud Rate Select.

Definition at line 1074 of file qm_soc_regs.h.

Referenced by qm_spi_restore_context(), and qm_spi_save_context().

5.63.2.2 uint32_t qm_spi_context_t::ctrlr0

Control Register 0.

Definition at line 1072 of file qm_soc_regs.h.

Referenced by qm_spi_restore_context(), and qm_spi_save_context().

5.63.2.3 uint32_t qm_spi_context_t::ser

Slave Enable Register.

Definition at line 1073 of file qm_soc_regs.h.

Referenced by qm_spi_restore_context(), and qm_spi_save_context().

5.64 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.*

5.64.1 Detailed Description

SPI register map.

Definition at line 712 of file `qm_soc_regs.h`.

5.64.2 Field Documentation

5.64.2.1 QM_RW uint32_t qm_spi_reg_t::baudr

Baud Rate Select.

Definition at line 718 of file `qm_soc_regs.h`.

Referenced by `qm_spi_restore_context()`, `qm_spi_save_context()`, and `qm_spi_set_config()`.

5.64.2.2 QM_RW uint32_t qm_spi_reg_t::ctrlr0

Control Register 0.

Definition at line 713 of file `qm_soc_regs.h`.

Referenced by `qm_spi_irq_transfer()`, `qm_spi_restore_context()`, `qm_spi_save_context()`, and `qm_spi_set_config()`.

5.64.2.3 QM_RW uint32_t qm_spi_reg_t::ctrlr1

Control Register 1.

Definition at line 714 of file `qm_soc_regs.h`.

Referenced by `qm_spi_dma_transfer()`, `qm_spi_irq_transfer()`, and `qm_spi_transfer()`.

5.64.2.4 QM_RW uint32_t qm_spi_reg_t::dmacr

DMA Control Register.

Definition at line 735 of file `qm_soc_regs.h`.

Referenced by `qm_spi_dma_transfer()`.

5.64.2.5 QM_RW uint32_t qm_spi_reg_t::dmardlr

DMA Receive Data Level.

Definition at line 737 of file `qm_soc_regs.h`.

Referenced by `qm_spi_dma_transfer()`.

5.64.2.6 QM_RW uint32_t qm_spi_reg_t::dmatdlr

DMA Transmit Data Level.

Definition at line 736 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer().

5.64.2.7 QM_RW uint32_t qm_spi_reg_t::dr

Data Register.

Definition at line 740 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer().

5.64.2.8 QM_RW uint32_t qm_spi_reg_t::icr

Interrupt Clear Register.

Definition at line 734 of file qm_soc_regs.h.

5.64.2.9 QM_RW uint32_t qm_spi_reg_t::idr

Identification Register.

Definition at line 738 of file qm_soc_regs.h.

5.64.2.10 QM_RW uint32_t qm_spi_reg_t::imr

Interrupt Mask Register.

Definition at line 724 of file qm_soc_regs.h.

Referenced by qm_spi_dma_transfer(), qm_spi_irq_transfer_terminate(), qm_spi_irq_update(), and qm_spi_transfer().

5.64.2.11 QM_RW uint32_t qm_spi_reg_t::isr

Interrupt Status Register.

Definition at line 725 of file qm_soc_regs.h.

5.64.2.12 QM_RW uint32_t qm_spi_reg_t::msticr

Multi-Master Interrupt Clear Register.

Definition at line 733 of file qm_soc_regs.h.

5.64.2.13 QM_RW uint32_t qm_spi_reg_t::mwcr

Microwire Control Register.

Definition at line 716 of file qm_soc_regs.h.

5.64.2.14 QM_RW uint32_t qm_spi_reg_t::risr

Raw Interrupt Status Register.

Definition at line 726 of file qm_soc_regs.h.

Referenced by qm_spi_get_status(), and qm_spi_transfer().

5.64.2.15 QM_RW uint32_t qm_spi_reg_t::rx_sample_dly

RX Sample Delay Register.

Definition at line 741 of file qm_soc_regs.h.

5.64.2.16 QM_RW uint32_t qm_spi_reg_t::rxflr

Receive FIFO Level Register.

Definition at line 722 of file qm_soc_regs.h.

5.64.2.17 QM_RW uint32_t qm_spi_reg_t::rxftlr

Receive FIFO Threshold Level.

Definition at line 720 of file qm_soc_regs.h.

Referenced by qm_spi_irq_transfer().

5.64.2.18 QM_RW uint32_t qm_spi_reg_t::rxoicr

Rx FIFO Overflow Interrupt Clear Register.

Definition at line 730 of file qm_soc_regs.h.

Referenced by qm_spi_transfer().

5.64.2.19 QM_RW uint32_t qm_spi_reg_t::rxuicr

Rx FIFO Underflow Interrupt Clear Register.

Definition at line 732 of file qm_soc_regs.h.

5.64.2.20 QM_RW uint32_t qm_spi_reg_t::ser

Slave Enable Register.

Definition at line 717 of file qm_soc_regs.h.

Referenced by qm_spi_restore_context(), and qm_spi_save_context().

5.64.2.21 QM_RW uint32_t qm_spi_reg_t::sr

Status Register.

Definition at line 723 of file qm_soc_regs.h.

Referenced by qm_spi_get_status(), and qm_spi_transfer().

5.64.2.22 QM_RW uint32_t qm_spi_reg_t::ssi_comp_version

coreKit Version ID register.

coreKit Version ID register

Definition at line 739 of file qm_soc_regs.h.

5.64.2.23 QM_RW uint32_t qm_spi_reg_t::ssienr

SSI Enable Register.

Definition at line 715 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().

5.64.2.24 QM_RW uint32_t qm_spi_reg_t::txflr

Transmit FIFO Level Register.

Definition at line 721 of file qm_soc_regs.h.

Referenced by qm_spi_irq_transfer_terminate().

5.64.2.25 QM_RW uint32_t qm_spi_reg_t::txftlr

Transmit FIFO Threshold Level.

Definition at line 719 of file qm_soc_regs.h.

Referenced by qm_spi_irq_transfer().

5.64.2.26 QM_RW uint32_t qm_spi_reg_t::txoicr

Tx FIFO Overflow Interrupt Clear Register.

Definition at line 728 of file qm_soc_regs.h.

5.65 qm_spi_transfer_t Struct Reference

SPI synchronous transfer type.

```
#include <qm_spi.h>
```

Data Fields

- void * [tx](#)
Write data.
- void * [rx](#)
Read data.
- uint16_t [tx_len](#)
Number of data frames to write.
- uint16_t [rx_len](#)
Number of data frames to read.

5.65.1 Detailed Description

SPI synchronous transfer type.

If the frame size is 8 bits or less, 1 byte is needed per data frame. If the frame size is 9-16 bits, 2 bytes are needed per data frame and frames of more than 16 bits require 4 bytes. In each case, the least significant bits are sent while the extra bits are discarded. The most significant bits of the frame are sent first.

Definition at line 181 of file qm_spi.h.

5.65.2 Field Documentation

5.65.2.1 void* qm_spi_transfer_t::rx

Read data.

Definition at line 183 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.65.2.2 uint16_t qm_spi_transfer_t::rx_len

Number of data frames to read.

Definition at line 185 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.65.2.3 void* qm_spi_transfer_t::tx

Write data.

Definition at line 182 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.65.2.4 uint16_t qm_spi_transfer_t::tx_len

Number of data frames to write.

Definition at line 184 of file qm_spi.h.

Referenced by qm_spi_transfer().

5.66 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.

5.66.1 Detailed Description

SS ADC configuration type.

Definition at line 98 of file qm_ss_adc.h.

5.66.2 Field Documentation

5.66.2.1 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().

5.67 qm_ss_adc_context_t Struct Reference

SS ADC context type.

```
#include <qm_sensor_regs.h>
```

Data Fields

- uint32_t [adc_set](#)
ADC settings.
- uint32_t [adc_divseqstat](#)
ADC clock divider and sequencer status.

- uint32_t [adc_seq](#)
ADC sequencer entry.
- uint32_t [adc_ctrl](#)
ADC control.

5.67.1 Detailed Description

SS ADC context type.

The application should not modify the content of this structure.

This structure is intended to be used by `qm_ss_adc_save_context` and `qm_ss_adc_restore_context` functions only.

Definition at line 564 of file `qm_sensor_regs.h`.

5.67.2 Field Documentation

5.67.2.1 uint32_t qm_ss_adc_context_t::adc_ctrl

ADC control.

Definition at line 568 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_adc_restore_context()`, and `qm_ss_adc_save_context()`.

5.67.2.2 uint32_t qm_ss_adc_context_t::adc_divseqstat

ADC clock divider and sequencer status.

Definition at line 566 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_adc_restore_context()`, and `qm_ss_adc_save_context()`.

5.67.2.3 uint32_t qm_ss_adc_context_t::adc_seq

ADC sequencer entry.

Definition at line 567 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_adc_restore_context()`, and `qm_ss_adc_save_context()`.

5.67.2.4 uint32_t qm_ss_adc_context_t::adc_set

ADC settings.

Definition at line 565 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_adc_restore_context()`, and `qm_ss_adc_save_context()`.

5.68 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.

5.68.1 Detailed Description

SS ADC transfer type.

Definition at line 111 of file `qm_ss_adc.h`.

5.68.2 Field Documentation

5.68.2.1 `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

<code>in</code>	<code>data</code>	The callback user data.
<code>in</code>	<code>error</code>	0 on success. Negative <code>errno</code> for possible error codes.
<code>in</code>	<code>status</code>	ADC status.
<code>in</code>	<code>source</code>	Interrupt callback source.

Definition at line 128 of file `qm_ss_adc.h`.

5.68.2.2 `void* qm_ss_adc_xfer_t::callback_data`

Callback user data.

Definition at line 130 of file `qm_ss_adc.h`.

5.68.2.3 `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()`.

5.68.2.4 `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()`.

5.68.2.5 `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()`.

5.68.2.6 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().

5.69 qm_ss_gpio_context_t Struct Reference

SS GPIO context type.

```
#include <qm_sensor_regs.h>
```

Data Fields

- uint32_t [gpio_swporta_dr](#)
Port A Data.
- uint32_t [gpio_swporta_ddr](#)
Port A Data Direction.
- uint32_t [gpio_inten](#)
Interrupt Enable.
- uint32_t [gpio_intmask](#)
Interrupt Mask.
- uint32_t [gpio_inttype_level](#)
Interrupt Type.
- uint32_t [gpio_int_polarity](#)
Interrupt Polarity.
- uint32_t [gpio_debounce](#)
Debounce Enable.
- uint32_t [gpio_ls_sync](#)
Synchronization Level.

5.69.1 Detailed Description

SS GPIO context type.

Application should not modify the content. This structure is only intended to be used by the qm_ss_gpio_save_-context and qm_ss_gpio_restore_context functions.

Definition at line 180 of file qm_sensor_regs.h.

5.69.2 Field Documentation

5.69.2.1 uint32_t qm_ss_gpio_context_t::gpio_debounce

Debounce Enable.

Definition at line 187 of file qm_sensor_regs.h.

Referenced by qm_ss_gpio_restore_context(), and qm_ss_gpio_save_context().

5.69.2.2 uint32_t qm_ss_gpio_context_t::gpio_int_polarity

Interrupt Polarity.

Definition at line 186 of file qm_sensor_regs.h.

Referenced by qm_ss_gpio_restore_context(), and qm_ss_gpio_save_context().

5.69.2.3 `uint32_t qm_ss_gpio_context_t::gpio_inten`

Interrupt Enable.

Definition at line 183 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_gpio_restore_context()`, and `qm_ss_gpio_save_context()`.

5.69.2.4 `uint32_t qm_ss_gpio_context_t::gpio_intmask`

Interrupt Mask.

Definition at line 184 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_gpio_restore_context()`, and `qm_ss_gpio_save_context()`.

5.69.2.5 `uint32_t qm_ss_gpio_context_t::gpio_inttype_level`

Interrupt Type.

Definition at line 185 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_gpio_restore_context()`, and `qm_ss_gpio_save_context()`.

5.69.2.6 `uint32_t qm_ss_gpio_context_t::gpio_ls_sync`

Synchronization Level.

Definition at line 188 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_gpio_restore_context()`, and `qm_ss_gpio_save_context()`.

5.69.2.7 `uint32_t qm_ss_gpio_context_t::gpio_swporta_ddr`

Port A Data Direction.

Definition at line 182 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_gpio_restore_context()`, and `qm_ss_gpio_save_context()`.

5.69.2.8 `uint32_t qm_ss_gpio_context_t::gpio_swporta_dr`

Port A Data.

Definition at line 181 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_gpio_restore_context()`, and `qm_ss_gpio_save_context()`.

5.70 `qm_ss_gpio_port_config_t` Struct Reference

SS 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.
- uint32_t [int_bothedge](#)
Interrupt on rising and falling edges.
- void(* [callback](#))(void *data, uint32_t int_status)
User callback.
- void * [callback_data](#)
Callback user data.

5.70.1 Detailed Description

SS GPIO port configuration type.

Each bit in the registers control a GPIO pin.

Definition at line 32 of file qm_ss_gpio.h.

5.70.2 Field Documentation

5.70.2.1 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

<i>in</i>	<i>data</i>	The callback user data.
<i>in</i>	<i>int_status</i>	Bitfield of triggered pins.

Definition at line 51 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.70.2.2 void* qm_ss_gpio_port_config_t::callback_data

Callback user data.

Definition at line 52 of file qm_ss_gpio.h.

Referenced by qm_ss_gpio_set_config().

5.70.2.3 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().

5.70.2.4 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().

5.70.2.5 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().

5.70.2.6 `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()`.

5.70.2.7 `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()`.

5.71 `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 or fast plus mode.
- [qm_ss_i2c_addr_t address_mode](#)
7 or 10 bit addressing.

5.71.1 Detailed Description

QM SS I2C configuration type.

Definition at line 87 of file `qm_ss_i2c.h`.

5.71.2 Field Documentation

5.71.2.1 `qm_ss_i2c_addr_t qm_ss_i2c_config_t::address_mode`

7 or 10 bit addressing.

Definition at line 89 of file `qm_ss_i2c.h`.

Referenced by `qm_ss_i2c_set_config()`.

5.71.2.2 `qm_ss_i2c_speed_t qm_ss_i2c_config_t::speed`

Standard, fast or fast plus mode.

Definition at line 88 of file `qm_ss_i2c.h`.

Referenced by `qm_ss_i2c_set_config()`.

5.72 `qm_ss_i2c_context_t` Struct Reference

SS I2C context type.

```
#include <qm_sensor_regs.h>
```

5.72.1 Detailed Description

SS I2C context type.

Application should not modify the content. This structure is only intended to be used by the qm_ss_gpio_save_-context and qm_ss_gpio_restore_context functions.

Definition at line 235 of file qm_sensor_regs.h.

5.73 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.

5.73.1 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 99 of file qm_ss_i2c.h.

5.73.2 Field Documentation

5.73.2.1 void(* qm_ss_i2c_transfer_t::callback)(void *data, int rc, qm_ss_i2c_status_t status, uint32_t len)

User callback.

Parameters

in	<i>data</i>	User defined data.
in	<i>rc</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	I2C status.
in	<i>len</i>	Length of the transfer if successful, 0 otherwise.

Definition at line 115 of file `qm_ss_i2c.h`.

5.73.2.2 `void* qm_ss_i2c_transfer_t::callback_data`

User callback data.

Definition at line 117 of file `qm_ss_i2c.h`.

5.73.2.3 `uint8_t* qm_ss_i2c_transfer_t::rx`

Read data.

Definition at line 102 of file `qm_ss_i2c.h`.

5.73.2.4 `uint32_t qm_ss_i2c_transfer_t::rx_len`

Read buffer length.

Definition at line 103 of file `qm_ss_i2c.h`.

Referenced by `qm_ss_i2c_master_irq_transfer()`.

5.73.2.5 `bool qm_ss_i2c_transfer_t::stop`

Generate master STOP.

Definition at line 104 of file `qm_ss_i2c.h`.

5.73.2.6 `uint8_t* qm_ss_i2c_transfer_t::tx`

Write data.

Definition at line 100 of file `qm_ss_i2c.h`.

5.73.2.7 `uint32_t qm_ss_i2c_transfer_t::tx_len`

Write data length.

Definition at line 101 of file `qm_ss_i2c.h`.

5.74 `qm_ss_spi_async_transfer_t` Struct Reference

SPI asynchronous transfer type.

```
#include <qm_ss_spi.h>
```

Data Fields

- `void * tx`
Write data.
- `void * rx`
Read data.
- `uint16_t tx_len`
Number of data frames to write.
- `uint16_t rx_len`
Number of data frames to read.

- void(* [callback](#))(void *data, int error, [qm_ss_spi_status_t](#) status, uint16_t len)
Transfer callback.
- void * [callback_data](#)
Callback user data.

5.74.1 Detailed Description

SPI asynchronous transfer type.

If the frame size is 8 bits or less, 1 byte is needed per data frame. If the frame size is 9-16 bits, 2 bytes are needed per data frame and frames of more than 16 bits require 4 bytes. In each case, the least significant bits are sent while the extra bits are discarded. The most significant bits of the frame are sent first.

Definition at line 153 of file `qm_ss_spi.h`.

5.74.2 Field Documentation

5.74.2.1 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	<i>data</i>	The callback user data.
in	<i>error</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	The SPI module status.
in	<i>len</i>	The amount of frames transmitted.

Definition at line 171 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_irq_transfer_terminate()`.

5.74.2.2 void* [qm_ss_spi_async_transfer_t::rx](#)

Read data.

Definition at line 155 of file `qm_ss_spi.h`.

5.74.2.3 uint16_t [qm_ss_spi_async_transfer_t::rx_len](#)

Number of data frames to read.

Definition at line 157 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_irq_transfer()`, and `qm_ss_spi_irq_transfer_terminate()`.

5.74.2.4 void* [qm_ss_spi_async_transfer_t::tx](#)

Write data.

Definition at line 154 of file `qm_ss_spi.h`.

5.74.2.5 uint16_t [qm_ss_spi_async_transfer_t::tx_len](#)

Number of data frames to write.

Definition at line 156 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_irq_transfer()`, and `qm_ss_spi_irq_transfer_terminate()`.

5.75 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](#)
SCK = SPI_clock/clk_divider.

5.75.1 Detailed Description

SPI configuration type.

Definition at line 132 of file qm_ss_spi.h.

5.75.2 Field Documentation

5.75.2.1 uint16_t qm_ss_spi_config_t::clk_divider

SCK = SPI_clock/clk_divider.

A value of 0 will disable SCK. The LSB of this value is ignored.

Definition at line 141 of file qm_ss_spi.h.

Referenced by qm_ss_spi_set_config().

5.76 qm_ss_spi_context_t Struct Reference

Sensor Subsystem SPI context type.

```
#include <qm_sensor_regs.h>
```

Data Fields

- [uint32_t spi_ctrl](#)
Control Register.
- [uint32_t spi_spie](#)
SPI Enable Register.
- [uint32_t spi_timing](#)
Timing Register.

5.76.1 Detailed Description

Sensor Subsystem SPI context type.

Applications should not modify the content. This structure is only intended to be used by the `qm_ss_spi_save_context` and `qm_ss_spi_restore_context` functions.

Definition at line 697 of file `qm_sensor_regs.h`.

5.76.2 Field Documentation

5.76.2.1 `uint32_t qm_ss_spi_context_t::spi_ctrl`

Control Register.

Definition at line 698 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_spi_restore_context()`, and `qm_ss_spi_save_context()`.

5.76.2.2 `uint32_t qm_ss_spi_context_t::spi_spie`

SPI Enable Register.

Definition at line 699 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_spi_restore_context()`, and `qm_ss_spi_save_context()`.

5.76.2.3 `uint32_t qm_ss_spi_context_t::spi_timing`

Timing Register.

Definition at line 700 of file `qm_sensor_regs.h`.

Referenced by `qm_ss_spi_restore_context()`, and `qm_ss_spi_save_context()`.

5.77 qm_ss_spi_transfer_t Struct Reference

SPI synchronous transfer type.

```
#include <qm_ss_spi.h>
```

Data Fields

- `void * tx`
Write data.
- `void * rx`
Read data.
- `uint16_t tx_len`
Number of data frames to write.
- `uint16_t rx_len`
Number of data frames to read.

5.77.1 Detailed Description

SPI synchronous transfer type.

If the frame size is 8 bits or less, 1 byte is needed per data frame. If the frame size is 9-16 bits, 2 bytes are needed per data frame and frames of more than 16 bits require 4 bytes. In each case, the least significant bits are sent while the extra bits are discarded. The most significant bits of the frame are sent first.

Definition at line 185 of file `qm_ss_spi.h`.

5.77.2 Field Documentation

5.77.2.1 void* qm_ss_spi_transfer_t::rx

Read data.

Definition at line 187 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.77.2.2 uint16_t qm_ss_spi_transfer_t::rx_len

Number of data frames to read.

Definition at line 189 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.77.2.3 void* qm_ss_spi_transfer_t::tx

Write data.

Definition at line 186 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.77.2.4 uint16_t qm_ss_spi_transfer_t::tx_len

Number of data frames to write.

Definition at line 188 of file qm_ss_spi.h.

Referenced by qm_ss_spi_transfer().

5.78 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.

5.78.1 Detailed Description

Sensor Subsystem Timer Configuration Type.

Definition at line 21 of file qm_ss_timer.h.

5.78.2 Field Documentation

5.78.2.1 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().

5.78.2.2 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().

5.78.2.3 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().

5.78.2.4 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().

5.78.2.5 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().

5.78.2.6 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().

5.79 qm_uart_config_t Struct Reference

UART configuration structure 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.

5.79.1 Detailed Description

UART configuration structure type.

Definition at line 67 of file `qm_uart.h`.

5.79.2 Field Documentation

5.79.2.1 `uint32_t qm_uart_config_t::baud_divisor`

Baud Divisor.

Definition at line 69 of file `qm_uart.h`.

Referenced by `qm_uart_set_config()`.

5.79.2.2 `bool qm_uart_config_t::hw_fc`

Hardware Automatic Flow Control.

Definition at line 70 of file `qm_uart.h`.

Referenced by `qm_uart_set_config()`.

5.79.2.3 `qm_uart_lc_t qm_uart_config_t::line_control`

Line control (enum).

Definition at line 68 of file `qm_uart.h`.

Referenced by `qm_uart_set_config()`.

5.80 `qm_uart_context_t` Struct Reference

UART context to be saved between sleep/resume.

```
#include <qm_soc_regs.h>
```

Data Fields

- [uint32_t ier](#)
Interrupt Enable Register.
- [uint32_t dlh](#)
Divisor Latch High.
- [uint32_t dll](#)
Divisor Latch Low.
- [uint32_t lcr](#)
Line Control.
- [uint32_t mcr](#)

- Modem Control.*
- uint32_t [scr](#)
- Scratchpad.*
- uint32_t [htx](#)
- Halt Transmission.*
- uint32_t [dlf](#)
- Divisor Latch Fraction.*

5.80.1 Detailed Description

UART context to be saved between sleep/resume.

Application should not modify the content. This structure is only intended to be used by the `qm_uart_save_context` and `qm_uart_restore_context` functions.

Definition at line 993 of file `qm_soc_regs.h`.

5.80.2 Field Documentation

5.80.2.1 uint32_t qm_uart_context_t::dlf

Divisor Latch Fraction.

Definition at line 1001 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.2 uint32_t qm_uart_context_t::dlh

Divisor Latch High.

Definition at line 995 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.3 uint32_t qm_uart_context_t::dll

Divisor Latch Low.

Definition at line 996 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.4 uint32_t qm_uart_context_t::htx

Halt Transmission.

Definition at line 1000 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.5 uint32_t qm_uart_context_t::ier

Interrupt Enable Register.

Definition at line 994 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.6 uint32_t qm_uart_context_t::lcr

Line Control.

Definition at line 997 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.7 `uint32_t qm_uart_context_t::mcr`

Modem Control.

Definition at line 998 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.80.2.8 `uint32_t qm_uart_context_t::scr`

Scratchpad.

Definition at line 999 of file `qm_soc_regs.h`.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.81 `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 dmaasa`
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.*

5.81.1 Detailed Description

UART register map.

Definition at line 658 of file qm_soc_regs.h.

5.81.2 Field Documentation

5.81.2.1 QM_RW uint32_t qm_uart_reg_t::de_en

Driver Output Enable Register.

Definition at line 674 of file qm_soc_regs.h.

5.81.2.2 QM_RW uint32_t qm_uart_reg_t::det

Driver Output Enable Timing Register.

Definition at line 676 of file qm_soc_regs.h.

5.81.2.3 QM_RW uint32_t qm_uart_reg_t::dlf

Divisor Latch Fraction.

Definition at line 678 of file qm_soc_regs.h.

Referenced by `qm_uart_restore_context()`, `qm_uart_save_context()`, and `qm_uart_set_config()`.

5.81.2.4 QM_RW uint32_t qm_uart_reg_t::dma_sa

DMA Software Acknowledge.

Definition at line 672 of file qm_soc_regs.h.

5.81.2.5 QM_RW uint32_t qm_uart_reg_t::htx

Halt Transmission.

Definition at line 671 of file qm_soc_regs.h.

Referenced by `qm_uart_restore_context()`, and `qm_uart_save_context()`.

5.81.2.6 QM_RW uint32_t qm_uart_reg_t::ier_dlh

Interrupt Enable / Divisor Latch High.

Definition at line 661 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()`, `qm_uart_restore_context()`, `qm_uart_save_context()`, and `qm_uart_set_config()`.

5.81.2.7 QM_RW uint32_t qm_uart_reg_t::iir_fcr

Interrupt Identification / FIFO Control.

Definition at line 662 of file qm_soc_regs.h.

Referenced by qm_uart_dma_read(), qm_uart_dma_write(), qm_uart_irq_read(), qm_uart_irq_write(), qm_uart_restore_context(), and qm_uart_set_config().

5.81.2.8 QM_RW uint32_t qm_uart_reg_t::lcr

Line Control.

Definition at line 663 of file qm_soc_regs.h.

Referenced by qm_uart_restore_context(), qm_uart_save_context(), and qm_uart_set_config().

5.81.2.9 QM_RW uint32_t qm_uart_reg_t::lcr_ext

Line Extended Control Register.

Definition at line 681 of file qm_soc_regs.h.

5.81.2.10 QM_RW uint32_t qm_uart_reg_t::lsr

Line Status.

Definition at line 665 of file qm_soc_regs.h.

Referenced by qm_uart_get_status(), qm_uart_read(), qm_uart_set_config(), qm_uart_write(), and qm_uart_write_buffer().

5.81.2.11 QM_RW uint32_t qm_uart_reg_t::mcr

MODEM Control.

Definition at line 664 of file qm_soc_regs.h.

Referenced by qm_uart_restore_context(), qm_uart_save_context(), and qm_uart_set_config().

5.81.2.12 QM_RW uint32_t qm_uart_reg_t::msr

MODEM Status.

Definition at line 666 of file qm_soc_regs.h.

5.81.2.13 QM_RW uint32_t qm_uart_reg_t::rar

Receive Address Register.

Definition at line 679 of file qm_soc_regs.h.

5.81.2.14 QM_RW uint32_t qm_uart_reg_t::rbr_thr_dll

Rx Buffer/ Tx Holding/ Div Latch Low.

Definition at line 660 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_restore_context(), qm_uart_save_context(), qm_uart_set_config(), qm_uart_write(), qm_uart_write_buffer(), and qm_uart_write_non_block().

5.81.2.15 QM_RW uint32_t qm_uart_reg_t::re_en

Receiver Output Enable Register.

Definition at line 675 of file qm_soc_regs.h.

5.81.2.16 QM_RW uint32_t qm_uart_reg_t::scr

Scratchpad.

Definition at line 667 of file qm_soc_regs.h.

Referenced by qm_uart_get_status(), qm_uart_restore_context(), and qm_uart_save_context().

5.81.2.17 QM_RW uint32_t qm_uart_reg_t::tar

Transmit Address Register.

Definition at line 680 of file qm_soc_regs.h.

5.81.2.18 QM_RW uint32_t qm_uart_reg_t::tat

TurnAround Timing Register.

Definition at line 677 of file qm_soc_regs.h.

5.81.2.19 QM_RW uint32_t qm_uart_reg_t::tcr

Transceiver Control Register.

Definition at line 673 of file qm_soc_regs.h.

5.81.2.20 QM_RW uint32_t qm_uart_reg_t::usr

UART Status.

Definition at line 669 of file qm_soc_regs.h.

5.82 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.

5.82.1 Detailed Description

UART asynchronous transfer structure.

Definition at line 76 of file qm_uart.h.

5.82.2 Field Documentation

5.82.2.1 void(* qm_uart_transfer_t::callback)(void *data, int error, qm_uart_status_t status, uint32_t len)

Transfer callback.

Parameters

in	<i>data</i>	Callback user data.
in	<i>error</i>	0 on success. Negative errno for possible error codes.
in	<i>status</i>	UART module status
in	<i>len</i>	Length of the UART transfer if successful, 0 otherwise.

Definition at line 89 of file `qm_uart.h`.

Referenced by `qm_uart_irq_read_terminate()`, and `qm_uart_irq_write_terminate()`.

5.82.2.2 void* qm_uart_transfer_t::callback_data

Callback identifier.

Definition at line 91 of file `qm_uart.h`.

Referenced by `qm_uart_irq_read_terminate()`, and `qm_uart_irq_write_terminate()`.

5.82.2.3 uint8_t* qm_uart_transfer_t::data

Pre-allocated write or read buffer.

Definition at line 77 of file `qm_uart.h`.

Referenced by `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

5.82.2.4 uint32_t qm_uart_transfer_t::data_len

Number of bytes to transfer.

Definition at line 78 of file `qm_uart.h`.

Referenced by `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

5.83 qm_usb_ep_config_t Struct Reference

USB Endpoint Configuration.

```
#include <qm_usb.h>
```

Data Fields

- [qm_usb_ep_type_t](#) *type*
Endpoint type.
- `uint16_t` [max_packet_size](#)
Endpoint max packet size.
- `void(*` [callback](#) `)(void *data, int error, qm_usb_ep_idx_t ep, qm_usb_ep_status_t status)`
Callback for the USB Endpoint status.
- `void *` [callback_data](#)
Callback user data.

5.83.1 Detailed Description

USB Endpoint Configuration.

Definition at line 51 of file `qm_usb.h`.

5.83.2 Field Documentation

5.83.2.1 void(* qm_usb_ep_config_t::callback)(void *data, int error, qm_usb_ep_idx_t ep, qm_usb_ep_status_t status)

Callback for the USB Endpoint status.

Called for notifying of data received and available to application on this endpoint.

Parameters

in	<i>data</i>	The callback user data.
in	<i>error</i>	0 on success. Negative errno for possible error codes.
in	<i>ep</i>	Endpoint index.
in	<i>status</i>	USB Endpoint status.

Definition at line 69 of file qm_usb.h.

5.83.2.2 void* qm_usb_ep_config_t::callback_data

Callback user data.

Definition at line 71 of file qm_usb.h.

5.83.2.3 uint16_t qm_usb_ep_config_t::max_packet_size

Endpoint max packet size.

Definition at line 53 of file qm_usb.h.

Referenced by qm_usb_ep_set_config().

5.83.2.4 qm_usb_ep_type_t qm_usb_ep_config_t::type

Endpoint type.

Definition at line 52 of file qm_usb.h.

Referenced by qm_usb_ep_set_config().

5.84 qm_usb_in_ep_reg_t Struct Reference

USB register map.

```
#include <qm_soc_regs.h>
```

5.84.1 Detailed Description

USB register map.

IN Endpoint Registers.

Definition at line 1962 of file qm_soc_regs.h.

5.85 qm_usb_out_ep_reg_t Struct Reference

OUT Endpoint Registers.

```
#include <qm_soc_regs.h>
```

5.85.1 Detailed Description

OUT Endpoint Registers.

Definition at line 1974 of file qm_soc_regs.h.

5.86 qm_usb_reg_t Struct Reference

USB Register block type.

```
#include <qm_soc_regs.h>
```

Data Fields

- QM_RW uint32_t [gotgctl](#)
OTG Control.
- QM_RW uint32_t [gotgint](#)
OTG Interrupt.
- QM_RW uint32_t [gahbcfg](#)
AHB Configuration.
- QM_RW uint32_t [gusbcbfg](#)
USB Configuration.
- QM_RW uint32_t [grstctl](#)
Reset Register.
- QM_RW uint32_t [gintsts](#)
Interrupt Status.
- QM_RW uint32_t [gintmsk](#)
Interrupt Mask.
- QM_R uint32_t [grxstsr](#)
Receive Status Read/Pop.
- QM_R uint32_t [grxstsp](#)
Receive Status Read/Pop.
- QM_R uint32_t [grxfsize](#)
Receive FIFO Size.
- QM_R uint32_t [gnptxfsize](#)
Non-periodic Transmit FIFO Size.
- QM_R uint32_t [gsnpsid](#)
Synopsys ID.
- QM_R uint32_t [ghwcfg1](#)
HW config - Endpoint direction.
- QM_R uint32_t [ghwcfg2](#)
HW config 2.
- QM_R uint32_t [ghwcfg3](#)
HW config 3.
- QM_R uint32_t [ghwcfg4](#)
HW config 4.
- QM_RW uint32_t [gdfifocfg](#)
Global DFIFO Configuration.
- QM_RW uint32_t [dcfg](#)
Device config.
- QM_RW uint32_t [dctl](#)
Device control.
- QM_RW uint32_t [dsts](#)
Device Status.
- QM_RW uint32_t [diepmsk](#)
IN EP Common Interrupt Mask.
- QM_RW uint32_t [doepmsk](#)

OUT EP Common Interrupt Mask.

- QM_R uint32_t [daint](#)

Device Interrupt Register.

- QM_RW uint32_t [daintmsk](#)

Device Interrupt Mask Register.

- QM_RW uint32_t [dvbusdis](#)

VBUS discharge time register.

- QM_RW uint32_t [dvbuspulse](#)

Device VBUS discharge time.

- QM_RW uint32_t [dthrctl](#)

Device Threshold Ctrl.

- QM_RW uint32_t [diepempmsk](#)

IN EP FIFO Empty Intr Mask.

5.86.1 Detailed Description

USB Register block type.

Definition at line 1987 of file qm_soc_regs.h.

5.86.2 Field Documentation

5.86.2.1 QM_R uint32_t qm_usb_reg_t::daint

Device Interrupt Register.

Definition at line 2019 of file qm_soc_regs.h.

5.86.2.2 QM_RW uint32_t qm_usb_reg_t::daintmsk

Device Interrupt Mask Register.

Definition at line 2020 of file qm_soc_regs.h.

5.86.2.3 QM_RW uint32_t qm_usb_reg_t::dcfg

Device config.

Definition at line 2013 of file qm_soc_regs.h.

5.86.2.4 QM_RW uint32_t qm_usb_reg_t::dctl

Device control.

Definition at line 2014 of file qm_soc_regs.h.

5.86.2.5 QM_RW uint32_t qm_usb_reg_t::diepempmsk

IN EP FIFO Empty Intr Mask.

Definition at line 2025 of file qm_soc_regs.h.

5.86.2.6 QM_RW uint32_t qm_usb_reg_t::diepmsk

IN EP Common Interrupt Mask.

Definition at line 2017 of file qm_soc_regs.h.

5.86.2.7 QM_RW uint32_t qm_usb_reg_t::doepmsk

OUT EP Common Interrupt Mask.

Definition at line 2018 of file qm_soc_regs.h.

5.86.2.8 QM_RW uint32_t qm_usb_reg_t::dsts

Device Status.

Definition at line 2015 of file qm_soc_regs.h.

5.86.2.9 QM_RW uint32_t qm_usb_reg_t::dthrctl

Device Threshold Ctrl.

Definition at line 2024 of file qm_soc_regs.h.

5.86.2.10 QM_RW uint32_t qm_usb_reg_t::dvbusdis

VBUS discharge time register.

Definition at line 2022 of file qm_soc_regs.h.

5.86.2.11 QM_RW uint32_t qm_usb_reg_t::dvbuspulse

Device VBUS discharge time.

Definition at line 2023 of file qm_soc_regs.h.

5.86.2.12 QM_RW uint32_t qm_usb_reg_t::gahbcfg

AHB Configuration.

Definition at line 1990 of file qm_soc_regs.h.

5.86.2.13 QM_RW uint32_t qm_usb_reg_t::gdfifocfg

Global DFIFO Configuration.

Definition at line 2005 of file qm_soc_regs.h.

5.86.2.14 QM_R uint32_t qm_usb_reg_t::ghwcfg1

HW config - Endpoint direction.

Definition at line 2001 of file qm_soc_regs.h.

5.86.2.15 QM_R uint32_t qm_usb_reg_t::ghwcfg2

HW config 2.

Definition at line 2002 of file qm_soc_regs.h.

5.86.2.16 QM_R uint32_t qm_usb_reg_t::ghwcfg3

HW config 3.

Definition at line 2003 of file qm_soc_regs.h.

5.86.2.17 QM_R uint32_t qm_usb_reg_t::ghwcfg4

HW config 4.

Definition at line 2004 of file qm_soc_regs.h.

5.86.2.18 QM_RW uint32_t qm_usb_reg_t::gintmsk

Interrupt Mask.

Definition at line 1994 of file qm_soc_regs.h.

5.86.2.19 QM_RW uint32_t qm_usb_reg_t::gintsts

Interrupt Status.

Definition at line 1993 of file qm_soc_regs.h.

5.86.2.20 QM_R uint32_t qm_usb_reg_t::gnptxsiz

Non-periodic Transmit FIFO Size.

Definition at line 1998 of file qm_soc_regs.h.

5.86.2.21 QM_RW uint32_t qm_usb_reg_t::gotgctl

OTG Control.

Definition at line 1988 of file qm_soc_regs.h.

5.86.2.22 QM_RW uint32_t qm_usb_reg_t::gotgint

OTG Interrupt.

Definition at line 1989 of file qm_soc_regs.h.

5.86.2.23 QM_RW uint32_t qm_usb_reg_t::grstctl

Reset Register.

Definition at line 1992 of file qm_soc_regs.h.

5.86.2.24 QM_R uint32_t qm_usb_reg_t::grxfsiz

Receive FIFO Size.

Definition at line 1997 of file qm_soc_regs.h.

5.86.2.25 QM_R uint32_t qm_usb_reg_t::grxstsp

Receive Status Read/Pop.

Definition at line 1996 of file qm_soc_regs.h.

5.86.2.26 QM_R uint32_t qm_usb_reg_t::grxstsr

Receive Status Read/Pop.

Definition at line 1995 of file qm_soc_regs.h.

5.86.2.27 QM_R uint32_t qm_usb_reg_t::gsnpsid

Synopsys ID.

Definition at line 2000 of file qm_soc_regs.h.

5.86.2.28 QM_RW uint32_t qm_usb_reg_t::gusbcfg

USB Configuration.

Definition at line 1991 of file qm_soc_regs.h.

5.87 qm_wdt_config_t Struct Reference

QM WDT configuration type.

```
#include <qm_wdt.h>
```

Data Fields

- [uint32_t timeout](#)
Index for the WDT timeout table.
- [qm_wdt_mode_t mode](#)
Watchdog response mode.
- [bool pause_en](#)
Pause enable in LMT power state C2 and C2 Plus.
- [void\(* callback\)\(void *data\)](#)
User callback.
- [void * callback_data](#)
Callback user data.

5.87.1 Detailed Description

QM WDT configuration type.

Definition at line 39 of file qm_wdt.h.

5.87.2 Field Documentation

5.87.2.1 void(* qm_wdt_config_t::callback)(void *data)

User callback.

param[in] data Callback user data.

Definition at line 68 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.87.2.2 void* qm_wdt_config_t::callback_data

Callback user data.

Definition at line 69 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.87.2.3 qm_wdt_mode_t qm_wdt_config_t::mode

Watchdog response mode.

Definition at line 48 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.87.2.4 bool qm_wdt_config_t::pause_en

Pause enable in LMT power state C2 and C2 Plus.

When equal to 1, the WDT is paused when LMT enters the C2 state. When equal to 0, the WDT is not paused when LMT enters the C2 state.

This field applies only to Watchdogs on AON power island.

Definition at line 60 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.87.2.5 uint32_t qm_wdt_config_t::timeout

Index for the WDT timeout table.

For each instantiation of WDT there are multiple timeout values pre-programmed in hardware. Reference the SoC datasheet or register file for the table associated to the WDT being configured.

Definition at line 47 of file qm_wdt.h.

Referenced by qm_wdt_set_config().

5.88 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.

5.88.1 Detailed Description

Watchdog timer register map.

Definition at line 470 of file qm_soc_regs.h.

5.88.2 Field Documentation

5.88.2.1 QM_RW uint32_t qm_wdt_reg_t::wdt_ccvr

Current Counter Value Register.

Definition at line 473 of file qm_soc_regs.h.

5.88.2.2 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_1

Component Parameters Register 1.

Definition at line 482 of file qm_soc_regs.h.

5.88.2.3 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_2

Component Parameters.

Definition at line 480 of file qm_soc_regs.h.

5.88.2.4 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_3

Component Parameters.

Definition at line 479 of file qm_soc_regs.h.

5.88.2.5 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_4

Component Parameters.

Definition at line 478 of file qm_soc_regs.h.

5.88.2.6 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_param_5

Component Parameters.

Definition at line 477 of file qm_soc_regs.h.

5.88.2.7 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_type

Component Type Register.

Definition at line 484 of file qm_soc_regs.h.

5.88.2.8 QM_RW uint32_t qm_wdt_reg_t::wdt_comp_version

Component Version Register.

Definition at line 483 of file qm_soc_regs.h.

5.88.2.9 QM_RW uint32_t qm_wdt_reg_t::wdt_cr

Control Register.

Definition at line 471 of file qm_soc_regs.h.

5.88.2.10 QM_RW uint32_t qm_wdt_reg_t::wdt_crr

Current Restart Register.

Definition at line 474 of file qm_soc_regs.h.

5.88.2.11 QM_RW uint32_t qm_wdt_reg_t::wdt_eoi

Interrupt Clear Register.

Definition at line 476 of file qm_soc_regs.h.

5.88.2.12 QM_RW uint32_t qm_wdt_reg_t::wdt_stat

Interrupt Status Register.

Definition at line 475 of file qm_soc_regs.h.

5.88.2.13 QM_RW uint32_t qm_wdt_reg_t::wdt_torr

Timeout Range Register.

Definition at line 472 of file qm_soc_regs.h.

Index

- adc_0_cal_int_mask
 - qm_interrupt_router_reg_t, [286](#)
- adc_0_pwr_int_mask
 - qm_interrupt_router_reg_t, [286](#)
- adc_ctrl
 - qm_ss_adc_context_t, [326](#)
- adc_divseqstat
 - qm_ss_adc_context_t, [326](#)
- adc_seq
 - qm_ss_adc_context_t, [326](#)
- adc_set
 - qm_ss_adc_context_t, [326](#)
- address_mode
 - qm_i2c_config_t, [270](#)
 - qm_ss_i2c_config_t, [331](#)
- aempty_thresh
 - qm_i2s_channel_cfg_data_t, [282](#)
- afull_thresh
 - qm_i2s_channel_cfg_data_t, [282](#)
- alarm_en
 - qm_rtc_config_t, [305](#)
- alarm_val
 - qm_rtc_config_t, [305](#)
- allow_agents
 - qm_fpr_config_t, [262](#)
- Always-on Counters, [16](#)
 - QM_AONPT_EXPIRED, [17](#)
 - QM_AONPT_READY, [17](#)
 - qm_aonc_disable, [17](#)
 - qm_aonc_enable, [17](#)
 - qm_aonc_get_value, [17](#)
 - qm_aonpt_clear, [18](#)
 - qm_aonpt_get_status, [18](#)
 - qm_aonpt_get_value, [18](#)
 - qm_aonpt_reset, [19](#)
 - qm_aonpt_restore_context, [19](#)
 - qm_aonpt_save_context, [19](#)
 - qm_aonpt_set_config, [20](#)
 - qm_aonpt_status_t, [17](#)
- Analog Comparator, [21](#)
 - qm_ac_set_config, [21](#)
- aon_gpio_0_int_mask
 - qm_interrupt_router_reg_t, [286](#)
- aon_vr
 - qm_scss_pmu_reg_t, [315](#)
- aonc_cfg
 - qm_aonc_reg_t, [249](#)
- aonc_cnt
 - qm_aonc_reg_t, [249](#)
- aonpt_0_int_mask
 - qm_interrupt_router_reg_t, [286](#)
- aonpt_cfg
 - qm_aonc_reg_t, [249](#)
- aonpt_cnt
 - qm_aonc_reg_t, [249](#)
- aonpt_ctrl
 - qm_aonc_reg_t, [249](#)
- aonpt_stat
 - qm_aonc_reg_t, [249](#)
- baud_divisor
 - qm_uart_config_t, [339](#)
- baudr
 - qm_spi_context_t, [319](#)
 - qm_spi_reg_t, [321](#)
- block_size
 - qm_dma_multi_transfer_t, [257](#)
 - qm_dma_transfer_t, [257](#)
 - qm_i2s_channel_cfg_data_t, [282](#)
- buff
 - qm_i2s_buffer_link, [281](#)
- buffer_len
 - qm_i2s_channel_cfg_data_t, [282](#)
- buffer_link
 - qm_i2s_channel_cfg_data_t, [282](#)
- CLK_EXT_DIV_1
 - Clock Management, [183](#)
- CLK_EXT_DIV_2
 - Clock Management, [183](#)
- CLK_EXT_DIV_4
 - Clock Management, [183](#)
- CLK_EXT_DIV_8
 - Clock Management, [183](#)
- CLK_GPIO_DB_DIV_1
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_128
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_16
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_2
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_32
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_4
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_64
 - Clock Management, [184](#)
- CLK_GPIO_DB_DIV_8
 - Clock Management, [184](#)
- CLK_PERIPH_ADC
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_ADC_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_ALL
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_CLK
 - SoC Registers (D2000), [208](#), [209](#)

- SoC Registers (SE), [231](#)
- CLK_PERIPH_DIV_1
 - Clock Management, [185](#)
- CLK_PERIPH_DIV_2
 - Clock Management, [185](#)
- CLK_PERIPH_DIV_4
 - Clock Management, [185](#)
- CLK_PERIPH_DIV_8
 - Clock Management, [185](#)
- CLK_PERIPH_GPIO_DB
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_GPIO_INTERRUPT
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_GPIO_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_I2C_M0
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_I2C_M0_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_I2C_M1
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_I2C_M1_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [232](#)
- CLK_PERIPH_I2S
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [232](#)
- CLK_PERIPH_I2S_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [232](#)
- CLK_PERIPH_PWM_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_RTC_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_SPI_M0
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_SPI_M0_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_SPI_M1
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_SPI_M1_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [232](#)
- CLK_PERIPH_SPI_S
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#)
- CLK_PERIPH_SPI_S_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_UARTA_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_UARTB_REGISTER
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_PERIPH_WDT_REGISTER
 - SoC Registers (D2000), [208](#), [209](#)
 - SoC Registers (SE), [231](#), [232](#)
- CLK_RTC_DIV_1
 - Clock Management, [185](#), [186](#)
- CLK_RTC_DIV_1024
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_128
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_16
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_16384
 - Clock Management, [186](#), [187](#)
- CLK_RTC_DIV_2
 - Clock Management, [185](#), [186](#)
- CLK_RTC_DIV_2048
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_256
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_32
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_32768
 - Clock Management, [186](#), [187](#)
- CLK_RTC_DIV_4
 - Clock Management, [185](#), [186](#)
- CLK_RTC_DIV_4096
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_512
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_64
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_8
 - Clock Management, [185](#)–[187](#)
- CLK_RTC_DIV_8192
 - Clock Management, [186](#), [187](#)
- CLK_SYS_CRYSTAL_OSC
 - Clock Management, [188](#)
- CLK_SYS_DIV_1
 - Clock Management, [187](#), [188](#)
- CLK_SYS_DIV_128
 - Clock Management, [187](#), [188](#)
- CLK_SYS_DIV_16
 - Clock Management, [187](#)
- CLK_SYS_DIV_2
 - Clock Management, [187](#), [188](#)
- CLK_SYS_DIV_32
 - Clock Management, [187](#)

- CLK_SYS_DIV_4
 - Clock Management, [187](#), [188](#)
- CLK_SYS_DIV_64
 - Clock Management, [187](#)
- CLK_SYS_DIV_8
 - Clock Management, [187](#), [188](#)
- CLK_SYS_HYB_OSC_16MHZ
 - Clock Management, [188](#)
- CLK_SYS_HYB_OSC_32MHZ
 - Clock Management, [188](#)
- CLK_SYS_HYB_OSC_4MHZ
 - Clock Management, [188](#)
- CLK_SYS_HYB_OSC_8MHZ
 - Clock Management, [188](#)
- CLK_SYS_RTC_OSC
 - Clock Management, [188](#)
- callback
 - qm_ac_config_t, [245](#)
 - qm_adc_xfer_t, [248](#)
 - qm_aonpt_config_t, [250](#)
 - qm_gpio_port_config_t, [265](#)
 - qm_i2c_transfer_t, [279](#)
 - qm_i2s_channel_cfg_data_t, [283](#)
 - qm_mbox_config_t, [294](#)
 - qm_pic_timer_config_t, [298](#)
 - qm_pwm_config_t, [302](#)
 - qm_rtc_config_t, [305](#)
 - qm_spi_async_transfer_t, [318](#)
 - qm_ss_adc_xfer_t, [327](#)
 - qm_ss_gpio_port_config_t, [330](#)
 - qm_ss_i2c_transfer_t, [332](#)
 - qm_ss_spi_async_transfer_t, [334](#)
 - qm_ss_timer_config_t, [338](#)
 - qm_uart_transfer_t, [344](#)
 - qm_usb_ep_config_t, [345](#)
 - qm_wdt_config_t, [351](#)
- callback_data
 - qm_ac_config_t, [245](#)
 - qm_adc_xfer_t, [248](#)
 - qm_aonpt_config_t, [250](#)
 - qm_gpio_port_config_t, [266](#)
 - qm_i2c_transfer_t, [280](#)
 - qm_i2s_channel_cfg_data_t, [283](#)
 - qm_pic_timer_config_t, [299](#)
 - qm_pwm_config_t, [302](#)
 - qm_rtc_config_t, [305](#)
 - qm_spi_async_transfer_t, [318](#)
 - qm_ss_adc_xfer_t, [327](#)
 - qm_ss_gpio_port_config_t, [330](#)
 - qm_ss_i2c_transfer_t, [333](#)
 - qm_ss_timer_config_t, [338](#)
 - qm_uart_transfer_t, [345](#)
 - qm_usb_ep_config_t, [346](#)
 - qm_wdt_config_t, [351](#)
- ccr
 - qm_mvic_reg_t, [297](#)
- ccu_ext_clock_ctl
 - qm_scss_ccu_reg_t, [308](#)
- ccu_gpio_db_clk_ctl
 - qm_scss_ccu_reg_t, [308](#)
- ccu_lp_clk_ctl
 - qm_scss_ccu_reg_t, [308](#)
- ccu_mlayer_ahb_ctl
 - qm_scss_ccu_reg_t, [308](#)
- ccu_periph_clk_div_ctl0
 - qm_scss_ccu_reg_t, [308](#)
- ccu_periph_clk_gate_ctl
 - qm_scss_ccu_reg_t, [308](#)
- ccu_ss_periph_clk_gate_ctl
 - qm_scss_ccu_reg_t, [308](#)
- ccu_sys_clk_ctl
 - qm_scss_ccu_reg_t, [308](#)
- cfg_high
 - qm_dma_context_t, [253](#)
- cfg_lock
 - qm_scss_peripheral_reg_t, [314](#)
- cfg_low
 - qm_dma_context_t, [253](#)
- ch
 - qm_adc_xfer_t, [248](#)
 - qm_ss_adc_xfer_t, [327](#)
- ch_len
 - qm_adc_xfer_t, [248](#)
 - qm_ss_adc_xfer_t, [327](#)
- client_callback
 - qm_dma_channel_config_t, [252](#)
- clk_adc_set_div
 - Clock Management, [189](#)
- clk_divider
 - qm_ss_spi_config_t, [335](#)
- clk_dma_disable
 - Clock Management, [189](#)
- clk_dma_enable
 - Clock Management, [189](#)
- clk_ext_div_t
 - Clock Management, [183](#)
- clk_ext_set_div
 - Clock Management, [189](#)
- clk_gpio_db_div_t
 - Clock Management, [183](#), [184](#)
- clk_gpio_db_set_div
 - Clock Management, [190](#)
- clk_periph_disable
 - Clock Management, [190](#)
- clk_periph_div_t
 - Clock Management, [184](#), [185](#)
- clk_periph_enable
 - Clock Management, [190](#)
- clk_periph_set_div
 - Clock Management, [191](#)
- clk_periph_t
 - SoC Registers (D2000), [208](#)
 - SoC Registers (SE), [231](#)
- clk_rtc_div_t
 - Clock Management, [185](#), [186](#)
- clk_rtc_set_div

- Clock Management, 191
- clk_sys_div_t
 - Clock Management, 187
- clk_sys_get_ticks_per_us
 - Clock Management, 192
- clk_sys_mode_t
 - Clock Management, 188
- clk_sys_set_mode
 - Clock Management, 192
- clk_sys_udelay
 - Clock Management, 192
- clk_sys_usb_disable
 - Clock Management, 193
- clk_sys_usb_enable
 - Clock Management, 193
- clk_trim_apply
 - Clock Management, 193
- clk_trim_read
 - Clock Management, 194
- Clock Management, 181
 - CLK_EXT_DIV_1, 183
 - CLK_EXT_DIV_2, 183
 - CLK_EXT_DIV_4, 183
 - CLK_EXT_DIV_8, 183
 - CLK_GPIO_DB_DIV_1, 184
 - CLK_GPIO_DB_DIV_128, 184
 - CLK_GPIO_DB_DIV_16, 184
 - CLK_GPIO_DB_DIV_2, 184
 - CLK_GPIO_DB_DIV_32, 184
 - CLK_GPIO_DB_DIV_4, 184
 - CLK_GPIO_DB_DIV_64, 184
 - CLK_GPIO_DB_DIV_8, 184
 - CLK_PERIPH_DIV_1, 185
 - CLK_PERIPH_DIV_2, 185
 - CLK_PERIPH_DIV_4, 185
 - CLK_PERIPH_DIV_8, 185
 - CLK_RTC_DIV_1, 185, 186
 - CLK_RTC_DIV_1024, 185–187
 - CLK_RTC_DIV_128, 185–187
 - CLK_RTC_DIV_16, 185–187
 - CLK_RTC_DIV_16384, 186, 187
 - CLK_RTC_DIV_2, 185, 186
 - CLK_RTC_DIV_2048, 185–187
 - CLK_RTC_DIV_256, 185–187
 - CLK_RTC_DIV_32, 185–187
 - CLK_RTC_DIV_32768, 186, 187
 - CLK_RTC_DIV_4, 185, 186
 - CLK_RTC_DIV_4096, 185–187
 - CLK_RTC_DIV_512, 185–187
 - CLK_RTC_DIV_64, 185–187
 - CLK_RTC_DIV_8, 185–187
 - CLK_RTC_DIV_8192, 186, 187
 - CLK_SYS_CRYSTAL_OSC, 188
 - CLK_SYS_DIV_1, 187, 188
 - CLK_SYS_DIV_128, 187, 188
 - CLK_SYS_DIV_16, 187
 - CLK_SYS_DIV_2, 187, 188
 - CLK_SYS_DIV_32, 187
 - CLK_SYS_DIV_4, 187, 188
 - CLK_SYS_DIV_64, 187
 - CLK_SYS_DIV_8, 187, 188
 - CLK_SYS_HYB_OSC_16MHZ, 188
 - CLK_SYS_HYB_OSC_32MHZ, 188
 - CLK_SYS_HYB_OSC_4MHZ, 188
 - CLK_SYS_HYB_OSC_8MHZ, 188
 - CLK_SYS_RTC_OSC, 188
 - clk_adc_set_div, 189
 - clk_dma_disable, 189
 - clk_dma_enable, 189
 - clk_ext_div_t, 183
 - clk_ext_set_div, 189
 - clk_gpio_db_div_t, 183, 184
 - clk_gpio_db_set_div, 190
 - clk_periph_disable, 190
 - clk_periph_div_t, 184, 185
 - clk_periph_enable, 190
 - clk_periph_set_div, 191
 - clk_rtc_div_t, 185, 186
 - clk_rtc_set_div, 191
 - clk_sys_div_t, 187
 - clk_sys_get_ticks_per_us, 192
 - clk_sys_mode_t, 188
 - clk_sys_set_mode, 192
 - clk_sys_udelay, 192
 - clk_sys_usb_disable, 193
 - clk_sys_usb_enable, 193
 - clk_trim_apply, 193
 - clk_trim_read, 194
 - get_i2c_clk_freq_in_mhz, 194
- cmp_en
 - qm_ac_config_t, 245
 - qm_scss_cmp_reg_t, 310
- cmp_pwr
 - qm_scss_cmp_reg_t, 310
- cmp_ref_pol
 - qm_scss_cmp_reg_t, 310
- cmp_ref_sel
 - qm_scss_cmp_reg_t, 310
- cmp_stat_clr
 - qm_scss_cmp_reg_t, 310
- comparator_0_host_halt_int_mask
 - qm_interrupt_router_reg_t, 286
- comparator_0_host_int_mask
 - qm_interrupt_router_reg_t, 286
- comparator_0_ss_halt_int_mask
 - qm_interrupt_router_reg_t, 287
- comparator_0_ss_int_mask
 - qm_interrupt_router_reg_t, 287
- con
 - qm_i2c_context_t, 271
- controlreg
 - qm_pwm_channel_t, 301
 - qm_pwm_context_t, 303
- cotps
 - qm_scss_info_reg_t, 313
- count

- qm_aonpt_config_t, 250
- qm_ss_timer_config_t, 338
- ctrl
 - qm_flash_context_t, 259
 - qm_flash_reg_t, 260
- ctrl_low
 - qm_dma_context_t, 253
- ctrlr0
 - qm_spi_context_t, 319
 - qm_spi_reg_t, 321
- ctrlr1
 - qm_spi_reg_t, 321
- currentvalue
 - qm_pwm_channel_t, 301
- DMA
 - QM_DMA_BURST_TRANS_LENGTH_1, 23
 - QM_DMA_BURST_TRANS_LENGTH_128, 23
 - QM_DMA_BURST_TRANS_LENGTH_16, 23
 - QM_DMA_BURST_TRANS_LENGTH_256, 23
 - QM_DMA_BURST_TRANS_LENGTH_32, 23
 - QM_DMA_BURST_TRANS_LENGTH_4, 23
 - QM_DMA_BURST_TRANS_LENGTH_64, 23
 - QM_DMA_BURST_TRANS_LENGTH_8, 23
 - QM_DMA_HANDSHAKE_POLARITY_HIGH, 23
 - QM_DMA_HANDSHAKE_POLARITY_LOW, 23
 - QM_DMA_MEMORY_TO_MEMORY, 23
 - QM_DMA_MEMORY_TO_PERIPHERAL, 23
 - QM_DMA_PERIPHERAL_TO_MEMORY, 23
 - QM_DMA_TRANS_WIDTH_128, 24
 - QM_DMA_TRANS_WIDTH_16, 24
 - QM_DMA_TRANS_WIDTH_256, 24
 - QM_DMA_TRANS_WIDTH_32, 24
 - QM_DMA_TRANS_WIDTH_64, 24
 - QM_DMA_TRANS_WIDTH_8, 24
 - QM_DMA_TYPE_MULTI_CONT, 24
 - QM_DMA_TYPE_MULTI_LL, 24
 - QM_DMA_TYPE_MULTI_LL_CIRCULAR, 24
 - QM_DMA_TYPE_SINGLE, 24
- DMA_HW_IF_I2C_MASTER_0_RX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_I2C_MASTER_0_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_I2C_MASTER_1_RX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_I2C_MASTER_1_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_I2S_CAPTURE
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_I2S_PLAYBACK
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_SPI_MASTER_0_RX
 - SoC Registers (SE), 233
- DMA_HW_IF_SPI_MASTER_0_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_SPI_MASTER_1_RX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_SPI_MASTER_1_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_SPI_SLAVE_RX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_SPI_SLAVE_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_UART_A_RX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_UART_A_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_UART_B_RX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA_HW_IF_UART_B_TX
 - SoC Registers (D2000), 210
 - SoC Registers (SE), 233
- DMA, 22
 - qm_dma_burst_length_t, 23
 - qm_dma_channel_direction_t, 23
 - qm_dma_channel_set_config, 24
 - qm_dma_handshake_polarity_t, 23
 - qm_dma_init, 25
 - qm_dma_multi_transfer_set_config, 25
 - qm_dma_restore_context, 26
 - qm_dma_save_context, 26
 - qm_dma_transfer_mem_to_mem, 27
 - qm_dma_transfer_set_config, 27
 - qm_dma_transfer_start, 28
 - qm_dma_transfer_terminate, 28
 - qm_dma_transfer_type_t, 23
 - qm_dma_transfer_width_t, 24
- daint
 - qm_usb_reg_t, 348
- daintmsk
 - qm_usb_reg_t, 348
- data
 - qm_mbox_msg_t, 295
 - qm_uart_transfer_t, 345
- data_len
 - qm_uart_transfer_t, 345
- dcfg
 - qm_usb_reg_t, 348
- dctl
 - qm_usb_reg_t, 348
- de_en
 - qm_uart_reg_t, 342

- dest
 - qm_mbox_config_t, 294
- destination_address
 - qm_dma_multi_transfer_t, 257
 - qm_dma_transfer_t, 258
- det
 - qm_uart_reg_t, 342
- diepempmsk
 - qm_usb_reg_t, 348
- diepmsk
 - qm_usb_reg_t, 348
- direction
 - qm_gpio_port_config_t, 266
 - qm_ss_gpio_port_config_t, 330
- dlf
 - qm_uart_context_t, 340
 - qm_uart_reg_t, 342
- dlh
 - qm_uart_context_t, 340
- dll
 - qm_uart_context_t, 340
- dma_0_error_int_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_0_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_1_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_2_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_3_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_4_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_5_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_6_mask
 - qm_interrupt_router_reg_t, 287
- dma_0_int_7_mask
 - qm_interrupt_router_reg_t, 288
- dma_link_head
 - qm_i2s_buffer_link, 281
- dmacr
 - qm_spi_reg_t, 321
- dmardlr
 - qm_spi_reg_t, 321
- dmasa
 - qm_uart_reg_t, 342
- dmatdlr
 - qm_spi_reg_t, 321
- doepmsk
 - qm_usb_reg_t, 348
- dotps
 - qm_scss_info_reg_t, 313
- dr
 - qm_spi_reg_t, 322
- dsts
 - qm_usb_reg_t, 349
- dthrctl
 - qm_usb_reg_t, 349
- dvbusdis
 - qm_usb_reg_t, 349
- dvbuspulse
 - qm_usb_reg_t, 349
- en_mask
 - qm_fpr_config_t, 262
- enable
 - qm_i2c_context_t, 271
- eoi
 - qm_mvic_reg_t, 297
 - qm_pwm_channel_t, 301
- Error Codes, 8
- FPR
 - FPR_VIOL_MODE_INTERRUPT, 38
 - FPR_VIOL_MODE_PROBE, 38
 - FPR_VIOL_MODE_RESET, 38
 - QM_FPR_DISABLE, 38
 - QM_FPR_DMA, 38
 - QM_FPR_ENABLE, 38
 - QM_FPR_HOST_PROCESSOR, 38
 - QM_FPR_LOCK_DISABLE, 38
 - QM_FPR_LOCK_ENABLE, 38
 - QM_FPR_OTHER_AGENTS, 38
 - QM_FPR_SENSOR_SUBSYSTEM, 38
 - QM_MAIN_FLASH_DATA, 37
 - QM_MAIN_FLASH_NUM, 37
 - QM_MAIN_FLASH_SYSTEM, 37
- FPR_VIOL_MODE_INTERRUPT
 - FPR, 38
- FPR_VIOL_MODE_PROBE
 - FPR, 38
- FPR_VIOL_MODE_RESET
 - FPR, 38
- FPR, 37
 - qm_flash_region_type_t, 37
 - qm_fpr_en_t, 37
 - qm_fpr_read_allow_t, 38
 - qm_fpr_restore_context, 38
 - qm_fpr_save_context, 39
 - qm_fpr_set_config, 39
 - qm_fpr_set_violation_policy, 39
 - qm_fpr_viol_mode_t, 38
- Flash, 30
 - QM_FLASH_REGION_DATA, 31
 - QM_FLASH_REGION_NUM, 31
 - QM_FLASH_REGION_OTP, 31
 - QM_FLASH_REGION_SYS, 31
 - QM_FLASH_WRITE_DISABLE, 30
 - QM_FLASH_WRITE_ENABLE, 30
 - qm_flash_disable_t, 30
 - qm_flash_mass_erase, 31
 - qm_flash_page_erase, 31
 - qm_flash_page_update, 32
 - qm_flash_page_write, 32
 - qm_flash_region_t, 31
 - qm_flash_restore_context, 34

- qm_flash_save_context, [34](#)
- qm_flash_set_config, [35](#)
- qm_flash_word_write, [35](#)
- flash_mpr_0_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- flash_mpr_1_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- flash_stts
 - qm_flash_reg_t, [260](#)
- flash_wr_ctrl
 - qm_flash_reg_t, [260](#)
- flash_wr_data
 - qm_flash_reg_t, [260](#)
- fpr_rd_cfg
 - qm_flash_reg_t, [261](#)
 - qm_fpr_context_t, [263](#)
- fs
 - qm_scss_info_reg_t, [313](#)
- fs_scl_hcnt
 - qm_i2c_context_t, [271](#)
- fs_scl_lcnt
 - qm_i2c_context_t, [271](#)
- fs_spklen
 - qm_i2c_context_t, [271](#)
- GPIO
 - QM_GPIO_HIGH, [42](#)
 - QM_GPIO_LOW, [42](#)
 - QM_GPIO_STATE_NUM, [42](#)
- GPIO, [42](#)
 - qm_gpio_clear_pin, [43](#)
 - qm_gpio_read_pin, [43](#)
 - qm_gpio_read_port, [43](#)
 - qm_gpio_restore_context, [44](#)
 - qm_gpio_save_context, [44](#)
 - qm_gpio_set_config, [44](#)
 - qm_gpio_set_pin, [45](#)
 - qm_gpio_set_pin_state, [45](#)
 - qm_gpio_state_t, [42](#)
 - qm_gpio_write_port, [46](#)
- gahbcfg
 - qm_usb_reg_t, [349](#)
- gdfifocfg
 - qm_usb_reg_t, [349](#)
- get_i2c_clk_freq_in_mhz
 - Clock Management, [194](#)
- ghwcfg1
 - qm_usb_reg_t, [349](#)
- ghwcfg2
 - qm_usb_reg_t, [349](#)
- ghwcfg3
 - qm_usb_reg_t, [349](#)
- ghwcfg4
 - qm_usb_reg_t, [349](#)
- gintmsk
 - qm_usb_reg_t, [349](#)
- gintsts
 - qm_usb_reg_t, [350](#)
- gnptxfisz
 - qm_usb_reg_t, [350](#)
- gotgctl
 - qm_usb_reg_t, [350](#)
- gotgint
 - qm_usb_reg_t, [350](#)
- gp0
 - qm_scss_gp_reg_t, [311](#)
- gp1
 - qm_scss_gp_reg_t, [311](#)
- gp2
 - qm_scss_gp_reg_t, [311](#)
- gp3
 - qm_scss_gp_reg_t, [311](#)
- gpio_0_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- gpio_config_reg1
 - qm_gpio_reg_t, [267](#)
- gpio_config_reg2
 - qm_gpio_reg_t, [267](#)
- gpio_debounce
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [268](#)
 - qm_ss_gpio_context_t, [328](#)
- gpio_ext_porta
 - qm_gpio_reg_t, [268](#)
- gpio_id_code
 - qm_gpio_reg_t, [268](#)
- gpio_int_bothedge
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [268](#)
- gpio_int_polarity
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [268](#)
 - qm_ss_gpio_context_t, [328](#)
- gpio_inten
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [268](#)
 - qm_ss_gpio_context_t, [328](#)
- gpio_intmask
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [268](#)
 - qm_ss_gpio_context_t, [329](#)
- gpio_intstatus
 - qm_gpio_reg_t, [268](#)
- gpio_inttype_level
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [268](#)
 - qm_ss_gpio_context_t, [329](#)
- gpio_ls_sync
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [269](#)
 - qm_ss_gpio_context_t, [329](#)
- gpio_porta_eoi
 - qm_gpio_reg_t, [269](#)
- gpio_raw_intstatus
 - qm_gpio_reg_t, [269](#)
- gpio_swporta_ctl
 - qm_gpio_context_t, [264](#)

- gpio_swporta_dds
 - qm_gpio_context_t, [264](#)
 - qm_gpio_reg_t, [269](#)
 - qm_ss_gpio_context_t, [329](#)
- gpio_swporta_dr
 - qm_gpio_context_t, [265](#)
 - qm_gpio_reg_t, [269](#)
 - qm_ss_gpio_context_t, [329](#)
- gpio_ver_id_code
 - qm_gpio_reg_t, [269](#)
- gps0
 - qm_scss_gp_reg_t, [311](#)
- gps1
 - qm_scss_gp_reg_t, [311](#)
- gps2
 - qm_scss_gp_reg_t, [312](#)
- gps3
 - qm_scss_gp_reg_t, [312](#)
- grstctl
 - qm_usb_reg_t, [350](#)
- grxfsiz
 - qm_usb_reg_t, [350](#)
- grxstsp
 - qm_usb_reg_t, [350](#)
- grxstsr
 - qm_usb_reg_t, [350](#)
- gsnpsid
 - qm_usb_reg_t, [350](#)
- gusbcfg
 - qm_usb_reg_t, [350](#)
- hi_count
 - qm_pwm_config_t, [302](#)
- host_bus_error_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- htx
 - qm_uart_context_t, [340](#)
 - qm_uart_reg_t, [342](#)
- hw_fc
 - qm_uart_config_t, [339](#)
- I2C
 - QM_I2C_10_BIT, [48](#)
 - QM_I2C_7_BIT, [48](#)
 - QM_I2C_BUSY, [49](#)
 - QM_I2C_GEN_CALL_DETECTED, [49](#)
 - QM_I2C_IDLE, [49](#)
 - QM_I2C_MASTER, [48](#)
 - QM_I2C_RX_FULL, [49](#)
 - QM_I2C_RX_OVER, [49](#)
 - QM_I2C_RX_UNDER, [49](#)
 - QM_I2C_SLAVE, [48](#)
 - QM_I2C_SLAVE_INTERRUPT_ALWAYS, [49](#)
 - QM_I2C_SLAVE_INTERRUPT_WHEN_ADDRESSED, [49](#)
 - QM_I2C_SPEED_FAST, [49](#)
 - QM_I2C_SPEED_FAST_PLUS, [49](#)
 - QM_I2C_SPEED_STD, [49](#)
 - QM_I2C_START_DETECTED, [49](#)
 - QM_I2C_TX_ABORT, [49](#)
 - QM_I2C_TX_ABORT_10ADDR1_NOACK, [49](#)
 - QM_I2C_TX_ABORT_10ADDR2_NOACK, [49](#)
 - QM_I2C_TX_ABORT_10B_RD_NORSTRT, [49](#)
 - QM_I2C_TX_ABORT_7B_ADDR_NOACK, [49](#)
 - QM_I2C_TX_ABORT_GCALL_NOACK, [49](#)
 - QM_I2C_TX_ABORT_GCALL_READ, [49](#)
 - QM_I2C_TX_ABORT_HS_ACKDET, [49](#)
 - QM_I2C_TX_ABORT_HS_NORSTRT, [49](#)
 - QM_I2C_TX_ABORT_MASTER_DIS, [49](#)
 - QM_I2C_TX_ABORT_SBYTE_ACKDET, [49](#)
 - QM_I2C_TX_ABORT_SLV_ARBLOST, [49](#)
 - QM_I2C_TX_ABORT_SLVFLUSH_TXFIFO, [49](#)
 - QM_I2C_TX_ABORT_SLVRD_INTX, [49](#)
 - QM_I2C_TX_ABORT_TXDATA_NOACK, [49](#)
 - QM_I2C_TX_ABORT_USER_ABORT, [49](#)
 - QM_I2C_TX_ARB_LOST, [49](#)
 - QM_I2C_TX_EMPTY, [49](#)
 - QM_I2C_TX_OVER, [49](#)
- I2S
 - QM_I2S_12_BIT_SAMPLE_RESOLUTION, [60](#)
 - QM_I2S_13_BIT_SAMPLE_RESOLUTION, [60](#)
 - QM_I2S_14_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_15_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_16_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_17_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_18_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_19_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_20_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_21_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_22_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_23_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_24_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_25_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_26_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_27_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_28_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_29_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_30_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_31_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_32_BIT_SAMPLE_RESOLUTION, [61](#)
 - QM_I2S_AUDIO_FORMAT_DSP_MODE, [59](#)
 - QM_I2S_AUDIO_FORMAT_I2S_MODE, [59](#)
 - QM_I2S_AUDIO_FORMAT_LEFT_J, [59](#)
 - QM_I2S_AUDIO_FORMAT_RIGHT_J, [59](#)
 - QM_I2S_AUDIO_STREAM_RECEIVE, [60](#)
 - QM_I2S_AUDIO_STREAM_TRANSMIT, [60](#)
 - QM_I2S_DMA_COMPLETE, [61](#)
 - QM_I2S_DMA_ERROR, [61](#)
 - QM_I2S_FIFO_OVERFLOW, [61](#)
 - QM_I2S_FIFO_UNDERRUN, [61](#)
 - QM_I2S_INT_CLK, [60](#)
 - QM_I2S_MASTER, [60](#)
 - QM_I2S_MCLK, [60](#)
 - QM_I2S_RATE_11025KHZ, [59](#)
 - QM_I2S_RATE_16000KHZ, [59](#)
 - QM_I2S_RATE_22050KHZ, [59](#)
 - QM_I2S_RATE_32000KHZ, [60](#)

- QM_I2S_RATE_4000KHZ, [59](#)
- QM_I2S_RATE_44100KHZ, [60](#)
- QM_I2S_RATE_48000KHZ, [60](#)
- QM_I2S_RATE_8000KHZ, [59](#)
- QM_I2S_RING_BUFFER, [60](#)
- QM_I2S_RX_FIFO_ALMOST_EMPTY, [61](#)
- QM_I2S_RX_FIFO_ALMOST_FULL, [61](#)
- QM_I2S_RX_FIFO_EMPTY, [61](#)
- QM_I2S_RX_FIFO_FULL, [61](#)
- QM_I2S_SLAVE, [60](#)
- QM_I2S_TERMINATED_BUFFER, [60](#)
- QM_I2S_TX_FIFO_ALMOST_EMPTY, [61](#)
- QM_I2S_TX_FIFO_ALMOST_FULL, [61](#)
- QM_I2S_TX_FIFO_EMPTY, [61](#)
- QM_I2S_TX_FIFO_FULL, [61](#)
- I2C, [47](#)
 - qm_i2c_addr_t, [48](#)
 - qm_i2c_dma_channel_config, [50](#)
 - qm_i2c_dma_transfer_terminate, [50](#)
 - qm_i2c_get_status, [51](#)
 - qm_i2c_irq_transfer_terminate, [51](#)
 - qm_i2c_master_dma_transfer, [51](#)
 - qm_i2c_master_irq_transfer, [52](#)
 - qm_i2c_master_read, [52](#)
 - qm_i2c_master_write, [53](#)
 - qm_i2c_mode_t, [48](#)
 - qm_i2c_restore_context, [53](#)
 - qm_i2c_save_context, [54](#)
 - qm_i2c_set_config, [54](#)
 - qm_i2c_set_speed, [55](#)
 - qm_i2c_slave_dma_transfer, [55](#)
 - qm_i2c_slave_irq_transfer, [56](#)
 - qm_i2c_slave_irq_transfer_update, [56](#)
 - qm_i2c_slave_stop_t, [48](#)
 - qm_i2c_speed_t, [49](#)
 - qm_i2c_status_t, [49](#)
- I2S, [58](#)
 - qm_i2s_audio_format_t, [59](#)
 - qm_i2s_audio_rate_t, [59](#)
 - qm_i2s_audio_stream_t, [60](#)
 - qm_i2s_buffer_mode_t, [60](#)
 - qm_i2s_channel_disable, [61](#)
 - qm_i2s_clk_src_t, [60](#)
 - qm_i2s_dma_channel_config, [62](#)
 - qm_i2s_dma_transfer, [62](#)
 - qm_i2s_dma_transfer_terminate, [62](#)
 - qm_i2s_master_slave_t, [60](#)
 - qm_i2s_sample_resolution_t, [60](#)
 - qm_i2s_set_clock_config, [63](#)
 - qm_i2s_status_t, [61](#)
- i2c_master_0_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- i2c_master_1_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- i2s_0_int_mask
 - qm_interrupt_router_reg_t, [288](#)
- i2s_mclk_divisor
 - qm_i2s_clock_cfg_data_t, [284](#)
- i2s_mclk_output_en
 - qm_i2s_clock_cfg_data_t, [284](#)
- ISR, [68](#)
 - QM_ISR_DECLARE, [69–76](#)
- ic_ack_general_call
 - qm_i2c_reg_t, [274](#)
- ic_clr_activity
 - qm_i2c_reg_t, [274](#)
- ic_clr_gen_call
 - qm_i2c_reg_t, [274](#)
- ic_clr_intr
 - qm_i2c_reg_t, [274](#)
- ic_clr_rd_req
 - qm_i2c_reg_t, [274](#)
- ic_clr_restart_det
 - qm_i2c_reg_t, [275](#)
- ic_clr_rx_done
 - qm_i2c_reg_t, [275](#)
- ic_clr_rx_over
 - qm_i2c_reg_t, [275](#)
- ic_clr_rx_under
 - qm_i2c_reg_t, [275](#)
- ic_clr_start_det
 - qm_i2c_reg_t, [275](#)
- ic_clr_stop_det
 - qm_i2c_reg_t, [275](#)
- ic_clr_tx_abrt
 - qm_i2c_reg_t, [275](#)
- ic_clr_tx_over
 - qm_i2c_reg_t, [275](#)
- ic_comp_param_1
 - qm_i2c_reg_t, [275](#)
- ic_comp_type
 - qm_i2c_reg_t, [275](#)
- ic_comp_version
 - qm_i2c_reg_t, [275](#)
- ic_con
 - qm_i2c_reg_t, [276](#)
- ic_data_cmd
 - qm_i2c_reg_t, [276](#)
- ic_dma_cr
 - qm_i2c_reg_t, [276](#)
- ic_dma_rdlr
 - qm_i2c_reg_t, [276](#)
- ic_dma_tdlr
 - qm_i2c_reg_t, [276](#)
- ic_enable
 - qm_i2c_reg_t, [276](#)
- ic_enable_status
 - qm_i2c_reg_t, [276](#)
- ic_fs_scl_hcnt
 - qm_i2c_reg_t, [276](#)
- ic_fs_scl_lcnt
 - qm_i2c_reg_t, [276](#)
- ic_fs_spklen
 - qm_i2c_reg_t, [277](#)
- ic_hs_maddr
 - qm_i2c_reg_t, [277](#)

- ic_hs_scl_hcnt
 - qm_i2c_reg_t, 277
- ic_hs_scl_lcnt
 - qm_i2c_reg_t, 277
- ic_hs_spklen
 - qm_i2c_reg_t, 277
- ic_intr_mask
 - qm_i2c_context_t, 271
 - qm_i2c_reg_t, 277
- ic_intr_stat
 - qm_i2c_reg_t, 277
- ic_raw_intr_stat
 - qm_i2c_reg_t, 277
- ic_rx_tl
 - qm_i2c_reg_t, 277
- ic_rxflr
 - qm_i2c_reg_t, 277
- ic_sar
 - qm_i2c_reg_t, 278
- ic_sda_hold
 - qm_i2c_reg_t, 278
- ic_sda_setup
 - qm_i2c_reg_t, 278
- ic_ss_scl_hcnt
 - qm_i2c_reg_t, 278
- ic_ss_scl_lcnt
 - qm_i2c_reg_t, 278
- ic_status
 - qm_i2c_reg_t, 278
- ic_tar
 - qm_i2c_reg_t, 278
- ic_tx_abrt_source
 - qm_i2c_reg_t, 278
- ic_tx_tl
 - qm_i2c_reg_t, 278
- ic_txflr
 - qm_i2c_reg_t, 279
- icr
 - qm_mvic_reg_t, 297
 - qm_spi_reg_t, 322
- id
 - qm_scss_info_reg_t, 313
- Identification, 64
 - qm_soc_id, 64
 - qm_soc_version, 64
- idr
 - qm_spi_reg_t, 322
- ier
 - qm_uart_context_t, 340
- ier_dlh
 - qm_uart_reg_t, 342
- iir_fcr
 - qm_uart_reg_t, 342
- imr
 - qm_spi_reg_t, 322
- inc_run_only
 - qm_ss_timer_config_t, 338
- init_val
 - qm_rtc_config_t, 305
- Initialisation, 65
 - QM_COLD_RESET, 65
 - QM_WARM_RESET, 65
 - qm_soc_reset, 65
 - qm_soc_reset_t, 65
- int_bothedge
 - qm_gpio_port_config_t, 266
- int_debounce
 - qm_gpio_port_config_t, 266
 - qm_ss_gpio_port_config_t, 330
- int_en
 - qm_aonpt_config_t, 250
 - qm_gpio_port_config_t, 266
 - qm_pic_timer_config_t, 299
 - qm_ss_gpio_port_config_t, 330
 - qm_ss_timer_config_t, 338
- int_polarity
 - qm_gpio_port_config_t, 266
 - qm_ss_gpio_port_config_t, 330
- int_ss_i2c_reg_t, 244
- int_ss_spi_reg_t, 244
- int_type
 - qm_gpio_port_config_t, 266
 - qm_ss_gpio_port_config_t, 331
- Interrupt, 66
 - qm_int_vector_request, 66
 - qm_irq_lock, 66
 - qm_irq_mask, 66
 - qm_irq_unlock, 67
 - qm_irq_unmask, 67
- intstatus
 - qm_pwm_channel_t, 301
- irq_config
 - qm_irq_context_t, 291
- irq_ctrl
 - qm_irq_context_t, 291
- irr
 - qm_mvic_reg_t, 297
- isr
 - qm_mvic_reg_t, 297
 - qm_spi_reg_t, 322
- keep_enabled
 - qm_spi_async_transfer_t, 318
- lcr
 - qm_uart_context_t, 340
 - qm_uart_reg_t, 343
- lcr_ext
 - qm_uart_reg_t, 343
- line_control
 - qm_uart_config_t, 339
- llp_low
 - qm_dma_context_t, 253
- lo_count
 - qm_pwm_config_t, 302
- loadcount
 - qm_pwm_channel_t, 301

- qm_pwm_context_t, 303
- loadcount2
 - qm_pwm_context_t, 303
- lock_int_mask_reg
 - qm_interrupt_router_reg_t, 288
- low_bound
 - qm_fpr_config_t, 262
- lsr
 - qm_uart_reg_t, 343
- lvtimer
 - qm_mvic_reg_t, 298
 - qm_pic_timer_context_t, 299
- MPR, 82
 - qm_mpr_restore_context, 82
 - qm_mpr_save_context, 82
 - qm_mpr_set_config, 84
 - qm_mpr_set_violation_policy, 84
- Mailbox, 78
 - QM_MBOX_CH_DATA_PEND, 79
 - QM_MBOX_CH_IDLE, 79
 - QM_MBOX_CH_INT_AND_DATA_PEND, 79
 - QM_MBOX_INTERRUPT_MODE, 79
 - QM_MBOX_PAYLOAD_0, 79
 - QM_MBOX_PAYLOAD_1, 79
 - QM_MBOX_PAYLOAD_2, 79
 - QM_MBOX_PAYLOAD_3, 79
 - QM_MBOX_PAYLOAD_NUM, 79
 - QM_MBOX_POLLING_MODE, 79
 - qm_mbox_callback_t, 78
 - qm_mbox_ch_get_status, 79
 - qm_mbox_ch_read, 80
 - qm_mbox_ch_set_config, 80
 - qm_mbox_ch_status_t, 79
 - qm_mbox_ch_write, 81
 - qm_mbox_mode_t, 79
 - qm_mbox_payload_t, 79
- Mailbox Definitions, 220
 - QM_MBOX_CH_0, 220
 - QM_MBOX_CH_1, 220
 - QM_MBOX_CH_2, 220
 - QM_MBOX_CH_3, 220
 - QM_MBOX_CH_4, 220
 - QM_MBOX_CH_5, 220
 - QM_MBOX_CH_6, 220
 - QM_MBOX_CH_7, 220
 - QM_MBOX_TO_LMT, 220
 - QM_MBOX_TO_SS, 220
 - qm_mbox_ch_t, 220
 - qm_mbox_destination_t, 220
- mailbox_0_int_mask
 - qm_interrupt_router_reg_t, 288
- mask_interrupt
 - qm_pwm_config_t, 302
- max_packet_size
 - qm_usb_ep_config_t, 346
- mcr
 - qm_uart_context_t, 341
 - qm_uart_reg_t, 343
- misc_cfg_low
 - qm_dma_context_t, 253
- mode
 - qm_i2c_config_t, 270
 - qm_mbox_config_t, 294
 - qm_pic_timer_config_t, 299
 - qm_pwm_config_t, 303
 - qm_wdt_config_t, 351
- mpr_cfg
 - qm_mpr_context_t, 296
- mpr_vsts
 - qm_flash_reg_t, 261
- mpr_wr_cfg
 - qm_flash_reg_t, 261
- msr
 - qm_uart_reg_t, 343
- msticr
 - qm_spi_reg_t, 322
- mvic_reg_pad_t, 244
- mwcr
 - qm_spi_reg_t, 322
- next
 - qm_i2s_buffer_link, 281
- num_blocks
 - qm_dma_multi_transfer_t, 257
- num_buffer_links
 - qm_i2s_channel_cfg_data_t, 283
- num_dma_link_per_buffer
 - qm_i2s_channel_cfg_data_t, 283
- num_dma_links
 - qm_i2s_channel_cfg_data_t, 283
- osc0_cfg0
 - qm_scss_ccu_reg_t, 308
- osc0_cfg1
 - qm_scss_ccu_reg_t, 308
- osc0_stat1
 - qm_scss_ccu_reg_t, 309
- osc1_cfg0
 - qm_scss_ccu_reg_t, 309
- osc1_stat0
 - qm_scss_ccu_reg_t, 309
- osc_lock_0
 - qm_scss_ccu_reg_t, 309
- PIC Timer
 - QM_PIC_TIMER_MODE_ONE_SHOT, 85
 - QM_PIC_TIMER_MODE_PERIODIC, 85
- PWM / Timer
 - QM_PWM_MODE_TIMER_COUNT, 94
 - QM_PWM_MODE_TIMER_FREE_RUNNING, 94
- p_sts
 - qm_scss_pmu_reg_t, 315
- PIC Timer, 85
 - qm_pic_timer_get, 85
 - qm_pic_timer_mode_t, 85
 - qm_pic_timer_restore_context, 86
 - qm_pic_timer_save_context, 86

- qm_pic_timer_set, 86
- qm_pic_timer_set_config, 88
- PWM / Timer, 94
 - qm_pwm_get, 95
 - qm_pwm_mode_t, 94
 - qm_pwm_restore_context, 96
 - qm_pwm_save_context, 96
 - qm_pwm_set, 97
 - qm_pwm_set_config, 97
 - qm_pwm_start, 98
 - qm_pwm_stop, 98
- pause_en
 - qm_wdt_config_t, 351
- periph_cfg0
 - qm_scss_peripheral_reg_t, 314
- pic_timer_reg_pad_t, 244
- pico_printf
 - Syscalls, 243
- Pin Muxing setup, 89
 - QM_PIN_ID_0, 90
 - QM_PIN_ID_1, 90
 - QM_PIN_ID_10, 90
 - QM_PIN_ID_11, 90
 - QM_PIN_ID_12, 90
 - QM_PIN_ID_13, 90
 - QM_PIN_ID_14, 90
 - QM_PIN_ID_15, 90
 - QM_PIN_ID_16, 90
 - QM_PIN_ID_17, 90
 - QM_PIN_ID_18, 90
 - QM_PIN_ID_19, 90
 - QM_PIN_ID_2, 90
 - QM_PIN_ID_20, 90
 - QM_PIN_ID_21, 90
 - QM_PIN_ID_22, 90
 - QM_PIN_ID_23, 90
 - QM_PIN_ID_24, 90
 - QM_PIN_ID_25, 90
 - QM_PIN_ID_26, 90
 - QM_PIN_ID_27, 90
 - QM_PIN_ID_28, 90
 - QM_PIN_ID_29, 90
 - QM_PIN_ID_3, 90
 - QM_PIN_ID_30, 90
 - QM_PIN_ID_31, 90
 - QM_PIN_ID_32, 90
 - QM_PIN_ID_33, 90
 - QM_PIN_ID_34, 90
 - QM_PIN_ID_35, 90
 - QM_PIN_ID_36, 90
 - QM_PIN_ID_37, 90
 - QM_PIN_ID_38, 90
 - QM_PIN_ID_39, 90
 - QM_PIN_ID_4, 90
 - QM_PIN_ID_40, 90
 - QM_PIN_ID_41, 90
 - QM_PIN_ID_42, 90
 - QM_PIN_ID_43, 90
 - QM_PIN_ID_44, 90
 - QM_PIN_ID_45, 91
 - QM_PIN_ID_46, 91
 - QM_PIN_ID_47, 91
 - QM_PIN_ID_48, 91
 - QM_PIN_ID_49, 91
 - QM_PIN_ID_5, 90
 - QM_PIN_ID_50, 91
 - QM_PIN_ID_51, 91
 - QM_PIN_ID_52, 91
 - QM_PIN_ID_53, 91
 - QM_PIN_ID_54, 91
 - QM_PIN_ID_55, 91
 - QM_PIN_ID_56, 91
 - QM_PIN_ID_57, 91
 - QM_PIN_ID_58, 91
 - QM_PIN_ID_59, 91
 - QM_PIN_ID_6, 90
 - QM_PIN_ID_60, 91
 - QM_PIN_ID_61, 91
 - QM_PIN_ID_62, 91
 - QM_PIN_ID_63, 91
 - QM_PIN_ID_64, 91
 - QM_PIN_ID_65, 91
 - QM_PIN_ID_66, 91
 - QM_PIN_ID_67, 91
 - QM_PIN_ID_68, 91
 - QM_PIN_ID_7, 90
 - QM_PIN_ID_8, 90
 - QM_PIN_ID_9, 90
 - QM_PMUX_FN_0, 91
 - QM_PMUX_FN_1, 91
 - QM_PMUX_FN_2, 91
 - QM_PMUX_FN_3, 91
 - QM_PMUX_SLEW_12MA, 91
 - QM_PMUX_SLEW_16MA, 91
 - QM_PMUX_SLEW_2MA, 91
 - QM_PMUX_SLEW_4MA, 91
 - QM_PMUX_SLEW_NUM, 91
 - qm_pin_id_t, 89
 - qm_pmux_fn_t, 91
 - qm_pmux_input_en, 92
 - qm_pmux_pullup_en, 92
 - qm_pmux_select, 92
 - qm_pmux_set_slew, 93
 - qm_pmux_slew_t, 91
- pm_lock
 - qm_scss_pmu_reg_t, 315
- pm_wait
 - qm_scss_pmu_reg_t, 315
- pmux_in_en
 - qm_scss_pmux_reg_t, 316
- pmux_in_en_lock
 - qm_scss_pmux_reg_t, 316
- pmux_pullup
 - qm_scss_pmux_reg_t, 316
- pmux_pullup_lock
 - qm_scss_pmux_reg_t, 316

pmux_sel
 qm_scss_pmux_reg_t, [316](#)
 pmux_sel_0_lock
 qm_scss_pmux_reg_t, [316](#)
 pmux_slew
 qm_scss_pmux_reg_t, [317](#)
 pmux_slew_lock
 qm_scss_pmux_reg_t, [317](#)
 ppr
 qm_mvic_reg_t, [298](#)
 prescaler
 qm_rtc_config_t, [305](#)
 pwm_0_int_mask
 qm_interrupt_router_reg_t, [289](#)

QM_ADC_CAL_COMPLETE
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_0
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_1
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_10
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_11
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_12
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_13
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_14
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_15
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_16
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_17
 Quark D2000 ADC, [11](#)
 QM_ADC_CH_18
 Quark D2000 ADC, [11](#)
 QM_ADC_CH_2
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_3
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_4
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_5
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_6
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_7
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_8
 Quark D2000 ADC, [10](#)
 QM_ADC_CH_9
 Quark D2000 ADC, [10](#)
 QM_ADC_COMPLETE
 Quark D2000 ADC, [11](#)
 QM_ADC_IDLE
 Quark D2000 ADC, [11](#)

QM_ADC_MODE_CHANGED
 Quark D2000 ADC, [10](#)
 QM_ADC_MODE_DEEP_PWR_DOWN
 Quark D2000 ADC, [11](#)
 QM_ADC_MODE_NORM_CAL
 Quark D2000 ADC, [11](#)
 QM_ADC_MODE_NORM_NO_CAL
 Quark D2000 ADC, [11](#)
 QM_ADC_MODE_PWR_DOWN
 Quark D2000 ADC, [11](#)
 QM_ADC_MODE_STDBY
 Quark D2000 ADC, [11](#)
 QM_ADC_OVERFLOW
 Quark D2000 ADC, [11](#)
 QM_ADC_RES_10_BITS
 Quark D2000 ADC, [11](#)
 QM_ADC_RES_12_BITS
 Quark D2000 ADC, [11](#)
 QM_ADC_RES_6_BITS
 Quark D2000 ADC, [11](#)
 QM_ADC_RES_8_BITS
 Quark D2000 ADC, [11](#)
 QM_ADC_TRANSFER
 Quark D2000 ADC, [10](#)
 QM_AONPT_EXPIRED
 Always-on Counters, [17](#)
 QM_AONPT_READY
 Always-on Counters, [17](#)
 QM_COLD_RESET
 Initialisation, [65](#)
 QM_DMA_0
 SoC Registers (D2000), [211](#)
 SoC Registers (SE), [233](#)
 QM_DMA_BURST_TRANS_LENGTH_1
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_128
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_16
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_256
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_32
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_4
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_64
 DMA, [23](#)
 QM_DMA_BURST_TRANS_LENGTH_8
 DMA, [23](#)
 QM_DMA_CHANNEL_0
 SoC Registers (D2000), [210](#)
 SoC Registers (SE), [232](#)
 QM_DMA_CHANNEL_1
 SoC Registers (D2000), [210](#)
 SoC Registers (SE), [232](#)
 QM_DMA_CHANNEL_2
 SoC Registers (D2000), [210](#)
 SoC Registers (SE), [232](#)

- QM_DMA_CHANNEL_3
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- QM_DMA_CHANNEL_4
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- QM_DMA_CHANNEL_5
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- QM_DMA_CHANNEL_6
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- QM_DMA_CHANNEL_7
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- QM_DMA_CHANNEL_NUM
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- QM_DMA_HANDSHAKE_POLARITY_HIGH
 - DMA, [23](#)
- QM_DMA_HANDSHAKE_POLARITY_LOW
 - DMA, [23](#)
- QM_DMA_MEMORY_TO_MEMORY
 - DMA, [23](#)
- QM_DMA_MEMORY_TO_PERIPHERAL
 - DMA, [23](#)
- QM_DMA_NUM
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [233](#)
- QM_DMA_PERIPHERAL_TO_MEMORY
 - DMA, [23](#)
- QM_DMA_TRANS_WIDTH_128
 - DMA, [24](#)
- QM_DMA_TRANS_WIDTH_16
 - DMA, [24](#)
- QM_DMA_TRANS_WIDTH_256
 - DMA, [24](#)
- QM_DMA_TRANS_WIDTH_32
 - DMA, [24](#)
- QM_DMA_TRANS_WIDTH_64
 - DMA, [24](#)
- QM_DMA_TRANS_WIDTH_8
 - DMA, [24](#)
- QM_DMA_TYPE_MULTI_CONT
 - DMA, [24](#)
- QM_DMA_TYPE_MULTI_LL
 - DMA, [24](#)
- QM_DMA_TYPE_MULTI_LL_CIRCULAR
 - DMA, [24](#)
- QM_DMA_TYPE_SINGLE
 - DMA, [24](#)
- QM_FLASH_REGION_DATA
 - Flash, [31](#)
- QM_FLASH_REGION_NUM
 - Flash, [31](#)
- QM_FLASH_REGION_OTP
 - Flash, [31](#)
- QM_FLASH_REGION_SYS
 - Flash, [31](#)
- QM_FLASH_WRITE_DISABLE
 - Flash, [30](#)
- QM_FLASH_WRITE_ENABLE
 - Flash, [30](#)
- QM_FPR_0
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- QM_FPR_1
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- QM_FPR_2
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- QM_FPR_3
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- QM_FPR_DISABLE
 - FPR, [38](#)
- QM_FPR_DMA
 - FPR, [38](#)
- QM_FPR_ENABLE
 - FPR, [38](#)
- QM_FPR_HOST_PROCESSOR
 - FPR, [38](#)
- QM_FPR_LOCK_DISABLE
 - FPR, [38](#)
- QM_FPR_LOCK_ENABLE
 - FPR, [38](#)
- QM_FPR_OTHER_AGENTS
 - FPR, [38](#)
- QM_FPR_SENSOR_SUBSYSTEM
 - FPR, [38](#)
- QM_GPIO_HIGH
 - GPIO, [42](#)
- QM_GPIO_LOW
 - GPIO, [42](#)
- QM_GPIO_STATE_NUM
 - GPIO, [42](#)
- QM_I2C_10_BIT
 - I2C, [48](#)
- QM_I2C_7_BIT
 - I2C, [48](#)
- QM_I2C_BUSY
 - I2C, [49](#)
- QM_I2C_GEN_CALL_DETECTED
 - I2C, [49](#)
- QM_I2C_IDLE
 - I2C, [49](#)
- QM_I2C_MASTER
 - I2C, [48](#)
- QM_I2C_RX_FULL
 - I2C, [49](#)
- QM_I2C_RX_OVER
 - I2C, [49](#)
- QM_I2C_RX_UNDER
 - I2C, [49](#)
- QM_I2C_SLAVE

I2C, 48	I2S, 61
QM_I2C_SLAVE_INTERRUPT_ALWAYS	QM_I2S_16_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_SLAVE_INTERRUPT_WHEN_ADDRESSED	QM_I2S_17_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_SPEED_FAST	QM_I2S_18_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_SPEED_FAST_PLUS	QM_I2S_19_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_SPEED_STD	QM_I2S_20_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_START_DETECTED	QM_I2S_21_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABORT	QM_I2S_22_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_10ADDR1_NOACK	QM_I2S_23_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_10ADDR2_NOACK	QM_I2S_24_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_10B_RD_NORSTR	QM_I2S_25_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_7B_ADDR_NOACK	QM_I2S_26_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_GCALL_NOACK	QM_I2S_27_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_GCALL_READ	QM_I2S_28_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_HS_ACKDET	QM_I2S_29_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_HS_NORSTR	QM_I2S_30_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_MASTER_DIS	QM_I2S_31_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_SBYTE_ACKDET	QM_I2S_32_BIT_SAMPLE_RESOLUTION
I2C, 49	I2S, 61
QM_I2C_TX_ABRT_SLV_ARBLOST	QM_I2S_AUDIO_FORMAT_DSP_MODE
I2C, 49	I2S, 59
QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO	QM_I2S_AUDIO_FORMAT_I2S_MODE
I2C, 49	I2S, 59
QM_I2C_TX_ABRT_SLVRD_INTX	QM_I2S_AUDIO_FORMAT_LEFT_J
I2C, 49	I2S, 59
QM_I2C_TX_ABRT_TXDATA_NOACK	QM_I2S_AUDIO_FORMAT_RIGHT_J
I2C, 49	I2S, 59
QM_I2C_TX_ABRT_USER_ABRT	QM_I2S_AUDIO_STREAM_RECEIVE
I2C, 49	I2S, 60
QM_I2C_TX_ARB_LOST	QM_I2S_AUDIO_STREAM_TRANSMIT
I2C, 49	I2S, 60
QM_I2C_TX_EMPTY	QM_I2S_DMA_COMPLETE
I2C, 49	I2S, 61
QM_I2C_TX_OVER	QM_I2S_DMA_ERROR
I2C, 49	I2S, 61
QM_I2S_12_BIT_SAMPLE_RESOLUTION	QM_I2S_FIFO_OVERFLOW
I2S, 60	I2S, 61
QM_I2S_13_BIT_SAMPLE_RESOLUTION	QM_I2S_FIFO_UNDERRUN
I2S, 60	I2S, 61
QM_I2S_14_BIT_SAMPLE_RESOLUTION	QM_I2S_INT_CLK
I2S, 61	I2S, 60
QM_I2S_15_BIT_SAMPLE_RESOLUTION	QM_I2S_MASTER

- I2S, [60](#)
- QM_I2S_MCLK
 - I2S, [60](#)
- QM_I2S_RATE_11025KHZ
 - I2S, [59](#)
- QM_I2S_RATE_16000KHZ
 - I2S, [59](#)
- QM_I2S_RATE_22050KHZ
 - I2S, [59](#)
- QM_I2S_RATE_32000KHZ
 - I2S, [60](#)
- QM_I2S_RATE_4000KHZ
 - I2S, [59](#)
- QM_I2S_RATE_44100KHZ
 - I2S, [60](#)
- QM_I2S_RATE_48000KHZ
 - I2S, [60](#)
- QM_I2S_RATE_8000KHZ
 - I2S, [59](#)
- QM_I2S_RING_BUFFER
 - I2S, [60](#)
- QM_I2S_RX_FIFO_ALMOST_EMPTY
 - I2S, [61](#)
- QM_I2S_RX_FIFO_ALMOST_FULL
 - I2S, [61](#)
- QM_I2S_RX_FIFO_EMPTY
 - I2S, [61](#)
- QM_I2S_RX_FIFO_FULL
 - I2S, [61](#)
- QM_I2S_SLAVE
 - I2S, [60](#)
- QM_I2S_TERMINATED_BUFFER
 - I2S, [60](#)
- QM_I2S_TX_FIFO_ALMOST_EMPTY
 - I2S, [61](#)
- QM_I2S_TX_FIFO_ALMOST_FULL
 - I2S, [61](#)
- QM_I2S_TX_FIFO_EMPTY
 - I2S, [61](#)
- QM_I2S_TX_FIFO_FULL
 - I2S, [61](#)
- QM_MAIN_FLASH_DATA
 - FPR, [37](#)
- QM_MAIN_FLASH_NUM
 - FPR, [37](#)
- QM_MAIN_FLASH_SYSTEM
 - FPR, [37](#)
- QM_MBOX_CH_0
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_1
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_2
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_3
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_4
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_5
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_6
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_7
 - Mailbox Definitions, [220](#)
- QM_MBOX_CH_DATA_PEND
 - Mailbox, [79](#)
- QM_MBOX_CH_IDLE
 - Mailbox, [79](#)
- QM_MBOX_CH_INT_AND_DATA_PEND
 - Mailbox, [79](#)
- QM_MBOX_INTERRUPT_MODE
 - Mailbox, [79](#)
- QM_MBOX_PAYLOAD_0
 - Mailbox, [79](#)
- QM_MBOX_PAYLOAD_1
 - Mailbox, [79](#)
- QM_MBOX_PAYLOAD_2
 - Mailbox, [79](#)
- QM_MBOX_PAYLOAD_3
 - Mailbox, [79](#)
- QM_MBOX_PAYLOAD_NUM
 - Mailbox, [79](#)
- QM_MBOX_POLLING_MODE
 - Mailbox, [79](#)
- QM_MBOX_TO_LMT
 - Mailbox Definitions, [220](#)
- QM_MBOX_TO_SS
 - Mailbox Definitions, [220](#)
- QM_MPR_0
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- QM_MPR_1
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- QM_MPR_2
 - SoC Registers (D2000), [211](#), [212](#)
 - SoC Registers (SE), [234](#)
- QM_MPR_3
 - SoC Registers (D2000), [211](#), [212](#)
 - SoC Registers (SE), [234](#)
- QM_MPR_NUM
 - SoC Registers (D2000), [211](#), [212](#)
 - SoC Registers (SE), [234](#)
- QM_PIC_TIMER_MODE_ONE_SHOT
 - PIC Timer, [85](#)
- QM_PIC_TIMER_MODE_PERIODIC
 - PIC Timer, [85](#)
- QM_PIN_ID_0
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_1
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_10
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_11
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_12
 - Pin Muxing setup, [90](#)

- QM_PIN_ID_13
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_14
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_15
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_16
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_17
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_18
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_19
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_2
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_20
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_21
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_22
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_23
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_24
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_25
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_26
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_27
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_28
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_29
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_3
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_30
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_31
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_32
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_33
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_34
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_35
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_36
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_37
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_38
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_39
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_4
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_40
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_41
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_42
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_43
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_44
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_45
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_46
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_47
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_48
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_49
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_5
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_50
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_51
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_52
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_53
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_54
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_55
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_56
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_57
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_58
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_59
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_6
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_60
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_61
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_62
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_63
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_64
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_65
 - Pin Muxing setup, [91](#)

- QM_PIN_ID_66
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_67
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_68
 - Pin Muxing setup, [91](#)
- QM_PIN_ID_7
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_8
 - Pin Muxing setup, [90](#)
- QM_PIN_ID_9
 - Pin Muxing setup, [90](#)
- QM_PMUX_FN_0
 - Pin Muxing setup, [91](#)
- QM_PMUX_FN_1
 - Pin Muxing setup, [91](#)
- QM_PMUX_FN_2
 - Pin Muxing setup, [91](#)
- QM_PMUX_FN_3
 - Pin Muxing setup, [91](#)
- QM_PMUX_SLEW_12MA
 - Pin Muxing setup, [91](#)
- QM_PMUX_SLEW_16MA
 - Pin Muxing setup, [91](#)
- QM_PMUX_SLEW_2MA
 - Pin Muxing setup, [91](#)
- QM_PMUX_SLEW_4MA
 - Pin Muxing setup, [91](#)
- QM_PMUX_SLEW_NUM
 - Pin Muxing setup, [91](#)
- QM_POWER_WAKE_FROM_GPIO_COMP
 - Quark D2000 Power states, [196](#)
- QM_POWER_WAKE_FROM_RTC
 - Quark D2000 Power states, [196](#)
- QM_PWM_MODE_TIMER_COUNT
 - PWM / Timer, [94](#)
- QM_PWM_MODE_TIMER_FREE_RUNNING
 - PWM / Timer, [94](#)
- QM_RAR_NORMAL
 - Quark(TM) D2000 Retention Alternator Regulator (RAR)., [199](#)
- QM_RAR_RETENTION
 - Quark(TM) D2000 Retention Alternator Regulator (RAR)., [199](#)
- QM_SPI_BMODE_0
 - SPI, [103](#)
- QM_SPI_BMODE_1
 - SPI, [103](#)
- QM_SPI_BMODE_2
 - SPI, [103](#)
- QM_SPI_BMODE_3
 - SPI, [103](#)
- QM_SPI_BUSY
 - SPI, [105](#)
- QM_SPI_FRAME_FORMAT_STANDARD
 - SPI, [103](#)
- QM_SPI_FRAME_SIZE_10_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_11_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_12_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_13_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_14_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_15_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_16_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_17_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_18_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_19_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_20_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_21_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_22_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_23_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_24_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_25_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_26_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_27_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_28_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_29_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_30_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_31_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_32_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_4_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_5_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_6_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_7_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_8_BIT
 - SPI, [104](#)
- QM_SPI_FRAME_SIZE_9_BIT
 - SPI, [104](#)
- QM_SPI_IDLE
 - SPI, [105](#)

QM_SPI_RX_FULL
 SPI, [105](#)
 QM_SPI_RX_OVERFLOW
 SPI, [105](#)
 QM_SPI_SS_0
 SPI, [104](#)
 QM_SPI_SS_1
 SPI, [104](#)
 QM_SPI_SS_2
 SPI, [104](#)
 QM_SPI_SS_3
 SPI, [104](#)
 QM_SPI_SS_DISABLED
 SPI, [104](#)
 QM_SPI_TMOD_EEPROM_READ
 SPI, [105](#)
 QM_SPI_TMOD_RX
 SPI, [105](#)
 QM_SPI_TMOD_TX
 SPI, [105](#)
 QM_SPI_TMOD_TX_RX
 SPI, [105](#)
 QM_SPI_TX_EMPTY
 SPI, [105](#)
 QM_SS_ADC_0
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_CAL_COMPLETE
 SS ADC, [113](#)
 QM_SS_ADC_CH_0
 SS ADC, [113](#)
 QM_SS_ADC_CH_1
 SS ADC, [113](#)
 QM_SS_ADC_CH_10
 SS ADC, [114](#)
 QM_SS_ADC_CH_11
 SS ADC, [114](#)
 QM_SS_ADC_CH_12
 SS ADC, [114](#)
 QM_SS_ADC_CH_13
 SS ADC, [114](#)
 QM_SS_ADC_CH_14
 SS ADC, [114](#)
 QM_SS_ADC_CH_15
 SS ADC, [114](#)
 QM_SS_ADC_CH_16
 SS ADC, [114](#)
 QM_SS_ADC_CH_17
 SS ADC, [114](#)
 QM_SS_ADC_CH_18
 SS ADC, [114](#)
 QM_SS_ADC_CH_2
 SS ADC, [113](#)
 QM_SS_ADC_CH_3
 SS ADC, [113](#)
 QM_SS_ADC_CH_4
 SS ADC, [113](#)
 QM_SS_ADC_CH_5
 SS ADC, [113](#)
 QM_SS_ADC_CH_6
 SS ADC, [113](#)
 QM_SS_ADC_CH_7
 SS ADC, [113](#)
 QM_SS_ADC_CH_8
 SS ADC, [113](#)
 QM_SS_ADC_CH_9
 SS ADC, [114](#)
 QM_SS_ADC_COMPLETE
 SS ADC, [114](#)
 QM_SS_ADC_CTRL
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_DIVSEQSTAT
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_IDLE
 SS ADC, [114](#)
 QM_SS_ADC_INTSTAT
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_MODE_CHANGED
 SS ADC, [113](#)
 QM_SS_ADC_MODE_DEEP_PWR_DOWN
 SS ADC, [114](#)
 QM_SS_ADC_MODE_NORM_CAL
 SS ADC, [114](#)
 QM_SS_ADC_MODE_NORM_NO_CAL
 SS ADC, [114](#)
 QM_SS_ADC_MODE_PWR_DOWN
 SS ADC, [114](#)
 QM_SS_ADC_MODE_STDBY
 SS ADC, [114](#)
 QM_SS_ADC_OVERFLOW
 SS ADC, [114](#)
 QM_SS_ADC_RES_10_BITS
 SS ADC, [114](#)
 QM_SS_ADC_RES_12_BITS
 SS ADC, [114](#)
 QM_SS_ADC_RES_6_BITS
 SS ADC, [114](#)
 QM_SS_ADC_RES_8_BITS
 SS ADC, [114](#)
 QM_SS_ADC_SAMPLE
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_SEQ
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_SEQERROR
 SS ADC, [114](#)
 QM_SS_ADC_SET
 SoC Registers (Sensor Subsystem), [222](#)
 QM_SS_ADC_TRANSFER
 SS ADC, [113](#)
 QM_SS_ADC_UNDERFLOW
 SS ADC, [114](#)
 QM_SS_GPIO_HIGH
 SS GPIO, [121](#)
 QM_SS_GPIO_LOW
 SS GPIO, [121](#)
 QM_SS_GPIO_STATE_NUM
 SS GPIO, [121](#)

- QM_SS_I2C_10_BIT
 - SS I2C, [127](#)
- QM_SS_I2C_7_BIT
 - SS I2C, [127](#)
- QM_SS_I2C_BUSY
 - SS I2C, [127](#)
- QM_SS_I2C_IDLE
 - SS I2C, [127](#)
- QM_SS_I2C_RX_OVER
 - SS I2C, [128](#)
- QM_SS_I2C_RX_UNDER
 - SS I2C, [128](#)
- QM_SS_I2C_SPEED_FAST
 - SS I2C, [127](#)
- QM_SS_I2C_SPEED_FAST_PLUS
 - SS I2C, [127](#)
- QM_SS_I2C_SPEED_STD
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABORT
 - SS I2C, [128](#)
- QM_SS_I2C_TX_ABRT_10ADDR1_NOACK
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_10ADDR2_NOACK
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_10B_RD_NORSTRT
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_7B_ADDR_NOACK
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_GCALL_NOACK
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_MASTER_DIS
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_NORSTRT
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_SBYTE_ACKDET
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_SLV_ARBLOST
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_SLVRD_INTX
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_TXDATA_NOACK
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ABRT_USER_ABRT
 - SS I2C, [127](#)
- QM_SS_I2C_TX_ARB_LOST
 - SS I2C, [127](#)
- QM_SS_I2C_TX_OVER
 - SS I2C, [128](#)
- QM_SS_IO_CREG_MST0_CTRL
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_IO_CREG_SLV0_OBSR
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_IO_CREG_SLV1_OBSR
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_POWER_CPU_SS1_TIMER_OFF
 - SS Power states, [237](#)
- QM_SS_POWER_CPU_SS1_TIMER_ON
 - SS Power states, [237](#)
- QM_SS_SPI_0
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_1
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_BMODE_0
 - SS SPI, [141](#)
- QM_SS_SPI_BMODE_1
 - SS SPI, [141](#)
- QM_SS_SPI_BMODE_2
 - SS SPI, [141](#)
- QM_SS_SPI_BMODE_3
 - SS SPI, [141](#)
- QM_SS_SPI_BUSY
 - SS SPI, [142](#)
- QM_SS_SPI_CLR_INTR
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_CTRL
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_DR
 - SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_FRAME_SIZE_10_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_11_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_12_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_13_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_14_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_15_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_16_BIT
 - SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_17_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_18_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_19_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_20_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_21_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_22_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_23_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_24_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_25_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_26_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_27_BIT
 - SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_28_BIT
 - SS SPI, [142](#)

- QM_SS_SPI_FRAME_SIZE_29_BIT
SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_30_BIT
SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_31_BIT
SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_32_BIT
SS SPI, [142](#)
- QM_SS_SPI_FRAME_SIZE_4_BIT
SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_5_BIT
SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_6_BIT
SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_7_BIT
SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_8_BIT
SS SPI, [141](#)
- QM_SS_SPI_FRAME_SIZE_9_BIT
SS SPI, [141](#)
- QM_SS_SPI_FTLR
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_IDLE
SS SPI, [142](#)
- QM_SS_SPI_INTR_MASK
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_INTR_STAT
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_RX_OVERFLOW
SS SPI, [142](#)
- QM_SS_SPI_RX_UNDERFLOW
SS SPI, [142](#)
- QM_SS_SPI_RXFLR
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_SPIEN
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_SR
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_SS_0
SS SPI, [142](#)
- QM_SS_SPI_SS_1
SS SPI, [142](#)
- QM_SS_SPI_SS_2
SS SPI, [142](#)
- QM_SS_SPI_SS_3
SS SPI, [142](#)
- QM_SS_SPI_SS_DISABLED
SS SPI, [142](#)
- QM_SS_SPI_TIMING
SoC Registers (Sensor Subsystem), [223](#)
- QM_SS_SPI_TMOD_EEPROM_READ
SS SPI, [143](#)
- QM_SS_SPI_TMOD_RX
SS SPI, [143](#)
- QM_SS_SPI_TMOD_TX
SS SPI, [143](#)
- QM_SS_SPI_TMOD_TX_RX
SS SPI, [143](#)
- QM_SS_SPI_TX_OVERFLOW
SS SPI, [142](#)
- QM_SS_SPI_TXFLR
SoC Registers (Sensor Subsystem), [223](#)
- QM_UART_IDLE
UART, [151](#)
- QM_UART_LC_5E1
UART, [150](#)
- QM_UART_LC_5E1_5
UART, [150](#)
- QM_UART_LC_5N1
UART, [150](#)
- QM_UART_LC_5N1_5
UART, [150](#)
- QM_UART_LC_5O1
UART, [150](#)
- QM_UART_LC_5O1_5
UART, [150](#)
- QM_UART_LC_6E1
UART, [150](#)
- QM_UART_LC_6E2
UART, [150](#)
- QM_UART_LC_6N1
UART, [150](#)
- QM_UART_LC_6N2
UART, [150](#)
- QM_UART_LC_6O1
UART, [150](#)
- QM_UART_LC_6O2
UART, [150](#)
- QM_UART_LC_7E1
UART, [150](#)
- QM_UART_LC_7E2
UART, [150](#)
- QM_UART_LC_7N1
UART, [150](#)
- QM_UART_LC_7N2
UART, [150](#)
- QM_UART_LC_7O1
UART, [150](#)
- QM_UART_LC_7O2
UART, [151](#)
- QM_UART_LC_8E1
UART, [151](#)
- QM_UART_LC_8E2
UART, [151](#)
- QM_UART_LC_8N1
UART, [151](#)
- QM_UART_LC_8N2
UART, [151](#)
- QM_UART_LC_8O1
UART, [151](#)
- QM_UART_LC_8O2
UART, [151](#)
- QM_UART_RX_BI
UART, [151](#)
- QM_UART_RX_BUSY
UART, [151](#)

- QM_UART_RX_FE
 - UART, [151](#)
- QM_UART_RX_EMPTY
 - UART, [151](#)
- QM_UART_RX_OE
 - UART, [151](#)
- QM_UART_RX_PE
 - UART, [151](#)
- QM_UART_TX_BUSY
 - UART, [151](#)
- QM_UART_TX_NFULL
 - UART, [151](#)
- QM_USB_CONFIGURED
 - USB, [162](#)
- QM_USB_CONNECTED
 - USB, [162](#)
- QM_USB_DISCONNECTED
 - USB, [162](#)
- QM_USB_EP_BULK
 - USB, [161](#)
- QM_USB_EP_CONTROL
 - USB, [161](#)
- QM_USB_EP_DATA_IN
 - USB, [161](#)
- QM_USB_EP_DATA_OUT
 - USB, [161](#)
- QM_USB_EP_INTERRUPT
 - USB, [161](#)
- QM_USB_EP_SETUP
 - USB, [161](#)
- QM_USB_RESET
 - USB, [162](#)
- QM_USB_RESUME
 - USB, [162](#)
- QM_USB_SUSPEND
 - USB, [162](#)
- QM_WARM_RESET
 - Initialisation, [65](#)
- QM_WDT_MODE_INTERRUPT_RESET
 - WDT, [169](#)
- QM_WDT_MODE_RESET
 - WDT, [169](#)
- QM_ISR_DECLARE
 - ISR, [69–76](#)
 - SS ISR, [135–139](#)
- qm_ac_config_t, [244](#)
 - callback, [245](#)
 - callback_data, [245](#)
 - cmp_en, [245](#)
 - reference, [245](#)
- qm_ac_set_config
 - Analog Comparator, [21](#)
- qm_adc_calibrate
 - Quark D2000 ADC, [11](#)
- qm_adc_cb_source_t
 - Quark D2000 ADC, [10](#)
- qm_adc_channel_t
 - Quark D2000 ADC, [10](#)
- qm_adc_config_t, [246](#)
 - resolution, [246](#)
- qm_adc_convert
 - Quark D2000 ADC, [12](#)
- qm_adc_get_calibration
 - Quark D2000 ADC, [12](#)
- qm_adc_irq_calibrate
 - Quark D2000 ADC, [12](#)
- qm_adc_irq_convert
 - Quark D2000 ADC, [13](#)
- qm_adc_irq_set_mode
 - Quark D2000 ADC, [13](#)
- qm_adc_mode_t
 - Quark D2000 ADC, [11](#)
- qm_adc_reg_t, [246](#)
- qm_adc_resolution_t
 - Quark D2000 ADC, [11](#)
- qm_adc_set_calibration
 - Quark D2000 ADC, [14](#)
- qm_adc_set_config
 - Quark D2000 ADC, [14](#)
- qm_adc_set_mode
 - Quark D2000 ADC, [14](#)
- qm_adc_status_t
 - Quark D2000 ADC, [11](#)
- qm_adc_t
 - SoC Registers (D2000), [209](#)
- qm_adc_xfer_t, [247](#)
 - callback, [248](#)
 - callback_data, [248](#)
 - ch, [248](#)
 - ch_len, [248](#)
 - samples, [248](#)
 - samples_len, [248](#)
- qm_aonc_disable
 - Always-on Counters, [17](#)
- qm_aonc_enable
 - Always-on Counters, [17](#)
- qm_aonc_get_value
 - Always-on Counters, [17](#)
- qm_aonc_reg_t, [248](#)
 - aonc_cfg, [249](#)
 - aonc_cnt, [249](#)
 - aonpt_cfg, [249](#)
 - aonpt_cnt, [249](#)
 - aonpt_ctrl, [249](#)
 - aonpt_stat, [249](#)
- qm_aonc_t
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [232](#)
- qm_aonpt_clear
 - Always-on Counters, [18](#)
- qm_aonpt_config_t, [249](#)
 - callback, [250](#)
 - callback_data, [250](#)
 - count, [250](#)
 - int_en, [250](#)
- qm_aonpt_get_status

- Always-on Counters, [18](#)
- qm_aonpt_get_value
 - Always-on Counters, [18](#)
- qm_aonpt_reset
 - Always-on Counters, [19](#)
- qm_aonpt_restore_context
 - Always-on Counters, [19](#)
- qm_aonpt_save_context
 - Always-on Counters, [19](#)
- qm_aonpt_set_config
 - Always-on Counters, [20](#)
- qm_aonpt_status_t
 - Always-on Counters, [17](#)
- qm_dma_burst_length_t
 - DMA, [23](#)
- qm_dma_chan_reg_t, [250](#)
- qm_dma_channel_config_t, [252](#)
 - client_callback, [252](#)
- qm_dma_channel_direction_t
 - DMA, [23](#)
- qm_dma_channel_id_t
 - SoC Registers (D2000), [209](#)
 - SoC Registers (SE), [232](#)
- qm_dma_channel_set_config
 - DMA, [24](#)
- qm_dma_context_t, [252](#)
 - cfg_high, [253](#)
 - cfg_low, [253](#)
 - ctrl_low, [253](#)
 - llp_low, [253](#)
 - misc_cfg_low, [253](#)
- qm_dma_handshake_interface_t
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [232](#)
- qm_dma_handshake_polarity_t
 - DMA, [23](#)
- qm_dma_init
 - DMA, [25](#)
- qm_dma_int_reg_t, [254](#)
- qm_dma_misc_reg_t, [255](#)
- qm_dma_multi_transfer_set_config
 - DMA, [25](#)
- qm_dma_multi_transfer_t, [256](#)
 - block_size, [257](#)
 - destination_address, [257](#)
 - num_blocks, [257](#)
 - source_address, [257](#)
- qm_dma_restore_context
 - DMA, [26](#)
- qm_dma_save_context
 - DMA, [26](#)
- qm_dma_t
 - SoC Registers (D2000), [210](#)
 - SoC Registers (SE), [233](#)
- qm_dma_transfer_mem_to_mem
 - DMA, [27](#)
- qm_dma_transfer_set_config
 - DMA, [27](#)
- qm_dma_transfer_start
 - DMA, [28](#)
- qm_dma_transfer_t, [257](#)
 - block_size, [257](#)
 - destination_address, [258](#)
 - source_address, [258](#)
- qm_dma_transfer_terminate
 - DMA, [28](#)
- qm_dma_transfer_type_t
 - DMA, [23](#)
- qm_dma_transfer_width_t
 - DMA, [24](#)
- qm_flash_config_t, [258](#)
 - us_count, [258](#)
 - wait_states, [258](#)
 - write_disable, [259](#)
- qm_flash_context_t, [259](#)
 - ctrl, [259](#)
 - tmg_ctrl, [259](#)
- qm_flash_disable_t
 - Flash, [30](#)
- qm_flash_mass_erase
 - Flash, [31](#)
- qm_flash_page_erase
 - Flash, [31](#)
- qm_flash_page_update
 - Flash, [32](#)
- qm_flash_page_write
 - Flash, [32](#)
- qm_flash_reg_t, [259](#)
 - ctrl, [260](#)
 - flash_stts, [260](#)
 - flash_wr_ctrl, [260](#)
 - flash_wr_data, [260](#)
 - fpr_rd_cfg, [261](#)
 - mpr_vsts, [261](#)
 - mpr_wr_cfg, [261](#)
 - rom_wr_ctrl, [261](#)
 - rom_wr_data, [261](#)
 - tmg_ctrl, [261](#)
- qm_flash_region_t
 - Flash, [31](#)
- qm_flash_region_type_t
 - FPR, [37](#)
- qm_flash_restore_context
 - Flash, [34](#)
- qm_flash_save_context
 - Flash, [34](#)
- qm_flash_set_config
 - Flash, [35](#)
- qm_flash_t
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [233](#)
- qm_flash_word_write
 - Flash, [35](#)
- qm_fpr_config_t, [261](#)
 - allow_agents, [262](#)
 - en_mask, [262](#)

- low_bound, [262](#)
- up_bound, [262](#)
- qm_fpr_context_t, [262](#)
 - fpr_rd_cfg, [263](#)
- qm_fpr_en_t
 - FPR, [37](#)
- qm_fpr_id_t
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [233](#)
- qm_fpr_read_allow_t
 - FPR, [38](#)
- qm_fpr_restore_context
 - FPR, [38](#)
- qm_fpr_save_context
 - FPR, [39](#)
- qm_fpr_set_config
 - FPR, [39](#)
- qm_fpr_set_violation_policy
 - FPR, [39](#)
- qm_fpr_viol_mode_t
 - FPR, [38](#)
- qm_gpio_clear_pin
 - GPIO, [43](#)
- qm_gpio_context_t, [263](#)
 - gpio_debounce, [264](#)
 - gpio_int_bothedge, [264](#)
 - gpio_int_polarity, [264](#)
 - gpio_inten, [264](#)
 - gpio_intmask, [264](#)
 - gpio_inttype_level, [264](#)
 - gpio_ls_sync, [264](#)
 - gpio_swporta_ctl, [264](#)
 - gpio_swporta_ddr, [264](#)
 - gpio_swporta_dr, [265](#)
- qm_gpio_port_config_t, [265](#)
 - callback, [265](#)
 - callback_data, [266](#)
 - direction, [266](#)
 - int_bothedge, [266](#)
 - int_debounce, [266](#)
 - int_en, [266](#)
 - int_polarity, [266](#)
 - int_type, [266](#)
- qm_gpio_read_pin
 - GPIO, [43](#)
- qm_gpio_read_port
 - GPIO, [43](#)
- qm_gpio_reg_t, [266](#)
 - gpio_config_reg1, [267](#)
 - gpio_config_reg2, [267](#)
 - gpio_debounce, [268](#)
 - gpio_ext_porta, [268](#)
 - gpio_id_code, [268](#)
 - gpio_int_bothedge, [268](#)
 - gpio_int_polarity, [268](#)
 - gpio_inten, [268](#)
 - gpio_intmask, [268](#)
 - gpio_intstatus, [268](#)
 - gpio_inttype_level, [268](#)
 - gpio_ls_sync, [269](#)
 - gpio_porta_eoi, [269](#)
 - gpio_raw_intstatus, [269](#)
 - gpio_swporta_ddr, [269](#)
 - gpio_swporta_dr, [269](#)
 - gpio_ver_id_code, [269](#)
- qm_gpio_restore_context
 - GPIO, [44](#)
- qm_gpio_save_context
 - GPIO, [44](#)
- qm_gpio_set_config
 - GPIO, [44](#)
- qm_gpio_set_pin
 - GPIO, [45](#)
- qm_gpio_set_pin_state
 - GPIO, [45](#)
- qm_gpio_state_t
 - GPIO, [42](#)
- qm_gpio_t
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- qm_gpio_write_port
 - GPIO, [46](#)
- qm_i2c
 - SoC Registers (D2000), [212](#)
 - SoC Registers (SE), [235](#)
- qm_i2c_addr_t
 - I2C, [48](#)
- qm_i2c_config_t, [269](#)
 - address_mode, [270](#)
 - mode, [270](#)
 - slave_addr, [270](#)
 - speed, [270](#)
- qm_i2c_context_t, [270](#)
 - con, [271](#)
 - enable, [271](#)
 - fs_scl_hcnt, [271](#)
 - fs_scl_lcnt, [271](#)
 - fs_spklen, [271](#)
 - ic_intr_mask, [271](#)
 - sar, [272](#)
 - ss_scl_hcnt, [272](#)
 - ss_scl_lcnt, [272](#)
 - tx_tl, [272](#)
- qm_i2c_dma_channel_config
 - I2C, [50](#)
- qm_i2c_dma_transfer_terminate
 - I2C, [50](#)
- qm_i2c_get_status
 - I2C, [51](#)
- qm_i2c_irq_transfer_terminate
 - I2C, [51](#)
- qm_i2c_master_dma_transfer
 - I2C, [51](#)
- qm_i2c_master_irq_transfer
 - I2C, [52](#)
- qm_i2c_master_read

- I2C, [52](#)
- qm_i2c_master_write
 - I2C, [53](#)
- qm_i2c_mode_t
 - I2C, [48](#)
- qm_i2c_reg_t, [272](#)
 - ic_ack_general_call, [274](#)
 - ic_clr_activity, [274](#)
 - ic_clr_gen_call, [274](#)
 - ic_clr_intr, [274](#)
 - ic_clr_rd_req, [274](#)
 - ic_clr_restart_det, [275](#)
 - ic_clr_rx_done, [275](#)
 - ic_clr_rx_over, [275](#)
 - ic_clr_rx_under, [275](#)
 - ic_clr_start_det, [275](#)
 - ic_clr_stop_det, [275](#)
 - ic_clr_tx_abrt, [275](#)
 - ic_clr_tx_over, [275](#)
 - ic_comp_param_1, [275](#)
 - ic_comp_type, [275](#)
 - ic_comp_version, [275](#)
 - ic_con, [276](#)
 - ic_data_cmd, [276](#)
 - ic_dma_cr, [276](#)
 - ic_dma_rdlr, [276](#)
 - ic_dma_tdlr, [276](#)
 - ic_enable, [276](#)
 - ic_enable_status, [276](#)
 - ic_fs_scl_hcnt, [276](#)
 - ic_fs_scl_lcnt, [276](#)
 - ic_fs_spklen, [277](#)
 - ic_hs_maddr, [277](#)
 - ic_hs_scl_hcnt, [277](#)
 - ic_hs_scl_lcnt, [277](#)
 - ic_hs_spklen, [277](#)
 - ic_intr_mask, [277](#)
 - ic_intr_stat, [277](#)
 - ic_raw_intr_stat, [277](#)
 - ic_rx_tl, [277](#)
 - ic_rxflr, [277](#)
 - ic_sar, [278](#)
 - ic_sda_hold, [278](#)
 - ic_sda_setup, [278](#)
 - ic_ss_scl_hcnt, [278](#)
 - ic_ss_scl_lcnt, [278](#)
 - ic_status, [278](#)
 - ic_tar, [278](#)
 - ic_tx_abrt_source, [278](#)
 - ic_tx_tl, [278](#)
 - ic_txflr, [279](#)
- qm_i2c_restore_context
 - I2C, [53](#)
- qm_i2c_save_context
 - I2C, [54](#)
- qm_i2c_set_config
 - I2C, [54](#)
- qm_i2c_set_speed
 - I2C, [55](#)
- qm_i2c_slave_dma_transfer
 - I2C, [55](#)
- qm_i2c_slave_irq_transfer
 - I2C, [56](#)
- qm_i2c_slave_irq_transfer_update
 - I2C, [56](#)
- qm_i2c_slave_stop_t
 - I2C, [48](#)
- qm_i2c_speed_t
 - I2C, [49](#)
- qm_i2c_status_t
 - I2C, [49](#)
- qm_i2c_t
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- qm_i2c_transfer_t, [279](#)
 - callback, [279](#)
 - callback_data, [280](#)
 - rx, [280](#)
 - rx_len, [280](#)
 - stop, [280](#)
 - tx, [280](#)
 - tx_len, [280](#)
- qm_i2s_audio_format_t
 - I2S, [59](#)
- qm_i2s_audio_rate_t
 - I2S, [59](#)
- qm_i2s_audio_stream_t
 - I2S, [60](#)
- qm_i2s_buffer_link, [280](#)
 - buff, [281](#)
 - dma_link_head, [281](#)
 - next, [281](#)
- qm_i2s_buffer_mode_t
 - I2S, [60](#)
- qm_i2s_channel_cfg_data_t, [281](#)
 - aempty_thresh, [282](#)
 - afull_thresh, [282](#)
 - block_size, [282](#)
 - buffer_len, [282](#)
 - buffer_link, [282](#)
 - callback, [283](#)
 - callback_data, [283](#)
 - num_buffer_links, [283](#)
 - num_dma_link_per_buffer, [283](#)
 - num_dma_links, [283](#)
- qm_i2s_channel_disable
 - I2S, [61](#)
- qm_i2s_clk_src_t
 - I2S, [60](#)
- qm_i2s_clock_cfg_data_t, [283](#)
 - i2s_mclk_divisor, [284](#)
 - i2s_mclk_output_en, [284](#)
- qm_i2s_dma_channel_config
 - I2S, [62](#)
- qm_i2s_dma_transfer
 - I2S, [62](#)

- qm_i2s_dma_transfer_terminate
 - I2S, [62](#)
- qm_i2s_master_slave_t
 - I2S, [60](#)
- qm_i2s_sample_resolution_t
 - I2S, [60](#)
- qm_i2s_set_clock_config
 - I2S, [63](#)
- qm_i2s_status_t
 - I2S, [61](#)
- qm_int_vector_request
 - Interrupt, [66](#)
- qm_interrupt_router_reg_t, [284](#)
 - adc_0_cal_int_mask, [286](#)
 - adc_0_pwr_int_mask, [286](#)
 - aon_gpio_0_int_mask, [286](#)
 - aonpt_0_int_mask, [286](#)
 - comparator_0_host_halt_int_mask, [286](#)
 - comparator_0_host_int_mask, [286](#)
 - comparator_0_ss_halt_int_mask, [287](#)
 - comparator_0_ss_int_mask, [287](#)
 - dma_0_error_int_mask, [287](#)
 - dma_0_int_0_mask, [287](#)
 - dma_0_int_1_mask, [287](#)
 - dma_0_int_2_mask, [287](#)
 - dma_0_int_3_mask, [287](#)
 - dma_0_int_4_mask, [287](#)
 - dma_0_int_5_mask, [287](#)
 - dma_0_int_6_mask, [287](#)
 - dma_0_int_7_mask, [288](#)
 - flash_mpr_0_int_mask, [288](#)
 - flash_mpr_1_int_mask, [288](#)
 - gpio_0_int_mask, [288](#)
 - host_bus_error_int_mask, [288](#)
 - i2c_master_0_int_mask, [288](#)
 - i2c_master_1_int_mask, [288](#)
 - i2s_0_int_mask, [288](#)
 - lock_int_mask_reg, [288](#)
 - mailbox_0_int_mask, [288](#)
 - pwm_0_int_mask, [289](#)
 - rtc_0_int_mask, [289](#)
 - spi_master_0_int_mask, [289](#)
 - spi_master_1_int_mask, [289](#)
 - spi_slave_0_int_mask, [289](#)
 - sram_mpr_0_int_mask, [289](#)
 - ss_adc_0_error_int_mask, [289](#)
 - ss_adc_0_int_mask, [289](#)
 - ss_gpio_0_int_mask, [289](#)
 - ss_gpio_1_int_mask, [289](#)
 - ss_i2c_0_int, [290](#)
 - ss_i2c_1_int, [290](#)
 - ss_spi_0_int, [290](#)
 - ss_spi_1_int, [290](#)
 - timer_0_int_mask, [290](#)
 - uart_0_int_mask, [290](#)
 - uart_1_int_mask, [290](#)
 - usb_0_int_mask, [290](#)
 - wdt_0_int_mask, [290](#)
- qm_irq_context_t, [291](#)
 - irq_config, [291](#)
 - irq_ctrl, [291](#)
 - redtbl_entries, [291](#)
 - status32_irq_enable, [291](#)
 - status32_irq_threshold, [291](#)
- qm_irq_lock
 - Interrupt, [66](#)
- qm_irq_mask
 - Interrupt, [66](#)
- qm_irq_unlock
 - Interrupt, [67](#)
- qm_irq_unmask
 - Interrupt, [67](#)
- qm_lapic_reg_t, [292](#)
- qm_mailbox_reg_t, [293](#)
- qm_mailbox_t, [293](#)
- qm_mbox_callback_t
 - Mailbox, [78](#)
- qm_mbox_ch_get_status
 - Mailbox, [79](#)
- qm_mbox_ch_read
 - Mailbox, [80](#)
- qm_mbox_ch_set_config
 - Mailbox, [80](#)
- qm_mbox_ch_status_t
 - Mailbox, [79](#)
- qm_mbox_ch_t
 - Mailbox Definitions, [220](#)
- qm_mbox_ch_write
 - Mailbox, [81](#)
- qm_mbox_config_t, [294](#)
 - callback, [294](#)
 - dest, [294](#)
 - mode, [294](#)
- qm_mbox_destination_t
 - Mailbox Definitions, [220](#)
- qm_mbox_mode_t
 - Mailbox, [79](#)
- qm_mbox_msg_t, [295](#)
 - data, [295](#)
- qm_mbox_payload_t
 - Mailbox, [79](#)
- qm_mpr_config_t, [295](#)
- qm_mpr_context_t, [296](#)
 - mpr_cfg, [296](#)
- qm_mpr_id_t
 - SoC Registers (D2000), [211](#)
 - SoC Registers (SE), [234](#)
- qm_mpr_reg_t, [296](#)
- qm_mpr_restore_context
 - MPR, [82](#)
- qm_mpr_save_context
 - MPR, [82](#)
- qm_mpr_set_config
 - MPR, [84](#)
- qm_mpr_set_violation_policy
 - MPR, [84](#)

- qm_mvic_reg_t, 297
 - ccr, 297
 - eoi, 297
 - icr, 297
 - irr, 297
 - isr, 297
 - lvtimer, 298
 - ppr, 298
 - sivr, 298
 - tpr, 298
- qm_pic_timer_config_t, 298
 - callback, 298
 - callback_data, 299
 - int_en, 299
 - mode, 299
- qm_pic_timer_context_t, 299
 - lvtimer, 299
 - timer_dcr, 300
 - timer_icr, 300
- qm_pic_timer_get
 - PIC Timer, 85
- qm_pic_timer_mode_t
 - PIC Timer, 85
- qm_pic_timer_reg_t, 300
- qm_pic_timer_restore_context
 - PIC Timer, 86
- qm_pic_timer_save_context
 - PIC Timer, 86
- qm_pic_timer_set
 - PIC Timer, 86
- qm_pic_timer_set_config
 - PIC Timer, 88
- qm_pin_id_t
 - Pin Muxing setup, 89
- qm_pmux_fn_t
 - Pin Muxing setup, 91
- qm_pmux_input_en
 - Pin Muxing setup, 92
- qm_pmux_pullup_en
 - Pin Muxing setup, 92
- qm_pmux_select
 - Pin Muxing setup, 92
- qm_pmux_set_slew
 - Pin Muxing setup, 93
- qm_pmux_slew_t
 - Pin Muxing setup, 91
- qm_power_cpu_c1
 - Quark SE Host Power states, 217
- qm_power_cpu_c2
 - Quark SE Host Power states, 217
- qm_power_cpu_c2lp
 - Quark SE Host Power states, 218
- qm_power_cpu_halt
 - Quark D2000 Power states, 196
- qm_power_sleep_wait
 - Quark SE SoC Power states, 214
- qm_power_soc_deep_sleep
 - Quark D2000 Power states, 197
- Quark SE SoC Power states, 214
- qm_power_soc_deep_sleep_restore
 - Quark SE SoC Power states, 215
- qm_power_soc_restore
 - Quark D2000 Power states, 197
- qm_power_soc_sleep
 - Quark D2000 Power states, 197
 - Quark SE SoC Power states, 215
- qm_power_soc_sleep_restore
 - Quark SE SoC Power states, 216
- qm_power_wake_event_t
 - Quark D2000 Power states, 196
- qm_pwm_channel_t, 300
 - controlreg, 301
 - currentvalue, 301
 - eoi, 301
 - intstatus, 301
 - loadcount, 301
- qm_pwm_config_t, 301
 - callback, 302
 - callback_data, 302
 - hi_count, 302
 - lo_count, 302
 - mask_interrupt, 302
 - mode, 303
- qm_pwm_context_t, 303
 - controlreg, 303
 - loadcount, 303
 - loadcount2, 303
- qm_pwm_get
 - PWM / Timer, 95
- qm_pwm_id_t
 - SoC Registers (D2000), 212
 - SoC Registers (SE), 234
- qm_pwm_mode_t
 - PWM / Timer, 94
- qm_pwm_reg_t, 303
 - timer, 304
- qm_pwm_restore_context
 - PWM / Timer, 96
- qm_pwm_save_context
 - PWM / Timer, 96
- qm_pwm_set
 - PWM / Timer, 97
- qm_pwm_set_config
 - PWM / Timer, 97
- qm_pwm_start
 - PWM / Timer, 98
- qm_pwm_stop
 - PWM / Timer, 98
- qm_pwm_t
 - SoC Registers (D2000), 212
 - SoC Registers (SE), 234
- qm_rar_set_mode
 - Quark(TM) D2000 Retention Alternator Regulator (RAR)., 199
- qm_rar_state_t

- Quark(TM) D2000 Retention Alternator Regulator (RAR)., 199
- qm_rtc_config_t, 304
 - alarm_en, 305
 - alarm_val, 305
 - callback, 305
 - callback_data, 305
 - init_val, 305
 - prescaler, 305
- qm_rtc_read
 - RTC, 99
- qm_rtc_reg_t, 305
 - rtc_ccr, 306
 - rtc_ccvr, 306
 - rtc_clr, 306
 - rtc_cmr, 306
 - rtc_comp_version, 306
 - rtc_eoi, 306
 - rtc_rstat, 306
 - rtc_stat, 307
- qm_rtc_restore_context
 - RTC, 100
- qm_rtc_save_context
 - RTC, 100
- qm_rtc_set_alarm
 - RTC, 100
- qm_rtc_set_config
 - RTC, 101
- qm_rtc_t
 - SoC Registers (D2000), 212
 - SoC Registers (SE), 234
- qm_scss_ccu_reg_t, 307
 - ccu_ext_clock_ctl, 308
 - ccu_gpio_db_clk_ctl, 308
 - ccu_lp_clk_ctl, 308
 - ccu_mlayer_ahb_ctl, 308
 - ccu_periph_clk_div_ctl0, 308
 - ccu_periph_clk_gate_ctl, 308
 - ccu_ss_periph_clk_gate_ctl, 308
 - ccu_sys_clk_ctl, 308
 - osc0_cfg0, 308
 - osc0_cfg1, 308
 - osc0_stat1, 309
 - osc1_cfg0, 309
 - osc1_stat0, 309
 - osc_lock_0, 309
 - soc_ctrl, 309
 - soc_ctrl_lock, 309
 - usb_pll_cfg0, 309
 - wake_mask, 309
- qm_scss_cmp_reg_t, 309
 - cmp_en, 310
 - cmp_pwr, 310
 - cmp_ref_pol, 310
 - cmp_ref_sel, 310
 - cmp_stat_clr, 310
- qm_scss_gp_reg_t, 310
 - gp0, 311
 - gp1, 311
 - gp2, 311
 - gp3, 311
 - gps0, 311
 - gps1, 311
 - gps2, 312
 - gps3, 312
 - wo_sp, 312
 - wo_st, 312
- qm_scss_info_reg_t, 312
 - cotps, 313
 - dotps, 313
 - fs, 313
 - id, 313
 - rev, 313
 - rs, 313
- qm_scss_mem_reg_t, 313
- qm_scss_peripheral_reg_t, 313
 - cfg_lock, 314
 - periph_cfg0, 314
- qm_scss_pmu_reg_t, 314
 - aon_vr, 315
 - p_sts, 315
 - pm_lock, 315
 - pm_wait, 315
 - rstc, 315
 - rsts, 315
- qm_scss_pmux_reg_t, 315
 - pmux_in_en, 316
 - pmux_in_en_lock, 316
 - pmux_pullup, 316
 - pmux_pullup_lock, 316
 - pmux_sel, 316
 - pmux_sel_0_lock, 316
 - pmux_slew, 317
 - pmux_slew_lock, 317
- qm_scss_ss_reg_t, 317
- qm_soc_id
 - Identification, 64
- qm_soc_reset
 - Initialisation, 65
- qm_soc_reset_t
 - Initialisation, 65
- qm_soc_version
 - Identification, 64
- qm_spi_async_transfer_t, 317
 - callback, 318
 - callback_data, 318
 - keep_enabled, 318
 - rx, 318
 - rx_len, 318
 - tx, 318
 - tx_len, 319
- qm_spi_bmode_t
 - SPI, 103
- qm_spi_context_t, 319
 - baudr, 319
 - ctrlr0, 319

- ser, [319](#)
- qm_spi_dma_channel_config
 - SPI, [105](#)
- qm_spi_dma_transfer
 - SPI, [106](#)
- qm_spi_dma_transfer_terminate
 - SPI, [106](#)
- qm_spi_frame_format_t
 - SPI, [103](#)
- qm_spi_frame_size_t
 - SPI, [103](#)
- qm_spi_get_status
 - SPI, [107](#)
- qm_spi_irq_transfer
 - SPI, [107](#)
- qm_spi_irq_transfer_terminate
 - SPI, [108](#)
- qm_spi_irq_update
 - SPI, [108](#)
- qm_spi_reg_t, [320](#)
 - baudr, [321](#)
 - ctrlr0, [321](#)
 - ctrlr1, [321](#)
 - dmacr, [321](#)
 - dmardlr, [321](#)
 - dmatdlr, [321](#)
 - dr, [322](#)
 - icr, [322](#)
 - idr, [322](#)
 - imr, [322](#)
 - isr, [322](#)
 - msticr, [322](#)
 - mwcr, [322](#)
 - risr, [322](#)
 - rx_sample_dly, [322](#)
 - rxflr, [322](#)
 - rxftlr, [323](#)
 - rxoicr, [323](#)
 - rxuicr, [323](#)
 - ser, [323](#)
 - sr, [323](#)
 - ssi_comp_version, [323](#)
 - ssienr, [323](#)
 - txflr, [323](#)
 - txftlr, [323](#)
 - txoicr, [324](#)
- qm_spi_restore_context
 - SPI, [109](#)
- qm_spi_save_context
 - SPI, [109](#)
- qm_spi_set_config
 - SPI, [110](#)
- qm_spi_slave_select
 - SPI, [110](#)
- qm_spi_slave_select_t
 - SPI, [104](#)
- qm_spi_status_t
 - SPI, [104](#)
- qm_spi_t
 - SoC Registers (D2000), [212](#)
 - SoC Registers (SE), [235](#)
- qm_spi_tmode_t
 - SPI, [105](#)
- qm_spi_transfer
 - SPI, [110](#)
- qm_spi_transfer_t, [324](#)
 - rx, [324](#)
 - rx_len, [324](#)
 - tx, [324](#)
 - tx_len, [325](#)
- qm_ss_adc_calibrate
 - SS ADC, [115](#)
- qm_ss_adc_cb_source_t
 - SS ADC, [113](#)
- qm_ss_adc_channel_t
 - SS ADC, [113](#)
- qm_ss_adc_config_t, [325](#)
 - resolution, [325](#)
- qm_ss_adc_context_t, [325](#)
 - adc_ctrl, [326](#)
 - adc_divseqstat, [326](#)
 - adc_seq, [326](#)
 - adc_set, [326](#)
- qm_ss_adc_convert
 - SS ADC, [115](#)
- qm_ss_adc_get_calibration
 - SS ADC, [115](#)
- qm_ss_adc_irq_calibrate
 - SS ADC, [116](#)
- qm_ss_adc_irq_convert
 - SS ADC, [116](#)
- qm_ss_adc_irq_set_mode
 - SS ADC, [116](#)
- qm_ss_adc_mode_t
 - SS ADC, [114](#)
- qm_ss_adc_reg_t
 - SoC Registers (Sensor Subsystem), [222](#)
- qm_ss_adc_resolution_t
 - SS ADC, [114](#)
- qm_ss_adc_restore_context
 - SS ADC, [118](#)
- qm_ss_adc_save_context
 - SS ADC, [118](#)
- qm_ss_adc_set_calibration
 - SS ADC, [119](#)
- qm_ss_adc_set_config
 - SS ADC, [119](#)
- qm_ss_adc_set_mode
 - SS ADC, [119](#)
- qm_ss_adc_status_t
 - SS ADC, [114](#)
- qm_ss_adc_t
 - SoC Registers (Sensor Subsystem), [222](#)
- qm_ss_adc_xfer_t, [326](#)
 - callback, [327](#)
 - callback_data, [327](#)

- ch, [327](#)
- ch_len, [327](#)
- samples, [327](#)
- samples_len, [327](#)
- qm_ss_creg_reg_t
 - SoC Registers (Sensor Subsystem), [222](#)
- qm_ss_gpio_clear_pin
 - SS GPIO, [122](#)
- qm_ss_gpio_context_t, [328](#)
 - gpio_debounce, [328](#)
 - gpio_int_polarity, [328](#)
 - gpio_inten, [328](#)
 - gpio_intmask, [329](#)
 - gpio_inttype_level, [329](#)
 - gpio_ls_sync, [329](#)
 - gpio_swporta_ddr, [329](#)
 - gpio_swporta_dr, [329](#)
- qm_ss_gpio_port_config_t, [329](#)
 - callback, [330](#)
 - callback_data, [330](#)
 - direction, [330](#)
 - int_debounce, [330](#)
 - int_en, [330](#)
 - int_polarity, [330](#)
 - int_type, [331](#)
- qm_ss_gpio_read_pin
 - SS GPIO, [122](#)
- qm_ss_gpio_read_port
 - SS GPIO, [122](#)
- qm_ss_gpio_reg_t
 - SoC Registers (Sensor Subsystem), [223](#)
- qm_ss_gpio_restore_context
 - SS GPIO, [123](#)
- qm_ss_gpio_save_context
 - SS GPIO, [123](#)
- qm_ss_gpio_set_config
 - SS GPIO, [123](#)
- qm_ss_gpio_set_pin
 - SS GPIO, [124](#)
- qm_ss_gpio_set_pin_state
 - SS GPIO, [124](#)
- qm_ss_gpio_state_t
 - SS GPIO, [121](#)
- qm_ss_gpio_t
 - SoC Registers (Sensor Subsystem), [223](#)
- qm_ss_gpio_write_port
 - SS GPIO, [125](#)
- qm_ss_i2c_addr_t
 - SS I2C, [127](#)
- qm_ss_i2c_config_t, [331](#)
 - address_mode, [331](#)
 - speed, [331](#)
- qm_ss_i2c_context_t, [331](#)
- qm_ss_i2c_get_status
 - SS I2C, [128](#)
- qm_ss_i2c_irq_transfer_terminate
 - SS I2C, [128](#)
- qm_ss_i2c_master_irq_transfer
 - SS I2C, [128](#)
- qm_ss_i2c_master_read
 - SS I2C, [129](#)
- qm_ss_i2c_master_write
 - SS I2C, [129](#)
- qm_ss_i2c_reg_t
 - SoC Registers (Sensor Subsystem), [223](#)
- qm_ss_i2c_restore_context
 - SS I2C, [130](#)
- qm_ss_i2c_save_context
 - SS I2C, [130](#)
- qm_ss_i2c_set_config
 - SS I2C, [131](#)
- qm_ss_i2c_set_speed
 - SS I2C, [131](#)
- qm_ss_i2c_speed_t
 - SS I2C, [127](#)
- qm_ss_i2c_status_t
 - SS I2C, [127](#)
- qm_ss_i2c_transfer_t, [332](#)
 - callback, [332](#)
 - callback_data, [333](#)
 - rx, [333](#)
 - rx_len, [333](#)
 - stop, [333](#)
 - tx, [333](#)
 - tx_len, [333](#)
- qm_ss_int_vector_request
 - SS Interrupt, [132](#)
- qm_ss_irq_mask
 - SS Interrupt, [132](#)
- qm_ss_irq_request
 - SS Interrupt, [132](#)
- qm_ss_irq_unmask
 - SS Interrupt, [133](#)
- qm_ss_power_cpu_ss1
 - SS Power states, [238](#)
- qm_ss_power_cpu_ss1_mode_t
 - SS Power states, [237](#)
- qm_ss_power_cpu_ss2
 - SS Power states, [238](#)
- qm_ss_power_sleep_wait
 - SS Power states, [238](#)
- qm_ss_power_soc_deep_sleep_restore
 - SS Power states, [238](#)
- qm_ss_power_soc_lpss_disable
 - SS Power states, [239](#)
- qm_ss_power_soc_lpss_enable
 - SS Power states, [239](#)
- qm_ss_power_soc_sleep_restore
 - SS Power states, [239](#)
- qm_ss_spi_async_transfer_t, [333](#)
 - callback, [334](#)
 - rx, [334](#)
 - rx_len, [334](#)
 - tx, [334](#)
 - tx_len, [334](#)
- qm_ss_spi_bmode_t

- SS SPI, [141](#)
- qm_ss_spi_config_t, [335](#)
 - clk_divider, [335](#)
- qm_ss_spi_context_t, [335](#)
 - spi_ctrl, [336](#)
 - spi_spie, [336](#)
 - spi_timing, [336](#)
- qm_ss_spi_frame_size_t
 - SS SPI, [141](#)
- qm_ss_spi_get_status
 - SS SPI, [143](#)
- qm_ss_spi_irq_transfer
 - SS SPI, [143](#)
- qm_ss_spi_irq_transfer_terminate
 - SS SPI, [144](#)
- qm_ss_spi_reg_t
 - SoC Registers (Sensor Subsystem), [223](#)
- qm_ss_spi_restore_context
 - SS SPI, [144](#)
- qm_ss_spi_save_context
 - SS SPI, [144](#)
- qm_ss_spi_set_config
 - SS SPI, [145](#)
- qm_ss_spi_slave_select
 - SS SPI, [145](#)
- qm_ss_spi_slave_select_t
 - SS SPI, [142](#)
- qm_ss_spi_status_t
 - SS SPI, [142](#)
- qm_ss_spi_t
 - SoC Registers (Sensor Subsystem), [223](#)
- qm_ss_spi_tmode_t
 - SS SPI, [142](#)
- qm_ss_spi_transfer
 - SS SPI, [146](#)
- qm_ss_spi_transfer_t, [336](#)
 - rx, [337](#)
 - rx_len, [337](#)
 - tx, [337](#)
 - tx_len, [337](#)
- qm_ss_timer_config_t, [337](#)
 - callback, [338](#)
 - callback_data, [338](#)
 - count, [338](#)
 - inc_run_only, [338](#)
 - int_en, [338](#)
 - watchdog_mode, [338](#)
- qm_ss_timer_get
 - SS Timer, [147](#)
- qm_ss_timer_set
 - SS Timer, [147](#)
- qm_ss_timer_set_config
 - SS Timer, [148](#)
- qm_uart_config_t, [338](#)
 - baud_divisor, [339](#)
 - hw_fc, [339](#)
 - line_control, [339](#)
- qm_uart_context_t, [339](#)
 - dlf, [340](#)
 - dlh, [340](#)
 - dll, [340](#)
 - htx, [340](#)
 - ier, [340](#)
 - lcr, [340](#)
 - mcr, [341](#)
 - scr, [341](#)
- qm_uart_dma_channel_config
 - UART, [151](#)
- qm_uart_dma_read
 - UART, [152](#)
- qm_uart_dma_read_terminate
 - UART, [152](#)
- qm_uart_dma_write
 - UART, [153](#)
- qm_uart_dma_write_terminate
 - UART, [153](#)
- qm_uart_get_status
 - UART, [153](#)
- qm_uart_irq_read
 - UART, [154](#)
- qm_uart_irq_read_terminate
 - UART, [154](#)
- qm_uart_irq_write
 - UART, [155](#)
- qm_uart_irq_write_terminate
 - UART, [155](#)
- qm_uart_lc_t
 - UART, [150](#)
- qm_uart_read
 - UART, [155](#)
- qm_uart_read_non_block
 - UART, [156](#)
- qm_uart_reg_t, [341](#)
 - de_en, [342](#)
 - det, [342](#)
 - dlf, [342](#)
 - dmasa, [342](#)
 - htx, [342](#)
 - ier_dlh, [342](#)
 - iir_fcr, [342](#)
 - lcr, [343](#)
 - lcr_ext, [343](#)
 - lsr, [343](#)
 - mcr, [343](#)
 - msr, [343](#)
 - rar, [343](#)
 - rbr_thr_dll, [343](#)
 - re_en, [343](#)
 - scr, [343](#)
 - tar, [344](#)
 - tat, [344](#)
 - tcr, [344](#)
 - usr, [344](#)
- qm_uart_restore_context
 - UART, [156](#)
- qm_uart_save_context

- UART, [157](#)
- qm_uart_set_config
 - UART, [157](#)
- qm_uart_status_t
 - UART, [151](#)
- qm_uart_t
 - SoC Registers (D2000), [212](#)
 - SoC Registers (SE), [235](#)
- qm_uart_transfer_t, [344](#)
 - callback, [344](#)
 - callback_data, [345](#)
 - data, [345](#)
 - data_len, [345](#)
- qm_uart_write
 - UART, [157](#)
- qm_uart_write_buffer
 - UART, [158](#)
- qm_uart_write_non_block
 - UART, [158](#)
- qm_usb_attach
 - USB, [162](#)
- qm_usb_detach
 - USB, [162](#)
- qm_usb_ep_config_t, [345](#)
 - callback, [345](#)
 - callback_data, [346](#)
 - max_packet_size, [346](#)
 - type, [346](#)
- qm_usb_ep_disable
 - USB, [163](#)
- qm_usb_ep_enable
 - USB, [163](#)
- qm_usb_ep_flush
 - USB, [163](#)
- qm_usb_ep_get_bytes_read
 - USB, [164](#)
- qm_usb_ep_halt
 - USB, [164](#)
- qm_usb_ep_is_stalled
 - USB, [164](#)
- qm_usb_ep_read
 - USB, [165](#)
- qm_usb_ep_set_config
 - USB, [165](#)
- qm_usb_ep_set_stall_state
 - USB, [166](#)
- qm_usb_ep_status_t
 - USB, [161](#)
- qm_usb_ep_type_t
 - USB, [161](#)
- qm_usb_ep_write
 - USB, [166](#)
- qm_usb_in_ep_reg_t, [346](#)
- qm_usb_out_ep_reg_t, [346](#)
- qm_usb_reg_t, [347](#)
 - daint, [348](#)
 - daintmsk, [348](#)
 - dcfg, [348](#)
 - dctl, [348](#)
 - diepempmsk, [348](#)
 - diepmsk, [348](#)
 - doepmsk, [348](#)
 - dsts, [349](#)
 - dthrcctl, [349](#)
 - dvbusdis, [349](#)
 - dvbuspulse, [349](#)
 - gahbcfg, [349](#)
 - gdfifocfg, [349](#)
 - ghwcfg1, [349](#)
 - ghwcfg2, [349](#)
 - ghwcfg3, [349](#)
 - ghwcfg4, [349](#)
 - gintmsk, [349](#)
 - gintsts, [350](#)
 - gnptxsiz, [350](#)
 - gotgctl, [350](#)
 - gotgint, [350](#)
 - grstctl, [350](#)
 - grxsiz, [350](#)
 - grxstsp, [350](#)
 - grxstsr, [350](#)
 - gsnpsid, [350](#)
 - gusbcbfg, [350](#)
- qm_usb_reset
 - USB, [166](#)
- qm_usb_set_address
 - USB, [167](#)
- qm_usb_set_status_callback
 - USB, [167](#)
- qm_usb_status_callback_t
 - USB, [161](#)
- qm_usb_status_t
 - USB, [161](#)
- qm_usb_t
 - SoC Registers (SE), [235](#)
- qm_ver_rom
 - Version, [168](#)
- qm_wdt_config_t, [351](#)
 - callback, [351](#)
 - callback_data, [351](#)
 - mode, [351](#)
 - pause_en, [351](#)
 - timeout, [352](#)
- qm_wdt_mode_t
 - WDT, [169](#)
- qm_wdt_reg_t, [352](#)
 - wdt_ccvr, [353](#)
 - wdt_comp_param_1, [353](#)
 - wdt_comp_param_2, [353](#)
 - wdt_comp_param_3, [353](#)
 - wdt_comp_param_4, [353](#)
 - wdt_comp_param_5, [353](#)
 - wdt_comp_type, [353](#)
 - wdt_comp_version, [353](#)
 - wdt_cr, [353](#)
 - wdt_crr, [353](#)

- wdt_eoi, [353](#)
- wdt_stat, [353](#)
- wdt_torr, [354](#)
- qm_wdt_reload
 - WDT, [169](#)
- qm_wdt_restore_context
 - WDT, [170](#)
- qm_wdt_save_context
 - WDT, [170](#)
- qm_wdt_set_config
 - WDT, [170](#)
- qm_wdt_start
 - WDT, [171](#)
- qm_wdt_t
 - SoC Registers (D2000), [212](#)
 - SoC Registers (SE), [235](#)
- Quark D2000 ADC
 - QM_ADC_CAL_COMPLETE, [10](#)
 - QM_ADC_CH_0, [10](#)
 - QM_ADC_CH_1, [10](#)
 - QM_ADC_CH_10, [10](#)
 - QM_ADC_CH_11, [10](#)
 - QM_ADC_CH_12, [10](#)
 - QM_ADC_CH_13, [10](#)
 - QM_ADC_CH_14, [10](#)
 - QM_ADC_CH_15, [10](#)
 - QM_ADC_CH_16, [10](#)
 - QM_ADC_CH_17, [11](#)
 - QM_ADC_CH_18, [11](#)
 - QM_ADC_CH_2, [10](#)
 - QM_ADC_CH_3, [10](#)
 - QM_ADC_CH_4, [10](#)
 - QM_ADC_CH_5, [10](#)
 - QM_ADC_CH_6, [10](#)
 - QM_ADC_CH_7, [10](#)
 - QM_ADC_CH_8, [10](#)
 - QM_ADC_CH_9, [10](#)
 - QM_ADC_COMPLETE, [11](#)
 - QM_ADC_IDLE, [11](#)
 - QM_ADC_MODE_CHANGED, [10](#)
 - QM_ADC_MODE_DEEP_PWR_DOWN, [11](#)
 - QM_ADC_MODE_NORM_CAL, [11](#)
 - QM_ADC_MODE_NORM_NO_CAL, [11](#)
 - QM_ADC_MODE_PWR_DOWN, [11](#)
 - QM_ADC_MODE_STDBY, [11](#)
 - QM_ADC_OVERFLOW, [11](#)
 - QM_ADC_RES_10_BITS, [11](#)
 - QM_ADC_RES_12_BITS, [11](#)
 - QM_ADC_RES_6_BITS, [11](#)
 - QM_ADC_RES_8_BITS, [11](#)
 - QM_ADC_TRANSFER, [10](#)
- Quark D2000 ADC, [9](#)
 - qm_adc_calibrate, [11](#)
 - qm_adc_cb_source_t, [10](#)
 - qm_adc_channel_t, [10](#)
 - qm_adc_convert, [12](#)
 - qm_adc_get_calibration, [12](#)
 - qm_adc_irq_calibrate, [12](#)
 - qm_adc_irq_convert, [13](#)
 - qm_adc_irq_set_mode, [13](#)
 - qm_adc_mode_t, [11](#)
 - qm_adc_resolution_t, [11](#)
 - qm_adc_set_calibration, [14](#)
 - qm_adc_set_config, [14](#)
 - qm_adc_set_mode, [14](#)
 - qm_adc_status_t, [11](#)
- Quark D2000 Flash Layout, [195](#)
- Quark D2000 Power states, [196](#)
 - QM_POWER_WAKE_FROM_GPIO_COMP, [196](#)
 - QM_POWER_WAKE_FROM_RTC, [196](#)
 - qm_power_cpu_halt, [196](#)
 - qm_power_soc_deep_sleep, [197](#)
 - qm_power_soc_restore, [197](#)
 - qm_power_soc_sleep, [197](#)
 - qm_power_wake_event_t, [196](#)
- Quark SE Flash Layout, [213](#)
- Quark SE Host Power states, [217](#)
 - qm_power_cpu_c1, [217](#)
 - qm_power_cpu_c2, [217](#)
 - qm_power_cpu_c2lp, [218](#)
- Quark SE Sensor Subsystem Initialisation, [236](#)
 - sensor_activation, [236](#)
- Quark SE SoC Power states, [214](#)
 - qm_power_sleep_wait, [214](#)
 - qm_power_soc_deep_sleep, [214](#)
 - qm_power_soc_deep_sleep_restore, [215](#)
 - qm_power_soc_sleep, [215](#)
 - qm_power_soc_sleep_restore, [216](#)
- Quark SE Voltage Regulators, [241](#)
 - vreg_aon_set_mode, [241](#)
 - vreg_host_set_mode, [241](#)
 - vreg_plat1p8_set_mode, [242](#)
 - vreg_plat3p3_set_mode, [242](#)
- Quark(TM) D2000 Retention Alternator Regulator (RAR).
 - QM_RAR_NORMAL, [199](#)
 - QM_RAR_RETENTION, [199](#)
- Quark(TM) D2000 Retention Alternator Regulator (RAR)., [199](#)
 - qm_rar_set_mode, [199](#)
 - qm_rar_state_t, [199](#)
- RTC, [99](#)
 - qm_rtc_read, [99](#)
 - qm_rtc_restore_context, [100](#)
 - qm_rtc_save_context, [100](#)
 - qm_rtc_set_alarm, [100](#)
 - qm_rtc_set_config, [101](#)
- rar
 - qm_uart_reg_t, [343](#)
- rbr_thr_dll
 - qm_uart_reg_t, [343](#)
- re_en
 - qm_uart_reg_t, [343](#)
- redtbl_entries
 - qm_irq_context_t, [291](#)
- reference

- qm_ac_config_t, 245
- resolution
 - qm_adc_config_t, 246
 - qm_ss_adc_config_t, 325
- rev
 - qm_scss_info_reg_t, 313
- risr
 - qm_spi_reg_t, 322
- rom_wr_ctrl
 - qm_flash_reg_t, 261
- rom_wr_data
 - qm_flash_reg_t, 261
- rs
 - qm_scss_info_reg_t, 313
- rstc
 - qm_scss_pmu_reg_t, 315
- rsts
 - qm_scss_pmu_reg_t, 315
- rtc_0_int_mask
 - qm_interrupt_router_reg_t, 289
- rtc_ccr
 - qm_rtc_reg_t, 306
- rtc_ccvr
 - qm_rtc_reg_t, 306
- rtc_clr
 - qm_rtc_reg_t, 306
- rtc_cmr
 - qm_rtc_reg_t, 306
- rtc_comp_version
 - qm_rtc_reg_t, 306
- rtc_eoi
 - qm_rtc_reg_t, 306
- rtc_rstat
 - qm_rtc_reg_t, 306
- rtc_stat
 - qm_rtc_reg_t, 307
- rx
 - qm_i2c_transfer_t, 280
 - qm_spi_async_transfer_t, 318
 - qm_spi_transfer_t, 324
 - qm_ss_i2c_transfer_t, 333
 - qm_ss_spi_async_transfer_t, 334
 - qm_ss_spi_transfer_t, 337
- rx_len
 - qm_i2c_transfer_t, 280
 - qm_spi_async_transfer_t, 318
 - qm_spi_transfer_t, 324
 - qm_ss_i2c_transfer_t, 333
 - qm_ss_spi_async_transfer_t, 334
 - qm_ss_spi_transfer_t, 337
- rx_sample_dly
 - qm_spi_reg_t, 322
- rxflr
 - qm_spi_reg_t, 322
- rxftlr
 - qm_spi_reg_t, 323
- rxoicr
 - qm_spi_reg_t, 323
- rxuicr
 - qm_spi_reg_t, 323
- SOC_WATCH
 - SOCW_EVENT_APP, 173
 - SOCW_EVENT_FREQ, 173
 - SOCW_EVENT_HALT, 173
 - SOCW_EVENT_INTERRUPT, 173
 - SOCW_EVENT_MAX, 173
 - SOCW_EVENT_REGISTER, 173
 - SOCW_EVENT_SLEEP, 173
 - SOCW_REG_CCU_EXT_CLK_CTL, 173, 174
 - SOCW_REG_CCU_LP_CLK_CTL, 173, 174
 - SOCW_REG_CCU_PERIPH_CLK_GATE_CTL, 173, 174
 - SOCW_REG_CCU_SS_PERIPH_CLK_GATE_CTL, 173, 174
 - SOCW_REG_CCU_SYS_CLK_CTL, 173, 174
 - SOCW_REG_CMP_PWR, 173, 174
 - SOCW_REG_MAX, 173–175
 - SOCW_REG_OSC0_CFG1, 173, 174
 - SOCW_REG_PMUX_IN_EN, 173, 174
 - SOCW_REG_PMUX_IN_EN0, 174, 175
 - SOCW_REG_PMUX_IN_EN1, 174, 175
 - SOCW_REG_PMUX_IN_EN2, 174, 175
 - SOCW_REG_PMUX_IN_EN3, 174, 175
 - SOCW_REG_PMUX_PULLUP, 173, 174
 - SOCW_REG_PMUX_PULLUP0, 174
 - SOCW_REG_PMUX_PULLUP1, 174
 - SOCW_REG_PMUX_PULLUP2, 174
 - SOCW_REG_PMUX_PULLUP3, 174
 - SOCW_REG_PMUX_SLEW, 173, 174
 - SOCW_REG_PMUX_SLEW0, 174
 - SOCW_REG_PMUX_SLEW1, 174
 - SOCW_REG_PMUX_SLEW2, 174, 175
 - SOCW_REG_PMUX_SLEW3, 174, 175
 - SOCW_REG_SLP_CFG, 174
- SOCW_EVENT_APP
 - SOC_WATCH, 173
- SOCW_EVENT_FREQ
 - SOC_WATCH, 173
- SOCW_EVENT_HALT
 - SOC_WATCH, 173
- SOCW_EVENT_INTERRUPT
 - SOC_WATCH, 173
- SOCW_EVENT_MAX
 - SOC_WATCH, 173
- SOCW_EVENT_REGISTER
 - SOC_WATCH, 173
- SOCW_EVENT_SLEEP
 - SOC_WATCH, 173
- SOCW_REG_CCU_EXT_CLK_CTL
 - SOC_WATCH, 173, 174
- SOCW_REG_CCU_LP_CLK_CTL
 - SOC_WATCH, 173, 174
- SOCW_REG_CCU_PERIPH_CLK_GATE_CTL
 - SOC_WATCH, 173, 174
- SOCW_REG_CCU_SS_PERIPH_CLK_GATE_CTL
 - SOC_WATCH, 173, 174

- SOCW_REG_CCU_SYS_CLK_CTL
 - SOC_WATCH, [173](#), [174](#)
- SOCW_REG_CMP_PWR
 - SOC_WATCH, [173](#), [174](#)
- SOCW_REG_MAX
 - SOC_WATCH, [173](#)–[175](#)
- SOCW_REG_OSC0_CFG1
 - SOC_WATCH, [173](#), [174](#)
- SOCW_REG_PMUX_IN_EN
 - SOC_WATCH, [173](#), [174](#)
- SOCW_REG_PMUX_IN_EN0
 - SOC_WATCH, [174](#), [175](#)
- SOCW_REG_PMUX_IN_EN1
 - SOC_WATCH, [174](#), [175](#)
- SOCW_REG_PMUX_IN_EN2
 - SOC_WATCH, [174](#), [175](#)
- SOCW_REG_PMUX_IN_EN3
 - SOC_WATCH, [174](#), [175](#)
- SOCW_REG_PMUX_PULLUP
 - SOC_WATCH, [173](#), [174](#)
- SOCW_REG_PMUX_PULLUP0
 - SOC_WATCH, [174](#)
- SOCW_REG_PMUX_PULLUP1
 - SOC_WATCH, [174](#)
- SOCW_REG_PMUX_PULLUP2
 - SOC_WATCH, [174](#)
- SOCW_REG_PMUX_PULLUP3
 - SOC_WATCH, [174](#)
- SOCW_REG_PMUX_SLEW
 - SOC_WATCH, [173](#), [174](#)
- SOCW_REG_PMUX_SLEW0
 - SOC_WATCH, [174](#)
- SOCW_REG_PMUX_SLEW1
 - SOC_WATCH, [174](#)
- SOCW_REG_PMUX_SLEW2
 - SOC_WATCH, [174](#), [175](#)
- SOCW_REG_PMUX_SLEW3
 - SOC_WATCH, [174](#), [175](#)
- SOCW_REG_SLP_CFG
 - SOC_WATCH, [174](#)
- SPI
 - QM_SPI_BMODE_0, [103](#)
 - QM_SPI_BMODE_1, [103](#)
 - QM_SPI_BMODE_2, [103](#)
 - QM_SPI_BMODE_3, [103](#)
 - QM_SPI_BUSY, [105](#)
 - QM_SPI_FRAME_FORMAT_STANDARD, [103](#)
 - QM_SPI_FRAME_SIZE_10_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_11_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_12_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_13_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_14_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_15_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_16_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_17_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_18_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_19_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_20_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_21_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_22_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_23_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_24_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_25_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_26_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_27_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_28_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_29_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_30_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_31_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_32_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_4_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_5_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_6_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_7_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_8_BIT, [104](#)
 - QM_SPI_FRAME_SIZE_9_BIT, [104](#)
 - QM_SPI_IDLE, [105](#)
 - QM_SPI_RX_FULL, [105](#)
 - QM_SPI_RX_OVERFLOW, [105](#)
 - QM_SPI_SS_0, [104](#)
 - QM_SPI_SS_1, [104](#)
 - QM_SPI_SS_2, [104](#)
 - QM_SPI_SS_3, [104](#)
 - QM_SPI_SS_DISABLED, [104](#)
 - QM_SPI_TMOD_EEPROM_READ, [105](#)
 - QM_SPI_TMOD_RX, [105](#)
 - QM_SPI_TMOD_TX, [105](#)
 - QM_SPI_TMOD_TX_RX, [105](#)
 - QM_SPI_TX_EMPTY, [105](#)
- SS ADC
 - QM_SS_ADC_CAL_COMPLETE, [113](#)
 - QM_SS_ADC_CH_0, [113](#)
 - QM_SS_ADC_CH_1, [113](#)
 - QM_SS_ADC_CH_10, [114](#)
 - QM_SS_ADC_CH_11, [114](#)
 - QM_SS_ADC_CH_12, [114](#)
 - QM_SS_ADC_CH_13, [114](#)
 - QM_SS_ADC_CH_14, [114](#)
 - QM_SS_ADC_CH_15, [114](#)
 - QM_SS_ADC_CH_16, [114](#)
 - QM_SS_ADC_CH_17, [114](#)
 - QM_SS_ADC_CH_18, [114](#)
 - QM_SS_ADC_CH_2, [113](#)
 - QM_SS_ADC_CH_3, [113](#)
 - QM_SS_ADC_CH_4, [113](#)
 - QM_SS_ADC_CH_5, [113](#)
 - QM_SS_ADC_CH_6, [113](#)
 - QM_SS_ADC_CH_7, [113](#)
 - QM_SS_ADC_CH_8, [113](#)
 - QM_SS_ADC_CH_9, [114](#)
 - QM_SS_ADC_COMPLETE, [114](#)
 - QM_SS_ADC_IDLE, [114](#)
 - QM_SS_ADC_MODE_CHANGED, [113](#)
 - QM_SS_ADC_MODE_DEEP_PWR_DOWN, [114](#)
 - QM_SS_ADC_MODE_NORM_CAL, [114](#)
 - QM_SS_ADC_MODE_NORM_NO_CAL, [114](#)

- QM_SS_ADC_MODE_PWR_DOWN, [114](#)
- QM_SS_ADC_MODE_STDBY, [114](#)
- QM_SS_ADC_OVERFLOW, [114](#)
- QM_SS_ADC_RES_10_BITS, [114](#)
- QM_SS_ADC_RES_12_BITS, [114](#)
- QM_SS_ADC_RES_6_BITS, [114](#)
- QM_SS_ADC_RES_8_BITS, [114](#)
- QM_SS_ADC_SEQERROR, [114](#)
- QM_SS_ADC_TRANSFER, [113](#)
- QM_SS_ADC_UNDERFLOW, [114](#)
- SS Clock
 - SS_CLK_PERIPH_ADC, [177](#)
 - SS_CLK_PERIPH_GPIO_0, [177](#)
 - SS_CLK_PERIPH_GPIO_1, [177](#)
 - SS_CLK_PERIPH_I2C_0, [177](#)
 - SS_CLK_PERIPH_I2C_1, [177](#)
 - SS_CLK_PERIPH_SPI_0, [177](#)
 - SS_CLK_PERIPH_SPI_1, [177](#)
- SS GPIO
 - QM_SS_GPIO_HIGH, [121](#)
 - QM_SS_GPIO_LOW, [121](#)
 - QM_SS_GPIO_STATE_NUM, [121](#)
- SS I2C
 - QM_SS_I2C_10_BIT, [127](#)
 - QM_SS_I2C_7_BIT, [127](#)
 - QM_SS_I2C_BUSY, [127](#)
 - QM_SS_I2C_IDLE, [127](#)
 - QM_SS_I2C_RX_OVER, [128](#)
 - QM_SS_I2C_RX_UNDER, [128](#)
 - QM_SS_I2C_SPEED_FAST, [127](#)
 - QM_SS_I2C_SPEED_FAST_PLUS, [127](#)
 - QM_SS_I2C_SPEED_STD, [127](#)
 - QM_SS_I2C_TX_ABORT, [128](#)
 - QM_SS_I2C_TX_ABORT_10ADDR1_NOACK, [127](#)
 - QM_SS_I2C_TX_ABORT_10ADDR2_NOACK, [127](#)
 - QM_SS_I2C_TX_ABORT_10B_RD_NORSTRT, [127](#)
 - QM_SS_I2C_TX_ABORT_7B_ADDR_NOACK, [127](#)
 - QM_SS_I2C_TX_ABORT_GCALL_NOACK, [127](#)
 - QM_SS_I2C_TX_ABORT_MASTER_DIS, [127](#)
 - QM_SS_I2C_TX_ABORT_NORSTRT, [127](#)
 - QM_SS_I2C_TX_ABORT_SBYTE_ACKDET, [127](#)
 - QM_SS_I2C_TX_ABORT_SLV_ARBLOST, [127](#)
 - QM_SS_I2C_TX_ABORT_SLVRD_INTX, [127](#)
 - QM_SS_I2C_TX_ABORT_TXDATA_NOACK, [127](#)
 - QM_SS_I2C_TX_ABORT_USER_ABORT, [127](#)
 - QM_SS_I2C_TX_ARB_LOST, [127](#)
 - QM_SS_I2C_TX_OVER, [128](#)
- SS Power states
 - QM_SS_POWER_CPU_SS1_TIMER_OFF, [237](#)
 - QM_SS_POWER_CPU_SS1_TIMER_ON, [237](#)
- SS SPI
 - QM_SS_SPI_BMODE_0, [141](#)
 - QM_SS_SPI_BMODE_1, [141](#)
 - QM_SS_SPI_BMODE_2, [141](#)
 - QM_SS_SPI_BMODE_3, [141](#)
 - QM_SS_SPI_BUSY, [142](#)
 - QM_SS_SPI_FRAME_SIZE_10_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_11_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_12_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_13_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_14_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_15_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_16_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_17_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_18_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_19_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_20_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_21_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_22_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_23_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_24_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_25_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_26_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_27_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_28_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_29_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_30_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_31_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_32_BIT, [142](#)
 - QM_SS_SPI_FRAME_SIZE_4_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_5_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_6_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_7_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_8_BIT, [141](#)
 - QM_SS_SPI_FRAME_SIZE_9_BIT, [141](#)
 - QM_SS_SPI_IDLE, [142](#)
 - QM_SS_SPI_RX_OVERFLOW, [142](#)
 - QM_SS_SPI_RX_UNDERFLOW, [142](#)
 - QM_SS_SPI_SS_0, [142](#)
 - QM_SS_SPI_SS_1, [142](#)
 - QM_SS_SPI_SS_2, [142](#)
 - QM_SS_SPI_SS_3, [142](#)
 - QM_SS_SPI_SS_DISABLED, [142](#)
 - QM_SS_SPI_TMOD_EEPROM_READ, [143](#)
 - QM_SS_SPI_TMOD_RX, [143](#)
 - QM_SS_SPI_TMOD_TX, [143](#)
 - QM_SS_SPI_TMOD_TX_RX, [143](#)
 - QM_SS_SPI_TX_OVERFLOW, [142](#)
- SS_CLK_PERIPH_ADC
 - SS Clock, [177](#)
- SS_CLK_PERIPH_GPIO_0
 - SS Clock, [177](#)
- SS_CLK_PERIPH_GPIO_1
 - SS Clock, [177](#)
- SS_CLK_PERIPH_I2C_0
 - SS Clock, [177](#)
- SS_CLK_PERIPH_I2C_1
 - SS Clock, [177](#)
- SS_CLK_PERIPH_SPI_0
 - SS Clock, [177](#)
- SS_CLK_PERIPH_SPI_1
 - SS Clock, [177](#)
- SOC_WATCH, [172](#)
 - soc_watch_event_t, [173](#)
 - soc_watch_log_app_event, [175](#)

- soc_watch_log_event, 175
- soc_watch_reg_t, 173, 174
- soc_watch_trigger_flush, 175
- SPI, 102
 - qm_spi_bmode_t, 103
 - qm_spi_dma_channel_config, 105
 - qm_spi_dma_transfer, 106
 - qm_spi_dma_transfer_terminate, 106
 - qm_spi_frame_format_t, 103
 - qm_spi_frame_size_t, 103
 - qm_spi_get_status, 107
 - qm_spi_irq_transfer, 107
 - qm_spi_irq_transfer_terminate, 108
 - qm_spi_irq_update, 108
 - qm_spi_restore_context, 109
 - qm_spi_save_context, 109
 - qm_spi_set_config, 110
 - qm_spi_slave_select, 110
 - qm_spi_slave_select_t, 104
 - qm_spi_status_t, 104
 - qm_spi_tmode_t, 105
 - qm_spi_transfer, 110
- SS ADC, 112
 - qm_ss_adc_calibrate, 115
 - qm_ss_adc_cb_source_t, 113
 - qm_ss_adc_channel_t, 113
 - qm_ss_adc_convert, 115
 - qm_ss_adc_get_calibration, 115
 - qm_ss_adc_irq_calibrate, 116
 - qm_ss_adc_irq_convert, 116
 - qm_ss_adc_irq_set_mode, 116
 - qm_ss_adc_mode_t, 114
 - qm_ss_adc_resolution_t, 114
 - qm_ss_adc_restore_context, 118
 - qm_ss_adc_save_context, 118
 - qm_ss_adc_set_calibration, 119
 - qm_ss_adc_set_config, 119
 - qm_ss_adc_set_mode, 119
 - qm_ss_adc_status_t, 114
- SS Clock, 176
 - ss_clk_adc_disable, 177
 - ss_clk_adc_enable, 177
 - ss_clk_adc_set_div, 177
 - ss_clk_gpio_disable, 178
 - ss_clk_gpio_enable, 178
 - ss_clk_i2c_disable, 178
 - ss_clk_i2c_enable, 179
 - ss_clk_periph_t, 176
 - ss_clk_spi_disable, 179
 - ss_clk_spi_enable, 179
- SS GPIO, 121
 - qm_ss_gpio_clear_pin, 122
 - qm_ss_gpio_read_pin, 122
 - qm_ss_gpio_read_port, 122
 - qm_ss_gpio_restore_context, 123
 - qm_ss_gpio_save_context, 123
 - qm_ss_gpio_set_config, 123
 - qm_ss_gpio_set_pin, 124
 - qm_ss_gpio_set_pin_state, 124
 - qm_ss_gpio_state_t, 121
 - qm_ss_gpio_write_port, 125
- SS I2C, 126
 - qm_ss_i2c_addr_t, 127
 - qm_ss_i2c_get_status, 128
 - qm_ss_i2c_irq_transfer_terminate, 128
 - qm_ss_i2c_master_irq_transfer, 128
 - qm_ss_i2c_master_read, 129
 - qm_ss_i2c_master_write, 129
 - qm_ss_i2c_restore_context, 130
 - qm_ss_i2c_save_context, 130
 - qm_ss_i2c_set_config, 131
 - qm_ss_i2c_set_speed, 131
 - qm_ss_i2c_speed_t, 127
 - qm_ss_i2c_status_t, 127
- SS ISR, 134
 - QM_ISR_DECLARE, 135–139
- SS Interrupt, 132
 - qm_ss_int_vector_request, 132
 - qm_ss_irq_mask, 132
 - qm_ss_irq_request, 132
 - qm_ss_irq_unmask, 133
- SS Power states, 237
 - qm_ss_power_cpu_ss1, 238
 - qm_ss_power_cpu_ss1_mode_t, 237
 - qm_ss_power_cpu_ss2, 238
 - qm_ss_power_sleep_wait, 238
 - qm_ss_power_soc_deep_sleep_restore, 238
 - qm_ss_power_soc_lpss_disable, 239
 - qm_ss_power_soc_lpss_enable, 239
 - qm_ss_power_soc_sleep_restore, 239
- SS SPI, 140
 - qm_ss_spi_bmode_t, 141
 - qm_ss_spi_frame_size_t, 141
 - qm_ss_spi_get_status, 143
 - qm_ss_spi_irq_transfer, 143
 - qm_ss_spi_irq_transfer_terminate, 144
 - qm_ss_spi_restore_context, 144
 - qm_ss_spi_save_context, 144
 - qm_ss_spi_set_config, 145
 - qm_ss_spi_slave_select, 145
 - qm_ss_spi_slave_select_t, 142
 - qm_ss_spi_status_t, 142
 - qm_ss_spi_tmode_t, 142
 - qm_ss_spi_transfer, 146
- SS Timer, 147
 - qm_ss_timer_get, 147
 - qm_ss_timer_set, 147
 - qm_ss_timer_set_config, 148
- samples
 - qm_adc_xfer_t, 248
 - qm_ss_adc_xfer_t, 327
- samples_len
 - qm_adc_xfer_t, 248
 - qm_ss_adc_xfer_t, 327
- sar
 - qm_i2c_context_t, 272

- scr
 - qm_uart_context_t, [341](#)
 - qm_uart_reg_t, [343](#)
- sensor_activation
 - Quark SE Sensor Subsystem Initialisation, [236](#)
- ser
 - qm_spi_context_t, [319](#)
 - qm_spi_reg_t, [323](#)
- sivr
 - qm_mvic_reg_t, [298](#)
- slave_addr
 - qm_i2c_config_t, [270](#)
- SoC Registers (D2000)
 - CLK_PERIPH_ADC, [209](#)
 - CLK_PERIPH_ADC_REGISTER, [209](#)
 - CLK_PERIPH_ALL, [209](#)
 - CLK_PERIPH_CLK, [208](#), [209](#)
 - CLK_PERIPH_GPIO_DB, [208](#), [209](#)
 - CLK_PERIPH_GPIO_INTERRUPT, [208](#), [209](#)
 - CLK_PERIPH_GPIO_REGISTER, [208](#), [209](#)
 - CLK_PERIPH_I2C_M0, [208](#), [209](#)
 - CLK_PERIPH_I2C_M0_REGISTER, [209](#)
 - CLK_PERIPH_I2C_M1, [209](#)
 - CLK_PERIPH_I2C_M1_REGISTER, [209](#)
 - CLK_PERIPH_I2S, [209](#)
 - CLK_PERIPH_I2S_REGISTER, [209](#)
 - CLK_PERIPH_PWM_REGISTER, [208](#), [209](#)
 - CLK_PERIPH_REGISTER, [208](#), [209](#)
 - CLK_PERIPH_RTC_REGISTER, [208](#), [209](#)
 - CLK_PERIPH_SPI_M0, [208](#), [209](#)
 - CLK_PERIPH_SPI_M0_REGISTER, [208](#), [209](#)
 - CLK_PERIPH_SPI_M1, [209](#)
 - CLK_PERIPH_SPI_M1_REGISTER, [209](#)
 - CLK_PERIPH_SPI_S, [208](#), [209](#)
 - CLK_PERIPH_SPI_S_REGISTER, [208](#), [209](#)
 - CLK_PERIPH_UARTA_REGISTER, [209](#)
 - CLK_PERIPH_UARTB_REGISTER, [209](#)
 - CLK_PERIPH_WDT_REGISTER, [208](#), [209](#)
 - DMA_HW_IF_I2C_MASTER_0_RX, [210](#)
 - DMA_HW_IF_I2C_MASTER_0_TX, [210](#)
 - DMA_HW_IF_I2C_MASTER_1_RX, [210](#)
 - DMA_HW_IF_I2C_MASTER_1_TX, [210](#)
 - DMA_HW_IF_I2S_CAPTURE, [210](#)
 - DMA_HW_IF_I2S_PLAYBACK, [210](#)
 - DMA_HW_IF_SPI_MASTER_0_RX, [210](#)
 - DMA_HW_IF_SPI_MASTER_0_TX, [210](#)
 - DMA_HW_IF_SPI_MASTER_1_RX, [210](#)
 - DMA_HW_IF_SPI_MASTER_1_TX, [210](#)
 - DMA_HW_IF_SPI_SLAVE_RX, [210](#)
 - DMA_HW_IF_SPI_SLAVE_TX, [210](#)
 - DMA_HW_IF_UART_A_RX, [210](#)
 - DMA_HW_IF_UART_A_TX, [210](#)
 - DMA_HW_IF_UART_B_RX, [210](#)
 - DMA_HW_IF_UART_B_TX, [210](#)
 - QM_DMA_0, [211](#)
 - QM_DMA_CHANNEL_0, [210](#)
 - QM_DMA_CHANNEL_1, [210](#)
 - QM_DMA_CHANNEL_2, [210](#)
 - QM_DMA_CHANNEL_3, [210](#)
 - QM_DMA_CHANNEL_4, [210](#)
 - QM_DMA_CHANNEL_5, [210](#)
 - QM_DMA_CHANNEL_6, [210](#)
 - QM_DMA_CHANNEL_7, [210](#)
 - QM_DMA_CHANNEL_NUM, [210](#)
 - QM_DMA_NUM, [211](#)
 - QM_FPR_0, [211](#)
 - QM_FPR_1, [211](#)
 - QM_FPR_2, [211](#)
 - QM_FPR_3, [211](#)
 - QM_MPR_0, [211](#)
 - QM_MPR_1, [211](#)
 - QM_MPR_2, [211](#), [212](#)
 - QM_MPR_3, [211](#), [212](#)
 - QM_MPR_NUM, [211](#), [212](#)
- SoC Registers (SE)
 - CLK_PERIPH_ADC, [231](#)
 - CLK_PERIPH_ADC_REGISTER, [231](#)
 - CLK_PERIPH_ALL, [231](#), [232](#)
 - CLK_PERIPH_CLK, [231](#)
 - CLK_PERIPH_GPIO_DB, [231](#), [232](#)
 - CLK_PERIPH_GPIO_INTERRUPT, [231](#), [232](#)
 - CLK_PERIPH_GPIO_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_I2C_M0, [231](#)
 - CLK_PERIPH_I2C_M0_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_I2C_M1, [231](#)
 - CLK_PERIPH_I2C_M1_REGISTER, [232](#)
 - CLK_PERIPH_I2S, [232](#)
 - CLK_PERIPH_I2S_REGISTER, [232](#)
 - CLK_PERIPH_PWM_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_REGISTER, [231](#)
 - CLK_PERIPH_RTC_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_SPI_M0, [231](#)
 - CLK_PERIPH_SPI_M0_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_SPI_M1, [231](#)
 - CLK_PERIPH_SPI_M1_REGISTER, [232](#)
 - CLK_PERIPH_SPI_S, [231](#)
 - CLK_PERIPH_SPI_S_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_UARTA_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_UARTB_REGISTER, [231](#), [232](#)
 - CLK_PERIPH_WDT_REGISTER, [231](#), [232](#)
 - DMA_HW_IF_I2C_MASTER_0_RX, [233](#)
 - DMA_HW_IF_I2C_MASTER_0_TX, [233](#)
 - DMA_HW_IF_I2C_MASTER_1_RX, [233](#)
 - DMA_HW_IF_I2C_MASTER_1_TX, [233](#)
 - DMA_HW_IF_I2S_CAPTURE, [233](#)
 - DMA_HW_IF_I2S_PLAYBACK, [233](#)
 - DMA_HW_IF_SPI_MASTER_0_RX, [233](#)
 - DMA_HW_IF_SPI_MASTER_0_TX, [233](#)
 - DMA_HW_IF_SPI_MASTER_1_RX, [233](#)
 - DMA_HW_IF_SPI_MASTER_1_TX, [233](#)
 - DMA_HW_IF_SPI_SLAVE_RX, [233](#)
 - DMA_HW_IF_SPI_SLAVE_TX, [233](#)
 - DMA_HW_IF_UART_A_RX, [233](#)
 - DMA_HW_IF_UART_A_TX, [233](#)
 - DMA_HW_IF_UART_B_RX, [233](#)
 - DMA_HW_IF_UART_B_TX, [233](#)

- QM_DMA_0, [233](#)
- QM_DMA_CHANNEL_0, [232](#)
- QM_DMA_CHANNEL_1, [232](#)
- QM_DMA_CHANNEL_2, [232](#)
- QM_DMA_CHANNEL_3, [232](#)
- QM_DMA_CHANNEL_4, [232](#)
- QM_DMA_CHANNEL_5, [232](#)
- QM_DMA_CHANNEL_6, [232](#)
- QM_DMA_CHANNEL_7, [232](#)
- QM_DMA_CHANNEL_NUM, [232](#)
- QM_DMA_NUM, [233](#)
- QM_FPR_0, [234](#)
- QM_FPR_1, [234](#)
- QM_FPR_2, [234](#)
- QM_FPR_3, [234](#)
- QM_MPR_0, [234](#)
- QM_MPR_1, [234](#)
- QM_MPR_2, [234](#)
- QM_MPR_3, [234](#)
- QM_MPR_NUM, [234](#)
- SoC Registers (Sensor Subsystem)
 - QM_SS_ADC_0, [222](#)
 - QM_SS_ADC_CTRL, [222](#)
 - QM_SS_ADC_DIVSEQSTAT, [222](#)
 - QM_SS_ADC_INTSTAT, [222](#)
 - QM_SS_ADC_SAMPLE, [222](#)
 - QM_SS_ADC_SEQ, [222](#)
 - QM_SS_ADC_SET, [222](#)
 - QM_SS_IO_CREG_MST0_CTRL, [223](#)
 - QM_SS_IO_CREG_SLV0_OBSR, [223](#)
 - QM_SS_IO_CREG_SLV1_OBSR, [223](#)
 - QM_SS_SPI_0, [223](#)
 - QM_SS_SPI_1, [223](#)
 - QM_SS_SPI_CLR_INTR, [223](#)
 - QM_SS_SPI_CTRL, [223](#)
 - QM_SS_SPI_DR, [223](#)
 - QM_SS_SPI_FTLR, [223](#)
 - QM_SS_SPI_INTR_MASK, [223](#)
 - QM_SS_SPI_INTR_STAT, [223](#)
 - QM_SS_SPI_RXFLR, [223](#)
 - QM_SS_SPI_SPIEN, [223](#)
 - QM_SS_SPI_SR, [223](#)
 - QM_SS_SPI_TIMING, [223](#)
 - QM_SS_SPI_TXFLR, [223](#)
- SoC Interrupt Router (D2000), [201](#)
- SoC Interrupt Router (SE), [219](#)
- SoC Interrupts (D2000), [202](#)
- SoC Interrupts (SE), [224](#)
- SoC Registers (D2000), [203](#)
 - clk_periph_t, [208](#)
 - qm_adc_t, [209](#)
 - qm_aonc_t, [209](#)
 - qm_dma_channel_id_t, [209](#)
 - qm_dma_handshake_interface_t, [210](#)
 - qm_dma_t, [210](#)
 - qm_flash_t, [211](#)
 - qm_fpr_id_t, [211](#)
 - qm_gpio_t, [211](#)
 - qm_i2c, [212](#)
 - qm_i2c_t, [211](#)
 - qm_mpr_id_t, [211](#)
 - qm_pwm_id_t, [212](#)
 - qm_pwm_t, [212](#)
 - qm_rtc_t, [212](#)
 - qm_spi_t, [212](#)
 - qm_uart_t, [212](#)
 - qm_wdt_t, [212](#)
- SoC Registers (SE), [225](#)
 - clk_periph_t, [231](#)
 - qm_aonc_t, [232](#)
 - qm_dma_channel_id_t, [232](#)
 - qm_dma_handshake_interface_t, [232](#)
 - qm_dma_t, [233](#)
 - qm_flash_t, [233](#)
 - qm_fpr_id_t, [233](#)
 - qm_gpio_t, [234](#)
 - qm_i2c, [235](#)
 - qm_i2c_t, [234](#)
 - qm_mpr_id_t, [234](#)
 - qm_pwm_id_t, [234](#)
 - qm_pwm_t, [234](#)
 - qm_rtc_t, [234](#)
 - qm_spi_t, [235](#)
 - qm_uart_t, [235](#)
 - qm_usb_t, [235](#)
 - qm_wdt_t, [235](#)
- SoC Registers (Sensor Subsystem), [221](#)
 - qm_ss_adc_reg_t, [222](#)
 - qm_ss_adc_t, [222](#)
 - qm_ss_creg_reg_t, [222](#)
 - qm_ss_gpio_reg_t, [223](#)
 - qm_ss_gpio_t, [223](#)
 - qm_ss_i2c_reg_t, [223](#)
 - qm_ss_spi_reg_t, [223](#)
 - qm_ss_spi_t, [223](#)
- soc_ctrl
 - qm_scss_ccu_reg_t, [309](#)
- soc_ctrl_lock
 - qm_scss_ccu_reg_t, [309](#)
- soc_watch_event_t
 - SOC_WATCH, [173](#)
- soc_watch_log_app_event
 - SOC_WATCH, [175](#)
- soc_watch_log_event
 - SOC_WATCH, [175](#)
- soc_watch_reg_t
 - SOC_WATCH, [173](#), [174](#)
- soc_watch_trigger_flush
 - SOC_WATCH, [175](#)
- source_address
 - qm_dma_multi_transfer_t, [257](#)
 - qm_dma_transfer_t, [258](#)
- speed
 - qm_i2c_config_t, [270](#)
 - qm_ss_i2c_config_t, [331](#)
- spi_ctrl

- qm_ss_spi_context_t, 336
- spi_master_0_int_mask
 - qm_interrupt_router_reg_t, 289
- spi_master_1_int_mask
 - qm_interrupt_router_reg_t, 289
- spi_slave_0_int_mask
 - qm_interrupt_router_reg_t, 289
- spi_spilen
 - qm_ss_spi_context_t, 336
- spi_timing
 - qm_ss_spi_context_t, 336
- sr
 - qm_spi_reg_t, 323
- sram_mpr_0_int_mask
 - qm_interrupt_router_reg_t, 289
- ss_adc_0_error_int_mask
 - qm_interrupt_router_reg_t, 289
- ss_adc_0_int_mask
 - qm_interrupt_router_reg_t, 289
- ss_clk_adc_disable
 - SS Clock, 177
- ss_clk_adc_enable
 - SS Clock, 177
- ss_clk_adc_set_div
 - SS Clock, 177
- ss_clk_gpio_disable
 - SS Clock, 178
- ss_clk_gpio_enable
 - SS Clock, 178
- ss_clk_i2c_disable
 - SS Clock, 178
- ss_clk_i2c_enable
 - SS Clock, 179
- ss_clk_periph_t
 - SS Clock, 176
- ss_clk_spi_disable
 - SS Clock, 179
- ss_clk_spi_enable
 - SS Clock, 179
- ss_gpio_0_int_mask
 - qm_interrupt_router_reg_t, 289
- ss_gpio_1_int_mask
 - qm_interrupt_router_reg_t, 289
- ss_i2c_0_int
 - qm_interrupt_router_reg_t, 290
- ss_i2c_1_int
 - qm_interrupt_router_reg_t, 290
- ss_scl_hcnt
 - qm_i2c_context_t, 272
- ss_scl_lcnt
 - qm_i2c_context_t, 272
- ss_spi_0_int
 - qm_interrupt_router_reg_t, 290
- ss_spi_1_int
 - qm_interrupt_router_reg_t, 290
- ssi_comp_version
 - qm_spi_reg_t, 323
- ssienr
 - qm_spi_reg_t, 323
- status32_irq_enable
 - qm_irq_context_t, 291
- status32_irq_threshold
 - qm_irq_context_t, 291
- stop
 - qm_i2c_transfer_t, 280
 - qm_ss_i2c_transfer_t, 333
- Syscalls, 243
 - pico_printf, 243
- tar
 - qm_uart_reg_t, 344
- tat
 - qm_uart_reg_t, 344
- tcr
 - qm_uart_reg_t, 344
- timeout
 - qm_wdt_config_t, 352
- timer
 - qm_pwm_reg_t, 304
- timer_0_int_mask
 - qm_interrupt_router_reg_t, 290
- timer_dcr
 - qm_pic_timer_context_t, 300
- timer_icr
 - qm_pic_timer_context_t, 300
- tmg_ctrl
 - qm_flash_context_t, 259
 - qm_flash_reg_t, 261
- tpr
 - qm_mvic_reg_t, 298
- tx
 - qm_i2c_transfer_t, 280
 - qm_spi_async_transfer_t, 318
 - qm_spi_transfer_t, 324
 - qm_ss_i2c_transfer_t, 333
 - qm_ss_spi_async_transfer_t, 334
 - qm_ss_spi_transfer_t, 337
- tx_len
 - qm_i2c_transfer_t, 280
 - qm_spi_async_transfer_t, 319
 - qm_spi_transfer_t, 325
 - qm_ss_i2c_transfer_t, 333
 - qm_ss_spi_async_transfer_t, 334
 - qm_ss_spi_transfer_t, 337
- tx_tl
 - qm_i2c_context_t, 272
- txflr
 - qm_spi_reg_t, 323
- txfltr
 - qm_spi_reg_t, 323
- txoicr
 - qm_spi_reg_t, 324
- type
 - qm_usb_ep_config_t, 346
- UART
 - QM_UART_IDLE, 151

- QM_UART_LC_5E1, 150
- QM_UART_LC_5E1_5, 150
- QM_UART_LC_5N1, 150
- QM_UART_LC_5N1_5, 150
- QM_UART_LC_5O1, 150
- QM_UART_LC_5O1_5, 150
- QM_UART_LC_6E1, 150
- QM_UART_LC_6E2, 150
- QM_UART_LC_6N1, 150
- QM_UART_LC_6N2, 150
- QM_UART_LC_6O1, 150
- QM_UART_LC_6O2, 150
- QM_UART_LC_7E1, 150
- QM_UART_LC_7E2, 150
- QM_UART_LC_7N1, 150
- QM_UART_LC_7N2, 150
- QM_UART_LC_7O1, 150
- QM_UART_LC_7O2, 151
- QM_UART_LC_8E1, 151
- QM_UART_LC_8E2, 151
- QM_UART_LC_8N1, 151
- QM_UART_LC_8N2, 151
- QM_UART_LC_8O1, 151
- QM_UART_LC_8O2, 151
- QM_UART_RX_BI, 151
- QM_UART_RX_BUSY, 151
- QM_UART_RX_FE, 151
- QM_UART_RX_NEMPTY, 151
- QM_UART_RX_OE, 151
- QM_UART_RX_PE, 151
- QM_UART_TX_BUSY, 151
- QM_UART_TX_NFULL, 151
- USB
 - QM_USB_CONFIGURED, 162
 - QM_USB_CONNECTED, 162
 - QM_USB_DISCONNECTED, 162
 - QM_USB_EP_BULK, 161
 - QM_USB_EP_CONTROL, 161
 - QM_USB_EP_DATA_IN, 161
 - QM_USB_EP_DATA_OUT, 161
 - QM_USB_EP_INTERRUPT, 161
 - QM_USB_EP_SETUP, 161
 - QM_USB_RESET, 162
 - QM_USB_RESUME, 162
 - QM_USB_SUSPEND, 162
- UART, 149
 - qm_uart_dma_channel_config, 151
 - qm_uart_dma_read, 152
 - qm_uart_dma_read_terminate, 152
 - qm_uart_dma_write, 153
 - qm_uart_dma_write_terminate, 153
 - qm_uart_get_status, 153
 - qm_uart_irq_read, 154
 - qm_uart_irq_read_terminate, 154
 - qm_uart_irq_write, 155
 - qm_uart_irq_write_terminate, 155
 - qm_uart_lc_t, 150
 - qm_uart_read, 155
 - qm_uart_read_non_block, 156
 - qm_uart_restore_context, 156
 - qm_uart_save_context, 157
 - qm_uart_set_config, 157
 - qm_uart_status_t, 151
 - qm_uart_write, 157
 - qm_uart_write_buffer, 158
 - qm_uart_write_non_block, 158
- USB, 160
 - qm_usb_attach, 162
 - qm_usb_detach, 162
 - qm_usb_ep_disable, 163
 - qm_usb_ep_enable, 163
 - qm_usb_ep_flush, 163
 - qm_usb_ep_get_bytes_read, 164
 - qm_usb_ep_halt, 164
 - qm_usb_ep_is_stalled, 164
 - qm_usb_ep_read, 165
 - qm_usb_ep_set_config, 165
 - qm_usb_ep_set_stall_state, 166
 - qm_usb_ep_status_t, 161
 - qm_usb_ep_type_t, 161
 - qm_usb_ep_write, 166
 - qm_usb_reset, 166
 - qm_usb_set_address, 167
 - qm_usb_set_status_callback, 167
 - qm_usb_status_callback_t, 161
 - qm_usb_status_t, 161
- uart_0_int_mask
 - qm_interrupt_router_reg_t, 290
- uart_1_int_mask
 - qm_interrupt_router_reg_t, 290
- up_bound
 - qm_fpr_config_t, 262
- us_count
 - qm_flash_config_t, 258
- usb_0_int_mask
 - qm_interrupt_router_reg_t, 290
- usb_pll_cfg0
 - qm_scss_ccu_reg_t, 309
- usr
 - qm_uart_reg_t, 344
- Version, 168
 - qm_ver_rom, 168
- vreg_aon_set_mode
 - Quark SE Voltage Regulators, 241
- vreg_host_set_mode
 - Quark SE Voltage Regulators, 241
- vreg_plat1p8_set_mode
 - Quark SE Voltage Regulators, 242
- vreg_plat3p3_set_mode
 - Quark SE Voltage Regulators, 242
- WDT
 - QM_WDT_MODE_INTERRUPT_RESET, 169
 - QM_WDT_MODE_RESET, 169
- WDT, 169
 - qm_wdt_mode_t, 169

- qm_wdt_reload, [169](#)
- qm_wdt_restore_context, [170](#)
- qm_wdt_save_context, [170](#)
- qm_wdt_set_config, [170](#)
- qm_wdt_start, [171](#)
- wait_states
 - qm_flash_config_t, [258](#)
- wake_mask
 - qm_scss_ccu_reg_t, [309](#)
- watchdog_mode
 - qm_ss_timer_config_t, [338](#)
- wdt_0_int_mask
 - qm_interrupt_router_reg_t, [290](#)
- wdt_ccvr
 - qm_wdt_reg_t, [353](#)
- wdt_comp_param_1
 - qm_wdt_reg_t, [353](#)
- wdt_comp_param_2
 - qm_wdt_reg_t, [353](#)
- wdt_comp_param_3
 - qm_wdt_reg_t, [353](#)
- wdt_comp_param_4
 - qm_wdt_reg_t, [353](#)
- wdt_comp_param_5
 - qm_wdt_reg_t, [353](#)
- wdt_comp_type
 - qm_wdt_reg_t, [353](#)
- wdt_comp_version
 - qm_wdt_reg_t, [353](#)
- wdt_cr
 - qm_wdt_reg_t, [353](#)
- wdt_crr
 - qm_wdt_reg_t, [353](#)
- wdt_eoi
 - qm_wdt_reg_t, [353](#)
- wdt_stat
 - qm_wdt_reg_t, [353](#)
- wdt_torr
 - qm_wdt_reg_t, [354](#)
- wo_sp
 - qm_scss_gp_reg_t, [312](#)
- wo_st
 - qm_scss_gp_reg_t, [312](#)
- write_disable
 - qm_flash_config_t, [259](#)