



## Intel® Quark™ Microcontroller Software Interface 1.1

---

July, 2016

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Licensing information</b>             | <b>1</b> |
| <b>2</b> | <b>Module Index</b>                      | <b>2</b> |
| 2.1      | Modules . . . . .                        | 2        |
| <b>3</b> | <b>Data Structure Index</b>              | <b>3</b> |
| 3.1      | Data Structures . . . . .                | 3        |
| <b>4</b> | <b>Module Documentation</b>              | <b>7</b> |
| 4.1      | Error Codes . . . . .                    | 7        |
| 4.2      | Clock Management . . . . .               | 8        |
| 4.2.1    | Detailed Description . . . . .           | 9        |
| 4.2.2    | Enumeration Type Documentation . . . . . | 9        |
| 4.2.3    | Function Documentation . . . . .         | 11       |
| 4.3      | Quark D2000 ADC . . . . .                | 16       |
| 4.3.1    | Detailed Description . . . . .           | 17       |
| 4.3.2    | Enumeration Type Documentation . . . . . | 17       |
| 4.3.3    | Function Documentation . . . . .         | 18       |
| 4.4      | Always-on Counters . . . . .             | 23       |
| 4.4.1    | Detailed Description . . . . .           | 23       |
| 4.4.2    | Function Documentation . . . . .         | 23       |
| 4.5      | Analog Comparator . . . . .              | 27       |
| 4.5.1    | Detailed Description . . . . .           | 27       |
| 4.5.2    | Function Documentation . . . . .         | 27       |
| 4.6      | DMA . . . . .                            | 28       |
| 4.6.1    | Detailed Description . . . . .           | 28       |
| 4.6.2    | Enumeration Type Documentation . . . . . | 29       |
| 4.6.3    | Function Documentation . . . . .         | 30       |
| 4.7      | Flash . . . . .                          | 34       |
| 4.7.1    | Detailed Description . . . . .           | 34       |
| 4.7.2    | Enumeration Type Documentation . . . . . | 34       |
| 4.7.3    | Function Documentation . . . . .         | 35       |
| 4.8      | FPR . . . . .                            | 39       |
| 4.8.1    | Detailed Description . . . . .           | 39       |
| 4.8.2    | Enumeration Type Documentation . . . . . | 39       |
| 4.8.3    | Function Documentation . . . . .         | 40       |
| 4.9      | GPIO . . . . .                           | 42       |
| 4.9.1    | Detailed Description . . . . .           | 42       |
| 4.9.2    | Enumeration Type Documentation . . . . . | 42       |

|        |                                |    |
|--------|--------------------------------|----|
| 4.9.3  | Function Documentation         | 42 |
| 4.10   | I2C                            | 46 |
| 4.10.1 | Detailed Description           | 47 |
| 4.10.2 | Enumeration Type Documentation | 47 |
| 4.10.3 | Function Documentation         | 48 |
| 4.11   | Identification                 | 55 |
| 4.11.1 | Detailed Description           | 55 |
| 4.11.2 | Function Documentation         | 55 |
| 4.12   | Initialisation                 | 56 |
| 4.12.1 | Detailed Description           | 56 |
| 4.12.2 | Enumeration Type Documentation | 56 |
| 4.12.3 | Function Documentation         | 56 |
| 4.13   | Interrupt                      | 57 |
| 4.13.1 | Detailed Description           | 57 |
| 4.13.2 | Function Documentation         | 57 |
| 4.14   | ISR                            | 59 |
| 4.14.1 | Detailed Description           | 60 |
| 4.14.2 | Function Documentation         | 60 |
| 4.15   | Mailbox                        | 67 |
| 4.15.1 | Detailed Description           | 67 |
| 4.15.2 | Typedef Documentation          | 67 |
| 4.15.3 | Enumeration Type Documentation | 68 |
| 4.15.4 | Function Documentation         | 69 |
| 4.16   | MPR                            | 72 |
| 4.16.1 | Detailed Description           | 72 |
| 4.16.2 | Function Documentation         | 72 |
| 4.17   | PIC Timer                      | 73 |
| 4.17.1 | Detailed Description           | 73 |
| 4.17.2 | Enumeration Type Documentation | 73 |
| 4.17.3 | Function Documentation         | 73 |
| 4.18   | Pin Muxing setup               | 75 |
| 4.18.1 | Detailed Description           | 75 |
| 4.18.2 | Enumeration Type Documentation | 75 |
| 4.18.3 | Function Documentation         | 78 |
| 4.19   | PWM / Timer                    | 80 |
| 4.19.1 | Detailed Description           | 80 |
| 4.19.2 | Enumeration Type Documentation | 80 |
| 4.19.3 | Function Documentation         | 80 |
| 4.20   | RTC                            | 83 |
| 4.20.1 | Detailed Description           | 83 |

|        |                                |     |
|--------|--------------------------------|-----|
| 4.20.2 | Function Documentation         | 83  |
| 4.21   | SPI                            | 86  |
| 4.21.1 | Detailed Description           | 87  |
| 4.21.2 | Enumeration Type Documentation | 87  |
| 4.21.3 | Function Documentation         | 89  |
| 4.22   | UART                           | 94  |
| 4.22.1 | Detailed Description           | 95  |
| 4.22.2 | Enumeration Type Documentation | 95  |
| 4.22.3 | Function Documentation         | 96  |
| 4.23   | Version                        | 104 |
| 4.23.1 | Detailed Description           | 104 |
| 4.23.2 | Function Documentation         | 104 |
| 4.24   | WDT                            | 105 |
| 4.24.1 | Detailed Description           | 105 |
| 4.24.2 | Enumeration Type Documentation | 105 |
| 4.24.3 | Function Documentation         | 106 |
| 4.25   | SOC_WATCH                      | 108 |
| 4.25.1 | Detailed Description           | 108 |
| 4.25.2 | Enumeration Type Documentation | 109 |
| 4.25.3 | Function Documentation         | 111 |
| 4.26   | SS ADC                         | 112 |
| 4.26.1 | Detailed Description           | 113 |
| 4.26.2 | Enumeration Type Documentation | 113 |
| 4.26.3 | Function Documentation         | 114 |
| 4.27   | SS GPIO                        | 119 |
| 4.27.1 | Detailed Description           | 119 |
| 4.27.2 | Enumeration Type Documentation | 119 |
| 4.27.3 | Function Documentation         | 119 |
| 4.28   | SS I2C                         | 125 |
| 4.28.1 | Detailed Description           | 125 |
| 4.28.2 | Enumeration Type Documentation | 126 |
| 4.28.3 | Function Documentation         | 126 |
| 4.29   | SS Interrupt                   | 130 |
| 4.29.1 | Detailed Description           | 130 |
| 4.29.2 | Function Documentation         | 130 |
| 4.30   | SS ISR                         | 132 |
| 4.30.1 | Detailed Description           | 132 |
| 4.30.2 | Function Documentation         | 132 |
| 4.31   | SS SPI                         | 136 |
| 4.31.1 | Detailed Description           | 137 |

|        |                                  |     |
|--------|----------------------------------|-----|
| 4.31.2 | Enumeration Type Documentation   | 137 |
| 4.31.3 | Function Documentation           | 138 |
| 4.32   | SS Timer                         | 142 |
| 4.32.1 | Detailed Description             | 142 |
| 4.32.2 | Function Documentation           | 142 |
| 4.33   | SS Clock                         | 144 |
| 4.33.1 | Detailed Description             | 144 |
| 4.33.2 | Enumeration Type Documentation   | 144 |
| 4.33.3 | Function Documentation           | 145 |
| 4.34   | SS Power states                  | 149 |
| 4.34.1 | Detailed Description             | 149 |
| 4.34.2 | Enumeration Type Documentation   | 149 |
| 4.34.3 | Function Documentation           | 149 |
| 4.35   | Quark D2000 Flash Layout         | 152 |
| 4.35.1 | Detailed Description             | 152 |
| 4.36   | Quark D2000 Power states         | 153 |
| 4.36.1 | Detailed Description             | 153 |
| 4.36.2 | Enumeration Type Documentation   | 153 |
| 4.36.3 | Function Documentation           | 153 |
| 4.37   | SoC Registers (D2000)            | 155 |
| 4.37.1 | Detailed Description             | 159 |
| 4.37.2 | Enumeration Type Documentation   | 159 |
| 4.37.3 | Variable Documentation           | 163 |
| 4.38   | RAR                              | 164 |
| 4.38.1 | Detailed Description             | 164 |
| 4.38.2 | Enumeration Type Documentation   | 164 |
| 4.38.3 | Function Documentation           | 164 |
| 4.39   | Quark SE Flash Layout            | 166 |
| 4.39.1 | Detailed Description             | 166 |
| 4.40   | Quark SE SoC Power states        | 167 |
| 4.40.1 | Detailed Description             | 167 |
| 4.40.2 | Function Documentation           | 167 |
| 4.41   | Quark SE Host Power states       | 169 |
| 4.41.1 | Detailed Description             | 169 |
| 4.41.2 | Function Documentation           | 169 |
| 4.42   | SoC Registers (Sensor Subsystem) | 171 |
| 4.42.1 | Detailed Description             | 172 |
| 4.42.2 | Enumeration Type Documentation   | 172 |
| 4.43   | SoC Registers (SE)               | 174 |
| 4.43.1 | Detailed Description             | 178 |

|          |  |            |
|----------|--|------------|
| 4.43.2   | Enumeration Type Documentation           | 178        |
| 4.43.3   | Variable Documentation                   | 182        |
| 4.44     | Quark SE Voltage Regulators              | 183        |
| 4.44.1   | Detailed Description                     | 183        |
| 4.44.2   | Function Documentation                   | 183        |
| 4.45     | Syscalls                                 | 185        |
| 4.45.1   | Detailed Description                     | 185        |
| 4.45.2   | Function Documentation                   | 185        |
| <b>5</b> | <b>Data Structure Documentation</b>      | <b>186</b> |
| 5.1      | int_ss_i2c_reg_t Struct Reference        | 186        |
| 5.1.1    | Detailed Description                     | 186        |
| 5.2      | int_ss_spi_reg_t Struct Reference        | 186        |
| 5.2.1    | Detailed Description                     | 186        |
| 5.3      | mvic_reg_pad_t Struct Reference          | 186        |
| 5.3.1    | Detailed Description                     | 186        |
| 5.4      | pic_timer_reg_pad_t Struct Reference     | 186        |
| 5.4.1    | Detailed Description                     | 186        |
| 5.5      | qm_ac_config_t Struct Reference          | 186        |
| 5.5.1    | Detailed Description                     | 187        |
| 5.5.2    | Field Documentation                      | 187        |
| 5.6      | qm_adc_config_t Struct Reference         | 188        |
| 5.6.1    | Detailed Description                     | 188        |
| 5.6.2    | Field Documentation                      | 188        |
| 5.7      | qm_adc_reg_t Struct Reference            | 188        |
| 5.7.1    | Detailed Description                     | 189        |
| 5.8      | qm_adc_xfer_t Struct Reference           | 189        |
| 5.8.1    | Detailed Description                     | 189        |
| 5.8.2    | Field Documentation                      | 189        |
| 5.9      | qm_aonpt_config_t Struct Reference       | 190        |
| 5.9.1    | Detailed Description                     | 191        |
| 5.9.2    | Field Documentation                      | 191        |
| 5.10     | qm_dma_chan_reg_t Struct Reference       | 191        |
| 5.10.1   | Detailed Description                     | 192        |
| 5.11     | qm_dma_channel_config_t Struct Reference | 192        |
| 5.11.1   | Detailed Description                     | 193        |
| 5.11.2   | Field Documentation                      | 193        |
| 5.12     | qm_dma_int_reg_t Struct Reference        | 193        |
| 5.12.1   | Detailed Description                     | 195        |
| 5.13     | qm_dma_misc_reg_t Struct Reference       | 195        |

|   |     |
|---|-----|
| 5.13.1 Detailed Description . . . . .                 | 196 |
| 5.14 qm_dma_transfer_t Struct Reference . . . . .     | 196 |
| 5.14.1 Detailed Description . . . . .                 | 196 |
| 5.14.2 Field Documentation . . . . .                  | 196 |
| 5.15 qm_flash_config_t Struct Reference . . . . .     | 197 |
| 5.15.1 Detailed Description . . . . .                 | 197 |
| 5.15.2 Field Documentation . . . . .                  | 197 |
| 5.16 qm_flash_reg_t Struct Reference . . . . .        | 198 |
| 5.16.1 Detailed Description . . . . .                 | 198 |
| 5.17 qm_fpr_config_t Struct Reference . . . . .       | 198 |
| 5.17.1 Detailed Description . . . . .                 | 199 |
| 5.17.2 Field Documentation . . . . .                  | 199 |
| 5.18 qm_gpio_port_config_t Struct Reference . . . . . | 199 |
| 5.18.1 Detailed Description . . . . .                 | 200 |
| 5.18.2 Field Documentation . . . . .                  | 200 |
| 5.19 qm_gpio_reg_t Struct Reference . . . . .         | 201 |
| 5.19.1 Detailed Description . . . . .                 | 202 |
| 5.19.2 Field Documentation . . . . .                  | 202 |
| 5.20 qm_i2c_config_t Struct Reference . . . . .       | 204 |
| 5.20.1 Detailed Description . . . . .                 | 204 |
| 5.20.2 Field Documentation . . . . .                  | 204 |
| 5.21 qm_i2c_reg_t Struct Reference . . . . .          | 205 |
| 5.21.1 Detailed Description . . . . .                 | 207 |
| 5.21.2 Field Documentation . . . . .                  | 207 |
| 5.22 qm_i2c_transfer_t Struct Reference . . . . .     | 211 |
| 5.22.1 Detailed Description . . . . .                 | 212 |
| 5.22.2 Field Documentation . . . . .                  | 212 |
| 5.23 qm_lapic_reg_t Struct Reference . . . . .        | 213 |
| 5.23.1 Detailed Description . . . . .                 | 214 |
| 5.24 qm_mailbox_t Struct Reference . . . . .          | 214 |
| 5.24.1 Detailed Description . . . . .                 | 214 |
| 5.25 qm_mbox_msg_t Struct Reference . . . . .         | 214 |
| 5.25.1 Detailed Description . . . . .                 | 215 |
| 5.25.2 Field Documentation . . . . .                  | 215 |
| 5.26 qm_mpr_config_t Struct Reference . . . . .       | 215 |
| 5.26.1 Detailed Description . . . . .                 | 215 |
| 5.27 qm_mpr_reg_t Struct Reference . . . . .          | 215 |
| 5.27.1 Detailed Description . . . . .                 | 216 |
| 5.28 qm_mvic_reg_t Struct Reference . . . . .         | 216 |
| 5.28.1 Detailed Description . . . . .                 | 216 |

|        |  |     |
|--------|--|-----|
| 5.28.2 | Field Documentation                    | 216 |
| 5.29   | qm_pic_timer_config_t Struct Reference | 217 |
| 5.29.1 | Detailed Description                   | 218 |
| 5.29.2 | Field Documentation                    | 218 |
| 5.30   | qm_pic_timer_reg_t Struct Reference    | 218 |
| 5.30.1 | Detailed Description                   | 219 |
| 5.31   | qm_pwm_channel_t Struct Reference      | 219 |
| 5.31.1 | Detailed Description                   | 219 |
| 5.31.2 | Field Documentation                    | 219 |
| 5.32   | qm_pwm_config_t Struct Reference       | 220 |
| 5.32.1 | Detailed Description                   | 220 |
| 5.32.2 | Field Documentation                    | 220 |
| 5.33   | qm_pwm_reg_t Struct Reference          | 221 |
| 5.33.1 | Detailed Description                   | 222 |
| 5.33.2 | Field Documentation                    | 222 |
| 5.34   | qm_rtc_config_t Struct Reference       | 222 |
| 5.34.1 | Detailed Description                   | 222 |
| 5.34.2 | Field Documentation                    | 222 |
| 5.35   | qm_rtc_reg_t Struct Reference          | 224 |
| 5.35.1 | Detailed Description                   | 224 |
| 5.35.2 | Field Documentation                    | 225 |
| 5.36   | qm_scss_aon_reg_t Struct Reference     | 225 |
| 5.36.1 | Detailed Description                   | 226 |
| 5.36.2 | Field Documentation                    | 226 |
| 5.37   | qm_scss_ccu_reg_t Struct Reference     | 226 |
| 5.37.1 | Detailed Description                   | 227 |
| 5.37.2 | Field Documentation                    | 227 |
| 5.38   | qm_scss_cmp_reg_t Struct Reference     | 229 |
| 5.38.1 | Detailed Description                   | 229 |
| 5.38.2 | Field Documentation                    | 230 |
| 5.39   | qm_scss_gp_reg_t Struct Reference      | 230 |
| 5.39.1 | Detailed Description                   | 231 |
| 5.39.2 | Field Documentation                    | 231 |
| 5.40   | qm_scss_info_reg_t Struct Reference    | 232 |
| 5.40.1 | Detailed Description                   | 232 |
| 5.40.2 | Field Documentation                    | 232 |
| 5.41   | qm_scss_int_reg_t Struct Reference     | 233 |
| 5.41.1 | Detailed Description                   | 234 |
| 5.41.2 | Field Documentation                    | 234 |
| 5.42   | qm_scss_mailbox_reg_t Struct Reference | 236 |



|  |     |
|--|-----|
| 5.42.1 Detailed Description . . . . .                    | 236 |
| 5.43 qm_scss_mem_reg_t Struct Reference . . . . .        | 236 |
| 5.43.1 Detailed Description . . . . .                    | 237 |
| 5.44 qm_scss_peripheral_reg_t Struct Reference . . . . . | 237 |
| 5.44.1 Detailed Description . . . . .                    | 237 |
| 5.44.2 Field Documentation . . . . .                     | 237 |
| 5.45 qm_scss_pmu_reg_t Struct Reference . . . . .        | 237 |
| 5.45.1 Detailed Description . . . . .                    | 238 |
| 5.45.2 Field Documentation . . . . .                     | 238 |
| 5.46 qm_scss_pmux_reg_t Struct Reference . . . . .       | 239 |
| 5.46.1 Detailed Description . . . . .                    | 239 |
| 5.46.2 Field Documentation . . . . .                     | 239 |
| 5.47 qm_scss_ss_reg_t Struct Reference . . . . .         | 240 |
| 5.47.1 Detailed Description . . . . .                    | 240 |
| 5.48 qm_spi_async_transfer_t Struct Reference . . . . .  | 241 |
| 5.48.1 Detailed Description . . . . .                    | 241 |
| 5.48.2 Field Documentation . . . . .                     | 241 |
| 5.49 qm_spi_config_t Struct Reference . . . . .          | 242 |
| 5.49.1 Detailed Description . . . . .                    | 242 |
| 5.49.2 Field Documentation . . . . .                     | 242 |
| 5.50 qm_spi_reg_t Struct Reference . . . . .             | 243 |
| 5.50.1 Detailed Description . . . . .                    | 244 |
| 5.50.2 Field Documentation . . . . .                     | 244 |
| 5.51 qm_spi_transfer_t Struct Reference . . . . .        | 247 |
| 5.51.1 Detailed Description . . . . .                    | 247 |
| 5.51.2 Field Documentation . . . . .                     | 248 |
| 5.52 qm_ss_adc_config_t Struct Reference . . . . .       | 248 |
| 5.52.1 Detailed Description . . . . .                    | 248 |
| 5.52.2 Field Documentation . . . . .                     | 248 |
| 5.53 qm_ss_adc_xfer_t Struct Reference . . . . .         | 249 |
| 5.53.1 Detailed Description . . . . .                    | 249 |
| 5.53.2 Field Documentation . . . . .                     | 249 |
| 5.54 qm_ss_gpio_port_config_t Struct Reference . . . . . | 250 |
| 5.54.1 Detailed Description . . . . .                    | 250 |
| 5.54.2 Field Documentation . . . . .                     | 251 |
| 5.55 qm_ss_i2c_config_t Struct Reference . . . . .       | 251 |
| 5.55.1 Detailed Description . . . . .                    | 252 |
| 5.55.2 Field Documentation . . . . .                     | 252 |
| 5.56 qm_ss_i2c_transfer_t Struct Reference . . . . .     | 252 |
| 5.56.1 Detailed Description . . . . .                    | 253 |

|        |   |     |
|--------|---|-----|
| 5.56.2 | Field Documentation                         | 253 |
| 5.57   | qm_ss_spi_async_transfer_t Struct Reference | 254 |
| 5.57.1 | Detailed Description                        | 254 |
| 5.57.2 | Field Documentation                         | 254 |
| 5.58   | qm_ss_spi_config_t Struct Reference         | 255 |
| 5.58.1 | Detailed Description                        | 255 |
| 5.58.2 | Field Documentation                         | 255 |
| 5.59   | qm_ss_spi_transfer_t Struct Reference       | 256 |
| 5.59.1 | Detailed Description                        | 256 |
| 5.59.2 | Field Documentation                         | 256 |
| 5.60   | qm_ss_timer_config_t Struct Reference       | 257 |
| 5.60.1 | Detailed Description                        | 257 |
| 5.60.2 | Field Documentation                         | 257 |
| 5.61   | qm_uart_config_t Struct Reference           | 258 |
| 5.61.1 | Detailed Description                        | 258 |
| 5.61.2 | Field Documentation                         | 259 |
| 5.62   | qm_uart_reg_t Struct Reference              | 259 |
| 5.62.1 | Detailed Description                        | 260 |
| 5.62.2 | Field Documentation                         | 260 |
| 5.63   | qm_uart_transfer_t Struct Reference         | 262 |
| 5.63.1 | Detailed Description                        | 263 |
| 5.63.2 | Field Documentation                         | 263 |
| 5.64   | qm_wdt_config_t Struct Reference            | 263 |
| 5.64.1 | Detailed Description                        | 264 |
| 5.64.2 | Field Documentation                         | 264 |
| 5.65   | qm_wdt_reg_t Struct Reference               | 264 |
| 5.65.1 | Detailed Description                        | 265 |
| 5.65.2 | Field Documentation                         | 265 |

Index 267

## 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:

|                           |           |
|---------------------------|-----------|
| <b>Error Codes</b>        | <b>7</b>  |
| <b>Clock Management</b>   | <b>8</b>  |
| <b>Quark D2000 ADC</b>    | <b>16</b> |
| <b>Always-on Counters</b> | <b>23</b> |
| <b>Analog Comparator</b>  | <b>27</b> |
| <b>DMA</b>                | <b>28</b> |
| <b>Flash</b>              | <b>34</b> |
| <b>FPR</b>                | <b>39</b> |
| <b>GPIO</b>               | <b>42</b> |
| <b>I2C</b>                | <b>46</b> |
| <b>Identification</b>     | <b>55</b> |
| <b>Initialisation</b>     | <b>56</b> |
| <b>Interrupt</b>          | <b>57</b> |
| <b>ISR</b>                | <b>59</b> |
| <b>Mailbox</b>            | <b>67</b> |
| <b>MPR</b>                | <b>72</b> |
| <b>PIC Timer</b>          | <b>73</b> |
| <b>Pin Muxing setup</b>   | <b>75</b> |
| <b>PWM / Timer</b>        | <b>80</b> |
| <b>RTC</b>                | <b>83</b> |
| <b>SPI</b>                | <b>86</b> |
| <b>UART</b>               | <b>94</b> |

|                                  |     |
|----------------------------------|-----|
| Version                          | 104 |
| WDT                              | 105 |
| SOC_WATCH                        | 108 |
| SS ADC                           | 112 |
| SS GPIO                          | 119 |
| SS I2C                           | 125 |
| SS Interrupt                     | 130 |
| SS ISR                           | 132 |
| SS SPI                           | 136 |
| SS Timer                         | 142 |
| SS Clock                         | 144 |
| SS Power states                  | 149 |
| Quark D2000 Flash Layout         | 152 |
| Quark D2000 Power states         | 153 |
| SoC Registers (D2000)            | 155 |
| RAR                              | 164 |
| Quark SE Flash Layout            | 166 |
| Quark SE SoC Power states        | 167 |
| Quark SE Host Power states       | 169 |
| SoC Registers (Sensor Subsystem) | 171 |
| SoC Registers (SE)               | 174 |
| Quark SE Voltage Regulators      | 183 |
| Syscalls                         | 185 |

## 3 Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

|   |     |
|---|-----|
| <a href="#">int_ss_i2c_reg_t</a><br>SS I2C Interrupt register map | 186 |
| <a href="#">int_ss_spi_reg_t</a><br>SS SPI Interrupt register map | 186 |
| <a href="#">mvic_reg_pad_t</a><br>MVIC register structure         | 186 |

|  |     |
|--|-----|
| <a href="#">pic_timer_reg_pad_t</a><br>PIC timer register structure                | 186 |
| <a href="#">qm_ac_config_t</a><br>Analog Comparator configuration type             | 186 |
| <a href="#">qm_adc_config_t</a><br>ADC configuration type                          | 188 |
| <a href="#">qm_adc_reg_t</a><br>ADC register map                                   | 188 |
| <a href="#">qm_adc_xfer_t</a><br>ADC transfer type                                 | 189 |
| <a href="#">qm_aonpt_config_t</a><br>Always-on Periodic Timer configuration type   | 190 |
| <a href="#">qm_dma_chan_reg_t</a><br>DMA channel register map                      | 191 |
| <a href="#">qm_dma_channel_config_t</a><br>DMA channel configuration structure     | 192 |
| <a href="#">qm_dma_int_reg_t</a><br>DMA interrupt register map                     | 193 |
| <a href="#">qm_dma_misc_reg_t</a><br>DMA miscellaneous register map                | 195 |
| <a href="#">qm_dma_transfer_t</a><br>DMA transfer configuration structure          | 196 |
| <a href="#">qm_flash_config_t</a><br>Flash configuration structure                 | 197 |
| <a href="#">qm_flash_reg_t</a><br>Flash register map                               | 198 |
| <a href="#">qm_fpr_config_t</a><br>Flash Protection Region configuration structure | 198 |
| <a href="#">qm_gpio_port_config_t</a><br>GPIO port configuration type              | 199 |
| <a href="#">qm_gpio_reg_t</a><br>GPIO register map                                 | 201 |
| <a href="#">qm_i2c_config_t</a><br>I2C configuration type                          | 204 |
| <a href="#">qm_i2c_reg_t</a><br>I2C register map                                   | 205 |
| <a href="#">qm_i2c_transfer_t</a><br>I2C transfer type                             | 211 |
| <a href="#">qm_lapic_reg_t</a><br>APIC register block type                         | 213 |
| <a href="#">qm_mailbox_t</a><br>Mailbox register structure                         | 214 |

|  |  |     |
|--|--|-----|
| <a href="#">qm_mbox_msg_t</a>            | Definition of the mailbox message                | 214 |
| <a href="#">qm_mpr_config_t</a>          | SRAM Memory Protection Region configuration type | 215 |
| <a href="#">qm_mpr_reg_t</a>             | Memory Protection Region register map            | 215 |
| <a href="#">qm_mvic_reg_t</a>            | MVIC register map                                | 216 |
| <a href="#">qm_pic_timer_config_t</a>    | PIC timer configuration type                     | 217 |
| <a href="#">qm_pic_timer_reg_t</a>       | PIC timer register map                           | 218 |
| <a href="#">qm_pwm_channel_t</a>         | PWM / Timer channel register map                 | 219 |
| <a href="#">qm_pwm_config_t</a>          | PWM / Timer configuration type                   | 220 |
| <a href="#">qm_pwm_reg_t</a>             | PWM / Timer register map                         | 221 |
| <a href="#">qm_rtc_config_t</a>          | RTC configuration type                           | 222 |
| <a href="#">qm_rtc_reg_t</a>             | RTC register map                                 | 224 |
| <a href="#">qm_scss_aon_reg_t</a>        | Always-on Controller register map                | 225 |
| <a href="#">qm_scss_ccu_reg_t</a>        | System Core register map                         | 226 |
| <a href="#">qm_scss_cmp_reg_t</a>        | Comparator register map                          | 229 |
| <a href="#">qm_scss_gp_reg_t</a>         | General Purpose register map                     | 230 |
| <a href="#">qm_scss_info_reg_t</a>       | Information register map                         | 232 |
| <a href="#">qm_scss_int_reg_t</a>        | Interrupt register map                           | 233 |
| <a href="#">qm_scss_mailbox_reg_t</a>    | Mailbox register map                             | 236 |
| <a href="#">qm_scss_mem_reg_t</a>        | Memory Control register map                      | 236 |
| <a href="#">qm_scss_peripheral_reg_t</a> | Peripheral Registers register map                | 237 |
| <a href="#">qm_scss_pmu_reg_t</a>        | Power Management register map                    | 237 |

|   |     |
|---|-----|
| <a href="#">qm_scss_pmux_reg_t</a><br>Pin MUX register map                        | 239 |
| <a href="#">qm_scss_ss_reg_t</a><br>Sensor Subsystem register map                 | 240 |
| <a href="#">qm_spi_async_transfer_t</a><br>SPI IRQ transfer type                  | 241 |
| <a href="#">qm_spi_config_t</a><br>SPI configuration type                         | 242 |
| <a href="#">qm_spi_reg_t</a><br>SPI register map                                  | 243 |
| <a href="#">qm_spi_transfer_t</a><br>SPI transfer type                            | 247 |
| <a href="#">qm_ss_adc_config_t</a><br>SS ADC configuration type                   | 248 |
| <a href="#">qm_ss_adc_xfer_t</a><br>SS ADC transfer type                          | 249 |
| <a href="#">qm_ss_gpio_port_config_t</a><br>GPIO port configuration type          | 250 |
| <a href="#">qm_ss_i2c_config_t</a><br>QM SS I2C configuration type                | 251 |
| <a href="#">qm_ss_i2c_transfer_t</a><br>QM SS I2C transfer type                   | 252 |
| <a href="#">qm_ss_spi_async_transfer_t</a><br>SPI IRQ transfer type               | 254 |
| <a href="#">qm_ss_spi_config_t</a><br>SPI configuration type                      | 255 |
| <a href="#">qm_ss_spi_transfer_t</a><br>SPI transfer type                         | 256 |
| <a href="#">qm_ss_timer_config_t</a><br>Sensor Subsystem Timer Configuration Type | 257 |
| <a href="#">qm_uart_config_t</a><br>UART configuration type                       | 258 |
| <a href="#">qm_uart_reg_t</a><br>UART register map                                | 259 |
| <a href="#">qm_uart_transfer_t</a><br>UART asynchronous transfer structure        | 262 |
| <a href="#">qm_wdt_config_t</a><br>QM WDT configuration type                      | 263 |
| <a href="#">qm_wdt_reg_t</a><br>Watchdog timer register map                       | 264 |

## 4 Module Documentation

### 4.1 Error Codes

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

The following return codes are used:

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



## 4.2 Clock Management

Clock Management.

### Enumerations

- enum `clk_sys_div_t` {  
`CLK_SYS_DIV_1`, `CLK_SYS_DIV_2`, `CLK_SYS_DIV_4`, `CLK_SYS_DIV_8`,  
`CLK_SYS_DIV_16`, `CLK_SYS_DIV_32`, `CLK_SYS_DIV_64`, `CLK_SYS_DIV_128` }  
*System clock divider type.*
- enum `clk_sys_mode_t` {  
`CLK_SYS_HYB_OSC_32MHZ`, `CLK_SYS_HYB_OSC_16MHZ`, `CLK_SYS_HYB_OSC_8MHZ`, `CLK_SYS_HYB_OSC_4MHZ`,  
`CLK_SYS_RTC_OSC`, `CLK_SYS_CRYSTAL_OSC` }  
*System clock mode type.*
- enum `clk_periph_div_t` { `CLK_PERIPH_DIV_1`, `CLK_PERIPH_DIV_2`, `CLK_PERIPH_DIV_4`, `CLK_PERIPH_DIV_8` }  
*Peripheral clock divider type.*
- enum `clk_gpio_db_div_t` {  
`CLK_GPIO_DB_DIV_1`, `CLK_GPIO_DB_DIV_2`, `CLK_GPIO_DB_DIV_4`, `CLK_GPIO_DB_DIV_8`,  
`CLK_GPIO_DB_DIV_16`, `CLK_GPIO_DB_DIV_32`, `CLK_GPIO_DB_DIV_64`, `CLK_GPIO_DB_DIV_128` }  
*GPIO clock debounce divider type.*
- enum `clk_ext_div_t` { `CLK_EXT_DIV_1`, `CLK_EXT_DIV_2`, `CLK_EXT_DIV_4`, `CLK_EXT_DIV_8` }  
*External crystal clock divider type.*
- enum `clk_rtc_div_t` {  
`CLK_RTC_DIV_1`, `CLK_RTC_DIV_2`, `CLK_RTC_DIV_4`, `CLK_RTC_DIV_8`,  
`CLK_RTC_DIV_16`, `CLK_RTC_DIV_32`, `CLK_RTC_DIV_64`, `CLK_RTC_DIV_128`,  
`CLK_RTC_DIV_256`, `CLK_RTC_DIV_512`, `CLK_RTC_DIV_1024`, `CLK_RTC_DIV_2048`,  
`CLK_RTC_DIV_4096`, `CLK_RTC_DIV_8192`, `CLK_RTC_DIV_16384`, `CLK_RTC_DIV_32768` }  
*RTC clock divider type.*

### Functions

- int `clk_sys_set_mode` (const `clk_sys_mode_t` mode, const `clk_sys_div_t` div)  
*Set clock mode and divisor.*
- int `clk_trim_read` (uint32\_t \*const value)  
*Read the silicon oscillator trim code for the current frequency.*
- int `clk_trim_apply` (const uint32\_t value)  
*Apply silicon oscillator trim code.*
- int `clk_adc_set_div` (const uint16\_t div)  
*Change divider value of ADC clock.*
- int `clk_periph_set_div` (const `clk_periph_div_t` div)  
*Change divider value of peripheral clock.*
- int `clk_gpio_db_set_div` (const `clk_gpio_db_div_t` div)  
*Change divider value of GPIO debounce clock.*
- int `clk_ext_set_div` (const `clk_ext_div_t` div)  
*Change divider value of external clock.*
- int `clk_rtc_set_div` (const `clk_rtc_div_t` div)  
*Change divider value of RTC.*
- int `clk_periph_enable` (const `clk_periph_t` clocks)  
*Enable clocks for peripherals / registers.*
- int `clk_periph_disable` (const `clk_periph_t` clocks)

*Disable clocks for peripherals / registers.*

- `uint32_t clk_sys_get_ticks_per_us` (void)

*Get number of system ticks per micro second.*

- void `clk_sys_udelay` (uint32\_t microseconds)

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

#### 4.2.1 Detailed Description

Clock Management.

#### 4.2.2 Enumeration Type Documentation

##### 4.2.2.1 enum `clk_ext_div_t`

External crystal clock divider type.

Enumerator

**`CLK_EXT_DIV_1`** External Crystal Clock Divider = 1.

**`CLK_EXT_DIV_2`** External Crystal Clock Divider = 2.

**`CLK_EXT_DIV_4`** External Crystal Clock Divider = 4.

**`CLK_EXT_DIV_8`** External Crystal Clock Divider = 8.

Definition at line 92 of file `clk.h`.

##### 4.2.2.2 enum `clk_gpio_db_div_t`

GPIO clock debounce divider type.

Enumerator

**`CLK_GPIO_DB_DIV_1`** GPIO Clock Debounce Divider = 1.

**`CLK_GPIO_DB_DIV_2`** GPIO Clock Debounce Divider = 2.

**`CLK_GPIO_DB_DIV_4`** GPIO Clock Debounce Divider = 4.

**`CLK_GPIO_DB_DIV_8`** GPIO Clock Debounce Divider = 8.

**`CLK_GPIO_DB_DIV_16`** GPIO Clock Debounce Divider = 16.

**`CLK_GPIO_DB_DIV_32`** GPIO Clock Debounce Divider = 32.

**`CLK_GPIO_DB_DIV_64`** GPIO Clock Debounce Divider = 64.

**`CLK_GPIO_DB_DIV_128`** GPIO Clock Debounce Divider = 128.

Definition at line 78 of file `clk.h`.

##### 4.2.2.3 enum `clk_periph_div_t`

Peripheral clock divider type.

Enumerator

**`CLK_PERIPH_DIV_1`** Peripheral Clock Divider = 1.

**`CLK_PERIPH_DIV_2`** Peripheral Clock Divider = 2.

**`CLK_PERIPH_DIV_4`** Peripheral Clock Divider = 4.

**`CLK_PERIPH_DIV_8`** Peripheral Clock Divider = 8.

Definition at line 68 of file `clk.h`.

#### 4.2.2.4 enum clk\_rtc\_div\_t

RTC clock divider type.

##### Enumerator

- CLK\_RTC\_DIV\_1** Real Time Clock Divider = 1.
- CLK\_RTC\_DIV\_2** Real Time Clock Divider = 2.
- CLK\_RTC\_DIV\_4** Real Time Clock Divider = 4.
- CLK\_RTC\_DIV\_8** Real Time Clock Divider = 8.
- CLK\_RTC\_DIV\_16** Real Time Clock Divider = 16.
- CLK\_RTC\_DIV\_32** Real Time Clock Divider = 32.
- CLK\_RTC\_DIV\_64** Real Time Clock Divider = 64.
- CLK\_RTC\_DIV\_128** Real Time Clock Divider = 128.
- CLK\_RTC\_DIV\_256** Real Time Clock Divider = 256.
- CLK\_RTC\_DIV\_512** Real Time Clock Divider = 512.
- CLK\_RTC\_DIV\_1024** Real Time Clock Divider = 1024.
- CLK\_RTC\_DIV\_2048** Real Time Clock Divider = 2048.
- CLK\_RTC\_DIV\_4096** Real Time Clock Divider = 4096.
- CLK\_RTC\_DIV\_8192** Real Time Clock Divider = 8192.
- CLK\_RTC\_DIV\_16384** Real Time Clock Divider = 16384.
- CLK\_RTC\_DIV\_32768** Real Time Clock Divider = 32768.

Definition at line 102 of file clk.h.

#### 4.2.2.5 enum clk\_sys\_div\_t

System clock divider type.

##### Enumerator

- CLK\_SYS\_DIV\_1** Clock Divider = 1.
- CLK\_SYS\_DIV\_2** Clock Divider = 2.
- CLK\_SYS\_DIV\_4** Clock Divider = 4.
- CLK\_SYS\_DIV\_8** Clock Divider = 8.
- CLK\_SYS\_DIV\_16** Clock Divider = 16.
- CLK\_SYS\_DIV\_32** Clock Divider = 32.
- CLK\_SYS\_DIV\_64** Clock Divider = 64.
- CLK\_SYS\_DIV\_128** Clock Divider = 128.

Definition at line 39 of file clk.h.

#### 4.2.2.6 enum clk\_sys\_mode\_t

System clock mode type.

##### Enumerator

- CLK\_SYS\_HYB\_OSC\_32MHZ** 32MHz Hybrid Oscillator Clock.
- CLK\_SYS\_HYB\_OSC\_16MHZ** 16MHz Hybrid Oscillator Clock.
- CLK\_SYS\_HYB\_OSC\_8MHZ** 8MHz Hybrid Oscillator Clock.
- CLK\_SYS\_HYB\_OSC\_4MHZ** 4MHz Hybrid Oscillator Clock.
- CLK\_SYS\_RTC\_OSC** Real Time Clock.
- CLK\_SYS\_CRYSTAL\_OSC** Crystal Oscillator Clock.

Definition at line 56 of file clk.h.

## 4.2.3 Function Documentation

4.2.3.1 `int clk_adc_set_div ( const uint16_t div )`

Change divider value of ADC clock.

Change ADC clock divider value. The new divider value is set to N, where N is the value set by the function and is between 1 and 1024. This function is only available on D2000.

## Parameters

|                 |                  |                                  |
|-----------------|------------------|----------------------------------|
| <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.                                     |
| <code>Negative</code> | <a href="#">errno</a> for possible error codes. |

Definition at line 259 of file clk.c.

4.2.3.2 `int clk_ext_set_div ( const clk_ext_div_t div )`

Change divider value of external clock.

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

## Parameters

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

## Returns

Standard errno return type for QMSI.

## Return values

|                       |   |
|-----------------------|---|
| <code>0</code>        | on success.                                     |
| <code>Negative</code> | <a href="#">errno</a> for possible error codes. |

Definition at line 316 of file clk.c.

References CLK\_EXT\_DIV\_8.

4.2.3.3 `int clk_gpio_db_set_div ( const clk_gpio_db_div_t div )`

Change divider value of GPIO debounce clock.

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

## Parameters

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

## Returns

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 302 of file clk.c.

References CLK\_GPIO\_DB\_DIV\_128.

**4.2.3.4 int clk\_periph\_disable ( const clk\_periph\_t clocks )**

Disable clocks for peripherals / registers.

**Parameters**

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

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 354 of file clk.c.

References CLK\_PERIPH\_ALL, SOCW\_EVENT\_REGISTER, and SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL.

Referenced by power\_soc\_deep\_sleep(), and power\_soc\_sleep().

**4.2.3.5 int clk\_periph\_enable ( const clk\_periph\_t clocks )**

Enable clocks for peripherals / registers.

**Parameters**

|           |               |   |
|-----------|---------------|---|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 342 of file clk.c.

References CLK\_PERIPH\_ALL, SOCW\_EVENT\_REGISTER, and SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL.

Referenced by power\_soc\_deep\_sleep().

**4.2.3.6 int clk\_periph\_set\_div ( const clk\_periph\_div\_t div )**

Change divider value of peripheral clock.

Change Peripheral clock divider value. The maximum divisor is /8. Refer to the list of supported peripherals for your SoC.

## Parameters

|           |            |   |
|-----------|------------|---|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 280 of file clk.c.

References CLK\_PERIPH\_DIV\_8.

4.2.3.7 int clk\_rtc\_set\_div ( const clk\_rtc\_div\_t *div* )

Change divider value of RTC.

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

## Parameters

|           |            |                            |
|-----------|------------|----------------------------|
| <i>in</i> | <i>div</i> | Divider value for the RTC. |
|-----------|------------|----------------------------|

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 329 of file clk.c.

References CLK\_RTC\_DIV\_32768.

Referenced by qm\_rtc\_set\_config().

## 4.2.3.8 uint32\_t clk\_sys\_get\_ticks\_per\_us ( void )

Get number of system ticks per micro second.

## Returns

uint32\_t Number of system ticks per micro second.

Definition at line 366 of file clk.c.

Referenced by ss\_clk\_adc\_set\_div().

4.2.3.9 int clk\_sys\_set\_mode ( const clk\_sys\_mode\_t *mode*, const clk\_sys\_div\_t *div* )

Set clock mode and divisor.

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

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

**Parameters**

|           |             |                                     |
|-----------|-------------|-------------------------------------|
| <i>in</i> | <i>mode</i> | System clock source operating mode. |
| <i>in</i> | <i>div</i>  | System clock divisor.               |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 109 of file clk.c.

References CLK\_SYS\_CRYSTAL\_OSC, CLK\_SYS\_HYB\_OSC\_16MHZ, CLK\_SYS\_HYB\_OSC\_32MHZ, CLK\_SYS\_HYB\_OSC\_4MHZ, CLK\_SYS\_HYB\_OSC\_8MHZ, CLK\_SYS\_RTC\_OSC, clk\_trim\_apply(), SOCW\_EVENT\_REGISTER, SOCW\_REG\_CCU\_SYS\_CLK\_CTL, and SOCW\_REG\_OSC0\_CFG1.

Referenced by power\_soc\_deep\_sleep(), and power\_soc\_sleep().

**4.2.3.10 void clk\_sys\_udelay ( uint32\_t *microseconds* )**

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

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

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

**Parameters**

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

Definition at line 371 of file clk.c.

Referenced by clk\_trim\_apply().

**4.2.3.11 int clk\_trim\_apply ( const uint32\_t *value* )**

Apply silicon oscillator trim code.

**Parameters**

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

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 234 of file clk.c.

References clk\_sys\_udelay().

Referenced by clk\_sys\_set\_mode().

**4.2.3.12 int clk\_trim\_read ( uint32\_t \*const *value* )**

Read the silicon oscillator trim code for the current frequency.

**Parameters**

|     |       |                                 |
|-----|-------|---------------------------------|
| out | value | Pointer to store the trim code. |
|-----|-------|---------------------------------|

**Returns**

Standard errno return type for QMSI.

**Return values**

|          |   |
|----------|---|
| 0        | on success.                                     |
| Negative | <a href="#">errno</a> for possible error codes. |

Definition at line 224 of file clk.c.



### 4.3 Quark D2000 ADC

Analog to Digital Converter (ADC).

#### Data Structures

- struct [qm\\_adc\\_config\\_t](#)  
*ADC configuration type.*
- struct [qm\\_adc\\_xfer\\_t](#)  
*ADC transfer type.*

#### Typedefs

- typedef uint16\_t [qm\\_adc\\_sample\\_t](#)  
*ADC sample size type.*
- typedef uint8\_t [qm\\_adc\\_calibration\\_t](#)  
*ADC calibration type.*

#### Enumerations

- enum [qm\\_adc\\_status\\_t](#) { [QM\\_ADC\\_IDLE](#), [QM\\_ADC\\_COMPLETE](#), [QM\\_ADC\\_OVERFLOW](#) }
- enum [qm\\_adc\\_resolution\\_t](#) { [QM\\_ADC\\_RES\\_6\\_BITS](#), [QM\\_ADC\\_RES\\_8\\_BITS](#), [QM\\_ADC\\_RES\\_10\\_BITS](#), [QM\\_ADC\\_RES\\_12\\_BITS](#) }  
*ADC resolution type.*
- enum [qm\\_adc\\_mode\\_t](#) { [QM\\_ADC\\_MODE\\_DEEP\\_PWR\\_DOWN](#), [QM\\_ADC\\_MODE\\_PWR\\_DOWN](#), [QM\\_ADC\\_MODE\\_STDBY](#), [QM\\_ADC\\_MODE\\_NORM\\_CAL](#), [QM\\_ADC\\_MODE\\_NORM\\_NO\\_CAL](#) }  
*ADC operating mode type.*
- enum [qm\\_adc\\_channel\\_t](#) { [QM\\_ADC\\_CH\\_0](#), [QM\\_ADC\\_CH\\_1](#), [QM\\_ADC\\_CH\\_2](#), [QM\\_ADC\\_CH\\_3](#), [QM\\_ADC\\_CH\\_4](#), [QM\\_ADC\\_CH\\_5](#), [QM\\_ADC\\_CH\\_6](#), [QM\\_ADC\\_CH\\_7](#), [QM\\_ADC\\_CH\\_8](#), [QM\\_ADC\\_CH\\_9](#), [QM\\_ADC\\_CH\\_10](#), [QM\\_ADC\\_CH\\_11](#), [QM\\_ADC\\_CH\\_12](#), [QM\\_ADC\\_CH\\_13](#), [QM\\_ADC\\_CH\\_14](#), [QM\\_ADC\\_CH\\_15](#), [QM\\_ADC\\_CH\\_16](#), [QM\\_ADC\\_CH\\_17](#), [QM\\_ADC\\_CH\\_18](#) }  
*ADC channels type.*
- enum [qm\\_adc\\_cb\\_source\\_t](#) { [QM\\_ADC\\_TRANSFER](#), [QM\\_ADC\\_MODE\\_CHANGED](#), [QM\\_ADC\\_CAL\\_COMPLETE](#) }  
*ADC interrupt callback source.*

#### Functions

- int [qm\\_adc\\_set\\_mode](#) (const [qm\\_adc\\_t](#) adc, const [qm\\_adc\\_mode\\_t](#) mode)  
*Switch operating mode of ADC.*
- int [qm\\_adc\\_irq\\_set\\_mode](#) (const [qm\\_adc\\_t](#) adc, const [qm\\_adc\\_mode\\_t](#) mode, void(\*callback)(void \*data, int error, [qm\\_adc\\_status\\_t](#) status, [qm\\_adc\\_cb\\_source\\_t](#) source), void \*callback\_data)  
*Switch operating mode of ADC.*
- int [qm\\_adc\\_calibrate](#) (const [qm\\_adc\\_t](#) adc)  
*Calibrate the ADC.*
- int [qm\\_adc\\_irq\\_calibrate](#) (const [qm\\_adc\\_t](#) adc, void(\*callback)(void \*data, int error, [qm\\_adc\\_status\\_t](#) status, [qm\\_adc\\_cb\\_source\\_t](#) source), void \*callback\_data)  
*Calibrate the ADC.*

- int [qm\\_adc\\_set\\_calibration](#) (const [qm\\_adc\\_t](#) adc, const [qm\\_adc\\_calibration\\_t](#) cal)  
*Set ADC calibration data.*
- int [qm\\_adc\\_get\\_calibration](#) (const [qm\\_adc\\_t](#) adc, [qm\\_adc\\_calibration\\_t](#) \*const cal)  
*Get the current calibration data for an ADC.*
- int [qm\\_adc\\_set\\_config](#) (const [qm\\_adc\\_t](#) adc, const [qm\\_adc\\_config\\_t](#) \*const cfg)  
*Set ADC configuration.*
- int [qm\\_adc\\_convert](#) (const [qm\\_adc\\_t](#) adc, [qm\\_adc\\_xfer\\_t](#) \*const xfer, [qm\\_adc\\_status\\_t](#) \*const status)  
*Synchronously read values from the ADC.*
- int [qm\\_adc\\_irq\\_convert](#) (const [qm\\_adc\\_t](#) adc, [qm\\_adc\\_xfer\\_t](#) \*const xfer)  
*Asynchronously read values from the ADC.*

#### 4.3.1 Detailed Description

Analog to Digital Converter (ADC).

#### 4.3.2 Enumeration Type Documentation

##### 4.3.2.1 enum [qm\\_adc\\_cb\\_source\\_t](#)

ADC interrupt callback source.

Enumerator

- [QM\\_ADC\\_TRANSFER](#)** Transfer complete or error callback.
- [QM\\_ADC\\_MODE\\_CHANGED](#)** Mode change complete callback.
- [QM\\_ADC\\_CAL\\_COMPLETE](#)** Calibration complete callback.

Definition at line 85 of file [qm\\_adc.h](#).

##### 4.3.2.2 enum [qm\\_adc\\_channel\\_t](#)

ADC channels type.

Enumerator

- [QM\\_ADC\\_CH\\_0](#)** ADC Channel 0.
- [QM\\_ADC\\_CH\\_1](#)** ADC Channel 1.
- [QM\\_ADC\\_CH\\_2](#)** ADC Channel 2.
- [QM\\_ADC\\_CH\\_3](#)** ADC Channel 3.
- [QM\\_ADC\\_CH\\_4](#)** ADC Channel 4.
- [QM\\_ADC\\_CH\\_5](#)** ADC Channel 5.
- [QM\\_ADC\\_CH\\_6](#)** ADC Channel 6.
- [QM\\_ADC\\_CH\\_7](#)** ADC Channel 7.
- [QM\\_ADC\\_CH\\_8](#)** ADC Channel 8.
- [QM\\_ADC\\_CH\\_9](#)** ADC Channel 9.
- [QM\\_ADC\\_CH\\_10](#)** ADC Channel 10.
- [QM\\_ADC\\_CH\\_11](#)** ADC Channel 11.
- [QM\\_ADC\\_CH\\_12](#)** ADC Channel 12.
- [QM\\_ADC\\_CH\\_13](#)** ADC Channel 13.
- [QM\\_ADC\\_CH\\_14](#)** ADC Channel 14.
- [QM\\_ADC\\_CH\\_15](#)** ADC Channel 15.

**QM\_ADC\_CH\_16** ADC Channel 16.

**QM\_ADC\_CH\_17** ADC Channel 17.

**QM\_ADC\_CH\_18** ADC Channel 18.

Definition at line 60 of file qm\_adc.h.

#### 4.3.2.3 enum qm\_adc\_mode\_t

ADC operating mode type.

##### Enumerator

**QM\_ADC\_MODE\_DEEP\_PWR\_DOWN** Deep power down mode.

**QM\_ADC\_MODE\_PWR\_DOWN** Power down mode.

**QM\_ADC\_MODE\_STDBY** Standby mode.

**QM\_ADC\_MODE\_NORM\_CAL** Normal mode, with calibration.

**QM\_ADC\_MODE\_NORM\_NO\_CAL** Normal mode, no calibration.

Definition at line 49 of file qm\_adc.h.

#### 4.3.2.4 enum qm\_adc\_resolution\_t

ADC resolution type.

##### Enumerator

**QM\_ADC\_RES\_6\_BITS** 6-bit mode.

**QM\_ADC\_RES\_8\_BITS** 8-bit mode.

**QM\_ADC\_RES\_10\_BITS** 10-bit mode.

**QM\_ADC\_RES\_12\_BITS** 12-bit mode.

Definition at line 39 of file qm\_adc.h.

#### 4.3.2.5 enum qm\_adc\_status\_t

##### Enumerator

**QM\_ADC\_IDLE** ADC idle.

**QM\_ADC\_COMPLETE** ADC transfer complete.

**QM\_ADC\_OVERFLOW** ADC FIFO overflow error.

Definition at line 30 of file qm\_adc.h.

### 4.3.3 Function Documentation

#### 4.3.3.1 int qm\_adc\_calibrate ( const qm\_adc\_t adc )

Calibrate the ADC.

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

**Parameters**

|           |            |                         |
|-----------|------------|-------------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 206 of file qm\_adc.c.

#### 4.3.3.2 int qm\_adc\_convert ( const qm\_adc\_t *adc*, qm\_adc\_xfer\_t \*const *xfer*, qm\_adc\_status\_t \*const *status* )

Synchronously read values from the ADC.

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

**Parameters**

|                |               |   |
|----------------|---------------|---|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 347 of file qm\_adc.c.

References qm\_adc\_xfer\_t::ch, qm\_adc\_xfer\_t::ch\_len, QM\_ADC\_COMPLETE, qm\_adc\_xfer\_t::samples, and qm\_adc\_xfer\_t::samples\_len.

#### 4.3.3.3 int qm\_adc\_get\_calibration ( const qm\_adc\_t *adc*, qm\_adc\_calibration\_t \*const *cal* )

Get the current calibration data for an ADC.

**Parameters**

|            |            |  |
|------------|------------|--|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 263 of file qm\_adc.c.

#### 4.3.3.4 int qm\_adc\_irq\_calibrate ( const qm\_adc\_t *adc*, void (\*)(void \*data, int error, qm\_adc\_status\_t status, qm\_adc\_cb\_source\_t source) *callback*, void \* *callback\_data* )

Calibrate the ADC.

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

#### Parameters

|    |                      |                                |
|----|----------------------|--------------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 222 of file qm\_adc.c.

#### 4.3.3.5 int qm\_adc\_irq\_convert ( const qm\_adc\_t *adc*, qm\_adc\_xfer\_t \*const *xfer* )

Asynchronously read values from the ADC.

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

#### Parameters

|         |             |  |
|---------|-------------|--|
| in      | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 393 of file qm\_adc.c.

References qm\_adc\_xfer\_t::ch, qm\_adc\_xfer\_t::ch\_len, qm\_adc\_xfer\_t::samples, and qm\_adc\_xfer\_t::samples\_len.

#### 4.3.3.6 int qm\_adc\_irq\_set\_mode ( const qm\_adc\_t *adc*, const qm\_adc\_mode\_t *mode*, void (\*)(void \*data, int error, qm\_adc\_status\_t status, qm\_adc\_cb\_source\_t source) *callback*, void \* *callback\_data* )

Switch operating mode of ADC.

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

#### Parameters

|    |                 |                                |
|----|-----------------|--------------------------------|
| in | <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> | <a href="#">errno</a> for possible error codes. |

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

References `QM_ADC_MODE_NORM_CAL`, and `QM_ADC_MODE_NORM_NO_CAL`.

#### 4.3.3.7 `int qm_adc_set_calibration ( const qm_adc_t adc, const qm_adc_calibration_t cal )`

Set ADC calibration data.

**Parameters**

|           |            |                                   |
|-----------|------------|-----------------------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 244 of file `qm_adc.c`.

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

Set ADC configuration.

This sets the sample window and resolution.

**Parameters**

|           |            |   |
|-----------|------------|---|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 331 of file `qm_adc.c`.

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

#### 4.3.3.9 `int qm_adc_set_mode ( const qm_adc_t adc, const qm_adc_mode_t mode )`

Switch operating mode of ADC.

This call is blocking.

**Parameters**

|           |             |                      |
|-----------|-------------|----------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 273 of file qm\_adc.c.

References QM\_ADC\_MODE\_NORM\_CAL, and QM\_ADC\_MODE\_NORM\_NO\_CAL.

Referenced by power\_soc\_deep\_sleep(), and power\_soc\_sleep().

## 4.4 Always-on Counters

Always-on Counters.

### Data Structures

- struct [qm\\_aonpt\\_config\\_t](#)  
*Always-on Periodic Timer configuration type.*

### Functions

- int [qm\\_aonc\\_enable](#) (const [qm\\_scss\\_aon\\_t](#) aonc)  
*Enable the Always-on Counter.*
- int [qm\\_aonc\\_disable](#) (const [qm\\_scss\\_aon\\_t](#) aonc)  
*Disable the Always-on Counter.*
- int [qm\\_aonc\\_get\\_value](#) (const [qm\\_scss\\_aon\\_t](#) aonc, uint32\_t \*const val)  
*Get the current value of the Always-on Counter.*
- int [qm\\_aonpt\\_set\\_config](#) (const [qm\\_scss\\_aon\\_t](#) aonc, const [qm\\_aonpt\\_config\\_t](#) \*const cfg)  
*Set the Always-on Periodic Timer configuration.*
- int [qm\\_aonpt\\_get\\_value](#) (const [qm\\_scss\\_aon\\_t](#) aonc, uint32\_t \*const val)  
*Get the current value of the Always-on Periodic Timer.*
- int [qm\\_aonpt\\_get\\_status](#) (const [qm\\_scss\\_aon\\_t](#) aonc, bool \*const status)  
*Get the current status of the Always-on Periodic Timer.*
- int [qm\\_aonpt\\_clear](#) (const [qm\\_scss\\_aon\\_t](#) aonc)  
*Clear the status of the Always-on Periodic Timer.*
- int [qm\\_aonpt\\_reset](#) (const [qm\\_scss\\_aon\\_t](#) aonc)  
*Reset the Always-on Periodic Timer back to the configured value.*

#### 4.4.1 Detailed Description

Always-on Counters.

#### 4.4.2 Function Documentation

##### 4.4.2.1 int [qm\\_aonc\\_disable](#) ( const [qm\\_scss\\_aon\\_t](#) aonc )

Disable the Always-on Counter.

#### Parameters

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

#### Returns

Standard errno return type for QMSI.

#### Return values

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



|                 |   |
|-----------------|---|
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |
|-----------------|---|

Definition at line 54 of file qm\_aon\_counters.c.

#### 4.4.2.2 int qm\_aonc\_enable ( const qm\_scss\_aon\_t aonc )

Enable the Always-on Counter.

##### Parameters

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

##### Returns

Standard errno return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 45 of file qm\_aon\_counters.c.

#### 4.4.2.3 int qm\_aonc\_get\_value ( const qm\_scss\_aon\_t aonc, uint32\_t \*const 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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 63 of file qm\_aon\_counters.c.

#### 4.4.2.4 int qm\_aonpt\_clear ( const qm\_scss\_aon\_t aonc )

Clear the status of the Always-on Periodic Timer.

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

##### Parameters

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

##### Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 109 of file `qm_aon_counters.c`.

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

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

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

## Parameters

|            |               |  |
|------------|---------------|--|
| <i>in</i>  | <i>aonc</i>   | Always-on counter to read.                                     |
| <i>out</i> | <i>status</i> | Status of the Always-on Periodic Timer. This must not be NULL. |

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 100 of file `qm_aon_counters.c`.

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

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

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

## Parameters

|            |             |   |
|------------|-------------|---|
| <i>in</i>  | <i>aonc</i> | Always-on counter to read.                                    |
| <i>out</i> | <i>val</i>  | Value of the Always-on Periodic Timer. This must not be NULL. |

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 91 of file `qm_aon_counters.c`.

4.4.2.7 `int qm_aonpt_reset ( const qm_scss_aon_t aonc )`

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

## Parameters

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

## Returns

Standard `errno` return type for QMSI.

## Return values

|  |                 |   |
|--|-----------------|---|
|  | <i>0</i>        | on success.                                     |
|  | <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 118 of file qm\_aon\_counters.c.

#### 4.4.2.8 int qm\_aonpt\_set\_config ( const qm\_scss\_aon\_t aonc, const qm\_aonpt\_config\_t \*const cfg )

Set the Always-on Periodic Timer configuration.

This includes the initial value of the Always-on Periodic Timer, the interrupt enable and the callback function that will be run when the timer 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

|  |                 |   |
|--|-----------------|---|
|  | <i>0</i>        | on success.                                     |
|  | <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 72 of file qm\_aon\_counters.c.

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

## 4.5 Analog Comparator

Analog Comparator.

### Data Structures

- struct [qm\\_ac\\_config\\_t](#)  
*Analog Comparator configuration type.*

### Functions

- int [qm\\_ac\\_set\\_config](#) (const [qm\\_ac\\_config\\_t](#) \*const config)  
*Set Analog Comparator configuration.*

#### 4.5.1 Detailed Description

Analog Comparator.

#### 4.5.2 Function Documentation

##### 4.5.2.1 int qm\_ac\_set\_config ( const qm\_ac\_config\_t \*const config )

Set Analog Comparator configuration.

#### Parameters

|           |               |   |
|-----------|---------------|---|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 42 of file [qm\\_comparator.c](#).

References [qm\\_ac\\_config\\_t::callback](#), [qm\\_ac\\_config\\_t::callback\\_data](#), [qm\\_ac\\_config\\_t::int\\_en](#), [qm\\_ac\\_config\\_t::polarity](#), [qm\\_ac\\_config\\_t::power](#), and [qm\\_ac\\_config\\_t::reference](#).

## 4.6 DMA

DMA Driver for Quark Microcontrollers.

### Data Structures

- struct [qm\\_dma\\_channel\\_config\\_t](#)  
*DMA channel configuration structure.*
- struct [qm\\_dma\\_transfer\\_t](#)  
*DMA transfer configuration structure.*

### Enumerations

- enum [qm\\_dma\\_handshake\\_polarity\\_t](#) { [QM\\_DMA\\_HANDSHAKE\\_POLARITY\\_HIGH](#) = 0x0, [QM\\_DMA\\_HANDSHAKE\\_POLARITY\\_LOW](#) = 0x1 }
- DMA Handshake Polarity.*
- enum [qm\\_dma\\_burst\\_length\\_t](#) { [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_1](#) = 0x0, [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_4](#) = 0x1, [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_8](#) = 0x2, [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_16](#) = 0x3, [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_32](#) = 0x4, [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_64](#) = 0x5, [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_128](#), [QM\\_DMA\\_BURST\\_TRANS\\_LENGTH\\_256](#) = 0x7 }
- DMA Burst Transfer Length.*
- enum [qm\\_dma\\_transfer\\_width\\_t](#) { [QM\\_DMA\\_TRANS\\_WIDTH\\_8](#) = 0x0, [QM\\_DMA\\_TRANS\\_WIDTH\\_16](#) = 0x1, [QM\\_DMA\\_TRANS\\_WIDTH\\_32](#) = 0x2, [QM\\_DMA\\_TRANS\\_WIDTH\\_64](#) = 0x3, [QM\\_DMA\\_TRANS\\_WIDTH\\_128](#) = 0x4, [QM\\_DMA\\_TRANS\\_WIDTH\\_256](#) = 0x5 }
- DMA Transfer Width.*
- enum [qm\\_dma\\_channel\\_direction\\_t](#) { [QM\\_DMA\\_MEMORY\\_TO\\_MEMORY](#) = 0x0, [QM\\_DMA\\_MEMORY\\_TO\\_PERIPHERAL](#), [QM\\_DMA\\_PERIPHERAL\\_TO\\_MEMORY](#) = 0x2 }
- DMA channel direction.*

### Functions

- int [qm\\_dma\\_init](#) (const [qm\\_dma\\_t](#) dma)  
*Initialise the DMA controller.*
- int [qm\\_dma\\_channel\\_set\\_config](#) (const [qm\\_dma\\_t](#) dma, const [qm\\_dma\\_channel\\_id\\_t](#) channel\_id, [qm\\_dma\\_channel\\_config\\_t](#) \*const channel\_config)  
*Setup a DMA channel configuration.*
- int [qm\\_dma\\_transfer\\_set\\_config](#) (const [qm\\_dma\\_t](#) dma, const [qm\\_dma\\_channel\\_id\\_t](#) channel\_id, [qm\\_dma\\_transfer\\_t](#) \*const transfer\_config)  
*Setup a DMA channel transfer.*
- int [qm\\_dma\\_transfer\\_start](#) (const [qm\\_dma\\_t](#) dma, const [qm\\_dma\\_channel\\_id\\_t](#) channel\_id)  
*Start a DMA transfer.*
- int [qm\\_dma\\_transfer\\_terminate](#) (const [qm\\_dma\\_t](#) dma, const [qm\\_dma\\_channel\\_id\\_t](#) channel\_id)  
*Terminate a DMA transfer.*
- int [qm\\_dma\\_transfer\\_mem\\_to\\_mem](#) (const [qm\\_dma\\_t](#) dma, const [qm\\_dma\\_channel\\_id\\_t](#) channel\_id, [qm\\_dma\\_transfer\\_t](#) \*const transfer\_config)  
*Setup and start memory to memory transfer.*

#### 4.6.1 Detailed Description

DMA Driver for Quark Microcontrollers.

## 4.6.2 Enumeration Type Documentation

### 4.6.2.1 enum qm\_dma\_burst\_length\_t

DMA Burst Transfer Length.

#### Enumerator

- QM\_DMA\_BURST\_TRANS\_LENGTH\_1** Burst length 1 data item.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_4** Burst length 4 data items.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_8** Burst length 8 data items.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_16** Burst length 16 data items.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_32** Burst length 32 data items.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_64** Burst length 64 data items.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_128** Burst length 128 data items.
- QM\_DMA\_BURST\_TRANS\_LENGTH\_256** Burst length 256 data items.

Definition at line 29 of file qm\_dma.h.

### 4.6.2.2 enum qm\_dma\_channel\_direction\_t

DMA channel direction.

#### Enumerator

- QM\_DMA\_MEMORY\_TO\_MEMORY** Memory to memory transfer.
- QM\_DMA\_MEMORY\_TO\_PERIPHERAL** Memory to peripheral transfer.
- QM\_DMA\_PERIPHERAL\_TO\_MEMORY** Peripheral to memory transfer.

Definition at line 56 of file qm\_dma.h.

### 4.6.2.3 enum qm\_dma\_handshake\_polarity\_t

DMA Handshake Polarity.

#### Enumerator

- QM\_DMA\_HANDSHAKE\_POLARITY\_HIGH** Set HS polarity high.
- QM\_DMA\_HANDSHAKE\_POLARITY\_LOW** Set HS polarity low.

Definition at line 21 of file qm\_dma.h.

### 4.6.2.4 enum qm\_dma\_transfer\_width\_t

DMA Transfer Width.

#### Enumerator

- QM\_DMA\_TRANS\_WIDTH\_8** Transfer width of 8 bits.
- QM\_DMA\_TRANS\_WIDTH\_16** Transfer width of 16 bits.
- QM\_DMA\_TRANS\_WIDTH\_32** Transfer width of 32 bits.
- QM\_DMA\_TRANS\_WIDTH\_64** Transfer width of 64 bits.
- QM\_DMA\_TRANS\_WIDTH\_128** Transfer width of 128 bits.
- QM\_DMA\_TRANS\_WIDTH\_256** Transfer width of 256 bits.

Definition at line 44 of file qm\_dma.h.

### 4.6.3 Function Documentation

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

Setup a DMA channel configuration.

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

##### Parameters

|    |                       |  |
|----|-----------------------|--|
| in | <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 189 of file `qm_dma.c`.

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

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

#### 4.6.3.2 `int qm_dma_init ( const qm_dma_t dma )`

Initialise the DMA controller.

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

##### Parameters

|    |            |               |
|----|------------|---------------|
| in | <i>dma</i> | DMA instance. |
|----|------------|---------------|

##### Returns

Standard errno return type for QMSI.

##### Return values

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

|                 |   |
|-----------------|---|
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |
|-----------------|---|

Definition at line 143 of file `qm_dma.c`.

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

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

|           |                        |   |
|-----------|------------------------|---|
| <i>in</i> | <i>dma</i>             | DMA instance.   |
| <i>in</i> | <i>channel_id</i>      | The channel to start.   |
| <i>in</i> | <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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 346 of file `qm_dma.c`.

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

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

Setup a DMA channel transfer.

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

#### Parameters

|           |                        |   |
|-----------|------------------------|---|
| <i>in</i> | <i>dma</i>             | DMA instance.   |
| <i>in</i> | <i>channel_id</i>      | The channel to start.   |
| <i>in</i> | <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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 267 of file `qm_dma.c`.

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

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

#### 4.6.3.5 `int qm_dma_transfer_start ( const qm_dma_t dma, const qm_dma_channel_id_t channel_id )`

Start a DMA transfer.

`qm_dma_transfer_set_config()` must first be called before starting a transfer.

## Parameters

|           |                   |                       |
|-----------|-------------------|-----------------------|
| <i>in</i> | <i>dma</i>        | DMA instance.         |
| <i>in</i> | <i>channel_id</i> | The channel to start. |

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 293 of file `qm_dma.c`.

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

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

#### 4.6.3.6 `int qm_dma_transfer_terminate ( const qm_dma_t dma, const qm_dma_channel_id_t channel_id )`

Terminate a DMA transfer.

This function is only called if a transfer needs to be terminated manually. This may be 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

|           |                   |                      |
|-----------|-------------------|----------------------|
| <i>in</i> | <i>dma</i>        | DMA instance.        |
| <i>in</i> | <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>   <a href="#">errno</a> for possible error codes. |
|---|

Definition at line 312 of file qm\_dma.c.

References qm\_dma\_int\_reg\_t::mask\_err\_low, qm\_dma\_int\_reg\_t::mask\_tfr\_low, QM\_DMA\_CHANNEL\_NUM, and QM\_DMA\_NUM.

Referenced by qm\_i2c\_dma\_transfer\_terminate(), qm\_spi\_dma\_transfer\_terminate(), qm\_uart\_dma\_read\_terminate(), and qm\_uart\_dma\_write\_terminate().

## 4.7 Flash

Flash controller.

### Data Structures

- struct `qm_flash_config_t`  
*Flash configuration structure.*

### Enumerations

- enum `qm_flash_region_t` { `QM_FLASH_REGION_OTP` = 0, `QM_FLASH_REGION_SYS`, `QM_FLASH_REGION_DATA`, `QM_FLASH_REGION_NUM` }  
*Flash region enum.*
- enum `qm_flash_disable_t` { `QM_FLASH_WRITE_ENABLE`, `QM_FLASH_WRITE_DISABLE` }  
*Flash write disable / enable enum.*

### Functions

- int `qm_flash_set_config` (const `qm_flash_t` flash, const `qm_flash_config_t` \*const cfg)  
*Configure a Flash controller.*
- int `qm_flash_word_write` (const `qm_flash_t` flash, const `qm_flash_region_t` region, uint32\_t f\_addr, const uint32\_t data)  
*Write 4 bytes of data to Flash.*
- int `qm_flash_page_update` (const `qm_flash_t` flash, const `qm_flash_region_t` reg, uint32\_t f\_addr, uint32\_t \*const page\_buf, const uint32\_t \*const data, uint32\_t len)  
*Write multiple of 4 bytes of data to Flash.*
- int `qm_flash_page_write` (const `qm_flash_t` flash, const `qm_flash_region_t` region, uint32\_t page\_num, const uint32\_t \*data, uint32\_t len)  
*Write a 2KB flash page.*
- int `qm_flash_page_erase` (const `qm_flash_t` flash, const `qm_flash_region_t` region, uint32\_t page\_num)  
*Erase one page of Flash.*
- int `qm_flash_mass_erase` (const `qm_flash_t` flash, const uint8\_t include\_rom)  
*Perform mass erase.*

#### 4.7.1 Detailed Description

Flash controller.

#### 4.7.2 Enumeration Type Documentation

##### 4.7.2.1 enum `qm_flash_disable_t`

Flash write disable / enable enum.

#### Enumerator

- `QM_FLASH_WRITE_ENABLE`** Flash write enable.  
**`QM_FLASH_WRITE_DISABLE`** Flash write disable.

Definition at line 72 of file `qm_flash.h`.

## 4.7.2.2 enum qm\_flash\_region\_t

Flash region enum.

## Enumerator

- QM\_FLASH\_REGION\_OTP** Flash OTP region.
- QM\_FLASH\_REGION\_SYS** Flash System region.
- QM\_FLASH\_REGION\_DATA** Flash Data region (Quark D2000 only).
- QM\_FLASH\_REGION\_NUM** Total number of flash regions.

Definition at line 60 of file qm\_flash.h.

## 4.7.3 Function Documentation

## 4.7.3.1 int qm\_flash\_mass\_erase ( const qm\_flash\_t flash, const uint8\_t include\_rom )

Perform mass erase.

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

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

## Parameters

|    |                    |  |
|----|--------------------|--|
| in | <i>flash</i>       | Flash controller index.                |
| in | <i>include_rom</i> | If set, it also erases the ROM region. |

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 287 of file qm\_flash.c.

References qm\_flash\_reg\_t::ctrl, and qm\_flash\_reg\_t::flash\_stts.

## 4.7.3.2 int qm\_flash\_page\_erase ( const qm\_flash\_t flash, const qm\_flash\_region\_t region, uint32\_t page\_num )

Erase one page of Flash.

Brownout check is performed before initiating the write.

## Parameters

|    |                 |  |
|----|-----------------|--|
| in | <i>flash</i>    | Flash controller index.                    |
| in | <i>region</i>   | Flash region to address.                   |
| in | <i>page_num</i> | Page within the Flash controller to erase. |

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 250 of file qm\_flash.c.

References qm\_flash\_reg\_t::flash\_stts, qm\_flash\_reg\_t::flash\_wr\_ctrl, QM\_FLASH\_REGION\_DATA, QM\_FLASH\_REGION\_NUM, QM\_FLASH\_REGION\_OTP, QM\_FLASH\_REGION\_SYS, and qm\_flash\_reg\_t::rom\_wr\_ctrl.

**4.7.3.3** int qm\_flash\_page\_update ( const qm\_flash\_t flash, const qm\_flash\_region\_t reg, uint32\_t f\_addr, uint32\_t \*const page\_buf, const uint32\_t \*const data, uint32\_t len )

Write multiple of 4 bytes of data to Flash.

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

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

## Parameters

|    |                 |  |
|----|-----------------|--|
| in | <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 QM_FLASH_PAGE_SIZE 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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 153 of file qm\_flash.c.

References qm\_flash\_reg\_t::flash\_stts, qm\_flash\_reg\_t::flash\_wr\_ctrl, qm\_flash\_reg\_t::flash\_wr\_data, QM\_FLASH\_REGION\_DATA, QM\_FLASH\_REGION\_NUM, QM\_FLASH\_REGION\_OTP, QM\_FLASH\_REGION\_SYS, qm\_flash\_reg\_t::rom\_wr\_ctrl, and qm\_flash\_reg\_t::rom\_wr\_data.

**4.7.3.4** int qm\_flash\_page\_write ( const qm\_flash\_t flash, const qm\_flash\_region\_t region, uint32\_t page\_num, const uint32\_t \* data, uint32\_t len )

Write a 2KB flash page.

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

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

## Parameters

|    |                 |                                   |
|----|-----------------|-----------------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 91 of file qm\_flash.c.

References qm\_flash\_reg\_t::flash\_stts, qm\_flash\_reg\_t::flash\_wr\_ctrl, qm\_flash\_reg\_t::flash\_wr\_data, QM\_FLASH\_REGION\_DATA, QM\_FLASH\_REGION\_NUM, QM\_FLASH\_REGION\_OTP, QM\_FLASH\_REGION\_SYS, qm\_flash\_reg\_t::rom\_wr\_ctrl, and qm\_flash\_reg\_t::rom\_wr\_data.

#### 4.7.3.5 int qm\_flash\_set\_config ( const qm\_flash\_t flash, const qm\_flash\_config\_t \*const cfg )

Configure a Flash controller.

The configuration includes timing and behavioral settings.

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

**Parameters**

|    |              |   |
|----|--------------|---|
| in | <i>flash</i> | Flash controller index.                   |
| in | <i>cfg</i>   | Flash configuration. It must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 16 of file qm\_flash.c.

References qm\_flash\_reg\_t::ctrl, QM\_FLASH\_WRITE\_DISABLE, qm\_flash\_reg\_t::tmg\_ctrl, qm\_flash\_config\_t::us\_count, qm\_flash\_config\_t::wait\_states, and qm\_flash\_config\_t::write\_disable.

#### 4.7.3.6 int qm\_flash\_word\_write ( const qm\_flash\_t flash, const qm\_flash\_region\_t region, uint32\_t f\_addr, const uint32\_t data )

Write 4 bytes of data to Flash.

Brownout check is performed before initiating the write.

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

**Parameters**

|    |              |                         |
|----|--------------|-------------------------|
| in | <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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 39 of file qm\_flash.c.

References qm\_flash\_reg\_t::flash\_stts, qm\_flash\_reg\_t::flash\_wr\_ctrl, qm\_flash\_reg\_t::flash\_wr\_data, QM\_FLASH\_REGION\_DATA, QM\_FLASH\_REGION\_NUM, QM\_FLASH\_REGION\_OTP, QM\_FLASH\_REGION\_SYS, qm\_flash\_reg\_t::rom\_wr\_ctrl, and qm\_flash\_reg\_t::rom\_wr\_data.

## 4.8 FPR

Flash Protection Region control.

### Data Structures

- struct `qm_fpr_config_t`  
*Flash Protection Region configuration structure.*

### Enumerations

- enum `qm_fpr_id_t` { `QM_FPR_0`, `QM_FPR_1`, `QM_FPR_2`, `QM_FPR_3` }  
*FPR register map.*
- enum `qm_fpr_en_t` { `QM_FPR_DISABLE`, `QM_FPR_ENABLE`, `QM_FPR_LOCK_DISABLE`, `QM_FPR_LOCK_ENABLE` }  
*FPR enable type.*
- enum `qm_fpr_viol_mode_t` { `FPR_VIOL_MODE_INTERRUPT` = 0, `FPR_VIOL_MODE_RESET`, `FPR_VIOL_MODE_PROBE` }  
*FPR 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.*

#### 4.8.1 Detailed Description

Flash Protection Region control.

#### 4.8.2 Enumeration Type Documentation

##### 4.8.2.1 enum `qm_flash_region_type_t`

FPR region type.

#### Enumerator

- `QM_MAIN_FLASH_SYSTEM`** System flash region.
- `QM_MAIN_FLASH_DATA`** Data flash region.
- `QM_MAIN_FLASH_NUM`** Number of flash regions.

Definition at line 52 of file `qm_fpr.h`.



#### 4.8.2.2 enum `qm_fpr_en_t`

FPR enable type.

Enumerator

- `QM_FPR_DISABLE`** Disable FPR.
- `QM_FPR_ENABLE`** Enable FPR.
- `QM_FPR_LOCK_DISABLE`** Disable FPR lock.
- `QM_FPR_LOCK_ENABLE`** Enable FPR lock.

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

#### 4.8.2.3 enum `qm_fpr_id_t`

FPR register map.

Enumerator

- `QM_FPR_0`** FPR 0.
- `QM_FPR_1`** FPR 1.
- `QM_FPR_2`** FPR 2.
- `QM_FPR_3`** FPR 3.

Definition at line 22 of file `qm_fpr.h`.

#### 4.8.2.4 enum `qm_fpr_read_allow_t`

FPR read allow type.

Enumerator

- `QM_FPR_HOST_PROCESSOR`** Allow host processor to access flash region.
- `QM_FPR_SENSOR_SUBSYSTEM`** Allow sensor subsystem to access flash region.
- `QM_FPR_DMA`** Allow DMA to access flash region.
- `QM_FPR_OTHER_AGENTS`** Allow other agents to access flash region.

Definition at line 63 of file `qm_fpr.h`.

#### 4.8.2.5 enum `qm_fpr_viol_mode_t`

FPR 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 43 of file `qm_fpr.h`.

### 4.8.3 Function Documentation

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

Configure a Flash controller's Flash Protection Region.

**Parameters**

|    |               |                                       |
|----|---------------|---------------------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 29 of file qm\_fpr.c.

References qm\_fpr\_config\_t::allow\_agents, qm\_fpr\_config\_t::en\_mask, qm\_flash\_reg\_t::fpr\_rd\_cfg, qm\_fpr\_config\_t::low\_bound, QM\_MAIN\_FLASH\_DATA, QM\_MAIN\_FLASH\_NUM, QM\_MAIN\_FLASH\_SYSTEM, and qm\_fpr\_config\_t::up\_bound.

**4.8.3.2** int qm\_fpr\_set\_violation\_policy ( const qm\_fpr\_viol\_mode\_t mode, const qm\_flash\_t flash, qm\_fpr\_callback\_t fpr\_cb, void \* data )

Configure FPR violation behaviour.

**Parameters**

|    |               |   |
|----|---------------|---|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 116 of file qm\_fpr.c.

References FPR\_VIOL\_MODE\_INTERRUPT, FPR\_VIOL\_MODE\_PROBE, qm\_irq\_mask(), and qm\_irq\_unmask().

## 4.9 GPIO

General Purpose IO.

### Data Structures

- struct [qm\\_gpio\\_port\\_config\\_t](#)  
*GPIO port configuration type.*

### Enumerations

- enum [qm\\_gpio\\_state\\_t](#) { [QM\\_GPIO\\_LOW](#), [QM\\_GPIO\\_HIGH](#), [QM\\_GPIO\\_STATE\\_NUM](#) }  
*GPIO pin states.*

### Functions

- int [qm\\_gpio\\_set\\_config](#) (const [qm\\_gpio\\_t](#) gpio, const [qm\\_gpio\\_port\\_config\\_t](#) \*const cfg)  
*Set GPIO port configuration.*
- int [qm\\_gpio\\_read\\_pin](#) (const [qm\\_gpio\\_t](#) gpio, const uint8\_t pin, [qm\\_gpio\\_state\\_t](#) \*const state)  
*Read the current state of a single pin on a given GPIO port.*
- int [qm\\_gpio\\_set\\_pin](#) (const [qm\\_gpio\\_t](#) gpio, const uint8\_t pin)  
*Set a single pin on a given GPIO port.*
- int [qm\\_gpio\\_clear\\_pin](#) (const [qm\\_gpio\\_t](#) gpio, const uint8\_t pin)  
*Clear a single pin on a given GPIO port.*
- int [qm\\_gpio\\_set\\_pin\\_state](#) (const [qm\\_gpio\\_t](#) gpio, const uint8\_t pin, const [qm\\_gpio\\_state\\_t](#) state)  
*Set or clear a single GPIO pin using a state variable.*
- int [qm\\_gpio\\_read\\_port](#) (const [qm\\_gpio\\_t](#) gpio, uint32\_t \*const port)  
*Read the value of every pin on a GPIO port.*
- int [qm\\_gpio\\_write\\_port](#) (const [qm\\_gpio\\_t](#) gpio, const uint32\_t val)  
*Write a value to every pin on a GPIO port.*

#### 4.9.1 Detailed Description

General Purpose IO.

#### 4.9.2 Enumeration Type Documentation

##### 4.9.2.1 enum [qm\\_gpio\\_state\\_t](#)

GPIO pin states.

#### Enumerator

- [QM\\_GPIO\\_LOW](#)** GPIO low state.
- [QM\\_GPIO\\_HIGH](#)** GPIO high state.
- [QM\\_GPIO\\_STATE\\_NUM](#)** Number of GPIO states.

Definition at line 21 of file [qm\\_gpio.h](#).

#### 4.9.3 Function Documentation

##### 4.9.3.1 int [qm\\_gpio\\_clear\\_pin](#) ( const [qm\\_gpio\\_t](#) *gpio*, const uint8\_t *pin* )

Clear a single pin on a given GPIO port.

## Parameters

|    |             |                            |
|----|-------------|----------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 114 of file qm\_gpio.c.

4.9.3.2 int qm\_gpio\_read\_pin ( const qm\_gpio\_t *gpio*, const uint8\_t *pin*, qm\_gpio\_state\_t \*const *state* )

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

## Parameters

|     |              |  |
|-----|--------------|--|
| in  | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 92 of file qm\_gpio.c.

4.9.3.3 int qm\_gpio\_read\_port ( const qm\_gpio\_t *gpio*, uint32\_t \*const *port* )

Read the value of every pin on a GPIO port.

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

## Parameters

|     |             |   |
|-----|-------------|---|
| in  | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 138 of file qm\_gpio.c.

4.9.3.4 int qm\_gpio\_set\_config ( const qm\_gpio\_t *gpio*, const qm\_gpio\_port\_config\_t \*const *cfg* )

Set GPIO port configuration.

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

**Parameters**

|    |             |   |
|----|-------------|---|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 67 of file `qm_gpio.c`.

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

#### 4.9.3.5 `int qm_gpio_set_pin ( const qm_gpio_t gpio, const uint8_t pin )`

Set a single pin on a given GPIO port.

**Parameters**

|    |             |                          |
|----|-------------|--------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 104 of file `qm_gpio.c`.

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

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

**Parameters**

|    |              |   |
|----|--------------|---|
| in | <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 124 of file qm\_gpio.c.

References QM\_GPIO\_STATE\_NUM.

#### 4.9.3.7 int qm\_gpio\_write\_port ( const qm\_gpio\_t gpio, const uint32\_t val )

Write a value to every pin on a GPIO port.

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

##### Parameters

|    |             |                                 |
|----|-------------|---------------------------------|
| in | <i>gpio</i> | GPIO port index.                |
| in | <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 148 of file qm\_gpio.c.

## 4.10 I2C

I2C.

### Data Structures

- struct `qm_i2c_config_t`  
*I2C configuration type.*
- struct `qm_i2c_transfer_t`  
*I2C transfer type.*

### Enumerations

- enum `qm_i2c_addr_t` { `QM_I2C_7_BIT` = 0, `QM_I2C_10_BIT` }  
*QM I2C addressing type.*
- enum `qm_i2c_mode_t` { `QM_I2C_MASTER`, `QM_I2C_SLAVE` }  
*QM I2C master / slave mode type.*
- enum `qm_i2c_speed_t` { `QM_I2C_SPEED_STD` = 1, `QM_I2C_SPEED_FAST` = 2, `QM_I2C_SPEED_FAST_PLUS` = 3 }  
*QM I2C Speed Type.*
- enum `qm_i2c_status_t` {  
`QM_I2C_IDLE` = 0, `QM_I2C_TX_ABRT_7B_ADDR_NOACK` = BIT(0), `QM_I2C_TX_ABRT_10ADDR1_NOACK` = BIT(1), `QM_I2C_TX_ABRT_10ADDR2_NOACK` = BIT(2),  
`QM_I2C_TX_ABRT_TXDATA_NOACK` = BIT(3), `QM_I2C_TX_ABRT_GCALL_NOACK` = BIT(4), `QM_I2C_TX_ABRT_GCALL_READ` = BIT(5), `QM_I2C_TX_ABRT_HS_ACKDET` = BIT(6),  
`QM_I2C_TX_ABRT_SBYTE_ACKDET` = BIT(7), `QM_I2C_TX_ABRT_HS_NORSTRT` = BIT(8), `QM_I2C_TX_ABRT_10B_RD_NORSTRT` = BIT(10), `QM_I2C_TX_ABRT_MASTER_DIS` = BIT(11),  
`QM_I2C_TX_ARB_LOST` = BIT(12), `QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO` = BIT(13), `QM_I2C_TX_ABRT_SLV_ARBLOST` = BIT(14), `QM_I2C_TX_ABRT_SLVRD_INTX` = BIT(15),  
`QM_I2C_TX_ABRT_USER_ABRT` = BIT(16), `QM_I2C_BUSY` = BIT(17) }  
*I2C status type.*

### Functions

- int `qm_i2c_set_config` (const `qm_i2c_t` i2c, const `qm_i2c_config_t` \*const cfg)  
*Set I2C configuration.*
- int `qm_i2c_set_speed` (const `qm_i2c_t` i2c, const `qm_i2c_speed_t` speed, const uint16\_t lo\_cnt, const uint16\_t hi\_cnt)  
*Set I2C speed.*
- int `qm_i2c_get_status` (const `qm_i2c_t` i2c, `qm_i2c_status_t` \*const status)  
*Retrieve I2C status.*
- int `qm_i2c_master_write` (const `qm_i2c_t` i2c, const uint16\_t slave\_addr, const uint8\_t \*const data, uint32\_t len, const bool stop, `qm_i2c_status_t` \*const status)  
*Master write on I2C.*
- int `qm_i2c_master_read` (const `qm_i2c_t` i2c, const uint16\_t slave\_addr, uint8\_t \*const data, uint32\_t len, const bool stop, `qm_i2c_status_t` \*const status)  
*Master read of I2C.*
- int `qm_i2c_master_irq_transfer` (const `qm_i2c_t` i2c, const `qm_i2c_transfer_t` \*const xfer, const uint16\_t slave\_addr)  
*Interrupt based master transfer on I2C.*
- int `qm_i2c_irq_transfer_terminate` (const `qm_i2c_t` i2c)  
*Terminate I2C IRQ transfer.*

- int `qm_i2c_dma_channel_config` (const `qm_i2c_t` i2c, const `qm_dma_t` dma\_controller\_id, const `qm_dma_channel_id_t` channel\_id, const `qm_dma_channel_direction_t` direction)  
*Configure a DMA channel with a specific transfer direction.*
- int `qm_i2c_master_dma_transfer` (const `qm_i2c_t` i2c, `qm_i2c_transfer_t` \*const xfer, const `uint16_t` slave\_addr)  
*Perform a DMA-based transfer on the I2C bus.*
- int `qm_i2c_dma_transfer_terminate` (const `qm_i2c_t` i2c)  
*Terminate any DMA transfer going on on the controller.*

#### 4.10.1 Detailed Description

I2C.

#### 4.10.2 Enumeration Type Documentation

##### 4.10.2.1 enum `qm_i2c_addr_t`

QM I2C addressing type.

Enumerator

**`QM_I2C_7_BIT`** 7-bit mode.

**`QM_I2C_10_BIT`** 10-bit mode.

Definition at line 36 of file `qm_i2c.h`.

##### 4.10.2.2 enum `qm_i2c_mode_t`

QM I2C master / slave mode type.

Enumerator

**`QM_I2C_MASTER`** Master mode.

**`QM_I2C_SLAVE`** Slave mode.

Definition at line 44 of file `qm_i2c.h`.

##### 4.10.2.3 enum `qm_i2c_speed_t`

QM I2C Speed Type.

Enumerator

**`QM_I2C_SPEED_STD`** Standard mode (100 Kbps).

**`QM_I2C_SPEED_FAST`** Fast mode (400 Kbps).

**`QM_I2C_SPEED_FAST_PLUS`** Fast plus mode (1 Mbps).

Definition at line 52 of file `qm_i2c.h`.

##### 4.10.2.4 enum `qm_i2c_status_t`

I2C status type.

Enumerator

**`QM_I2C_IDLE`** Controller idle.



**QM\_I2C\_TX\_ABRT\_7B\_ADDR\_NOACK** 7-bit address noack.  
**QM\_I2C\_TX\_ABRT\_10ADDR1\_NOACK** 10-bit address noack.  
**QM\_I2C\_TX\_ABRT\_10ADDR2\_NOACK** 10-bit second address byte address noack.  
**QM\_I2C\_TX\_ABRT\_TXDATA\_NOACK** Tx data noack.  
**QM\_I2C\_TX\_ABRT\_GCALL\_NOACK** General call noack.  
**QM\_I2C\_TX\_ABRT\_GCALL\_READ** Read after general call.  
**QM\_I2C\_TX\_ABRT\_HS\_ACKDET** High Speed master ID ACK.  
**QM\_I2C\_TX\_ABRT\_SBYTE\_ACKDET** Start ACK.  
**QM\_I2C\_TX\_ABRT\_HS\_NORSTR** High Speed with restart disabled.  
**QM\_I2C\_TX\_ABRT\_10B\_RD\_NORSTR** 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.

Definition at line 61 of file qm\_i2c.h.

#### 4.10.3 Function Documentation

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

Configure a DMA channel with a specific transfer direction.

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

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

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

##### Parameters

|    |                          |   |
|----|--------------------------|---|
| in | <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 | <a href="#">errno</a> for possible error codes. |

Configure a DMA channel with a specific transfer direction.

Definition at line 952 of file qm\_i2c.c.

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

4.10.3.2 `int qm_i2c_dma_transfer_terminate ( const qm_i2c_t i2c )`

Terminate any DMA transfer going on on the controller.

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

## Parameters

|                 |                  |  |
|-----------------|------------------|--|
| <code>in</code> | <code>i2c</code> | Which I2C to terminate transfers from. |
|-----------------|------------------|--|

## Returns

Standard `errno` return type for QMSI.

## Return values

|          |   |
|----------|---|
| 0        | on success.                                     |
| Negative | <a href="#">errno</a> for possible error codes. |

Terminate any DMA transfer going on on the controller.

Definition at line 718 of file qm\_i2c.c.

References `qm_dma_transfer_terminate()`.

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

Retrieve I2C status.

## Parameters

|                  |                     |  |
|------------------|---------------------|--|
| <code>in</code>  | <code>i2c</code>    | Which I2C to read the status of.       |
| <code>out</code> | <code>status</code> | Get i2c status. This must not be NULL. |

## Returns

Standard `errno` return type for QMSI.

## Return values

|          |   |
|----------|---|
| 0        | on success.                                     |
| Negative | <a href="#">errno</a> for possible error codes. |

Definition at line 453 of file qm\_i2c.c.

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

Referenced by `qm_i2c_master_read()`, and `qm_i2c_master_write()`.

#### 4.10.3.4 int qm\_i2c\_irq\_transfer\_terminate ( const qm\_i2c\_t i2c )

Terminate I2C IRQ transfer.

Terminate the current IRQ or DMA transfer on the I2C bus. This will cause the user callback to be called with status QM\_I2C\_TX\_ABRT\_USER\_ABRT.

##### Parameters

|           |            |                             |
|-----------|------------|-----------------------------|
| <i>in</i> | <i>i2c</i> | I2C register block pointer. |
|-----------|------------|-----------------------------|

##### Returns

Standard errno return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 697 of file qm\_i2c.c.

#### 4.10.3.5 int qm\_i2c\_master\_dma\_transfer ( const qm\_i2c\_t i2c, qm\_i2c\_transfer\_t \*const xfer, const uint16\_t slave\_addr )

Perform a DMA-based transfer on the I2C bus.

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

If it's an RX only transfer, it will require 2 channels, one for 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

|           |                   |  |
|-----------|-------------------|--|
| <i>in</i> | <i>i2c</i>        | I2C controller identifier.   |
| <i>in</i> | <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. |
| <i>in</i> | <i>slave_addr</i> | Slave address.   |

##### Returns

Standard errno return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Perform a DMA-based transfer on the I2C bus.

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

Definition at line 1002 of file qm\_i2c.c.

References [qm\\_dma\\_transfer\\_set\\_config\(\)](#), [qm\\_dma\\_transfer\\_start\(\)](#), [qm\\_i2c\\_transfer\\_t::rx](#), [qm\\_i2c\\_transfer\\_t::rx\\_len](#), [qm\\_i2c\\_transfer\\_t::stop](#), [qm\\_i2c\\_transfer\\_t::tx](#), and [qm\\_i2c\\_transfer\\_t::tx\\_len](#).

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

Interrupt based master transfer on I2C.

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

#### Parameters

|    |                   |  |
|----|-------------------|--|
| in | <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 | <a href="#">errno</a> for possible error codes. |

Definition at line 617 of file qm\_i2c.c.

References qm\_i2c\_reg\_t::ic\_intr\_mask, qm\_i2c\_reg\_t::ic\_rx\_tl, qm\_i2c\_reg\_t::ic\_tar, qm\_i2c\_reg\_t::ic\_tx\_tl, and qm\_i2c\_transfer\_t::rx\_len.

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

Master read of I2C.

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

#### Parameters

|     |                   |  |
|-----|-------------------|--|
| in  | <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 tx.                        |
| out | <i>status</i>     | Get i2c status.  |

#### Returns

Standard errno return type for QMSI.

#### Return values

|          |   |
|----------|---|
| 0        | on success.                                     |
| Negative | <a href="#">errno</a> for possible error codes. |

Definition at line 544 of file qm\_i2c.c.

References qm\_i2c\_reg\_t::ic\_clr\_tx\_abrt, qm\_i2c\_reg\_t::ic\_data\_cmd, qm\_i2c\_reg\_t::ic\_raw\_intr\_stat, qm\_i2c\_reg\_t::ic\_status, qm\_i2c\_reg\_t::ic\_tar, qm\_i2c\_reg\_t::ic\_tx\_abrt\_source, and qm\_i2c\_get\_status().

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

Master write on I2C.

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

**Parameters**

|     |                   |   |
|-----|-------------------|---|
| in  | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 474 of file qm\_i2c.c.

References qm\_i2c\_reg\_t::ic\_clr\_tx\_abrt, qm\_i2c\_reg\_t::ic\_data\_cmd, qm\_i2c\_reg\_t::ic\_status, qm\_i2c\_reg\_t::ic\_tar, qm\_i2c\_reg\_t::ic\_tx\_abrt\_source, and qm\_i2c\_get\_status().

#### 4.10.3.9 int qm\_i2c\_set\_config ( const qm\_i2c\_t i2c, const qm\_i2c\_config\_t \*const cfg )

Set I2C configuration.

**Parameters**

|     |            |   |
|-----|------------|---|
| in  | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 308 of file qm\_i2c.c.

References qm\_i2c\_config\_t::address\_mode, qm\_i2c\_reg\_t::ic\_con, qm\_i2c\_reg\_t::ic\_fs\_scl\_hcnt, qm\_i2c\_reg\_t::ic\_fs\_scl\_lcnc, 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\_lcnc, qm\_i2c\_config\_t::mode, QM\_I2C\_MASTER, QM\_I2C\_SLAVE, QM\_I2C\_SPEED\_FAST, QM\_I2C\_SPEED\_FAST\_PLUS, QM\_I2C\_SPEED\_STD, qm\_i2c\_config\_t::slave\_addr, and qm\_i2c\_config\_t::speed.

#### 4.10.3.10 int qm\_i2c\_set\_speed ( const qm\_i2c\_t i2c, const qm\_i2c\_speed\_t speed, const uint16\_t lo\_cnt, const uint16\_t hi\_cnt )

Set I2C speed.

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

**Parameters**

|    |               |   |
|----|---------------|---|
| in | <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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 417 of file qm\_i2c.c.

References qm\_i2c\_reg\_t::ic\_con, qm\_i2c\_reg\_t::ic\_fs\_scl\_hcnt, qm\_i2c\_reg\_t::ic\_fs\_scl\_lcnt, qm\_i2c\_reg\_t::ic\_fs\_spklen, qm\_i2c\_reg\_t::ic\_ss\_scl\_hcnt, qm\_i2c\_reg\_t::ic\_ss\_scl\_lcnt, QM\_I2C\_SPEED\_FAST, QM\_I2C\_SPEED\_FAST\_PLUS, and QM\_I2C\_SPEED\_STD.

## 4.11 Identification

Identification functions for Quark Microcontrollers.

### Functions

- `uint32_t qm_soc_id` (void)  
*Get Quark SoC identification number.*
- `uint32_t qm_soc_version` (void)  
*Get Quark SoC version number.*

### 4.11.1 Detailed Description

Identification functions for Quark Microcontrollers.

### 4.11.2 Function Documentation

#### 4.11.2.1 `uint32_t qm_soc_id ( void )`

Get Quark SoC identification number.

#### Returns

`uint32_t` SoC identifier number.

Definition at line 10 of file `qm_identification.c`.

#### 4.11.2.2 `uint32_t qm_soc_version ( void )`

Get Quark SoC version number.

#### Returns

`uint32_t` SoC version number.

Definition at line 21 of file `qm_identification.c`.



## 4.12 Initialisation

Initialisation and reset.

### Enumerations

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

### Functions

- void `qm_soc_reset` (`qm_soc_reset_t` `reset_type`)  
*Reset the SoC.*

### 4.12.1 Detailed Description

Initialisation and reset.

### 4.12.2 Enumeration Type Documentation

#### 4.12.2.1 enum `qm_soc_reset_t`

Reset Mode type.

#### Enumerator

**`QM_WARM_RESET`** Warm reset.

**`QM_COLD_RESET`** Cold reset.

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

### 4.12.3 Function Documentation

#### 4.12.3.1 void `qm_soc_reset` ( `qm_soc_reset_t` `reset_type` )

Reset the SoC.

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

#### Parameters

|                 |                         |                                       |
|-----------------|-------------------------|---------------------------------------|
| <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)  
*Enable interrupt delivery for the SoC.*
- void [qm\\_irq\\_disable](#) (void)  
*Disable interrupt delivery for the SoC.*
- void [qm\\_irq\\_unmask](#) (uint32\_t irq)  
*Unmask a given interrupt line.*
- void [qm\\_irq\\_mask](#) (uint32\_t irq)  
*Mask a given interrupt line.*
- void [qm\\_int\\_vector\\_request](#) (uint32\_t vector, [qm\\_isr\\_t](#) isr)  
*Request an interrupt vector and register Interrupt Service Routine to it.*

#### 4.13.1 Detailed Description

Interrupt driver.

#### 4.13.2 Function Documentation

##### 4.13.2.1 static \_\_inline\_\_ void qm\_int\_vector\_request ( uint32\_t vector, qm\_isr\_t isr )

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

##### Parameters

|    |               |                               |
|----|---------------|-------------------------------|
| in | <i>vector</i> | Vector number.                |
| in | <i>isr</i>    | ISR to register to given IRQ. |

Definition at line 98 of file qm\_interrupt.h.

##### 4.13.2.2 void qm\_irq\_mask ( uint32\_t irq )

Mask a given interrupt line.

##### Parameters

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

Definition at line 52 of file qm\_interrupt.c.

References [qm\\_ss\\_irq\\_mask\(\)](#).

Referenced by [qm\\_fpr\\_set\\_violation\\_policy\(\)](#), [qm\\_mbox\\_ch\\_set\\_config\(\)](#), and [qm\\_mpr\\_set\\_violation\\_policy\(\)](#).

##### 4.13.2.3 void qm\_irq\_unmask ( uint32\_t irq )

Unmask a given interrupt line.

**Parameters**

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

Definition at line 66 of file `qm_interrupt.c`.

References `qm_ss_irq_unmask()`.

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

## 4.14 ISR

Interrupt Service Routines.

### Functions

- [QM\\_ISR\\_DECLARE \(qm\\_adc\\_0\\_isr\)](#)  
*ISR for ADC 0 convert and calibration interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_adc\\_pwr\\_0\\_isr\)](#)  
*ISR for ADC 0 change mode interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_aonpt\\_isr\\_0\)](#)  
*ISR for Always-on Periodic Timer 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ac\\_isr\)](#)  
*ISR for Analog Comparator 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_err\)](#)  
*ISR for DMA error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_0\)](#)  
*ISR for DMA channel 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_1\)](#)  
*ISR for DMA channel 1 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_2\)](#)  
*ISR for DMA channel 2 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_3\)](#)  
*ISR for DMA channel 3 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_4\)](#)  
*ISR for DMA channel 4 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_5\)](#)  
*ISR for DMA channel 5 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_6\)](#)  
*ISR for DMA channel 6 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_dma\\_0\\_isr\\_7\)](#)  
*ISR for DMA 0 channel 7 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_fpr\\_isr\\_0\)](#)  
*ISR for FPR 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_fpr\\_isr\\_1\)](#)  
*ISR for FPR 1 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_gpio\\_isr\\_0\)](#)  
*ISR for GPIO 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_aon\\_gpio\\_isr\\_0\)](#)  
*ISR for AON GPIO 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_i2c\\_0\\_isr\)](#)  
*ISR for I2C 0 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_i2c\\_1\\_isr\)](#)  
*ISR for I2C 1 interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_mbox\\_isr\)](#)  
*ISR for Mailbox interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_mpr\\_isr\)](#)  
*ISR for Memory Protection Region interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_pic\\_timer\\_isr\)](#)  
*ISR for PIC Timer interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_pwm\\_isr\\_0\)](#)

*ISR for PWM 0 interrupt.*

- [QM\\_ISR\\_DECLARE](#) (qm\_rtc\_isr\_0)

*ISR for RTC 0 interrupt.*

- [QM\\_ISR\\_DECLARE](#) (qm\_spi\_master\_0\_isr)

*ISR for SPI Master 0 interrupt.*

- [QM\\_ISR\\_DECLARE](#) (qm\_spi\_master\_1\_isr)

*ISR for SPI Master 1 interrupt.*

- [QM\\_ISR\\_DECLARE](#) (qm\_uart\_0\_isr)

*ISR for UART 0 interrupt.*

- [QM\\_ISR\\_DECLARE](#) (qm\_uart\_1\_isr)

*ISR for UART 1 interrupt.*

- [QM\\_ISR\\_DECLARE](#) (qm\_wdt\_isr\_0)

*ISR for WDT 0 interrupt.*

#### 4.14.1 Detailed Description

Interrupt Service Routines.

#### 4.14.2 Function Documentation

##### 4.14.2.1 [QM\\_ISR\\_DECLARE](#) ( qm\_adc\_0\_isr )

ISR for ADC 0 convert and calibration interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 168 of file qm\_adc.c.

##### 4.14.2.2 [QM\\_ISR\\_DECLARE](#) ( qm\_adc\_pwr\_0\_isr )

ISR for ADC 0 change mode interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 176 of file qm\_adc.c.

##### 4.14.2.3 [QM\\_ISR\\_DECLARE](#) ( qm\_aonpt\_isr\_0 )

ISR for Always-on Periodic Timer 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 35 of file qm\_aon\_counters.c.

#### 4.14.2.4 QM\_ISR\_DECLARE ( qm\_ac\_isr )

ISR for Analog Comparator 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 10 of file qm\_comparator.c.

#### 4.14.2.5 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_err )

ISR for DMA error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 87 of file qm\_dma.c.

References QM\_DMA\_0.

#### 4.14.2.6 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_0 )

ISR for DMA channel 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 93 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_0.

#### 4.14.2.7 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_1 )

ISR for DMA channel 1 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 99 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_1.

#### 4.14.2.8 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_2 )

ISR for DMA channel 2 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 106 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_2.

#### 4.14.2.9 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_3 )

ISR for DMA channel 3 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 112 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_3.

#### 4.14.2.10 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_4 )

ISR for DMA channel 4 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 118 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_4.

#### 4.14.2.11 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_5 )

ISR for DMA channel 5 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 124 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_5.

#### 4.14.2.12 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_6 )

ISR for DMA channel 6 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 130 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_6.

#### 4.14.2.13 QM\_ISR\_DECLARE ( qm\_dma\_0\_isr\_7 )

ISR for DMA 0 channel 7 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 136 of file qm\_dma.c.

References QM\_DMA\_0, and QM\_DMA\_CHANNEL\_7.

#### 4.14.2.14 QM\_ISR\_DECLARE ( qm\_fpr\_isr\_0 )

ISR for FPR 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 11 of file qm\_fpr.c.

#### 4.14.2.15 QM\_ISR\_DECLARE ( qm\_fpr\_isr\_1 )

ISR for FPR 1 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 20 of file qm\_fpr.c.

#### 4.14.2.16 QM\_ISR\_DECLARE ( qm\_gpio\_isr\_0 )

ISR for GPIO 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 53 of file qm\_gpio.c.

#### 4.14.2.17 QM\_ISR\_DECLARE ( qm\_aon\_gpio\_isr\_0 )

ISR for AON GPIO 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 60 of file qm\_gpio.c.

#### 4.14.2.18 QM\_ISR\_DECLARE ( qm\_i2c\_0\_isr )

ISR for I2C 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 265 of file qm\_i2c.c.

#### 4.14.2.19 QM\_ISR\_DECLARE ( qm\_i2c\_1\_isr )

ISR for I2C 1 interrupt.

This function needs to be registered with



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

if IRQ based transfers are used.

Definition at line 272 of file qm\_i2c.c.

#### 4.14.2.20 QM\_ISR\_DECLARE ( qm\_mbox\_isr )

ISR for Mailbox interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 30 of file qm\_mailbox.c.

References qm\_mailbox\_t::ch\_sts, and QM\_MBOX\_CH\_INT.

#### 4.14.2.21 QM\_ISR\_DECLARE ( qm\_mpr\_isr )

ISR for Memory Protection Region interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 13 of file qm\_mpr.c.

#### 4.14.2.22 QM\_ISR\_DECLARE ( qm\_pic\_timer\_isr )

ISR for PIC Timer interrupt.

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

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

if IRQ based transfers are used.

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

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

if IRQ based transfers are used.

Definition at line 27 of file qm\_pic\_timer.c.

#### 4.14.2.23 QM\_ISR\_DECLARE ( qm\_pwm\_isr\_0 )

ISR for PWM 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 11 of file qm\_pwm.c.

**4.14.2.24 QM\_ISR\_DECLARE ( qm\_rtc\_isr\_0 )**

ISR for RTC 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 10 of file qm\_rtc.c.

**4.14.2.25 QM\_ISR\_DECLARE ( qm\_spi\_master\_0\_isr )**

ISR for SPI Master 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 446 of file qm\_spi.c.

**4.14.2.26 QM\_ISR\_DECLARE ( qm\_spi\_master\_1\_isr )**

ISR for SPI Master 1 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 453 of file qm\_spi.c.

**4.14.2.27 QM\_ISR\_DECLARE ( qm\_uart\_0\_isr )**

ISR for UART 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 203 of file qm\_uart.c.

**4.14.2.28 QM\_ISR\_DECLARE ( qm\_uart\_1\_isr )**

ISR for UART 1 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 209 of file qm\_uart.c.

**4.14.2.29 QM\_ISR\_DECLARE ( qm\_wdt\_isr\_0 )**

ISR for WDT 0 interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 13 of file qm\_wdt.c.

## 4.15 Mailbox

Mailbox driver.

### Data Structures

- struct [qm\\_mbox\\_msg\\_t](#)  
*Definition of the mailbox message.*

### Typedefs

- typedef void(\* [qm\\_mbox\\_callback\\_t](#))(void \*data)  
*Definition of the mailbox callback function prototype.*

### Enumerations

- enum [qm\\_mbox\\_ch\\_status\\_t](#) { [QM\\_MBOX\\_CH\\_IDLE](#) = 0, [QM\\_MBOX\\_CH\\_DATA](#) = BIT(0), [QM\\_MBOX\\_CH\\_INT](#) = BIT(1), [QM\\_MBOX\\_CH\\_STATUS\\_MASK](#) = BIT(1) | BIT(0) }  
*Mailbox status.*
- enum [qm\\_mbox\\_ch\\_t](#) { [QM\\_MBOX\\_CH\\_0](#) = 0, [QM\\_MBOX\\_CH\\_1](#), [QM\\_MBOX\\_CH\\_2](#), [QM\\_MBOX\\_CH\\_3](#), [QM\\_MBOX\\_CH\\_4](#), [QM\\_MBOX\\_CH\\_5](#), [QM\\_MBOX\\_CH\\_6](#), [QM\\_MBOX\\_CH\\_7](#), [QM\\_MBOX\\_CH\\_NUM](#) }  
*Mailbox channel.*
- enum [qm\\_mbox\\_payload\\_t](#) { [QM\\_MBOX\\_PAYLOAD\\_0](#) = 0, [QM\\_MBOX\\_PAYLOAD\\_1](#), [QM\\_MBOX\\_PAYLOAD\\_2](#), [QM\\_MBOX\\_PAYLOAD\\_3](#), [QM\\_MBOX\\_PAYLOAD\\_NUM](#) }  
*mailbox message pay-load index values.*

### Functions

- int [qm\\_mbox\\_ch\\_set\\_config](#) (const [qm\\_mbox\\_ch\\_t](#) mbox\_ch, [qm\\_mbox\\_callback\\_t](#) mpr\_cb, void \*cb\_data, const bool irq\_en)  
*Set the mailbox channel configuration.*
- int [qm\\_mbox\\_ch\\_write](#) (const [qm\\_mbox\\_ch\\_t](#) mbox\_ch, const [qm\\_mbox\\_msg\\_t](#) \*const msg)  
*Write to a specified mailbox channel.*
- int [qm\\_mbox\\_ch\\_read](#) (const [qm\\_mbox\\_ch\\_t](#) mbox\_ch, [qm\\_mbox\\_msg\\_t](#) \*const msg)  
*Read specified mailbox channel.*
- int [qm\\_mbox\\_ch\\_get\\_status](#) (const [qm\\_mbox\\_ch\\_t](#) mbox\_ch, [qm\\_mbox\\_ch\\_status\\_t](#) \*const status)  
*Retrieve the specified mailbox channel status.*
- int [qm\\_mbox\\_ch\\_data\\_ack](#) (const [qm\\_mbox\\_ch\\_t](#) mbox\_ch)  
*Acknowledge the data arrival.*

#### 4.15.1 Detailed Description

Mailbox driver.

#### 4.15.2 Typedef Documentation

##### 4.15.2.1 typedef void(\* [qm\\_mbox\\_callback\\_t](#))(void \*data)

Definition of the mailbox callback function prototype.

## Parameters

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

Definition at line 69 of file qm\_mailbox.h.

## 4.15.3 Enumeration Type Documentation

## 4.15.3.1 enum qm\_mbox\_ch\_status\_t

Mailbox status.

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

## Enumerator

**QM\_MBOX\_CH\_IDLE** No interrupt pending nor any data to consume.

**QM\_MBOX\_CH\_DATA** Message has not been consumed.

**QM\_MBOX\_CH\_INT** Channel interrupt pending.

**QM\_MBOX\_CH\_STATUS\_MASK** Status mask.

Definition at line 23 of file qm\_mailbox.h.

## 4.15.3.2 enum qm\_mbox\_ch\_t

Mailbox channel.

## Enumerator

**QM\_MBOX\_CH\_0** Channel 0.

**QM\_MBOX\_CH\_1** Channel 1.

**QM\_MBOX\_CH\_2** Channel 2.

**QM\_MBOX\_CH\_3** Channel 3.

**QM\_MBOX\_CH\_4** Channel 4.

**QM\_MBOX\_CH\_5** Channel 5.

**QM\_MBOX\_CH\_6** Channel 6.

**QM\_MBOX\_CH\_7** Channel 7.

**QM\_MBOX\_CH\_NUM** Mailbox number of channels.

Definition at line 34 of file qm\_mailbox.h.

## 4.15.3.3 enum qm\_mbox\_payload\_t

mailbox message pay-load index values.

## Enumerator

**QM\_MBOX\_PAYLOAD\_0** Payload index value 0.

**QM\_MBOX\_PAYLOAD\_1** Payload index value 1.

**QM\_MBOX\_PAYLOAD\_2** Payload index value 2.

**QM\_MBOX\_PAYLOAD\_3** Payload index value 3.

**QM\_MBOX\_PAYLOAD\_NUM** Numbers of payloads.

Definition at line 49 of file qm\_mailbox.h.

#### 4.15.4 Function Documentation

##### 4.15.4.1 `int qm_mbox_ch_data_ack ( const qm_mbox_ch_t mbox_ch )`

Acknowledge the data arrival.

**Parameters**

|           |                |   |
|-----------|----------------|---|
| <i>in</i> | <i>mbox_ch</i> | Mailbox identifier to retrieve the status from. |
|-----------|----------------|---|

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 161 of file qm\_mailbox.c.

References QM\_MBOX\_CH\_DATA, and QM\_MBOX\_CH\_NUM.

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

Retrieve the specified mailbox channel status.

**Parameters**

|            |                |   |
|------------|----------------|---|
| <i>in</i>  | <i>mbox_ch</i> | Mailbox identifier to retrieve the status from. |
| <i>out</i> | <i>status</i>  | Mailbox status. This must not be NULL.          |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 151 of file qm\_mailbox.c.

References qm\_mailbox\_t::ch\_sts, QM\_MBOX\_CH\_NUM, and QM\_MBOX\_CH\_STATUS\_MASK.

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

Read specified mailbox channel.

**Parameters**

|            |                |  |
|------------|----------------|--|
| <i>in</i>  | <i>mbox_ch</i> | Mailbox channel identifier.  |
| <i>out</i> | <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**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 128 of file qm\_mailbox.c.

References qm\_mailbox\_t::ch\_ctrl, qm\_mailbox\_t::ch\_data, qm\_mailbox\_t::ch\_sts, qm\_mbox\_msg\_t::ctrl, qm\_mbox\_msg\_t::data, QM\_MBOX\_CH\_DATA, QM\_MBOX\_PAYLOAD\_0, QM\_MBOX\_PAYLOAD\_1, QM\_MBOX\_PAYLOAD\_2, and QM\_MBOX\_PAYLOAD\_3.

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

Set the mailbox channel configuration.

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

#### Parameters

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

#### Returns

Standard errno return type for QMSI.

#### Return values

|                 |   |
|-----------------|---|
| <code>0</code>  | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 68 of file `qm_mailbox.c`.

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

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

Write to a specified mailbox channel.

#### Parameters

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

#### Returns

Standard errno return type for QMSI.

#### Return values

|                 |   |
|-----------------|---|
| <code>0</code>  | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 106 of file `qm_mailbox.c`.

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



## 4.16 MPR

Memory Protection Region control for Quark Microcontrollers.

### Data Structures

- struct [qm\\_mpr\\_config\\_t](#)  
*SRAM Memory Protection Region configuration type.*

### Functions

- int [qm\\_mpr\\_set\\_config](#) (const qm\_mpr\_id\_t id, const [qm\\_mpr\\_config\\_t](#) \*const cfg)  
*Configure SRAM controller's Memory Protection Region.*
- int [qm\\_mpr\\_set\\_violation\\_policy](#) (const qm\_mpr\_viol\_mode\_t mode, qm\_mpr\_callback\_t callback\_fn, void \*data)  
*Configure MPR violation behaviour.*

#### 4.16.1 Detailed Description

Memory Protection Region control for Quark Microcontrollers.

#### 4.16.2 Function Documentation

##### 4.16.2.1 int qm\_mpr\_set\_config ( const qm\_mpr\_id\_t id, const qm\_mpr\_config\_t \*const cfg )

Configure SRAM controller's Memory Protection Region.

#### Parameters

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

#### Returns

int 0 on success, error code otherwise.

Definition at line 23 of file qm\_mpr.c.

References [qm\\_mpr\\_config\\_t::agent\\_read\\_en\\_mask](#), [qm\\_mpr\\_config\\_t::agent\\_write\\_en\\_mask](#), [qm\\_mpr\\_config\\_t::en\\_lock\\_mask](#), [qm\\_mpr\\_config\\_t::low\\_bound](#), and [qm\\_mpr\\_config\\_t::up\\_bound](#).

##### 4.16.2.2 int qm\_mpr\_set\_violation\_policy ( const qm\_mpr\_viol\_mode\_t mode, qm\_mpr\_callback\_t callback\_fn, void \* data )

Configure MPR violation behaviour.

#### Parameters

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

#### Returns

int 0 on success, error code otherwise.

Definition at line 91 of file qm\_mpr.c.

References [qm\\_irq\\_mask\(\)](#), and [qm\\_irq\\_unmask\(\)](#).

## 4.17 PIC Timer

PIC timer.

### Data Structures

- struct `qm_pic_timer_config_t`  
*PIC timer configuration type.*

### Enumerations

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

### Functions

- int `qm_pic_timer_set_config` (const `qm_pic_timer_config_t` \*const `cfg`)  
*Set the PIC timer configuration.*
- int `qm_pic_timer_set` (const uint32\_t `count`)  
*Set the current count value of the PIC timer.*
- int `qm_pic_timer_get` (uint32\_t \*const `count`)  
*Get the current count value of the PIC timer.*

#### 4.17.1 Detailed Description

PIC timer.

#### 4.17.2 Enumeration Type Documentation

##### 4.17.2.1 enum `qm_pic_timer_mode_t`

PIC timer mode type.

#### Enumerator

- `QM_PIC_TIMER_MODE_ONE_SHOT`** One shot mode.  
**`QM_PIC_TIMER_MODE_PERIODIC`** Periodic mode.

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

#### 4.17.3 Function Documentation

##### 4.17.3.1 int `qm_pic_timer_get` ( uint32\_t \*const `count` )

Get the current count value of the PIC timer.

#### Parameters

---

|            |              |  |
|------------|--------------|--|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 76 of file qm\_pic\_timer.c.

**4.17.3.2 int qm\_pic\_timer\_set ( const uint32\_t count )**

Set the current count value of the PIC timer.

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

**Parameters**

|           |              |                               |
|-----------|--------------|-------------------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 69 of file qm\_pic\_timer.c.

**4.17.3.3 int qm\_pic\_timer\_set\_config ( const qm\_pic\_timer\_config\_t \*const cfg )**

Set the PIC timer configuration.

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

**Parameters**

|           |            |   |
|-----------|------------|---|
| <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> | <a href="#">errno</a> 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> | <a href="#">errno</a> 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> | <a href="#">errno</a> 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> | <a href="#">errno</a> for possible error codes. |

Definition at line 37 of file qm\_pinmux.c.

References QM\_PMUX\_FN\_3.

#### 4.18.3.4 int qm\_pmux\_set\_slew ( const qm\_pin\_id\_t pin, const qm\_pmux\_slew\_t slew )

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

##### Parameters

|    |             |                                     |
|----|-------------|-------------------------------------|
| in | <i>pin</i>  | which pin to configure.             |
| in | <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 51 of file qm\_pinmux.c.

References QM\_PMUX\_SLEW\_NUM.



## 4.19 PWM / Timer

Pulse width modulation and Timer driver.

### Data Structures

- struct `qm_pwm_config_t`  
*PWM / Timer configuration type.*

### Enumerations

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

### Functions

- int `qm_pwm_set_config` (const `qm_pwm_t` pwm, const `qm_pwm_id_t` id, const `qm_pwm_config_t` \*const cfg)  
*Change the configuration of a PWM channel.*
- int `qm_pwm_set` (const `qm_pwm_t` pwm, const `qm_pwm_id_t` id, const uint32\_t lo\_count, const uint32\_t hi\_count)  
*Set the next period values of a PWM channel.*
- int `qm_pwm_get` (const `qm_pwm_t` pwm, const `qm_pwm_id_t` id, uint32\_t \*const lo\_count, uint32\_t \*const hi\_count)  
*Get the current period values of a PWM channel.*
- int `qm_pwm_start` (const `qm_pwm_t` pwm, const `qm_pwm_id_t` id)  
*Start a PWM/timer channel.*
- int `qm_pwm_stop` (const `qm_pwm_t` pwm, const `qm_pwm_id_t` id)  
*Stop a PWM/timer channel.*

### 4.19.1 Detailed Description

Pulse width modulation and Timer driver.

### 4.19.2 Enumeration Type Documentation

#### 4.19.2.1 enum `qm_pwm_mode_t`

PWM operating mode type.

#### Enumerator

**`QM_PWM_MODE_TIMER_FREE_RUNNING`** Timer: free running mode.

**`QM_PWM_MODE_TIMER_COUNT`** Timer: Counter mode.

**`QM_PWM_MODE_PWM`** Pwm mode.

Definition at line 25 of file `qm_pwm.h`.

### 4.19.3 Function Documentation

#### 4.19.3.1 int `qm_pwm_get` ( const `qm_pwm_t` pwm, const `qm_pwm_id_t` id, uint32\_t \*const lo\_count, uint32\_t \*const hi\_count )

Get the current period values of a PWM channel.

**Parameters**

|     |                 |   |
|-----|-----------------|---|
| in  | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 87 of file qm\_pwm.c.

```
4.19.3.2 int qm_pwm_set ( const qm_pwm_t pwm, const qm_pwm_id_t id, const uint32_t lo_count, const uint32_t hi_count )
```

Set the next period values of a PWM channel.

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

Set PWM period counts.

**Parameters**

|    |                 |  |
|----|-----------------|--|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 68 of file qm\_pwm.c.

References QM\_PWM\_MODE\_PWM.

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

Change the configuration of a PWM channel.

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

Set PWM channel configuration.

**Parameters**

|           |            |   |
|-----------|------------|---|
| <i>in</i> | <i>pwm</i> | Which PWM module to configure.                    |
| <i>in</i> | <i>id</i>  | PWM channel id to configure.                      |
| <i>in</i> | <i>cfg</i> | New configuration for PWM. This must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 44 of file qm\_pwm.c.

References qm\_pwm\_config\_t::callback, qm\_pwm\_config\_t::callback\_data, qm\_pwm\_config\_t::hi\_count, qm\_pwm\_config\_t::lo\_count, qm\_pwm\_config\_t::mask\_interrupt, qm\_pwm\_config\_t::mode, and QM\_PWM\_MODE\_PWM.

#### 4.19.3.4 int qm\_pwm\_start ( const qm\_pwm\_t *pwm*, const qm\_pwm\_id\_t *id* )

Start a PWM/timer channel.

**Parameters**

|           |            |                                |
|-----------|------------|--------------------------------|
| <i>in</i> | <i>pwm</i> | Which PWM block the PWM is in. |
| <i>in</i> | <i>id</i>  | PWM channel id to start.       |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 24 of file qm\_pwm.c.

#### 4.19.3.5 int qm\_pwm\_stop ( const qm\_pwm\_t *pwm*, const qm\_pwm\_id\_t *id* )

Stop a PWM/timer channel.

**Parameters**

|           |            |                                |
|-----------|------------|--------------------------------|
| <i>in</i> | <i>pwm</i> | Which PWM block the PWM is in. |
| <i>in</i> | <i>id</i>  | PWM channel id to stop.        |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 34 of file qm\_pwm.c.

## 4.20 RTC

Real Time clock.

### Data Structures

- struct [qm\\_rtc\\_config\\_t](#)  
*RTC configuration type.*

### Functions

- int [qm\\_rtc\\_set\\_config](#) (const [qm\\_rtc\\_t](#) rtc, const [qm\\_rtc\\_config\\_t](#) \*const cfg)  
*Set RTC configuration.*
- int [qm\\_rtc\\_set\\_alarm](#) (const [qm\\_rtc\\_t](#) rtc, const uint32\_t alarm\_val)  
*Set Alarm value.*

#### 4.20.1 Detailed Description

Real Time clock.

#### 4.20.2 Function Documentation

##### 4.20.2.1 int [qm\\_rtc\\_set\\_alarm](#) ( const [qm\\_rtc\\_t](#) rtc, const uint32\_t alarm\_val )

Set Alarm value.

Set a new RTC alarm value after an alarm, that has been set using the [qm\\_rtc\\_set\\_config](#) function, has expired and a new alarm value is required.

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

#### Parameters

|    |                  |                        |
|----|------------------|------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 77 of file [qm\\_rtc.c](#).

Referenced by [qm\\_rtc\\_set\\_config](#)().

##### 4.20.2.2 int [qm\\_rtc\\_set\\_config](#) ( const [qm\\_rtc\\_t](#) rtc, const [qm\\_rtc\\_config\\_t](#) \*const cfg )

Set RTC configuration.

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

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

**Parameters**

|           |            |   |
|-----------|------------|---|
| <i>in</i> | <i>rtc</i> | RTC index.                                    |
| <i>in</i> | <i>cfg</i> | New RTC configuration. This must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 51 of file `qm_rtc.c`.

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

## 4.21 SPI

SPI peripheral driver for Quark Microcontrollers.

### Data Structures

- struct `qm_spi_config_t`  
*SPI configuration type.*
- struct `qm_spi_async_transfer_t`  
*SPI IRQ transfer type.*
- struct `qm_spi_transfer_t`  
*SPI transfer type.*

### Enumerations

- enum `qm_spi_frame_size_t` {  
`QM_SPI_FRAME_SIZE_4_BIT = 3, QM_SPI_FRAME_SIZE_5_BIT, QM_SPI_FRAME_SIZE_6_BIT, QM_SPI_FRAME_SIZE_7_BIT,`  
`QM_SPI_FRAME_SIZE_8_BIT, QM_SPI_FRAME_SIZE_9_BIT, QM_SPI_FRAME_SIZE_10_BIT, QM_SPI_FRAME_SIZE_11_BIT,`  
`QM_SPI_FRAME_SIZE_12_BIT, QM_SPI_FRAME_SIZE_13_BIT, QM_SPI_FRAME_SIZE_14_BIT, QM_SPI_FRAME_SIZE_15_BIT,`  
`QM_SPI_FRAME_SIZE_16_BIT, QM_SPI_FRAME_SIZE_17_BIT, QM_SPI_FRAME_SIZE_18_BIT, QM_SPI_FRAME_SIZE_19_BIT,`  
`QM_SPI_FRAME_SIZE_20_BIT, QM_SPI_FRAME_SIZE_21_BIT, QM_SPI_FRAME_SIZE_22_BIT, QM_SPI_FRAME_SIZE_23_BIT,`  
`QM_SPI_FRAME_SIZE_24_BIT, QM_SPI_FRAME_SIZE_25_BIT, QM_SPI_FRAME_SIZE_26_BIT, QM_SPI_FRAME_SIZE_27_BIT,`  
`QM_SPI_FRAME_SIZE_28_BIT, QM_SPI_FRAME_SIZE_29_BIT, QM_SPI_FRAME_SIZE_30_BIT, QM_SPI_FRAME_SIZE_31_BIT,`  
`QM_SPI_FRAME_SIZE_32_BIT` }  
*QM SPI frame size type.*
- enum `qm_spi_tmode_t` { `QM_SPI_TMOD_TX_RX, QM_SPI_TMOD_TX, QM_SPI_TMOD_RX, QM_SPI_TMOD_EEPROM_READ` }  
*SPI transfer mode type.*
- enum `qm_spi_bmode_t` { `QM_SPI_BMODE_0, QM_SPI_BMODE_1, QM_SPI_BMODE_2, QM_SPI_BMODE_3` }  
*SPI bus mode type.*
- enum `qm_spi_slave_select_t` {  
`QM_SPI_SS_DISABLED = 0, QM_SPI_SS_0 = BIT(0), QM_SPI_SS_1 = BIT(1), QM_SPI_SS_2 = BIT(2),`  
`QM_SPI_SS_3 = BIT(3)` }  
*SPI slave select type.*
- enum `qm_spi_status_t` { `QM_SPI_IDLE, QM_SPI_BUSY, QM_SPI_RX_OVERFLOW` }  
*SPI status.*

### Functions

- int `qm_spi_set_config` (const `qm_spi_t` spi, const `qm_spi_config_t` \*const cfg)  
*Set SPI configuration.*
- int `qm_spi_slave_select` (const `qm_spi_t` spi, const `qm_spi_slave_select_t` ss)  
*Select which slave to perform SPI transmissions on.*
- int `qm_spi_get_status` (const `qm_spi_t` spi, `qm_spi_status_t` \*const status)  
*Get SPI bus status.*

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

#### 4.21.1 Detailed Description

SPI peripheral driver for Quark Microcontrollers.

#### 4.21.2 Enumeration Type Documentation

##### 4.21.2.1 enum `qm_spi_bmode_t`

SPI bus mode type.

##### Enumerator

- `QM_SPI_BMODE_0`** Clock Polarity = 0, Clock Phase = 0.
- `QM_SPI_BMODE_1`** Clock Polarity = 0, Clock Phase = 1.
- `QM_SPI_BMODE_2`** Clock Polarity = 1, Clock Phase = 0.
- `QM_SPI_BMODE_3`** Clock Polarity = 1, Clock Phase = 1.

Definition at line 67 of file `qm_spi.h`.

##### 4.21.2.2 enum `qm_spi_frame_size_t`

QM SPI frame size type.

##### Enumerator

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



**QM\_SPI\_FRAME\_SIZE\_16\_BIT** 16 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_17\_BIT** 17 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_18\_BIT** 18 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_19\_BIT** 19 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_20\_BIT** 20 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_21\_BIT** 21 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_22\_BIT** 22 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_23\_BIT** 23 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_24\_BIT** 24 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_25\_BIT** 25 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_26\_BIT** 26 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_27\_BIT** 27 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_28\_BIT** 28 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_29\_BIT** 29 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_30\_BIT** 30 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_31\_BIT** 31 bit frame.  
**QM\_SPI\_FRAME\_SIZE\_32\_BIT** 32 bit frame.

Definition at line 22 of file qm\_spi.h.

#### 4.21.2.3 enum qm\_spi\_slave\_select\_t

SPI slave select type.

Slave selects can be combined by logical OR if multiple slaves are selected during one transfer. Setting only QM\_SPI\_SS\_DISABLED prevents the controller from starting the transfer.

##### Enumerator

**QM\_SPI\_SS\_DISABLED** Slave select disable.  
**QM\_SPI\_SS\_0** Slave Select 0.  
**QM\_SPI\_SS\_1** Slave Select 1.  
**QM\_SPI\_SS\_2** Slave Select 2.  
**QM\_SPI\_SS\_3** Slave Select 3.

Definition at line 81 of file qm\_spi.h.

#### 4.21.2.4 enum qm\_spi\_status\_t

SPI status.

##### Enumerator

**QM\_SPI\_IDLE** SPI device is not in use.  
**QM\_SPI\_BUSY** SPI device is busy.  
**QM\_SPI\_RX\_OVERFLOW** RX transfer has overflowed.

Definition at line 92 of file qm\_spi.h.

## 4.21.2.5 enum qm\_spi\_tmode\_t

SPI transfer mode type.

## Enumerator

- QM\_SPI\_TMOD\_TX\_RX** Transmit & Receive.
- QM\_SPI\_TMOD\_TX** Transmit Only.
- QM\_SPI\_TMOD\_RX** Receive Only.
- QM\_SPI\_TMOD\_EEPROM\_READ** EEPROM Read.

Definition at line 57 of file qm\_spi.h.

## 4.21.3 Function Documentation

## 4.21.3.1 int qm\_spi\_dma\_channel\_config ( const qm\_spi\_t spi, const qm\_dma\_t dma\_ctrl\_id, const qm\_dma\_channel\_id\_t dma\_channel\_id, const qm\_dma\_channel\_direction\_t dma\_channel\_direction )

Configure a DMA channel with a specific transfer direction.

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

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

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

Note that [qm\\_dma\\_init\(\)](#) must first be called before configuring a channel.

## Parameters

|    |                                 |   |
|----|---------------------------------|---|
| in | <i>spi</i>                      | SPI controller 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 (TX transfer) or QM_DMA_PERIPHERAL_TO_MEMORY (RX transfer). |

## Returns

Standard errno return type for QMSI.

## Return values

|          |   |
|----------|---|
| 0        | on success.                                     |
| Negative | <a href="#">errno</a> for possible error codes. |

Definition at line 565 of file qm\_spi.c.

References [qm\\_dma\\_channel\\_config\\_t::callback\\_context](#), [qm\\_dma\\_channel\\_config\\_t::channel\\_direction](#), [qm\\_dma\\_channel\\_config\\_t::client\\_callback](#), [qm\\_dma\\_channel\\_config\\_t::destination\\_burst\\_length](#), [qm\\_dma\\_channel\\_config\\_t::destination\\_transfer\\_width](#), [DMA\\_HW\\_IF\\_SPI\\_MASTER\\_0\\_RX](#), [DMA\\_HW\\_IF\\_SPI\\_MASTER\\_0\\_TX](#), [DMA\\_HW\\_IF\\_SPI\\_MASTER\\_1\\_RX](#), [DMA\\_HW\\_IF\\_SPI\\_MASTER\\_1\\_TX](#), [qm\\_dma\\_channel\\_config\\_t::handshake\\_interface](#), [qm\\_dma\\_channel\\_config\\_t::handshake\\_polarity](#), [QM\\_DMA\\_CHANNEL\\_NUM](#), [qm\\_dma\\_channel\\_set\\_config\(\)](#), [QM\\_DMA\\_HANDSHAKE\\_POLARITY\\_HIGH](#), [QM\\_DMA\\_MEMORY\\_TO\\_PERIPHERAL](#), [QM\\_DMA\\_NUM](#), [QM\\_DMA\\_PERIPHERAL\\_TO\\_MEMORY](#), [QM\\_DMA\\_TRANS\\_WIDTH\\_16](#), [QM\\_DMA\\_TRANS\\_WIDTH\\_32](#), [QM\\_DMA\\_TRANS\\_WIDTH\\_8](#), [qm\\_dma\\_channel\\_config\\_t::source\\_burst\\_length](#), and [qm\\_dma\\_channel\\_config\\_t::source\\_transfer\\_width](#).

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

Perform a DMA-based transfer on the SPI bus.

If transfer mode is full duplex (QM\_SPI\_TMOD\_TX\_RX), then `tx_len` and `rx_len` must be equal and neither of both callbacks can be NULL. Similarly, for transmit-only transfers (QM\_SPI\_TMOD\_TX) `rx_len` must be 0 and `tx_callback` cannot be NULL, while for receive-only transfers (QM\_SPI\_TMOD\_RX) `tx_len` must be 0 and `rx_callback` cannot be NULL. Transfer length is limited to 4KB.

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

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

##### Parameters

|    |             |   |
|----|-------------|---|
| in | <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. |

##### Returns

Standard `errno` return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 664 of file `qm_spi.c`.

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

#### 4.21.3.3 `int qm_spi_dma_transfer_terminate ( const qm_spi_t spi )`

Terminate the current DMA transfer on the SPI bus.

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

##### Parameters

|    |            |                            |
|----|------------|----------------------------|
| in | <i>spi</i> | SPI controller identifier. |
|----|------------|----------------------------|

##### Returns

Standard `errno` return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 797 of file `qm_spi.c`.

References `QM_DMA_CHANNEL_NUM`, and `qm_dma_transfer_terminate()`.

Referenced by `qm_spi_dma_transfer()`.

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

Get SPI bus status.

Retrieve SPI bus status. Return QM\_SPI\_BUSY if transmitting data or data Tx FIFO not empty.

##### Parameters

|     |               |  |
|-----|---------------|--|
| in  | <i>spi</i>    | Which SPI to read the status of.       |
| out | <i>status</i> | Get spi status. This must not be null. |

##### Returns

Standard errno return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 290 of file qm\_spi.c.

References QM\_SPI\_BUSY, QM\_SPI\_IDLE, QM\_SPI\_RX\_OVERFLOW, qm\_spi\_reg\_t::risr, and qm\_spi\_reg\_t::sr.

#### 4.21.3.5 `int qm_spi_irq_transfer ( const qm_spi_t spi, const qm_spi_async_transfer_t *const xfer )`

Interrupt based transfer on SPI.

Perform an interrupt based transfer on the SPI bus. The function will replenish/empty TX/RX FIFOs on SPI empty/full interrupts. If transfer mode is full duplex (QM\_SPI\_TMOD\_TX\_RX), then tx\_len and rx\_len must be equal. For transmit-only transfers (QM\_SPI\_TMOD\_TX) rx\_len must be 0 while for receive-only transfers (QM\_SPI\_TMOD\_RX) tx\_len must be 0.

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

##### Parameters

|    |             |   |
|----|-------------|---|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Enable SPI Device

Definition at line 390 of file qm\_spi.c.

References qm\_spi\_reg\_t::ctrlr1, qm\_spi\_reg\_t::imr, QM\_SPI\_TMOD\_EEPROM\_READ, QM\_SPI\_TMOD\_RX, QM\_SPI\_TMOD\_TX, QM\_SPI\_TMOD\_TX\_RX, qm\_spi\_async\_transfer\_t::rx\_len, qm\_spi\_reg\_t::rxflr, qm\_spi\_reg\_t::ssiernr, 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**

|           |            |   |
|-----------|------------|---|
| <i>in</i> | <i>spi</i> | Which SPI to cancel the current transfer. |
|-----------|------------|---|

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

**Disable SPI device**

Definition at line 460 of file qm\_spi.c.

References qm\_spi\_async\_transfer\_t::callback, qm\_spi\_async\_transfer\_t::callback\_data, qm\_spi\_reg\_t::imr, QM\_SPI\_IDLE, QM\_SPI\_TMOD\_TX, QM\_SPI\_TMOD\_TX\_RX, and qm\_spi\_reg\_t::ssierr.

#### 4.21.3.7 int qm\_spi\_set\_config ( const qm\_spi\_t spi, const qm\_spi\_config\_t \*const cfg )

Set SPI configuration.

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

**Parameters**

|           |            |   |
|-----------|------------|---|
| <i>in</i> | <i>spi</i> | Which SPI module to configure.                    |
| <i>in</i> | <i>cfg</i> | New configuration for SPI. This must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 249 of file qm\_spi.c.

References qm\_spi\_reg\_t::baudr, qm\_spi\_config\_t::bus\_mode, qm\_spi\_config\_t::clk\_divider, qm\_spi\_reg\_t::ctrlr0, qm\_spi\_config\_t::frame\_size, and qm\_spi\_config\_t::transfer\_mode.

#### 4.21.3.8 int qm\_spi\_slave\_select ( const qm\_spi\_t spi, const qm\_spi\_slave\_select\_t ss )

Select which slave to perform SPI transmissions on.

**Parameters**

|           |            |   |
|-----------|------------|---|
| <i>in</i> | <i>spi</i> | Which SPI module to configure.                              |
| <i>in</i> | <i>ss</i>  | Which slave select line to enable when doing transmissions. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 278 of file qm\_spi.c.

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

Multi-frame read / write on SPI.

Perform a multi-frame read/write on the SPI bus. This is a blocking synchronous call. If the SPI is currently in use, the function will wait until the SPI is free before beginning the transfer. If transfer mode is full duplex (QM\_SPI\_TMOD\_TX\_RX), then tx\_len and rx\_len must be equal. Similarly, for transmit-only transfers (QM\_SPI\_TMOD\_TX) rx\_len must be 0, while for receive-only transfers (QM\_SPI\_TMOD\_RX) tx\_len must be 0.

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

#### Parameters

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

#### Returns

Standard errno return type for QMSI.

#### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 310 of file qm\_spi.c.

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

## 4.22 UART

UART peripheral driver.

### Data Structures

- struct `qm_uart_config_t`  
*UART configuration type.*
- struct `qm_uart_transfer_t`  
*UART asynchronous transfer structure.*

### Enumerations

- enum `qm_uart_lc_t` {  
`QM_UART_LC_5N1 = 0x00, QM_UART_LC_5N1_5 = 0x04, QM_UART_LC_5E1 = 0x18, QM_UART_LC_5E1_5 = 0x1c,`  
`QM_UART_LC_5O1 = 0x08, QM_UART_LC_5O1_5 = 0x0c, QM_UART_LC_6N1 = 0x01, QM_UART_LC_6N2 = 0x05,`  
`QM_UART_LC_6E1 = 0x19, QM_UART_LC_6E2 = 0x1d, QM_UART_LC_6O1 = 0x09, QM_UART_LC_6O2 = 0x0d,`  
`QM_UART_LC_7N1 = 0x02, QM_UART_LC_7N2 = 0x06, QM_UART_LC_7E1 = 0x1a, QM_UART_LC_7E2 = 0x1e,`  
`QM_UART_LC_7O1 = 0x0a, QM_UART_LC_7O2 = 0x0e, QM_UART_LC_8N1 = 0x03, QM_UART_LC_8N2 = 0x07,`  
`QM_UART_LC_8E1 = 0x1b, QM_UART_LC_8E2 = 0x1f, QM_UART_LC_8O1 = 0x0b, QM_UART_LC_8O2 = 0x0f }`  
*UART Line control.*
- enum `qm_uart_status_t` {  
`QM_UART_IDLE = 0, QM_UART_RX_OE = BIT(1), QM_UART_RX_PE = BIT(2), QM_UART_RX_FE = BIT(3),`  
`QM_UART_RX_BI = BIT(4), QM_UART_TX_BUSY = BIT(5), QM_UART_RX_BUSY = BIT(6), QM_UART_TX_NFULL = BIT(7),`  
`QM_UART_RX_NEMPTY = BIT(8) }`  
*UART Status type.*

### Functions

- int `qm_uart_set_config` (const `qm_uart_t` uart, const `qm_uart_config_t` \*const cfg)  
*Set UART configuration.*
- int `qm_uart_get_status` (const `qm_uart_t` uart, `qm_uart_status_t` \*const status)  
*Get UART bus status.*
- int `qm_uart_write` (const `qm_uart_t` uart, const `uint8_t` data)  
*UART character data write.*
- int `qm_uart_read` (const `qm_uart_t` uart, `uint8_t` \*const data, `qm_uart_status_t` \*const status)  
*UART character data read.*
- int `qm_uart_write_non_block` (const `qm_uart_t` uart, const `uint8_t` data)  
*UART character data write.*
- int `qm_uart_read_non_block` (const `qm_uart_t` uart, `uint8_t` \*const data)  
*UART character data read.*
- int `qm_uart_write_buffer` (const `qm_uart_t` uart, const `uint8_t` \*const data, `uint32_t` len)  
*UART multi-byte data write.*
- int `qm_uart_irq_write` (const `qm_uart_t` uart, const `qm_uart_transfer_t` \*const xfer)  
*Interrupt based TX on UART.*

- int `qm_uart_irq_read` (const `qm_uart_t` uart, const `qm_uart_transfer_t` \*const xfer)  
*Interrupt based RX on UART.*
- int `qm_uart_irq_write_terminate` (const `qm_uart_t` uart)  
*Terminate UART IRQ TX transfer.*
- int `qm_uart_irq_read_terminate` (const `qm_uart_t` uart)  
*Terminate UART IRQ RX transfer.*
- int `qm_uart_dma_channel_config` (const `qm_uart_t` uart, const `qm_dma_t` dma\_ctrl\_id, const `qm_dma_channel_id_t` dma\_channel\_id, const `qm_dma_channel_direction_t` dma\_channel\_direction)  
*Configure a DMA channel with a specific transfer direction.*
- int `qm_uart_dma_write` (const `qm_uart_t` uart, const `qm_uart_transfer_t` \*const xfer)  
*Perform a DMA-based TX transfer on the UART bus.*
- int `qm_uart_dma_read` (const `qm_uart_t` uart, const `qm_uart_transfer_t` \*const xfer)  
*Perform a DMA-based RX transfer on the UART bus.*
- int `qm_uart_dma_write_terminate` (const `qm_uart_t` uart)  
*Terminate the current DMA TX transfer on the UART bus.*
- int `qm_uart_dma_read_terminate` (const `qm_uart_t` uart)  
*Terminate the current DMA RX transfer on the UART bus.*

#### 4.22.1 Detailed Description

UART peripheral driver.

#### 4.22.2 Enumeration Type Documentation

##### 4.22.2.1 enum `qm_uart_lc_t`

UART Line control.

##### Enumerator

- `QM_UART_LC_5N1`** 5 data bits, no parity, 1 stop bit.
- `QM_UART_LC_5N1_5`** 5 data bits, no parity, 1.5 stop bits.
- `QM_UART_LC_5E1`** 5 data bits, even parity, 1 stop bit.
- `QM_UART_LC_5E1_5`** 5 data bits, even par., 1.5 stop bits.
- `QM_UART_LC_5O1`** 5 data bits, odd parity, 1 stop bit.
- `QM_UART_LC_5O1_5`** 5 data bits, odd parity, 1.5 stop bits.
- `QM_UART_LC_6N1`** 6 data bits, no parity, 1 stop bit.
- `QM_UART_LC_6N2`** 6 data bits, no parity, 2 stop bits.
- `QM_UART_LC_6E1`** 6 data bits, even parity, 1 stop bit.
- `QM_UART_LC_6E2`** 6 data bits, even parity, 2 stop bits.
- `QM_UART_LC_6O1`** 6 data bits, odd parity, 1 stop bit.
- `QM_UART_LC_6O2`** 6 data bits, odd parity, 2 stop bits.
- `QM_UART_LC_7N1`** 7 data bits, no parity, 1 stop bit.
- `QM_UART_LC_7N2`** 7 data bits, no parity, 2 stop bits.
- `QM_UART_LC_7E1`** 7 data bits, even parity, 1 stop bit.
- `QM_UART_LC_7E2`** 7 data bits, even parity, 2 stop bits.
- `QM_UART_LC_7O1`** 7 data bits, odd parity, 1 stop bit.
- `QM_UART_LC_7O2`** 7 data bits, odd parity, 2 stop bits.
- `QM_UART_LC_8N1`** 8 data bits, no parity, 1 stop bit.



**QM\_UART\_LC\_8N2** 8 data bits, no parity, 2 stop bits.

**QM\_UART\_LC\_8E1** 8 data bits, even parity, 1 stop bit.

**QM\_UART\_LC\_8E2** 8 data bits, even parity, 2 stop bits.

**QM\_UART\_LC\_8O1** 8 data bits, odd parity, 1 stop bit.

**QM\_UART\_LC\_8O2** 8 data bits, odd parity, 2 stop bits.

Definition at line 116 of file `qm_uart.h`.

#### 4.22.2.2 enum `qm_uart_status_t`

UART Status type.

##### Enumerator

**QM\_UART\_IDLE** IDLE.

**QM\_UART\_RX\_OE** Receiver overrun.

**QM\_UART\_RX\_PE** Parity error.

**QM\_UART\_RX\_FE** Framing error.

**QM\_UART\_RX\_BI** Break interrupt.

**QM\_UART\_TX\_BUSY** TX Busy flag.

**QM\_UART\_RX\_BUSY** RX Busy flag.

**QM\_UART\_TX\_NFULL** TX FIFO not full.

**QM\_UART\_RX\_NEMPTY** RX FIFO not empty.

Definition at line 146 of file `qm_uart.h`.

#### 4.22.3 Function Documentation

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

Configure a DMA channel with a specific transfer direction.

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

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

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

##### Parameters

|    |                                    |  |
|----|------------------------------------|--|
| in | <code>uart</code>                  | UART index.  |
| in | <code>dma_ctrl_id</code>           | DMA controller identifier.   |
| in | <code>dma_channel_id</code>        | DMA channel identifier.  |
| in | <code>dma_channel_direction</code> | DMA channel direction, either <code>QM_DMA_MEMORY_TO_PERIPHERAL</code> (write transfer) or <code>QM_DMA_PERIPHERAL_TO_MEMORY</code> (read transfer). |

##### Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 526 of file `qm_uart.c`.

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

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

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

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

## Parameters

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

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 620 of file `qm_uart.c`.

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

#### 4.22.3.3 `int qm_uart_dma_read_terminate ( const qm_uart_t uart )`

Terminate the current DMA RX transfer on the UART bus.

This will cause the relevant callbacks to be called.

## Parameters

|           |             |             |
|-----------|-------------|-------------|
| <i>in</i> | <i>uart</i> | UART index. |
|-----------|-------------|-------------|

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 663 of file qm\_uart.c.

References `qm_dma_transfer_terminate()`.

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

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

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

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

## Parameters

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

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 586 of file qm\_uart.c.

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

#### 4.22.3.5 `int qm_uart_dma_write_terminate ( const qm_uart_t uart )`

Terminate the current DMA TX transfer on the UART bus.

This will cause the relevant callbacks to be called.

## Parameters

|           |             |             |
|-----------|-------------|-------------|
| <i>in</i> | <i>uart</i> | UART index. |
|-----------|-------------|-------------|

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 653 of file qm\_uart.c.

References `qm_dma_transfer_terminate()`.

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

Get UART bus status.

Retrieve UART interface status. Return QM\_UART\_BUSY if transmitting data; QM\_UART\_IDLE if available for transfer QM\_UART\_TX\_ERROR if an error has occurred in transmission.

#### Parameters

|            |               |  |
|------------|---------------|--|
| <i>in</i>  | <i>uart</i>   | Which UART to read the status of.            |
| <i>out</i> | <i>status</i> | UART specific status. This must not be NULL. |

#### Returns

Standard errno return type for QMSI.

#### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 249 of file qm\_uart.c.

References qm\_uart\_reg\_t::lsr, QM\_UART\_RX\_BUSY, QM\_UART\_TX\_BUSY, and qm\_uart\_reg\_t::scr.

#### 4.22.3.7 int qm\_uart\_irq\_read ( const qm\_uart\_t *uart*, const qm\_uart\_transfer\_t \*const *xfer* )

Interrupt based RX on UART.

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

#### Parameters

|           |             |  |
|-----------|-------------|--|
| <i>in</i> | <i>uart</i> | UART index.  |
| <i>in</i> | <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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 385 of file qm\_uart.c.

References qm\_uart\_reg\_t::ier\_dlh, and qm\_uart\_reg\_t::iir\_fcr.

#### 4.22.3.8 int qm\_uart\_irq\_read\_terminate ( const qm\_uart\_t *uart* )

Terminate UART IRQ RX transfer.

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

#### Parameters

|           |             |             |
|-----------|-------------|-------------|
| <i>in</i> | <i>uart</i> | UART index. |
|-----------|-------------|-------------|

#### Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 425 of file qm\_uart.c.

References qm\_uart\_transfer\_t::callback, qm\_uart\_transfer\_t::callback\_data, qm\_uart\_reg\_t::ier\_dlh, and QM\_UART\_IDLE.

#### 4.22.3.9 int qm\_uart\_irq\_write ( const qm\_uart\_t *uart*, const qm\_uart\_transfer\_t \*const *xfer* )

Interrupt based TX on UART.

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

## Parameters

|           |             |   |
|-----------|-------------|---|
| <i>in</i> | <i>uart</i> | UART index.   |
| <i>in</i> | <i>xfer</i> | 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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 364 of file qm\_uart.c.

References qm\_uart\_reg\_t::ier\_dlh, and qm\_uart\_reg\_t::iir\_fcr.

#### 4.22.3.10 int qm\_uart\_irq\_write\_terminate ( const qm\_uart\_t *uart* )

Terminate UART IRQ TX transfer.

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

## Parameters

|           |             |             |
|-----------|-------------|-------------|
| <i>in</i> | <i>uart</i> | UART index. |
|-----------|-------------|-------------|

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 408 of file qm\_uart.c.

References qm\_uart\_transfer\_t::callback, qm\_uart\_transfer\_t::callback\_data, qm\_uart\_reg\_t::ier\_dlh, and QM\_UART\_IDLE.

#### 4.22.3.11 int qm\_uart\_read ( const qm\_uart\_t *uart*, uint8\_t \*const *data*, qm\_uart\_status\_t \*const *status* )

UART character data read.

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

**Parameters**

|     |               |  |
|-----|---------------|--|
| in  | <i>uart</i>   | UART index.                                    |
| 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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 292 of file qm\_uart.c.

References qm\_uart\_reg\_t::lsr, and qm\_uart\_reg\_t::rbr\_thr\_dll.

4.22.3.12 int qm\_uart\_read\_non\_block ( const qm\_uart\_t *uart*, uint8\_t \*const *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 index.                            |
| out | <i>data</i> | Character read. This must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 327 of file qm\_uart.c.

References qm\_uart\_reg\_t::rbr\_thr\_dll.

4.22.3.13 int qm\_uart\_set\_config ( const qm\_uart\_t *uart*, const qm\_uart\_config\_t \*const *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**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 215 of file `qm_uart.c`.

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

#### 4.2.2.3.14 `int qm_uart_write ( const qm_uart_t uart, const uint8_t data )`

UART character data write.

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

##### Parameters

|           |             |                        |
|-----------|-------------|------------------------|
| <i>in</i> | <i>uart</i> | UART index.            |
| <i>in</i> | <i>data</i> | Data to write to UART. |

##### Returns

Standard `errno` return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 276 of file `qm_uart.c`.

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

#### 4.2.2.3.15 `int qm_uart_write_buffer ( const qm_uart_t uart, const uint8_t *const data, uint32_t len )`

UART multi-byte data write.

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

##### Parameters

|           |             |   |
|-----------|-------------|---|
| <i>in</i> | <i>uart</i> | UART index.                                   |
| <i>in</i> | <i>data</i> | Data to write to UART. This must not be NULL. |
| <i>in</i> | <i>len</i>  | Length of data to write to UART.              |

##### Returns

Standard `errno` return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 338 of file `qm_uart.c`.

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

#### 4.2.2.3.16 `int qm_uart_write_non_block ( const qm_uart_t uart, const uint8_t data )`

UART character data write.

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

**Parameters**

|           |             |                        |
|-----------|-------------|------------------------|
| <i>in</i> | <i>uart</i> | UART index.            |
| <i>in</i> | <i>data</i> | Data to write to UART. |

**Returns**

Standard `errno` return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 316 of file `qm_uart.c`.

References `qm_uart_reg_t::rbr_thr_dll`.



## 4.23 Version

Version number functions for API.

### Functions

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

### 4.23.1 Detailed Description

Version number functions for API.

### 4.23.2 Function Documentation

#### 4.23.2.1 `uint32_t qm_ver_rom ( void )`

Get the ROM version number.

Reads the ROM version information from flash and returns it.

### Returns

`uint32_t` ROM version.

Definition at line 7 of file `qm_version.c`.

## 4.24 WDT

Watchdog timer.

### Data Structures

- struct `qm_wdt_config_t`  
*QM WDT configuration type.*

### Enumerations

- enum `qm_wdt_mode_t` { `QM_WDT_MODE_RESET`, `QM_WDT_MODE_INTERRUPT_RESET` }  
*WDT Mode type.*
- enum `qm_wdt_clock_timeout_cycles_t` {  
`QM_WDT_2_POW_16_CYCLES`, `QM_WDT_2_POW_17_CYCLES`, `QM_WDT_2_POW_18_CYCLES`, `QM_WDT_2_POW_19_CYCLES`,  
`QM_WDT_2_POW_20_CYCLES`, `QM_WDT_2_POW_21_CYCLES`, `QM_WDT_2_POW_22_CYCLES`, `QM_WDT_2_POW_23_CYCLES`,  
`QM_WDT_2_POW_24_CYCLES`, `QM_WDT_2_POW_25_CYCLES`, `QM_WDT_2_POW_26_CYCLES`, `QM_WDT_2_POW_27_CYCLES`,  
`QM_WDT_2_POW_28_CYCLES`, `QM_WDT_2_POW_29_CYCLES`, `QM_WDT_2_POW_30_CYCLES`, `QM_WDT_2_POW_31_CYCLES` }  
*WDT clock cycles for timeout type.*

### Functions

- int `qm_wdt_start` (const `qm_wdt_t` wdt)  
*Start WDT.*
- int `qm_wdt_set_config` (const `qm_wdt_t` wdt, const `qm_wdt_config_t` \*const cfg)  
*Set configuration of WDT module.*
- int `qm_wdt_reload` (const `qm_wdt_t` wdt)  
*Reload the WDT counter.*

#### 4.24.1 Detailed Description

Watchdog timer.

#### 4.24.2 Enumeration Type Documentation

##### 4.24.2.1 enum `qm_wdt_clock_timeout_cycles_t`

WDT clock cycles for timeout type.

This value is a power of 2.

#### Enumerator

- `QM_WDT_2_POW_16_CYCLES`** 16 clock cycles timeout.
- `QM_WDT_2_POW_17_CYCLES`** 17 clock cycles timeout.
- `QM_WDT_2_POW_18_CYCLES`** 18 clock cycles timeout.
- `QM_WDT_2_POW_19_CYCLES`** 19 clock cycles timeout.
- `QM_WDT_2_POW_20_CYCLES`** 20 clock cycles timeout.
- `QM_WDT_2_POW_21_CYCLES`** 21 clock cycles timeout.

**QM\_WDT\_2\_POW\_22\_CYCLES** 22 clock cycles timeout.  
**QM\_WDT\_2\_POW\_23\_CYCLES** 23 clock cycles timeout.  
**QM\_WDT\_2\_POW\_24\_CYCLES** 24 clock cycles timeout.  
**QM\_WDT\_2\_POW\_25\_CYCLES** 25 clock cycles timeout.  
**QM\_WDT\_2\_POW\_26\_CYCLES** 26 clock cycles timeout.  
**QM\_WDT\_2\_POW\_27\_CYCLES** 27 clock cycles timeout.  
**QM\_WDT\_2\_POW\_28\_CYCLES** 28 clock cycles timeout.  
**QM\_WDT\_2\_POW\_29\_CYCLES** 29 clock cycles timeout.  
**QM\_WDT\_2\_POW\_30\_CYCLES** 30 clock cycles timeout.  
**QM\_WDT\_2\_POW\_31\_CYCLES** 31 clock cycles timeout.

Definition at line 49 of file qm\_wdt.h.

#### 4.24.2.2 enum qm\_wdt\_mode\_t

WDT Mode type.

#### Enumerator

**QM\_WDT\_MODE\_RESET** Watchdog Reset Response Mode. The watchdog will request a SoC Warm Reset on a timeout.  
**QM\_WDT\_MODE\_INTERRUPT\_RESET** Watchdog Interrupt Reset Response Mode. The watchdog will generate an interrupt on first timeout. If interrupt has not been cleared by the second timeout the watchdog will then request a SoC Warm Reset.

Definition at line 30 of file qm\_wdt.h.

#### 4.24.3 Function Documentation

##### 4.24.3.1 int qm\_wdt\_reload ( const qm\_wdt\_t wdt )

Reload the WDT counter.

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

#### Parameters

|           |            |            |
|-----------|------------|------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 64 of file qm\_wdt.c.

Referenced by qm\_wdt\_set\_config().

##### 4.24.3.2 int qm\_wdt\_set\_config ( const qm\_wdt\_t wdt, const qm\_wdt\_config\_t \*const cfg )

Set configuration of WDT module.

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

**Parameters**

|           |            |  |
|-----------|------------|--|
| <i>in</i> | <i>wdt</i> | WDT index.   |
| <i>in</i> | <i>cfg</i> | New configuration for WDT. This must not be NULL. If QM_WDT_MODE_INTERRUPT_RESET mode is set, the 'callback' cannot be null. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 45 of file qm\_wdt.c.

References qm\_wdt\_config\_t::callback, qm\_wdt\_config\_t::callback\_data, qm\_wdt\_config\_t::mode, QM\_WDT\_MODE\_INTERRUPT\_RESET, qm\_wdt\_reload(), and qm\_wdt\_config\_t::timeout.

**4.24.3.3 int qm\_wdt\_start ( const qm\_wdt\_t wdt )**

Start WDT.

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

**Parameters**

|           |            |            |
|-----------|------------|------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 21 of file qm\_wdt.c.

References SOCW\_EVENT\_REGISTER, and SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL.

## 4.25 SOC\_WATCH

SoC Watch (Energy Analyzer).

### Enumerations

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

*Power profiling events enumeration.*

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

*Register ID enumeration.*

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

### Functions

- void `soc_watch_log_event` (`soc_watch_event_t` event\_id, `uintptr_t` ev\_data)  
*Log a power profile event.*
- void `soc_watch_log_app_event` (`soc_watch_event_t` event\_id, `uint8_t` ev\_subtype, `uintptr_t` ev\_data)  
*Log an application event via the power profile logger.*

#### 4.25.1 Detailed Description

SoC Watch (Energy Analyzer).

## 4.25.2 Enumeration Type Documentation

## 4.25.2.1 enum soc\_watch\_event\_t

Power profiling events enumeration.

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

## Enumerator

**SOCW\_EVENT\_HALT** CPU Halt.  
**SOCW\_EVENT\_INTERRUPT** CPU interrupt generated.  
**SOCW\_EVENT\_SLEEP** Sleep mode entered.  
**SOCW\_EVENT\_REGISTER** SOC register altered.  
**SOCW\_EVENT\_APP** Application-defined event.  
**SOCW\_EVENT\_MAX** End of events sentinel.

Definition at line 36 of file soc\_watch.h.

## 4.25.2.2 enum soc\_watch\_reg\_t

Register ID enumeration.

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

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

## Enumerator

**SOCW\_REG\_OSC0\_CFG1** 0x000 OSC0\_CFG1 register.  
**SOCW\_REG\_CCU\_LP\_CLK\_CTL** 0x02C Clock Control register.  
**SOCW\_REG\_CCU\_SYS\_CLK\_CTL** 0x038 System Clock Control.  
**SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL** 0x018 Perip Clock Gate Ctl.  
**SOCW\_REG\_CCU\_EXT\_CLK\_CTL** 0x024 CCU Ext Clock Gate Ctl.  
**SOCW\_REG\_CMP\_PWR** 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.  
**SOCW\_REG\_PMUX\_PULLUP** 0x900 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_SLEW** 0x910 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_IN\_EN** 0x920 Pin Mux In Enable.  
**SOCW\_REG\_MAX** Register enum sentinel.  
**SOCW\_REG\_OSC0\_CFG1** 0x000 OSC0\_CFG1 register.  
**SOCW\_REG\_CCU\_LP\_CLK\_CTL** 0x02C Clock Control register.  
**SOCW\_REG\_CCU\_SYS\_CLK\_CTL** 0x038 System Clock Control.  
**SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL** 0x018 Perip Clock Gate Ctl.  
**SOCW\_REG\_CCU\_SS\_PERIPH\_CLK\_GATE\_CTL** 0x0028 SS PCL Gate Ctl.  
**SOCW\_REG\_CCU\_EXT\_CLK\_CTL** 0x024 CCU Ext Clock Gate Ctl.  
**SOCW\_REG\_CMP\_PWR** 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.  
**SOCW\_REG\_SLP\_CFG** 0x550 Sleep Configuration.  
**SOCW\_REG\_PMUX\_PULLUP0** 0x900 Pin Mux Pullup.

**SOCW\_REG\_PMUX\_PULLUP1** 0x904 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_PULLUP2** 0x908 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_PULLUP3** 0x90c Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_SLEW0** 0x910 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_SLEW1** 0x914 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_SLEW2** 0x918 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_SLEW3** 0x91c Pin Mux Slew.  
**SOCW\_REG\_PMUX\_IN\_EN0** 0x920 Pin Mux In Enable.  
**SOCW\_REG\_PMUX\_IN\_EN1** 0x924 Pin Mux In Enable.  
**SOCW\_REG\_PMUX\_IN\_EN2** 0x928 Pin Mux In Enable.  
**SOCW\_REG\_PMUX\_IN\_EN3** 0x92c Pin Mux In Enable.  
**SOCW\_REG\_MAX** Register enum sentinel.

Definition at line 61 of file soc\_watch.h.

#### 4.25.2.3 enum soc\_watch\_reg\_t

Enumerator

**SOCW\_REG\_OSC0\_CFG1** 0x000 OSC0\_CFG1 register.  
**SOCW\_REG\_CCU\_LP\_CLK\_CTL** 0x02C Clock Control register.  
**SOCW\_REG\_CCU\_SYS\_CLK\_CTL** 0x038 System Clock Control.  
**SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL** 0x018 Perip Clock Gate Ctl.  
**SOCW\_REG\_CCU\_EXT\_CLK\_CTL** 0x024 CCU Ext Clock Gate Ctl.  
**SOCW\_REG\_CMP\_PWR** 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.  
**SOCW\_REG\_PMUX\_PULLUP** 0x900 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_SLEW** 0x910 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_IN\_EN** 0x920 Pin Mux In Enable.  
**SOCW\_REG\_MAX** Register enum sentinel.  
**SOCW\_REG\_OSC0\_CFG1** 0x000 OSC0\_CFG1 register.  
**SOCW\_REG\_CCU\_LP\_CLK\_CTL** 0x02C Clock Control register.  
**SOCW\_REG\_CCU\_SYS\_CLK\_CTL** 0x038 System Clock Control.  
**SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL** 0x018 Perip Clock Gate Ctl.  
**SOCW\_REG\_CCU\_SS\_PERIPH\_CLK\_GATE\_CTL** 0x0028 SS PCL Gate Ctl.  
**SOCW\_REG\_CCU\_EXT\_CLK\_CTL** 0x024 CCU Ext Clock Gate Ctl.  
**SOCW\_REG\_CMP\_PWR** 0x30C Comprtr Power Enable. 0x30C Comparator Power Enable.  
**SOCW\_REG\_SLP\_CFG** 0x550 Sleep Configuration.  
**SOCW\_REG\_PMUX\_PULLUP0** 0x900 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_PULLUP1** 0x904 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_PULLUP2** 0x908 Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_PULLUP3** 0x90c Pin Mux Pullup.  
**SOCW\_REG\_PMUX\_SLEW0** 0x910 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_SLEW1** 0x914 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_SLEW2** 0x918 Pin Mux Slew.  
**SOCW\_REG\_PMUX\_SLEW3** 0x91c Pin Mux Slew.  
**SOCW\_REG\_PMUX\_IN\_EN0** 0x920 Pin Mux In Enable.  
**SOCW\_REG\_PMUX\_IN\_EN1** 0x924 Pin Mux In Enable.  
**SOCW\_REG\_PMUX\_IN\_EN2** 0x928 Pin Mux In Enable.  
**SOCW\_REG\_PMUX\_IN\_EN3** 0x92c Pin Mux In Enable.  
**SOCW\_REG\_MAX** Register enum sentinel.

Definition at line 78 of file soc\_watch.h.

## 4.25.3 Function Documentation

## 4.25.3.1 void soc\_watch\_log\_app\_event ( soc\_watch\_event\_t event\_id, uint8\_t ev\_subtype, uintptr\_t ev\_data )

Log an application event via the power profile logger.

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

## Parameters

|    |                   |   |
|----|-------------------|---|
| in | <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 227 of file soc\_watch.c.

References qm\_irq\_disable(), qm\_irq\_enable(), SOCW\_EVENT\_HALT, SOCW\_EVENT\_INTERRUPT, SOCW\_EVENT\_MAX, and SOCW\_EVENT\_SLEEP.

Referenced by soc\_watch\_log\_event().

## 4.25.3.2 void soc\_watch\_log\_event ( soc\_watch\_event\_t event\_id, uintptr\_t ev\_data )

Log a power profile event.

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

## Parameters

|    |                 |   |
|----|-----------------|---|
| in | <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 217 of file soc\_watch.c.

References soc\_watch\_log\_app\_event().



## 4.26 SS ADC

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

### Data Structures

- struct `qm_ss_adc_config_t`  
*SS ADC configuration type.*
- struct `qm_ss_adc_xfer_t`  
*SS ADC transfer type.*

### Typedefs

- typedef `uint16_t qm_ss_adc_sample_t`  
*SS ADC sample size type.*
- typedef `uint8_t qm_ss_adc_calibration_t`  
*SS ADC calibration type.*

### Enumerations

- enum `qm_ss_adc_status_t` {  
`QM_SS_ADC_IDLE = 0x0`, `QM_SS_ADC_COMPLETE = 0x1`, `QM_SS_ADC_OVERFLOW = 0x2`, `QM_SS_ADC_UNDERFLOW = 0x4`,  
`QM_SS_ADC_SEQERROR = 0x8` }  
*SS ADC status.*
- enum `qm_ss_adc_resolution_t` { `QM_SS_ADC_RES_6_BITS = 0x5`, `QM_SS_ADC_RES_8_BITS = 0x7`, `QM_SS_ADC_RES_10_BITS = 0x9`, `QM_SS_ADC_RES_12_BITS = 0xB` }  
*SS ADC resolution type.*
- enum `qm_ss_adc_mode_t` {  
`QM_SS_ADC_MODE_DEEP_PWR_DOWN`, `QM_SS_ADC_MODE_PWR_DOWN`, `QM_SS_ADC_MODE_STDBY`, `QM_SS_ADC_MODE_NORM_CAL`,  
`QM_SS_ADC_MODE_NORM_NO_CAL` }  
*SS ADC operating mode type.*
- enum `qm_ss_adc_channel_t` {  
`QM_SS_ADC_CH_0`, `QM_SS_ADC_CH_1`, `QM_SS_ADC_CH_2`, `QM_SS_ADC_CH_3`,  
`QM_SS_ADC_CH_4`, `QM_SS_ADC_CH_5`, `QM_SS_ADC_CH_6`, `QM_SS_ADC_CH_7`,  
`QM_SS_ADC_CH_8`, `QM_SS_ADC_CH_9`, `QM_SS_ADC_CH_10`, `QM_SS_ADC_CH_11`,  
`QM_SS_ADC_CH_12`, `QM_SS_ADC_CH_13`, `QM_SS_ADC_CH_14`, `QM_SS_ADC_CH_15`,  
`QM_SS_ADC_CH_16`, `QM_SS_ADC_CH_17`, `QM_SS_ADC_CH_18` }  
*SS ADC channels type.*
- enum `qm_ss_adc_cb_source_t` { `QM_SS_ADC_TRANSFER`, `QM_SS_ADC_MODE_CHANGED`, `QM_SS_ADC_CAL_COMPLETE` }  
*SS ADC interrupt callback source.*

### Functions

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

*Calibrate the SS ADC.*

- int `qm_ss_adc_irq_calibrate` (const `qm_ss_adc_t` adc, void(\*callback)(void \*data, int error, `qm_ss_adc_status_t` status, `qm_ss_adc_cb_source_t` source), void \*callback\_data)

*Calibrate the SS ADC.*

- int `qm_ss_adc_set_calibration` (const `qm_ss_adc_t` adc, const `qm_ss_adc_calibration_t` cal)

*Set SS ADC calibration data.*

- int `qm_ss_adc_get_calibration` (const `qm_ss_adc_t` adc, `qm_ss_adc_calibration_t` \*const cal)

*Get the current calibration data for an SS ADC.*

- int `qm_ss_adc_set_config` (const `qm_ss_adc_t` adc, const `qm_ss_adc_config_t` \*const cfg)

*Set SS ADC configuration.*

- int `qm_ss_adc_convert` (const `qm_ss_adc_t` adc, `qm_ss_adc_xfer_t` \*const xfer, `qm_ss_adc_status_t` \*const status)

*Synchronously read values from the ADC.*

- int `qm_ss_adc_irq_convert` (const `qm_ss_adc_t` adc, `qm_ss_adc_xfer_t` \*const xfer)

*Asynchronously read values from the SS ADC.*

#### 4.26.1 Detailed Description

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

#### 4.26.2 Enumeration Type Documentation

##### 4.26.2.1 enum `qm_ss_adc_cb_source_t`

SS ADC interrupt callback source.

Enumerator

**`QM_SS_ADC_TRANSFER`** Transfer complete or error callback.

**`QM_SS_ADC_MODE_CHANGED`** Mode change complete callback.

**`QM_SS_ADC_CAL_COMPLETE`** Calibration complete callback.

Definition at line 89 of file `qm_ss_adc.h`.

##### 4.26.2.2 enum `qm_ss_adc_channel_t`

SS ADC channels type.

Enumerator

**`QM_SS_ADC_CH_0`** ADC Channel 0.

**`QM_SS_ADC_CH_1`** ADC Channel 1.

**`QM_SS_ADC_CH_2`** ADC Channel 2.

**`QM_SS_ADC_CH_3`** ADC Channel 3.

**`QM_SS_ADC_CH_4`** ADC Channel 4.

**`QM_SS_ADC_CH_5`** ADC Channel 5.

**`QM_SS_ADC_CH_6`** ADC Channel 6.

**`QM_SS_ADC_CH_7`** ADC Channel 7.

**`QM_SS_ADC_CH_8`** ADC Channel 8.

**`QM_SS_ADC_CH_9`** ADC Channel 9.

**`QM_SS_ADC_CH_10`** ADC Channel 10.

**`QM_SS_ADC_CH_11`** ADC Channel 11.

**QM\_SS\_ADC\_CH\_12** ADC Channel 12.  
**QM\_SS\_ADC\_CH\_13** ADC Channel 13.  
**QM\_SS\_ADC\_CH\_14** ADC Channel 14.  
**QM\_SS\_ADC\_CH\_15** ADC Channel 15.  
**QM\_SS\_ADC\_CH\_16** ADC Channel 16.  
**QM\_SS\_ADC\_CH\_17** ADC Channel 17.  
**QM\_SS\_ADC\_CH\_18** ADC Channel 18.

Definition at line 64 of file qm\_ss\_adc.h.

#### 4.26.2.3 enum qm\_ss\_adc\_mode\_t

SS ADC operating mode type.

Enumerator

**QM\_SS\_ADC\_MODE\_DEEP\_PWR\_DOWN** Deep power down mode.  
**QM\_SS\_ADC\_MODE\_PWR\_DOWN** Power down mode.  
**QM\_SS\_ADC\_MODE\_STDBY** Standby mode.  
**QM\_SS\_ADC\_MODE\_NORM\_CAL** Normal mode, with calibration.  
**QM\_SS\_ADC\_MODE\_NORM\_NO\_CAL** Normal mode, no calibration.

Definition at line 53 of file qm\_ss\_adc.h.

#### 4.26.2.4 enum qm\_ss\_adc\_resolution\_t

SS ADC resolution type.

Enumerator

**QM\_SS\_ADC\_RES\_6\_BITS** 6-bit mode.  
**QM\_SS\_ADC\_RES\_8\_BITS** 8-bit mode.  
**QM\_SS\_ADC\_RES\_10\_BITS** 10-bit mode.  
**QM\_SS\_ADC\_RES\_12\_BITS** 12-bit mode.

Definition at line 43 of file qm\_ss\_adc.h.

#### 4.26.2.5 enum qm\_ss\_adc\_status\_t

SS ADC status.

Enumerator

**QM\_SS\_ADC\_IDLE** ADC idle.  
**QM\_SS\_ADC\_COMPLETE** ADC data available.  
**QM\_SS\_ADC\_OVERFLOW** ADC overflow error.  
**QM\_SS\_ADC\_UNDERFLOW** ADC underflow error.  
**QM\_SS\_ADC\_SEQERROR** ADC sequencer error.

Definition at line 32 of file qm\_ss\_adc.h.

### 4.26.3 Function Documentation

#### 4.26.3.1 int qm\_ss\_adc\_calibrate ( const qm\_ss\_adc\_t adc )

Calibrate the SS ADC.

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

**Parameters**

|           |            |                         |
|-----------|------------|-------------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

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

Synchronously read values from the ADC.

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

**Parameters**

|                |               |   |
|----------------|---------------|---|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 553 of file `qm_ss_adc.c`.

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

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

Get the current calibration data for an SS ADC.

**Parameters**

|            |            |  |
|------------|------------|--|
| <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> | <a href="#">errno</a> for possible error codes. |

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

Calibrate the SS ADC.

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

#### Parameters

|    |                      |                                |
|----|----------------------|--------------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 473 of file `qm_ss_adc.c`.

References `QM_SS_IO_CREG_MST0_CTRL`.

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

Asynchronously read values from the SS ADC.

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

#### Parameters

|         |             |  |
|---------|-------------|--|
| in      | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 627 of file `qm_ss_adc.c`.

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

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

Switch operating mode of SS ADC.

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

#### Parameters

|    |            |                      |
|----|------------|----------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 403 of file `qm_ss_adc.c`.

References `QM_SS_ADC_MODE_NORM_NO_CAL`, and `QM_SS_IO_CREG_MST0_CTRL`.

#### 4.26.3.7 `int qm_ss_adc_set_calibration ( const qm_ss_adc_t adc, const qm_ss_adc_calibration_t cal )`

Set SS ADC calibration data.

**Parameters**

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

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

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

Set SS ADC configuration.

This sets the sample window and resolution.

**Parameters**

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

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 335 of file `qm_ss_adc.c`.

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

#### 4.26.3.9 `int qm_ss_adc_set_mode ( const qm_ss_adc_t adc, const qm_ss_adc_mode_t mode )`

Switch operating mode of SS ADC.

This call is blocking.

**Parameters**

|           |             |                      |
|-----------|-------------|----------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 361 of file `qm_ss_adc.c`.

References `QM_SS_ADC_MODE_NORM_CAL`, `QM_SS_ADC_MODE_NORM_NO_CAL`, `QM_SS_IO_CREG_MST0_CTRL`, and `QM_SS_IO_CREG_SLV0_OBSR`.

## 4.27 SS GPIO

General Purpose IO for Sensor Subsystem.

### Data Structures

- struct [qm\\_ss\\_gpio\\_port\\_config\\_t](#)  
*GPIO port configuration type.*

### Enumerations

- enum [qm\\_ss\\_gpio\\_state\\_t](#) { [QM\\_SS\\_GPIO\\_LOW](#), [QM\\_SS\\_GPIO\\_HIGH](#) }  
*GPIO SS pin states.*

### Functions

- int [qm\\_ss\\_gpio\\_set\\_config](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, const [qm\\_ss\\_gpio\\_port\\_config\\_t](#) \*const cfg)  
*Set SS GPIO port configuration.*
- int [qm\\_ss\\_gpio\\_read\\_pin](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, const uint8\_t pin, [qm\\_ss\\_gpio\\_state\\_t](#) \*const state)  
*Read the current value of a single pin on a given SS GPIO port.*
- int [qm\\_ss\\_gpio\\_set\\_pin](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, const uint8\_t pin)  
*Set a single pin on a given SS GPIO port.*
- int [qm\\_ss\\_gpio\\_clear\\_pin](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, const uint8\_t pin)  
*Clear a single pin on a given SS GPIO port.*
- int [qm\\_ss\\_gpio\\_set\\_pin\\_state](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, const uint8\_t pin, const [qm\\_ss\\_gpio\\_state\\_t](#) state)  
*Set or clear a single SS GPIO pin using a state variable.*
- int [qm\\_ss\\_gpio\\_read\\_port](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, uint32\_t \*const port)  
*Get SS GPIO port values.*
- int [qm\\_ss\\_gpio\\_write\\_port](#) (const [qm\\_ss\\_gpio\\_t](#) gpio, const uint32\_t val)  
*Get SS GPIO port values.*

#### 4.27.1 Detailed Description

General Purpose IO for Sensor Subsystem.

#### 4.27.2 Enumeration Type Documentation

##### 4.27.2.1 enum [qm\\_ss\\_gpio\\_state\\_t](#)

GPIO SS pin states.

#### Enumerator

**[QM\\_SS\\_GPIO\\_LOW](#)** Pin level high.

**[QM\\_SS\\_GPIO\\_HIGH](#)** Pin level low.

Definition at line 21 of file [qm\\_ss\\_gpio.h](#).

#### 4.27.3 Function Documentation

##### 4.27.3.1 int [qm\\_ss\\_gpio\\_clear\\_pin](#) ( const [qm\\_ss\\_gpio\\_t](#) *gpio*, const uint8\_t *pin* )

Clear a single pin on a given SS GPIO port.



**Parameters**

|    |             |                               |
|----|-------------|-------------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

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

4.27.3.2 `int qm_ss_gpio_read_pin ( const qm_ss_gpio_t gpio, const uint8_t pin, qm_ss_gpio_state_t *const 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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 62 of file `qm_ss_gpio.c`.

4.27.3.3 `int qm_ss_gpio_read_port ( const qm_ss_gpio_t gpio, uint32_t *const 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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 116 of file `qm_ss_gpio.c`.

4.27.3.4 `int qm_ss_gpio_set_config ( const qm_ss_gpio_t gpio, const qm_ss_gpio_port_config_t *const 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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

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

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

#### 4.27.3.5 `int qm_ss_gpio_set_pin ( const qm_ss_gpio_t gpio, const uint8_t pin )`

Set a single pin on a given SS GPIO port.

**Parameters**

|    |             |                             |
|----|-------------|-----------------------------|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 76 of file `qm_ss_gpio.c`.

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

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

**Parameters**

|    |              |   |
|----|--------------|---|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 102 of file `qm_ss_gpio.c`.

#### 4.27.3.7 `int qm_ss_gpio_write_port ( const qm_ss_gpio_t gpio, const uint32_t val )`

Get SS GPIO port values.

---

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

**Parameters**

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

**Returns**

Standard `errno` return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 126 of file `qm_ss_gpio.c`.

## 4.28 SS I2C

I2C driver for Sensor Subsystem.

### Data Structures

- struct `qm_ss_i2c_config_t`  
*QM SS I2C configuration type.*
- struct `qm_ss_i2c_transfer_t`  
*QM SS I2C transfer type.*

### Enumerations

- enum `qm_ss_i2c_addr_t` { `QM_SS_I2C_7_BIT = 0`, `QM_SS_I2C_10_BIT` }  
*QM SS I2C addressing type.*
- enum `qm_ss_i2c_speed_t` { `QM_SS_I2C_SPEED_STD = 1`, `QM_SS_I2C_SPEED_FAST = 2` }  
*QM SS I2C Speed Type.*
- enum `qm_ss_i2c_status_t` {  
`QM_I2C_IDLE = 0`, `QM_I2C_TX_ABRT_7B_ADDR_NOACK = BIT(0)`, `QM_I2C_TX_ABRT_TXDATA_NOACK = BIT(3)`, `QM_I2C_TX_ABRT_SBYTE_ACKDET = BIT(7)`,  
`QM_I2C_TX_ABRT_MASTER_DIS = BIT(11)`, `QM_I2C_TX_ARB_LOST = BIT(12)`, `QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO = BIT(13)`, `QM_I2C_TX_ABRT_SLV_ARBLOST = BIT(14)`,  
`QM_I2C_TX_ABRT_SLVRD_INTX = BIT(15)`, `QM_I2C_TX_ABRT_USER_ABRT = BIT(16)`, `QM_I2C_BUSY = BIT(17)` }  
*QM SS I2C status type.*

### Functions

- int `qm_ss_i2c_set_config` (const `qm_ss_i2c_t` i2c, const `qm_ss_i2c_config_t` \*const cfg)  
*Set SS I2C configuration.*
- int `qm_ss_i2c_set_speed` (const `qm_ss_i2c_t` i2c, const `qm_ss_i2c_speed_t` speed, const `uint16_t` lo\_cnt, const `uint16_t` hi\_cnt)  
*Set I2C speed.*
- int `qm_ss_i2c_get_status` (const `qm_ss_i2c_t` i2c, `qm_ss_i2c_status_t` \*const status)  
*Retrieve SS I2C status.*
- int `qm_ss_i2c_master_write` (const `qm_ss_i2c_t` i2c, const `uint16_t` slave\_addr, const `uint8_t` \*const data, `uint32_t` len, const bool stop, `qm_ss_i2c_status_t` \*const status)  
*Master write on I2C.*
- int `qm_ss_i2c_master_read` (const `qm_ss_i2c_t` i2c, const `uint16_t` slave\_addr, `uint8_t` \*const data, `uint32_t` len, const bool stop, `qm_ss_i2c_status_t` \*const status)  
*Master read of I2C.*
- int `qm_ss_i2c_master_irq_transfer` (const `qm_ss_i2c_t` i2c, const `qm_ss_i2c_transfer_t` \*const xfer, const `uint16_t` slave\_addr)  
*Interrupt based master transfer on I2C.*
- int `qm_ss_i2c_irq_transfer_terminate` (const `qm_ss_i2c_t` i2c)  
*Terminate I2C IRQ/DMA transfer.*

#### 4.28.1 Detailed Description

I2C driver for Sensor Subsystem.

#### 4.28.2 Enumeration Type Documentation

##### 4.28.2.1 enum `qm_ss_i2c_addr_t`

QM SS I2C addressing type.

###### Enumerator

**`QM_SS_I2C_7_BIT`** 7-bit mode.

**`QM_SS_I2C_10_BIT`** 10-bit mode.

Definition at line 37 of file `qm_ss_i2c.h`.

##### 4.28.2.2 enum `qm_ss_i2c_speed_t`

QM SS I2C Speed Type.

###### Enumerator

**`QM_SS_I2C_SPEED_STD`** Standard mode (100 Kbps).

**`QM_SS_I2C_SPEED_FAST`** Fast mode (400 Kbps).

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

##### 4.28.2.3 enum `qm_ss_i2c_status_t`

QM SS I2C status type.

###### Enumerator

**`QM_I2C_IDLE`** Controller idle.

**`QM_I2C_TX_ABRT_7B_ADDR_NOACK`** 7-bit address noack.

**`QM_I2C_TX_ABRT_TXDATA_NOACK`** Tx data noack.

**`QM_I2C_TX_ABRT_SBYTE_ACKDET`** Start ACK.

**`QM_I2C_TX_ABRT_MASTER_DIS`** Master disabled.

**`QM_I2C_TX_ARB_LOST`** Master lost arbitration.

**`QM_I2C_TX_ABRT_SLVFLUSH_TXFIFO`** Slave flush tx FIFO.

**`QM_I2C_TX_ABRT_SLV_ARBLOST`** Slave lost bus.

**`QM_I2C_TX_ABRT_SLVRD_INTX`** Slave read completion.

**`QM_I2C_TX_ABRT_USER_ABORT`** User abort.

**`QM_I2C_BUSY`** Controller busy.

Definition at line 53 of file `qm_ss_i2c.h`.

#### 4.28.3 Function Documentation

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

Retrieve SS I2C status.

###### Parameters

---

|     |               |  |
|-----|---------------|--|
| in  | <i>i2c</i>    | Which I2C to read the status of.       |
| out | <i>status</i> | Get i2c status. This must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 408 of file `qm_ss_i2c.c`.

References `QM_I2C_BUSY`.

Referenced by `qm_ss_i2c_master_read()`, and `qm_ss_i2c_master_write()`.

#### 4.28.3.2 `int qm_ss_i2c_irq_transfer_terminate ( const qm_ss_i2c_t i2c )`

Terminate I2C IRQ/DMA transfer.

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

**Parameters**

|    |            |                             |
|----|------------|-----------------------------|
| in | <i>i2c</i> | I2C register block pointer. |
|----|------------|-----------------------------|

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| 0               | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 680 of file `qm_ss_i2c.c`.

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

Interrupt based master transfer on I2C.

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

**Parameters**

|    |                   |  |
|----|-------------------|--|
| in | <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

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 589 of file `qm_ss_i2c.c`.

References `qm_ss_i2c_transfer_t::rx_len`.

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

Master read of I2C.

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

## Parameters

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

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 507 of file `qm_ss_i2c.c`.

References `qm_ss_i2c_get_status()`.

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

Master write on I2C.

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

## Parameters

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

## Returns

Standard `errno` return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 429 of file `qm_ss_i2c.c`.

References `qm_ss_i2c_get_status()`.

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

Set SS I2C configuration.

##### Parameters

|           |            |   |
|-----------|------------|---|
| <i>in</i> | <i>i2c</i> | Which I2C to set the configuration of.    |
| <i>in</i> | <i>cfg</i> | I2C configuration. This must not be NULL. |

##### Returns

Standard `errno` return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 284 of file `qm_ss_i2c.c`.

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

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

Set I2C speed.

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

##### Parameters

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

##### Returns

Standard `errno` return type for QMSI.

##### Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 370 of file `qm_ss_i2c.c`.

References `QM_SS_I2C_SPEED_FAST`, and `QM_SS_I2C_SPEED_STD`.

## 4.29 SS Interrupt

Interrupt driver for Sensor Subsystem.

### Typedefs

- typedef void(\* [qm\\_ss\\_isr\\_t](#) )(struct interrupt\_frame \*frame)  
*Interrupt service routine type.*

### Functions

- void [qm\\_ss\\_irq\\_enable](#) (void)  
*Enable interrupt delivery for the Sensor Subsystem.*
- void [qm\\_ss\\_irq\\_disable](#) (void)  
*Disable interrupt delivery for the Sensor Subsystem.*
- void [qm\\_ss\\_irq\\_unmask](#) (uint32\_t irq)  
*Unmask a given interrupt line.*
- void [qm\\_ss\\_irq\\_mask](#) (uint32\_t irq)  
*Mask a given interrupt line.*
- void [qm\\_ss\\_irq\\_request](#) (uint32\_t irq, [qm\\_ss\\_isr\\_t](#) isr)  
*Request a given IRQ and register ISR to interrupt vector.*
- void [qm\\_ss\\_int\\_vector\\_request](#) (uint32\_t vector, [qm\\_ss\\_isr\\_t](#) isr)  
*Register an Interrupt Service Routine to a given interrupt vector.*

#### 4.29.1 Detailed Description

Interrupt driver for Sensor Subsystem.

#### 4.29.2 Function Documentation

##### 4.29.2.1 void [qm\\_ss\\_int\\_vector\\_request](#) ( uint32\_t vector, [qm\\_ss\\_isr\\_t](#) isr )

Register an Interrupt Service Routine to a given interrupt vector.

#### Parameters

|           |               |  |
|-----------|---------------|--|
| <i>in</i> | <i>vector</i> | Interrupt Vector number.   |
| <i>in</i> | <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](#)().

##### 4.29.2.2 void [qm\\_ss\\_irq\\_mask](#) ( uint32\_t irq )

Mask a given interrupt line.

#### Parameters

|           |            |                    |
|-----------|------------|--------------------|
| <i>in</i> | <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](#)().

##### 4.29.2.3 void [qm\\_ss\\_irq\\_request](#) ( uint32\_t irq, [qm\\_ss\\_isr\\_t](#) isr )

Request a given IRQ and register ISR to interrupt vector.

**Parameters**

|           |            |                               |
|-----------|------------|-------------------------------|
| <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()`.

**4.29.2.4 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()`.

## 4.30 SS ISR

Sensor Subsystem Interrupt Service Routines.

### Functions

- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_adc\\_0\\_isr\)](#)  
*ISR for ADC interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_adc\\_0\\_err\\_isr\)](#)  
*ISR for ADC error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_adc\\_0\\_cal\\_isr\)](#)  
*ISR for SS ADC 0 calibration interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_adc\\_0\\_pwr\\_isr\)](#)  
*ISR for SS ADC 0 mode change interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_gpio\\_isr\\_0\)](#)  
*ISR for GPIO 0 error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_gpio\\_isr\\_1\)](#)  
*ISR for GPIO 1 error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_i2c\\_isr\\_0\)](#)  
*ISR for I2C 0 error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_i2c\\_isr\\_1\)](#)  
*ISR for I2C 1 error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_spi\\_0\\_err\\_isr\)](#)  
*ISR for SPI 0 error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_spi\\_1\\_err\\_isr\)](#)  
*ISR for SPI 1 error interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_spi\\_0\\_tx\\_isr\)](#)  
*ISR for SPI 0 TX data requested interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_spi\\_1\\_tx\\_isr\)](#)  
*ISR for SPI 1 TX data requested interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_spi\\_0\\_rx\\_isr\)](#)  
*ISR for SPI 0 RX data available interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_spi\\_1\\_rx\\_isr\)](#)  
*ISR for SPI 1 data available interrupt.*
- [QM\\_ISR\\_DECLARE \(qm\\_ss\\_timer\\_isr\\_0\)](#)  
*ISR for SS Timer 0 interrupt.*

### 4.30.1 Detailed Description

Sensor Subsystem Interrupt Service Routines.

### 4.30.2 Function Documentation

#### 4.30.2.1 QM\_ISR\_DECLARE ( qm\_ss\_adc\_0\_isr )

ISR for ADC interrupt.

This function needs to be registered with

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

if IRQ based conversions are used.

Definition at line 197 of file qm\_ss\_adc.c.

References QM\_SS\_ADC\_0.

#### 4.30.2.2 QM\_ISR\_DECLARE ( qm\_ss\_adc\_0\_err\_isr )

ISR for ADC error interrupt.

This function needs to be registered with

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

if IRQ based conversions are used.

Definition at line 203 of file qm\_ss\_adc.c.

References QM\_SS\_ADC\_0.

#### 4.30.2.3 QM\_ISR\_DECLARE ( qm\_ss\_adc\_0\_cal\_isr )

ISR for SS ADC 0 calibration interrupt.

This function needs to be registered with

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

if IRQ based calibration is used.

Definition at line 215 of file qm\_ss\_adc.c.

References QM\_SS\_ADC\_0.

#### 4.30.2.4 QM\_ISR\_DECLARE ( qm\_ss\_adc\_0\_pwr\_isr )

ISR for SS ADC 0 mode change interrupt.

This function needs to be registered with

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

if IRQ based mode change is used.

Definition at line 209 of file qm\_ss\_adc.c.

References QM\_SS\_ADC\_0.

#### 4.30.2.5 QM\_ISR\_DECLARE ( qm\_ss\_gpio\_isr\_0 )

ISR for GPIO 0 error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 27 of file qm\_ss\_gpio.c.

#### 4.30.2.6 QM\_ISR\_DECLARE ( qm\_ss\_gpio\_isr\_1 )

ISR for GPIO 1 error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 32 of file qm\_ss\_gpio.c.

#### 4.30.2.7 QM\_ISR\_DECLARE ( qm\_ss\_i2c\_isr\_0 )

ISR for I2C 0 error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 259 of file qm\_ss\_i2c.c.

#### 4.30.2.8 QM\_ISR\_DECLARE ( qm\_ss\_i2c\_isr\_1 )

ISR for I2C 1 error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 264 of file qm\_ss\_i2c.c.

#### 4.30.2.9 QM\_ISR\_DECLARE ( qm\_ss\_spi\_0\_err\_isr )

ISR for SPI 0 error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 384 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_0.

#### 4.30.2.10 QM\_ISR\_DECLARE ( qm\_ss\_spi\_1\_err\_isr )

ISR for SPI 1 error interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 388 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_1.

#### 4.30.2.11 QM\_ISR\_DECLARE ( qm\_ss\_spi\_0\_tx\_isr )

ISR for SPI 0 TX data requested interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 400 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_0.

#### 4.30.2.12 QM\_ISR\_DECLARE ( qm\_ss\_spi\_1\_tx\_isr )

ISR for SPI 1 TX data requested interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 404 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_1.

#### 4.30.2.13 QM\_ISR\_DECLARE ( qm\_ss\_spi\_0\_rx\_isr )

ISR for SPI 0 RX data available interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 392 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_0.

#### 4.30.2.14 QM\_ISR\_DECLARE ( qm\_ss\_spi\_1\_rx\_isr )

ISR for SPI 1 data available interrupt.

This function needs to be registered with

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

if IRQ based transfers are used.

Definition at line 396 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_1.

#### 4.30.2.15 QM\_ISR\_DECLARE ( qm\_ss\_timer\_isr\_0 )

ISR for SS Timer 0 interrupt.

This function needs to be registered with

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

Definition at line 24 of file qm\_ss\_timer.c.



## 4.31 SS SPI

SPI peripheral driver for Sensor Subsystem.

### Data Structures

- struct [qm\\_ss\\_spi\\_config\\_t](#)  
*SPI configuration type.*
- struct [qm\\_ss\\_spi\\_async\\_transfer\\_t](#)  
*SPI IRQ transfer type.*
- struct [qm\\_ss\\_spi\\_transfer\\_t](#)  
*SPI transfer type.*

### Enumerations

- enum [qm\\_ss\\_spi\\_frame\\_size\\_t](#) {  
[QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_4\\_BIT](#) = 3, [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_5\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_6\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_7\\_BIT](#),  
[QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_8\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_9\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_10\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_11\\_BIT](#),  
[QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_12\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_13\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_14\\_BIT](#), [QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_15\\_BIT](#),  
[QM\\_SS\\_SPI\\_FRAME\\_SIZE\\_16\\_BIT](#) }  
*QM SPI frame size type.*
- enum [qm\\_ss\\_spi\\_tmode\\_t](#) { [QM\\_SS\\_SPI\\_TMOD\\_TX\\_RX](#), [QM\\_SS\\_SPI\\_TMOD\\_TX](#), [QM\\_SS\\_SPI\\_TMOD\\_RX](#), [QM\\_SS\\_SPI\\_TMOD\\_EEPROM\\_READ](#) }  
*SPI transfer mode type.*
- enum [qm\\_ss\\_spi\\_bmode\\_t](#) { [QM\\_SS\\_SPI\\_BMODE\\_0](#), [QM\\_SS\\_SPI\\_BMODE\\_1](#), [QM\\_SS\\_SPI\\_BMODE\\_2](#), [QM\\_SS\\_SPI\\_BMODE\\_3](#) }  
*SPI bus mode type.*
- enum [qm\\_ss\\_spi\\_slave\\_select\\_t](#) {  
[QM\\_SS\\_SPI\\_SS\\_DISABLED](#) = 0, [QM\\_SS\\_SPI\\_SS\\_0](#) = BIT(0), [QM\\_SS\\_SPI\\_SS\\_1](#) = BIT(1), [QM\\_SS\\_SPI\\_SS\\_2](#) = BIT(2),  
[QM\\_SS\\_SPI\\_SS\\_3](#) = BIT(3) }  
*SPI slave select type.*
- enum [qm\\_ss\\_spi\\_status\\_t](#) { [QM\\_SS\\_SPI\\_IDLE](#), [QM\\_SS\\_SPI\\_BUSY](#), [QM\\_SS\\_SPI\\_RX\\_OVERFLOW](#) }  
*SPI status.*

### Functions

- int [qm\\_ss\\_spi\\_set\\_config](#) (const [qm\\_ss\\_spi\\_t](#) spi, const [qm\\_ss\\_spi\\_config\\_t](#) \*const cfg)  
*Set SPI configuration.*
- int [qm\\_ss\\_spi\\_slave\\_select](#) (const [qm\\_ss\\_spi\\_t](#) spi, const [qm\\_ss\\_spi\\_slave\\_select\\_t](#) ss)  
*Set Slave Select lines.*
- int [qm\\_ss\\_spi\\_get\\_status](#) (const [qm\\_ss\\_spi\\_t](#) spi, [qm\\_ss\\_spi\\_status\\_t](#) \*const status)  
*Get SPI bus status.*
- int [qm\\_ss\\_spi\\_transfer](#) (const [qm\\_ss\\_spi\\_t](#) spi, const [qm\\_ss\\_spi\\_transfer\\_t](#) \*const xfer, [qm\\_ss\\_spi\\_status\\_t](#) \*const status)  
*Perform a blocking SPI transfer.*
- int [qm\\_ss\\_spi\\_irq\\_transfer](#) (const [qm\\_ss\\_spi\\_t](#) spi, const [qm\\_ss\\_spi\\_async\\_transfer\\_t](#) \*const xfer)  
*Initiate a interrupt based SPI transfer.*
- int [qm\\_ss\\_spi\\_transfer\\_terminate](#) (const [qm\\_ss\\_spi\\_t](#) spi)  
*Terminate SPI IRQ transfer.*

## 4.31.1 Detailed Description

SPI peripheral driver for Sensor Subsystem.

## 4.31.2 Enumeration Type Documentation

## 4.31.2.1 enum qm\_ss\_spi\_bmode\_t

SPI bus mode type.

## Enumerator

- QM\_SS\_SPI\_BMODE\_0** Clock Polarity = 0, Clock Phase = 0.
- QM\_SS\_SPI\_BMODE\_1** Clock Polarity = 0, Clock Phase = 1.
- QM\_SS\_SPI\_BMODE\_2** Clock Polarity = 1, Clock Phase = 0.
- QM\_SS\_SPI\_BMODE\_3** Clock Polarity = 1, Clock Phase = 1.

Definition at line 78 of file qm\_ss\_spi.h.

## 4.31.2.2 enum qm\_ss\_spi\_frame\_size\_t

QM SPI frame size type.

## Enumerator

- QM\_SS\_SPI\_FRAME\_SIZE\_4\_BIT** 4 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_5\_BIT** 5 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_6\_BIT** 6 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_7\_BIT** 7 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_8\_BIT** 8 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_9\_BIT** 9 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_10\_BIT** 10 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_11\_BIT** 11 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_12\_BIT** 12 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_13\_BIT** 13 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_14\_BIT** 14 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_15\_BIT** 15 bit frame.
- QM\_SS\_SPI\_FRAME\_SIZE\_16\_BIT** 16 bit frame.

Definition at line 21 of file qm\_ss\_spi.h.

## 4.31.2.3 enum qm\_ss\_spi\_slave\_select\_t

SPI slave select type.

Slave selects can be combined by logical OR if multiple slaves are selected during one transfer. Setting only QM\_SS\_SPI\_SS\_DISABLED prevents the controller from starting the transfer.

## Enumerator

- QM\_SS\_SPI\_SS\_DISABLED** Slave select disable.
- QM\_SS\_SPI\_SS\_0** Slave select 0.
- QM\_SS\_SPI\_SS\_1** Slave select 1.
- QM\_SS\_SPI\_SS\_2** Slave select 2.
- QM\_SS\_SPI\_SS\_3** Slave select 3.

Definition at line 92 of file qm\_ss\_spi.h.

## 4.31.2.4 enum qm\_ss\_spi\_status\_t

SPI status.

## Enumerator

**QM\_SS\_SPI\_IDLE** SPI device is not in use.

**QM\_SS\_SPI\_BUSY** SPI device is busy.

**QM\_SS\_SPI\_RX\_OVERFLOW** RX transfer has overflowed.

Definition at line 103 of file qm\_ss\_spi.h.

## 4.31.2.5 enum qm\_ss\_spi\_tmode\_t

SPI transfer mode type.

## Enumerator

**QM\_SS\_SPI\_TMOD\_TX\_RX** Transmit & Receive mode. This mode synchronously receives and transmits data during the transfer. rx\_len and tx\_len buffer need to be the same length.

**QM\_SS\_SPI\_TMOD\_TX** Transmit-Only mode. This mode only transmits data. The rx buffer is not accessed and rx\_len need to be set to 0.

**QM\_SS\_SPI\_TMOD\_RX** Receive-Only mode. This mode only receives data. The tx buffer is not accessed and tx\_len need to be set to 0.

**QM\_SS\_SPI\_TMOD\_EEPROM\_READ** EEPROM-Read Mode. This mode transmits the data stored in the tx buffer (EEPROM address). After the transmit is completed it populates the rx buffer (EEPROM data) with received data.

Definition at line 40 of file qm\_ss\_spi.h.

## 4.31.3 Function Documentation

## 4.31.3.1 int qm\_ss\_spi\_get\_status ( const qm\_ss\_spi\_t spi, qm\_ss\_spi\_status\_t \*const status )

Get SPI bus status.

## Parameters

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

## Returns

Standard errno return type for QMSI.

## Return values

|          |                                 |
|----------|---------------------------------|
| 0        | on success.                     |
| Negative | errno for possible error codes. |

Definition at line 100 of file qm\_ss\_spi.c.

References QM\_SS\_SPI\_BUSY, QM\_SS\_SPI\_IDLE, and QM\_SS\_SPI\_SR.

---

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

Initiate a interrupt based SPI transfer.

Perform an interrupt based SPI transfer. If transfer mode is full duplex (QM\_SS\_SPI\_TMOD\_TX\_RX), then tx\_len and rx\_len must be equal. Similarly, for transmit-only transfers (QM\_SS\_SPI\_TMOD\_TX) rx\_len must be 0, while for receive-only transfers (QM\_SS\_SPI\_TMOD\_RX) tx\_len must be 0. This function is non blocking.

**Parameters**

|    |             |  |
|----|-------------|--|
| in | <i>spi</i>  | SPI module identifier.   |
| in | <i>xfer</i> | 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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 197 of file `qm_ss_spi.c`.

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

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

Set SPI configuration.

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

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

**Parameters**

|    |            |   |
|----|------------|---|
| in | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

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

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

#### 4.31.3.4 `int qm_ss_spi_slave_select ( const qm_ss_spi_t spi, const qm_ss_spi_slave_select_t ss )`

Set Slave Select lines.

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

**Parameters**

|    |            |   |
|----|------------|---|
| in | <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 82 of file `qm_ss_spi.c`.

References `QM_SS_SPI_SPIEN`, and `QM_SS_SPI_SR`.

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

Perform a blocking SPI transfer.

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

## Parameters

|            |               |   |
|------------|---------------|---|
| <i>in</i>  | <i>spi</i>    | SPI module identifier.  |
| <i>in</i>  | <i>xfer</i>   | Structure containing transfer information. This must not be NULL.                   |
| <i>out</i> | <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 115 of file `qm_ss_spi.c`.

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

**4.31.3.6** `int qm_ss_spi_transfer_terminate ( const qm_ss_spi_t spi )`

Terminate SPI IRQ transfer.

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

## Parameters

|           |            |                        |
|-----------|------------|------------------------|
| <i>in</i> | <i>spi</i> | SPI module identifier. |
|-----------|------------|------------------------|

## Returns

Standard `errno` return type for QMSI.

## Return values

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

|   |
|---|
| <i>Negative</i>   <a href="#">errno</a> for possible error codes. |
|---|

Definition at line 246 of file `qm_ss_spi.c`.

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

## 4.32 SS Timer

Timer driver for Sensor Subsystem.

### Data Structures

- struct [qm\\_ss\\_timer\\_config\\_t](#)  
*Sensor Subsystem Timer Configuration Type.*

### Functions

- int [qm\\_ss\\_timer\\_set\\_config](#) (const [qm\\_ss\\_timer\\_t](#) timer, const [qm\\_ss\\_timer\\_config\\_t](#) \*const cfg)  
*Set the SS timer configuration.*
- int [qm\\_ss\\_timer\\_set](#) (const [qm\\_ss\\_timer\\_t](#) timer, const uint32\_t count)  
*Set SS timer count value.*
- int [qm\\_ss\\_timer\\_get](#) (const [qm\\_ss\\_timer\\_t](#) timer, uint32\_t \*const count)  
*Get SS timer count value.*

#### 4.32.1 Detailed Description

Timer driver for Sensor Subsystem.

#### 4.32.2 Function Documentation

##### 4.32.2.1 int [qm\\_ss\\_timer\\_get](#) ( const [qm\\_ss\\_timer\\_t](#) timer, uint32\_t \*const count )

Get SS timer count value.

Get the current count value of the SS timer.

#### Parameters

|     |              |  |
|-----|--------------|--|
| in  | <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.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 59 of file [qm\\_ss\\_timer.c](#).

##### 4.32.2.2 int [qm\\_ss\\_timer\\_set](#) ( const [qm\\_ss\\_timer\\_t](#) timer, const uint32\_t count )

Set SS timer count value.

Set the current count value of the SS timer.

#### Parameters



|                 |                    |                                     |
|-----------------|--------------------|-------------------------------------|
| <code>in</code> | <code>timer</code> | Which SS timer to set the count of. |
| <code>in</code> | <code>count</code> | Value to load the timer with.       |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <code>0</code>  | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 50 of file `qm_ss_timer.c`.

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

Set the SS timer configuration.

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

**Parameters**

|                 |                    |  |
|-----------------|--------------------|--|
| <code>in</code> | <code>timer</code> | Which SS timer to configure.                   |
| <code>in</code> | <code>cfg</code>   | SS timer configuration. This must not be NULL. |

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <code>0</code>  | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 29 of file `qm_ss_timer.c`.

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

## 4.33 SS Clock

Clock Management for Sensor Subsystem.

### Enumerations

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

*Peripheral clocks selection type.*

### Functions

- int `ss_clk_gpio_enable` (const `qm_ss_gpio_t` gpio)  
*Enable clocking for SS GPIO peripheral.*
- int `ss_clk_gpio_disable` (const `qm_ss_gpio_t` gpio)  
*Disable clocking for SS GPIO peripheral.*
- int `ss_clk_spi_enable` (const `qm_ss_spi_t` spi)  
*Enable clocking for SS SPI peripheral.*
- int `ss_clk_spi_disable` (const `qm_ss_spi_t` spi)  
*Disable clocking for SS SPI peripheral.*
- int `ss_clk_i2c_enable` (const `qm_ss_i2c_t` i2c)  
*Enable clocking for SS I2C peripheral.*
- int `ss_clk_i2c_disable` (const `qm_ss_i2c_t` i2c)  
*Disable clocking for SS I2C peripheral.*
- int `ss_clk_adc_enable` (void)  
*Enable the SS ADC clock.*
- int `ss_clk_adc_disable` (void)  
*Disable the SS ADC clock.*
- int `ss_clk_adc_set_div` (const `uint32_t` div)  
*Set clock divisor for SS ADC.*

#### 4.33.1 Detailed Description

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

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

#### 4.33.2 Enumeration Type Documentation

##### 4.33.2.1 enum `ss_clk_periph_t`

Peripheral clocks selection type.

## Enumerator

**SS\_CLK\_PERIPH\_ADC** ADC clock selector.

**SS\_CLK\_PERIPH\_I2C\_1** I2C 1 clock selector.

**SS\_CLK\_PERIPH\_I2C\_0** I2C 0 clock selector.

**SS\_CLK\_PERIPH\_SPI\_1** SPI 1 clock selector.

**SS\_CLK\_PERIPH\_SPI\_0** SPI 0 clock selector.

**SS\_CLK\_PERIPH\_GPIO\_1** GPIO 1 clock selector. Special domain peripherals - these do not map onto the standard register.

**SS\_CLK\_PERIPH\_GPIO\_0** GPIO 0 clock selector. Special domain peripherals - these do not map onto the standard register.

Definition at line 31 of file `ss_clk.h`.

## 4.33.3 Function Documentation

4.33.3.1 `int ss_clk_adc_disable ( void )`

Disable the SS ADC clock.

## Returns

Standard errno return type for QMSI.

## Return values

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

Definition at line 68 of file `ss_clk.c`.

References `QM_SS_ADC_CTRL`.

4.33.3.2 `int ss_clk_adc_enable ( void )`

Enable the SS ADC clock.

## Returns

Standard errno return type for QMSI.

## Return values

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

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

References `QM_SS_ADC_CTRL`.

4.33.3.3 `int ss_clk_adc_set_div ( const uint32_t div )`

Set clock divisor for SS ADC.

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

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

## Parameters

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

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 76 of file `ss_clk.c`.

References `clk_sys_get_ticks_per_us()`, and `QM_SS_ADC_DIVSEQSTAT`.

4.33.3.4 `int ss_clk_gpio_disable ( const qm_ss_gpio_t gpio )`

Disable clocking for SS GPIO peripheral.

## Parameters

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

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 19 of file `ss_clk.c`.

4.33.3.5 `int ss_clk_gpio_enable ( const qm_ss_gpio_t gpio )`

Enable clocking for SS GPIO peripheral.

## Parameters

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

## Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 8 of file `ss_clk.c`.

4.33.3.6 `int ss_clk_i2c_disable ( const qm_ss_i2c_t i2c )`

Disable clocking for SS I2C peripheral.

**Parameters**

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

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 52 of file `ss_clk.c`.

#### 4.33.3.7 `int ss_clk_i2c_enable ( const qm_ss_i2c_t i2c )`

Enable clocking for SS I2C peripheral.

**Parameters**

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

**Returns**

Standard errno return type for QMSI.

**Return values**

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

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

#### 4.33.3.8 `int ss_clk_spi_disable ( const qm_ss_spi_t spi )`

Disable clocking for SS SPI peripheral.

**Parameters**

|           |            |                 |
|-----------|------------|-----------------|
| <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> | <a href="#">errno</a> for possible error codes. |

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

References `QM_SS_SPI_0`, and `QM_SS_SPI_CTRL`.

#### 4.33.3.9 `int ss_clk_spi_enable ( const qm_ss_spi_t spi )`

Enable clocking for SS SPI peripheral.

**Parameters**

|           |            |                 |
|-----------|------------|-----------------|
| <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> | <a href="#">errno</a> for possible error codes. |

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

References `QM_SS_SPI_0`, and `QM_SS_SPI_CTRL`.

## 4.34 SS Power states

SS Power mode control for Quark SE Microcontrollers.

### Enumerations

- enum `ss_power_cpu_ss1_mode_t` { `SS_POWER_CPU_SS1_TIMER_OFF` = 0, `SS_POWER_CPU_SS1_TIMER_ON` }  
*Sensor Subsystem SS1 Timers mode type.*

### Functions

- void `ss_power_soc_lpss_enable` (void)  
*Enable LPSS state entry.*
- void `ss_power_soc_lpss_disable` (void)  
*Disable LPSS state entry.*
- void `ss_power_cpu_ss1` (const `ss_power_cpu_ss1_mode_t` mode)  
*Enter Sensor SS1 state.*
- void `ss_power_cpu_ss2` (void)  
*Enter Sensor SS2 state or SoC LPSS state.*

#### 4.34.1 Detailed Description

SS Power mode control for Quark SE Microcontrollers.

#### 4.34.2 Enumeration Type Documentation

##### 4.34.2.1 enum `ss_power_cpu_ss1_mode_t`

Sensor Subsystem SS1 Timers mode type.

### Enumerator

- `SS_POWER_CPU_SS1_TIMER_OFF`** Disable SS Timers in SS1.
- `SS_POWER_CPU_SS1_TIMER_ON`** Keep SS Timers enabled in SS1.

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

#### 4.34.3 Function Documentation

##### 4.34.3.1 void `ss_power_cpu_ss1` ( const `ss_power_cpu_ss1_mode_t` mode )

Enter Sensor SS1 state.

Put the Sensor Subsystem into SS1.

Processor Clock is gated in this state.

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

A wake event is a sensor subsystem interrupt.

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

## Parameters

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

Definition at line 82 of file `ss_power_states.c`.

References `SS_POWER_CPU_SS1_TIMER_OFF`, and `SS_POWER_CPU_SS1_TIMER_ON`.

#### 4.34.3.2 void `ss_power_cpu_ss2` ( void )

Enter Sensor SS2 state or SoC LPSS state.

Put the Sensor Subsystem into SS2.

Sensor Complex Clock is gated in this state.

Sensor Peripherals are gated in this state.

This enables entry in LPSS if:

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

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

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

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

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

Definition at line 123 of file `ss_power_states.c`.

#### 4.34.3.3 void `ss_power_soc_lpss_disable` ( void )

Disable LPSS state entry.

Clear LPSS enable flag.

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

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

Definition at line 55 of file `ss_power_states.c`.

References `SOCW_EVENT_REGISTER`, and `SOCW_REG_CCUCPU_LP_CLK_CTL`.

#### 4.34.3.4 void `ss_power_soc_lpss_enable` ( void )

Enable LPSS state entry.

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

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

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

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

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

SoC Hybrid Clock is gated in this state.

Core Well Clocks are gated.



RTC is the only clock running.

Possible SoC wake events are:

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

Definition at line 32 of file `ss_power_states.c`.

References `SOCW_EVENT_REGISTER`, and `SOCW_REG_CCU_LP_CLK_CTL`.

**4.35 Quark D2000 Flash Layout**

Flash Layout for Quark D2000 Microcontrollers.

**4.35.1 Detailed Description**

Flash Layout for Quark D2000 Microcontrollers.

## 4.36 Quark D2000 Power states

Power mode control for Quark D2000 Microcontrollers.

### Enumerations

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

### Functions

- void `power_cpu_halt` (void)  
*Put CPU in halt state.*
- void `power_soc_sleep` ()  
*Put SoC to sleep.*
- void `power_soc_deep_sleep` (const `power_wake_event_t` wake\_event)  
*Put SoC to deep sleep.*

#### 4.36.1 Detailed Description

Power mode control for Quark D2000 Microcontrollers.

#### 4.36.2 Enumeration Type Documentation

##### 4.36.2.1 enum `power_wake_event_t`

Wake source for deep sleep mode type.

### Enumerator

- `POWER_WAKE_FROM_GPIO_COMP`** Use GPIO / Comparator as wake source.
- `POWER_WAKE_FROM_RTC`** Use RTC as wake source.

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

#### 4.36.3 Function Documentation

##### 4.36.3.1 void `power_cpu_halt` ( void )

Put CPU in halt state.

Halts the CPU until next interrupt or reset.

Definition at line 15 of file `power_states.c`.

References `SOCW_EVENT_HALT`.

Referenced by `power_soc_deep_sleep()`, and `power_soc_sleep()`.

##### 4.36.3.2 void `power_soc_deep_sleep` ( const `power_wake_event_t` wake\_event )

Put SoC to deep sleep.

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

## Parameters

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

Definition at line 149 of file `power_states.c`.

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

#### 4.36.3.3 void power\_soc\_sleep ( )

Put SoC to sleep.

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

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

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

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

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

## 4.37 SoC Registers (D2000)

Quark D2000 SoC Registers.

### Data Structures

- struct [qm\\_scss\\_ccu\\_reg\\_t](#)  
*System Core register map.*
- struct [qm\\_scss\\_gp\\_reg\\_t](#)  
*General Purpose register map.*
- struct [qm\\_scss\\_cmp\\_reg\\_t](#)  
*Comparator register map.*
- struct [qm\\_scss\\_int\\_reg\\_t](#)  
*Interrupt register map.*
- struct [qm\\_scss\\_pmu\\_reg\\_t](#)  
*Power Management register map.*
- struct [qm\\_scss\\_aon\\_reg\\_t](#)  
*Always-on Controller register map.*
- struct [qm\\_scss\\_peripheral\\_reg\\_t](#)  
*Peripheral Registers register map.*
- struct [qm\\_scss\\_pmux\\_reg\\_t](#)  
*Pin MUX register map.*
- struct [qm\\_scss\\_info\\_reg\\_t](#)  
*Information register map.*
- struct [qm\\_pwm\\_channel\\_t](#)  
*PWM / Timer channel register map.*
- struct [qm\\_pwm\\_reg\\_t](#)  
*PWM / Timer register map.*
- struct [qm\\_wdt\\_reg\\_t](#)  
*Watchdog timer register map.*
- struct [qm\\_uart\\_reg\\_t](#)  
*UART register map.*
- struct [qm\\_spi\\_reg\\_t](#)  
*SPI register map.*
- struct [qm\\_rtc\\_reg\\_t](#)  
*RTC register map.*
- struct [qm\\_i2c\\_reg\\_t](#)  
*I2C register map.*
- struct [qm\\_gpio\\_reg\\_t](#)  
*GPIO register map.*
- struct [qm\\_adc\\_reg\\_t](#)  
*ADC register map.*
- struct [qm\\_flash\\_reg\\_t](#)  
*Flash register map.*
- struct [qm\\_mpr\\_reg\\_t](#)  
*Memory Protection Region register map.*
- struct [pic\\_timer\\_reg\\_pad\\_t](#)  
*PIC timer register structure.*
- struct [qm\\_pic\\_timer\\_reg\\_t](#)  
*PIC timer register map.*
- struct [mVIC\\_reg\\_pad\\_t](#)

*MVIC register structure.*

- struct [qm\\_mvic\\_reg\\_t](#)

*MVIC register map.*

- struct [qm\\_dma\\_chan\\_reg\\_t](#)

*DMA channel register map.*

- struct [qm\\_dma\\_int\\_reg\\_t](#)

*DMA interrupt register map.*

- struct [qm\\_dma\\_misc\\_reg\\_t](#)

*DMA miscellaneous register map.*

#### System Core

- [qm\\_scss\\_ccu\\_reg\\_t](#) **test\_scss\_ccu**

#### General Purpose

- [qm\\_scss\\_gp\\_reg\\_t](#) **test\_scss\_gp**

#### Comparator

- [qm\\_scss\\_cmp\\_reg\\_t](#) **test\_scss\_cmp**

#### Interrupt

- [qm\\_scss\\_int\\_reg\\_t](#) **test\_scss\_int**

#### Power Management

- [qm\\_scss\\_pmu\\_reg\\_t](#) **test\_scss\_pmu**

#### Always-on controllers.

- enum [qm\\_scss\\_aon\\_t](#)

*Number of SCSS Always-on controllers.*

- [qm\\_scss\\_aon\\_reg\\_t](#) **test\_scss\_aon**

#### Peripheral Registers

- [qm\\_scss\\_peripheral\\_reg\\_t](#) **test\_scss\_peripheral**

#### Pin MUX

- [qm\\_scss\\_pmux\\_reg\\_t](#) **test\_scss\_pmux**

#### ID

- [qm\\_scss\\_info\\_reg\\_t](#) **test\_scss\_info**

## PWM / Timer

- enum [qm\\_pwm\\_t](#)  
*Number of PWM / Timer controllers.*
- enum [qm\\_pwm\\_id\\_t](#)  
*PWM ID type.*
- [qm\\_pwm\\_reg\\_t](#) [test\\_pwm\\_t](#)

## WDT

- enum [qm\\_wdt\\_t](#)  
*Number of WDT controllers.*
- [qm\\_wdt\\_reg\\_t](#) [test\\_wdt](#)

## UART

- enum [qm\\_uart\\_t](#)  
*Number of UART controllers.*
- [qm\\_uart\\_reg\\_t](#) [test\\_uart\\_instance](#)
- [qm\\_uart\\_reg\\_t](#) \* [test\\_uart](#) [QM\_UART\_NUM]
- [qm\\_uart\\_reg\\_t](#) \* [qm\\_uart](#) [QM\_UART\_NUM]

## SPI

- enum [qm\\_spi\\_t](#)  
*Number of SPI controllers.*
- [qm\\_spi\\_reg\\_t](#) [test\\_spi](#)
- [qm\\_spi\\_reg\\_t](#) \* [test\\_spi\\_controllers](#) [QM\_SPI\_NUM]
- [qm\\_spi\\_reg\\_t](#) \* [qm\\_spi\\_controllers](#) [QM\_SPI\_NUM]  
*Extern [qm\\_spi\\_reg\\_t](#)\* array declared at [qm\\_soc\\_regs.h](#) .*

## RTC

- enum [qm\\_rtc\\_t](#)  
*Number of RTC controllers.*
- [qm\\_rtc\\_reg\\_t](#) [test\\_rtc](#)

## I2C

- enum [qm\\_i2c\\_t](#)  
*Number of I2C controllers.*
- [qm\\_i2c\\_reg\\_t](#) [test\\_i2c\\_instance](#) [QM\_I2C\_NUM]
- [qm\\_i2c\\_reg\\_t](#) \* [test\\_i2c](#) [QM\_I2C\_NUM]
- [qm\\_i2c\\_reg\\_t](#) \* [qm\\_i2c](#) [QM\_I2C\_NUM]  
*I2C register block.*

## GPIO

- enum [qm\\_gpio\\_t](#)  
*Number of GPIO controllers.*
- [qm\\_gpio\\_reg\\_t](#) [test\\_gpio\\_instance](#)
- [qm\\_gpio\\_reg\\_t](#) \* [test\\_gpio](#) [QM\_GPIO\_NUM]
- [qm\\_gpio\\_reg\\_t](#) \* [qm\\_gpio](#) [QM\_GPIO\_NUM]

## ADC

- enum `qm_adc_t`  
*Number of ADC controllers.*
- `qm_adc_reg_t test_adc`

## Flash

- enum `qm_flash_t`  
*Number of Flash controllers.*
- `qm_flash_reg_t test_flash_instance`
- `qm_flash_reg_t * test_flash` [QM\_FLASH\_NUM]
- `uint8_t test_flash_page` [0x800]
- `qm_flash_reg_t * qm_flash` [QM\_FLASH\_NUM]

## Memory Protection Region

- `qm_mpr_reg_t test_mpr`

## PIC

- `qm_pic_timer_reg_t test_pic_timer`

## Peripheral Clock

- enum `clk_periph_t` {  
`CLK_PERIPH_REGISTER = BIT(0), CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_PERIPH_SPI_S = BIT(4),`  
`CLK_PERIPH_SPI_M0 = BIT(5), CLK_PERIPH_GPIO_INTERRUPT = BIT(7), CLK_PERIPH_GPIO_DB = BIT(8), CLK_PERIPH_WDT_REGISTER = BIT(10),`  
`CLK_PERIPH_RTC_REGISTER = BIT(11), CLK_PERIPH_PWM_REGISTER = BIT(12), CLK_PERIPH_GPIO_REGISTER = BIT(13), CLK_PERIPH_SPI_M0_REGISTER,`  
`CLK_PERIPH_SPI_S_REGISTER, CLK_PERIPH_UARTA_REGISTER = BIT(17), CLK_PERIPH_UARTB_REGISTER = BIT(18), CLK_PERIPH_I2C_M0_REGISTER,`  
`CLK_PERIPH_ADC = BIT(22), CLK_PERIPH_ADC_REGISTER = BIT(23), CLK_PERIPH_ALL = 0xCFFFFFF,`  
`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 = 0x3FFFFFF }`
- Peripheral clock register map.*

## MVIC

- `qm_mvic_reg_t test_mvic`
- `qm_ioapic_reg_t test_ioapic`



## DMA

- enum `qm_dma_t` { `QM_DMA_0`, `QM_DMA_NUM`, `QM_DMA_0`, `QM_DMA_NUM` }  
*DMA instances.*
- enum `qm_dma_channel_id_t` {  
`QM_DMA_CHANNEL_0` = 0, `QM_DMA_CHANNEL_1`, `QM_DMA_CHANNEL_NUM`, `QM_DMA_CHANNEL_0` = 0,  
`QM_DMA_CHANNEL_1`, `QM_DMA_CHANNEL_2`, `QM_DMA_CHANNEL_3`, `QM_DMA_CHANNEL_4`,  
`QM_DMA_CHANNEL_5`, `QM_DMA_CHANNEL_6`, `QM_DMA_CHANNEL_7`, `QM_DMA_CHANNEL_NUM` }  
*DMA channel IDs.*
- enum `qm_dma_handshake_interface_t` {  
`DMA_HW_IF_UART_A_TX` = 0x0, `DMA_HW_IF_UART_A_RX` = 0x1, `DMA_HW_IF_UART_B_TX` = 0x2, `DMA_HW_IF_UART_B_RX` = 0x3,  
`DMA_HW_IF_SPI_MASTER_0_TX` = 0x4, `DMA_HW_IF_SPI_MASTER_0_RX` = 0x5, `DMA_HW_IF_SPI_SLAVE_TX` = 0x8, `DMA_HW_IF_SPI_SLAVE_RX` = 0x9,  
`DMA_HW_IF_I2C_MASTER_0_TX` = 0xc, `DMA_HW_IF_I2C_MASTER_0_RX` = 0xd, `DMA_HW_IF_UART_A_TX` = 0x0, `DMA_HW_IF_UART_A_RX` = 0x1,  
`DMA_HW_IF_UART_B_TX` = 0x2, `DMA_HW_IF_UART_B_RX` = 0x3, `DMA_HW_IF_SPI_MASTER_0_TX` = 0x4, `DMA_HW_IF_SPI_MASTER_0_RX` = 0x5,  
`DMA_HW_IF_SPI_MASTER_1_TX` = 0x6, `DMA_HW_IF_SPI_MASTER_1_RX` = 0x7, `DMA_HW_IF_SPI_SLAVE_TX` = 0x8, `DMA_HW_IF_SPI_SLAVE_RX` = 0x9,  
`DMA_HW_IF_I2S_PLAYBACK` = 0xa, `DMA_HW_IF_I2S_CAPTURE` = 0xb, `DMA_HW_IF_I2C_MASTER_0_TX` = 0xc, `DMA_HW_IF_I2C_MASTER_0_RX` = 0xd,  
`DMA_HW_IF_I2C_MASTER_1_TX` = 0xe, `DMA_HW_IF_I2C_MASTER_1_RX` = 0xf }  
*DMA hardware handshake interfaces.*
- `qm_dma_reg_t test_dma_instance` [`QM_DMA_NUM`]
- `qm_dma_reg_t * test_dma` [`QM_DMA_NUM`]
- `qm_dma_reg_t * qm_dma` [`QM_DMA_NUM`]

## Versioning

- `uint32_t test_rom_version`

## 4.37.1 Detailed Description

Quark D2000 SoC Registers.

## 4.37.2 Enumeration Type Documentation

4.37.2.1 enum `clk_periph_t`

Peripheral clock register map.

## Enumerator

- `CLK_PERIPH_REGISTER`** Peripheral Clock Gate Enable.
- `CLK_PERIPH_CLK`** Peripheral Clock Enable.
- `CLK_PERIPH_I2C_M0`** I2C Master 0 Clock Enable.
- `CLK_PERIPH_SPI_S`** SPI Slave Clock Enable.
- `CLK_PERIPH_SPI_M0`** SPI Master 0 Clock Enable.
- `CLK_PERIPH_GPIO_INTERRUPT`** GPIO Interrupt Clock Enable.
- `CLK_PERIPH_GPIO_DB`** GPIO Debounce Clock Enable.
- `CLK_PERIPH_WDT_REGISTER`** Watchdog Clock Enable.

**CLK\_PERIPH\_RTC\_REGISTER** RTC Clock Gate Enable.

**CLK\_PERIPH\_PWM\_REGISTER** PWM Clock Gate Enable.

**CLK\_PERIPH\_GPIO\_REGISTER** GPIO Clock Gate Enable.

**CLK\_PERIPH\_SPI\_M0\_REGISTER** SPI Master 0 Clock Gate Enable.

**CLK\_PERIPH\_SPI\_S\_REGISTER** SPI Slave Clock Gate Enable.

**CLK\_PERIPH\_UARTA\_REGISTER** UARTA Clock Gate Enable.

**CLK\_PERIPH\_UARTB\_REGISTER** UARTB Clock Gate Enable.

**CLK\_PERIPH\_I2C\_M0\_REGISTER** I2C Master 0 Clock Gate Enable.

**CLK\_PERIPH\_ADC** ADC Clock Enable.

**CLK\_PERIPH\_ADC\_REGISTER** ADC Clock Gate Enable.

**CLK\_PERIPH\_ALL** Quark D2000 peripherals Enable.

**CLK\_PERIPH\_REGISTER** Peripheral Clock Gate Enable.

**CLK\_PERIPH\_CLK** Peripheral Clock Enable.

**CLK\_PERIPH\_I2C\_M0** I2C Master 0 Clock Enable.

**CLK\_PERIPH\_I2C\_M1** I2C Master 1 Clock Enable.

**CLK\_PERIPH\_SPI\_S** SPI Slave Clock Enable.

**CLK\_PERIPH\_SPI\_M0** SPI Master 0 Clock Enable.

**CLK\_PERIPH\_SPI\_M1** SPI Master 1 Clock Enable.

**CLK\_PERIPH\_GPIO\_INTERRUPT** GPIO Interrupt Clock Enable.

**CLK\_PERIPH\_GPIO\_DB** GPIO Debounce Clock Enable.

**CLK\_PERIPH\_I2S** I2S Clock Enable.

**CLK\_PERIPH\_WDT\_REGISTER** Watchdog Clock Enable.

**CLK\_PERIPH\_RTC\_REGISTER** RTC Clock Gate Enable.

**CLK\_PERIPH\_PWM\_REGISTER** PWM Clock Gate Enable.

**CLK\_PERIPH\_GPIO\_REGISTER** GPIO Clock Gate Enable.

**CLK\_PERIPH\_SPI\_M0\_REGISTER** SPI Master 0 Clock Gate Enable.

**CLK\_PERIPH\_SPI\_M1\_REGISTER** SPI Master 1 Clock Gate Enable.

**CLK\_PERIPH\_SPI\_S\_REGISTER** SPI Slave Clock Gate Enable.

**CLK\_PERIPH\_UARTA\_REGISTER** UARTA Clock Gate Enable.

**CLK\_PERIPH\_UARTB\_REGISTER** UARTB Clock Gate Enable.

**CLK\_PERIPH\_I2C\_M0\_REGISTER** I2C Master 0 Clock Gate Enable.

**CLK\_PERIPH\_I2C\_M1\_REGISTER** I2C Master 1 Clock Gate Enable.

**CLK\_PERIPH\_I2S\_REGISTER** I2S Clock Gate Enable.

**CLK\_PERIPH\_ALL** Quark SE peripherals Mask.

Definition at line 1183 of file qm\_soc\_regs.h.

#### 4.37.2.2 enum qm\_adc\_t

Number of ADC controllers.

Definition at line 976 of file qm\_soc\_regs.h.

## 4.37.2.3 enum qm\_dma\_channel\_id\_t

DMA channel IDs.

Enumerator

**QM\_DMA\_CHANNEL\_0** DMA channel id for channel 0.  
**QM\_DMA\_CHANNEL\_1** DMA channel id for channel 1.  
**QM\_DMA\_CHANNEL\_NUM** Number of DMA channels.  
**QM\_DMA\_CHANNEL\_0** DMA channel id for channel 0.  
**QM\_DMA\_CHANNEL\_1** DMA channel id for channel 1.  
**QM\_DMA\_CHANNEL\_2** DMA channel id for channel 2.  
**QM\_DMA\_CHANNEL\_3** DMA channel id for channel 3.  
**QM\_DMA\_CHANNEL\_4** DMA channel id for channel 4.  
**QM\_DMA\_CHANNEL\_5** DMA channel id for channel 5.  
**QM\_DMA\_CHANNEL\_6** DMA channel id for channel 6.  
**QM\_DMA\_CHANNEL\_7** DMA channel id for channel 7.  
**QM\_DMA\_CHANNEL\_NUM** Number of DMA channels.

Definition at line 1299 of file qm\_soc\_regs.h.

## 4.37.2.4 enum qm\_dma\_handshake\_interface\_t

DMA hardware handshake interfaces.

Enumerator

**DMA\_HW\_IF\_UART\_A\_TX** UART\_A\_TX.  
**DMA\_HW\_IF\_UART\_A\_RX** UART\_A\_RX.  
**DMA\_HW\_IF\_UART\_B\_TX** UART\_B\_TX.  
**DMA\_HW\_IF\_UART\_B\_RX** UART\_B\_RX.  
**DMA\_HW\_IF\_SPI\_MASTER\_0\_TX** SPI\_Master\_0\_TX.  
**DMA\_HW\_IF\_SPI\_MASTER\_0\_RX** SPI\_Master\_0\_RX.  
**DMA\_HW\_IF\_SPI\_SLAVE\_TX** SPI\_Slave\_TX.  
**DMA\_HW\_IF\_SPI\_SLAVE\_RX** SPI\_Slave\_RX.  
**DMA\_HW\_IF\_I2C\_MASTER\_0\_TX** I2C\_Master\_0\_TX.  
**DMA\_HW\_IF\_I2C\_MASTER\_0\_RX** I2C\_Master\_0\_RX.  
**DMA\_HW\_IF\_UART\_A\_TX** UART\_A\_TX.  
**DMA\_HW\_IF\_UART\_A\_RX** UART\_A\_RX.  
**DMA\_HW\_IF\_UART\_B\_TX** UART\_B\_TX.  
**DMA\_HW\_IF\_UART\_B\_RX** UART\_B\_RX.  
**DMA\_HW\_IF\_SPI\_MASTER\_0\_TX** SPI\_Master\_0\_TX.  
**DMA\_HW\_IF\_SPI\_MASTER\_0\_RX** SPI\_Master\_0\_RX.  
**DMA\_HW\_IF\_SPI\_MASTER\_1\_TX** SPI\_Master\_1\_TX.  
**DMA\_HW\_IF\_SPI\_MASTER\_1\_RX** SPI\_Master\_1\_RX.  
**DMA\_HW\_IF\_SPI\_SLAVE\_TX** SPI\_Slave\_TX.  
**DMA\_HW\_IF\_SPI\_SLAVE\_RX** SPI\_Slave\_RX.  
**DMA\_HW\_IF\_I2S\_PLAYBACK** I2S\_Playback channel.  
**DMA\_HW\_IF\_I2S\_CAPTURE** I2S\_Capture channel.  
**DMA\_HW\_IF\_I2C\_MASTER\_0\_TX** I2C\_Master\_0\_TX.  
**DMA\_HW\_IF\_I2C\_MASTER\_0\_RX** I2C\_Master\_0\_RX.  
**DMA\_HW\_IF\_I2C\_MASTER\_1\_TX** I2C\_Master\_1\_TX.  
**DMA\_HW\_IF\_I2C\_MASTER\_1\_RX** I2C\_Master\_1\_RX.

Definition at line 1306 of file qm\_soc\_regs.h.

#### 4.37.2.5 enum qm\_dma\_t

DMA instances.

Enumerator

**QM\_DMA\_0** DMA controller id.

**QM\_DMA\_NUM** Number of DMA controllers.

**QM\_DMA\_0** DMA controller id.

**QM\_DMA\_NUM** Number of DMA controllers.

Definition at line 1293 of file qm\_soc\_regs.h.

#### 4.37.2.6 enum qm\_flash\_t

Number of Flash controllers.

Definition at line 1048 of file qm\_soc\_regs.h.

#### 4.37.2.7 enum qm\_gpio\_t

Number of GPIO controllers.

Definition at line 926 of file qm\_soc\_regs.h.

#### 4.37.2.8 enum qm\_i2c\_t

Number of I2C controllers.

Definition at line 790 of file qm\_soc\_regs.h.

#### 4.37.2.9 enum qm\_pwm\_id\_t

PWM ID type.

Definition at line 515 of file qm\_soc\_regs.h.

#### 4.37.2.10 enum qm\_pwm\_t

Number of PWM / Timer controllers.

Definition at line 512 of file qm\_soc\_regs.h.

#### 4.37.2.11 enum qm\_rtc\_t

Number of RTC controllers.

Definition at line 756 of file qm\_soc\_regs.h.

#### 4.37.2.12 enum qm\_scss\_aon\_t

Number of SCSS Always-on controllers.

Definition at line 311 of file qm\_soc\_regs.h.

#### 4.37.2.13 enum qm\_spi\_t

Number of SPI controllers.

Definition at line 650 of file qm\_soc\_regs.h.

#### 4.37.2.14 enum qm\_uart\_t

Number of UART controllers.

Definition at line 598 of file qm\_soc\_regs.h.

#### 4.37.2.15 enum `qm_wdt_t`

Number of WDT controllers.

Definition at line 558 of file `qm_soc_regs.h`.

### 4.37.3 Variable Documentation

#### 4.37.3.1 `qm_i2c_reg_t`\* `qm_i2c[QM_I2C_NUM]`

I2C register block.

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

## 4.38 RAR

Retention alternator regulator for Quark D2000.

### Enumerations

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

### Functions

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

#### 4.38.1 Detailed Description

Retention alternator regulator for Quark D2000.

#### 4.38.2 Enumeration Type Documentation

##### 4.38.2.1 enum `rar_state_t`

RAR modes type.

#### Enumerator

**`RAR_NORMAL`** Normal mode = 50 mA.

**`RAR_RETENTION`** Retention mode = 300 uA.

Definition at line 22 of file rar.h.

#### 4.38.3 Function Documentation

##### 4.38.3.1 int `rar_set_mode` ( const `rar_state_t` mode )

Change operating mode of RAR.

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

#### Parameters

|                 |                   |                            |
|-----------------|-------------------|----------------------------|
| <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>   <a href="#">errno</a> for possible error codes. |
|---|

Definition at line 8 of file rar.c.

References RAR\_NORMAL, and RAR\_RETENTION.

Referenced by power\_soc\_deep\_sleep(), and power\_soc\_sleep().

**4.39 Quark SE Flash Layout**

Flash Layout for Quark SE Microcontrollers.

**4.39.1 Detailed Description**

Flash Layout for Quark SE Microcontrollers.



## 4.40 Quark SE SoC Power states

SoC Power mode control for Quark SE Microcontrollers.

### Functions

- void `power_soc_sleep` (void)  
*Enter SoC sleep state.*
- void `power_soc_deep_sleep` (void)  
*Enter SoC deep sleep state.*

#### 4.40.1 Detailed Description

SoC Power mode control for Quark SE Microcontrollers.

#### 4.40.2 Function Documentation

##### 4.40.2.1 void `power_soc_deep_sleep` ( void )

Enter SoC deep sleep state.

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

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

Possible SoC wake events are:

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

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

Definition at line 33 of file `power_states.c`.

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

##### 4.40.2.2 void `power_soc_sleep` ( void )

Enter SoC sleep state.

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

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

- RTC Clock is on

Possible SoC wake events are:

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

Enter SoC sleep state.

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

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

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

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

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

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

## 4.41 Quark SE Host Power states

Host Power mode control for Quark SE Microcontrollers.

### Functions

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

### 4.41.1 Detailed Description

Host Power mode control for Quark SE Microcontrollers.

These functions cannot be called from the Sensor Subsystem.

### 4.41.2 Function Documentation

#### 4.41.2.1 void `power_cpu_c1` ( void )

Enter Host C1 state.

Put the Host into C1.

Processor Clock is gated in this state.

Nothing is turned off in this state.

A wake event causes the Host to transition to C0.

A wake event is a host interrupt.

Definition at line 61 of file `power_states.c`.

References `SOCW_EVENT_HALT`.

#### 4.41.2.2 void `power_cpu_c2` ( void )

Enter Host C2 state or SoC LPSS state.

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

This enables entry in LPSS if:

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

If C2 is entered:

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

If LPSS is entered:

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

Definition at line 67 of file power\_states.c.

References SOCW\_EVENT\_REGISTER, SOCW\_EVENT\_SLEEP, and SOCW\_REG\_CCU\_LP\_CLK\_CTL.

#### 4.41.2.3 void power\_cpu\_c2lp ( void )

Enter Host C2LP state or SoC LPSS state.

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

This enables entry in LPSS if:

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

If C2LP is entered:

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

If LPSS is entered:

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

Definition at line 77 of file power\_states.c.

References SOCW\_EVENT\_REGISTER, SOCW\_EVENT\_SLEEP, and SOCW\_REG\_CCU\_LP\_CLK\_CTL.

## 4.42 SoC Registers (Sensor Subsystem)

Quark SE SoC Sensor Subsystem Registers.

### SS Timer

- enum [qm\\_ss\\_timer\\_reg\\_t](#)
- enum [qm\\_ss\\_timer\\_t](#)

*Sensor Subsystem Timers.*

### SS GPIO

GPIO registers and definitions.

- enum [qm\\_ss\\_gpio\\_reg\\_t](#)  
*Sensor Subsystem GPIO register block type.*
- enum [qm\\_ss\\_gpio\\_t](#)  
*Sensor Subsystem GPIO.*

### SS I2C

I2C registers and definitions.

- enum [qm\\_ss\\_i2c\\_reg\\_t](#)  
*Sensor Subsystem I2C register block type.*
- enum [qm\\_ss\\_i2c\\_t](#)  
*Sensor Subsystem I2C.*
- enum [qm\\_ss\\_adc\\_reg\\_t](#) {  
[QM\\_SS\\_ADC\\_SET](#) = 0, [QM\\_SS\\_ADC\\_DIVSEQSTAT](#), [QM\\_SS\\_ADC\\_SEQ](#), [QM\\_SS\\_ADC\\_CTRL](#),  
[QM\\_SS\\_ADC\\_INTSTAT](#), [QM\\_SS\\_ADC\\_SAMPLE](#) }  
*Sensor Subsystem ADC.*
- enum [qm\\_ss\\_adc\\_t](#) { [QM\\_SS\\_ADC\\_0](#) = 0 }  
*Sensor Subsystem ADC.*

### SS CREG

CREG Registers.

- enum [qm\\_ss\\_creg\\_reg\\_t](#) { [QM\\_SS\\_IO\\_CREG\\_MST0\\_CTRL](#) = 0x0, [QM\\_SS\\_IO\\_CREG\\_SLV0\\_OBSR](#) = 0x80, [QM\\_SS\\_IO\\_CREG\\_SLV1\\_OBSR](#) = 0x180 }

### SS Interrupt

IRQs and interrupt vectors.

- enum [qm\\_ss\\_irq\\_priority\\_t](#)
- enum [qm\\_ss\\_irq\\_mask\\_t](#)
- enum [qm\\_ss\\_irq\\_trigger\\_t](#)

## SS SPI

I2C registers and definitions.

- enum `qm_ss_spi_reg_t` {  
`QM_SS_SPI_CTRL = 0, QM_SS_SPI_SPIEN = 2, QM_SS_SPI_TIMING = 4, QM_SS_SPI_FTLR,`  
`QM_SS_SPI_TXFLR = 7, QM_SS_SPI_RXFLR, QM_SS_SPI_SR, QM_SS_SPI_INTR_STAT,`  
`QM_SS_SPI_INTR_MASK, QM_SS_SPI_CLR_INTR, QM_SS_SPI_DR` }  
*Sensor Subsystem SPI register map.*
- enum `qm_ss_spi_t` { `QM_SS_SPI_0 = 0, QM_SS_SPI_1` }  
*Sensor Subsystem SPI modules.*

## 4.42.1 Detailed Description

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

## 4.42.2 Enumeration Type Documentation

4.42.2.1 enum `qm_ss_adc_reg_t`

Sensor Subsystem ADC.

Sensor Subsystem ADC registers

## Enumerator

**`QM_SS_ADC_SET`** ADC and sequencer settings register.

**`QM_SS_ADC_DIVSEQSTAT`** ADC clock and sequencer status register.

**`QM_SS_ADC_SEQ`** ADC sequence entry register.

**`QM_SS_ADC_CTRL`** ADC control register.

**`QM_SS_ADC_INTSTAT`** ADC interrupt status register.

**`QM_SS_ADC_SAMPLE`** ADC sample register.

Definition at line 235 of file `qm_sensor_regs.h`.

4.42.2.2 enum `qm_ss_adc_t`

Sensor Subsystem ADC.

## Enumerator

**`QM_SS_ADC_0`** ADC first module.

Definition at line 245 of file `qm_sensor_regs.h`.

4.42.2.3 enum `qm_ss_creg_reg_t`

## Enumerator

**`QM_SS_IO_CREG_MST0_CTRL`** Master control register.

**`QM_SS_IO_CREG_SLV0_OBSR`** Slave control register.

**`QM_SS_IO_CREG_SLV1_OBSR`** Slave control register.

Definition at line 307 of file `qm_sensor_regs.h`.

#### 4.42.2.4 enum qm\_ss\_gpio\_reg\_t

Sensor Subsystem GPIO register block type.

Definition at line 108 of file qm\_sensor\_regs.h.

#### 4.42.2.5 enum qm\_ss\_gpio\_t

Sensor Subsystem GPIO.

Definition at line 127 of file qm\_sensor\_regs.h.

#### 4.42.2.6 enum qm\_ss\_i2c\_reg\_t

Sensor Subsystem I2C register block type.

Definition at line 142 of file qm\_sensor\_regs.h.

#### 4.42.2.7 enum qm\_ss\_spi\_reg\_t

Sensor Subsystem SPI register map.

#### Enumerator

- QM\_SS\_SPI\_CTRL** SPI control register.
- QM\_SS\_SPI\_SPIEN** SPI enable register.
- QM\_SS\_SPI\_TIMING** SPI serial clock divider value.
- QM\_SS\_SPI\_FTLR** Threshold value for TX/RX FIFO.
- QM\_SS\_SPI\_TXFLR** Number of valid data entries in TX FIFO.
- QM\_SS\_SPI\_RXFLR** Number of valid data entries in RX FIFO.
- QM\_SS\_SPI\_SR** SPI status register.
- QM\_SS\_SPI\_INTR\_STAT** Interrupt status register.
- QM\_SS\_SPI\_INTR\_MASK** Interrupt mask register.
- QM\_SS\_SPI\_CLR\_INTR** Interrupt clear register.
- QM\_SS\_SPI\_DR** RW buffer for FIFOs.

Definition at line 456 of file qm\_sensor\_regs.h.

#### 4.42.2.8 enum qm\_ss\_spi\_t

Sensor Subsystem SPI modules.

#### Enumerator

- QM\_SS\_SPI\_0** SPI module 0.
- QM\_SS\_SPI\_1** SPI module 1.

Definition at line 471 of file qm\_sensor\_regs.h.

## 4.43 SoC Registers (SE)

Quark SE SoC Registers.

### Data Structures

- struct [qm\\_scss\\_ccu\\_reg\\_t](#)  
*System Core register map.*
- struct [qm\\_scss\\_gp\\_reg\\_t](#)  
*General Purpose register map.*
- struct [qm\\_scss\\_mem\\_reg\\_t](#)  
*Memory Control register map.*
- struct [qm\\_scss\\_cmp\\_reg\\_t](#)  
*Comparator register map.*
- struct [qm\\_lapic\\_reg\\_t](#)  
*APIC register block type.*
- struct [int\\_ss\\_i2c\\_reg\\_t](#)  
*SS I2C Interrupt register map.*
- struct [int\\_ss\\_spi\\_reg\\_t](#)  
*SS SPI Interrupt register map.*
- struct [qm\\_scss\\_int\\_reg\\_t](#)  
*Interrupt register map.*
- struct [qm\\_scss\\_pmu\\_reg\\_t](#)  
*Power Management register map.*
- struct [qm\\_scss\\_ss\\_reg\\_t](#)  
*Sensor Subsystem register map.*
- struct [qm\\_scss\\_aon\\_reg\\_t](#)  
*Always-on Controller register map.*
- struct [qm\\_scss\\_peripheral\\_reg\\_t](#)  
*Peripheral Registers register map.*
- struct [qm\\_scss\\_pmux\\_reg\\_t](#)  
*Pin MUX register map.*
- struct [qm\\_scss\\_info\\_reg\\_t](#)  
*Information register map.*
- struct [qm\\_mailbox\\_t](#)  
*Mailbox register structure.*
- struct [qm\\_scss\\_mailbox\\_reg\\_t](#)  
*Mailbox register map.*
- struct [qm\\_pwm\\_channel\\_t](#)  
*PWM / Timer channel register map.*
- struct [qm\\_pwm\\_reg\\_t](#)  
*PWM / Timer register map.*
- struct [qm\\_wdt\\_reg\\_t](#)  
*Watchdog timer register map.*
- struct [qm\\_uart\\_reg\\_t](#)  
*UART register map.*
- struct [qm\\_spi\\_reg\\_t](#)  
*SPI register map.*
- struct [qm\\_rtc\\_reg\\_t](#)  
*RTC register map.*
- struct [qm\\_i2c\\_reg\\_t](#)



- I2C register map.*
  - struct [qm\\_gpio\\_reg\\_t](#)
  - GPIO register map.*
  - struct [qm\\_flash\\_reg\\_t](#)
  - Flash register map.*
  - struct [qm\\_mpr\\_reg\\_t](#)
  - Memory Protection Region 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**

### 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**

### Interrupt

- [qm\\_scss\\_int\\_reg\\_t](#) **test\_scss\_int**

### Power Management

- [qm\\_scss\\_pmu\\_reg\\_t](#) **test\_scss\_pmu**

### Sensor Subsystem

- [qm\\_scss\\_ss\\_reg\\_t](#) **test\_scss\_ss**

### Always-on controllers.

- enum [qm\\_scss\\_aon\\_t](#)
- Number of SCSS Always-on controllers.*
- [qm\\_scss\\_aon\\_reg\\_t](#) **test\_scss\_aon**

## Peripheral Registers

- [qm\\_scss\\_peripheral\\_reg\\_t](#) **test\_scss\_peripheral**

## Pin MUX

- [qm\\_scss\\_pmux\\_reg\\_t](#) **test\_scss\_pmux**

## ID

- [qm\\_scss\\_info\\_reg\\_t](#) **test\_scss\_info**

## Mailbox

- [qm\\_scss\\_mailbox\\_reg\\_t](#) **test\_scss\_mailbox**

## PWM / Timer

- enum [qm\\_pwm\\_t](#)  
*Number of PWM / Timer controllers.*
- enum [qm\\_pwm\\_id\\_t](#)  
*PWM ID type.*
- [qm\\_pwm\\_reg\\_t](#) **test\_pwm\_t**

## WDT

- enum [qm\\_wdt\\_t](#)  
*Number of WDT controllers.*
- [qm\\_wdt\\_reg\\_t](#) **test\_wdt**

## UART

- enum [qm\\_uart\\_t](#)  
*Number of UART controllers.*
- [qm\\_uart\\_reg\\_t](#) **test\_uart\_instance**
- [qm\\_uart\\_reg\\_t](#) \* **test\_uart** [QM\_UART\_NUM]
- [qm\\_uart\\_reg\\_t](#) \* **qm\_uart** [QM\_UART\_NUM]

## SPI

- enum [qm\\_spi\\_t](#)  
*Number of SPI controllers.*
- [qm\\_spi\\_reg\\_t](#) **test\_spi**
- [qm\\_spi\\_reg\\_t](#) \* **test\_spi\_controllers** [QM\_SPI\_NUM]
- [qm\\_spi\\_reg\\_t](#) \* **qm\_spi\_controllers** [QM\_SPI\_NUM]  
*Extern qm\_spi\_reg\_t\* array declared at qm\_soc\_regs.h .*

## RTC

- enum [qm\\_rtc\\_t](#)  
*Number of RTC controllers.*
- [qm\\_rtc\\_reg\\_t](#) **test\_rtc**

## I2C

- enum `qm_i2c_t`  
*Number of I2C controllers.*
- `qm_i2c_reg_t test_i2c_instance` [QM\_I2C\_NUM]
- `qm_i2c_reg_t * test_i2c` [QM\_I2C\_NUM]
- `qm_i2c_reg_t * qm_i2c` [QM\_I2C\_NUM]  
*I2C register block.*

## GPIO

- enum `qm_gpio_t`  
*Number of GPIO controllers.*
- `qm_gpio_reg_t test_gpio_instance`
- `qm_gpio_reg_t * test_gpio` [QM\_GPIO\_NUM]
- `qm_gpio_reg_t * qm_gpio` [QM\_GPIO\_NUM]  
*GPIO register block.*

## Flash

- enum `qm_flash_t`  
*Number of Flash controllers.*
- `qm_flash_reg_t test_flash_instance`
- `qm_flash_reg_t * test_flash` [QM\_FLASH\_NUM]
- `uint8_t test_flash_page` [0x800]
- `qm_flash_reg_t * qm_flash` [QM\_FLASH\_NUM]

## Memory Protection Region

- `qm_mpr_reg_t test_mpr`

## Peripheral Clock

- enum `clk_periph_t` {  
`CLK_PERIPH_REGISTER = BIT(0), CLK_PERIPH_CLK = BIT(1), CLK_PERIPH_I2C_M0 = BIT(2), CLK_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_P-`  
`ERIPH_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_P-`  
`ERIPH_I2C_M0_REGISTER, CLK_PERIPH_I2C_M1_REGISTER,`  
`CLK_PERIPH_I2S_REGISTER = BIT(21), CLK_PERIPH_ALL = 0x3FFFFFF }`

Peripheral clock type.

## DMA

- enum `qm_dma_t` { `QM_DMA_0`, `QM_DMA_NUM`, `QM_DMA_0`, `QM_DMA_NUM` }  
DMA instances.
- enum `qm_dma_channel_id_t` {  
`QM_DMA_CHANNEL_0` = 0, `QM_DMA_CHANNEL_1`, `QM_DMA_CHANNEL_NUM`, `QM_DMA_CHANNEL_0` = 0,  
`QM_DMA_CHANNEL_1`, `QM_DMA_CHANNEL_2`, `QM_DMA_CHANNEL_3`, `QM_DMA_CHANNEL_4`,  
`QM_DMA_CHANNEL_5`, `QM_DMA_CHANNEL_6`, `QM_DMA_CHANNEL_7`, `QM_DMA_CHANNEL_NUM` }  
DMA channel IDs.
- enum `qm_dma_handshake_interface_t` {  
`DMA_HW_IF_UART_A_TX` = 0x0, `DMA_HW_IF_UART_A_RX` = 0x1, `DMA_HW_IF_UART_B_TX` = 0x2, `DMA_HW_IF_UART_B_RX` = 0x3,  
`DMA_HW_IF_SPI_MASTER_0_TX` = 0x4, `DMA_HW_IF_SPI_MASTER_0_RX` = 0x5, `DMA_HW_IF_SPI_SLAVE_TX` = 0x8, `DMA_HW_IF_SPI_SLAVE_RX` = 0x9,  
`DMA_HW_IF_I2C_MASTER_0_TX` = 0xc, `DMA_HW_IF_I2C_MASTER_0_RX` = 0xd, `DMA_HW_IF_UART_A_TX` = 0x0, `DMA_HW_IF_UART_A_RX` = 0x1,  
`DMA_HW_IF_UART_B_TX` = 0x2, `DMA_HW_IF_UART_B_RX` = 0x3, `DMA_HW_IF_SPI_MASTER_0_TX` = 0x4, `DMA_HW_IF_SPI_MASTER_0_RX` = 0x5,  
`DMA_HW_IF_SPI_MASTER_1_TX` = 0x6, `DMA_HW_IF_SPI_MASTER_1_RX` = 0x7, `DMA_HW_IF_SPI_SLAVE_TX` = 0x8, `DMA_HW_IF_SPI_SLAVE_RX` = 0x9,  
`DMA_HW_IF_I2S_PLAYBACK` = 0xa, `DMA_HW_IF_I2S_CAPTURE` = 0xb, `DMA_HW_IF_I2C_MASTER_0_TX` = 0xc, `DMA_HW_IF_I2C_MASTER_0_RX` = 0xd,  
`DMA_HW_IF_I2C_MASTER_1_TX` = 0xe, `DMA_HW_IF_I2C_MASTER_1_RX` = 0xf }  
DMA hardware handshake interfaces.
- `qm_dma_reg_t test_dma_instance` [`QM_DMA_NUM`]
- `qm_dma_reg_t * test_dma` [`QM_DMA_NUM`]
- `qm_dma_reg_t * qm_dma` [`QM_DMA_NUM`]

## Versioning

- `uint32_t test_rom_version`

### 4.43.1 Detailed Description

Quark SE SoC Registers.

### 4.43.2 Enumeration Type Documentation

#### 4.43.2.1 enum `clk_periph_t`

Peripheral clock type.

#### Enumerator

- `CLK_PERIPH_REGISTER`** Peripheral Clock Gate Enable.
- `CLK_PERIPH_CLK`** Peripheral Clock Enable.
- `CLK_PERIPH_I2C_M0`** I2C Master 0 Clock Enable.
- `CLK_PERIPH_SPI_S`** SPI Slave Clock Enable.
- `CLK_PERIPH_SPI_M0`** SPI Master 0 Clock Enable.
- `CLK_PERIPH_GPIO_INTERRUPT`** GPIO Interrupt Clock Enable.

**CLK\_PERIPH\_GPIO\_DB** GPIO Debounce Clock Enable.  
**CLK\_PERIPH\_WDT\_REGISTER** Watchdog Clock Enable.  
**CLK\_PERIPH\_RTC\_REGISTER** RTC Clock Gate Enable.  
**CLK\_PERIPH\_PWM\_REGISTER** PWM Clock Gate Enable.  
**CLK\_PERIPH\_GPIO\_REGISTER** GPIO Clock Gate Enable.  
**CLK\_PERIPH\_SPI\_M0\_REGISTER** SPI Master 0 Clock Gate Enable.  
**CLK\_PERIPH\_SPI\_S\_REGISTER** SPI Slave Clock Gate Enable.  
**CLK\_PERIPH\_UARTA\_REGISTER** UARTA Clock Gate Enable.  
**CLK\_PERIPH\_UARTB\_REGISTER** UARTB Clock Gate Enable.  
**CLK\_PERIPH\_I2C\_M0\_REGISTER** I2C Master 0 Clock Gate Enable.  
**CLK\_PERIPH\_ADC** ADC Clock Enable.  
**CLK\_PERIPH\_ADC\_REGISTER** ADC Clock Gate Enable.  
**CLK\_PERIPH\_ALL** Quark D2000 peripherals Enable.  
**CLK\_PERIPH\_REGISTER** Peripheral Clock Gate Enable.  
**CLK\_PERIPH\_CLK** Peripheral Clock Enable.  
**CLK\_PERIPH\_I2C\_M0** I2C Master 0 Clock Enable.  
**CLK\_PERIPH\_I2C\_M1** I2C Master 1 Clock Enable.  
**CLK\_PERIPH\_SPI\_S** SPI Slave Clock Enable.  
**CLK\_PERIPH\_SPI\_M0** SPI Master 0 Clock Enable.  
**CLK\_PERIPH\_SPI\_M1** SPI Master 1 Clock Enable.  
**CLK\_PERIPH\_GPIO\_INTERRUPT** GPIO Interrupt Clock Enable.  
**CLK\_PERIPH\_GPIO\_DB** GPIO Debounce Clock Enable.  
**CLK\_PERIPH\_I2S** I2S Clock Enable.  
**CLK\_PERIPH\_WDT\_REGISTER** Watchdog Clock Enable.  
**CLK\_PERIPH\_RTC\_REGISTER** RTC Clock Gate Enable.  
**CLK\_PERIPH\_PWM\_REGISTER** PWM Clock Gate Enable.  
**CLK\_PERIPH\_GPIO\_REGISTER** GPIO Clock Gate Enable.  
**CLK\_PERIPH\_SPI\_M0\_REGISTER** SPI Master 0 Clock Gate Enable.  
**CLK\_PERIPH\_SPI\_M1\_REGISTER** SPI Master 1 Clock Gate Enable.  
**CLK\_PERIPH\_SPI\_S\_REGISTER** SPI Slave Clock Gate Enable.  
**CLK\_PERIPH\_UARTA\_REGISTER** UARTA Clock Gate Enable.  
**CLK\_PERIPH\_UARTB\_REGISTER** UARTB Clock Gate Enable.  
**CLK\_PERIPH\_I2C\_M0\_REGISTER** I2C Master 0 Clock Gate Enable.  
**CLK\_PERIPH\_I2C\_M1\_REGISTER** I2C Master 1 Clock Gate Enable.  
**CLK\_PERIPH\_I2S\_REGISTER** I2S Clock Gate Enable.  
**CLK\_PERIPH\_ALL** Quark SE peripherals Mask.

Definition at line 1354 of file qm\_soc\_regs.h.

#### 4.43.2.2 enum qm\_dma\_channel\_id\_t

DMA channel IDs.

Enumerator

**QM\_DMA\_CHANNEL\_0** DMA channel id for channel 0.  
**QM\_DMA\_CHANNEL\_1** DMA channel id for channel 1.  
**QM\_DMA\_CHANNEL\_NUM** Number of DMA channels.

***QM\_DMA\_CHANNEL\_0*** DMA channel id for channel 0.  
***QM\_DMA\_CHANNEL\_1*** DMA channel id for channel 1.  
***QM\_DMA\_CHANNEL\_2*** DMA channel id for channel 2.  
***QM\_DMA\_CHANNEL\_3*** DMA channel id for channel 3.  
***QM\_DMA\_CHANNEL\_4*** DMA channel id for channel 4.  
***QM\_DMA\_CHANNEL\_5*** DMA channel id for channel 5.  
***QM\_DMA\_CHANNEL\_6*** DMA channel id for channel 6.  
***QM\_DMA\_CHANNEL\_7*** DMA channel id for channel 7.  
***QM\_DMA\_CHANNEL\_NUM*** Number of DMA channels.

Definition at line 1406 of file qm\_soc\_regs.h.

#### 4.43.2.3 enum qm\_dma\_handshake\_interface\_t

DMA hardware handshake interfaces.

##### Enumerator

***DMA\_HW\_IF\_UART\_A\_TX*** UART\_A\_TX.  
***DMA\_HW\_IF\_UART\_A\_RX*** UART\_A\_RX.  
***DMA\_HW\_IF\_UART\_B\_TX*** UART\_B\_TX.  
***DMA\_HW\_IF\_UART\_B\_RX*** UART\_B\_RX.  
***DMA\_HW\_IF\_SPI\_MASTER\_0\_TX*** SPI\_Master\_0\_TX.  
***DMA\_HW\_IF\_SPI\_MASTER\_0\_RX*** SPI\_Master\_0\_RX.  
***DMA\_HW\_IF\_SPI\_SLAVE\_TX*** SPI\_Slave\_TX.  
***DMA\_HW\_IF\_SPI\_SLAVE\_RX*** SPI\_Slave\_RX.  
***DMA\_HW\_IF\_I2C\_MASTER\_0\_TX*** I2C\_Master\_0\_TX.  
***DMA\_HW\_IF\_I2C\_MASTER\_0\_RX*** I2C\_Master\_0\_RX.  
***DMA\_HW\_IF\_UART\_A\_TX*** UART\_A\_TX.  
***DMA\_HW\_IF\_UART\_A\_RX*** UART\_A\_RX.  
***DMA\_HW\_IF\_UART\_B\_TX*** UART\_B\_TX.  
***DMA\_HW\_IF\_UART\_B\_RX*** UART\_B\_RX.  
***DMA\_HW\_IF\_SPI\_MASTER\_0\_TX*** SPI\_Master\_0\_TX.  
***DMA\_HW\_IF\_SPI\_MASTER\_0\_RX*** SPI\_Master\_0\_RX.  
***DMA\_HW\_IF\_SPI\_MASTER\_1\_TX*** SPI\_Master\_1\_TX.  
***DMA\_HW\_IF\_SPI\_MASTER\_1\_RX*** SPI\_Master\_1\_RX.  
***DMA\_HW\_IF\_SPI\_SLAVE\_TX*** SPI\_Slave\_TX.  
***DMA\_HW\_IF\_SPI\_SLAVE\_RX*** SPI\_Slave\_RX.  
***DMA\_HW\_IF\_I2S\_PLAYBACK*** I2S\_Playback channel.  
***DMA\_HW\_IF\_I2S\_CAPTURE*** I2S\_Capture channel.  
***DMA\_HW\_IF\_I2C\_MASTER\_0\_TX*** I2C\_Master\_0\_TX.  
***DMA\_HW\_IF\_I2C\_MASTER\_0\_RX*** I2C\_Master\_0\_RX.  
***DMA\_HW\_IF\_I2C\_MASTER\_1\_TX*** I2C\_Master\_1\_TX.  
***DMA\_HW\_IF\_I2C\_MASTER\_1\_RX*** I2C\_Master\_1\_RX.

Definition at line 1419 of file qm\_soc\_regs.h.

#### 4.43.2.4 enum qm\_dma\_t

DMA instances.

Enumerator

**QM\_DMA\_0** DMA controller id.

**QM\_DMA\_NUM** Number of DMA controllers.

**QM\_DMA\_0** DMA controller id.

**QM\_DMA\_NUM** Number of DMA controllers.

Definition at line 1400 of file qm\_soc\_regs.h.

#### 4.43.2.5 enum qm\_flash\_t

Number of Flash controllers.

Definition at line 1246 of file qm\_soc\_regs.h.

#### 4.43.2.6 enum qm\_gpio\_t

Number of GPIO controllers.

Definition at line 1193 of file qm\_soc\_regs.h.

#### 4.43.2.7 enum qm\_i2c\_t

Number of I2C controllers.

Definition at line 1057 of file qm\_soc\_regs.h.

#### 4.43.2.8 enum qm\_pwm\_id\_t

PWM ID type.

Definition at line 781 of file qm\_soc\_regs.h.

#### 4.43.2.9 enum qm\_pwm\_t

Number of PWM / Timer controllers.

Definition at line 778 of file qm\_soc\_regs.h.

#### 4.43.2.10 enum qm\_rtc\_t

Number of RTC controllers.

Definition at line 1023 of file qm\_soc\_regs.h.

#### 4.43.2.11 enum qm\_scss\_aon\_t

Number of SCSS Always-on controllers.

Definition at line 492 of file qm\_soc\_regs.h.

#### 4.43.2.12 enum qm\_spi\_t

Number of SPI controllers.

Definition at line 914 of file qm\_soc\_regs.h.

#### 4.43.2.13 enum qm\_uart\_t

Number of UART controllers.

Definition at line 870 of file qm\_soc\_regs.h.

#### 4.43.2.14 enum `qm_wdt_t`

Number of WDT controllers.

Definition at line 830 of file `qm_soc_regs.h`.

### 4.43.3 Variable Documentation

#### 4.43.3.1 `qm_i2c_reg_t`\* `qm_i2c[QM_I2C_NUM]`

I2C register block.

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



## 4.44 Quark SE Voltage Regulators

Voltage Regulators Control.

### Functions

- int [vreg\\_aon\\_set\\_mode](#) (const vreg\_mode\_t mode)  
Set AON Voltage Regulator mode.
- int [vreg\\_plat3p3\\_set\\_mode](#) (const vreg\_mode\_t mode)  
Set Platform 3P3 Voltage Regulator mode.
- int [vreg\\_plat1p8\\_set\\_mode](#) (const vreg\_mode\_t mode)  
Set Platform 1P8 Voltage Regulator mode.
- int [vreg\\_host\\_set\\_mode](#) (const vreg\_mode\_t mode)  
Set Host Voltage Regulator mode.

### 4.44.1 Detailed Description

Voltage Regulators Control.

### 4.44.2 Function Documentation

#### 4.44.2.1 int vreg\_aon\_set\_mode ( const vreg\_mode\_t mode )

Set AON Voltage Regulator mode.

The AON Voltage Regulator is not a switching regulator and only acts as a linear regulator. VREG\_SWITCHING\_MODE is not a value mode for the AON Voltage Regulator.

#### Parameters

|                 |                   |                         |
|-----------------|-------------------|-------------------------|
| <code>in</code> | <code>mode</code> | Voltage Regulator mode. |
|-----------------|-------------------|-------------------------|

#### Returns

Standard errno return type for QMSI.

#### Return values

|                 |   |
|-----------------|---|
| <code>0</code>  | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 45 of file vreg.c.

#### 4.44.2.2 int vreg\_host\_set\_mode ( const vreg\_mode\_t mode )

Set Host Voltage Regulator mode.

#### Parameters

|                 |                   |                         |
|-----------------|-------------------|-------------------------|
| <code>in</code> | <code>mode</code> | Voltage Regulator mode. |
|-----------------|-------------------|-------------------------|

#### Returns

Standard errno return type for QMSI.

## Return values

|                 |   |
|-----------------|---|
| <i>0</i>        | on success.                                     |
| <i>Negative</i> | <a href="#">errno</a> for possible error codes. |

Definition at line 65 of file vreg.c.

4.44.2.3 `int vreg_plat1p8_set_mode ( const vreg_mode_t mode )`

Set Platform 1P8 Voltage Regulator mode.

## Parameters

|           |             |                         |
|-----------|-------------|-------------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 59 of file vreg.c.

Referenced by `power_soc_deep_sleep()`.

4.44.2.4 `int vreg_plat3p3_set_mode ( const vreg_mode_t mode )`

Set Platform 3P3 Voltage Regulator mode.

## Parameters

|           |             |                         |
|-----------|-------------|-------------------------|
| <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> | <a href="#">errno</a> for possible error codes. |

Definition at line 53 of file vreg.c.

Referenced by `power_soc_deep_sleep()`.

## 4.45 Syscalls

newlib syscalls implementation

### Functions

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

#### 4.45.1 Detailed Description

newlib syscalls implementation

#### 4.45.2 Function Documentation

##### 4.45.2.1 int `pico_printf` ( const char \* *format*, ... )

This is an minimally useful subset of the POSIX printf() function.

To reduce code size, this `pico_printf()` implementation only supports a few conversion specifiers:

- 'd', 'u': for signed and unsigned decimal numbers, respectively;
- 'x', 'X': for hexadecimal numbers, downcase and upcase;
- 's': for NULL terminated strings.

Other limitations:

- No flag specifier is implemented;
- No field width specifier is implemented;
- The only supported length modifier is 'l', which is parsed and ignored, on supported 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_soc_regs.h>
```

#### 5.1.1 Detailed Description

SS I2C Interrupt register map.

Definition at line 302 of file `qm_soc_regs.h`.

### 5.2 `int_ss_spi_reg_t` Struct Reference

SS SPI Interrupt register map.

```
#include <qm_soc_regs.h>
```

#### 5.2.1 Detailed Description

SS SPI Interrupt register map.

Definition at line 310 of file `qm_soc_regs.h`.

### 5.3 `mvic_reg_pad_t` Struct Reference

MVIC register structure.

```
#include <qm_soc_regs.h>
```

#### 5.3.1 Detailed Description

MVIC register structure.

Definition at line 1224 of file `qm_soc_regs.h`.

### 5.4 `pic_timer_reg_pad_t` Struct Reference

PIC timer register structure.

```
#include <qm_soc_regs.h>
```

#### 5.4.1 Detailed Description

PIC timer register structure.

Definition at line 1152 of file `qm_soc_regs.h`.

### 5.5 `qm_ac_config_t` Struct Reference

Analog Comparator configuration type.

```
#include <qm_comparator.h>
```

## Data Fields

- uint32\_t [int\\_en](#)  
*Interrupt enable.*
- uint32\_t [reference](#)  
*Reference voltage, 1b: VREF; 0b: AR\_PIN.*
- uint32\_t [polarity](#)  
*0b: input>ref; 1b: input<ref*
- uint32\_t [power](#)  
*1b: Normal mode; 0b:Power-down/Shutdown mode*
- void(\* [callback](#))(void \*data, uint32\_t int\_status)  
*Transfer callback.*
- void \* [callback\\_data](#)  
*Callback user data.*

### 5.5.1 Detailed Description

Analog Comparator configuration type.

Each bit in the registers controls a single Analog Comparator pin.

Definition at line 23 of file qm\_comparator.h.

### 5.5.2 Field Documentation

#### 5.5.2.1 void(\* qm\_ac\_config\_t::callback)(void \*data, uint32\_t int\_status)

Transfer callback.

##### Parameters

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

Definition at line 35 of file qm\_comparator.h.

Referenced by qm\_ac\_set\_config().

#### 5.5.2.2 void\* qm\_ac\_config\_t::callback\_data

Callback user data.

Definition at line 36 of file qm\_comparator.h.

Referenced by qm\_ac\_set\_config().

#### 5.5.2.3 uint32\_t qm\_ac\_config\_t::int\_en

Interrupt enable.

Definition at line 24 of file qm\_comparator.h.

Referenced by qm\_ac\_set\_config().

#### 5.5.2.4 uint32\_t qm\_ac\_config\_t::reference

Reference voltage, 1b: VREF; 0b: AR\_PIN.

Definition at line 25 of file qm\_comparator.h.

Referenced by qm\_ac\_set\_config().

## 5.6 qm\_adc\_config\_t Struct Reference

ADC configuration type.

```
#include <qm_adc.h>
```

### Data Fields

- [uint8\\_t window](#)  
*Sample interval in ADC clock cycles, defines the period to wait between the start of each sample and can be in the range [(resolution+2) - 255].*
- [qm\\_adc\\_resolution\\_t resolution](#)  
*12, 10, 8, 6-bit resolution.*

### 5.6.1 Detailed Description

ADC configuration type.

Definition at line 94 of file qm\_adc.h.

### 5.6.2 Field Documentation

#### 5.6.2.1 qm\_adc\_resolution\_t qm\_adc\_config\_t::resolution

12, 10, 8, 6-bit resolution.

Definition at line 101 of file qm\_adc.h.

Referenced by qm\_adc\_set\_config().

## 5.7 qm\_adc\_reg\_t Struct Reference

ADC register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [adc\\_seq0](#)  
*ADC Channel Sequence Table Entry 0.*
- QM\_RW uint32\_t [adc\\_seq1](#)  
*ADC Channel Sequence Table Entry 1.*
- QM\_RW uint32\_t [adc\\_seq2](#)  
*ADC Channel Sequence Table Entry 2.*
- QM\_RW uint32\_t [adc\\_seq3](#)  
*ADC Channel Sequence Table Entry 3.*
- QM\_RW uint32\_t [adc\\_seq4](#)  
*ADC Channel Sequence Table Entry 4.*
- QM\_RW uint32\_t [adc\\_seq5](#)  
*ADC Channel Sequence Table Entry 5.*
- QM\_RW uint32\_t [adc\\_seq6](#)  
*ADC Channel Sequence Table Entry 6.*
- QM\_RW uint32\_t [adc\\_seq7](#)  
*ADC Channel Sequence Table Entry 7.*

- QM\_RW uint32\_t [adc\\_cmd](#)  
*ADC Command Register.*
- QM\_RW uint32\_t [adc\\_intr\\_status](#)  
*ADC Interrupt Status Register.*
- QM\_RW uint32\_t [adc\\_intr\\_enable](#)  
*ADC Interrupt Enable Register.*
- QM\_RW uint32\_t [adc\\_sample](#)  
*ADC Sample Register.*
- QM\_RW uint32\_t [adc\\_calibration](#)  
*ADC Calibration Data Register.*
- QM\_RW uint32\_t [adc\\_fifo\\_count](#)  
*ADC FIFO Count Register.*
- QM\_RW uint32\_t [adc\\_op\\_mode](#)  
*ADC Operating Mode Register.*

### 5.7.1 Detailed Description

ADC register map.

Definition at line 979 of file `qm_soc_regs.h`.

## 5.8 qm\_adc\_xfer\_t Struct Reference

ADC transfer type.

```
#include <qm_adc.h>
```

### Data Fields

- [qm\\_adc\\_channel\\_t](#) \* `ch`  
*Channel sequence array (1-32 channels).*
- [uint8\\_t](#) `ch_len`  
*Number of channels in the above array.*
- [qm\\_adc\\_sample\\_t](#) \* `samples`  
*Array to store samples.*
- [uint32\\_t](#) `samples_len`  
*Length of sample array.*
- `void(* callback)(void *data, int error, qm\_adc\_status\_t status, qm\_adc\_cb\_source\_t source)`  
*Transfer callback.*
- `void * callback\_data`  
*Callback user data.*

### 5.8.1 Detailed Description

ADC transfer type.

Definition at line 107 of file `qm_adc.h`.

### 5.8.2 Field Documentation

#### 5.8.2.1 `void(* qm\_adc\_xfer\_t::callback)(void *data, int error, qm\_adc\_status\_t status, qm\_adc\_cb\_source\_t source)`

Transfer callback.

Called when a conversion is performed or an error is detected.

## Parameters

|    |               |  |
|----|---------------|--|
| in | <i>data</i>   | The callback user data.  |
| in | <i>error</i>  | 0 on success. Negative <a href="#">errno</a> 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\_aonpt\_config\_t Struct Reference

Always-on Periodic Timer configuration type.

```
#include <qm_aon_counters.h>
```

## Data Fields

- [uint32\\_t count](#)  
*Time to count down from in clock cycles.*
- [bool int\\_en](#)  
*Enable/disable the interrupts.*
- [void\(\\* callback\)\(void \\*data\)](#)  
*User callback.*
- [void \\* callback\\_data](#)  
*Callback data.*



### 5.9.1 Detailed Description

Always-on Periodic Timer configuration type.

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

### 5.9.2 Field Documentation

#### 5.9.2.1 `void(* qm_aonpt_config_t::callback)(void *data)`

User callback.

##### Parameters

|                 |                   |                    |
|-----------------|-------------------|--------------------|
| <code>in</code> | <code>data</code> | User defined data. |
|-----------------|-------------------|--------------------|

Definition at line 30 of file `qm_aon_counters.h`.

Referenced by `qm_aonpt_set_config()`.

#### 5.9.2.2 `void* qm_aonpt_config_t::callback_data`

Callback data.

Definition at line 31 of file `qm_aon_counters.h`.

Referenced by `qm_aonpt_set_config()`.

#### 5.9.2.3 `uint32_t qm_aonpt_config_t::count`

Time to count down from in clock cycles.

Definition at line 22 of file `qm_aon_counters.h`.

Referenced by `qm_aonpt_set_config()`.

#### 5.9.2.4 `bool qm_aonpt_config_t::int_en`

Enable/disable the interrupts.

Definition at line 23 of file `qm_aon_counters.h`.

Referenced by `qm_aonpt_set_config()`.

## 5.10 `qm_dma_chan_reg_t` Struct Reference

DMA channel register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW `uint32_t sar_low`  
SAR.
- QM\_RW `uint32_t sar_high`  
SAR.
- QM\_RW `uint32_t dar_low`  
DAR.
- QM\_RW `uint32_t dar_high`  
DAR.
- QM\_RW `uint32_t llp_low`

- QM\_RW uint32\_t `llp_high`  
*LLP.*
- QM\_RW uint32\_t `ctrl_low`  
*LLP.*
- QM\_RW uint32\_t `ctrl_high`  
*CTL.*
- QM\_RW uint32\_t `src_stat_low`  
*CTL.*
- QM\_RW uint32\_t `src_stat_high`  
*SSTAT.*
- QM\_RW uint32\_t `dst_stat_low`  
*SSTAT.*
- QM\_RW uint32\_t `dst_stat_high`  
*DSTAT.*
- QM\_RW uint32\_t `src_stat_addr_low`  
*DSTAT.*
- QM\_RW uint32\_t `src_stat_addr_high`  
*SSTATAR.*
- QM\_RW uint32\_t `dst_stat_addr_low`  
*SSTATAR.*
- QM\_RW uint32\_t `dst_stat_addr_high`  
*DSTATAR.*
- QM\_RW uint32\_t `cfg_low`  
*DSTATAR.*
- QM\_RW uint32\_t `cfg_high`  
*CFG.*
- QM\_RW uint32\_t `src_sg_low`  
*CFG.*
- QM\_RW uint32\_t `src_sg_high`  
*SGR.*
- QM\_RW uint32\_t `dst_sg_low`  
*SGR.*
- QM\_RW uint32\_t `dst_sg_high`  
*DSR.*

### 5.10.1 Detailed Description

DMA channel register map.

Definition at line 1320 of file `qm_soc_regs.h`.

## 5.11 qm\_dma\_channel\_config\_t Struct Reference

DMA channel configuration structure.

```
#include <qm_dma.h>
```

## Data Fields

- [qm\\_dma\\_handshake\\_interface\\_t handshake\\_interface](#)  
*DMA channel handshake interface ID.*
- [qm\\_dma\\_handshake\\_polarity\\_t handshake\\_polarity](#)  
*DMA channel handshake polarity.*
- [qm\\_dma\\_channel\\_direction\\_t channel\\_direction](#)  
*DMA channel direction.*
- [qm\\_dma\\_transfer\\_width\\_t source\\_transfer\\_width](#)  
*DMA source transfer width.*
- [qm\\_dma\\_transfer\\_width\\_t destination\\_transfer\\_width](#)  
*DMA destination transfer width.*
- [qm\\_dma\\_burst\\_length\\_t source\\_burst\\_length](#)  
*DMA source burst length.*
- [qm\\_dma\\_burst\\_length\\_t destination\\_burst\\_length](#)  
*DMA destination burst length.*
- `void(* client\_callback )(void *callback\_context, uint32_t len, int error_code)`  
*Client callback for DMA transfer ISR.*
- `void * callback\_context`  
*DMA client context passed to the callbacks.*

### 5.11.1 Detailed Description

DMA channel configuration structure.

Definition at line 66 of file `qm_dma.h`.

### 5.11.2 Field Documentation

#### 5.11.2.1 `void(* qm\_dma\_channel\_config\_t::client\_callback)(void *callback\_context, uint32_t len, int error_code)`

Client callback for DMA transfer ISR.

#### Parameters

|    |   |                          |
|----|---|--------------------------|
| in | <i><a href="#">callback_context</a></i> | DMA client context.      |
| in | <i><a href="#">len</a></i>              | Data length transferred. |
| in | <i><a href="#">error</a></i>            | Error code.              |

Definition at line 95 of file `qm_dma.h`.

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

## 5.12 [qm\\_dma\\_int\\_reg\\_t](#) Struct Reference

DMA interrupt register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [raw\\_tfr\\_low](#)  
*RawTfr.*
- QM\_RW uint32\_t [raw\\_tfr\\_high](#)

- RawTfr.*

  - QM\_RW uint32\_t [raw\\_block\\_low](#)
- RawBlock.*

  - QM\_RW uint32\_t [raw\\_block\\_high](#)
- RawBlock.*

  - QM\_RW uint32\_t [raw\\_src\\_trans\\_low](#)
- RawSrcTran.*

  - QM\_RW uint32\_t [raw\\_src\\_trans\\_high](#)
- RawSrcTran.*

  - QM\_RW uint32\_t [raw\\_dst\\_trans\\_low](#)
- RawDstTran.*

  - QM\_RW uint32\_t [raw\\_dst\\_trans\\_high](#)
- RawDstTran.*

  - QM\_RW uint32\_t [raw\\_err\\_low](#)
- RawErr.*

  - QM\_RW uint32\_t [raw\\_err\\_high](#)
- RawErr.*

  - QM\_RW uint32\_t [status\\_tfr\\_low](#)
- StatusTfr.*

  - QM\_RW uint32\_t [status\\_tfr\\_high](#)
- StatusTfr.*

  - QM\_RW uint32\_t [status\\_block\\_low](#)
- StatusBlock.*

  - QM\_RW uint32\_t [status\\_block\\_high](#)
- StatusBlock.*

  - QM\_RW uint32\_t [status\\_src\\_trans\\_low](#)
- StatusSrcTran.*

  - QM\_RW uint32\_t [status\\_src\\_trans\\_high](#)
- StatusSrcTran.*

  - QM\_RW uint32\_t [status\\_dst\\_trans\\_low](#)
- StatusDstTran.*

  - QM\_RW uint32\_t [status\\_dst\\_trans\\_high](#)
- StatusDstTran.*

  - QM\_RW uint32\_t [status\\_err\\_low](#)
- StatusErr.*

  - QM\_RW uint32\_t [status\\_err\\_high](#)
- StatusErr.*

  - QM\_RW uint32\_t [mask\\_tfr\\_low](#)
- MaskTfr.*

  - QM\_RW uint32\_t [mask\\_tfr\\_high](#)
- MaskTfr.*

  - QM\_RW uint32\_t [mask\\_block\\_low](#)
- MaskBlock.*

  - QM\_RW uint32\_t [mask\\_block\\_high](#)
- MaskBlock.*

  - QM\_RW uint32\_t [mask\\_src\\_trans\\_low](#)
- MaskSrcTran.*

  - QM\_RW uint32\_t [mask\\_src\\_trans\\_high](#)
- MaskSrcTran.*

  - QM\_RW uint32\_t [mask\\_dst\\_trans\\_low](#)
- MaskDstTran.*

- QM\_RW uint32\_t [mask\\_dst\\_trans\\_high](#)  
*MaskDstTran.*
- QM\_RW uint32\_t [mask\\_err\\_low](#)  
*MaskErr.*
- QM\_RW uint32\_t [mask\\_err\\_high](#)  
*MaskErr.*
- QM\_RW uint32\_t [clear\\_tfr\\_low](#)  
*ClearTfr.*
- QM\_RW uint32\_t [clear\\_tfr\\_high](#)  
*ClearTfr.*
- QM\_RW uint32\_t [clear\\_block\\_low](#)  
*ClearBlock.*
- QM\_RW uint32\_t [clear\\_block\\_high](#)  
*ClearBlock.*
- QM\_RW uint32\_t [clear\\_src\\_trans\\_low](#)  
*ClearSrcTran.*
- QM\_RW uint32\_t [clear\\_src\\_trans\\_high](#)  
*ClearSrcTran.*
- QM\_RW uint32\_t [clear\\_dst\\_trans\\_low](#)  
*ClearDstTran.*
- QM\_RW uint32\_t [clear\\_dst\\_trans\\_high](#)  
*ClearDstTran.*
- QM\_RW uint32\_t [clear\\_err\\_low](#)  
*ClearErr.*
- QM\_RW uint32\_t [clear\\_err\\_high](#)  
*ClearErr.*
- QM\_RW uint32\_t [status\\_int\\_low](#)  
*StatusInt.*
- QM\_RW uint32\_t [status\\_int\\_high](#)  
*StatusInt.*

### 5.12.1 Detailed Description

DMA interrupt register map.

Definition at line 1387 of file `qm_soc_regs.h`.

### 5.13 `qm_dma_misc_reg_t` Struct Reference

DMA miscellaneous register map.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- QM\_RW uint32\_t [cfg\\_low](#)  
*DmaCfgReg.*
- QM\_RW uint32\_t [cfg\\_high](#)  
*DmaCfgReg.*
- QM\_RW uint32\_t [chan\\_en\\_low](#)  
*ChEnReg.*

- QM\_RW uint32\_t [chan\\_en\\_high](#)  
*ChEnReg.*
- QM\_RW uint32\_t [id\\_low](#)  
*DmaldReg.*
- QM\_RW uint32\_t [id\\_high](#)  
*DmaldReg.*
- QM\_RW uint32\_t [test\\_low](#)  
*DmaTestReg.*
- QM\_RW uint32\_t [test\\_high](#)  
*DmaTestReg.*
- QM\_RW uint32\_t [reserved](#) [4]  
*Reserved.*

### 5.13.1 Detailed Description

DMA miscellaneous register map.

Definition at line 1437 of file qm\_soc\_regs.h.

## 5.14 qm\_dma\_transfer\_t Struct Reference

DMA transfer configuration structure.

```
#include <qm_dma.h>
```

### Data Fields

- uint32\_t [block\\_size](#)  
*DMA block size, Min = 1, Max = 4095.*
- uint32\_t \* [source\\_address](#)  
*DMA source transfer address.*
- uint32\_t \* [destination\\_address](#)  
*DMA destination transfer address.*

### 5.14.1 Detailed Description

DMA transfer configuration structure.

Definition at line 105 of file qm\_dma.h.

### 5.14.2 Field Documentation

#### 5.14.2.1 uint32\_t qm\_dma\_transfer\_t::block\_size

DMA block size, Min = 1, Max = 4095.

Definition at line 106 of file qm\_dma.h.

Referenced by `qm_dma_transfer_mem_to_mem()`, `qm_dma_transfer_set_config()`, `qm_spi_dma_transfer()`, `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

#### 5.14.2.2 `uint32_t* qm_dma_transfer_t::destination_address`

DMA destination transfer address.

Definition at line 108 of file `qm_dma.h`.

Referenced by `qm_dma_transfer_mem_to_mem()`, `qm_dma_transfer_set_config()`, `qm_spi_dma_transfer()`, `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

#### 5.14.2.3 `uint32_t* qm_dma_transfer_t::source_address`

DMA source transfer address.

Definition at line 107 of file `qm_dma.h`.

Referenced by `qm_dma_transfer_mem_to_mem()`, `qm_dma_transfer_set_config()`, `qm_spi_dma_transfer()`, `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

### 5.15 `qm_flash_config_t` Struct Reference

Flash configuration structure.

```
#include <qm_flash.h>
```

#### Data Fields

- [uint8\\_t wait\\_states](#)  
*Read wait state.*
- [uint8\\_t us\\_count](#)  
*Number of clocks in a microsecond.*
- [qm\\_flash\\_disable\\_t write\\_disable](#)  
*Write disable.*

#### 5.15.1 Detailed Description

Flash configuration structure.

Definition at line 80 of file `qm_flash.h`.

#### 5.15.2 Field Documentation

##### 5.15.2.1 `uint8_t qm_flash_config_t::us_count`

Number of clocks in a microsecond.

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

Referenced by `qm_flash_set_config()`.

##### 5.15.2.2 `uint8_t qm_flash_config_t::wait_states`

Read wait state.

Definition at line 81 of file `qm_flash.h`.

Referenced by `qm_flash_set_config()`.

##### 5.15.2.3 `qm_flash_disable_t qm_flash_config_t::write_disable`

Write disable.

Definition at line 83 of file qm\_flash.h.

Referenced by qm\_flash\_set\_config().

## 5.16 qm\_flash\_reg\_t Struct Reference

Flash register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [tmg\\_ctrl](#)  
*TMG\_CTRL.*
- QM\_RW uint32\_t [rom\\_wr\\_ctrl](#)  
*ROM\_WR\_CTRL.*
- QM\_RW uint32\_t [rom\\_wr\\_data](#)  
*ROM\_WR\_DATA.*
- QM\_RW uint32\_t [flash\\_wr\\_ctrl](#)  
*FLASH\_WR\_CTRL.*
- QM\_RW uint32\_t [flash\\_wr\\_data](#)  
*FLASH\_WR\_DATA.*
- QM\_RW uint32\_t [flash\\_stts](#)  
*FLASH\_STTS.*
- QM\_RW uint32\_t [ctrl](#)  
*CTRL.*
- QM\_RW uint32\_t [fpr\\_rd\\_cfg](#) [4]  
*4 FPR\_RD\_CFG registers*
- QM\_RW uint32\_t [mpr\\_wr\\_cfg](#)  
*Flash Write Protection Control Register.*
- QM\_RW uint32\_t [mpr\\_vsts](#)  
*Protection Status Register.*
- QM\_RW uint32\_t [mpr\\_vdata](#)  
*MPR Violation Data Value Register.*

### 5.16.1 Detailed Description

Flash register map.

Definition at line 1051 of file qm\_soc\_regs.h.

## 5.17 qm\_fpr\_config\_t Struct Reference

Flash Protection Region configuration structure.

```
#include <qm_fpr.h>
```

### Data Fields

- [qm\\_fpr\\_en\\_t en\\_mask](#)  
*Enable/lock bitmask.*
- [qm\\_fpr\\_read\\_allow\\_t allow\\_agents](#)



*Per-agent read enable bitmask.*

- [uint8\\_t up\\_bound](#)  
*1KB-aligned upper Flash phys addr.*
- [uint8\\_t low\\_bound](#)  
*1KB-aligned lower Flash phys addr.*

### 5.17.1 Detailed Description

Flash Protection Region configuration structure.

Definition at line 80 of file `qm_fpr.h`.

### 5.17.2 Field Documentation

#### 5.17.2.1 `qm_fpr_read_allow_t qm_fpr_config_t::allow_agents`

Per-agent read enable bitmask.

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

Referenced by `qm_fpr_set_config()`.

#### 5.17.2.2 `qm_fpr_en_t qm_fpr_config_t::en_mask`

Enable/lock bitmask.

Definition at line 81 of file `qm_fpr.h`.

Referenced by `qm_fpr_set_config()`.

#### 5.17.2.3 `uint8_t qm_fpr_config_t::low_bound`

1KB-aligned lower Flash phys addr.

Definition at line 84 of file `qm_fpr.h`.

Referenced by `qm_fpr_set_config()`.

#### 5.17.2.4 `uint8_t qm_fpr_config_t::up_bound`

1KB-aligned upper Flash phys addr.

Definition at line 83 of file `qm_fpr.h`.

Referenced by `qm_fpr_set_config()`.

## 5.18 `qm_gpio_port_config_t` Struct Reference

GPIO port configuration type.

```
#include <qm_gpio.h>
```

### Data Fields

- [uint32\\_t direction](#)  
*GPIO direction, 0b: input, 1b: output.*
- [uint32\\_t int\\_en](#)  
*Interrupt enable.*
- [uint32\\_t int\\_type](#)  
*Interrupt type, 0b: level; 1b: edge.*

- uint32\_t [int\\_polarity](#)  
*Interrupt polarity, 0b: low, 1b: high.*
- uint32\_t [int\\_debounce](#)  
*Interrupt debounce on/off.*
- uint32\_t [int\\_bothedge](#)  
*Interrupt on rising and falling edges.*
- void(\* [callback](#) )(void \*data, uint32\_t int\_status)  
*Transfer callback.*
- void \* [callback\\_data](#)  
*Callback user data.*

### 5.18.1 Detailed Description

GPIO port configuration type.

Each bit in the registers control a GPIO pin.

Definition at line 32 of file qm\_gpio.h.

### 5.18.2 Field Documentation

#### 5.18.2.1 void(\* qm\_gpio\_port\_config\_t::callback)(void \*data, uint32\_t int\_status)

Transfer callback.

Parameters

|    |                   |                        |
|----|-------------------|------------------------|
| in | <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\(\)](#).

#### 5.18.2.2 void\* qm\_gpio\_port\_config\_t::callback\_data

Callback user data.

Definition at line 47 of file qm\_gpio.h.

Referenced by [qm\\_gpio\\_set\\_config\(\)](#).

#### 5.18.2.3 uint32\_t qm\_gpio\_port\_config\_t::direction

GPIO direction, 0b: input, 1b: output.

Definition at line 33 of file qm\_gpio.h.

Referenced by [qm\\_gpio\\_set\\_config\(\)](#).

#### 5.18.2.4 uint32\_t qm\_gpio\_port\_config\_t::int\_bothedge

Interrupt on rising and falling edges.

Definition at line 38 of file qm\_gpio.h.

Referenced by [qm\\_gpio\\_set\\_config\(\)](#).

#### 5.18.2.5 uint32\_t qm\_gpio\_port\_config\_t::int\_debounce

Interrupt debounce on/off.

Definition at line 37 of file qm\_gpio.h.

Referenced by `qm_gpio_set_config()`.

#### 5.18.2.6 `uint32_t qm_gpio_port_config_t::int_en`

Interrupt enable.

Definition at line 34 of file `qm_gpio.h`.

Referenced by `qm_gpio_set_config()`.

#### 5.18.2.7 `uint32_t qm_gpio_port_config_t::int_polarity`

Interrupt polarity, 0b: low, 1b: high.

Definition at line 36 of file `qm_gpio.h`.

Referenced by `qm_gpio_set_config()`.

#### 5.18.2.8 `uint32_t qm_gpio_port_config_t::int_type`

Interrupt type, 0b: level; 1b: edge.

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

Referenced by `qm_gpio_set_config()`.

## 5.19 `qm_gpio_reg_t` Struct Reference

GPIO register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW `uint32_t gpio_swporta_dr`  
*Port A Data.*
- QM\_RW `uint32_t gpio_swporta_ddr`  
*Port A Data Direction.*
- QM\_RW `uint32_t gpio_inten`  
*Interrupt Enable.*
- QM\_RW `uint32_t gpio_intmask`  
*Interrupt Mask.*
- QM\_RW `uint32_t gpio_inttype_level`  
*Interrupt Type.*
- QM\_RW `uint32_t gpio_int_polarity`  
*Interrupt Polarity.*
- QM\_RW `uint32_t gpio_intstatus`  
*Interrupt Status.*
- QM\_RW `uint32_t gpio_raw_intstatus`  
*Raw Interrupt Status.*
- QM\_RW `uint32_t gpio_debounce`  
*Debounce Enable.*
- QM\_RW `uint32_t gpio_porta_eoi`  
*Clear Interrupt.*
- QM\_RW `uint32_t gpio_ext_porta`  
*Port A External Port.*
- QM\_RW `uint32_t gpio_ls_sync`  
*Synchronization Level.*

- QM\_RW uint32\_t [gpio\\_id\\_code](#)  
*GPIO ID code.*
- QM\_RW uint32\_t [gpio\\_int\\_bothedge](#)  
*Interrupt both edge type.*
- QM\_RW uint32\_t [gpio\\_ver\\_id\\_code](#)  
*GPIO Component Version.*
- QM\_RW uint32\_t [gpio\\_config\\_reg2](#)  
*GPIO Configuration Register 2.*
- QM\_RW uint32\_t [gpio\\_config\\_reg1](#)  
*GPIO Configuration Register 1.*
- QM\_RW uint32\_t [gpio\\_swporta\\_ctl](#)  
*Port A Data Source.*

### 5.19.1 Detailed Description

GPIO register map.

Definition at line 929 of file `qm_soc_regs.h`.

### 5.19.2 Field Documentation

#### 5.19.2.1 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_config\_reg1

GPIO Configuration Register 1.

Definition at line 948 of file `qm_soc_regs.h`.

#### 5.19.2.2 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_config\_reg2

GPIO Configuration Register 2.

Definition at line 947 of file `qm_soc_regs.h`.

#### 5.19.2.3 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_debounce

Debounce Enable.

Definition at line 939 of file `qm_soc_regs.h`.

Referenced by `qm_gpio_set_config()`.

#### 5.19.2.4 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_ext\_porta

Port A External Port.

Definition at line 941 of file `qm_soc_regs.h`.

#### 5.19.2.5 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_id\_code

GPIO ID code.

Definition at line 944 of file `qm_soc_regs.h`.

#### 5.19.2.6 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_int\_bothedge

Interrupt both edge type.

Definition at line 945 of file `qm_soc_regs.h`.

Referenced by `qm_gpio_set_config()`.

#### 5.19.2.7 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_int\_polarity

Interrupt Polarity.

Definition at line 936 of file qm\_soc\_regs.h.

Referenced by qm\_gpio\_set\_config().

#### 5.19.2.8 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_inten

Interrupt Enable.

Definition at line 933 of file qm\_soc\_regs.h.

Referenced by qm\_gpio\_set\_config().

#### 5.19.2.9 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_intmask

Interrupt Mask.

Definition at line 934 of file qm\_soc\_regs.h.

Referenced by qm\_gpio\_set\_config().

#### 5.19.2.10 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_intstatus

Interrupt Status.

Definition at line 937 of file qm\_soc\_regs.h.

#### 5.19.2.11 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_inttype\_level

Interrupt Type.

Definition at line 935 of file qm\_soc\_regs.h.

Referenced by qm\_gpio\_set\_config().

#### 5.19.2.12 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_ls\_sync

Synchronization Level.

Definition at line 943 of file qm\_soc\_regs.h.

#### 5.19.2.13 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_porta\_eoi

Clear Interrupt.

Definition at line 940 of file qm\_soc\_regs.h.

#### 5.19.2.14 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_raw\_intstatus

Raw Interrupt Status.

Definition at line 938 of file qm\_soc\_regs.h.

#### 5.19.2.15 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_swporta\_ddr

Port A Data Direction.

Definition at line 931 of file qm\_soc\_regs.h.

Referenced by qm\_gpio\_set\_config().

#### 5.19.2.16 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_swporta\_dr

Port A Data.

Definition at line 930 of file qm\_soc\_regs.h.

## 5.19.2.17 QM\_RW uint32\_t qm\_gpio\_reg\_t::gpio\_ver\_id\_code

GPIO Component Version.

Definition at line 946 of file qm\_soc\_regs.h.

## 5.20 qm\_i2c\_config\_t Struct Reference

I2C configuration type.

```
#include <qm_i2c.h>
```

## Data Fields

- [qm\\_i2c\\_speed\\_t speed](#)  
*Standard, Fast Mode.*
- [qm\\_i2c\\_addr\\_t address\\_mode](#)  
*7 or 10 bit addressing.*
- [qm\\_i2c\\_mode\\_t mode](#)  
*Master or slave mode.*
- [uint16\\_t slave\\_addr](#)  
*I2C address when in slave mode.*

## 5.20.1 Detailed Description

I2C configuration type.

Definition at line 88 of file qm\_i2c.h.

## 5.20.2 Field Documentation

## 5.20.2.1 qm\_i2c\_addr\_t qm\_i2c\_config\_t::address\_mode

7 or 10 bit addressing.

Definition at line 90 of file qm\_i2c.h.

Referenced by [qm\\_i2c\\_set\\_config\(\)](#).

## 5.20.2.2 qm\_i2c\_mode\_t qm\_i2c\_config\_t::mode

Master or slave mode.

Definition at line 91 of file qm\_i2c.h.

Referenced by [qm\\_i2c\\_set\\_config\(\)](#).

## 5.20.2.3 uint16\_t qm\_i2c\_config\_t::slave\_addr

I2C address when in slave mode.

Definition at line 92 of file qm\_i2c.h.

Referenced by [qm\\_i2c\\_set\\_config\(\)](#).

## 5.20.2.4 qm\_i2c\_speed\_t qm\_i2c\_config\_t::speed

Standard, Fast Mode.

Definition at line 89 of file qm\_i2c.h.

Referenced by [qm\\_i2c\\_set\\_config\(\)](#).

## 5.21 qm\_i2c\_reg\_t Struct Reference

I2C register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [ic\\_con](#)  
*Control Register.*
- QM\_RW uint32\_t [ic\\_tar](#)  
*Master Target Address.*
- QM\_RW uint32\_t [ic\\_sar](#)  
*Slave Address.*
- QM\_RW uint32\_t [ic\\_hs\\_maddr](#)  
*High Speed Master ID.*
- QM\_RW uint32\_t [ic\\_data\\_cmd](#)  
*Data Buffer and Command.*
- QM\_RW uint32\_t [ic\\_ss\\_scl\\_hcnt](#)  
*Standard Speed Clock SCL High Count.*
- QM\_RW uint32\_t [ic\\_ss\\_scl\\_lcnt](#)  
*Standard Speed Clock SCL Low Count.*
- QM\_RW uint32\_t [ic\\_fs\\_scl\\_hcnt](#)  
*Fast Speed Clock SCL High Count.*
- QM\_RW uint32\_t [ic\\_fs\\_scl\\_lcnt](#)  
*Fast Speed I2C Clock SCL Low Count.*
- QM\_RW uint32\_t [ic\\_hs\\_scl\\_hcnt](#)  
*High Speed I2C Clock SCL High Count.*
- QM\_RW uint32\_t [ic\\_hs\\_scl\\_lcnt](#)  
*High Speed I2C Clock SCL Low Count.*
- QM\_RW uint32\_t [ic\\_intr\\_stat](#)  
*Interrupt Status.*
- QM\_RW uint32\_t [ic\\_intr\\_mask](#)  
*Interrupt Mask.*
- QM\_RW uint32\_t [ic\\_raw\\_intr\\_stat](#)  
*Raw Interrupt Status.*
- QM\_RW uint32\_t [ic\\_rx\\_tl](#)  
*Receive FIFO Threshold Level.*
- QM\_RW uint32\_t [ic\\_tx\\_tl](#)  
*Transmit FIFO Threshold Level.*
- QM\_RW uint32\_t [ic\\_clr\\_intr](#)  
*Clear Combined and Individual Interrupt.*
- QM\_RW uint32\_t [ic\\_clr\\_rx\\_under](#)  
*Clear RX\_UNDER Interrupt.*
- QM\_RW uint32\_t [ic\\_clr\\_rx\\_over](#)  
*Clear RX\_OVER Interrupt.*
- QM\_RW uint32\_t [ic\\_clr\\_tx\\_over](#)  
*Clear TX\_OVER Interrupt.*
- QM\_RW uint32\_t [ic\\_clr\\_rd\\_req](#)  
*Clear RD\_REQ Interrupt.*
- QM\_RW uint32\_t [ic\\_clr\\_tx\\_abrt](#)

- Clear TX\_ABRT Interrupt.*

  - QM\_RW uint32\_t [ic\\_clr\\_rx\\_done](#)  
*Clear RX\_DONE Interrupt.*
  - QM\_RW uint32\_t [ic\\_clr\\_activity](#)  
*Clear ACTIVITY Interrupt.*
  - QM\_RW uint32\_t [ic\\_clr\\_stop\\_det](#)  
*Clear STOP\_DET Interrupt.*
  - QM\_RW uint32\_t [ic\\_clr\\_start\\_det](#)  
*Clear START\_DET Interrupt.*
  - QM\_RW uint32\_t [ic\\_clr\\_gen\\_call](#)  
*Clear GEN\_CALL Interrupt.*
  - QM\_RW uint32\_t [ic\\_enable](#)  
*Enable.*
  - QM\_RW uint32\_t [ic\\_status](#)  
*Status.*
  - QM\_RW uint32\_t [ic\\_txflr](#)  
*Transmit FIFO Level.*
  - QM\_RW uint32\_t [ic\\_rxflr](#)  
*Receive FIFO Level.*
  - QM\_RW uint32\_t [ic\\_sda\\_hold](#)  
*SDA Hold.*
  - QM\_RW uint32\_t [ic\\_tx\\_abrt\\_source](#)  
*Transmit Abort Source.*
  - QM\_RW uint32\_t [ic\\_dma\\_cr](#)  
*SDA Setup.*
  - QM\_RW uint32\_t [ic\\_dma\\_tdlr](#)  
*DMA Transmit Data Level Register.*
  - QM\_RW uint32\_t [ic\\_dma\\_rdlr](#)  
*I2C Receive Data Level Register.*
  - QM\_RW uint32\_t [ic\\_sda\\_setup](#)  
*SDA Setup.*
  - QM\_RW uint32\_t [ic\\_ack\\_general\\_call](#)  
*General Call Ack.*
  - QM\_RW uint32\_t [ic\\_enable\\_status](#)  
*Enable Status.*
  - QM\_RW uint32\_t [ic\\_fs\\_spklen](#)  
*SS and FS Spike Suppression Limit.*
  - QM\_RW uint32\_t [ic\\_hs\\_spklen](#)  
*HS spike suppression limit.*
  - QM\_RW uint32\_t [ic\\_clr\\_restart\\_det](#)  
*clear the RESTART\_DET interrupt.*
  - QM\_RW uint32\_t [ic\\_comp\\_param\\_1](#)  
*Configuration Parameters.*
  - QM\_RW uint32\_t [ic\\_comp\\_version](#)  
*Component Version.*
  - QM\_RW uint32\_t [ic\\_comp\\_type](#)  
*Component Type.*



### 5.21.1 Detailed Description

I2C register map.

Definition at line 793 of file qm\_soc\_regs.h.

### 5.21.2 Field Documentation

#### 5.21.2.1 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_ack\_general\_call

General Call Ack.

Definition at line 838 of file qm\_soc\_regs.h.

#### 5.21.2.2 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_activity

Clear ACTIVITY Interrupt.

Definition at line 823 of file qm\_soc\_regs.h.

#### 5.21.2.3 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_gen\_call

Clear GEN\_CALL Interrupt.

Definition at line 826 of file qm\_soc\_regs.h.

#### 5.21.2.4 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_intr

Clear Combined and Individual Interrupt.

Definition at line 816 of file qm\_soc\_regs.h.

#### 5.21.2.5 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_rd\_req

Clear RD\_REQ Interrupt.

Definition at line 820 of file qm\_soc\_regs.h.

#### 5.21.2.6 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_restart\_det

clear the RESTART\_DET interrupt.

Definition at line 843 of file qm\_soc\_regs.h.

#### 5.21.2.7 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_rx\_done

Clear RX\_DONE Interrupt.

Definition at line 822 of file qm\_soc\_regs.h.

#### 5.21.2.8 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_rx\_over

Clear RX\_OVER Interrupt.

Definition at line 818 of file qm\_soc\_regs.h.

#### 5.21.2.9 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_rx\_under

Clear RX\_UNDER Interrupt.

Definition at line 817 of file qm\_soc\_regs.h.

#### 5.21.2.10 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_start\_det

Clear START\_DET Interrupt.

Definition at line 825 of file qm\_soc\_regs.h.

#### 5.21.2.11 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_stop\_det

Clear STOP\_DET Interrupt.

Definition at line 824 of file qm\_soc\_regs.h.

#### 5.21.2.12 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_tx\_abrt

Clear TX\_ABRT Interrupt.

Definition at line 821 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_master\_read(), and qm\_i2c\_master\_write().

#### 5.21.2.13 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_clr\_tx\_over

Clear TX\_OVER Interrupt.

Definition at line 819 of file qm\_soc\_regs.h.

#### 5.21.2.14 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_comp\_param\_1

Configuration Parameters.

Definition at line 845 of file qm\_soc\_regs.h.

#### 5.21.2.15 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_comp\_type

Component Type.

Definition at line 847 of file qm\_soc\_regs.h.

#### 5.21.2.16 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_comp\_version

Component Version.

Definition at line 846 of file qm\_soc\_regs.h.

#### 5.21.2.17 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_con

Control Register.

Definition at line 794 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_set\_config(), and qm\_i2c\_set\_speed().

#### 5.21.2.18 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_data\_cmd

Data Buffer and Command.

Definition at line 798 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_master\_read(), and qm\_i2c\_master\_write().

#### 5.21.2.19 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_dma\_cr

SDA Setup.

Definition at line 834 of file qm\_soc\_regs.h.

#### 5.21.2.20 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_dma\_rdlr

I2C Receive Data Level Register.

Definition at line 836 of file qm\_soc\_regs.h.

**5.21.2.21 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_dma\_tdlr**

DMA Transmit Data Level Register.

Definition at line 835 of file qm\_soc\_regs.h.

**5.21.2.22 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_enable**

Enable.

Definition at line 827 of file qm\_soc\_regs.h.

**5.21.2.23 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_enable\_status**

Enable Status.

Definition at line 839 of file qm\_soc\_regs.h.

**5.21.2.24 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_fs\_scl\_hcnt**

Fast Speed Clock SCL High Count.

Definition at line 803 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_set\_config(), and qm\_i2c\_set\_speed().

**5.21.2.25 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_fs\_scl\_lcnt**

Fast Speed I2C Clock SCL Low Count.

Definition at line 805 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_set\_config(), and qm\_i2c\_set\_speed().

**5.21.2.26 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_fs\_spklen**

SS and FS Spike Suppression Limit.

Definition at line 840 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_set\_config(), and qm\_i2c\_set\_speed().

**5.21.2.27 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_hs\_maddr**

High Speed Master ID.

Definition at line 797 of file qm\_soc\_regs.h.

**5.21.2.28 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_hs\_scl\_hcnt**

High Speed I2C Clock SCL High Count.

Definition at line 807 of file qm\_soc\_regs.h.

**5.21.2.29 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_hs\_scl\_lcnt**

High Speed I2C Clock SCL Low Count.

Definition at line 809 of file qm\_soc\_regs.h.

**5.21.2.30 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_hs\_spklen**

HS spike suppression limit.

Definition at line 841 of file qm\_soc\_regs.h.

**5.21.2.31 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_intr\_mask**

Interrupt Mask.

Definition at line 811 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_master\_irq\_transfer(), and qm\_i2c\_set\_config().

**5.21.2.32 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_intr\_stat**

Interrupt Status.

Definition at line 810 of file qm\_soc\_regs.h.

**5.21.2.33 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_raw\_intr\_stat**

Raw Interrupt Status.

Definition at line 812 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_master\_read().

**5.21.2.34 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_rx\_tl**

Receive FIFO Threshold Level.

Definition at line 813 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_master\_irq\_transfer().

**5.21.2.35 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_rxflr**

Receive FIFO Level.

Definition at line 830 of file qm\_soc\_regs.h.

**5.21.2.36 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_sar**

Slave Address.

Definition at line 796 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_set\_config().

**5.21.2.37 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_sda\_hold**

SDA Hold.

Definition at line 831 of file qm\_soc\_regs.h.

**5.21.2.38 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_sda\_setup**

SDA Setup.

Definition at line 837 of file qm\_soc\_regs.h.

**5.21.2.39 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_ss\_scl\_hcnt**

Standard Speed Clock SCL High Count.

Definition at line 800 of file qm\_soc\_regs.h.

Referenced by qm\_i2c\_set\_config(), and qm\_i2c\_set\_speed().

**5.21.2.40 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_ss\_scl\_lcnt**

Standard Speed Clock SCL Low Count.

Definition at line 802 of file qm\_soc\_regs.h.

Referenced by `qm_i2c_set_config()`, and `qm_i2c_set_speed()`.

#### 5.21.2.41 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_status

Status.

Definition at line 828 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_get_status()`, `qm_i2c_master_read()`, and `qm_i2c_master_write()`.

#### 5.21.2.42 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_tar

Master Target Address.

Definition at line 795 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_master_irq_transfer()`, `qm_i2c_master_read()`, and `qm_i2c_master_write()`.

#### 5.21.2.43 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_tx\_abrt\_source

Transmit Abort Source.

Definition at line 832 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_get_status()`, `qm_i2c_master_read()`, and `qm_i2c_master_write()`.

#### 5.21.2.44 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_tx\_tl

Transmit FIFO Threshold Level.

Definition at line 814 of file `qm_soc_regs.h`.

Referenced by `qm_i2c_master_irq_transfer()`.

#### 5.21.2.45 QM\_RW uint32\_t qm\_i2c\_reg\_t::ic\_txflr

Transmit FIFO Level.

Definition at line 829 of file `qm_soc_regs.h`.

## 5.22 qm\_i2c\_transfer\_t Struct Reference

I2C transfer type.

```
#include <qm_i2c.h>
```

### Data Fields

- `uint8_t * tx`  
*Write data.*
- `uint32_t tx_len`  
*Write data length.*
- `uint8_t * rx`  
*Read data.*
- `uint32_t rx_len`  
*Read buffer length.*
- `bool stop`  
*Generate master STOP.*
- `void(* callback)(void *data, int rc, qm_i2c_status_t status, uint32_t len)`  
*Callback.*
- `void * callback_data`  
*Callback identifier.*

### 5.22.1 Detailed Description

I2C transfer type.

Master mode:

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

Definition at line 105 of file qm\_i2c.h.

### 5.22.2 Field Documentation

#### 5.22.2.1 void(\* qm\_i2c\_transfer\_t::callback)(void \*data, int rc, qm\_i2c\_status\_t status, uint32\_t len)

Callback.

Definition at line 111 of file qm\_i2c.h.

#### 5.22.2.2 void\* qm\_i2c\_transfer\_t::callback\_data

Callback identifier.

Definition at line 113 of file qm\_i2c.h.

#### 5.22.2.3 uint8\_t\* qm\_i2c\_transfer\_t::rx

Read data.

Definition at line 108 of file qm\_i2c.h.

Referenced by qm\_i2c\_master\_dma\_transfer().

#### 5.22.2.4 uint32\_t qm\_i2c\_transfer\_t::rx\_len

Read buffer length.

Definition at line 109 of file qm\_i2c.h.

Referenced by qm\_i2c\_master\_dma\_transfer(), and qm\_i2c\_master\_irq\_transfer().

#### 5.22.2.5 bool qm\_i2c\_transfer\_t::stop

Generate master STOP.

Definition at line 110 of file qm\_i2c.h.

Referenced by qm\_i2c\_master\_dma\_transfer().

#### 5.22.2.6 uint8\_t\* qm\_i2c\_transfer\_t::tx

Write data.

Definition at line 106 of file qm\_i2c.h.

Referenced by qm\_i2c\_master\_dma\_transfer().

#### 5.22.2.7 uint32\_t qm\_i2c\_transfer\_t::tx\_len

Write data length.

Definition at line 107 of file qm\_i2c.h.

Referenced by `qm_i2c_master_dma_transfer()`.

### 5.23 `qm_lapic_reg_t` Struct Reference

APIC register block type.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- `QM_RW apic_reg_pad_t id`  
*LAPIC ID.*
- `QM_RW apic_reg_pad_t version`  
*LAPIC version.*
- `QM_RW apic_reg_pad_t tpr`  
*Task priority.*
- `QM_RW apic_reg_pad_t apr`  
*Arbitration priority.*
- `QM_RW apic_reg_pad_t ppr`  
*Processor priority.*
- `QM_RW apic_reg_pad_t eoi`  
*End of interrupt.*
- `QM_RW apic_reg_pad_t rrd`  
*Remote read.*
- `QM_RW apic_reg_pad_t ldr`  
*Logical destination.*
- `QM_RW apic_reg_pad_t dfr`  
*Destination format.*
- `QM_RW apic_reg_pad_t svr`  
*Spurious vector.*
- `QM_RW apic_reg_pad_t isr [8]`  
*In-service.*
- `QM_RW apic_reg_pad_t tmr [8]`  
*Trigger mode.*
- `QM_RW apic_reg_pad_t irr [8]`  
*Interrupt request.*
- `QM_RW apic_reg_pad_t esr`  
*Error status.*
- `QM_RW apic_reg_pad_t lvtcmci`  
*Corrected Machine Check vector.*
- `QM_RW apic_reg_pad_t icr [2]`  
*Interrupt command.*
- `QM_RW apic_reg_pad_t lvttimer`  
*Timer vector.*
- `QM_RW apic_reg_pad_t lvtts`  
*Thermal sensor vector.*
- `QM_RW apic_reg_pad_t lvtpmcr`  
*Perfmon counter vector.*
- `QM_RW apic_reg_pad_t lvtlint0`  
*Local interrupt 0 vector.*
- `QM_RW apic_reg_pad_t lvtlint1`

*Local interrupt 1 vector.*

- QM\_RW apic\_reg\_pad\_t [lvterr](#)

*Error vector.*

- QM\_RW apic\_reg\_pad\_t [timer\\_icr](#)

*Timer initial count.*

- QM\_RW apic\_reg\_pad\_t [timer\\_ccr](#)

*Timer current count.*

- QM\_RW apic\_reg\_pad\_t [timer\\_dcr](#)

*Timer divide configuration.*

### 5.23.1 Detailed Description

APIC register block type.

Definition at line 215 of file qm\_soc\_regs.h.

## 5.24 qm\_mailbox\_t Struct Reference

Mailbox register structure.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [ch\\_ctrl](#)

*Channel Control Word.*

- QM\_RW uint32\_t [ch\\_data](#) [4]

*Channel Payload Data Word 0.*

- QM\_RW uint32\_t [ch\\_sts](#)

*Channel status.*

### 5.24.1 Detailed Description

Mailbox register structure.

Definition at line 606 of file qm\_soc\_regs.h.

## 5.25 qm\_mbox\_msg\_t Struct Reference

Definition of the mailbox message.

```
#include <qm_mailbox.h>
```

### Data Fields

- uint32\_t [ctrl](#)

*Mailbox control element.*

- uint32\_t [data](#) [QM\_MBOX\_PAYLOAD\_NUM]

*Mailbox data buffer.*



### 5.25.1 Detailed Description

Definition of the mailbox message.

Definition at line 60 of file qm\_mailbox.h.

### 5.25.2 Field Documentation

#### 5.25.2.1 uint32\_t qm\_mbox\_msg\_t::ctrl

Mailbox control element.

Definition at line 61 of file qm\_mailbox.h.

Referenced by qm\_mbox\_ch\_read(), and qm\_mbox\_ch\_write().

#### 5.25.2.2 uint32\_t qm\_mbox\_msg\_t::data[QM\_MBOX\_PAYLOAD\_NUM]

Mailbox data buffer.

Definition at line 62 of file qm\_mailbox.h.

Referenced by qm\_mbox\_ch\_read(), and qm\_mbox\_ch\_write().

## 5.26 qm\_mpr\_config\_t Struct Reference

SRAM Memory Protection Region configuration type.

```
#include <qm_mpr.h>
```

### Data Fields

- [uint8\\_t en\\_lock\\_mask](#)  
*Enable/lock bitmask.*
- [uint8\\_t agent\\_read\\_en\\_mask](#)  
*Per-agent read enable bitmask.*
- [uint8\\_t agent\\_write\\_en\\_mask](#)  
*Per-agent write enable bitmask.*
- [uint8\\_t up\\_bound](#)  
*1KB-aligned upper addr*
- [uint8\\_t low\\_bound](#)  
*1KB-aligned lower addr*

### 5.26.1 Detailed Description

SRAM Memory Protection Region configuration type.

Definition at line 43 of file qm\_mpr.h.

## 5.27 qm\_mpr\_reg\_t Struct Reference

Memory Protection Region register map.

```
#include <qm_soc_regs.h>
```

## Data Fields

- QM\_RW uint32\_t [mpr\\_cfg](#) [4]  
*MPR\_CFG.*
- QM\_RW uint32\_t [mpr\\_vdata](#)  
*MPR\_VDATA.*
- QM\_RW uint32\_t [mpr\\_vsts](#)  
*MPR\_VSTS.*

## 5.27.1 Detailed Description

Memory Protection Region register map.

Definition at line 1117 of file qm\_soc\_regs.h.

## 5.28 qm\_mvic\_reg\_t Struct Reference

MVIC register map.

```
#include <qm_soc_regs.h>
```

## Data Fields

- QM\_RW [mvic\\_reg\\_pad\\_t tpr](#)  
*Task priority.*
- QM\_RW [mvic\\_reg\\_pad\\_t ppr](#)  
*Processor priority.*
- QM\_RW [mvic\\_reg\\_pad\\_t eoi](#)  
*End of interrupt.*
- QM\_RW [mvic\\_reg\\_pad\\_t sivr](#)  
*Spurious vector.*
- QM\_RW [mvic\\_reg\\_pad\\_t isr](#)  
*In-service.*
- QM\_RW [mvic\\_reg\\_pad\\_t irr](#)  
*Interrupt request.*
- QM\_RW [mvic\\_reg\\_pad\\_t lvtimer](#)  
*Timer vector.*
- QM\_RW [mvic\\_reg\\_pad\\_t icr](#)  
*Timer initial count.*
- QM\_RW [mvic\\_reg\\_pad\\_t ccr](#)  
*Timer current count.*

## 5.28.1 Detailed Description

MVIC register map.

Definition at line 1230 of file qm\_soc\_regs.h.

## 5.28.2 Field Documentation

## 5.28.2.1 QM\_RW mvic\_reg\_pad\_t qm\_mvic\_reg\_t::ccr

Timer current count.

Definition at line 1245 of file qm\_soc\_regs.h.

### 5.28.2.2 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::eoi

End of interrupt.

Definition at line 1234 of file qm\_soc\_regs.h.

### 5.28.2.3 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::icr

Timer initial count.

Definition at line 1244 of file qm\_soc\_regs.h.

### 5.28.2.4 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::irr

Interrupt request.

Definition at line 1240 of file qm\_soc\_regs.h.

### 5.28.2.5 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::isr

In-service.

Definition at line 1238 of file qm\_soc\_regs.h.

### 5.28.2.6 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::lvtimer

Timer vector.

Definition at line 1242 of file qm\_soc\_regs.h.

### 5.28.2.7 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::ppr

Processor priority.

Definition at line 1233 of file qm\_soc\_regs.h.

### 5.28.2.8 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::sivr

Spurious vector.

Definition at line 1236 of file qm\_soc\_regs.h.

### 5.28.2.9 QM\_RW mVIC\_reg\_pad\_t qm\_mVIC\_reg\_t::tpr

Task priority.

Definition at line 1231 of file qm\_soc\_regs.h.

## 5.29 qm\_pic\_timer\_config\_t Struct Reference

PIC timer configuration type.

```
#include <qm_pic_timer.h>
```

### Data Fields

- [qm\\_pic\\_timer\\_mode\\_t mode](#)  
*Operation mode.*
- bool [int\\_en](#)  
*Interrupt enable.*
- void(\* [callback](#))(void \*data)  
*User callback.*
- void \* [callback\\_data](#)

*Callback user data.*

### 5.29.1 Detailed Description

PIC timer configuration type.

Definition at line 29 of file qm\_pic\_timer.h.

### 5.29.2 Field Documentation

#### 5.29.2.1 void(\* qm\_pic\_timer\_config\_t::callback)(void \*data)

User callback.

##### Parameters

|    |             |                    |
|----|-------------|--------------------|
| in | <i>data</i> | User defined data. |
|----|-------------|--------------------|

Definition at line 38 of file qm\_pic\_timer.h.

Referenced by qm\_pic\_timer\_set\_config().

#### 5.29.2.2 void\* qm\_pic\_timer\_config\_t::callback\_data

Callback user data.

Definition at line 39 of file qm\_pic\_timer.h.

Referenced by qm\_pic\_timer\_set\_config().

#### 5.29.2.3 bool qm\_pic\_timer\_config\_t::int\_en

Interrupt enable.

Definition at line 31 of file qm\_pic\_timer.h.

Referenced by qm\_pic\_timer\_set\_config().

#### 5.29.2.4 qm\_pic\_timer\_mode\_t qm\_pic\_timer\_config\_t::mode

Operation mode.

Definition at line 30 of file qm\_pic\_timer.h.

Referenced by qm\_pic\_timer\_set\_config().

## 5.30 qm\_pic\_timer\_reg\_t Struct Reference

PIC timer register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW [pic\\_timer\\_reg\\_pad\\_t lvttimer](#)  
*Local Vector Table Timer.*
- QM\_RW [pic\\_timer\\_reg\\_pad\\_t timer\\_icr](#)  
*Initial Count Register.*
- QM\_RW [pic\\_timer\\_reg\\_pad\\_t timer\\_ccr](#)  
*Current Count Register.*

### 5.30.1 Detailed Description

PIC timer register map.

Definition at line 1158 of file `qm_soc_regs.h`.

## 5.31 `qm_pwm_channel_t` Struct Reference

PWM / Timer channel register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t `loadcount`  
*Load Coun.t.*
- QM\_RW uint32\_t `currentvalue`  
*Current Value.*
- QM\_RW uint32\_t `controlreg`  
*Control.*
- QM\_RW uint32\_t `eoi`  
*End Of Interrupt.*
- QM\_RW uint32\_t `intstatus`  
*Interrupt Status.*

### 5.31.1 Detailed Description

PWM / Timer channel register map.

Definition at line 518 of file `qm_soc_regs.h`.

### 5.31.2 Field Documentation

#### 5.31.2.1 QM\_RW uint32\_t `qm_pwm_channel_t::controlreg`

Control.

Definition at line 521 of file `qm_soc_regs.h`.

#### 5.31.2.2 QM\_RW uint32\_t `qm_pwm_channel_t::currentvalue`

Current Value.

Definition at line 520 of file `qm_soc_regs.h`.

#### 5.31.2.3 QM\_RW uint32\_t `qm_pwm_channel_t::eoi`

End Of Interrupt.

Definition at line 522 of file `qm_soc_regs.h`.

#### 5.31.2.4 QM\_RW uint32\_t `qm_pwm_channel_t::intstatus`

Interrupt Status.

Definition at line 523 of file `qm_soc_regs.h`.

## 5.31.2.5 QM\_RW uint32\_t qm\_pwm\_channel\_t::loadcount

Load Coun.t.

Load Count.

Definition at line 519 of file qm\_soc\_regs.h.

## 5.32 qm\_pwm\_config\_t Struct Reference

PWM / Timer configuration type.

```
#include <qm_pwm.h>
```

## Data Fields

- uint32\_t [lo\\_count](#)  
*Number of cycles the PWM output is driven low.*
- uint32\_t [hi\\_count](#)  
*Number of cycles the PWM output is driven high.*
- bool [mask\\_interrupt](#)  
*Mask interrupt.*
- [qm\\_pwm\\_mode\\_t mode](#)  
*Pwm mode.*
- void(\* [callback](#))(void \*data, uint32\_t int\_status)  
*User callback.*
- void \* [callback\\_data](#)  
*Callback user data.*

## 5.32.1 Detailed Description

PWM / Timer configuration type.

Definition at line 34 of file qm\_pwm.h.

## 5.32.2 Field Documentation

## 5.32.2.1 void(\* qm\_pwm\_config\_t::callback)(void \*data, uint32\_t int\_status)

User callback.

## Parameters

|           |                   |                         |
|-----------|-------------------|-------------------------|
| <i>in</i> | <i>data</i>       | The callback user data. |
| <i>in</i> | <i>int_status</i> | The timer status.       |

Definition at line 49 of file qm\_pwm.h.

Referenced by [qm\\_pwm\\_set\\_config\(\)](#).

## 5.32.2.2 void\* qm\_pwm\_config\_t::callback\_data

Callback user data.

Definition at line 50 of file qm\_pwm.h.

Referenced by [qm\\_pwm\\_set\\_config\(\)](#).

### 5.32.2.3 uint32\_t qm\_pwm\_config\_t::hi\_count

Number of cycles the PWM output is driven high.

Not applicable in timer mode. Must be > 0.

Definition at line 39 of file qm\_pwm.h.

Referenced by qm\_pwm\_set\_config().

### 5.32.2.4 uint32\_t qm\_pwm\_config\_t::lo\_count

Number of cycles the PWM output is driven low.

In timer mode, this is the timer load count. Must be

0.

Definition at line 36 of file qm\_pwm.h.

Referenced by qm\_pwm\_set\_config().

### 5.32.2.5 bool qm\_pwm\_config\_t::mask\_interrupt

Mask interrupt.

Definition at line 41 of file qm\_pwm.h.

Referenced by qm\_pwm\_set\_config().

### 5.32.2.6 qm\_pwm\_mode\_t qm\_pwm\_config\_t::mode

Pwm mode.

Definition at line 42 of file qm\_pwm.h.

Referenced by qm\_pwm\_set\_config().

## 5.33 qm\_pwm\_reg\_t Struct Reference

PWM / Timer register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- [qm\\_pwm\\_channel\\_t timer](#) [QM\_PWM\_ID\_NUM]  
*2 Timers.*
- QM\_RW uint32\_t [timersintstatus](#)  
*Timers Interrupt Status.*
- QM\_RW uint32\_t [timerseoi](#)  
*Timers End Of Interrupt.*
- QM\_RW uint32\_t [timersrawintstatus](#)  
*Timers Raw Interrupt Status.*
- QM\_RW uint32\_t [timerscompversion](#)  
*Timers Component Version.*
- QM\_RW uint32\_t [timer\\_loadcount2](#) [4]  
*Timer Load Count 2.*

### 5.33.1 Detailed Description

PWM / Timer register map.

Definition at line 527 of file qm\_soc\_regs.h.

### 5.33.2 Field Documentation

#### 5.33.2.1 qm\_pwm\_channel\_t qm\_pwm\_reg\_t::timer

2 Timers.

4 Timers

Definition at line 528 of file qm\_soc\_regs.h.

## 5.34 qm\_rtc\_config\_t Struct Reference

RTC configuration type.

```
#include <qm_rtc.h>
```

### Data Fields

- uint32\_t [init\\_val](#)  
*Initial value in RTC clocks.*
- bool [alarm\\_en](#)  
*Alarm enable.*
- uint32\_t [alarm\\_val](#)  
*Alarm value in RTC clocks.*
- void(\* [callback](#))(void \*data)  
*User callback.*
- void \* [callback\\_data](#)  
*Callback user data.*

### 5.34.1 Detailed Description

RTC configuration type.

Definition at line 33 of file qm\_rtc.h.

### 5.34.2 Field Documentation

#### 5.34.2.1 bool qm\_rtc\_config\_t::alarm\_en

Alarm enable.

Definition at line 35 of file qm\_rtc.h.

Referenced by [qm\\_rtc\\_set\\_config\(\)](#).

#### 5.34.2.2 uint32\_t qm\_rtc\_config\_t::alarm\_val

Alarm value in RTC clocks.

Definition at line 36 of file qm\_rtc.h.

Referenced by [qm\\_rtc\\_set\\_config\(\)](#).



5.34.2.3 `void(* qm_rtc_config_t::callback)(void *data)`

User callback.

## Parameters

|    |      |                    |
|----|------|--------------------|
| in | data | User defined data. |
|----|------|--------------------|

Definition at line 43 of file qm\_rtc.h.

Referenced by qm\_rtc\_set\_config().

## 5.34.2.4 void\* qm\_rtc\_config\_t::callback\_data

Callback user data.

Definition at line 44 of file qm\_rtc.h.

Referenced by qm\_rtc\_set\_config().

## 5.34.2.5 uint32\_t qm\_rtc\_config\_t::init\_val

Initial value in RTC clocks.

Definition at line 34 of file qm\_rtc.h.

Referenced by qm\_rtc\_set\_config().

## 5.35 qm\_rtc\_reg\_t Struct Reference

RTC register map.

```
#include <qm_soc_regs.h>
```

## Data Fields

- QM\_RW uint32\_t [rtc\\_ccvr](#)  
*Current Counter Value Register.*
- QM\_RW uint32\_t [rtc\\_cmr](#)  
*Current Match Register.*
- QM\_RW uint32\_t [rtc\\_clr](#)  
*Counter Load Register.*
- QM\_RW uint32\_t [rtc\\_ccr](#)  
*Counter Control Register.*
- QM\_RW uint32\_t [rtc\\_stat](#)  
*Interrupt Status Register.*
- QM\_RW uint32\_t [rtc\\_rstat](#)  
*Interrupt Raw Status Register.*
- QM\_RW uint32\_t [rtc\\_eoi](#)  
*End of Interrupt Register.*
- QM\_RW uint32\_t [rtc\\_comp\\_version](#)  
*End of Interrupt Register.*

## 5.35.1 Detailed Description

RTC register map.

Definition at line 759 of file qm\_soc\_regs.h.

### 5.35.2 Field Documentation

#### 5.35.2.1 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_ccr

Counter Control Register.

Definition at line 763 of file qm\_soc\_regs.h.

#### 5.35.2.2 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_ccvr

Current Counter Value Register.

Definition at line 760 of file qm\_soc\_regs.h.

#### 5.35.2.3 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_clr

Counter Load Register.

Definition at line 762 of file qm\_soc\_regs.h.

#### 5.35.2.4 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_cmr

Current Match Register.

Definition at line 761 of file qm\_soc\_regs.h.

#### 5.35.2.5 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_comp\_version

End of Interrupt Register.

Definition at line 767 of file qm\_soc\_regs.h.

#### 5.35.2.6 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_eoi

End of Interrupt Register.

Definition at line 766 of file qm\_soc\_regs.h.

#### 5.35.2.7 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_rstat

Interrupt Raw Status Register.

Definition at line 765 of file qm\_soc\_regs.h.

#### 5.35.2.8 QM\_RW uint32\_t qm\_rtc\_reg\_t::rtc\_stat

Interrupt Status Register.

Definition at line 764 of file qm\_soc\_regs.h.

## 5.36 qm\_scss\_aon\_reg\_t Struct Reference

Always-on Controller register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [aonc\\_cnt](#)  
*Always-on counter register.*
- QM\_RW uint32\_t [aonc\\_cfg](#)  
*Always-on counter enable.*
- QM\_RW uint32\_t [aonpt\\_cnt](#)

- Always-on periodic timer.*

  - QM\_RW uint32\_t [aonpt\\_stat](#)  
*Always-on periodic timer status register.*
  - QM\_RW uint32\_t [aonpt\\_ctrl](#)  
*Always-on periodic timer control.*
  - QM\_RW uint32\_t [aonpt\\_cfg](#)  
*Always-on periodic timer configuration register.*

### 5.36.1 Detailed Description

Always-on Controller register map.

Definition at line 314 of file qm\_soc\_regs.h.

### 5.36.2 Field Documentation

#### 5.36.2.1 QM\_RW uint32\_t qm\_scss\_aon\_reg\_t::aonc\_cfg

Always-on counter enable.

Definition at line 316 of file qm\_soc\_regs.h.

#### 5.36.2.2 QM\_RW uint32\_t qm\_scss\_aon\_reg\_t::aonc\_cnt

Always-on counter register.

Definition at line 315 of file qm\_soc\_regs.h.

#### 5.36.2.3 QM\_RW uint32\_t qm\_scss\_aon\_reg\_t::aonpt\_cfg

Always-on periodic timer configuration register.

Definition at line 322 of file qm\_soc\_regs.h.

#### 5.36.2.4 QM\_RW uint32\_t qm\_scss\_aon\_reg\_t::aonpt\_cnt

Always-on periodic timer.

Definition at line 317 of file qm\_soc\_regs.h.

#### 5.36.2.5 QM\_RW uint32\_t qm\_scss\_aon\_reg\_t::aonpt\_ctrl

Always-on periodic timer control.

Definition at line 320 of file qm\_soc\_regs.h.

#### 5.36.2.6 QM\_RW uint32\_t qm\_scss\_aon\_reg\_t::aonpt\_stat

Always-on periodic timer status register.

Definition at line 319 of file qm\_soc\_regs.h.

## 5.37 qm\_scss\_ccu\_reg\_t Struct Reference

System Core register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [osc0\\_cfg0](#)

- Hybrid Oscillator Configuration 0.*
  - QM\_RW uint32\_t [osc0\\_stat1](#)
  - Hybrid Oscillator status 1.*
  - QM\_RW uint32\_t [osc0\\_cfg1](#)
  - Hybrid Oscillator configuration 1.*
  - QM\_RW uint32\_t [osc1\\_stat0](#)
  - RTC Oscillator status 0.*
  - QM\_RW uint32\_t [osc1\\_cfg0](#)
  - RTC Oscillator Configuration 0.*
  - QM\_RW uint32\_t [ccu\\_periph\\_clk\\_gate\\_ctl](#)
  - Peripheral Clock Gate Control.*
  - QM\_RW uint32\_t [ccu\\_periph\\_clk\\_div\\_ctl0](#)
  - Peripheral Clock Divider Control 0.*
  - QM\_RW uint32\_t [ccu\\_gpio\\_db\\_clk\\_ctl](#)
  - Peripheral Clock Divider Control 1.*
  - QM\_RW uint32\_t [ccu\\_ext\\_clock\\_ctl](#)
  - External Clock Control Register.*
  - QM\_RW uint32\_t [ccu\\_lp\\_clk\\_ctl](#)
  - System Low Power Clock Control.*
  - QM\_RW uint32\_t [wake\\_mask](#)
  - Wake Mask register.*
  - QM\_RW uint32\_t [ccu\\_mlayer\\_ahb\\_ctl](#)
  - AHB Control Register.*
  - QM\_RW uint32\_t [ccu\\_sys\\_clk\\_ctl](#)
  - System Clock Control Register.*
  - QM\_RW uint32\_t [osc\\_lock\\_0](#)
  - Clocks Lock Register.*
  - QM\_RW uint32\_t [soc\\_ctrl](#)
  - SoC Control Register.*
  - QM\_RW uint32\_t [soc\\_ctrl\\_lock](#)
  - SoC Control Register Lock.*
  - QM\_RW uint32\_t [usb\\_pll\\_cfg0](#)
  - USB Phase lock look configuration.*
  - QM\_RW uint32\_t [ccu\\_ss\\_periph\\_clk\\_gate\\_ctl](#)
  - Sensor Subsystem peripheral clock gate control.*

### 5.37.1 Detailed Description

System Core register map.

Definition at line 27 of file `qm_soc_regs.h`.

### 5.37.2 Field Documentation

#### 5.37.2.1 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_ext\_clock\_ctl

External Clock Control Register.

Definition at line 41 of file `qm_soc_regs.h`.

#### 5.37.2.2 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_gpio\_db\_clk\_ctl

Peripheral Clock Divider Control 1.

Definition at line 39 of file `qm_soc_regs.h`.

**5.37.2.3 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_lp\_clk\_ctl**

System Low Power Clock Control.

Definition at line 43 of file qm\_soc\_regs.h.

**5.37.2.4 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_mlayer\_ahb\_ctl**

AHB Control Register.

Definition at line 45 of file qm\_soc\_regs.h.

**5.37.2.5 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_periph\_clk\_div\_ctl0**

Peripheral Clock Divider Control 0.

Peripheral Clock Divider Control.

0

Definition at line 37 of file qm\_soc\_regs.h.

**5.37.2.6 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_periph\_clk\_gate\_ctl**

Peripheral Clock Gate Control.

Definition at line 35 of file qm\_soc\_regs.h.

**5.37.2.7 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_ss\_periph\_clk\_gate\_ctl**

Sensor Subsystem peripheral clock gate control.

Definition at line 48 of file qm\_soc\_regs.h.

**5.37.2.8 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::ccu\_sys\_clk\_ctl**

System Clock Control Register.

Definition at line 46 of file qm\_soc\_regs.h.

**5.37.2.9 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::osc0\_cfg0**

Hybrid Oscillator Configuration 0.

Definition at line 28 of file qm\_soc\_regs.h.

**5.37.2.10 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::osc0\_cfg1**

Hybrid Oscillator configuration 1.

Definition at line 30 of file qm\_soc\_regs.h.

**5.37.2.11 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::osc0\_stat1**

Hybrid Oscillator status 1.

Definition at line 29 of file qm\_soc\_regs.h.

**5.37.2.12 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::osc1\_cfg0**

RTC Oscillator Configuration 0.

Definition at line 32 of file qm\_soc\_regs.h.

**5.37.2.13 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::osc1\_stat0**

RTC Oscillator status 0.

Definition at line 31 of file qm\_soc\_regs.h.

#### 5.37.2.14 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::osc\_lock\_0

Clocks Lock Register.

Definition at line 47 of file qm\_soc\_regs.h.

#### 5.37.2.15 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::soc\_ctrl

SoC Control Register.

Definition at line 48 of file qm\_soc\_regs.h.

#### 5.37.2.16 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::soc\_ctrl\_lock

SoC Control Register Lock.

Definition at line 49 of file qm\_soc\_regs.h.

#### 5.37.2.17 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::usb\_pll\_cfg0

USB Phase lock look configuration.

Definition at line 38 of file qm\_soc\_regs.h.

#### 5.37.2.18 QM\_RW uint32\_t qm\_scss\_ccu\_reg\_t::wake\_mask

Wake Mask register.

Definition at line 44 of file qm\_soc\_regs.h.

## 5.38 qm\_scss\_cmp\_reg\_t Struct Reference

Comparator register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [cmp\\_en](#)  
*Comparator enable.*
- QM\_RW uint32\_t [cmp\\_ref\\_sel](#)  
*Comparator reference select.*
- QM\_RW uint32\_t [cmp\\_ref\\_pol](#)  
*Comparator reference polarity select register.*
- QM\_RW uint32\_t [cmp\\_pwr](#)  
*Comparator power enable register.*
- QM\_RW uint32\_t [cmp\\_stat\\_clr](#)  
*Comparator clear register.*

### 5.38.1 Detailed Description

Comparator register map.

Definition at line 174 of file qm\_soc\_regs.h.

## 5.38.2 Field Documentation

## 5.38.2.1 QM\_RW uint32\_t qm\_scss\_cmp\_reg\_t::cmp\_en

Comparator enable.

Definition at line 175 of file qm\_soc\_regs.h.

## 5.38.2.2 QM\_RW uint32\_t qm\_scss\_cmp\_reg\_t::cmp\_pwr

Comparator power enable register.

Definition at line 179 of file qm\_soc\_regs.h.

## 5.38.2.3 QM\_RW uint32\_t qm\_scss\_cmp\_reg\_t::cmp\_ref\_pol

Comparator reference polarity select register.

Definition at line 178 of file qm\_soc\_regs.h.

## 5.38.2.4 QM\_RW uint32\_t qm\_scss\_cmp\_reg\_t::cmp\_ref\_sel

Comparator reference select.

Definition at line 176 of file qm\_soc\_regs.h.

## 5.38.2.5 QM\_RW uint32\_t qm\_scss\_cmp\_reg\_t::cmp\_stat\_clr

Comparator clear register.

Definition at line 181 of file qm\_soc\_regs.h.

## 5.39 qm\_scss\_gp\_reg\_t Struct Reference

General Purpose register map.

```
#include <qm_soc_regs.h>
```

## Data Fields

- QM\_RW uint32\_t [gps0](#)  
*General Purpose Sticky Register 0.*
- QM\_RW uint32\_t [gps1](#)  
*General Purpose Sticky Register 1.*
- QM\_RW uint32\_t [gps2](#)  
*General Purpose Sticky Register 2.*
- QM\_RW uint32\_t [gps3](#)  
*General Purpose Sticky Register 3.*
- QM\_RW uint32\_t [gp0](#)  
*General Purpose Scratchpad Register 0.*
- QM\_RW uint32\_t [gp1](#)  
*General Purpose Scratchpad Register 1.*
- QM\_RW uint32\_t [gp2](#)  
*General Purpose Scratchpad Register 2.*
- QM\_RW uint32\_t [gp3](#)  
*General Purpose Scratchpad Register 3.*
- QM\_RW uint32\_t [wo\\_sp](#)  
*Write-One-to-Set Scratchpad Register.*



- QM\_RW uint32\_t [wo\\_st](#)  
*Write-One-to-Set Sticky Scratchpad Register.*
- QM\_RW uint32\_t [id](#)  
*Identification Register.*
- QM\_RW uint32\_t [rev](#)  
*Revision Register.*

### 5.39.1 Detailed Description

General Purpose register map.

Definition at line 138 of file `qm_soc_regs.h`.

### 5.39.2 Field Documentation

#### 5.39.2.1 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gp0

General Purpose Scratchpad Register 0.

Definition at line 144 of file `qm_soc_regs.h`.

#### 5.39.2.2 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gp1

General Purpose Scratchpad Register 1.

Definition at line 145 of file `qm_soc_regs.h`.

#### 5.39.2.3 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gp2

General Purpose Scratchpad Register 2.

Definition at line 146 of file `qm_soc_regs.h`.

#### 5.39.2.4 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gp3

General Purpose Scratchpad Register 3.

Definition at line 147 of file `qm_soc_regs.h`.

#### 5.39.2.5 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gps0

General Purpose Sticky Register 0.

Definition at line 139 of file `qm_soc_regs.h`.

#### 5.39.2.6 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gps1

General Purpose Sticky Register 1.

Definition at line 140 of file `qm_soc_regs.h`.

#### 5.39.2.7 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gps2

General Purpose Sticky Register 2.

Definition at line 141 of file `qm_soc_regs.h`.

#### 5.39.2.8 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::gps3

General Purpose Sticky Register 3.

Definition at line 142 of file `qm_soc_regs.h`.

#### 5.39.2.9 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::wo\_sp

Write-One-to-Set Scratchpad Register.

Definition at line 149 of file qm\_soc\_regs.h.

#### 5.39.2.10 QM\_RW uint32\_t qm\_scss\_gp\_reg\_t::wo\_st

Write-One-to-Set Sticky Scratchpad Register.

Definition at line 151 of file qm\_soc\_regs.h.

## 5.40 qm\_scss\_info\_reg\_t Struct Reference

Information register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [id](#)  
*Identification Register.*
- QM\_RW uint32\_t [rev](#)  
*Revision Register.*
- QM\_RW uint32\_t [fs](#)  
*Flash Size Register.*
- QM\_RW uint32\_t [rs](#)  
*RAM Size Register.*
- QM\_RW uint32\_t [cotps](#)  
*Code OTP Size Register.*
- QM\_RW uint32\_t [dotps](#)  
*Data OTP Size Register.*

### 5.40.1 Detailed Description

Information register map.

Definition at line 398 of file qm\_soc\_regs.h.

### 5.40.2 Field Documentation

#### 5.40.2.1 QM\_RW uint32\_t qm\_scss\_info\_reg\_t::cotps

Code OTP Size Register.

Definition at line 403 of file qm\_soc\_regs.h.

#### 5.40.2.2 QM\_RW uint32\_t qm\_scss\_info\_reg\_t::dotps

Data OTP Size Register.

Definition at line 404 of file qm\_soc\_regs.h.

#### 5.40.2.3 QM\_RW uint32\_t qm\_scss\_info\_reg\_t::fs

Flash Size Register.

Definition at line 401 of file qm\_soc\_regs.h.

#### 5.40.2.4 QM\_RW uint32\_t qm\_scss\_info\_reg\_t::id

Identification Register.

Definition at line 399 of file qm\_soc\_regs.h.

#### 5.40.2.5 QM\_RW uint32\_t qm\_scss\_info\_reg\_t::rev

Revision Register.

Definition at line 400 of file qm\_soc\_regs.h.

#### 5.40.2.6 QM\_RW uint32\_t qm\_scss\_info\_reg\_t::rs

RAM Size Register.

Definition at line 402 of file qm\_soc\_regs.h.

### 5.41 qm\_scss\_int\_reg\_t Struct Reference

Interrupt register map.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- QM\_RW uint32\_t [int\\_i2c\\_mst\\_0\\_mask](#)  
*Interrupt Routing Mask 0.*
- QM\_RW uint32\_t [int\\_spi\\_mst\\_0\\_mask](#)  
*Interrupt Routing Mask 2.*
- QM\_RW uint32\_t [int\\_spi\\_slv\\_0\\_mask](#)  
*Interrupt Routing Mask 4.*
- QM\_RW uint32\_t [int\\_uart\\_0\\_mask](#)  
*Interrupt Routing Mask 5.*
- QM\_RW uint32\_t [int\\_uart\\_1\\_mask](#)  
*Interrupt Routing Mask 6.*
- QM\_RW uint32\_t [int\\_gpio\\_mask](#)  
*Interrupt Routing Mask 8.*
- QM\_RW uint32\_t [int\\_timer\\_mask](#)  
*Interrupt Routing Mask 9.*
- QM\_RW uint32\_t [int\\_rtc\\_mask](#)  
*Interrupt Routing Mask 11.*
- QM\_RW uint32\_t [int\\_watchdog\\_mask](#)  
*Interrupt Routing Mask 12.*
- QM\_RW uint32\_t [int\\_dma\\_channel\\_0\\_mask](#)  
*Interrupt Routing Mask 13.*
- QM\_RW uint32\_t [int\\_dma\\_channel\\_1\\_mask](#)  
*Interrupt Routing Mask 14.*
- QM\_RW uint32\_t [int\\_comparators\\_host\\_halt\\_mask](#)  
*Interrupt Routing Mask 23.*
- QM\_RW uint32\_t [int\\_comparators\\_host\\_mask](#)  
*Interrupt Routing Mask 25.*
- QM\_RW uint32\_t [int\\_host\\_bus\\_err\\_mask](#)  
*Interrupt Routing Mask 26.*
- QM\_RW uint32\_t [int\\_dma\\_error\\_mask](#)

- Interrupt Routing Mask 27.*
  - QM\_RW uint32\_t [int\\_sram\\_controller\\_mask](#)
- Interrupt Routing Mask 28.*
  - QM\_RW uint32\_t [int\\_flash\\_controller\\_0\\_mask](#)
- Interrupt Routing Mask 29.*
  - QM\_RW uint32\_t [int\\_aon\\_timer\\_mask](#)
- Interrupt Routing Mask 31.*
  - QM\_RW uint32\_t [int\\_adc\\_pwr\\_mask](#)
- Interrupt Routing Mask 32.*
  - QM\_RW uint32\_t [int\\_adc\\_calib\\_mask](#)
- Interrupt Routing Mask 33.*
  - QM\_RW uint32\_t [lock\\_int\\_mask\\_reg](#)
- Interrupt Mask Lock Register.*
  - QM\_RW uint32\_t [int\\_flash\\_controller\\_1\\_mask](#)
- Interrupt Routing Mask 30.*

#### 5.41.1 Detailed Description

Interrupt register map.

Definition at line 203 of file qm\_soc\_regs.h.

#### 5.41.2 Field Documentation

##### 5.41.2.1 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_adc\_calib\_mask

Interrupt Routing Mask 33.

Definition at line 236 of file qm\_soc\_regs.h.

##### 5.41.2.2 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_adc\_pwr\_mask

Interrupt Routing Mask 32.

Definition at line 235 of file qm\_soc\_regs.h.

##### 5.41.2.3 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_aon\_timer\_mask

Interrupt Routing Mask 31.

Definition at line 234 of file qm\_soc\_regs.h.

##### 5.41.2.4 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_comparators\_host\_halt\_mask

Interrupt Routing Mask 23.

Definition at line 223 of file qm\_soc\_regs.h.

##### 5.41.2.5 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_comparators\_host\_mask

Interrupt Routing Mask 25.

Definition at line 226 of file qm\_soc\_regs.h.

##### 5.41.2.6 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_dma\_channel\_0\_mask

Interrupt Routing Mask 13.

Definition at line 218 of file qm\_soc\_regs.h.

**5.41.2.7 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_dma\_channel\_1\_mask**

Interrupt Routing Mask 14.

Definition at line 220 of file qm\_soc\_regs.h.

**5.41.2.8 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_dma\_error\_mask**

Interrupt Routing Mask 27.

Definition at line 228 of file qm\_soc\_regs.h.

**5.41.2.9 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_flash\_controller\_0\_mask**

Interrupt Routing Mask 29.

Definition at line 232 of file qm\_soc\_regs.h.

**5.41.2.10 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_flash\_controller\_1\_mask**

Interrupt Routing Mask 30.

Definition at line 360 of file qm\_soc\_regs.h.

**5.41.2.11 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_gpio\_mask**

Interrupt Routing Mask 8.

Definition at line 212 of file qm\_soc\_regs.h.

**5.41.2.12 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_host\_bus\_err\_mask**

Interrupt Routing Mask 26.

Definition at line 227 of file qm\_soc\_regs.h.

**5.41.2.13 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_i2c\_mst\_0\_mask**

Interrupt Routing Mask 0.

Definition at line 204 of file qm\_soc\_regs.h.

**5.41.2.14 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_rtc\_mask**

Interrupt Routing Mask 11.

Definition at line 215 of file qm\_soc\_regs.h.

**5.41.2.15 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_spi\_mst\_0\_mask**

Interrupt Routing Mask 2.

Definition at line 206 of file qm\_soc\_regs.h.

**5.41.2.16 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_spi\_slv\_0\_mask**

Interrupt Routing Mask 4.

Definition at line 208 of file qm\_soc\_regs.h.

**5.41.2.17 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_sram\_controller\_mask**

Interrupt Routing Mask 28.

Definition at line 230 of file qm\_soc\_regs.h.

#### 5.41.2.18 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_timer\_mask

Interrupt Routing Mask 9.

Definition at line 213 of file qm\_soc\_regs.h.

#### 5.41.2.19 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_uart\_0\_mask

Interrupt Routing Mask 5.

Definition at line 209 of file qm\_soc\_regs.h.

#### 5.41.2.20 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_uart\_1\_mask

Interrupt Routing Mask 6.

Definition at line 210 of file qm\_soc\_regs.h.

#### 5.41.2.21 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::int\_watchdog\_mask

Interrupt Routing Mask 12.

Definition at line 216 of file qm\_soc\_regs.h.

#### 5.41.2.22 QM\_RW uint32\_t qm\_scss\_int\_reg\_t::lock\_int\_mask\_reg

Interrupt Mask Lock Register.

Definition at line 238 of file qm\_soc\_regs.h.

## 5.42 qm\_scss\_mailbox\_reg\_t Struct Reference

Mailbox register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- [qm\\_mailbox\\_t mbox](#) [8]  
*8 Mailboxes*
- QM\_RW uint32\_t [mbox\\_chall\\_sts](#)  
*All channel status.*

#### 5.42.1 Detailed Description

Mailbox register map.

Definition at line 613 of file qm\_soc\_regs.h.

## 5.43 qm\_scss\_mem\_reg\_t Struct Reference

Memory Control register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [mem\\_ctrl](#)  
*Memory control.*

#### 5.43.1 Detailed Description

Memory Control register map.

Definition at line 160 of file qm\_soc\_regs.h.

### 5.44 qm\_scss\_peripheral\_reg\_t Struct Reference

Peripheral Registers register map.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- QM\_RW uint32\_t [periph\\_cfg0](#)  
*Peripheral Configuration.*
- QM\_RW uint32\_t [cfg\\_lock](#)  
*Configuration Lock.*
- QM\_RW uint32\_t [usb\\_phy\\_cfg0](#)  
*USB Configuration.*

#### 5.44.1 Detailed Description

Peripheral Registers register map.

Definition at line 342 of file qm\_soc\_regs.h.

#### 5.44.2 Field Documentation

##### 5.44.2.1 QM\_RW uint32\_t qm\_scss\_peripheral\_reg\_t::cfg\_lock

Configuration Lock.

Definition at line 345 of file qm\_soc\_regs.h.

##### 5.44.2.2 QM\_RW uint32\_t qm\_scss\_peripheral\_reg\_t::periph\_cfg0

Peripheral Configuration.

Definition at line 343 of file qm\_soc\_regs.h.

### 5.45 qm\_scss\_pmu\_reg\_t Struct Reference

Power Management register map.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- QM\_RW uint32\_t [aon\\_vr](#)  
*AON Voltage Regulator.*
- QM\_RW uint32\_t [pm\\_wait](#)  
*Power Management Wait.*
- QM\_RW uint32\_t [p\\_sts](#)  
*Processor Status.*

- QM\_RW uint32\_t [rstc](#)  
*Reset Control.*
- QM\_RW uint32\_t [rstcs](#)  
*Reset Status.*
- QM\_RW uint32\_t [pm\\_lock](#)  
*Power Management Lock.*
- QM\_RW uint32\_t [p\\_lv12](#)  
*Processor level 2.*
- QM\_RW uint32\_t [pm1c](#)  
*Power management 1 control.*
- QM\_RW uint32\_t [plat3p3\\_vr](#)  
*Platform 3p3 voltage regulator.*
- QM\_RW uint32\_t [plat1p8\\_vr](#)  
*Platform 1p8 voltage regulator.*
- QM\_RW uint32\_t [host\\_vr](#)  
*Host Voltage Regulator.*
- QM\_RW uint32\_t [slp\\_cfg](#)  
*Sleeping Configuration.*
- QM\_RW uint32\_t [pmnetcs](#)  
*Power Management Network (PMNet) Control and Status.*
- QM\_RW uint32\_t [vr\\_lock](#)  
*Voltage regulator lock.*

#### 5.45.1 Detailed Description

Power Management register map.

Definition at line 271 of file qm\_soc\_regs.h.

#### 5.45.2 Field Documentation

##### 5.45.2.1 QM\_RW uint32\_t qm\_scss\_pmu\_reg\_t::aon\_vr

AON Voltage Regulator.

Definition at line 272 of file qm\_soc\_regs.h.

##### 5.45.2.2 QM\_RW uint32\_t qm\_scss\_pmu\_reg\_t::p\_sts

Processor Status.

Definition at line 276 of file qm\_soc\_regs.h.

##### 5.45.2.3 QM\_RW uint32\_t qm\_scss\_pmu\_reg\_t::pm\_lock

Power Management Lock.

Definition at line 281 of file qm\_soc\_regs.h.

##### 5.45.2.4 QM\_RW uint32\_t qm\_scss\_pmu\_reg\_t::pm\_wait

Power Management Wait.

Definition at line 274 of file qm\_soc\_regs.h.



#### 5.45.2.5 QM\_RW uint32\_t qm\_scss\_pmu\_reg\_t::rstc

Reset Control.

Definition at line 278 of file qm\_soc\_regs.h.

#### 5.45.2.6 QM\_RW uint32\_t qm\_scss\_pmu\_reg\_t::rstc

Reset Status.

Definition at line 279 of file qm\_soc\_regs.h.

### 5.46 qm\_scss\_pmux\_reg\_t Struct Reference

Pin MUX register map.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- QM\_RW uint32\_t [pmux\\_pullup](#) [1]  
*Pin Mux Pullup.*
- QM\_RW uint32\_t [pmux\\_slew](#) [1]  
*Pin Mux Slew Rate.*
- QM\_RW uint32\_t [pmux\\_in\\_en](#) [1]  
*Pin Mux Input Enable.*
- QM\_RW uint32\_t [pmux\\_sel](#) [2]  
*Pin Mux Select.*
- QM\_RW uint32\_t [pmux\\_pullup\\_lock](#)  
*Pin Mux Pullup Lock.*
- QM\_RW uint32\_t [pmux\\_slew\\_lock](#)  
*Pin Mux Slew Rate Lock.*
- QM\_RW uint32\_t [pmux\\_sel\\_0\\_lock](#)  
*Pin Mux Select Lock 0.*
- QM\_RW uint32\_t [pmux\\_in\\_en\\_lock](#)  
*Pin Mux Slew Rate Lock.*
- QM\_RW uint32\_t [pmux\\_sel\\_lock](#) [3]  
*Pin Mux Select Lock.*

#### 5.46.1 Detailed Description

Pin MUX register map.

Definition at line 365 of file qm\_soc\_regs.h.

#### 5.46.2 Field Documentation

##### 5.46.2.1 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_in\_en

Pin Mux Input Enable.

Definition at line 370 of file qm\_soc\_regs.h.

#### 5.46.2.2 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_in\_en\_lock

Pin Mux Slew Rate Lock.

Definition at line 378 of file qm\_soc\_regs.h.

#### 5.46.2.3 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_pullup

Pin Mux Pullup.

Definition at line 366 of file qm\_soc\_regs.h.

#### 5.46.2.4 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_pullup\_lock

Pin Mux Pullup Lock.

Definition at line 374 of file qm\_soc\_regs.h.

#### 5.46.2.5 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_sel

Pin Mux Select.

Definition at line 372 of file qm\_soc\_regs.h.

#### 5.46.2.6 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_sel\_0\_lock

Pin Mux Select Lock 0.

Definition at line 376 of file qm\_soc\_regs.h.

#### 5.46.2.7 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_slew

Pin Mux Slew Rate.

Definition at line 368 of file qm\_soc\_regs.h.

#### 5.46.2.8 QM\_RW uint32\_t qm\_scss\_pmux\_reg\_t::pmux\_slew\_lock

Pin Mux Slew Rate Lock.

Definition at line 375 of file qm\_soc\_regs.h.

## 5.47 qm\_scss\_ss\_reg\_t Struct Reference

Sensor Subsystem register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [ss\\_cfg](#)  
*Sensor Subsystem Configuration.*
- QM\_RW uint32\_t [ss\\_sts](#)  
*Sensor Subsystem status.*

### 5.47.1 Detailed Description

Sensor Subsystem register map.

Definition at line 461 of file qm\_soc\_regs.h.

## 5.48 qm\_spi\_async\_transfer\_t Struct Reference

SPI IRQ transfer type.

```
#include <qm_spi.h>
```

### Data Fields

- `uint8_t * tx`  
*Write data.*
- `uint8_t * rx`  
*Read data.*
- `uint16_t tx_len`  
*Write data Length.*
- `uint16_t rx_len`  
*Read buffer length.*
- `void(* callback)(void *data, int error, qm_spi_status_t status, uint16_t len)`  
*Transfer callback.*
- `void * callback_data`  
*Callback user data.*

### 5.48.1 Detailed Description

SPI IRQ transfer type.

Definition at line 117 of file `qm_spi.h`.

### 5.48.2 Field Documentation

#### 5.48.2.1 `void(* qm_spi_async_transfer_t::callback)(void *data, int error, qm_spi_status_t status, uint16_t len)`

Transfer callback.

Called after all data is transmitted/received or if the driver detects an error during the SPI transfer.

#### Parameters

|                 |                     |   |
|-----------------|---------------------|---|
| <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> | SPI driver status.  |
| <code>in</code> | <code>len</code>    | Length of the SPI transfer if successful, 0 otherwise.              |

Definition at line 136 of file `qm_spi.h`.

Referenced by `qm_spi_irq_transfer_terminate()`.

#### 5.48.2.2 `void* qm_spi_async_transfer_t::callback_data`

Callback user data.

Definition at line 138 of file `qm_spi.h`.

Referenced by `qm_spi_irq_transfer_terminate()`.

#### 5.48.2.3 `uint8_t* qm_spi_async_transfer_t::rx`

Read data.

Definition at line 119 of file `qm_spi.h`.

Referenced by `qm_spi_dma_transfer()`.

#### 5.48.2.4 uint16\_t qm\_spi\_async\_transfer\_t::rx\_len

Read buffer length.

Definition at line 121 of file qm\_spi.h.

Referenced by qm\_spi\_dma\_transfer(), and qm\_spi\_irq\_transfer().

#### 5.48.2.5 uint8\_t\* qm\_spi\_async\_transfer\_t::tx

Write data.

Definition at line 118 of file qm\_spi.h.

Referenced by qm\_spi\_dma\_transfer().

#### 5.48.2.6 uint16\_t qm\_spi\_async\_transfer\_t::tx\_len

Write data Length.

Definition at line 120 of file qm\_spi.h.

Referenced by qm\_spi\_dma\_transfer(), and qm\_spi\_irq\_transfer().

## 5.49 qm\_spi\_config\_t Struct Reference

SPI configuration type.

```
#include <qm_spi.h>
```

### Data Fields

- [qm\\_spi\\_frame\\_size\\_t frame\\_size](#)  
*Frame Size.*
- [qm\\_spi\\_tmode\\_t transfer\\_mode](#)  
*Transfer mode (enum).*
- [qm\\_spi\\_bmode\\_t bus\\_mode](#)  
*Bus mode (enum).*
- [uint16\\_t clk\\_divider](#)  
*SCK = SPI\_clock/clk\_divider.*

### 5.49.1 Detailed Description

SPI configuration type.

Definition at line 101 of file qm\_spi.h.

### 5.49.2 Field Documentation

#### 5.49.2.1 qm\_spi\_bmode\_t qm\_spi\_config\_t::bus\_mode

Bus mode (enum).

Definition at line 104 of file qm\_spi.h.

Referenced by qm\_spi\_set\_config().

#### 5.49.2.2 uint16\_t qm\_spi\_config\_t::clk\_divider

SCK = SPI\_clock/clk\_divider.

A value of 0 will disable SCK.

Definition at line 111 of file qm\_spi.h.

Referenced by qm\_spi\_set\_config().

#### 5.49.2.3 qm\_spi\_frame\_size\_t qm\_spi\_config\_t::frame\_size

Frame Size.

Definition at line 102 of file qm\_spi.h.

Referenced by qm\_spi\_set\_config().

#### 5.49.2.4 qm\_spi\_tmode\_t qm\_spi\_config\_t::transfer\_mode

Transfer mode (enum).

Definition at line 103 of file qm\_spi.h.

Referenced by qm\_spi\_set\_config().

## 5.50 qm\_spi\_reg\_t Struct Reference

SPI register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [ctrlr0](#)  
*Control Register 0.*
- QM\_RW uint32\_t [ctrlr1](#)  
*Control Register 1.*
- QM\_RW uint32\_t [ssienr](#)  
*SSI Enable Register.*
- QM\_RW uint32\_t [mwcr](#)  
*Microwire Control Register.*
- QM\_RW uint32\_t [ser](#)  
*Slave Enable Register.*
- QM\_RW uint32\_t [baudr](#)  
*Baud Rate Select.*
- QM\_RW uint32\_t [txflr](#)  
*Transmit FIFO Threshold Level.*
- QM\_RW uint32\_t [rxflr](#)  
*Receive FIFO Threshold Level.*
- QM\_RW uint32\_t [txflr](#)  
*Transmit FIFO Level Register.*
- QM\_RW uint32\_t [rxflr](#)  
*Receive FIFO Level Register.*
- QM\_RW uint32\_t [sr](#)  
*Status Register.*
- QM\_RW uint32\_t [imr](#)  
*Interrupt Mask Register.*

- QM\_RW uint32\_t [isr](#)  
*Interrupt Status Register.*
- QM\_RW uint32\_t [risr](#)  
*Raw Interrupt Status Register.*
- QM\_RW uint32\_t [txoicr](#)  
*Tx FIFO Overflow Interrupt Clear Register.*
- QM\_RW uint32\_t [rxoicr](#)  
*Rx FIFO Overflow Interrupt Clear Register.*
- QM\_RW uint32\_t [rxuicr](#)  
*Rx FIFO Underflow Interrupt Clear Register.*
- QM\_RW uint32\_t [msticr](#)  
*Multi-Master Interrupt Clear Register.*
- QM\_RW uint32\_t [icr](#)  
*Interrupt Clear Register.*
- QM\_RW uint32\_t [dmacr](#)  
*DMA Control Register.*
- QM\_RW uint32\_t [dmatdlr](#)  
*DMA Transmit Data Level.*
- QM\_RW uint32\_t [dmardlr](#)  
*DMA Receive Data Level.*
- QM\_RW uint32\_t [idr](#)  
*Identification Register.*
- QM\_RW uint32\_t [ssi\\_comp\\_version](#)  
*coreKit Version ID register.*
- QM\_RW uint32\_t [dr](#) [36]  
*Data Register.*
- QM\_RW uint32\_t [rx\\_sample\\_dly](#)  
*RX Sample Delay Register.*

### 5.50.1 Detailed Description

SPI register map.

Definition at line 653 of file qm\_soc\_regs.h.

### 5.50.2 Field Documentation

#### 5.50.2.1 QM\_RW uint32\_t qm\_spi\_reg\_t::baudr

Baud Rate Select.

Definition at line 659 of file qm\_soc\_regs.h.

Referenced by [qm\\_spi\\_set\\_config\(\)](#).

#### 5.50.2.2 QM\_RW uint32\_t qm\_spi\_reg\_t::ctrlr0

Control Register 0.

Definition at line 654 of file qm\_soc\_regs.h.

Referenced by [qm\\_spi\\_set\\_config\(\)](#).

### 5.50.2.3 QM\_RW uint32\_t qm\_spi\_reg\_t::ctrlr1

Control Register 1.

Definition at line 655 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer(), qm\_spi\_irq\_transfer(), and qm\_spi\_transfer().

### 5.50.2.4 QM\_RW uint32\_t qm\_spi\_reg\_t::dmacr

DMA Control Register.

Definition at line 676 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer().

### 5.50.2.5 QM\_RW uint32\_t qm\_spi\_reg\_t::dmardlr

DMA Receive Data Level.

Definition at line 678 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer().

### 5.50.2.6 QM\_RW uint32\_t qm\_spi\_reg\_t::dmatdlr

DMA Transmit Data Level.

Definition at line 677 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer().

### 5.50.2.7 QM\_RW uint32\_t qm\_spi\_reg\_t::dr

Data Register.

Definition at line 681 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer().

### 5.50.2.8 QM\_RW uint32\_t qm\_spi\_reg\_t::icr

Interrupt Clear Register.

Definition at line 675 of file qm\_soc\_regs.h.

### 5.50.2.9 QM\_RW uint32\_t qm\_spi\_reg\_t::idr

Identification Register.

Definition at line 679 of file qm\_soc\_regs.h.

### 5.50.2.10 QM\_RW uint32\_t qm\_spi\_reg\_t::imr

Interrupt Mask Register.

Definition at line 665 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer(), qm\_spi\_irq\_transfer(), qm\_spi\_irq\_transfer\_terminate(), and qm\_spi\_transfer().

### 5.50.2.11 QM\_RW uint32\_t qm\_spi\_reg\_t::isr

Interrupt Status Register.

Definition at line 666 of file qm\_soc\_regs.h.

**5.50.2.12 QM\_RW uint32\_t qm\_spi\_reg\_t::msticr**

Multi-Master Interrupt Clear Register.

Definition at line 674 of file qm\_soc\_regs.h.

**5.50.2.13 QM\_RW uint32\_t qm\_spi\_reg\_t::mwcr**

Microwire Control Register.

Definition at line 657 of file qm\_soc\_regs.h.

**5.50.2.14 QM\_RW uint32\_t qm\_spi\_reg\_t::risr**

Raw Interrupt Status Register.

Definition at line 667 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_get\_status(), and qm\_spi\_transfer().

**5.50.2.15 QM\_RW uint32\_t qm\_spi\_reg\_t::rx\_sample\_dly**

RX Sample Delay Register.

Definition at line 682 of file qm\_soc\_regs.h.

**5.50.2.16 QM\_RW uint32\_t qm\_spi\_reg\_t::rxflr**

Receive FIFO Level Register.

Definition at line 663 of file qm\_soc\_regs.h.

**5.50.2.17 QM\_RW uint32\_t qm\_spi\_reg\_t::rxftlr**

Receive FIFO Threshold Level.

Definition at line 661 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_irq\_transfer().

**5.50.2.18 QM\_RW uint32\_t qm\_spi\_reg\_t::rxoicr**

Rx FIFO Overflow Interrupt Clear Register.

Definition at line 671 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_transfer().

**5.50.2.19 QM\_RW uint32\_t qm\_spi\_reg\_t::rxuicr**

Rx FIFO Underflow Interrupt Clear Register.

Definition at line 673 of file qm\_soc\_regs.h.

**5.50.2.20 QM\_RW uint32\_t qm\_spi\_reg\_t::ser**

Slave Enable Register.

Definition at line 658 of file qm\_soc\_regs.h.

**5.50.2.21 QM\_RW uint32\_t qm\_spi\_reg\_t::sr**

Status Register.

Definition at line 664 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_get\_status(), and qm\_spi\_transfer().



#### 5.50.2.22 QM\_RW uint32\_t qm\_spi\_reg\_t::ssi\_comp\_version

coreKit Version ID register.

coreKit Version ID register

Definition at line 680 of file qm\_soc\_regs.h.

#### 5.50.2.23 QM\_RW uint32\_t qm\_spi\_reg\_t::ssienr

SSI Enable Register.

Definition at line 656 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_dma\_transfer(), qm\_spi\_irq\_transfer(), qm\_spi\_irq\_transfer\_terminate(), and qm\_spi\_transfer().

#### 5.50.2.24 QM\_RW uint32\_t qm\_spi\_reg\_t::txflr

Transmit FIFO Level Register.

Definition at line 662 of file qm\_soc\_regs.h.

#### 5.50.2.25 QM\_RW uint32\_t qm\_spi\_reg\_t::txftlr

Transmit FIFO Threshold Level.

Definition at line 660 of file qm\_soc\_regs.h.

Referenced by qm\_spi\_irq\_transfer().

#### 5.50.2.26 QM\_RW uint32\_t qm\_spi\_reg\_t::txoicr

Tx FIFO Overflow Interrupt Clear Register.

Definition at line 669 of file qm\_soc\_regs.h.

### 5.51 qm\_spi\_transfer\_t Struct Reference

SPI transfer type.

```
#include <qm_spi.h>
```

#### Data Fields

- [uint8\\_t \\* tx](#)  
*Write Data.*
- [uint8\\_t \\* rx](#)  
*Read Data.*
- [uint16\\_t tx\\_len](#)  
*Write Data Length.*
- [uint16\\_t rx\\_len](#)  
*Receive Data Length.*

#### 5.51.1 Detailed Description

SPI transfer type.

Definition at line 144 of file qm\_spi.h.

### 5.51.2 Field Documentation

#### 5.51.2.1 uint8\_t\* qm\_spi\_transfer\_t::rx

Read Data.

Definition at line 146 of file qm\_spi.h.

Referenced by qm\_spi\_transfer().

#### 5.51.2.2 uint16\_t qm\_spi\_transfer\_t::rx\_len

Receive Data Length.

Definition at line 148 of file qm\_spi.h.

Referenced by qm\_spi\_transfer().

#### 5.51.2.3 uint8\_t\* qm\_spi\_transfer\_t::tx

Write Data.

Definition at line 145 of file qm\_spi.h.

Referenced by qm\_spi\_transfer().

#### 5.51.2.4 uint16\_t qm\_spi\_transfer\_t::tx\_len

Write Data Length.

Definition at line 147 of file qm\_spi.h.

Referenced by qm\_spi\_transfer().

## 5.52 qm\_ss\_adc\_config\_t Struct Reference

SS ADC configuration type.

```
#include <qm_ss_adc.h>
```

### Data Fields

- [uint8\\_t window](#)

*Sample interval in ADC clock cycles, defines the period to wait between the start of each sample and can be in the range [(resolution+2) - 255].*

- [qm\\_ss\\_adc\\_resolution\\_t resolution](#)

*12, 10, 8, 6-bit resolution.*

### 5.52.1 Detailed Description

SS ADC configuration type.

Definition at line 98 of file qm\_ss\_adc.h.

### 5.52.2 Field Documentation

#### 5.52.2.1 qm\_ss\_adc\_resolution\_t qm\_ss\_adc\_config\_t::resolution

12, 10, 8, 6-bit resolution.

Definition at line 105 of file qm\_ss\_adc.h.

Referenced by `qm_ss_adc_set_config()`.

### 5.53 `qm_ss_adc_xfer_t` Struct Reference

SS ADC transfer type.

```
#include <qm_ss_adc.h>
```

#### Data Fields

- [`qm\_ss\_adc\_channel\_t \* ch`](#)  
*Channel sequence array (1-32 channels).*
- `uint8_t ch_len`  
*Number of channels in the above array.*
- [`qm\_ss\_adc\_sample\_t \* samples`](#)  
*Array to store samples.*
- `uint32_t samples_len`  
*Length of sample array.*
- `void(* callback)(void *data, int error, qm\_ss\_adc\_status\_t status, qm\_ss\_adc\_cb\_source\_t source)`  
*Transfer callback.*
- `void * callback_data`  
*Callback user data.*

#### 5.53.1 Detailed Description

SS ADC transfer type.

Definition at line 111 of file `qm_ss_adc.h`.

#### 5.53.2 Field Documentation

##### 5.53.2.1 `void(* qm_ss_adc_xfer_t::callback)(void *data, int error, qm\_ss\_adc\_status\_t status, qm\_ss\_adc\_cb\_source\_t source)`

Transfer callback.

Called when a conversion is performed or an error is detected.

#### Parameters

|                 |               |   |
|-----------------|---------------|---|
| <code>in</code> | <i>data</i>   | The callback user data.   |
| <code>in</code> | <i>error</i>  | 0 on success. Negative <a href="#"><code>errno</code></a> for possible error codes. |
| <code>in</code> | <i>status</i> | ADC status.   |
| <code>in</code> | <i>source</i> | Interrupt callback source.  |

Definition at line 128 of file `qm_ss_adc.h`.

##### 5.53.2.2 `void* qm_ss_adc_xfer_t::callback_data`

Callback user data.

Definition at line 130 of file `qm_ss_adc.h`.

## 5.53.2.3 qm\_ss\_adc\_channel\_t\* qm\_ss\_adc\_xfer\_t::ch

Channel sequence array (1-32 channels).

Definition at line 112 of file qm\_ss\_adc.h.

Referenced by qm\_ss\_adc\_convert(), and qm\_ss\_adc\_irq\_convert().

## 5.53.2.4 uint8\_t qm\_ss\_adc\_xfer\_t::ch\_len

Number of channels in the above array.

Definition at line 113 of file qm\_ss\_adc.h.

Referenced by qm\_ss\_adc\_convert(), and qm\_ss\_adc\_irq\_convert().

## 5.53.2.5 qm\_ss\_adc\_sample\_t\* qm\_ss\_adc\_xfer\_t::samples

Array to store samples.

Definition at line 114 of file qm\_ss\_adc.h.

Referenced by qm\_ss\_adc\_convert(), and qm\_ss\_adc\_irq\_convert().

## 5.53.2.6 uint32\_t qm\_ss\_adc\_xfer\_t::samples\_len

Length of sample array.

Definition at line 115 of file qm\_ss\_adc.h.

Referenced by qm\_ss\_adc\_convert(), and qm\_ss\_adc\_irq\_convert().

## 5.54 qm\_ss\_gpio\_port\_config\_t Struct Reference

GPIO port configuration type.

```
#include <qm_ss_gpio.h>
```

## Data Fields

- uint32\_t [direction](#)  
*SS GPIO direction, 0b: input, 1b: output.*
- uint32\_t [int\\_en](#)  
*Interrupt enable.*
- uint32\_t [int\\_type](#)  
*Interrupt type, 0b: level; 1b: edge.*
- uint32\_t [int\\_polarity](#)  
*Interrupt polarity, 0b: low, 1b: high.*
- uint32\_t [int\\_debounce](#)  
*Debounce on/off.*
- void(\* [callback](#))(void \*data, uint32\_t int\_status)  
*User callback.*
- void \* [callback\\_data](#)  
*Callback user data.*

## 5.54.1 Detailed Description

GPIO port configuration type.

Each bit in the registers control a GPIO pin.

Definition at line 32 of file qm\_ss\_gpio.h.

### 5.54.2 Field Documentation

#### 5.54.2.1 void(\* qm\_ss\_gpio\_port\_config\_t::callback)(void \*data, uint32\_t int\_status)

User callback.

Called for any interrupt on the Sensor Subsystem GPIO.

##### Parameters

|           |                   |                             |
|-----------|-------------------|-----------------------------|
| <i>in</i> | <i>data</i>       | The callback user data.     |
| <i>in</i> | <i>int_status</i> | Bitfield of triggered pins. |

Definition at line 47 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

#### 5.54.2.2 void\* qm\_ss\_gpio\_port\_config\_t::callback\_data

Callback user data.

Definition at line 48 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

#### 5.54.2.3 uint32\_t qm\_ss\_gpio\_port\_config\_t::direction

SS GPIO direction, 0b: input, 1b: output.

Definition at line 33 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

#### 5.54.2.4 uint32\_t qm\_ss\_gpio\_port\_config\_t::int\_debounce

Debounce on/off.

Definition at line 37 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

#### 5.54.2.5 uint32\_t qm\_ss\_gpio\_port\_config\_t::int\_en

Interrupt enable.

Definition at line 34 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

#### 5.54.2.6 uint32\_t qm\_ss\_gpio\_port\_config\_t::int\_polarity

Interrupt polarity, 0b: low, 1b: high.

Definition at line 36 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

#### 5.54.2.7 uint32\_t qm\_ss\_gpio\_port\_config\_t::int\_type

Interrupt type, 0b: level; 1b: edge.

Definition at line 35 of file qm\_ss\_gpio.h.

Referenced by qm\_ss\_gpio\_set\_config().

## 5.55 qm\_ss\_i2c\_config\_t Struct Reference

QM SS I2C configuration type.

```
#include <qm_ss_i2c.h>
```

#### Data Fields

- [qm\\_ss\\_i2c\\_speed\\_t speed](#)  
*Standard, Fast Mode.*
- [qm\\_ss\\_i2c\\_addr\\_t address\\_mode](#)  
*7 or 10 bit addressing.*

#### 5.55.1 Detailed Description

QM SS I2C configuration type.

Definition at line 70 of file qm\_ss\_i2c.h.

#### 5.55.2 Field Documentation

##### 5.55.2.1 qm\_ss\_i2c\_addr\_t qm\_ss\_i2c\_config\_t::address\_mode

7 or 10 bit addressing.

Definition at line 72 of file qm\_ss\_i2c.h.

Referenced by [qm\\_ss\\_i2c\\_set\\_config\(\)](#).

##### 5.55.2.2 qm\_ss\_i2c\_speed\_t qm\_ss\_i2c\_config\_t::speed

Standard, Fast Mode.

Definition at line 71 of file qm\_ss\_i2c.h.

Referenced by [qm\\_ss\\_i2c\\_set\\_config\(\)](#).

## 5.56 qm\_ss\_i2c\_transfer\_t Struct Reference

QM SS I2C transfer type.

```
#include <qm_ss_i2c.h>
```

#### Data Fields

- [uint8\\_t \\* tx](#)  
*Write data.*
- [uint32\\_t tx\\_len](#)  
*Write data length.*
- [uint8\\_t \\* rx](#)  
*Read data.*
- [uint32\\_t rx\\_len](#)  
*Read buffer length.*
- [bool stop](#)  
*Generate master STOP.*
- [void\(\\* callback\)\(void \\*data, int rc, qm\\_ss\\_i2c\\_status\\_t status, uint32\\_t len\)](#)  
*User callback.*
- [void \\* callback\\_data](#)  
*User callback data.*

### 5.56.1 Detailed Description

QM SS I2C transfer type.

- if tx len is 0: perform receive-only transaction.
- if rx len is 0: perform transmit-only transaction.
- both tx and rx len not 0: perform a transmit-then-receive combined transaction.

Definition at line 82 of file qm\_ss\_i2c.h.

### 5.56.2 Field Documentation

#### 5.56.2.1 void(\* qm\_ss\_i2c\_transfer\_t::callback)(void \*data, int rc, qm\_ss\_i2c\_status\_t status, uint32\_t len)

User callback.

Parameters

|    |               |  |
|----|---------------|--|
| in | <i>data</i>   | User defined data.   |
| in | <i>rc</i>     | 0 on success. Negative <a href="#">errno</a> for possible error codes. |
| in | <i>status</i> | I2C status.  |
| in | <i>len</i>    | Length of the transfer if succesfull, 0 otherwise.                     |

Definition at line 98 of file qm\_ss\_i2c.h.

#### 5.56.2.2 void\* qm\_ss\_i2c\_transfer\_t::callback\_data

User callback data.

Definition at line 100 of file qm\_ss\_i2c.h.

#### 5.56.2.3 uint8\_t\* qm\_ss\_i2c\_transfer\_t::rx

Read data.

Definition at line 85 of file qm\_ss\_i2c.h.

#### 5.56.2.4 uint32\_t qm\_ss\_i2c\_transfer\_t::rx\_len

Read buffer length.

Definition at line 86 of file qm\_ss\_i2c.h.

Referenced by qm\_ss\_i2c\_master\_irq\_transfer().

#### 5.56.2.5 bool qm\_ss\_i2c\_transfer\_t::stop

Generate master STOP.

Definition at line 87 of file qm\_ss\_i2c.h.

#### 5.56.2.6 uint8\_t\* qm\_ss\_i2c\_transfer\_t::tx

Write data.

Definition at line 83 of file qm\_ss\_i2c.h.

#### 5.56.2.7 uint32\_t qm\_ss\_i2c\_transfer\_t::tx\_len

Write data length.

Definition at line 84 of file qm\_ss\_i2c.h.

## 5.57 qm\_ss\_spi\_async\_transfer\_t Struct Reference

SPI IRQ transfer type.

```
#include <qm_ss_spi.h>
```

### Data Fields

- `uint8_t * tx`  
*Write data buffer pointer.*
- `uint16_t tx_len`  
*Write data length.*
- `uint8_t * rx`  
*Read data buffer pointer.*
- `uint16_t rx_len`  
*Read buffer length.*
- `void(* callback)(void *data, int error, qm_ss_spi_status_t status, uint16_t len)`  
*Transfer callback.*
- `void * callback_data`  
*Callback user data.*

### 5.57.1 Detailed Description

SPI IRQ transfer type.

Definition at line 129 of file `qm_ss_spi.h`.

### 5.57.2 Field Documentation

#### 5.57.2.1 `void(* qm_ss_spi_async_transfer_t::callback)(void *data, int error, qm_ss_spi_status_t status, uint16_t len)`

Transfer callback.

Called after all data is transmitted/received or if the driver detects an error during the SPI transfer.

#### Parameters

|                 |                     |   |
|-----------------|---------------------|---|
| <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> | The SPI module status.  |
| <code>in</code> | <code>len</code>    | The amount of frames transmitted.                                   |

Definition at line 147 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_transfer_terminate()`.

#### 5.57.2.2 `void* qm_ss_spi_async_transfer_t::callback_data`

Callback user data.

Definition at line 149 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_transfer_terminate()`.

#### 5.57.2.3 `uint8_t* qm_ss_spi_async_transfer_t::rx`

Read data buffer pointer.

Definition at line 132 of file `qm_ss_spi.h`.



#### 5.57.2.4 `uint16_t qm_ss_spi_async_transfer_t::rx_len`

Read buffer length.

Definition at line 133 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_irq_transfer()`, and `qm_ss_spi_transfer_terminate()`.

#### 5.57.2.5 `uint8_t* qm_ss_spi_async_transfer_t::tx`

Write data buffer pointer.

Definition at line 130 of file `qm_ss_spi.h`.

#### 5.57.2.6 `uint16_t qm_ss_spi_async_transfer_t::tx_len`

Write data length.

Definition at line 131 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_irq_transfer()`, and `qm_ss_spi_transfer_terminate()`.

### 5.58 `qm_ss_spi_config_t` Struct Reference

SPI configuration type.

```
#include <qm_ss_spi.h>
```

#### Data Fields

- [`qm\_ss\_spi\_frame\_size\_t frame\_size`](#)  
*Frame Size.*
- [`qm\_ss\_spi\_tmode\_t transfer\_mode`](#)  
*Transfer mode (enum).*
- [`qm\_ss\_spi\_bmode\_t bus\_mode`](#)  
*Bus mode (enum).*
- [`uint16\_t clk\_divider`](#)  
*Clock divider.*

#### 5.58.1 Detailed Description

SPI configuration type.

Definition at line 112 of file `qm_ss_spi.h`.

#### 5.58.2 Field Documentation

##### 5.58.2.1 `qm_ss_spi_bmode_t qm_ss_spi_config_t::bus_mode`

Bus mode (enum).

Definition at line 115 of file `qm_ss_spi.h`.

Referenced by `qm_ss_spi_set_config()`.

##### 5.58.2.2 `uint16_t qm_ss_spi_config_t::clk_divider`

Clock divider.

The clock divider sets the SPI speed on the interface. A value of 0 will disable SCK. The LSB of this value is ignored.

Definition at line 123 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_set\_config().

#### 5.58.2.3 qm\_ss\_spi\_frame\_size\_t qm\_ss\_spi\_config\_t::frame\_size

Frame Size.

Definition at line 113 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_set\_config().

#### 5.58.2.4 qm\_ss\_spi\_tmode\_t qm\_ss\_spi\_config\_t::transfer\_mode

Transfer mode (enum).

Definition at line 114 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_set\_config().

## 5.59 qm\_ss\_spi\_transfer\_t Struct Reference

SPI transfer type.

```
#include <qm_ss_spi.h>
```

### Data Fields

- `uint8_t * tx`  
*Write data buffer pointer.*
- `uint16_t tx_len`  
*Write data length.*
- `uint8_t * rx`  
*Read data buffer pointer.*
- `uint16_t rx_len`  
*Read buffer length.*

### 5.59.1 Detailed Description

SPI transfer type.

Definition at line 155 of file qm\_ss\_spi.h.

### 5.59.2 Field Documentation

#### 5.59.2.1 `uint8_t*` qm\_ss\_spi\_transfer\_t::rx

Read data buffer pointer.

Definition at line 158 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_transfer().

#### 5.59.2.2 `uint16_t` qm\_ss\_spi\_transfer\_t::rx\_len

Read buffer length.

Definition at line 159 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_transfer().

### 5.59.2.3 uint8\_t\* qm\_ss\_spi\_transfer\_t::tx

Write data buffer pointer.

Definition at line 156 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_transfer().

### 5.59.2.4 uint16\_t qm\_ss\_spi\_transfer\_t::tx\_len

Write data length.

Definition at line 157 of file qm\_ss\_spi.h.

Referenced by qm\_ss\_spi\_transfer().

## 5.60 qm\_ss\_timer\_config\_t Struct Reference

Sensor Subsystem Timer Configuration Type.

```
#include <qm_ss_timer.h>
```

### Data Fields

- bool [watchdog\\_mode](#)  
*Watchdog mode.*
- bool [inc\\_run\\_only](#)  
*Increments in run state only.*
- bool [int\\_en](#)  
*Interrupt enable.*
- uint32\_t [count](#)  
*Final count value.*
- void(\* [callback](#))(void \*data)  
*User callback.*
- void \* [callback\\_data](#)  
*Callback user data.*

### 5.60.1 Detailed Description

Sensor Subsystem Timer Configuration Type.

Definition at line 21 of file qm\_ss\_timer.h.

### 5.60.2 Field Documentation

#### 5.60.2.1 void(\* qm\_ss\_timer\_config\_t::callback)(void \*data)

User callback.

Called for any interrupt on the Sensor Subsystem Timer.

#### Parameters

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

Definition at line 43 of file qm\_ss\_timer.h.

Referenced by qm\_ss\_timer\_set\_config().

#### 5.60.2.2 void\* qm\_ss\_timer\_config\_t::callback\_data

Callback user data.

Definition at line 44 of file qm\_ss\_timer.h.

Referenced by qm\_ss\_timer\_set\_config().

#### 5.60.2.3 uint32\_t qm\_ss\_timer\_config\_t::count

Final count value.

Definition at line 34 of file qm\_ss\_timer.h.

Referenced by qm\_ss\_timer\_set\_config().

#### 5.60.2.4 bool qm\_ss\_timer\_config\_t::inc\_run\_only

Increments in run state only.

If this field is set to 0, the timer will count in both halt state and running state. When set to 1, this will only increment in running state.

Definition at line 32 of file qm\_ss\_timer.h.

Referenced by qm\_ss\_timer\_set\_config().

#### 5.60.2.5 bool qm\_ss\_timer\_config\_t::int\_en

Interrupt enable.

Definition at line 33 of file qm\_ss\_timer.h.

Referenced by qm\_ss\_timer\_set\_config().

#### 5.60.2.6 bool qm\_ss\_timer\_config\_t::watchdog\_mode

Watchdog mode.

Definition at line 22 of file qm\_ss\_timer.h.

Referenced by qm\_ss\_timer\_set\_config().

## 5.61 qm\_uart\_config\_t Struct Reference

UART configuration type.

```
#include <qm_uart.h>
```

### Data Fields

- [qm\\_uart\\_lc\\_t line\\_control](#)  
*Line control (enum).*
- [uint32\\_t baud\\_divisor](#)  
*Baud Divisor.*
- [bool hw\\_fc](#)  
*Hardware Automatic Flow Control.*
- [bool int\\_en](#)  
*Interrupt enable.*

### 5.61.1 Detailed Description

UART configuration type.

Definition at line 161 of file qm\_uart.h.

## 5.61.2 Field Documentation

### 5.61.2.1 uint32\_t qm\_uart\_config\_t::baud\_divisor

Baud Divisor.

Definition at line 163 of file qm\_uart.h.

Referenced by qm\_uart\_set\_config().

### 5.61.2.2 bool qm\_uart\_config\_t::hw\_fc

Hardware Automatic Flow Control.

Definition at line 164 of file qm\_uart.h.

Referenced by qm\_uart\_set\_config().

### 5.61.2.3 bool qm\_uart\_config\_t::int\_en

Interrupt enable.

Definition at line 165 of file qm\_uart.h.

### 5.61.2.4 qm\_uart\_lc\_t qm\_uart\_config\_t::line\_control

Line control (enum).

Definition at line 162 of file qm\_uart.h.

Referenced by qm\_uart\_set\_config().

## 5.62 qm\_uart\_reg\_t Struct Reference

UART register map.

```
#include <qm_soc_regs.h>
```

### Data Fields

- QM\_RW uint32\_t [rbr\\_thr\\_dll](#)  
*Rx Buffer/ Tx Holding/ Div Latch Low.*
- QM\_RW uint32\_t [ier\\_dlh](#)  
*Interrupt Enable / Divisor Latch High.*
- QM\_RW uint32\_t [iir\\_fcr](#)  
*Interrupt Identification / FIFO Control.*
- QM\_RW uint32\_t [lcr](#)  
*Line Control.*
- QM\_RW uint32\_t [mcr](#)  
*MODEM Control.*
- QM\_RW uint32\_t [lsr](#)  
*Line Status.*
- QM\_RW uint32\_t [msr](#)  
*MODEM Status.*
- QM\_RW uint32\_t [scr](#)  
*Scratchpad.*
- QM\_RW uint32\_t [usr](#)

- UART Status.*
- QM\_RW uint32\_t [htx](#)  
*Halt Transmission.*
- QM\_RW uint32\_t [dmasa](#)  
*DMA Software Acknowledge.*
- QM\_RW uint32\_t [tcr](#)  
*Transceiver Control Register.*
- QM\_RW uint32\_t [de\\_en](#)  
*Driver Output Enable Register.*
- QM\_RW uint32\_t [re\\_en](#)  
*Receiver Output Enable Register.*
- QM\_RW uint32\_t [det](#)  
*Driver Output Enable Timing Register.*
- QM\_RW uint32\_t [tat](#)  
*TurnAround Timing Register.*
- QM\_RW uint32\_t [dlf](#)  
*Divisor Latch Fraction.*
- QM\_RW uint32\_t [rar](#)  
*Receive Address Register.*
- QM\_RW uint32\_t [tar](#)  
*Transmit Address Register.*
- QM\_RW uint32\_t [lcr\\_ext](#)  
*Line Extended Control Register.*

### 5.62.1 Detailed Description

UART register map.

Definition at line 601 of file qm\_soc\_regs.h.

### 5.62.2 Field Documentation

#### 5.62.2.1 QM\_RW uint32\_t qm\_uart\_reg\_t::de\_en

Driver Output Enable Register.

Definition at line 617 of file qm\_soc\_regs.h.

#### 5.62.2.2 QM\_RW uint32\_t qm\_uart\_reg\_t::det

Driver Output Enable Timing Register.

Definition at line 619 of file qm\_soc\_regs.h.

#### 5.62.2.3 QM\_RW uint32\_t qm\_uart\_reg\_t::dlf

Divisor Latch Fraction.

Definition at line 621 of file qm\_soc\_regs.h.

Referenced by `qm_uart_set_config()`.

#### 5.62.2.4 QM\_RW uint32\_t qm\_uart\_reg\_t::dmasa

DMA Software Acknowledge.

Definition at line 615 of file qm\_soc\_regs.h.

#### 5.62.2.5 QM\_RW uint32\_t qm\_uart\_reg\_t::htx

Halt Transmission.

Definition at line 614 of file qm\_soc\_regs.h.

#### 5.62.2.6 QM\_RW uint32\_t qm\_uart\_reg\_t::ier\_dlh

Interrupt Enable / Divisor Latch High.

Definition at line 604 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_irq\_read(), qm\_uart\_irq\_read\_terminate(), qm\_uart\_irq\_write(), qm\_uart\_irq\_write\_terminate(), and qm\_uart\_set\_config().

#### 5.62.2.7 QM\_RW uint32\_t qm\_uart\_reg\_t::iir\_fcr

Interrupt Identification / FIFO Control.

Definition at line 605 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_dma\_read(), qm\_uart\_dma\_write(), qm\_uart\_irq\_read(), qm\_uart\_irq\_write(), and qm\_uart\_set\_config().

#### 5.62.2.8 QM\_RW uint32\_t qm\_uart\_reg\_t::lcr

Line Control.

Definition at line 606 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_set\_config().

#### 5.62.2.9 QM\_RW uint32\_t qm\_uart\_reg\_t::lcr\_ext

Line Extended Control Register.

Definition at line 624 of file qm\_soc\_regs.h.

#### 5.62.2.10 QM\_RW uint32\_t qm\_uart\_reg\_t::lsr

Line Status.

Definition at line 608 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_get\_status(), qm\_uart\_read(), qm\_uart\_write(), and qm\_uart\_write\_buffer().

#### 5.62.2.11 QM\_RW uint32\_t qm\_uart\_reg\_t::mcr

MODEM Control.

Definition at line 607 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_set\_config().

#### 5.62.2.12 QM\_RW uint32\_t qm\_uart\_reg\_t::msr

MODEM Status.

Definition at line 609 of file qm\_soc\_regs.h.

#### 5.62.2.13 QM\_RW uint32\_t qm\_uart\_reg\_t::rar

Receive Address Register.

Definition at line 622 of file qm\_soc\_regs.h.

## 5.62.2.14 QM\_RW uint32\_t qm\_uart\_reg\_t::rbr\_thr\_dll

Rx Buffer/ Tx Holding/ Div Latch Low.

Definition at line 603 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_dma\_read(), qm\_uart\_dma\_write(), qm\_uart\_read(), qm\_uart\_read\_non\_block(), qm\_uart\_set\_config(), qm\_uart\_write(), qm\_uart\_write\_buffer(), and qm\_uart\_write\_non\_block().

## 5.62.2.15 QM\_RW uint32\_t qm\_uart\_reg\_t::re\_en

Receiver Output Enable Register.

Definition at line 618 of file qm\_soc\_regs.h.

## 5.62.2.16 QM\_RW uint32\_t qm\_uart\_reg\_t::scr

Scratchpad.

Definition at line 610 of file qm\_soc\_regs.h.

Referenced by qm\_uart\_get\_status().

## 5.62.2.17 QM\_RW uint32\_t qm\_uart\_reg\_t::tar

Transmit Address Register.

Definition at line 623 of file qm\_soc\_regs.h.

## 5.62.2.18 QM\_RW uint32\_t qm\_uart\_reg\_t::tat

TurnAround Timing Register.

Definition at line 620 of file qm\_soc\_regs.h.

## 5.62.2.19 QM\_RW uint32\_t qm\_uart\_reg\_t::tcr

Transceiver Control Register.

Definition at line 616 of file qm\_soc\_regs.h.

## 5.62.2.20 QM\_RW uint32\_t qm\_uart\_reg\_t::usr

UART Status.

Definition at line 612 of file qm\_soc\_regs.h.

## 5.63 qm\_uart\_transfer\_t Struct Reference

UART asynchronous transfer structure.

```
#include <qm_uart.h>
```

## Data Fields

- uint8\_t \* [data](#)  
*Pre-allocated write or read buffer.*
- uint32\_t [data\\_len](#)  
*Number of bytes to transfer.*
- void(\* [callback](#))(void \*[data](#), int error, [qm\\_uart\\_status\\_t](#) status, uint32\_t len)  
*Transfer callback.*
- void \* [callback\\_data](#)  
*Callback identifier.*



### 5.63.1 Detailed Description

UART asynchronous transfer structure.

Definition at line 171 of file `qm_uart.h`.

### 5.63.2 Field Documentation

#### 5.63.2.1 `void(* qm_uart_transfer_t::callback)(void *data, int error, qm_uart_status_t status, uint32_t len)`

Transfer callback.

##### Parameters

|    |               |  |
|----|---------------|--|
| in | <i>data</i>   | Callback user data.  |
| in | <i>error</i>  | 0 on success. Negative <a href="#">errno</a> 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 184 of file `qm_uart.h`.

Referenced by `qm_uart_irq_read_terminate()`, and `qm_uart_irq_write_terminate()`.

#### 5.63.2.2 `void* qm_uart_transfer_t::callback_data`

Callback identifier.

Definition at line 186 of file `qm_uart.h`.

Referenced by `qm_uart_irq_read_terminate()`, and `qm_uart_irq_write_terminate()`.

#### 5.63.2.3 `uint8_t* qm_uart_transfer_t::data`

Pre-allocated write or read buffer.

Definition at line 172 of file `qm_uart.h`.

Referenced by `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

#### 5.63.2.4 `uint32_t qm_uart_transfer_t::data_len`

Number of bytes to transfer.

Definition at line 173 of file `qm_uart.h`.

Referenced by `qm_uart_dma_read()`, and `qm_uart_dma_write()`.

## 5.64 `qm_wdt_config_t` Struct Reference

QM WDT configuration type.

```
#include <qm_wdt.h>
```

### Data Fields

- [qm\\_wdt\\_clock\\_timeout\\_cycles\\_t](#) `timeout`  
*Timeout in cycles.*
- [qm\\_wdt\\_mode\\_t](#) `mode`  
*Watchdog response mode.*
- `void(* callback )(void *data)`  
*User callback.*
- `void * callback\_data`

*Callback user data.*

#### 5.64.1 Detailed Description

QM WDT configuration type.

Definition at line 72 of file qm\_wdt.h.

#### 5.64.2 Field Documentation

##### 5.64.2.1 void(\* qm\_wdt\_config\_t::callback)(void \*data)

User callback.

param[in] data Callback user data.

Definition at line 81 of file qm\_wdt.h.

Referenced by qm\_wdt\_set\_config().

##### 5.64.2.2 void\* qm\_wdt\_config\_t::callback\_data

Callback user data.

Definition at line 82 of file qm\_wdt.h.

Referenced by qm\_wdt\_set\_config().

##### 5.64.2.3 qm\_wdt\_mode\_t qm\_wdt\_config\_t::mode

Watchdog response mode.

Definition at line 74 of file qm\_wdt.h.

Referenced by qm\_wdt\_set\_config().

##### 5.64.2.4 qm\_wdt\_clock\_timeout\_cycles\_t qm\_wdt\_config\_t::timeout

Timeout in cycles.

Definition at line 73 of file qm\_wdt.h.

Referenced by qm\_wdt\_set\_config().

## 5.65 qm\_wdt\_reg\_t Struct Reference

Watchdog timer register map.

```
#include <qm_soc_regs.h>
```

#### Data Fields

- QM\_RW uint32\_t [wdt\\_cr](#)  
*Control Register.*
- QM\_RW uint32\_t [wdt\\_torr](#)  
*Timeout Range Register.*
- QM\_RW uint32\_t [wdt\\_ccvr](#)  
*Current Counter Value Register.*
- QM\_RW uint32\_t [wdt\\_crr](#)  
*Current Restart Register.*
- QM\_RW uint32\_t [wdt\\_stat](#)

- Interrupt Status Register.*
  - QM\_RW uint32\_t [wdt\\_eoi](#)
  - Interrupt Clear Register.*
  - QM\_RW uint32\_t [wdt\\_comp\\_param\\_5](#)
  - Component Parameters.*
  - QM\_RW uint32\_t [wdt\\_comp\\_param\\_4](#)
  - Component Parameters.*
  - QM\_RW uint32\_t [wdt\\_comp\\_param\\_3](#)
  - Component Parameters.*
  - QM\_RW uint32\_t [wdt\\_comp\\_param\\_2](#)
  - Component Parameters.*
  - QM\_RW uint32\_t [wdt\\_comp\\_param\\_1](#)
  - Component Parameters Register 1.*
  - QM\_RW uint32\_t [wdt\\_comp\\_version](#)
  - Component Version Register.*
  - QM\_RW uint32\_t [wdt\\_comp\\_type](#)
  - Component Type Register.*

### 5.65.1 Detailed Description

Watchdog timer register map.

Definition at line 561 of file `qm_soc_regs.h`.

### 5.65.2 Field Documentation

#### 5.65.2.1 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_ccvr

Current Counter Value Register.

Definition at line 564 of file `qm_soc_regs.h`.

#### 5.65.2.2 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_param\_1

Component Parameters Register 1.

Definition at line 573 of file `qm_soc_regs.h`.

#### 5.65.2.3 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_param\_2

Component Parameters.

Definition at line 571 of file `qm_soc_regs.h`.

#### 5.65.2.4 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_param\_3

Component Parameters.

Definition at line 570 of file `qm_soc_regs.h`.

#### 5.65.2.5 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_param\_4

Component Parameters.

Definition at line 569 of file `qm_soc_regs.h`.

#### 5.65.2.6 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_param\_5

Component Parameters.

Definition at line 568 of file `qm_soc_regs.h`.

**5.65.2.7 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_type**

Component Type Register.

Definition at line 575 of file qm\_soc\_regs.h.

**5.65.2.8 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_comp\_version**

Component Version Register.

Definition at line 574 of file qm\_soc\_regs.h.

**5.65.2.9 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_cr**

Control Register.

Definition at line 562 of file qm\_soc\_regs.h.

**5.65.2.10 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_crr**

Current Restart Register.

Definition at line 565 of file qm\_soc\_regs.h.

**5.65.2.11 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_eoi**

Interrupt Clear Register.

Definition at line 567 of file qm\_soc\_regs.h.

**5.65.2.12 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_stat**

Interrupt Status Register.

Definition at line 566 of file qm\_soc\_regs.h.

**5.65.2.13 QM\_RW uint32\_t qm\_wdt\_reg\_t::wdt\_torr**

Timeout Range Register.

Definition at line 563 of file qm\_soc\_regs.h.

## Index

- address\_mode
  - qm\_i2c\_config\_t, 204
  - qm\_ss\_i2c\_config\_t, 252
- alarm\_en
  - qm\_rtc\_config\_t, 222
- alarm\_val
  - qm\_rtc\_config\_t, 222
- allow\_agents
  - qm\_fpr\_config\_t, 199
- Always-on Counters, 23
  - qm\_aonc\_disable, 23
  - qm\_aonc\_enable, 24
  - qm\_aonc\_get\_value, 24
  - qm\_aonpt\_clear, 24
  - qm\_aonpt\_get\_status, 25
  - qm\_aonpt\_get\_value, 25
  - qm\_aonpt\_reset, 25
  - qm\_aonpt\_set\_config, 26
- Analog Comparator, 27
  - qm\_ac\_set\_config, 27
- aon\_vr
  - qm\_scss\_pmu\_reg\_t, 238
- aonc\_cfg
  - qm\_scss\_aon\_reg\_t, 226
- aonc\_cnt
  - qm\_scss\_aon\_reg\_t, 226
- aonpt\_cfg
  - qm\_scss\_aon\_reg\_t, 226
- aonpt\_cnt
  - qm\_scss\_aon\_reg\_t, 226
- aonpt\_ctrl
  - qm\_scss\_aon\_reg\_t, 226
- aonpt\_stat
  - qm\_scss\_aon\_reg\_t, 226
- baud\_divisor
  - qm\_uart\_config\_t, 259
- baudr
  - qm\_spi\_reg\_t, 244
- block\_size
  - qm\_dma\_transfer\_t, 196
- bus\_mode
  - qm\_spi\_config\_t, 242
  - qm\_ss\_spi\_config\_t, 255
- CLK\_EXT\_DIV\_1
  - Clock Management, 9
- CLK\_EXT\_DIV\_2
  - Clock Management, 9
- CLK\_EXT\_DIV\_4
  - Clock Management, 9
- CLK\_EXT\_DIV\_8
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_1
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_128
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_16
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_2
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_32
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_4
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_64
  - Clock Management, 9
- CLK\_GPIO\_DB\_DIV\_8
  - Clock Management, 9
- CLK\_PERIPH\_ADC
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_ADC\_REGISTER
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_ALL
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_CLK
  - SoC Registers (D2000), 159, 160
  - SoC Registers (SE), 178, 179
- CLK\_PERIPH\_DIV\_1
  - Clock Management, 9
- CLK\_PERIPH\_DIV\_2
  - Clock Management, 9
- CLK\_PERIPH\_DIV\_4
  - Clock Management, 9
- CLK\_PERIPH\_DIV\_8
  - Clock Management, 9
- CLK\_PERIPH\_GPIO\_DB
  - SoC Registers (D2000), 159, 160
  - SoC Registers (SE), 178, 179
- CLK\_PERIPH\_GPIO\_INTERRUPT
  - SoC Registers (D2000), 159, 160
  - SoC Registers (SE), 178, 179
- CLK\_PERIPH\_GPIO\_REGISTER
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_I2C\_M0
  - SoC Registers (D2000), 159, 160
  - SoC Registers (SE), 178, 179
- CLK\_PERIPH\_I2C\_M0\_REGISTER
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_I2C\_M1
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_I2C\_M1\_REGISTER
  - SoC Registers (D2000), 160
  - SoC Registers (SE), 179
- CLK\_PERIPH\_I2S
  - Clock Management, 9

- SoC Registers (D2000), [160](#)
- SoC Registers (SE), [179](#)
- CLK\_PERIPH\_I2S\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_PWM\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_REGISTER
  - SoC Registers (D2000), [159](#), [160](#)
  - SoC Registers (SE), [178](#), [179](#)
- CLK\_PERIPH\_RTC\_REGISTER
  - SoC Registers (D2000), [159](#), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_SPI\_M0
  - SoC Registers (D2000), [159](#), [160](#)
  - SoC Registers (SE), [178](#), [179](#)
- CLK\_PERIPH\_SPI\_M0\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_SPI\_M1
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_SPI\_M1\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_SPI\_S
  - SoC Registers (D2000), [159](#), [160](#)
  - SoC Registers (SE), [178](#), [179](#)
- CLK\_PERIPH\_SPI\_S\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_UARTA\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_UARTB\_REGISTER
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_PERIPH\_WDT\_REGISTER
  - SoC Registers (D2000), [159](#), [160](#)
  - SoC Registers (SE), [179](#)
- CLK\_RTC\_DIV\_1
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_1024
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_128
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_16
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_16384
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_2
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_2048
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_256
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_32
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_32768
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_4
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_4096
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_512
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_64
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_8
  - Clock Management, [10](#)
- CLK\_RTC\_DIV\_8192
  - Clock Management, [10](#)
- CLK\_SYS\_CRYSTAL\_OSC
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_1
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_128
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_16
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_2
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_32
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_4
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_64
  - Clock Management, [10](#)
- CLK\_SYS\_DIV\_8
  - Clock Management, [10](#)
- CLK\_SYS\_HYB\_OSC\_16MHZ
  - Clock Management, [10](#)
- CLK\_SYS\_HYB\_OSC\_32MHZ
  - Clock Management, [10](#)
- CLK\_SYS\_HYB\_OSC\_4MHZ
  - Clock Management, [10](#)
- CLK\_SYS\_HYB\_OSC\_8MHZ
  - Clock Management, [10](#)
- CLK\_SYS\_RTC\_OSC
  - Clock Management, [10](#)
- callback
  - qm\_ac\_config\_t, [187](#)
  - qm\_adc\_xfer\_t, [189](#)
  - qm\_aonpt\_config\_t, [191](#)
  - qm\_gpio\_port\_config\_t, [200](#)
  - qm\_i2c\_transfer\_t, [212](#)
  - qm\_pic\_timer\_config\_t, [218](#)
  - qm\_pwm\_config\_t, [220](#)
  - qm\_rtc\_config\_t, [222](#)
  - qm\_spi\_async\_transfer\_t, [241](#)
  - qm\_ss\_adc\_xfer\_t, [249](#)
  - qm\_ss\_gpio\_port\_config\_t, [251](#)
  - qm\_ss\_i2c\_transfer\_t, [253](#)
  - qm\_ss\_spi\_async\_transfer\_t, [254](#)
  - qm\_ss\_timer\_config\_t, [257](#)

- qm\_uart\_transfer\_t, 263
- qm\_wdt\_config\_t, 264
- callback\_data
  - qm\_ac\_config\_t, 187
  - qm\_adc\_xfer\_t, 190
  - qm\_aonpt\_config\_t, 191
  - qm\_gpio\_port\_config\_t, 200
  - qm\_i2c\_transfer\_t, 212
  - qm\_pic\_timer\_config\_t, 218
  - qm\_pwm\_config\_t, 220
  - qm\_rtc\_config\_t, 224
  - qm\_spi\_async\_transfer\_t, 241
  - qm\_ss\_adc\_xfer\_t, 249
  - qm\_ss\_gpio\_port\_config\_t, 251
  - qm\_ss\_i2c\_transfer\_t, 253
  - qm\_ss\_spi\_async\_transfer\_t, 254
  - qm\_ss\_timer\_config\_t, 257
  - qm\_uart\_transfer\_t, 263
  - qm\_wdt\_config\_t, 264
- ccr
  - qm\_mvic\_reg\_t, 216
- ccu\_ext\_clock\_ctl
  - qm\_scss\_ccu\_reg\_t, 227
- ccu\_gpio\_db\_clk\_ctl
  - qm\_scss\_ccu\_reg\_t, 227
- ccu\_lp\_clk\_ctl
  - qm\_scss\_ccu\_reg\_t, 227
- ccu\_mlayer\_ahb\_ctl
  - qm\_scss\_ccu\_reg\_t, 228
- ccu\_periph\_clk\_div\_ctl0
  - qm\_scss\_ccu\_reg\_t, 228
- ccu\_periph\_clk\_gate\_ctl
  - qm\_scss\_ccu\_reg\_t, 228
- ccu\_ss\_periph\_clk\_gate\_ctl
  - qm\_scss\_ccu\_reg\_t, 228
- ccu\_sys\_clk\_ctl
  - qm\_scss\_ccu\_reg\_t, 228
- cfg\_lock
  - qm\_scss\_peripheral\_reg\_t, 237
- ch
  - qm\_adc\_xfer\_t, 190
  - qm\_ss\_adc\_xfer\_t, 249
- ch\_len
  - qm\_adc\_xfer\_t, 190
  - qm\_ss\_adc\_xfer\_t, 250
- client\_callback
  - qm\_dma\_channel\_config\_t, 193
- clk\_adc\_set\_div
  - Clock Management, 11
- clk\_divider
  - qm\_spi\_config\_t, 242
  - qm\_ss\_spi\_config\_t, 255
- clk\_ext\_div\_t
  - Clock Management, 9
- clk\_ext\_set\_div
  - Clock Management, 11
- clk\_gpio\_db\_div\_t
  - Clock Management, 9
- clk\_gpio\_db\_set\_div
  - Clock Management, 11
- clk\_periph\_disable
  - Clock Management, 12
- clk\_periph\_div\_t
  - Clock Management, 9
- clk\_periph\_enable
  - Clock Management, 12
- clk\_periph\_set\_div
  - Clock Management, 12
- clk\_periph\_t
  - SoC Registers (D2000), 159
  - SoC Registers (SE), 178
- clk\_rtc\_div\_t
  - Clock Management, 9
- clk\_rtc\_set\_div
  - Clock Management, 13
- clk\_sys\_div\_t
  - Clock Management, 10
- clk\_sys\_get\_ticks\_per\_us
  - Clock Management, 13
- clk\_sys\_mode\_t
  - Clock Management, 10
- clk\_sys\_set\_mode
  - Clock Management, 13
- clk\_sys\_udelay
  - Clock Management, 14
- clk\_trim\_apply
  - Clock Management, 14
- clk\_trim\_read
  - Clock Management, 14
- Clock Management, 8
  - CLK\_EXT\_DIV\_1, 9
  - CLK\_EXT\_DIV\_2, 9
  - CLK\_EXT\_DIV\_4, 9
  - CLK\_EXT\_DIV\_8, 9
  - CLK\_GPIO\_DB\_DIV\_1, 9
  - CLK\_GPIO\_DB\_DIV\_128, 9
  - CLK\_GPIO\_DB\_DIV\_16, 9
  - CLK\_GPIO\_DB\_DIV\_2, 9
  - CLK\_GPIO\_DB\_DIV\_32, 9
  - CLK\_GPIO\_DB\_DIV\_4, 9
  - CLK\_GPIO\_DB\_DIV\_64, 9
  - CLK\_GPIO\_DB\_DIV\_8, 9
  - CLK\_PERIPH\_DIV\_1, 9
  - CLK\_PERIPH\_DIV\_2, 9
  - CLK\_PERIPH\_DIV\_4, 9
  - CLK\_PERIPH\_DIV\_8, 9
  - CLK\_RTC\_DIV\_1, 10
  - CLK\_RTC\_DIV\_1024, 10
  - CLK\_RTC\_DIV\_128, 10
  - CLK\_RTC\_DIV\_16, 10
  - CLK\_RTC\_DIV\_16384, 10
  - CLK\_RTC\_DIV\_2, 10
  - CLK\_RTC\_DIV\_2048, 10
  - CLK\_RTC\_DIV\_256, 10
  - CLK\_RTC\_DIV\_32, 10
  - CLK\_RTC\_DIV\_32768, 10

- CLK\_RTC\_DIV\_4, 10
- CLK\_RTC\_DIV\_4096, 10
- CLK\_RTC\_DIV\_512, 10
- CLK\_RTC\_DIV\_64, 10
- CLK\_RTC\_DIV\_8, 10
- CLK\_RTC\_DIV\_8192, 10
- CLK\_SYS\_CRYSTAL\_OSC, 10
- CLK\_SYS\_DIV\_1, 10
- CLK\_SYS\_DIV\_128, 10
- CLK\_SYS\_DIV\_16, 10
- CLK\_SYS\_DIV\_2, 10
- CLK\_SYS\_DIV\_32, 10
- CLK\_SYS\_DIV\_4, 10
- CLK\_SYS\_DIV\_64, 10
- CLK\_SYS\_DIV\_8, 10
- CLK\_SYS\_HYB\_OSC\_16MHZ, 10
- CLK\_SYS\_HYB\_OSC\_32MHZ, 10
- CLK\_SYS\_HYB\_OSC\_4MHZ, 10
- CLK\_SYS\_HYB\_OSC\_8MHZ, 10
- CLK\_SYS\_RTC\_OSC, 10
- clk\_adc\_set\_div, 11
- clk\_ext\_div\_t, 9
- clk\_ext\_set\_div, 11
- clk\_gpio\_db\_div\_t, 9
- clk\_gpio\_db\_set\_div, 11
- clk\_periph\_disable, 12
- clk\_periph\_div\_t, 9
- clk\_periph\_enable, 12
- clk\_periph\_set\_div, 12
- clk\_rtc\_div\_t, 9
- clk\_rtc\_set\_div, 13
- clk\_sys\_div\_t, 10
- clk\_sys\_get\_ticks\_per\_us, 13
- clk\_sys\_mode\_t, 10
- clk\_sys\_set\_mode, 13
- clk\_sys\_udelay, 14
- clk\_trim\_apply, 14
- clk\_trim\_read, 14
- cmp\_en
  - qm\_scss\_cmp\_reg\_t, 230
- cmp\_pwr
  - qm\_scss\_cmp\_reg\_t, 230
- cmp\_ref\_pol
  - qm\_scss\_cmp\_reg\_t, 230
- cmp\_ref\_sel
  - qm\_scss\_cmp\_reg\_t, 230
- cmp\_stat\_clr
  - qm\_scss\_cmp\_reg\_t, 230
- controlreg
  - qm\_pwm\_channel\_t, 219
- cotps
  - qm\_scss\_info\_reg\_t, 232
- count
  - qm\_aonpt\_config\_t, 191
  - qm\_ss\_timer\_config\_t, 258
- ctrl
  - qm\_mbox\_msg\_t, 215
- ctrlr0
  - qm\_spi\_reg\_t, 244
- ctrlr1
  - qm\_spi\_reg\_t, 244
- currentvalue
  - qm\_pwm\_channel\_t, 219
- DMA
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_1, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_128, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_16, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_256, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_32, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_4, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_64, 29
  - QM\_DMA\_BURST\_TRANS\_LENGTH\_8, 29
  - QM\_DMA\_HANDSHAKE\_POLARITY\_HIGH, 29
  - QM\_DMA\_HANDSHAKE\_POLARITY\_LOW, 29
  - QM\_DMA\_MEMORY\_TO\_MEMORY, 29
  - QM\_DMA\_MEMORY\_TO\_PERIPHERAL, 29
  - QM\_DMA\_PERIPHERAL\_TO\_MEMORY, 29
  - QM\_DMA\_TRANS\_WIDTH\_128, 29
  - QM\_DMA\_TRANS\_WIDTH\_16, 29
  - QM\_DMA\_TRANS\_WIDTH\_256, 29
  - QM\_DMA\_TRANS\_WIDTH\_32, 29
  - QM\_DMA\_TRANS\_WIDTH\_64, 29
  - QM\_DMA\_TRANS\_WIDTH\_8, 29
- DMA\_HW\_IF\_I2C\_MASTER\_0\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_I2C\_MASTER\_0\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_I2C\_MASTER\_1\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_I2C\_MASTER\_1\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_I2S\_CAPTURE
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_I2S\_PLAYBACK
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_SPI\_MASTER\_0\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_SPI\_MASTER\_0\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_SPI\_MASTER\_1\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_SPI\_MASTER\_1\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_SPI\_SLAVE\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180



- DMA\_HW\_IF\_SPI\_SLAVE\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_UART\_A\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_UART\_A\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_UART\_B\_RX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA\_HW\_IF\_UART\_B\_TX
  - SoC Registers (D2000), 161
  - SoC Registers (SE), 180
- DMA, 28
  - qm\_dma\_burst\_length\_t, 29
  - qm\_dma\_channel\_direction\_t, 29
  - qm\_dma\_channel\_set\_config, 30
  - qm\_dma\_handshake\_polarity\_t, 29
  - qm\_dma\_init, 30
  - qm\_dma\_transfer\_mem\_to\_mem, 31
  - qm\_dma\_transfer\_set\_config, 31
  - qm\_dma\_transfer\_start, 32
  - qm\_dma\_transfer\_terminate, 32
  - qm\_dma\_transfer\_width\_t, 29
- data
  - qm\_mbox\_msg\_t, 215
  - qm\_uart\_transfer\_t, 263
- data\_len
  - qm\_uart\_transfer\_t, 263
- de\_en
  - qm\_uart\_reg\_t, 260
- destination\_address
  - qm\_dma\_transfer\_t, 196
- det
  - qm\_uart\_reg\_t, 260
- direction
  - qm\_gpio\_port\_config\_t, 200
  - qm\_ss\_gpio\_port\_config\_t, 251
- dlf
  - qm\_uart\_reg\_t, 260
- dmacr
  - qm\_spi\_reg\_t, 245
- dmardlr
  - qm\_spi\_reg\_t, 245
- dmasa
  - qm\_uart\_reg\_t, 260
- dmatdlr
  - qm\_spi\_reg\_t, 245
- dotps
  - qm\_scss\_info\_reg\_t, 232
- dr
  - qm\_spi\_reg\_t, 245
- en\_mask
  - qm\_fpr\_config\_t, 199
- eoi
  - qm\_mvic\_reg\_t, 216
  - qm\_pwm\_channel\_t, 219
- Error Codes, 7
- FPR
  - FPR\_VIOL\_MODE\_INTERRUPT, 40
  - FPR\_VIOL\_MODE\_PROBE, 40
  - FPR\_VIOL\_MODE\_RESET, 40
  - QM\_FPR\_0, 40
  - QM\_FPR\_1, 40
  - QM\_FPR\_2, 40
  - QM\_FPR\_3, 40
  - QM\_FPR\_DISABLE, 40
  - QM\_FPR\_DMA, 40
  - QM\_FPR\_ENABLE, 40
  - QM\_FPR\_HOST\_PROCESSOR, 40
  - QM\_FPR\_LOCK\_DISABLE, 40
  - QM\_FPR\_LOCK\_ENABLE, 40
  - QM\_FPR\_OTHER\_AGENTS, 40
  - QM\_FPR\_SENSOR\_SUBSYSTEM, 40
  - QM\_MAIN\_FLASH\_DATA, 39
  - QM\_MAIN\_FLASH\_NUM, 39
  - QM\_MAIN\_FLASH\_SYSTEM, 39
- FPR\_VIOL\_MODE\_INTERRUPT
  - FPR, 40
- FPR\_VIOL\_MODE\_PROBE
  - FPR, 40
- FPR\_VIOL\_MODE\_RESET
  - FPR, 40
- FPR, 39
  - qm\_flash\_region\_type\_t, 39
  - qm\_fpr\_en\_t, 39
  - qm\_fpr\_id\_t, 40
  - qm\_fpr\_read\_allow\_t, 40
  - qm\_fpr\_set\_config, 40
  - qm\_fpr\_set\_violation\_policy, 41
  - qm\_fpr\_viol\_mode\_t, 40
- Flash, 34
  - QM\_FLASH\_REGION\_DATA, 35
  - QM\_FLASH\_REGION\_NUM, 35
  - QM\_FLASH\_REGION\_OTP, 35
  - QM\_FLASH\_REGION\_SYS, 35
  - QM\_FLASH\_WRITE\_DISABLE, 34
  - QM\_FLASH\_WRITE\_ENABLE, 34
  - qm\_flash\_disable\_t, 34
  - qm\_flash\_mass\_erase, 35
  - qm\_flash\_page\_erase, 35
  - qm\_flash\_page\_update, 36
  - qm\_flash\_page\_write, 36
  - qm\_flash\_region\_t, 34
  - qm\_flash\_set\_config, 37
  - qm\_flash\_word\_write, 37
- frame\_size
  - qm\_spi\_config\_t, 243
  - qm\_ss\_spi\_config\_t, 256
- fs
  - qm\_scss\_info\_reg\_t, 232
- GPIO
  - QM\_GPIO\_HIGH, 42

- QM\_GPIO\_LOW, [42](#)
- QM\_GPIO\_STATE\_NUM, [42](#)
- GPIO, [42](#)
  - qm\_gpio\_clear\_pin, [42](#)
  - qm\_gpio\_read\_pin, [43](#)
  - qm\_gpio\_read\_port, [43](#)
  - qm\_gpio\_set\_config, [43](#)
  - qm\_gpio\_set\_pin, [44](#)
  - qm\_gpio\_set\_pin\_state, [44](#)
  - qm\_gpio\_state\_t, [42](#)
  - qm\_gpio\_write\_port, [45](#)
- gp0
  - qm\_scss\_gp\_reg\_t, [231](#)
- gp1
  - qm\_scss\_gp\_reg\_t, [231](#)
- gp2
  - qm\_scss\_gp\_reg\_t, [231](#)
- gp3
  - qm\_scss\_gp\_reg\_t, [231](#)
- gpio\_config\_reg1
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_config\_reg2
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_debounce
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_ext\_porta
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_id\_code
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_int\_bothedge
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_int\_polarity
  - qm\_gpio\_reg\_t, [202](#)
- gpio\_inten
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_intmask
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_intstatus
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_inttype\_level
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_ls\_sync
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_porta\_eoi
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_raw\_intstatus
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_swporta\_ddr
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_swporta\_dr
  - qm\_gpio\_reg\_t, [203](#)
- gpio\_ver\_id\_code
  - qm\_gpio\_reg\_t, [203](#)
- gps0
  - qm\_scss\_gp\_reg\_t, [231](#)
- gps1
  - qm\_scss\_gp\_reg\_t, [231](#)
- gps2
  - qm\_scss\_gp\_reg\_t, [231](#)
- gps3
  - qm\_scss\_gp\_reg\_t, [231](#)
- hi\_count
  - qm\_pwm\_config\_t, [220](#)
- htx
  - qm\_uart\_reg\_t, [260](#)
- hw\_fc
  - qm\_uart\_config\_t, [259](#)
- I2C
  - QM\_I2C\_10\_BIT, [47](#)
  - QM\_I2C\_7\_BIT, [47](#)
  - QM\_I2C\_BUSY, [48](#)
  - QM\_I2C\_IDLE, [47](#)
  - QM\_I2C\_MASTER, [47](#)
  - QM\_I2C\_SLAVE, [47](#)
  - QM\_I2C\_SPEED\_FAST, [47](#)
  - QM\_I2C\_SPEED\_FAST\_PLUS, [47](#)
  - QM\_I2C\_SPEED\_STD, [47](#)
  - QM\_I2C\_TX\_ABRT\_10ADDR1\_NOACK, [48](#)
  - QM\_I2C\_TX\_ABRT\_10ADDR2\_NOACK, [48](#)
  - QM\_I2C\_TX\_ABRT\_10B\_RD\_NORSTRT, [48](#)
  - QM\_I2C\_TX\_ABRT\_7B\_ADDR\_NOACK, [47](#)
  - QM\_I2C\_TX\_ABRT\_GCALL\_NOACK, [48](#)
  - QM\_I2C\_TX\_ABRT\_GCALL\_READ, [48](#)
  - QM\_I2C\_TX\_ABRT\_HS\_ACKDET, [48](#)
  - QM\_I2C\_TX\_ABRT\_HS\_NORSTRT, [48](#)
  - QM\_I2C\_TX\_ABRT\_MASTER\_DIS, [48](#)
  - QM\_I2C\_TX\_ABRT\_SBYTE\_ACKDET, [48](#)
  - QM\_I2C\_TX\_ABRT\_SLV\_ARBLOST, [48](#)
  - QM\_I2C\_TX\_ABRT\_SLVFLUSH\_TXFIFO, [48](#)
  - QM\_I2C\_TX\_ABRT\_SLVRD\_INTX, [48](#)
  - QM\_I2C\_TX\_ABRT\_TXDATA\_NOACK, [48](#)
  - QM\_I2C\_TX\_ABRT\_USER\_ABRT, [48](#)
  - QM\_I2C\_TX\_ARB\_LOST, [48](#)
- I2C, [46](#)
  - qm\_i2c\_addr\_t, [47](#)
  - qm\_i2c\_dma\_channel\_config, [48](#)
  - qm\_i2c\_dma\_transfer\_terminate, [49](#)
  - qm\_i2c\_get\_status, [49](#)
  - qm\_i2c\_irq\_transfer\_terminate, [49](#)
  - qm\_i2c\_master\_dma\_transfer, [50](#)
  - qm\_i2c\_master\_irq\_transfer, [50](#)
  - qm\_i2c\_master\_read, [51](#)
  - qm\_i2c\_master\_write, [51](#)
  - qm\_i2c\_mode\_t, [47](#)
  - qm\_i2c\_set\_config, [53](#)
  - qm\_i2c\_set\_speed, [53](#)
  - qm\_i2c\_speed\_t, [47](#)
  - qm\_i2c\_status\_t, [47](#)
- ISR, [59](#)
  - QM\_ISR\_DECLARE, [60–65](#)
- ic\_ack\_general\_call
  - qm\_i2c\_reg\_t, [207](#)
- ic\_clr\_activity
  - qm\_i2c\_reg\_t, [207](#)
- ic\_clr\_gen\_call

- qm\_i2c\_reg\_t, 207
- ic\_clr\_intr
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_rd\_req
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_restart\_det
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_rx\_done
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_rx\_over
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_rx\_under
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_start\_det
  - qm\_i2c\_reg\_t, 207
- ic\_clr\_stop\_det
  - qm\_i2c\_reg\_t, 208
- ic\_clr\_tx\_abrt
  - qm\_i2c\_reg\_t, 208
- ic\_clr\_tx\_over
  - qm\_i2c\_reg\_t, 208
- ic\_comp\_param\_1
  - qm\_i2c\_reg\_t, 208
- ic\_comp\_type
  - qm\_i2c\_reg\_t, 208
- ic\_comp\_version
  - qm\_i2c\_reg\_t, 208
- ic\_con
  - qm\_i2c\_reg\_t, 208
- ic\_data\_cmd
  - qm\_i2c\_reg\_t, 208
- ic\_dma\_cr
  - qm\_i2c\_reg\_t, 208
- ic\_dma\_rdlr
  - qm\_i2c\_reg\_t, 208
- ic\_dma\_tdlr
  - qm\_i2c\_reg\_t, 208
- ic\_enable
  - qm\_i2c\_reg\_t, 209
- ic\_enable\_status
  - qm\_i2c\_reg\_t, 209
- ic\_fs\_scl\_hcnt
  - qm\_i2c\_reg\_t, 209
- ic\_fs\_scl\_lcnt
  - qm\_i2c\_reg\_t, 209
- ic\_fs\_spklen
  - qm\_i2c\_reg\_t, 209
- ic\_hs\_maddr
  - qm\_i2c\_reg\_t, 209
- ic\_hs\_scl\_hcnt
  - qm\_i2c\_reg\_t, 209
- ic\_hs\_scl\_lcnt
  - qm\_i2c\_reg\_t, 209
- ic\_hs\_spklen
  - qm\_i2c\_reg\_t, 209
- ic\_intr\_mask
  - qm\_i2c\_reg\_t, 209
- ic\_intr\_stat
  - qm\_i2c\_reg\_t, 210
- ic\_raw\_intr\_stat
  - qm\_i2c\_reg\_t, 210
- ic\_rx\_tl
  - qm\_i2c\_reg\_t, 210
- ic\_rxflr
  - qm\_i2c\_reg\_t, 210
- ic\_sar
  - qm\_i2c\_reg\_t, 210
- ic\_sda\_hold
  - qm\_i2c\_reg\_t, 210
- ic\_sda\_setup
  - qm\_i2c\_reg\_t, 210
- ic\_ss\_scl\_hcnt
  - qm\_i2c\_reg\_t, 210
- ic\_ss\_scl\_lcnt
  - qm\_i2c\_reg\_t, 210
- ic\_status
  - qm\_i2c\_reg\_t, 211
- ic\_tar
  - qm\_i2c\_reg\_t, 211
- ic\_tx\_abrt\_source
  - qm\_i2c\_reg\_t, 211
- ic\_tx\_tl
  - qm\_i2c\_reg\_t, 211
- ic\_txflr
  - qm\_i2c\_reg\_t, 211
- icr
  - qm\_mvic\_reg\_t, 217
  - qm\_spi\_reg\_t, 245
- id
  - qm\_scss\_info\_reg\_t, 232
- Identification, 55
  - qm\_soc\_id, 55
  - qm\_soc\_version, 55
- idr
  - qm\_spi\_reg\_t, 245
- ier\_dlh
  - qm\_uart\_reg\_t, 261
- iir\_fcr
  - qm\_uart\_reg\_t, 261
- imr
  - qm\_spi\_reg\_t, 245
- inc\_run\_only
  - qm\_ss\_timer\_config\_t, 258
- init\_val
  - qm\_rtc\_config\_t, 224
- Initialisation, 56
  - QM\_COLD\_RESET, 56
  - QM\_WARM\_RESET, 56
  - qm\_soc\_reset, 56
  - qm\_soc\_reset\_t, 56
- int\_adc\_calib\_mask
  - qm\_scss\_int\_reg\_t, 234
- int\_adc\_pwr\_mask
  - qm\_scss\_int\_reg\_t, 234
- int\_aon\_timer\_mask
  - qm\_scss\_int\_reg\_t, 234

- int\_bothedge
  - qm\_gpio\_port\_config\_t, 200
- int\_comparators\_host\_halt\_mask
  - qm\_scss\_int\_reg\_t, 234
- int\_comparators\_host\_mask
  - qm\_scss\_int\_reg\_t, 234
- int\_debounce
  - qm\_gpio\_port\_config\_t, 200
  - qm\_ss\_gpio\_port\_config\_t, 251
- int\_dma\_channel\_0\_mask
  - qm\_scss\_int\_reg\_t, 234
- int\_dma\_channel\_1\_mask
  - qm\_scss\_int\_reg\_t, 234
- int\_dma\_error\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_en
  - qm\_ac\_config\_t, 187
  - qm\_aonpt\_config\_t, 191
  - qm\_gpio\_port\_config\_t, 201
  - qm\_pic\_timer\_config\_t, 218
  - qm\_ss\_gpio\_port\_config\_t, 251
  - qm\_ss\_timer\_config\_t, 258
  - qm\_uart\_config\_t, 259
- int\_flash\_controller\_0\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_flash\_controller\_1\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_gpio\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_host\_bus\_err\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_i2c\_mst\_0\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_polarity
  - qm\_gpio\_port\_config\_t, 201
  - qm\_ss\_gpio\_port\_config\_t, 251
- int\_rtc\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_spi\_mst\_0\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_spi\_slv\_0\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_sram\_controller\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_ss\_i2c\_reg\_t, 186
- int\_ss\_spi\_reg\_t, 186
- int\_timer\_mask
  - qm\_scss\_int\_reg\_t, 235
- int\_type
  - qm\_gpio\_port\_config\_t, 201
  - qm\_ss\_gpio\_port\_config\_t, 251
- int\_uart\_0\_mask
  - qm\_scss\_int\_reg\_t, 236
- int\_uart\_1\_mask
  - qm\_scss\_int\_reg\_t, 236
- int\_watchdog\_mask
  - qm\_scss\_int\_reg\_t, 236
- Interrupt, 57
  - qm\_int\_vector\_request, 57
  - qm\_irq\_mask, 57
  - qm\_irq\_unmask, 57
- intstatus
  - qm\_pwm\_channel\_t, 219
- irr
  - qm\_mvic\_reg\_t, 217
- isr
  - qm\_mvic\_reg\_t, 217
  - qm\_spi\_reg\_t, 245
- lcr
  - qm\_uart\_reg\_t, 261
- lcr\_ext
  - qm\_uart\_reg\_t, 261
- line\_control
  - qm\_uart\_config\_t, 259
- lo\_count
  - qm\_pwm\_config\_t, 221
- loadcount
  - qm\_pwm\_channel\_t, 219
- lock\_int\_mask\_reg
  - qm\_scss\_int\_reg\_t, 236
- low\_bound
  - qm\_fpr\_config\_t, 199
- lsr
  - qm\_uart\_reg\_t, 261
- lvtimer
  - qm\_mvic\_reg\_t, 217
- MPR, 72
  - qm\_mpr\_set\_config, 72
  - qm\_mpr\_set\_violation\_policy, 72
- Mailbox, 67
  - QM\_MBOX\_CH\_0, 68
  - QM\_MBOX\_CH\_1, 68
  - QM\_MBOX\_CH\_2, 68
  - QM\_MBOX\_CH\_3, 68
  - QM\_MBOX\_CH\_4, 68
  - QM\_MBOX\_CH\_5, 68
  - QM\_MBOX\_CH\_6, 68
  - QM\_MBOX\_CH\_7, 68
  - QM\_MBOX\_CH\_DATA, 68
  - QM\_MBOX\_CH\_IDLE, 68
  - QM\_MBOX\_CH\_INT, 68
  - QM\_MBOX\_CH\_NUM, 68
  - QM\_MBOX\_CH\_STATUS\_MASK, 68
  - QM\_MBOX\_PAYLOAD\_0, 68
  - QM\_MBOX\_PAYLOAD\_1, 68
  - QM\_MBOX\_PAYLOAD\_2, 68
  - QM\_MBOX\_PAYLOAD\_3, 68
  - QM\_MBOX\_PAYLOAD\_NUM, 68
  - qm\_mbox\_callback\_t, 67
  - qm\_mbox\_ch\_data\_ack, 69
  - qm\_mbox\_ch\_get\_status, 70
  - qm\_mbox\_ch\_read, 70
  - qm\_mbox\_ch\_set\_config, 70
  - qm\_mbox\_ch\_status\_t, 68
  - qm\_mbox\_ch\_t, 68

- qm\_mbox\_ch\_write, 71
- qm\_mbox\_payload\_t, 68
- mask\_interrupt
  - qm\_pwm\_config\_t, 221
- mcr
  - qm\_uart\_reg\_t, 261
- mode
  - qm\_i2c\_config\_t, 204
  - qm\_pic\_timer\_config\_t, 218
  - qm\_pwm\_config\_t, 221
  - qm\_wdt\_config\_t, 264
- msr
  - qm\_uart\_reg\_t, 261
- msticr
  - qm\_spi\_reg\_t, 245
- mvic\_reg\_pad\_t, 186
- mwcr
  - qm\_spi\_reg\_t, 246
- osc0\_cfg0
  - qm\_scss\_ccu\_reg\_t, 228
- osc0\_cfg1
  - qm\_scss\_ccu\_reg\_t, 228
- osc0\_stat1
  - qm\_scss\_ccu\_reg\_t, 228
- osc1\_cfg0
  - qm\_scss\_ccu\_reg\_t, 228
- osc1\_stat0
  - qm\_scss\_ccu\_reg\_t, 228
- osc\_lock\_0
  - qm\_scss\_ccu\_reg\_t, 229
- PIC Timer
  - QM\_PIC\_TIMER\_MODE\_ONE\_SHOT, 73
  - QM\_PIC\_TIMER\_MODE\_PERIODIC, 73
- POWER\_WAKE\_FROM\_GPIO\_COMP
  - Quark D2000 Power states, 153
- POWER\_WAKE\_FROM\_RTC
  - Quark D2000 Power states, 153
- PWM / Timer
  - QM\_PWM\_MODE\_PWM, 80
  - QM\_PWM\_MODE\_TIMER\_COUNT, 80
  - QM\_PWM\_MODE\_TIMER\_FREE\_RUNNING, 80
- p\_sts
  - qm\_scss\_pmu\_reg\_t, 238
- PIC Timer, 73
  - qm\_pic\_timer\_get, 73
  - qm\_pic\_timer\_mode\_t, 73
  - qm\_pic\_timer\_set, 74
  - qm\_pic\_timer\_set\_config, 74
- PWM / Timer, 80
  - qm\_pwm\_get, 80
  - qm\_pwm\_mode\_t, 80
  - qm\_pwm\_set, 81
  - qm\_pwm\_set\_config, 81
  - qm\_pwm\_start, 82
  - qm\_pwm\_stop, 82
- periph\_cfg0
  - qm\_scss\_peripheral\_reg\_t, 237
- pic\_timer\_reg\_pad\_t, 186
- pico\_printf
  - Syscalls, 185
- Pin Muxing setup, 75
  - QM\_PIN\_ID\_0, 76
  - QM\_PIN\_ID\_1, 76
  - QM\_PIN\_ID\_10, 76
  - QM\_PIN\_ID\_11, 76
  - QM\_PIN\_ID\_12, 76
  - QM\_PIN\_ID\_13, 76
  - QM\_PIN\_ID\_14, 76
  - QM\_PIN\_ID\_15, 76
  - QM\_PIN\_ID\_16, 76
  - QM\_PIN\_ID\_17, 76
  - QM\_PIN\_ID\_18, 76
  - QM\_PIN\_ID\_19, 76
  - QM\_PIN\_ID\_2, 76
  - QM\_PIN\_ID\_20, 76
  - QM\_PIN\_ID\_21, 76
  - QM\_PIN\_ID\_22, 76
  - QM\_PIN\_ID\_23, 76
  - QM\_PIN\_ID\_24, 76
  - QM\_PIN\_ID\_25, 76
  - QM\_PIN\_ID\_26, 76
  - QM\_PIN\_ID\_27, 76
  - QM\_PIN\_ID\_28, 76
  - QM\_PIN\_ID\_29, 76
  - QM\_PIN\_ID\_3, 76
  - QM\_PIN\_ID\_30, 76
  - QM\_PIN\_ID\_31, 76
  - QM\_PIN\_ID\_32, 76
  - QM\_PIN\_ID\_33, 76
  - QM\_PIN\_ID\_34, 76
  - QM\_PIN\_ID\_35, 76
  - QM\_PIN\_ID\_36, 76
  - QM\_PIN\_ID\_37, 76
  - QM\_PIN\_ID\_38, 76
  - QM\_PIN\_ID\_39, 76
  - QM\_PIN\_ID\_4, 76
  - QM\_PIN\_ID\_40, 76
  - QM\_PIN\_ID\_41, 76
  - QM\_PIN\_ID\_42, 76
  - QM\_PIN\_ID\_43, 76
  - QM\_PIN\_ID\_44, 76
  - QM\_PIN\_ID\_45, 77
  - QM\_PIN\_ID\_46, 77
  - QM\_PIN\_ID\_47, 77
  - QM\_PIN\_ID\_48, 77
  - QM\_PIN\_ID\_49, 77
  - QM\_PIN\_ID\_5, 76
  - QM\_PIN\_ID\_50, 77
  - QM\_PIN\_ID\_51, 77
  - QM\_PIN\_ID\_52, 77
  - QM\_PIN\_ID\_53, 77
  - QM\_PIN\_ID\_54, 77
  - QM\_PIN\_ID\_55, 77
  - QM\_PIN\_ID\_56, 77
  - QM\_PIN\_ID\_57, 77

- QM\_PIN\_ID\_58, [77](#)
- QM\_PIN\_ID\_59, [77](#)
- QM\_PIN\_ID\_6, [76](#)
- QM\_PIN\_ID\_60, [77](#)
- QM\_PIN\_ID\_61, [77](#)
- QM\_PIN\_ID\_62, [77](#)
- QM\_PIN\_ID\_63, [77](#)
- QM\_PIN\_ID\_64, [77](#)
- QM\_PIN\_ID\_65, [77](#)
- QM\_PIN\_ID\_66, [77](#)
- QM\_PIN\_ID\_67, [77](#)
- QM\_PIN\_ID\_68, [77](#)
- QM\_PIN\_ID\_7, [76](#)
- QM\_PIN\_ID\_8, [76](#)
- QM\_PIN\_ID\_9, [76](#)
- QM\_PMUX\_FN\_0, [77](#)
- QM\_PMUX\_FN\_1, [77](#)
- QM\_PMUX\_FN\_2, [77](#)
- QM\_PMUX\_FN\_3, [77](#)
- QM\_PMUX\_SLEW\_12MA, [77](#)
- QM\_PMUX\_SLEW\_16MA, [77](#)
- QM\_PMUX\_SLEW\_2MA, [77](#)
- QM\_PMUX\_SLEW\_4MA, [77](#)
- QM\_PMUX\_SLEW\_NUM, [77](#)
- qm\_pin\_id\_t, [75](#)
- qm\_pmux\_fn\_t, [77](#)
- qm\_pmux\_input\_en, [78](#)
- qm\_pmux\_pullup\_en, [78](#)
- qm\_pmux\_select, [78](#)
- qm\_pmux\_set\_slew, [79](#)
- qm\_pmux\_slew\_t, [77](#)
- pm\_lock
  - qm\_scss\_pmu\_reg\_t, [238](#)
- pm\_wait
  - qm\_scss\_pmu\_reg\_t, [238](#)
- pmux\_in\_en
  - qm\_scss\_pmux\_reg\_t, [239](#)
- pmux\_in\_en\_lock
  - qm\_scss\_pmux\_reg\_t, [239](#)
- pmux\_pullup
  - qm\_scss\_pmux\_reg\_t, [240](#)
- pmux\_pullup\_lock
  - qm\_scss\_pmux\_reg\_t, [240](#)
- pmux\_sel
  - qm\_scss\_pmux\_reg\_t, [240](#)
- pmux\_sel\_0\_lock
  - qm\_scss\_pmux\_reg\_t, [240](#)
- pmux\_slew
  - qm\_scss\_pmux\_reg\_t, [240](#)
- pmux\_slew\_lock
  - qm\_scss\_pmux\_reg\_t, [240](#)
- power\_cpu\_c1
  - Quark SE Host Power states, [169](#)
- power\_cpu\_c2
  - Quark SE Host Power states, [169](#)
- power\_cpu\_c2lp
  - Quark SE Host Power states, [170](#)
- power\_cpu\_halt
  - Quark D2000 Power states, [153](#)
- power\_soc\_deep\_sleep
  - Quark D2000 Power states, [153](#)
  - Quark SE SoC Power states, [167](#)
- power\_soc\_sleep
  - Quark D2000 Power states, [154](#)
  - Quark SE SoC Power states, [167](#)
- power\_wake\_event\_t
  - Quark D2000 Power states, [153](#)
- ppr
  - qm\_mvic\_reg\_t, [217](#)
- QM\_ADC\_CAL\_COMPLETE
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_0
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_1
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_10
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_11
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_12
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_13
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_14
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_15
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_16
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_17
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_CH\_18
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_CH\_2
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_3
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_4
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_5
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_6
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_7
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_8
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_CH\_9
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_COMPLETE
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_IDLE
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_MODE\_CHANGED
  - Quark D2000 ADC, [17](#)
- QM\_ADC\_MODE\_DEEP\_PWR\_DOWN

- Quark D2000 ADC, [18](#)
- QM\_ADC\_MODE\_NORM\_CAL
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_MODE\_NORM\_NO\_CAL
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_MODE\_PWR\_DOWN
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_MODE\_STDBY
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_OVERFLOW
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_RES\_10\_BITS
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_RES\_12\_BITS
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_RES\_6\_BITS
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_RES\_8\_BITS
  - Quark D2000 ADC, [18](#)
- QM\_ADC\_TRANSFER
  - Quark D2000 ADC, [17](#)
- QM\_COLD\_RESET
  - Initialisation, [56](#)
- QM\_DMA\_0
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_1
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_128
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_16
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_256
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_32
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_4
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_64
  - DMA, [29](#)
- QM\_DMA\_BURST\_TRANS\_LENGTH\_8
  - DMA, [29](#)
- QM\_DMA\_CHANNEL\_0
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [179](#)
- QM\_DMA\_CHANNEL\_1
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [179](#), [180](#)
- QM\_DMA\_CHANNEL\_2
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- QM\_DMA\_CHANNEL\_3
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- QM\_DMA\_CHANNEL\_4
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- QM\_DMA\_CHANNEL\_5
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- QM\_DMA\_CHANNEL\_6
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- QM\_DMA\_CHANNEL\_7
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- QM\_DMA\_CHANNEL\_NUM
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [179](#), [180](#)
- QM\_DMA\_HANDSHAKE\_POLARITY\_HIGH
  - DMA, [29](#)
- QM\_DMA\_HANDSHAKE\_POLARITY\_LOW
  - DMA, [29](#)
- QM\_DMA\_MEMORY\_TO\_MEMORY
  - DMA, [29](#)
- QM\_DMA\_MEMORY\_TO\_PERIPHERAL
  - DMA, [29](#)
- QM\_DMA\_NUM
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- QM\_DMA\_PERIPHERAL\_TO\_MEMORY
  - DMA, [29](#)
- QM\_DMA\_TRANS\_WIDTH\_128
  - DMA, [29](#)
- QM\_DMA\_TRANS\_WIDTH\_16
  - DMA, [29](#)
- QM\_DMA\_TRANS\_WIDTH\_256
  - DMA, [29](#)
- QM\_DMA\_TRANS\_WIDTH\_32
  - DMA, [29](#)
- QM\_DMA\_TRANS\_WIDTH\_64
  - DMA, [29](#)
- QM\_DMA\_TRANS\_WIDTH\_8
  - DMA, [29](#)
- QM\_FLASH\_REGION\_DATA
  - Flash, [35](#)
- QM\_FLASH\_REGION\_NUM
  - Flash, [35](#)
- QM\_FLASH\_REGION\_OTP
  - Flash, [35](#)
- QM\_FLASH\_REGION\_SYS
  - Flash, [35](#)
- QM\_FLASH\_WRITE\_DISABLE
  - Flash, [34](#)
- QM\_FLASH\_WRITE\_ENABLE
  - Flash, [34](#)
- QM\_FPR\_0
  - FPR, [40](#)
- QM\_FPR\_1
  - FPR, [40](#)
- QM\_FPR\_2
  - FPR, [40](#)
- QM\_FPR\_3
  - FPR, [40](#)
- QM\_FPR\_DISABLE
  - FPR, [40](#)

QM\_FPR\_DMA  
     FPR, 40  
 QM\_FPR\_ENABLE  
     FPR, 40  
 QM\_FPR\_HOST\_PROCESSOR  
     FPR, 40  
 QM\_FPR\_LOCK\_DISABLE  
     FPR, 40  
 QM\_FPR\_LOCK\_ENABLE  
     FPR, 40  
 QM\_FPR\_OTHER\_AGENTS  
     FPR, 40  
 QM\_FPR\_SENSOR\_SUBSYSTEM  
     FPR, 40  
 QM\_GPIO\_HIGH  
     GPIO, 42  
 QM\_GPIO\_LOW  
     GPIO, 42  
 QM\_GPIO\_STATE\_NUM  
     GPIO, 42  
 QM\_I2C\_10\_BIT  
     I2C, 47  
 QM\_I2C\_7\_BIT  
     I2C, 47  
 QM\_I2C\_BUSY  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_IDLE  
     I2C, 47  
     SS I2C, 126  
 QM\_I2C\_MASTER  
     I2C, 47  
 QM\_I2C\_SLAVE  
     I2C, 47  
 QM\_I2C\_SPEED\_FAST  
     I2C, 47  
 QM\_I2C\_SPEED\_FAST\_PLUS  
     I2C, 47  
 QM\_I2C\_SPEED\_STD  
     I2C, 47  
 QM\_I2C\_TX\_ABRT\_10ADDR1\_NOACK  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_10ADDR2\_NOACK  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_10B\_RD\_NORSTRT  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_7B\_ADDR\_NOACK  
     I2C, 47  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_GCALL\_NOACK  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_GCALL\_READ  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_HS\_ACKDET  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_HS\_NORSTRT  
     I2C, 48  
 QM\_I2C\_TX\_ABRT\_MASTER\_DIS  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_SBYTE\_ACKDET  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_SLV\_ARBLOST  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_SLVFLUSH\_TXFIFO  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_SLVRD\_INTX  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_TXDATA\_NOACK  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ABRT\_USER\_ABRT  
     I2C, 48  
     SS I2C, 126  
 QM\_I2C\_TX\_ARB\_LOST  
     I2C, 48  
     SS I2C, 126  
 QM\_MAIN\_FLASH\_DATA  
     FPR, 39  
 QM\_MAIN\_FLASH\_NUM  
     FPR, 39  
 QM\_MAIN\_FLASH\_SYSTEM  
     FPR, 39  
 QM\_MBOX\_CH\_0  
     Mailbox, 68  
 QM\_MBOX\_CH\_1  
     Mailbox, 68  
 QM\_MBOX\_CH\_2  
     Mailbox, 68  
 QM\_MBOX\_CH\_3  
     Mailbox, 68  
 QM\_MBOX\_CH\_4  
     Mailbox, 68  
 QM\_MBOX\_CH\_5  
     Mailbox, 68  
 QM\_MBOX\_CH\_6  
     Mailbox, 68  
 QM\_MBOX\_CH\_7  
     Mailbox, 68  
 QM\_MBOX\_CH\_DATA  
     Mailbox, 68  
 QM\_MBOX\_CH\_IDLE  
     Mailbox, 68  
 QM\_MBOX\_CH\_INT  
     Mailbox, 68  
 QM\_MBOX\_CH\_NUM  
     Mailbox, 68  
 QM\_MBOX\_CH\_STATUS\_MASK  
     Mailbox, 68  
 QM\_MBOX\_PAYLOAD\_0  
     Mailbox, 68  
 QM\_MBOX\_PAYLOAD\_1



- Mailbox, [68](#)
- QM\_MBOX\_PAYLOAD\_2
  - Mailbox, [68](#)
- QM\_MBOX\_PAYLOAD\_3
  - Mailbox, [68](#)
- QM\_MBOX\_PAYLOAD\_NUM
  - Mailbox, [68](#)
- QM\_PIC\_TIMER\_MODE\_ONE\_SHOT
  - PIC Timer, [73](#)
- QM\_PIC\_TIMER\_MODE\_PERIODIC
  - PIC Timer, [73](#)
- QM\_PIN\_ID\_0
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_1
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_10
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_11
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_12
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_13
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_14
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_15
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_16
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_17
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_18
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_19
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_2
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_20
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_21
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_22
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_23
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_24
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_25
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_26
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_27
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_28
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_29
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_3
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_30
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_31
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_32
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_33
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_34
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_35
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_36
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_37
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_38
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_39
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_4
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_40
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_41
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_42
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_43
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_44
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_45
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_46
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_47
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_48
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_49
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_5
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_50
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_51
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_52
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_53
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_54
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_55
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_56
  - Pin Muxing setup, [76](#)

- Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_57
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_58
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_59
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_6
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_60
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_61
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_62
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_63
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_64
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_65
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_66
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_67
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_68
  - Pin Muxing setup, [77](#)
- QM\_PIN\_ID\_7
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_8
  - Pin Muxing setup, [76](#)
- QM\_PIN\_ID\_9
  - Pin Muxing setup, [76](#)
- QM\_PMUX\_FN\_0
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_FN\_1
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_FN\_2
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_FN\_3
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_SLEW\_12MA
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_SLEW\_16MA
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_SLEW\_2MA
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_SLEW\_4MA
  - Pin Muxing setup, [77](#)
- QM\_PMUX\_SLEW\_NUM
  - Pin Muxing setup, [77](#)
- QM\_PWM\_MODE\_PWM
  - PWM / Timer, [80](#)
- QM\_PWM\_MODE\_TIMER\_COUNT
  - PWM / Timer, [80](#)
- QM\_PWM\_MODE\_TIMER\_FREE\_RUNNING
  - PWM / Timer, [80](#)
- QM\_SPI\_BMODE\_0
  - SPI, [87](#)
- QM\_SPI\_BMODE\_1
  - SPI, [87](#)
- QM\_SPI\_BMODE\_2
  - SPI, [87](#)
- QM\_SPI\_BMODE\_3
  - SPI, [87](#)
- QM\_SPI\_BUSY
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_10\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_11\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_12\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_13\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_14\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_15\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_16\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_17\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_18\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_19\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_20\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_21\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_22\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_23\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_24\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_25\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_26\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_27\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_28\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_29\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_30\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_31\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_32\_BIT
  - SPI, [88](#)
- QM\_SPI\_FRAME\_SIZE\_4\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_5\_BIT

- SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_6\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_7\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_8\_BIT
  - SPI, [87](#)
- QM\_SPI\_FRAME\_SIZE\_9\_BIT
  - SPI, [87](#)
- QM\_SPI\_IDLE
  - SPI, [88](#)
- QM\_SPI\_RX\_OVERFLOW
  - SPI, [88](#)
- QM\_SPI\_SS\_0
  - SPI, [88](#)
- QM\_SPI\_SS\_1
  - SPI, [88](#)
- QM\_SPI\_SS\_2
  - SPI, [88](#)
- QM\_SPI\_SS\_3
  - SPI, [88](#)
- QM\_SPI\_SS\_DISABLED
  - SPI, [88](#)
- QM\_SPI\_TMOD\_EEPROM\_READ
  - SPI, [89](#)
- QM\_SPI\_TMOD\_RX
  - SPI, [89](#)
- QM\_SPI\_TMOD\_TX
  - SPI, [89](#)
- QM\_SPI\_TMOD\_TX\_RX
  - SPI, [89](#)
- QM\_SS\_ADC\_0
  - SoC Registers (Sensor Subsystem), [172](#)
- 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, [113](#)
- QM\_SS\_ADC\_CH\_11
  - SS ADC, [113](#)
- QM\_SS\_ADC\_CH\_12
  - SS ADC, [113](#)
- 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, [113](#)
- QM\_SS\_ADC\_COMPLETE
  - SS ADC, [114](#)
- QM\_SS\_ADC\_CTRL
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_ADC\_DIVSEQSTAT
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_ADC\_IDLE
  - SS ADC, [114](#)
- QM\_SS\_ADC\_INTSTAT
  - SoC Registers (Sensor Subsystem), [172](#)
- 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), [172](#)
- QM\_SS\_ADC\_SEQ
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_ADC\_SEQERROR
  - SS ADC, [114](#)
- QM\_SS\_ADC\_SET
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_ADC\_TRANSFER
  - SS ADC, [113](#)
- QM\_SS\_ADC\_UNDERFLOW

- SS ADC, [114](#)
- QM\_SS\_GPIO\_HIGH
  - SS GPIO, [119](#)
- QM\_SS\_GPIO\_LOW
  - SS GPIO, [119](#)
- QM\_SS\_I2C\_10\_BIT
  - SS I2C, [126](#)
- QM\_SS\_I2C\_7\_BIT
  - SS I2C, [126](#)
- QM\_SS\_I2C\_SPEED\_FAST
  - SS I2C, [126](#)
- QM\_SS\_I2C\_SPEED\_STD
  - SS I2C, [126](#)
- QM\_SS\_IO\_CREG\_MST0\_CTRL
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_IO\_CREG\_SLV0\_OBSR
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_IO\_CREG\_SLV1\_OBSR
  - SoC Registers (Sensor Subsystem), [172](#)
- QM\_SS\_SPI\_0
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_1
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_BMODE\_0
  - SS SPI, [137](#)
- QM\_SS\_SPI\_BMODE\_1
  - SS SPI, [137](#)
- QM\_SS\_SPI\_BMODE\_2
  - SS SPI, [137](#)
- QM\_SS\_SPI\_BMODE\_3
  - SS SPI, [137](#)
- QM\_SS\_SPI\_BUSY
  - SS SPI, [138](#)
- QM\_SS\_SPI\_CLR\_INTR
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_CTRL
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_DR
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_10\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_11\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_12\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_13\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_14\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_15\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_16\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_4\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_5\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_6\_BIT
  - SS SPI, [137](#)
- SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_7\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_8\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FRAME\_SIZE\_9\_BIT
  - SS SPI, [137](#)
- QM\_SS\_SPI\_FTLR
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_IDLE
  - SS SPI, [138](#)
- QM\_SS\_SPI\_INTR\_MASK
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_INTR\_STAT
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_RX\_OVERFLOW
  - SS SPI, [138](#)
- QM\_SS\_SPI\_RXFLR
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_SPIEN
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_SR
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_SS\_0
  - SS SPI, [137](#)
- QM\_SS\_SPI\_SS\_1
  - SS SPI, [137](#)
- QM\_SS\_SPI\_SS\_2
  - SS SPI, [137](#)
- QM\_SS\_SPI\_SS\_3
  - SS SPI, [137](#)
- QM\_SS\_SPI\_SS\_DISABLED
  - SS SPI, [137](#)
- QM\_SS\_SPI\_TIMING
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_SS\_SPI\_TMOD\_EEPROM\_READ
  - SS SPI, [138](#)
- QM\_SS\_SPI\_TMOD\_RX
  - SS SPI, [138](#)
- QM\_SS\_SPI\_TMOD\_TX
  - SS SPI, [138](#)
- QM\_SS\_SPI\_TMOD\_TX\_RX
  - SS SPI, [138](#)
- QM\_SS\_SPI\_TXFLR
  - SoC Registers (Sensor Subsystem), [173](#)
- QM\_UART\_IDLE
  - UART, [96](#)
- QM\_UART\_LC\_5E1
  - UART, [95](#)
- QM\_UART\_LC\_5E1\_5
  - UART, [95](#)
- QM\_UART\_LC\_5N1
  - UART, [95](#)
- QM\_UART\_LC\_5N1\_5
  - UART, [95](#)
- QM\_UART\_LC\_5O1
  - UART, [95](#)
- QM\_UART\_LC\_5O1\_5

- UART, [95](#)
- QM\_UART\_LC\_6E1
  - UART, [95](#)
- QM\_UART\_LC\_6E2
  - UART, [95](#)
- QM\_UART\_LC\_6N1
  - UART, [95](#)
- QM\_UART\_LC\_6N2
  - UART, [95](#)
- QM\_UART\_LC\_6O1
  - UART, [95](#)
- QM\_UART\_LC\_6O2
  - UART, [95](#)
- QM\_UART\_LC\_7E1
  - UART, [95](#)
- QM\_UART\_LC\_7E2
  - UART, [95](#)
- QM\_UART\_LC\_7N1
  - UART, [95](#)
- QM\_UART\_LC\_7N2
  - UART, [95](#)
- QM\_UART\_LC\_7O1
  - UART, [95](#)
- QM\_UART\_LC\_7O2
  - UART, [95](#)
- QM\_UART\_LC\_8E1
  - UART, [96](#)
- QM\_UART\_LC\_8E2
  - UART, [96](#)
- QM\_UART\_LC\_8N1
  - UART, [95](#)
- QM\_UART\_LC\_8N2
  - UART, [95](#)
- QM\_UART\_LC\_8O1
  - UART, [96](#)
- QM\_UART\_LC\_8O2
  - UART, [96](#)
- QM\_UART\_RX\_BI
  - UART, [96](#)
- QM\_UART\_RX\_BUSY
  - UART, [96](#)
- QM\_UART\_RX\_FE
  - UART, [96](#)
- QM\_UART\_RX\_NEMPTY
  - UART, [96](#)
- QM\_UART\_RX\_OE
  - UART, [96](#)
- QM\_UART\_RX\_PE
  - UART, [96](#)
- QM\_UART\_TX\_BUSY
  - UART, [96](#)
- QM\_UART\_TX\_NFULL
  - UART, [96](#)
- QM\_WARM\_RESET
  - Initialisation, [56](#)
- QM\_WDT\_2\_POW\_16\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_17\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_18\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_19\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_20\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_21\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_22\_CYCLES
  - WDT, [105](#)
- QM\_WDT\_2\_POW\_23\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_24\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_25\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_26\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_27\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_28\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_29\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_30\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_2\_POW\_31\_CYCLES
  - WDT, [106](#)
- QM\_WDT\_MODE\_INTERRUPT\_RESET
  - WDT, [106](#)
- QM\_WDT\_MODE\_RESET
  - WDT, [106](#)
- QM\_ISR\_DECLARE
  - ISR, [60–65](#)
  - SS ISR, [132–135](#)
- qm\_ac\_config\_t, [186](#)
  - callback, [187](#)
  - callback\_data, [187](#)
  - int\_en, [187](#)
  - reference, [187](#)
- qm\_ac\_set\_config
  - Analog Comparator, [27](#)
- qm\_adc\_calibrate
  - Quark D2000 ADC, [18](#)
- qm\_adc\_cb\_source\_t
  - Quark D2000 ADC, [17](#)
- qm\_adc\_channel\_t
  - Quark D2000 ADC, [17](#)
- qm\_adc\_config\_t, [188](#)
  - resolution, [188](#)
- qm\_adc\_convert
  - Quark D2000 ADC, [19](#)
- qm\_adc\_get\_calibration
  - Quark D2000 ADC, [19](#)
- qm\_adc\_irq\_calibrate
  - Quark D2000 ADC, [19](#)
- qm\_adc\_irq\_convert

- Quark D2000 ADC, [20](#)
- qm\_adc\_irq\_set\_mode
  - Quark D2000 ADC, [20](#)
- qm\_adc\_mode\_t
  - Quark D2000 ADC, [18](#)
- qm\_adc\_reg\_t, [188](#)
- qm\_adc\_resolution\_t
  - Quark D2000 ADC, [18](#)
- qm\_adc\_set\_calibration
  - Quark D2000 ADC, [21](#)
- qm\_adc\_set\_config
  - Quark D2000 ADC, [21](#)
- qm\_adc\_set\_mode
  - Quark D2000 ADC, [21](#)
- qm\_adc\_status\_t
  - Quark D2000 ADC, [18](#)
- qm\_adc\_t
  - SoC Registers (D2000), [160](#)
- qm\_adc\_xfer\_t, [189](#)
  - callback, [189](#)
  - callback\_data, [190](#)
  - ch, [190](#)
  - ch\_len, [190](#)
  - samples, [190](#)
  - samples\_len, [190](#)
- qm\_aonc\_disable
  - Always-on Counters, [23](#)
- qm\_aonc\_enable
  - Always-on Counters, [24](#)
- qm\_aonc\_get\_value
  - Always-on Counters, [24](#)
- qm\_aonpt\_clear
  - Always-on Counters, [24](#)
- qm\_aonpt\_config\_t, [190](#)
  - callback, [191](#)
  - callback\_data, [191](#)
  - count, [191](#)
  - int\_en, [191](#)
- qm\_aonpt\_get\_status
  - Always-on Counters, [25](#)
- qm\_aonpt\_get\_value
  - Always-on Counters, [25](#)
- qm\_aonpt\_reset
  - Always-on Counters, [25](#)
- qm\_aonpt\_set\_config
  - Always-on Counters, [26](#)
- qm\_dma\_burst\_length\_t
  - DMA, [29](#)
- qm\_dma\_chan\_reg\_t, [191](#)
- qm\_dma\_channel\_config\_t, [192](#)
  - client\_callback, [193](#)
- qm\_dma\_channel\_direction\_t
  - DMA, [29](#)
- qm\_dma\_channel\_id\_t
  - SoC Registers (D2000), [160](#)
  - SoC Registers (SE), [179](#)
- qm\_dma\_channel\_set\_config
  - DMA, [30](#)
- qm\_dma\_handshake\_interface\_t
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- qm\_dma\_handshake\_polarity\_t
  - DMA, [29](#)
- qm\_dma\_init
  - DMA, [30](#)
- qm\_dma\_int\_reg\_t, [193](#)
- qm\_dma\_misc\_reg\_t, [195](#)
- qm\_dma\_t
  - SoC Registers (D2000), [161](#)
  - SoC Registers (SE), [180](#)
- qm\_dma\_transfer\_mem\_to\_mem
  - DMA, [31](#)
- qm\_dma\_transfer\_set\_config
  - DMA, [31](#)
- qm\_dma\_transfer\_start
  - DMA, [32](#)
- qm\_dma\_transfer\_t, [196](#)
  - block\_size, [196](#)
  - destination\_address, [196](#)
  - source\_address, [197](#)
- qm\_dma\_transfer\_terminate
  - DMA, [32](#)
- qm\_dma\_transfer\_width\_t
  - DMA, [29](#)
- qm\_flash\_config\_t, [197](#)
  - us\_count, [197](#)
  - wait\_states, [197](#)
  - write\_disable, [197](#)
- qm\_flash\_disable\_t
  - Flash, [34](#)
- qm\_flash\_mass\_erase
  - Flash, [35](#)
- qm\_flash\_page\_erase
  - Flash, [35](#)
- qm\_flash\_page\_update
  - Flash, [36](#)
- qm\_flash\_page\_write
  - Flash, [36](#)
- qm\_flash\_reg\_t, [198](#)
- qm\_flash\_region\_t
  - Flash, [34](#)
- qm\_flash\_region\_type\_t
  - FPR, [39](#)
- qm\_flash\_set\_config
  - Flash, [37](#)
- qm\_flash\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_flash\_word\_write
  - Flash, [37](#)
- qm\_fpr\_config\_t, [198](#)
  - allow\_agents, [199](#)
  - en\_mask, [199](#)
  - low\_bound, [199](#)
  - up\_bound, [199](#)
- qm\_fpr\_en\_t

- FPR, [39](#)
- qm\_fpr\_id\_t
  - FPR, [40](#)
- qm\_fpr\_read\_allow\_t
  - FPR, [40](#)
- qm\_fpr\_set\_config
  - FPR, [40](#)
- qm\_fpr\_set\_violation\_policy
  - FPR, [41](#)
- qm\_fpr\_viol\_mode\_t
  - FPR, [40](#)
- qm\_gpio\_clear\_pin
  - GPIO, [42](#)
- qm\_gpio\_port\_config\_t, [199](#)
  - callback, [200](#)
  - callback\_data, [200](#)
  - direction, [200](#)
  - int\_bothedge, [200](#)
  - int\_debounce, [200](#)
  - int\_en, [201](#)
  - int\_polarity, [201](#)
  - int\_type, [201](#)
- qm\_gpio\_read\_pin
  - GPIO, [43](#)
- qm\_gpio\_read\_port
  - GPIO, [43](#)
- qm\_gpio\_reg\_t, [201](#)
  - gpio\_config\_reg1, [202](#)
  - gpio\_config\_reg2, [202](#)
  - gpio\_debounce, [202](#)
  - gpio\_ext\_porta, [202](#)
  - gpio\_id\_code, [202](#)
  - gpio\_int\_bothedge, [202](#)
  - gpio\_int\_polarity, [202](#)
  - gpio\_inten, [203](#)
  - gpio\_intmask, [203](#)
  - gpio\_intstatus, [203](#)
  - gpio\_inttype\_level, [203](#)
  - gpio\_ls\_sync, [203](#)
  - gpio\_porta\_eoi, [203](#)
  - gpio\_raw\_intstatus, [203](#)
  - gpio\_swporta\_ddr, [203](#)
  - gpio\_swporta\_dr, [203](#)
  - gpio\_ver\_id\_code, [203](#)
- qm\_gpio\_set\_config
  - GPIO, [43](#)
- qm\_gpio\_set\_pin
  - GPIO, [44](#)
- qm\_gpio\_set\_pin\_state
  - GPIO, [44](#)
- qm\_gpio\_state\_t
  - GPIO, [42](#)
- qm\_gpio\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_gpio\_write\_port
  - GPIO, [45](#)
- qm\_i2c
  - SoC Registers (D2000), [163](#)
  - SoC Registers (SE), [182](#)
- qm\_i2c\_addr\_t
  - I2C, [47](#)
- qm\_i2c\_config\_t, [204](#)
  - address\_mode, [204](#)
  - mode, [204](#)
  - slave\_addr, [204](#)
  - speed, [204](#)
- qm\_i2c\_dma\_channel\_config
  - I2C, [48](#)
- qm\_i2c\_dma\_transfer\_terminate
  - I2C, [49](#)
- qm\_i2c\_get\_status
  - I2C, [49](#)
- qm\_i2c\_irq\_transfer\_terminate
  - I2C, [49](#)
- qm\_i2c\_master\_dma\_transfer
  - I2C, [50](#)
- qm\_i2c\_master\_irq\_transfer
  - I2C, [50](#)
- qm\_i2c\_master\_read
  - I2C, [51](#)
- qm\_i2c\_master\_write
  - I2C, [51](#)
- qm\_i2c\_mode\_t
  - I2C, [47](#)
- qm\_i2c\_reg\_t, [205](#)
  - ic\_ack\_general\_call, [207](#)
  - ic\_clr\_activity, [207](#)
  - ic\_clr\_gen\_call, [207](#)
  - ic\_clr\_intr, [207](#)
  - ic\_clr\_rd\_req, [207](#)
  - ic\_clr\_restart\_det, [207](#)
  - ic\_clr\_rx\_done, [207](#)
  - ic\_clr\_rx\_over, [207](#)
  - ic\_clr\_rx\_under, [207](#)
  - ic\_clr\_start\_det, [207](#)
  - ic\_clr\_stop\_det, [208](#)
  - ic\_clr\_tx\_abrt, [208](#)
  - ic\_clr\_tx\_over, [208](#)
  - ic\_comp\_param\_1, [208](#)
  - ic\_comp\_type, [208](#)
  - ic\_comp\_version, [208](#)
  - ic\_con, [208](#)
  - ic\_data\_cmd, [208](#)
  - ic\_dma\_cr, [208](#)
  - ic\_dma\_rdlr, [208](#)
  - ic\_dma\_tdlr, [208](#)
  - ic\_enable, [209](#)
  - ic\_enable\_status, [209](#)
  - ic\_fs\_scl\_hcnt, [209](#)
  - ic\_fs\_scl\_lcnt, [209](#)
  - ic\_fs\_spklen, [209](#)
  - ic\_hs\_maddr, [209](#)
  - ic\_hs\_scl\_hcnt, [209](#)
  - ic\_hs\_scl\_lcnt, [209](#)
  - ic\_hs\_spklen, [209](#)

- ic\_intr\_mask, [209](#)
- ic\_intr\_stat, [210](#)
- ic\_raw\_intr\_stat, [210](#)
- ic\_rx\_tl, [210](#)
- ic\_rxflr, [210](#)
- ic\_sar, [210](#)
- ic\_sda\_hold, [210](#)
- ic\_sda\_setup, [210](#)
- ic\_ss\_scl\_hcnt, [210](#)
- ic\_ss\_scl\_lcnt, [210](#)
- ic\_status, [211](#)
- ic\_tar, [211](#)
- ic\_tx\_abrt\_source, [211](#)
- ic\_tx\_tl, [211](#)
- ic\_txflr, [211](#)
- qm\_i2c\_set\_config
  - I2C, [53](#)
- qm\_i2c\_set\_speed
  - I2C, [53](#)
- qm\_i2c\_speed\_t
  - I2C, [47](#)
- qm\_i2c\_status\_t
  - I2C, [47](#)
- qm\_i2c\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_i2c\_transfer\_t, [211](#)
  - callback, [212](#)
  - callback\_data, [212](#)
  - rx, [212](#)
  - rx\_len, [212](#)
  - stop, [212](#)
  - tx, [212](#)
  - tx\_len, [212](#)
- qm\_int\_vector\_request
  - Interrupt, [57](#)
- qm\_irq\_mask
  - Interrupt, [57](#)
- qm\_irq\_unmask
  - Interrupt, [57](#)
- qm\_lapic\_reg\_t, [213](#)
- qm\_mailbox\_t, [214](#)
- qm\_mbox\_callback\_t
  - Mailbox, [67](#)
- qm\_mbox\_ch\_data\_ack
  - Mailbox, [69](#)
- qm\_mbox\_ch\_get\_status
  - Mailbox, [70](#)
- qm\_mbox\_ch\_read
  - Mailbox, [70](#)
- qm\_mbox\_ch\_set\_config
  - Mailbox, [70](#)
- qm\_mbox\_ch\_status\_t
  - Mailbox, [68](#)
- qm\_mbox\_ch\_t
  - Mailbox, [68](#)
- qm\_mbox\_ch\_write
  - Mailbox, [71](#)
- qm\_mbox\_msg\_t, [214](#)
  - ctrl, [215](#)
  - data, [215](#)
- qm\_mbox\_payload\_t
  - Mailbox, [68](#)
- qm\_mpr\_config\_t, [215](#)
- qm\_mpr\_reg\_t, [215](#)
- qm\_mpr\_set\_config
  - MPR, [72](#)
- qm\_mpr\_set\_violation\_policy
  - MPR, [72](#)
- qm\_mvic\_reg\_t, [216](#)
  - ccr, [216](#)
  - eoi, [216](#)
  - icr, [217](#)
  - irr, [217](#)
  - isr, [217](#)
  - lvtimer, [217](#)
  - ppr, [217](#)
  - sivr, [217](#)
  - tpr, [217](#)
- qm\_pic\_timer\_config\_t, [217](#)
  - callback, [218](#)
  - callback\_data, [218](#)
  - int\_en, [218](#)
  - mode, [218](#)
- qm\_pic\_timer\_get
  - PIC Timer, [73](#)
- qm\_pic\_timer\_mode\_t
  - PIC Timer, [73](#)
- qm\_pic\_timer\_reg\_t, [218](#)
- qm\_pic\_timer\_set
  - PIC Timer, [74](#)
- qm\_pic\_timer\_set\_config
  - PIC Timer, [74](#)
- qm\_pin\_id\_t
  - Pin Muxing setup, [75](#)
- qm\_pmux\_fn\_t
  - Pin Muxing setup, [77](#)
- qm\_pmux\_input\_en
  - Pin Muxing setup, [78](#)
- qm\_pmux\_pullup\_en
  - Pin Muxing setup, [78](#)
- qm\_pmux\_select
  - Pin Muxing setup, [78](#)
- qm\_pmux\_set\_slew
  - Pin Muxing setup, [79](#)
- qm\_pmux\_slew\_t
  - Pin Muxing setup, [77](#)
- qm\_pwm\_channel\_t, [219](#)
  - controlreg, [219](#)
  - currentvalue, [219](#)
  - eoi, [219](#)
  - intstatus, [219](#)
  - loadcount, [219](#)
- qm\_pwm\_config\_t, [220](#)
  - callback, [220](#)
  - callback\_data, [220](#)



- hi\_count, [220](#)
  - lo\_count, [221](#)
  - mask\_interrupt, [221](#)
  - mode, [221](#)
- qm\_pwm\_get
  - PWM / Timer, [80](#)
- qm\_pwm\_id\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_pwm\_mode\_t
  - PWM / Timer, [80](#)
- qm\_pwm\_reg\_t, [221](#)
  - timer, [222](#)
- qm\_pwm\_set
  - PWM / Timer, [81](#)
- qm\_pwm\_set\_config
  - PWM / Timer, [81](#)
- qm\_pwm\_start
  - PWM / Timer, [82](#)
- qm\_pwm\_stop
  - PWM / Timer, [82](#)
- qm\_pwm\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_rtc\_config\_t, [222](#)
  - alarm\_en, [222](#)
  - alarm\_val, [222](#)
  - callback, [222](#)
  - callback\_data, [224](#)
  - init\_val, [224](#)
- qm\_rtc\_reg\_t, [224](#)
  - rtc\_ccr, [225](#)
  - rtc\_ccvr, [225](#)
  - rtc\_clr, [225](#)
  - rtc\_cmr, [225](#)
  - rtc\_comp\_version, [225](#)
  - rtc\_eoi, [225](#)
  - rtc\_rstat, [225](#)
  - rtc\_stat, [225](#)
- qm\_rtc\_set\_alarm
  - RTC, [83](#)
- qm\_rtc\_set\_config
  - RTC, [83](#)
- qm\_rtc\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_scss\_aon\_reg\_t, [225](#)
  - aonc\_cfg, [226](#)
  - aonc\_cnt, [226](#)
  - aonpt\_cfg, [226](#)
  - aonpt\_cnt, [226](#)
  - aonpt\_ctrl, [226](#)
  - aonpt\_stat, [226](#)
- qm\_scss\_aon\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_scss\_ccu\_reg\_t, [226](#)
  - ccu\_ext\_clock\_ctl, [227](#)
  - ccu\_gpio\_db\_clk\_ctl, [227](#)
  - ccu\_lp\_clk\_ctl, [227](#)
  - ccu\_mlayer\_ahb\_ctl, [228](#)
  - ccu\_periph\_clk\_div\_ctl0, [228](#)
  - ccu\_periph\_clk\_gate\_ctl, [228](#)
  - ccu\_ss\_periph\_clk\_gate\_ctl, [228](#)
  - ccu\_sys\_clk\_ctl, [228](#)
  - osc0\_cfg0, [228](#)
  - osc0\_cfg1, [228](#)
  - osc0\_stat1, [228](#)
  - osc1\_cfg0, [228](#)
  - osc1\_stat0, [228](#)
  - osc\_lock\_0, [229](#)
  - soc\_ctrl, [229](#)
  - soc\_ctrl\_lock, [229](#)
  - usb\_pll\_cfg0, [229](#)
  - wake\_mask, [229](#)
- qm\_scss\_cmp\_reg\_t, [229](#)
  - cmp\_en, [230](#)
  - cmp\_pwr, [230](#)
  - cmp\_ref\_pol, [230](#)
  - cmp\_ref\_sel, [230](#)
  - cmp\_stat\_clr, [230](#)
- qm\_scss\_gp\_reg\_t, [230](#)
  - gp0, [231](#)
  - gp1, [231](#)
  - gp2, [231](#)
  - gp3, [231](#)
  - gps0, [231](#)
  - gps1, [231](#)
  - gps2, [231](#)
  - gps3, [231](#)
  - wo\_sp, [231](#)
  - wo\_st, [232](#)
- qm\_scss\_info\_reg\_t, [232](#)
  - cotps, [232](#)
  - dotps, [232](#)
  - fs, [232](#)
  - id, [232](#)
  - rev, [233](#)
  - rs, [233](#)
- qm\_scss\_int\_reg\_t, [233](#)
  - int\_adc\_calib\_mask, [234](#)
  - int\_adc\_pwr\_mask, [234](#)
  - int\_aon\_timer\_mask, [234](#)
  - int\_comparators\_host\_halt\_mask, [234](#)
  - int\_comparators\_host\_mask, [234](#)
  - int\_dma\_channel\_0\_mask, [234](#)
  - int\_dma\_channel\_1\_mask, [234](#)
  - int\_dma\_error\_mask, [235](#)
  - int\_flash\_controller\_0\_mask, [235](#)
  - int\_flash\_controller\_1\_mask, [235](#)
  - int\_gpio\_mask, [235](#)
  - int\_host\_bus\_err\_mask, [235](#)
  - int\_i2c\_mst\_0\_mask, [235](#)
  - int\_rtc\_mask, [235](#)
  - int\_spi\_mst\_0\_mask, [235](#)
  - int\_spi\_slv\_0\_mask, [235](#)

- int\_sram\_controller\_mask, [235](#)
- int\_timer\_mask, [235](#)
- int\_uart\_0\_mask, [236](#)
- int\_uart\_1\_mask, [236](#)
- int\_watchdog\_mask, [236](#)
- lock\_int\_mask\_reg, [236](#)
- qm\_scss\_mailbox\_reg\_t, [236](#)
- qm\_scss\_mem\_reg\_t, [236](#)
- qm\_scss\_peripheral\_reg\_t, [237](#)
  - cfg\_lock, [237](#)
  - periph\_cfg0, [237](#)
- qm\_scss\_pmu\_reg\_t, [237](#)
  - aon\_vr, [238](#)
  - p\_sts, [238](#)
  - pm\_lock, [238](#)
  - pm\_wait, [238](#)
  - rstc, [238](#)
  - rsts, [239](#)
- qm\_scss\_pmux\_reg\_t, [239](#)
  - pmux\_in\_en, [239](#)
  - pmux\_in\_en\_lock, [239](#)
  - pmux\_pullup, [240](#)
  - pmux\_pullup\_lock, [240](#)
  - pmux\_sel, [240](#)
  - pmux\_sel\_0\_lock, [240](#)
  - pmux\_slew, [240](#)
  - pmux\_slew\_lock, [240](#)
- qm\_scss\_ss\_reg\_t, [240](#)
- qm\_soc\_id
  - Identification, [55](#)
- qm\_soc\_reset
  - Initialisation, [56](#)
- qm\_soc\_reset\_t
  - Initialisation, [56](#)
- qm\_soc\_version
  - Identification, [55](#)
- qm\_spi\_async\_transfer\_t, [241](#)
  - callback, [241](#)
  - callback\_data, [241](#)
  - rx, [241](#)
  - rx\_len, [241](#)
  - tx, [242](#)
  - tx\_len, [242](#)
- qm\_spi\_bmode\_t
  - SPI, [87](#)
- qm\_spi\_config\_t, [242](#)
  - bus\_mode, [242](#)
  - clk\_divider, [242](#)
  - frame\_size, [243](#)
  - transfer\_mode, [243](#)
- qm\_spi\_dma\_channel\_config
  - SPI, [89](#)
- qm\_spi\_dma\_transfer
  - SPI, [89](#)
- qm\_spi\_dma\_transfer\_terminate
  - SPI, [90](#)
- qm\_spi\_frame\_size\_t
  - SPI, [87](#)
- qm\_spi\_get\_status
  - SPI, [90](#)
- qm\_spi\_irq\_transfer
  - SPI, [91](#)
- qm\_spi\_irq\_transfer\_terminate
  - SPI, [91](#)
- qm\_spi\_reg\_t, [243](#)
  - baudr, [244](#)
  - ctrlr0, [244](#)
  - ctrlr1, [244](#)
  - dmacr, [245](#)
  - dmardlr, [245](#)
  - dmatdlr, [245](#)
  - dr, [245](#)
  - icr, [245](#)
  - idr, [245](#)
  - imr, [245](#)
  - isr, [245](#)
  - msticr, [245](#)
  - mwcr, [246](#)
  - risr, [246](#)
  - rx\_sample\_dly, [246](#)
  - rxflr, [246](#)
  - rxftlr, [246](#)
  - rxoicr, [246](#)
  - rxuicr, [246](#)
  - ser, [246](#)
  - sr, [246](#)
  - ssi\_comp\_version, [246](#)
  - ssienr, [247](#)
  - txflr, [247](#)
  - txftlr, [247](#)
  - txoicr, [247](#)
- qm\_spi\_set\_config
  - SPI, [92](#)
- qm\_spi\_slave\_select
  - SPI, [92](#)
- qm\_spi\_slave\_select\_t
  - SPI, [88](#)
- qm\_spi\_status\_t
  - SPI, [88](#)
- qm\_spi\_t
  - SoC Registers (D2000), [162](#)
  - SoC Registers (SE), [181](#)
- qm\_spi\_tmode\_t
  - SPI, [88](#)
- qm\_spi\_transfer
  - SPI, [93](#)
- qm\_spi\_transfer\_t, [247](#)
  - rx, [248](#)
  - rx\_len, [248](#)
  - tx, [248](#)
  - tx\_len, [248](#)
- qm\_ss\_adc\_calibrate
  - SS ADC, [114](#)
- qm\_ss\_adc\_cb\_source\_t
  - SS ADC, [113](#)
- qm\_ss\_adc\_channel\_t

- SS ADC, [113](#)
- qm\_ss\_adc\_config\_t, [248](#)
  - resolution, [248](#)
- qm\_ss\_adc\_convert
  - SS ADC, [115](#)
- qm\_ss\_adc\_get\_calibration
  - SS ADC, [115](#)
- qm\_ss\_adc\_irq\_calibrate
  - SS ADC, [115](#)
- 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), [172](#)
- qm\_ss\_adc\_resolution\_t
  - SS ADC, [114](#)
- qm\_ss\_adc\_set\_calibration
  - SS ADC, [117](#)
- qm\_ss\_adc\_set\_config
  - SS ADC, [117](#)
- qm\_ss\_adc\_set\_mode
  - SS ADC, [117](#)
- qm\_ss\_adc\_status\_t
  - SS ADC, [114](#)
- qm\_ss\_adc\_t
  - SoC Registers (Sensor Subsystem), [172](#)
- qm\_ss\_adc\_xfer\_t, [249](#)
  - callback, [249](#)
  - callback\_data, [249](#)
  - ch, [249](#)
  - ch\_len, [250](#)
  - samples, [250](#)
  - samples\_len, [250](#)
- qm\_ss\_creg\_reg\_t
  - SoC Registers (Sensor Subsystem), [172](#)
- qm\_ss\_gpio\_clear\_pin
  - SS GPIO, [119](#)
- qm\_ss\_gpio\_port\_config\_t, [250](#)
  - callback, [251](#)
  - callback\_data, [251](#)
  - direction, [251](#)
  - int\_debounce, [251](#)
  - int\_en, [251](#)
  - int\_polarity, [251](#)
  - int\_type, [251](#)
- qm\_ss\_gpio\_read\_pin
  - SS GPIO, [120](#)
- qm\_ss\_gpio\_read\_port
  - SS GPIO, [120](#)
- qm\_ss\_gpio\_reg\_t
  - SoC Registers (Sensor Subsystem), [172](#)
- qm\_ss\_gpio\_set\_config
  - SS GPIO, [120](#)
- qm\_ss\_gpio\_set\_pin
  - SS GPIO, [122](#)
- qm\_ss\_gpio\_set\_pin\_state
  - SS GPIO, [122](#)
- qm\_ss\_gpio\_state\_t
  - SS GPIO, [119](#)
- qm\_ss\_gpio\_t
  - SoC Registers (Sensor Subsystem), [173](#)
- qm\_ss\_gpio\_write\_port
  - SS GPIO, [122](#)
- qm\_ss\_i2c\_addr\_t
  - SS I2C, [126](#)
- qm\_ss\_i2c\_config\_t, [251](#)
  - address\_mode, [252](#)
  - speed, [252](#)
- qm\_ss\_i2c\_get\_status
  - SS I2C, [126](#)
- qm\_ss\_i2c\_irq\_transfer\_terminate
  - SS I2C, [127](#)
- qm\_ss\_i2c\_master\_irq\_transfer
  - SS I2C, [127](#)
- qm\_ss\_i2c\_master\_read
  - SS I2C, [128](#)
- qm\_ss\_i2c\_master\_write
  - SS I2C, [128](#)
- qm\_ss\_i2c\_reg\_t
  - SoC Registers (Sensor Subsystem), [173](#)
- qm\_ss\_i2c\_set\_config
  - SS I2C, [129](#)
- qm\_ss\_i2c\_set\_speed
  - SS I2C, [129](#)
- qm\_ss\_i2c\_speed\_t
  - SS I2C, [126](#)
- qm\_ss\_i2c\_status\_t
  - SS I2C, [126](#)
- qm\_ss\_i2c\_transfer\_t, [252](#)
  - callback, [253](#)
  - callback\_data, [253](#)
  - rx, [253](#)
  - rx\_len, [253](#)
  - stop, [253](#)
  - tx, [253](#)
  - tx\_len, [253](#)
- qm\_ss\_int\_vector\_request
  - SS Interrupt, [130](#)
- qm\_ss\_irq\_mask
  - SS Interrupt, [130](#)
- qm\_ss\_irq\_request
  - SS Interrupt, [130](#)
- qm\_ss\_irq\_unmask
  - SS Interrupt, [131](#)
- qm\_ss\_spi\_async\_transfer\_t, [254](#)
  - callback, [254](#)
  - callback\_data, [254](#)
  - rx, [254](#)
  - rx\_len, [254](#)
  - tx, [255](#)
  - tx\_len, [255](#)
- qm\_ss\_spi\_bmode\_t
  - SS SPI, [137](#)

- qm\_ss\_spi\_config\_t, 255
  - bus\_mode, 255
  - clk\_divider, 255
  - frame\_size, 256
  - transfer\_mode, 256
- qm\_ss\_spi\_frame\_size\_t
  - SS SPI, 137
- qm\_ss\_spi\_get\_status
  - SS SPI, 138
- qm\_ss\_spi\_irq\_transfer
  - SS SPI, 138
- qm\_ss\_spi\_reg\_t
  - SoC Registers (Sensor Subsystem), 173
- qm\_ss\_spi\_set\_config
  - SS SPI, 140
- qm\_ss\_spi\_slave\_select
  - SS SPI, 140
- qm\_ss\_spi\_slave\_select\_t
  - SS SPI, 137
- qm\_ss\_spi\_status\_t
  - SS SPI, 137
- qm\_ss\_spi\_t
  - SoC Registers (Sensor Subsystem), 173
- qm\_ss\_spi\_tmode\_t
  - SS SPI, 138
- qm\_ss\_spi\_transfer
  - SS SPI, 141
- qm\_ss\_spi\_transfer\_t, 256
  - rx, 256
  - rx\_len, 256
  - tx, 256
  - tx\_len, 257
- qm\_ss\_spi\_transfer\_terminate
  - SS SPI, 141
- qm\_ss\_timer\_config\_t, 257
  - callback, 257
  - callback\_data, 257
  - count, 258
  - inc\_run\_only, 258
  - int\_en, 258
  - watchdog\_mode, 258
- qm\_ss\_timer\_get
  - SS Timer, 142
- qm\_ss\_timer\_set
  - SS Timer, 142
- qm\_ss\_timer\_set\_config
  - SS Timer, 143
- qm\_uart\_config\_t, 258
  - baud\_divisor, 259
  - hw\_fc, 259
  - int\_en, 259
  - line\_control, 259
- qm\_uart\_dma\_channel\_config
  - UART, 96
- qm\_uart\_dma\_read
  - UART, 97
- qm\_uart\_dma\_read\_terminate
  - UART, 97
- qm\_uart\_dma\_write
  - UART, 98
- qm\_uart\_dma\_write\_terminate
  - UART, 98
- qm\_uart\_get\_status
  - UART, 98
- qm\_uart\_irq\_read
  - UART, 99
- qm\_uart\_irq\_read\_terminate
  - UART, 99
- qm\_uart\_irq\_write
  - UART, 100
- qm\_uart\_irq\_write\_terminate
  - UART, 100
- qm\_uart\_lc\_t
  - UART, 95
- qm\_uart\_read
  - UART, 100
- qm\_uart\_read\_non\_block
  - UART, 101
- qm\_uart\_reg\_t, 259
  - de\_en, 260
  - det, 260
  - dlf, 260
  - dmasa, 260
  - htx, 260
  - ier\_dlh, 261
  - iir\_fcr, 261
  - lcr, 261
  - lcr\_ext, 261
  - lsr, 261
  - mcr, 261
  - msr, 261
  - rar, 261
  - rbr\_thr\_dll, 261
  - re\_en, 262
  - scr, 262
  - tar, 262
  - tat, 262
  - tcr, 262
  - usr, 262
- qm\_uart\_set\_config
  - UART, 101
- qm\_uart\_status\_t
  - UART, 96
- qm\_uart\_t
  - SoC Registers (D2000), 162
  - SoC Registers (SE), 181
- qm\_uart\_transfer\_t, 262
  - callback, 263
  - callback\_data, 263
  - data, 263
  - data\_len, 263
- qm\_uart\_write
  - UART, 102
- qm\_uart\_write\_buffer
  - UART, 102
- qm\_uart\_write\_non\_block

- UART, 102
- qm\_ver\_rom
  - Version, 104
- qm\_wdt\_clock\_timeout\_cycles\_t
  - WDT, 105
- qm\_wdt\_config\_t, 263
  - callback, 264
  - callback\_data, 264
  - mode, 264
  - timeout, 264
- qm\_wdt\_mode\_t
  - WDT, 106
- qm\_wdt\_reg\_t, 264
  - wdt\_ccvr, 265
  - wdt\_comp\_param\_1, 265
  - wdt\_comp\_param\_2, 265
  - wdt\_comp\_param\_3, 265
  - wdt\_comp\_param\_4, 265
  - wdt\_comp\_param\_5, 265
  - wdt\_comp\_type, 265
  - wdt\_comp\_version, 266
  - wdt\_cr, 266
  - wdt\_crr, 266
  - wdt\_eoi, 266
  - wdt\_stat, 266
  - wdt\_torr, 266
- qm\_wdt\_reload
  - WDT, 106
- qm\_wdt\_set\_config
  - WDT, 106
- qm\_wdt\_start
  - WDT, 107
- qm\_wdt\_t
  - SoC Registers (D2000), 162
  - SoC Registers (SE), 181
- Quark D2000 ADC
  - QM\_ADC\_CAL\_COMPLETE, 17
  - QM\_ADC\_CH\_0, 17
  - QM\_ADC\_CH\_1, 17
  - QM\_ADC\_CH\_10, 17
  - QM\_ADC\_CH\_11, 17
  - QM\_ADC\_CH\_12, 17
  - QM\_ADC\_CH\_13, 17
  - QM\_ADC\_CH\_14, 17
  - QM\_ADC\_CH\_15, 17
  - QM\_ADC\_CH\_16, 17
  - QM\_ADC\_CH\_17, 18
  - QM\_ADC\_CH\_18, 18
  - QM\_ADC\_CH\_2, 17
  - QM\_ADC\_CH\_3, 17
  - QM\_ADC\_CH\_4, 17
  - QM\_ADC\_CH\_5, 17
  - QM\_ADC\_CH\_6, 17
  - QM\_ADC\_CH\_7, 17
  - QM\_ADC\_CH\_8, 17
  - QM\_ADC\_CH\_9, 17
  - QM\_ADC\_COMPLETE, 18
  - QM\_ADC\_IDLE, 18
  - QM\_ADC\_MODE\_CHANGED, 17
  - QM\_ADC\_MODE\_DEEP\_PWR\_DOWN, 18
  - QM\_ADC\_MODE\_NORM\_CAL, 18
  - QM\_ADC\_MODE\_NORM\_NO\_CAL, 18
  - QM\_ADC\_MODE\_PWR\_DOWN, 18
  - QM\_ADC\_MODE\_STDBY, 18
  - QM\_ADC\_OVERFLOW, 18
  - QM\_ADC\_RES\_10\_BITS, 18
  - QM\_ADC\_RES\_12\_BITS, 18
  - QM\_ADC\_RES\_6\_BITS, 18
  - QM\_ADC\_RES\_8\_BITS, 18
  - QM\_ADC\_TRANSFER, 17
- Quark D2000 ADC, 16
  - qm\_adc\_calibrate, 18
  - qm\_adc\_cb\_source\_t, 17
  - qm\_adc\_channel\_t, 17
  - qm\_adc\_convert, 19
  - qm\_adc\_get\_calibration, 19
  - qm\_adc\_irq\_calibrate, 19
  - qm\_adc\_irq\_convert, 20
  - qm\_adc\_irq\_set\_mode, 20
  - qm\_adc\_mode\_t, 18
  - qm\_adc\_resolution\_t, 18
  - qm\_adc\_set\_calibration, 21
  - qm\_adc\_set\_config, 21
  - qm\_adc\_set\_mode, 21
  - qm\_adc\_status\_t, 18
- Quark D2000 Flash Layout, 152
- Quark D2000 Power states, 153
  - POWER\_WAKE\_FROM\_GPIO\_COMP, 153
  - POWER\_WAKE\_FROM\_RTC, 153
  - power\_cpu\_halt, 153
  - power\_soc\_deep\_sleep, 153
  - power\_soc\_sleep, 154
  - power\_wake\_event\_t, 153
- Quark SE Flash Layout, 166
- Quark SE Host Power states, 169
  - power\_cpu\_c1, 169
  - power\_cpu\_c2, 169
  - power\_cpu\_c2lp, 170
- Quark SE SoC Power states, 167
  - power\_soc\_deep\_sleep, 167
  - power\_soc\_sleep, 167
- Quark SE Voltage Regulators, 183
  - vreg\_aon\_set\_mode, 183
  - vreg\_host\_set\_mode, 183
  - vreg\_plat1p8\_set\_mode, 184
  - vreg\_plat3p3\_set\_mode, 184
- RAR
  - RAR\_NORMAL, 164
  - RAR\_RETENTION, 164
- RAR\_NORMAL
  - RAR, 164
- RAR\_RETENTION
  - RAR, 164
- RAR, 164
  - rar\_set\_mode, 164
  - rar\_state\_t, 164

- RTC, [83](#)
  - qm\_rtc\_set\_alarm, [83](#)
  - qm\_rtc\_set\_config, [83](#)
- rar
  - qm\_uart\_reg\_t, [261](#)
- rar\_set\_mode
  - RAR, [164](#)
- rar\_state\_t
  - RAR, [164](#)
- rbr\_thr\_dll
  - qm\_uart\_reg\_t, [261](#)
- re\_en
  - qm\_uart\_reg\_t, [262](#)
- reference
  - qm\_ac\_config\_t, [187](#)
- resolution
  - qm\_adc\_config\_t, [188](#)
  - qm\_ss\_adc\_config\_t, [248](#)
- rev
  - qm\_scss\_info\_reg\_t, [233](#)
- risr
  - qm\_spi\_reg\_t, [246](#)
- rs
  - qm\_scss\_info\_reg\_t, [233](#)
- rstc
  - qm\_scss\_pmu\_reg\_t, [238](#)
- rsts
  - qm\_scss\_pmu\_reg\_t, [239](#)
- rtc\_ccr
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_ccvr
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_clr
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_cmr
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_comp\_version
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_eoi
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_rstat
  - qm\_rtc\_reg\_t, [225](#)
- rtc\_stat
  - qm\_rtc\_reg\_t, [225](#)
- rx
  - qm\_i2c\_transfer\_t, [212](#)
  - qm\_spi\_async\_transfer\_t, [241](#)
  - qm\_spi\_transfer\_t, [248](#)
  - qm\_ss\_i2c\_transfer\_t, [253](#)
  - qm\_ss\_spi\_async\_transfer\_t, [254](#)
  - qm\_ss\_spi\_transfer\_t, [256](#)
- rx\_len
  - qm\_i2c\_transfer\_t, [212](#)
  - qm\_spi\_async\_transfer\_t, [241](#)
  - qm\_spi\_transfer\_t, [248](#)
  - qm\_ss\_i2c\_transfer\_t, [253](#)
  - qm\_ss\_spi\_async\_transfer\_t, [254](#)
  - qm\_ss\_spi\_transfer\_t, [256](#)
- rx\_sample\_dly
  - qm\_spi\_reg\_t, [246](#)
- rxflr
  - qm\_spi\_reg\_t, [246](#)
- rxftlr
  - qm\_spi\_reg\_t, [246](#)
- rxoicr
  - qm\_spi\_reg\_t, [246](#)
- rxuicr
  - qm\_spi\_reg\_t, [246](#)
- SOC\_WATCH
  - SOCW\_EVENT\_APP, [109](#)
  - SOCW\_EVENT\_HALT, [109](#)
  - SOCW\_EVENT\_INTERRUPT, [109](#)
  - SOCW\_EVENT\_MAX, [109](#)
  - SOCW\_EVENT\_REGISTER, [109](#)
  - SOCW\_EVENT\_SLEEP, [109](#)
  - SOCW\_REG\_CCU\_EXT\_CLK\_CTL, [109, 110](#)
  - SOCW\_REG\_CCU\_LP\_CLK\_CTL, [109, 110](#)
  - SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL, [109, 110](#)
  - SOCW\_REG\_CCU\_SS\_PERIPH\_CLK\_GATE\_CTL, [109, 110](#)
  - SOCW\_REG\_CCU\_SYS\_CLK\_CTL, [109, 110](#)
  - SOCW\_REG\_CMP\_PWR, [109, 110](#)
  - SOCW\_REG\_MAX, [109, 110](#)
  - SOCW\_REG\_OSC0\_CFG1, [109, 110](#)
  - SOCW\_REG\_PMUX\_IN\_EN, [109, 110](#)
  - SOCW\_REG\_PMUX\_IN\_EN0, [110](#)
  - SOCW\_REG\_PMUX\_IN\_EN1, [110](#)
  - SOCW\_REG\_PMUX\_IN\_EN2, [110](#)
  - SOCW\_REG\_PMUX\_IN\_EN3, [110](#)
  - SOCW\_REG\_PMUX\_PULLUP, [109, 110](#)
  - SOCW\_REG\_PMUX\_PULLUP0, [109, 110](#)
  - SOCW\_REG\_PMUX\_PULLUP1, [109, 110](#)
  - SOCW\_REG\_PMUX\_PULLUP2, [110](#)
  - SOCW\_REG\_PMUX\_PULLUP3, [110](#)
  - SOCW\_REG\_PMUX\_SLEW, [109, 110](#)
  - SOCW\_REG\_PMUX\_SLEW0, [110](#)
  - SOCW\_REG\_PMUX\_SLEW1, [110](#)
  - SOCW\_REG\_PMUX\_SLEW2, [110](#)
  - SOCW\_REG\_PMUX\_SLEW3, [110](#)
  - SOCW\_REG\_SLP\_CFG, [109, 110](#)
- SOCW\_EVENT\_APP
  - SOC\_WATCH, [109](#)
- SOCW\_EVENT\_HALT
  - SOC\_WATCH, [109](#)
- SOCW\_EVENT\_INTERRUPT
  - SOC\_WATCH, [109](#)
- SOCW\_EVENT\_MAX
  - SOC\_WATCH, [109](#)
- SOCW\_EVENT\_REGISTER
  - SOC\_WATCH, [109](#)
- SOCW\_EVENT\_SLEEP
  - SOC\_WATCH, [109](#)
- SOCW\_REG\_CCU\_EXT\_CLK\_CTL
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_CCU\_LP\_CLK\_CTL

- SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_CCU\_PERIPH\_CLK\_GATE\_CTL
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_CCU\_SS\_PERIPH\_CLK\_GATE\_CTL
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_CCU\_SYS\_CLK\_CTL
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_CMP\_PWR
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_MAX
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_OSC0\_CFG1
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_PMUX\_IN\_EN
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_PMUX\_IN\_EN0
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_IN\_EN1
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_IN\_EN2
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_IN\_EN3
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_PULLUP
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_PMUX\_PULLUP0
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_PMUX\_PULLUP1
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_PMUX\_PULLUP2
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_PULLUP3
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_SLEW
  - SOC\_WATCH, [109, 110](#)
- SOCW\_REG\_PMUX\_SLEW0
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_SLEW1
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_SLEW2
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_PMUX\_SLEW3
  - SOC\_WATCH, [110](#)
- SOCW\_REG\_SLP\_CFG
  - SOC\_WATCH, [109, 110](#)
- SPI
  - QM\_SPI\_BMODE\_0, [87](#)
  - QM\_SPI\_BMODE\_1, [87](#)
  - QM\_SPI\_BMODE\_2, [87](#)
  - QM\_SPI\_BMODE\_3, [87](#)
  - QM\_SPI\_BUSY, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_10\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_11\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_12\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_13\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_14\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_15\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_16\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_17\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_18\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_19\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_20\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_21\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_22\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_23\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_24\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_25\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_26\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_27\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_28\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_29\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_30\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_31\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_32\_BIT, [88](#)
  - QM\_SPI\_FRAME\_SIZE\_4\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_5\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_6\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_7\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_8\_BIT, [87](#)
  - QM\_SPI\_FRAME\_SIZE\_9\_BIT, [87](#)
  - QM\_SPI\_IDLE, [88](#)
  - QM\_SPI\_RX\_OVERFLOW, [88](#)
  - QM\_SPI\_SS\_0, [88](#)
  - QM\_SPI\_SS\_1, [88](#)
  - QM\_SPI\_SS\_2, [88](#)
  - QM\_SPI\_SS\_3, [88](#)
  - QM\_SPI\_SS\_DISABLED, [88](#)
  - QM\_SPI\_TMOD\_EEPROM\_READ, [89](#)
  - QM\_SPI\_TMOD\_RX, [89](#)
  - QM\_SPI\_TMOD\_TX, [89](#)
  - QM\_SPI\_TMOD\_TX\_RX, [89](#)
- 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, [113](#)
  - QM\_SS\_ADC\_CH\_11, [113](#)
  - QM\_SS\_ADC\_CH\_12, [113](#)
  - 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, [113](#)
  - 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, 145
  - SS\_CLK\_PERIPH\_GPIO\_0, 145
  - SS\_CLK\_PERIPH\_GPIO\_1, 145
  - SS\_CLK\_PERIPH\_I2C\_0, 145
  - SS\_CLK\_PERIPH\_I2C\_1, 145
  - SS\_CLK\_PERIPH\_SPI\_0, 145
  - SS\_CLK\_PERIPH\_SPI\_1, 145
- SS GPIO
  - QM\_SS\_GPIO\_HIGH, 119
  - QM\_SS\_GPIO\_LOW, 119
- SS I2C
  - QM\_I2C\_BUSY, 126
  - QM\_I2C\_IDLE, 126
  - QM\_I2C\_TX\_ABRT\_7B\_ADDR\_NOACK, 126
  - QM\_I2C\_TX\_ABRT\_MASTER\_DIS, 126
  - QM\_I2C\_TX\_ABRT\_SBYTE\_ACKDET, 126
  - QM\_I2C\_TX\_ABRT\_SLV\_ARBLOST, 126
  - QM\_I2C\_TX\_ABRT\_SLVFLUSH\_TXFIFO, 126
  - QM\_I2C\_TX\_ABRT\_SLVRD\_INTX, 126
  - QM\_I2C\_TX\_ABRT\_TXDATA\_NOACK, 126
  - QM\_I2C\_TX\_ABRT\_USER\_ABRT, 126
  - QM\_I2C\_TX\_ARB\_LOST, 126
  - QM\_SS\_I2C\_10\_BIT, 126
  - QM\_SS\_I2C\_7\_BIT, 126
  - QM\_SS\_I2C\_SPEED\_FAST, 126
  - QM\_SS\_I2C\_SPEED\_STD, 126
- SS Power states
  - SS\_POWER\_CPU\_SS1\_TIMER\_OFF, 149
  - SS\_POWER\_CPU\_SS1\_TIMER\_ON, 149
- SS SPI
  - QM\_SS\_SPI\_BMODE\_0, 137
  - QM\_SS\_SPI\_BMODE\_1, 137
  - QM\_SS\_SPI\_BMODE\_2, 137
  - QM\_SS\_SPI\_BMODE\_3, 137
  - QM\_SS\_SPI\_BUSY, 138
  - QM\_SS\_SPI\_FRAME\_SIZE\_10\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_11\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_12\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_13\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_14\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_15\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_16\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_4\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_5\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_6\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_7\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_8\_BIT, 137
  - QM\_SS\_SPI\_FRAME\_SIZE\_9\_BIT, 137
  - QM\_SS\_SPI\_IDLE, 138
  - QM\_SS\_SPI\_RX\_OVERFLOW, 138
  - QM\_SS\_SPI\_SS\_0, 137
  - QM\_SS\_SPI\_SS\_1, 137
  - QM\_SS\_SPI\_SS\_2, 137
  - QM\_SS\_SPI\_SS\_3, 137
  - QM\_SS\_SPI\_SS\_DISABLED, 137
  - QM\_SS\_SPI\_TMOD\_EEPROM\_READ, 138
  - QM\_SS\_SPI\_TMOD\_RX, 138
  - QM\_SS\_SPI\_TMOD\_TX, 138
  - QM\_SS\_SPI\_TMOD\_TX\_RX, 138
- SS\_CLK\_PERIPH\_ADC
  - SS Clock, 145
- SS\_CLK\_PERIPH\_GPIO\_0
  - SS Clock, 145
- SS\_CLK\_PERIPH\_GPIO\_1
  - SS Clock, 145
- SS\_CLK\_PERIPH\_I2C\_0
  - SS Clock, 145
- SS\_CLK\_PERIPH\_I2C\_1
  - SS Clock, 145
- SS\_CLK\_PERIPH\_SPI\_0
  - SS Clock, 145
- SS\_CLK\_PERIPH\_SPI\_1
  - SS Clock, 145
- SS\_POWER\_CPU\_SS1\_TIMER\_OFF
  - SS Power states, 149
- SS\_POWER\_CPU\_SS1\_TIMER\_ON
  - SS Power states, 149
- SOC\_WATCH, 108
  - soc\_watch\_event\_t, 109
  - soc\_watch\_log\_app\_event, 111
  - soc\_watch\_log\_event, 111
  - soc\_watch\_reg\_t, 109, 110
- SPI, 86
  - qm\_spi\_bmode\_t, 87
  - qm\_spi\_dma\_channel\_config, 89
  - qm\_spi\_dma\_transfer, 89
  - qm\_spi\_dma\_transfer\_terminate, 90
  - qm\_spi\_frame\_size\_t, 87
  - qm\_spi\_get\_status, 90
  - qm\_spi\_irq\_transfer, 91
  - qm\_spi\_irq\_transfer\_terminate, 91
  - qm\_spi\_set\_config, 92
  - qm\_spi\_slave\_select, 92
  - qm\_spi\_slave\_select\_t, 88
  - qm\_spi\_status\_t, 88
  - qm\_spi\_tmode\_t, 88
  - qm\_spi\_transfer, 93
- SS ADC, 112
  - qm\_ss\_adc\_calibrate, 114
  - 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, 115
- 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\_set\_calibration, 117
- qm\_ss\_adc\_set\_config, 117
- qm\_ss\_adc\_set\_mode, 117
- qm\_ss\_adc\_status\_t, 114
- SS Clock, 144
  - ss\_clk\_adc\_disable, 145
  - ss\_clk\_adc\_enable, 145
  - ss\_clk\_adc\_set\_div, 145
  - ss\_clk\_gpio\_disable, 146
  - ss\_clk\_gpio\_enable, 146
  - ss\_clk\_i2c\_disable, 146
  - ss\_clk\_i2c\_enable, 147
  - ss\_clk\_periph\_t, 144
  - ss\_clk\_spi\_disable, 147
  - ss\_clk\_spi\_enable, 147
- SS GPIO, 119
  - qm\_ss\_gpio\_clear\_pin, 119
  - qm\_ss\_gpio\_read\_pin, 120
  - qm\_ss\_gpio\_read\_port, 120
  - qm\_ss\_gpio\_set\_config, 120
  - qm\_ss\_gpio\_set\_pin, 122
  - qm\_ss\_gpio\_set\_pin\_state, 122
  - qm\_ss\_gpio\_state\_t, 119
  - qm\_ss\_gpio\_write\_port, 122
- SS I2C, 125
  - qm\_ss\_i2c\_addr\_t, 126
  - qm\_ss\_i2c\_get\_status, 126
  - qm\_ss\_i2c\_irq\_transfer\_terminate, 127
  - qm\_ss\_i2c\_master\_irq\_transfer, 127
  - qm\_ss\_i2c\_master\_read, 128
  - qm\_ss\_i2c\_master\_write, 128
  - qm\_ss\_i2c\_set\_config, 129
  - qm\_ss\_i2c\_set\_speed, 129
  - qm\_ss\_i2c\_speed\_t, 126
  - qm\_ss\_i2c\_status\_t, 126
- SS ISR, 132
  - QM\_ISR\_DECLARE, 132–135
- SS Interrupt, 130
  - qm\_ss\_int\_vector\_request, 130
  - qm\_ss\_irq\_mask, 130
  - qm\_ss\_irq\_request, 130
  - qm\_ss\_irq\_unmask, 131
- SS Power states, 149
  - ss\_power\_cpu\_ss1, 149
  - ss\_power\_cpu\_ss1\_mode\_t, 149
  - ss\_power\_cpu\_ss2, 150
  - ss\_power\_soc\_lpss\_disable, 150
  - ss\_power\_soc\_lpss\_enable, 150
- SS SPI, 136
  - qm\_ss\_spi\_bmode\_t, 137
  - qm\_ss\_spi\_frame\_size\_t, 137
  - qm\_ss\_spi\_get\_status, 138
  - qm\_ss\_spi\_irq\_transfer, 138
  - qm\_ss\_spi\_set\_config, 140
  - qm\_ss\_spi\_slave\_select, 140
  - qm\_ss\_spi\_slave\_select\_t, 137
  - qm\_ss\_spi\_status\_t, 137
  - qm\_ss\_spi\_tmode\_t, 138
  - qm\_ss\_spi\_transfer, 141
  - qm\_ss\_spi\_transfer\_terminate, 141
- SS Timer, 142
  - qm\_ss\_timer\_get, 142
  - qm\_ss\_timer\_set, 142
  - qm\_ss\_timer\_set\_config, 143
- samples
  - qm\_adc\_xfer\_t, 190
  - qm\_ss\_adc\_xfer\_t, 250
- samples\_len
  - qm\_adc\_xfer\_t, 190
  - qm\_ss\_adc\_xfer\_t, 250
- scr
  - qm\_uart\_reg\_t, 262
- ser
  - qm\_spi\_reg\_t, 246
- sivr
  - qm\_mvic\_reg\_t, 217
- slave\_addr
  - qm\_i2c\_config\_t, 204
- SoC Registers (D2000)
  - CLK\_PERIPH\_ADC, 160
  - CLK\_PERIPH\_ADC\_REGISTER, 160
  - CLK\_PERIPH\_ALL, 160
  - CLK\_PERIPH\_CLK, 159, 160
  - CLK\_PERIPH\_GPIO\_DB, 159, 160
  - CLK\_PERIPH\_GPIO\_INTERRUPT, 159, 160
  - CLK\_PERIPH\_GPIO\_REGISTER, 160
  - CLK\_PERIPH\_I2C\_M0, 159, 160
  - CLK\_PERIPH\_I2C\_M0\_REGISTER, 160
  - CLK\_PERIPH\_I2C\_M1, 160
  - CLK\_PERIPH\_I2C\_M1\_REGISTER, 160
  - CLK\_PERIPH\_I2S, 160
  - CLK\_PERIPH\_I2S\_REGISTER, 160
  - CLK\_PERIPH\_PWM\_REGISTER, 160
  - CLK\_PERIPH\_REGISTER, 159, 160
  - CLK\_PERIPH\_RTC\_REGISTER, 159, 160
  - CLK\_PERIPH\_SPI\_M0, 159, 160
  - CLK\_PERIPH\_SPI\_M0\_REGISTER, 160
  - CLK\_PERIPH\_SPI\_M1, 160
  - CLK\_PERIPH\_SPI\_M1\_REGISTER, 160
  - CLK\_PERIPH\_SPI\_S, 159, 160
  - CLK\_PERIPH\_SPI\_S\_REGISTER, 160
  - CLK\_PERIPH\_UARTA\_REGISTER, 160
  - CLK\_PERIPH\_UARTB\_REGISTER, 160
  - CLK\_PERIPH\_WDT\_REGISTER, 159, 160
  - DMA\_HW\_IF\_I2C\_MASTER\_0\_RX, 161
  - DMA\_HW\_IF\_I2C\_MASTER\_0\_TX, 161
  - DMA\_HW\_IF\_I2C\_MASTER\_1\_RX, 161
  - DMA\_HW\_IF\_I2C\_MASTER\_1\_TX, 161
  - DMA\_HW\_IF\_I2S\_CAPTURE, 161
  - DMA\_HW\_IF\_I2S\_PLAYBACK, 161
  - DMA\_HW\_IF\_SPI\_MASTER\_0\_RX, 161

- DMA\_HW\_IF\_SPI\_MASTER\_0\_TX, 161
- DMA\_HW\_IF\_SPI\_MASTER\_1\_RX, 161
- DMA\_HW\_IF\_SPI\_MASTER\_1\_TX, 161
- DMA\_HW\_IF\_SPI\_SLAVE\_RX, 161
- DMA\_HW\_IF\_SPI\_SLAVE\_TX, 161
- DMA\_HW\_IF\_UART\_A\_RX, 161
- DMA\_HW\_IF\_UART\_A\_TX, 161
- DMA\_HW\_IF\_UART\_B\_RX, 161
- DMA\_HW\_IF\_UART\_B\_TX, 161
- QM\_DMA\_0, 162
- QM\_DMA\_CHANNEL\_0, 161
- QM\_DMA\_CHANNEL\_1, 161
- QM\_DMA\_CHANNEL\_2, 161
- QM\_DMA\_CHANNEL\_3, 161
- QM\_DMA\_CHANNEL\_4, 161
- QM\_DMA\_CHANNEL\_5, 161
- QM\_DMA\_CHANNEL\_6, 161
- QM\_DMA\_CHANNEL\_7, 161
- QM\_DMA\_CHANNEL\_NUM, 161
- QM\_DMA\_NUM, 162
- SoC Registers (SE)
  - CLK\_PERIPH\_ADC, 179
  - CLK\_PERIPH\_ADC\_REGISTER, 179
  - CLK\_PERIPH\_ALL, 179
  - CLK\_PERIPH\_CLK, 178, 179
  - CLK\_PERIPH\_GPIO\_DB, 178, 179
  - CLK\_PERIPH\_GPIO\_INTERRUPT, 178, 179
  - CLK\_PERIPH\_GPIO\_REGISTER, 179
  - CLK\_PERIPH\_I2C\_M0, 178, 179
  - CLK\_PERIPH\_I2C\_M0\_REGISTER, 179
  - CLK\_PERIPH\_I2C\_M1, 179
  - CLK\_PERIPH\_I2C\_M1\_REGISTER, 179
  - CLK\_PERIPH\_I2S, 179
  - CLK\_PERIPH\_I2S\_REGISTER, 179
  - CLK\_PERIPH\_PWM\_REGISTER, 179
  - CLK\_PERIPH\_REGISTER, 178, 179
  - CLK\_PERIPH\_RTC\_REGISTER, 179
  - CLK\_PERIPH\_SPI\_M0, 178, 179
  - CLK\_PERIPH\_SPI\_M0\_REGISTER, 179
  - CLK\_PERIPH\_SPI\_M1, 179
  - CLK\_PERIPH\_SPI\_M1\_REGISTER, 179
  - CLK\_PERIPH\_SPI\_S, 178, 179
  - CLK\_PERIPH\_SPI\_S\_REGISTER, 179
  - CLK\_PERIPH\_UARTA\_REGISTER, 179
  - CLK\_PERIPH\_UARTB\_REGISTER, 179
  - CLK\_PERIPH\_WDT\_REGISTER, 179
  - DMA\_HW\_IF\_I2C\_MASTER\_0\_RX, 180
  - DMA\_HW\_IF\_I2C\_MASTER\_0\_TX, 180
  - DMA\_HW\_IF\_I2C\_MASTER\_1\_RX, 180
  - DMA\_HW\_IF\_I2C\_MASTER\_1\_TX, 180
  - DMA\_HW\_IF\_I2S\_CAPTURE, 180
  - DMA\_HW\_IF\_I2S\_PLAYBACK, 180
  - DMA\_HW\_IF\_SPI\_MASTER\_0\_RX, 180
  - DMA\_HW\_IF\_SPI\_MASTER\_0\_TX, 180
  - DMA\_HW\_IF\_SPI\_MASTER\_1\_RX, 180
  - DMA\_HW\_IF\_SPI\_MASTER\_1\_TX, 180
  - DMA\_HW\_IF\_SPI\_SLAVE\_RX, 180
  - DMA\_HW\_IF\_SPI\_SLAVE\_TX, 180
- DMA\_HW\_IF\_UART\_A\_RX, 180
- DMA\_HW\_IF\_UART\_A\_TX, 180
- DMA\_HW\_IF\_UART\_B\_RX, 180
- DMA\_HW\_IF\_UART\_B\_TX, 180
- QM\_DMA\_0, 181
- QM\_DMA\_CHANNEL\_0, 179
- QM\_DMA\_CHANNEL\_1, 179, 180
- QM\_DMA\_CHANNEL\_2, 180
- QM\_DMA\_CHANNEL\_3, 180
- QM\_DMA\_CHANNEL\_4, 180
- QM\_DMA\_CHANNEL\_5, 180
- QM\_DMA\_CHANNEL\_6, 180
- QM\_DMA\_CHANNEL\_7, 180
- QM\_DMA\_CHANNEL\_NUM, 179, 180
- QM\_DMA\_NUM, 181
- SoC Registers (Sensor Subsystem)
  - QM\_SS\_ADC\_0, 172
  - QM\_SS\_ADC\_CTRL, 172
  - QM\_SS\_ADC\_DIVSEQSTAT, 172
  - QM\_SS\_ADC\_INTSTAT, 172
  - QM\_SS\_ADC\_SAMPLE, 172
  - QM\_SS\_ADC\_SEQ, 172
  - QM\_SS\_ADC\_SET, 172
  - QM\_SS\_IO\_CREG\_MST0\_CTRL, 172
  - QM\_SS\_IO\_CREG\_SLV0\_OBSR, 172
  - QM\_SS\_IO\_CREG\_SLV1\_OBSR, 172
  - QM\_SS\_SPI\_0, 173
  - QM\_SS\_SPI\_1, 173
  - QM\_SS\_SPI\_CLR\_INTR, 173
  - QM\_SS\_SPI\_CTRL, 173
  - QM\_SS\_SPI\_DR, 173
  - QM\_SS\_SPI\_FTLR, 173
  - QM\_SS\_SPI\_INTR\_MASK, 173
  - QM\_SS\_SPI\_INTR\_STAT, 173
  - QM\_SS\_SPI\_RXFLR, 173
  - QM\_SS\_SPI\_SPIEN, 173
  - QM\_SS\_SPI\_SR, 173
  - QM\_SS\_SPI\_TIMING, 173
  - QM\_SS\_SPI\_TXFLR, 173
- SoC Registers (D2000), 155
  - clk\_periph\_t, 159
  - qm\_adc\_t, 160
  - qm\_dma\_channel\_id\_t, 160
  - qm\_dma\_handshake\_interface\_t, 161
  - qm\_dma\_t, 161
  - qm\_flash\_t, 162
  - qm\_gpio\_t, 162
  - qm\_i2c, 163
  - qm\_i2c\_t, 162
  - qm\_pwm\_id\_t, 162
  - qm\_pwm\_t, 162
  - qm\_rtc\_t, 162
  - qm\_scss\_aon\_t, 162
  - qm\_spi\_t, 162
  - qm\_uart\_t, 162
  - qm\_wdt\_t, 162
- SoC Registers (SE), 174
  - clk\_periph\_t, 178

- qm\_dma\_channel\_id\_t, 179
- qm\_dma\_handshake\_interface\_t, 180
- qm\_dma\_t, 180
- qm\_flash\_t, 181
- qm\_gpio\_t, 181
- qm\_i2c, 182
- qm\_i2c\_t, 181
- qm\_pwm\_id\_t, 181
- qm\_pwm\_t, 181
- qm\_rtc\_t, 181
- qm\_scss\_aon\_t, 181
- qm\_spi\_t, 181
- qm\_uart\_t, 181
- qm\_wdt\_t, 181
- SoC Registers (Sensor Subsystem), 171
  - qm\_ss\_adc\_reg\_t, 172
  - qm\_ss\_adc\_t, 172
  - qm\_ss\_creg\_reg\_t, 172
  - qm\_ss\_gpio\_reg\_t, 172
  - qm\_ss\_gpio\_t, 173
  - qm\_ss\_i2c\_reg\_t, 173
  - qm\_ss\_spi\_reg\_t, 173
  - qm\_ss\_spi\_t, 173
- soc\_ctrl
  - qm\_scss\_ccu\_reg\_t, 229
- soc\_ctrl\_lock
  - qm\_scss\_ccu\_reg\_t, 229
- soc\_watch\_event\_t
  - SOC\_WATCH, 109
- soc\_watch\_log\_app\_event
  - SOC\_WATCH, 111
- soc\_watch\_log\_event
  - SOC\_WATCH, 111
- soc\_watch\_reg\_t
  - SOC\_WATCH, 109, 110
- source\_address
  - qm\_dma\_transfer\_t, 197
- speed
  - qm\_i2c\_config\_t, 204
  - qm\_ss\_i2c\_config\_t, 252
- sr
  - qm\_spi\_reg\_t, 246
- ss\_clk\_adc\_disable
  - SS Clock, 145
- ss\_clk\_adc\_enable
  - SS Clock, 145
- ss\_clk\_adc\_set\_div
  - SS Clock, 145
- ss\_clk\_gpio\_disable
  - SS Clock, 146
- ss\_clk\_gpio\_enable
  - SS Clock, 146
- ss\_clk\_i2c\_disable
  - SS Clock, 146
- ss\_clk\_i2c\_enable
  - SS Clock, 147
- ss\_clk\_periph\_t
  - SS Clock, 144
- ss\_clk\_spi\_disable
  - SS Clock, 147
- ss\_clk\_spi\_enable
  - SS Clock, 147
- ss\_power\_cpu\_ss1
  - SS Power states, 149
- ss\_power\_cpu\_ss1\_mode\_t
  - SS Power states, 149
- ss\_power\_cpu\_ss2
  - SS Power states, 150
- ss\_power\_soc\_lpss\_disable
  - SS Power states, 150
- ss\_power\_soc\_lpss\_enable
  - SS Power states, 150
- ssi\_comp\_version
  - qm\_spi\_reg\_t, 246
- ssienr
  - qm\_spi\_reg\_t, 247
- stop
  - qm\_i2c\_transfer\_t, 212
  - qm\_ss\_i2c\_transfer\_t, 253
- Syscalls, 185
  - pico\_printf, 185
- tar
  - qm\_uart\_reg\_t, 262
- tat
  - qm\_uart\_reg\_t, 262
- tcr
  - qm\_uart\_reg\_t, 262
- timeout
  - qm\_wdt\_config\_t, 264
- timer
  - qm\_pwm\_reg\_t, 222
- tpr
  - qm\_mvic\_reg\_t, 217
- transfer\_mode
  - qm\_spi\_config\_t, 243
  - qm\_ss\_spi\_config\_t, 256
- tx
  - qm\_i2c\_transfer\_t, 212
  - qm\_spi\_async\_transfer\_t, 242
  - qm\_spi\_transfer\_t, 248
  - qm\_ss\_i2c\_transfer\_t, 253
  - qm\_ss\_spi\_async\_transfer\_t, 255
  - qm\_ss\_spi\_transfer\_t, 256
- tx\_len
  - qm\_i2c\_transfer\_t, 212
  - qm\_spi\_async\_transfer\_t, 242
  - qm\_spi\_transfer\_t, 248
  - qm\_ss\_i2c\_transfer\_t, 253
  - qm\_ss\_spi\_async\_transfer\_t, 255
  - qm\_ss\_spi\_transfer\_t, 257
- txflr
  - qm\_spi\_reg\_t, 247
- txftlr
  - qm\_spi\_reg\_t, 247
- txoicr
  - qm\_spi\_reg\_t, 247

## UART

QM\_UART\_IDLE, 96  
 QM\_UART\_LC\_5E1, 95  
 QM\_UART\_LC\_5E1\_5, 95  
 QM\_UART\_LC\_5N1, 95  
 QM\_UART\_LC\_5N1\_5, 95  
 QM\_UART\_LC\_5O1, 95  
 QM\_UART\_LC\_5O1\_5, 95  
 QM\_UART\_LC\_6E1, 95  
 QM\_UART\_LC\_6E2, 95  
 QM\_UART\_LC\_6N1, 95  
 QM\_UART\_LC\_6N2, 95  
 QM\_UART\_LC\_6O1, 95  
 QM\_UART\_LC\_6O2, 95  
 QM\_UART\_LC\_7E1, 95  
 QM\_UART\_LC\_7E2, 95  
 QM\_UART\_LC\_7N1, 95  
 QM\_UART\_LC\_7N2, 95  
 QM\_UART\_LC\_7O1, 95  
 QM\_UART\_LC\_7O2, 95  
 QM\_UART\_LC\_8E1, 96  
 QM\_UART\_LC\_8E2, 96  
 QM\_UART\_LC\_8N1, 95  
 QM\_UART\_LC\_8N2, 95  
 QM\_UART\_LC\_8O1, 96  
 QM\_UART\_LC\_8O2, 96  
 QM\_UART\_RX\_BI, 96  
 QM\_UART\_RX\_BUSY, 96  
 QM\_UART\_RX\_FE, 96  
 QM\_UART\_RX\_NEMPTY, 96  
 QM\_UART\_RX\_OE, 96  
 QM\_UART\_RX\_PE, 96  
 QM\_UART\_TX\_BUSY, 96  
 QM\_UART\_TX\_NFULL, 96

## UART, 94

qm\_uart\_dma\_channel\_config, 96  
 qm\_uart\_dma\_read, 97  
 qm\_uart\_dma\_read\_terminate, 97  
 qm\_uart\_dma\_write, 98  
 qm\_uart\_dma\_write\_terminate, 98  
 qm\_uart\_get\_status, 98  
 qm\_uart\_irq\_read, 99  
 qm\_uart\_irq\_read\_terminate, 99  
 qm\_uart\_irq\_write, 100  
 qm\_uart\_irq\_write\_terminate, 100  
 qm\_uart\_lc\_t, 95  
 qm\_uart\_read, 100  
 qm\_uart\_read\_non\_block, 101  
 qm\_uart\_set\_config, 101  
 qm\_uart\_status\_t, 96  
 qm\_uart\_write, 102  
 qm\_uart\_write\_buffer, 102  
 qm\_uart\_write\_non\_block, 102

## up\_bound

qm\_fpr\_config\_t, 199

## us\_count

qm\_flash\_config\_t, 197

## usb\_pll\_cfg0

qm\_scss\_ccu\_reg\_t, 229

## usr

qm\_uart\_reg\_t, 262

## Version, 104

qm\_ver\_rom, 104

## vreg\_aon\_set\_mode

Quark SE Voltage Regulators, 183

## vreg\_host\_set\_mode

Quark SE Voltage Regulators, 183

## vreg\_plat1p8\_set\_mode

Quark SE Voltage Regulators, 184

## vreg\_plat3p3\_set\_mode

Quark SE Voltage Regulators, 184

## WDT

QM\_WDT\_2\_POW\_16\_CYCLES, 105  
 QM\_WDT\_2\_POW\_17\_CYCLES, 105  
 QM\_WDT\_2\_POW\_18\_CYCLES, 105  
 QM\_WDT\_2\_POW\_19\_CYCLES, 105  
 QM\_WDT\_2\_POW\_20\_CYCLES, 105  
 QM\_WDT\_2\_POW\_21\_CYCLES, 105  
 QM\_WDT\_2\_POW\_22\_CYCLES, 105  
 QM\_WDT\_2\_POW\_23\_CYCLES, 106  
 QM\_WDT\_2\_POW\_24\_CYCLES, 106  
 QM\_WDT\_2\_POW\_25\_CYCLES, 106  
 QM\_WDT\_2\_POW\_26\_CYCLES, 106  
 QM\_WDT\_2\_POW\_27\_CYCLES, 106  
 QM\_WDT\_2\_POW\_28\_CYCLES, 106  
 QM\_WDT\_2\_POW\_29\_CYCLES, 106  
 QM\_WDT\_2\_POW\_30\_CYCLES, 106  
 QM\_WDT\_2\_POW\_31\_CYCLES, 106  
 QM\_WDT\_MODE\_INTERRUPT\_RESET, 106  
 QM\_WDT\_MODE\_RESET, 106

## WDT, 105

qm\_wdt\_clock\_timeout\_cycles\_t, 105

qm\_wdt\_mode\_t, 106

qm\_wdt\_reload, 106

qm\_wdt\_set\_config, 106

qm\_wdt\_start, 107

## wait\_states

qm\_flash\_config\_t, 197

## wake\_mask

qm\_scss\_ccu\_reg\_t, 229

## watchdog\_mode

qm\_ss\_timer\_config\_t, 258

## wdt\_ccvr

qm\_wdt\_reg\_t, 265

## wdt\_comp\_param\_1

qm\_wdt\_reg\_t, 265

## wdt\_comp\_param\_2

qm\_wdt\_reg\_t, 265

## wdt\_comp\_param\_3

qm\_wdt\_reg\_t, 265

## wdt\_comp\_param\_4

qm\_wdt\_reg\_t, 265

## wdt\_comp\_param\_5

qm\_wdt\_reg\_t, 265

## wdt\_comp\_type

qm\_wdt\_reg\_t, [265](#)  
wdt\_comp\_version  
qm\_wdt\_reg\_t, [266](#)  
wdt\_cr  
qm\_wdt\_reg\_t, [266](#)  
wdt\_crr  
qm\_wdt\_reg\_t, [266](#)  
wdt\_eoi  
qm\_wdt\_reg\_t, [266](#)  
wdt\_stat  
qm\_wdt\_reg\_t, [266](#)  
wdt\_torr  
qm\_wdt\_reg\_t, [266](#)  
wo\_sp  
qm\_scss\_gp\_reg\_t, [231](#)  
wo\_st  
qm\_scss\_gp\_reg\_t, [232](#)  
write\_disable  
qm\_flash\_config\_t, [197](#)