UWB QHAL API

Qorvo

Release R12.7.0-405-gb33c5c4272

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

Contents

1 Overview

The Qorvo Hardware Abstraction Layer (QHAL) is an abstraction layer of the hardware and most importantly of the SDK used. The objective of this layer is to provide a common API for Qorvo applications to abstract any platform specific functions from our generic software avoiding direct dependencies with the platform.

The selection of a specific implementation is made through the `CONFIG_QHAL_IMPL_<impl>` cmake option, however it needs to also be set as a compilation option. It can also be selected through Kconfig.

In the scope of the UWB stack, the QHAL is one of the common dependency of all its components and is only dependent on the drivers or SDK and the Qosal with some exceptions. Be careful as there might be some cyclic dependencies with the Qorvo Operating System Abstraction Layer (QOSAL) which abstracts the OS.

1.1 Porting guide

The QHAL can be re-implemented for an unsupported SDK or adapted for an unsupported version of one of the supported SDK as long as it follows the interface.

The specific code for an implementation is contained in the directories:

- `qhal/src/<impl>` for the specific sources
- `qhal/src/<chip>` for the common sources for a specific chip

The behavior of each functions to define is descriped in their *api documentation*.

To modify the build, the file `qhal/CMakeLists.txt` shall be updated, either by adding a new `CONFIG_QHAL_IMPL_<impl>` option, or by modifying the existing one.

Some parameters can be set as cmake option and as compile definition of the build to configure the qosal, as this parameters are in the interface, they shall be set during the build of the components that depends on it:

- `CONFIG_QHAL_MAX_GPIO_CALLBACKS` to set the maximum of GPIO callbacks allowed (default: `3` in Kconfig, undefined if not using it)

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

Warning: The input parameters of the qhal QSPI, QTIMER and QGPIO are implementation dependent. Therefore pre-compiled libraries depending on the qhal are only compatible with the qhal implementation type against which they were built. This is a current limitation of this version of the qhal. In the future, the qhal will be replaced by qplatform which should solve this issue.

1.2 Supported SDK

The QHAL is tested on the following SDK and versions:

- nRF5 Software Development Kit v17.1.0 (`nrfx`)

Be careful as the support depends on the hardware and the software used.

SDK	QM33 + NRF52XX	QM33 + NRF53XX
nRF5 Software Development Kit v17.1.0	Supported	Not supported

2 QHAL API

2.1 QFLASH

2.1.1 qflash_write

enum qerr **`qflash_write`**(uint32_t dst_addr, void *src_addr, uint32_t size)

Write an address in flash memory.

Parameters

- **`dst_addr`** (`uint32_t`) – Destination address. Allowed value is implementation dependent.
- **`src_addr`** (`void*`) – Source address.
- **`size`** (`uint32_t`) – Size of the memory to write.

2.1.1.1 Description

Allowed values for dst_addr are implementation dependent. Depending on the flash and the implementation, the parameter might have to be aligned to 2 or 4 bytes or a sector or a block.

2.1.1.2 Return

QERR_SUCCESS or error.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.2 QGPIO

2.2.1 struct qgpio

struct **qgpio**

Descriptor of a GPIO pin.

2.2.1.1 Definition

```
struct qgpio {
    uint32_t pin_number;
    uint32_t port;
    void *dev;
}
```

2.2.1.2 Members

pin_number
pin number.

port
port/bank for this GPIO.

dev
User pointer to pass device structure used by HAL.

2.2.2 macro QGPIO_SINGLE_ENDED

`QGPIO_SINGLE_ENDED()`

Configure GPIO output in single-ended mode (open drain or open source).

2.2.3 macro QGPIO_PUSH_PULL

`QGPIO_PUSH_PULL()`

Configure GPIO output in push-pull mode

2.2.4 macro QGPIO_LINE_OPEN_DRAIN

`QGPIO_LINE_OPEN_DRAIN()`

Configure single ended open drain mode (wired AND).

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.2.5 macro QGPIO_LINE_OPEN_SOURCE

`QGPIO_LINE_OPEN_SOURCE()`

Configure single ended open source mode (wired OR).

2.2.6 macro QGPIO_OPEN_DRAIN

`QGPIO_OPEN_DRAIN()`

Configure GPIO output in open drain mode (wired AND).

2.2.7 macro QGPIO_OPEN_SOURCE

`QGPIO_OPEN_SOURCE()`

Configure GPIO output in open source mode (wired OR).

2.2.8 macro QGPIO_PULL_UP

`QGPIO_PULL_UP()`

Configure GPIO pin pull-up.

2.2.9 macro QGPIO_PULL_DOWN

`QGPIO_PULL_DOWN()`

Configure GPIO pin pull-down.

2.2.10 macro QGPIO_SPEED_2MHZ

`QGPIO_SPEED_2MHZ()`

Configure GPIO output slew rate to 2 MHz.

2.2.11 macro QGPIO_SPEED_25MHZ

`QGPIO_SPEED_25MHZ()`

Configure GPIO output slew rate to 25 MHz.

2.2.12 macro QGPIO_SPEED_50MHZ

`QGPIO_SPEED_50MHZ()`

Configure GPIO output slew rate to 50 MHz.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.2.13 macro QGPIO_SPEED_100MHZ

`QGPIO_SPEED_100MHZ()`

Configure GPIO output slew rate to 100 MHz.

2.2.14 macro QGPIO_INPUT

`QGPIO_INPUT()`

Enable pin as input.

2.2.15 macro QGPIO_DIR_MASK

`QGPIO_DIR_MASK()`

GPIO direction mask.

2.2.16 macro QGPIO_OUTPUT_LOW

`QGPIO_OUTPUT_LOW()`

Configure GPIO pin as output and initializes it to a low state.

2.2.17 macro QGPIO_OUTPUT_HIGH

`QGPIO_OUTPUT_HIGH()`

Configure GPIO pin as output and initializes it to a high state.

2.2.18 macro QGPIO_IRQ_EDGE_RISING

`QGPIO_IRQ_EDGE_RISING()`

Configure GPIO interrupt to be triggered on pin rising edge and enables it.

2.2.19 macro QGPIO_IRQ_EDGE_FALLING

`QGPIO_IRQ_EDGE_FALLING()`

Configure GPIO interrupt to be triggered on pin falling edge and enables it.

2.2.20 macro QGPIO_IRQ_EDGE_BOTH

`QGPIO_IRQ_EDGE_BOTH()`

Configure GPIO interrupt to be triggered on pin rising or falling edge and enables it.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.2.21 macro QGPIO_IRQ_LEVEL_LOW

`QGPIO_IRQ_LEVEL_LOW()`

Configure GPIO interrupt to be triggered on pin physical level low and enables it.

2.2.22 macro QGPIO_IRQ_LEVEL_HIGH

`QGPIO_IRQ_LEVEL_HIGH()`

Configure GPIO interrupt to be triggered on pin physical level high and enables it.

2.2.23 macro QGPIO_IRQ_MASK

`QGPIO_IRQ_MASK()`

GPIO interrupt mask.

2.2.24 macro QGPIO_IRQ_DISABLED

`QGPIO_IRQ_DISABLED()`

Disable QGPIO IRQ.

2.2.25 typedef qgpio_irq_cb

void **qgpio_irq_cb**(void *arg)

Pointer to a callback called when interrupt triggered.

Parameters

- **arg** (`void*`) – private data of the GPIO interruption handler.

2.2.25.1 NOTE

Considered this called from ISR context unless indicated otherwise on implementation.

2.2.25.2 Return

nothing.

2.2.26 qgpio_pin_configure

enum qerr **qgpio_pin_configure**(const struct *qgpio* *qgpio_pin, uint32_t flags)

Configure a GPIO.

Parameters

- **qgpio_pin** (`const struct` *`qgpio`*`*`) – Pin descriptor.
- **flags** (`uint32_t`) – Pin configuration flags.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.2.26.1 NOTE

To be called before any others functions of qgpio.

2.2.26.2 Return

QERR_SUCCESS or error.

2.2.27 qgpio_pin_write

enum qerr **`qgpio_pin_write`**(const struct *qgpio* *qgpio_pin, uint8_t value)

Write a GPIO.

Parameters

- **`qgpio_pin`** (`const struct` *`qgpio`*`*`) – Pin descriptor.
- **`value`** (`uint8_t`) – Value to be written.

2.2.27.1 Return

QERR_SUCCESS or error.

2.2.27.2 NOTE

To be called after *`qgpio_pin_configure()`*.

2.2.28 qgpio_pin_read

enum qerr **`qgpio_pin_read`**(const struct *qgpio* *qgpio_pin, uint8_t *value)

Read a GPIO.

Parameters

- **`qgpio_pin`** (`const struct` *`qgpio`*`*`) – Pin descriptor.
- **`value`** (`uint8_t*`) – Pointer to value being read.

2.2.28.1 Return

QERR_SUCCESS or error.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.2.28.2 NOTE

To be called after *qgpio_pin_configure()*.

2.2.29 qgpio_pin_irq_configure

enum qerr **qgpio_pin_irq_configure**(const struct *qgpio* *qgpio_pin, uint32_t flags)

Configure interrupt for GPIO.

Parameters

- **qgpio_pin** (const struct *qgpio**) – Pin descriptor.
- **flags** (uint32_t) – Interrupt pin configuration flags.

2.2.29.1 Return

QERR_SUCCESS or error.

2.2.29.2 NOTE

To be called after *qgpio_pin_irq_set_callback()*.

2.2.30 qgpio_pin_irq_set_callback

enum qerr **qgpio_pin_irq_set_callback**(const struct *qgpio* *qgpio_pin, *qgpio_irq_cb* cb, void *arg)

Set callback function to be called when GPIO interrupt.

Parameters

- **qgpio_pin** (const struct *qgpio**) – Pin descriptor.
- **cb** (*qgpio_irq_cb*) – Callback function pointer to be called in case of interrupt.
- **arg** (void*) – Private data for callback.

2.2.30.1 Return

QERR_SUCCESS or error.

2.2.30.2 NOTE

To be called after *qgpio_pin_configure()* and before *qgpio_pin_irq_configure()*.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.3 QOTP

2.3.1 qotp_read

enum qerr **qotp_read**(uint32_t address, uint32_t *data, uint8_t length)

Read the OTP data from given address into provided array

Parameters

- **address** (uint32_t) – OTP address to read from
- **data** (uint32_t*) – pointer to the array into which to read the data
- **length** (uint8_t) – number of 32 bit words to read (array needs to be at least this length)

2.3.1.1 Return

QERR_SUCCESS or error

2.3.2 qotp_write

enum qerr **qotp_write**(uint32_t value, uint32_t address)

Write the data to given address in OTP memory.

Parameters

- **value** (uint32_t) – data to write
- **address** (uint32_t) – address to write

2.3.2.1 Return

QERR_SUCCESS or error

2.4 QPWR

2.4.1 qpwr_enable_lpm

void **qpwr_enable_lpm**(void)

Enable low power mode.

Parameters

- **void** – no arguments

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.4.2 qpwr_disable_lpm

void **qpwr_disable_lpm**(void)

Disable low power mode.

Parameters

- **void** – no arguments

2.4.3 qpwr_is_lpm_enabled

bool **qpwr_is_lpm_enabled**(void)

Check if low power mode is enabled.

Parameters

- **void** – no arguments

2.4.3.1 Return

True if low power mode is enabled, otherwise false.

2.4.4 qpwr_set_min_inactivity_s4

enum qerr **qpwr_set_min_inactivity_s4**(uint32_t time_ms)

Set the minimum inactivity time to get in S4.

Parameters

- **time_ms** (`uint32_t`) – minimum inactivity time to get in S4, in ms.

2.4.4.1 Return

QERR_SUCCESS or negative QERR error

2.4.5 qpwr_get_min_inactivity_s4

enum qerr **qpwr_get_min_inactivity_s4**(uint32_t *time_ms)

Get the minimum inactivity time to get in S4.

Parameters

- **time_ms** (`uint32_t*`) – minimum inactivity time to get in S4, in ms.

2.4.5.1 Return

QERR_SUCCESS or negative QERR error

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.4.6 qpwr_uwb_sleep

enum qerr **qpwr_uwb_sleep**(void)

Put UWB to sleep. Do nothing if it is already sleeping.

Parameters

- **void** – no arguments

2.4.6.1 Return

QERR_SUCCESS or negative QERR error

2.4.7 qpwr_uwb_wakeup

enum qerr **qpwr_uwb_wakeup**(void)

Wake up UWB from sleep. Do nothing if it is already awake.

Parameters

- **void** – no arguments

2.4.7.1 Return

QERR_SUCCESS or negative QERR error

2.4.8 qpwr_uwb_is_sleeping

bool **qpwr_uwb_is_sleeping**(void)

Check if UWB is sleeping.

Parameters

- **void** – no arguments

2.4.8.1 Return

true if UWB is sleeping, otherwise false.

2.5 QRTC

2.5.1 qrtc_get_us

int64_t **qrtc_get_us**(void)

Get RTC value in microseconds.

Parameters

- **void** – no arguments

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.5.1.1 Return

64-bits RTC value (in microseconds).

2.5.2 qrtc_resync_rtc_systime

void **qrtc_resync_rtc_systime**(int64_t *rtc_us, uint32_t *systime)

Resync RTC and SysTime.

Parameters

- **rtc_us** (int64_t*) – Returned current RTC value (in microseconds).
- **systime** (uint32_t*) – Returned current UWB transceiver SysTime (in ticks).

2.5.3 qrtc_update_rtc_systime

void **qrtc_update_rtc_systime**(int64_t updated_rtc_us, uint32_t updated_systime)

Overwrite current RTC and SysTime used in QRTC.

Parameters

- **updated_rtc_us** (int64_t) – New current RTC value (in microseconds).
- **updated_systime** (uint32_t) – New UWB transceiver SysTime (in ticks).

2.5.3.1 NOTE

that function is used as a workaround when RTC is based on Systime.

2.6 QSPI

2.6.1 struct qspi_instance

struct **qspi_instance**

spi input parameters.

2.6.1.1 Definition

```
struct qspi_instance {
    uint32_t instance_number;
    const void *dev;
}
```

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.1.2 Members

instance_number
SPI instance number. Implementation dependent.

dev
Device pointer. Implementation dependent.

2.6.2 macro QSPI_MASTER

`QSPI_MASTER()`

Master mode. Used in *qspi_config.op_flags* field.

2.6.3 macro QSPI_SLAVE

`QSPI_SLAVE()`

Slave mode. Used in *qspi_config.op_flags* field.

2.6.4 macro QSPI_ROLE_MASK

`QSPI_ROLE_MASK()`

Master/Slave mode mask.

2.6.5 macro QSPI_GET_ROLE

`QSPI_GET_ROLE(flags)`

Get Master/Slave mode.

Parameters

- **flags** – QSPI bitfield flags from *qspi_config.op_flags*.

2.6.5.1 Return

Master/Slave mode

2.6.6 macro QSPI_CPOL

`QSPI_CPOL()`

QSPI clock polarity, 0 to LOW, 1 to HIGH, used in combination with clock phase. Used in *qspi_config.op_flags* field. See *struct QSPI_CPHA*

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.6.1 Description

- `if` QSPI_MODE_CPOL=0 and QSPI_CPHA=0, sample on leading edge.
- `if` QSPI_MODE_CPOL=0 and QSPI_CPHA=1, sample on trailing edge.
- `if` QSPI_MODE_CPOL=1 and QSPI_CPHA=0, sample on leading edge.
- `if` QSPI_MODE_CPOL=1 and QSPI_CPHA=1, sample on trailing edge.

2.6.7 macro QSPI_CPHA

`QSPI_CPHA()`

QSPI clock phase, used in combination with clock polarity. Used in *qspi_config.op_flags* field. See *struct QSPI_CPOL*

2.6.7.1 Description

- `if` QSPI_MODE_CPOL=0 and QSPI_CPHA=0, sample on leading edge.
- `if` QSPI_MODE_CPOL=0 and QSPI_CPHA=1, sample on trailing edge.
- `if` QSPI_MODE_CPOL=1 and QSPI_CPHA=0, sample on leading edge.
- `if` QSPI_MODE_CPOL=1 and QSPI_CPHA=1, sample on trailing edge.

2.6.8 macro QSPI_LOOP

`QSPI_LOOP()`

Loopback mode. Used in *qspi_config.op_flags* field.

2.6.8.1 Description

- `1` - loopback mode enabled.
- `0` - loopback mode disabled.

2.6.9 macro QSPI_MSB_FIRST

`QSPI_MSB_FIRST()`

MSB first for transfers. Used in *qspi_config.op_flags* field.

2.6.10 macro QSPI_LSB_FIRST

`QSPI_LSB_FIRST()`

LSB first for transfers. Used in *qspi_config.op_flags*.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.11 macro QSPI_FRAME_SIZE_SHIFT

`QSPI_FRAME_SIZE_SHIFT()`

Frame size shift in *qspi_config.op_flags*.

2.6.12 macro QSPI_FRAME_SIZE_MASK

`QSPI_FRAME_SIZE_MASK()`

Frame size mask.

2.6.13 macro QSPI_FRAMED_SIZE_GET

`QSPI_FRAMED_SIZE_GET(flags)`

Get frame size from *qspi_config.op_flags*.

Parameters

- **flags** – QSPI bitfield flags from *qspi_config.op_flags*.

2.6.13.1 Return

Frame length in bytes.

2.6.14 macro QSPI_SET_FRAME_LEN

`QSPI_SET_FRAME_LEN(frame_length)`

Set frame size on QSPI_FRAME_SIZE_SHIFT bits for a maximum of 64 bytes, inside *qspi_config.op_flags*.

Parameters

- **frame_length** – Frame length in bytes.

2.6.15 macro QSPI_CS_ACTIVE_HIGH

`QSPI_CS_ACTIVE_HIGH()`

Chip/Slave select pin logic. 1 is CS pin is active HIGH, 0 if active LOW. Used in *qspi_config.op_flags* field.

2.6.16 macro QSPI_MISO_SINGLE

`QSPI_MISO_SINGLE()`

Extended mode for MISO single line. Used in *qspi_config.op_flags*.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.17 macro QSPI_MISO_DUAL

`QSPI_MISO_DUAL()`

Extended mode for MISO dual line. Used in *qspi_config.op_flags*.

2.6.18 macro QSPI_MISO_QUAD

`QSPI_MISO_QUAD()`

Extended mode for MISO quad line. Used in *qspi_config.op_flags*.

2.6.19 macro QSPI_MISO_OCTAL

`QSPI_MISO_OCTAL()`

Extended mode for MISO octal line. Used in *qspi_config.op_flags*.

2.6.20 macro QSPI_LINES_MASK

`QSPI_LINES_MASK()`

Extended mode mask.

2.6.21 macro QSPI_HALF_DUPLEX

`QSPI_HALF_DUPLEX()`

QSPI half duplex transaction mode. Used in *qspi_config.op_flags*.

2.6.22 macro QSPI_FULL_DUPLEX

`QSPI_FULL_DUPLEX()`

QSPI full duplex transaction mode. Used in *qspi_config.op_flags*.

2.6.23 macro QSPI_DUPLEX_MASK

`QSPI_DUPLEX_MASK()`

QSPI transaction mode mask.

2.6.24 struct qspi_config

struct **qspi_config**

QSPI configuration.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.24.1 Definition

```
struct qspi_config {
    struct qgpio sck_pin;
    struct qgpio cs_pin;
    struct qgpio mosi_pin;
    struct qgpio miso_pin;
    uint32_t freq_hz;
    uint8_t irq_priority;
    uint32_t op_flags;
}
```

2.6.24.2 Members

sck_pin
Serial clock pin descriptor.

cs_pin
Chip select pin descriptor.

mosi_pin
MOSI pin descriptor.

miso_pin
MISO pin descriptor.

freq_hz
SPI frequency in Hz.

irq_priority
SPI IRQ priority.

op_flags
Bit field operation flags.

2.6.25 struct qspi_transfer

struct **qspi_transfer**
QSPI transfer data.

2.6.25.1 Definition

```
struct qspi_transfer {
    uint8_t *tx_buf;
    uint8_t *rx_buf;
    uint32_t tx_size;
    uint32_t rx_size;
    uint32_t flags;
}
```

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.25.2 Members

tx_buf
: Pointer to transmit buffer.

rx_buf
: Pointer to receive buffer.

tx_size
: Size of transmit buffer.

rx_size
: Size of receive buffer.

flags
: Specific transfer flags. Implementation dependent.

2.6.26 typedef qspi_xfer_cb

void **qspi_xfer_cb**(void *arg)

Pointer to callback function.

Parameters

- **arg** (void*) – private data returned after transfer.

2.6.26.1 Return

nothing.

2.6.27 qspi_open

struct qspi ***qspi_open**(const struct *qspi_instance* *instance)

Create a SPI descriptor for a given SPI instance.

Parameters

- **instance** (const struct *qspi_instance**) – SPI instance to open. How to fill it is implementation dependent.

2.6.27.1 NOTE

This function should be called first.

2.6.27.2 Return

Pointer to the initialized SPI descriptor.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.28 qspi_close

enum qerr **qspi_close**(struct qspi *spi)

Destroy a SPI descriptor.

Parameters

- **spi** (struct qspi*) – Pointer to the SPI descriptor.

2.6.28.1 Return

QERR_SUCCESS or error.

2.6.29 qspi_configure

enum qerr **qspi_configure**(struct qspi *spi, const struct *qspi_config* *config)

Initialize and configure an SPI device.

Parameters

- **spi** (struct qspi*) – Pointer to the SPI descriptor.
- **config** (const struct *qspi_config**) – Pointer to SPI configuration.

2.6.29.1 NOTE

This function should be called before any transfers.

2.6.29.2 Return

QERR_SUCCESS or error.

2.6.30 qspi_irq_set_callback

enum qerr **qspi_irq_set_callback**(struct qspi *spi, *qspi_xfer_cb* handler, void *arg)

Set callback function to be called when SPI IRQ.

Parameters

- **spi** (struct qspi*) – Pointer to the SPI descriptor.
- **handler** (*qspi_xfer_cb*) – Callback to call when xfer done in async mode. If NULL, synchronous mode is used.
- **arg** (void*) – Private data of the xfer done callback.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.6.30.1 Return

QERR_SUCCESS or error.

2.6.31 qspi_transceive

enum qerr **qspi_transceive**(struct qspi *spi, const struct *qspi_transfer* *xfer)

Make a SPI transaction.

Parameters

- **spi** (struct qspi*) – Pointer to the SPI descriptor.
- **xfer** (const struct *qspi_transfer**) – Pointer to the SPI transfer data.

2.6.31.1 Return

QERR_SUCCESS or error.

2.7 QTIMER

2.7.1 struct qtimer_config

struct **qtimer_config**

Timer configuration.

2.7.1.1 Definition

```
struct qtimer_config {
    uint32_t freq_hz;
    uint8_t irq_priority;
    enum qtimer_width width;
}
```

2.7.1.2 Members

freq_hz
Timer frequency in Hz.

irq_priority
Timer IRQ priority.

width
Timer bit width.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.7.2 typedef qtimer_cb

void **qtimer_cb**(void *arg)

Pointer to a timer callback.

Parameters

- **arg** (void*) – private data of the timer.

2.7.2.1 Return

nothing.

2.7.3 qtimer_init

struct qtimer ***qtimer_init**(uint8_t qtimer_id, const struct *qtimer_config* *config, *qtimer_cb* handler, void *arg)

Initialize and configure a TIMER device.

Parameters

- **qtimer_id** (uint8_t) – ID of the timer. Implementation and hardware dependent.
- **config** (const struct *qtimer_config**) – Pointer to timer configuration.
- **handler** (*qtimer_cb*) – Callback to called when timer ends.
- **arg** (void*) – Private data of the timer callback.

2.7.3.1 NOTE

This function should be called first.

2.7.3.2 Return

Pointer to the initialized timer descriptor.

2.7.4 qtimer_start

enum qerr **qtimer_start**(const struct qtimer *timer, uint32_t us, bool periodic)

Start a timer.

Parameters

- **timer** (const struct qtimer*) – Pointer to the timer descriptor.
- **us** (uint32_t) – Number of microseconds until timer expiration.
- **periodic** (bool) – true for a one-shot timer, false for a cyclic timer.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

2.7.4.1 Return

QERR_SUCCESS or error.

2.7.5 qtimer_stop

enum qerr **`qtimer_stop`**(const struct qtimer *timer)

Start a timer.

Parameters

- **`timer`** (`const struct qtimer*`) – Pointer to the timer descriptor.

2.7.5.1 Return

QERR_SUCCESS or error.

2.7.6 qtimer_read

enum qerr **`qtimer_read`**(const struct qtimer *timer, uint32_t *us)

Read qtimer value.

Parameters

- **`timer`** (`const struct qtimer*`) – Pointer to the timer descriptor.
- **`us`** (`uint32_t*`) – Elapsed timer in µs.

2.7.6.1 Return

QERR_SUCCESS or error.

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information

Index

© 2025 Qorvo US, Inc.
Qorvo® Proprietary Information