I2C Driver

Version 1.0.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 i2c_config_t Struct Reference

`#include <i2c_stm32f411_config.h>`

Data Fields

- i2c_control_t en
- i2c_ack_en_t ack_en
- uint32_t periph_clk_freq_MHz
- uint32_t i2c_op_freq_kHz
- i2c_fast_slow_t fast_or_std
- i2c_fm_duty_cycle_t duty_cycle

3.1.1 Detailed Description

Struct contains the settings required to configure an i2c device.

3.1.2 Field Documentation

3.1.2.1 ack_en

`i2c_ack_en_t ack_en`

Whether the device sends ACK upon byte reception

3.1.2.2 duty_cycle

`i2c_fm_duty_cycle_t duty_cycle`

The ratio of the period of low vs high cycles of bit pulses

3.1.2.3 en

```
i2c_control_t en
```

Whether the device is enabled or not

3.1.2.4 fast_or_std

```
i2c_fast_slow_t fast_or_std
```

Whether the I2C device will be in fast or standard mode

3.1.2.5 i2c_op_freq_kHz

```
uint32_t i2c_op_freq_kHz
```

The operational frequency of the I2C bus

3.1.2.6 periph_clk_freq_MHz

```
uint32_t periph_clk_freq_MHz
```

The frequency of the device in MHz

The documentation for this struct was generated from the following file:

- /home/marko/Documents/embedded_workspace/i2c_driver/i2c_stm32f411_config.h

3.2 i2c_transfer_t Struct Reference

```
#include <i2c_interface.h>
```

Data Fields

- i2c_channel_t channel
- uint8_t * buffer
- uint32_t data_length
- uint8_t slave_address

3.2.1 Detailed Description

Generic transfer structure, independent of implementation. Passed into transmission functions.

3.2.2 Field Documentation

3.2.2.1 buffer

```
uint8_t* buffer
```

The data buffer

3.2.2.2 channel

```
i2c_channel_t channel
```

The target I2C peripheral

3.2.2.3 data_length

```
uint32_t data_length
```

The number of bytes to be receive/sent

3.2.2.4 slave_address

```
uint8_t slave_address
```

The 7-bit slave address

The documentation for this struct was generated from the following file:

- /home/marko/Documents/embedded_workspace/i2c_driver/i2c_interface.h

Chapter 4

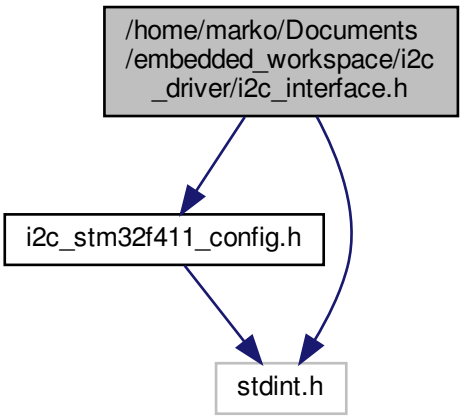
File Documentation

4.1 /home/marko/Documents/embedded_workspace/i2c_driver/i2c_interface.h File Reference

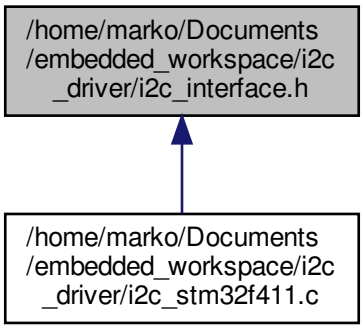
General interface covering user accesses to the i2c communication bus.

```
#include "i2c_stm32f411_config.h"
#include <stdint.h>
```

Include dependency graph for i2c_interface.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct i2c_transfer_t

Enumerations

- enum **i2c_interrupt_control_t** { **INTERRUPT_DISABLED**, **INTERRUPT_ENABLED** }

Functions

- void i2c_init (const i2c_config_t *config_table)
- void i2c_irq_handler (i2c_channel_t channel)
- void i2c_control (i2c_channel_t channel, i2c_control_t signal)
- void i2c_interrupt_control (i2c_channel_t channel, i2c_interrupt_dma_t interrupt, i2c_interrupt_control_t signal)
- void i2c_master_transmit (i2c_transfer_t *transfer)
- void i2c_master_receive (i2c_transfer_t *transfer)
- void i2c_slave_transmit (i2c_transfer_t *transfer)
- void i2c_slave_receive (i2c_transfer_t *transfer)
- uint32_t i2c_master_transmit_it (i2c_transfer_t *transfer)
- uint32_t i2c_master_receive_it (i2c_transfer_t *transfer)
- uint32_t i2c_slave_transmit_it (i2c_transfer_t *transfer)
- uint32_t i2c_slave_receive_it (i2c_transfer_t *transfer)
- void **i2c_register_write** (uint32_t i2c_register)
- uint32_t i2c_register_read (uint32_t i2c_regsister)

4.1.1 Detailed Description

General interface covering user accesses to the i2c communication bus.

4.1.2 Function Documentation

4.1.2.1 i2c_control()

```
void i2c_control (
            i2c_channel_t channel,
            i2c_control_t signal )
```

Description:

```
"Emergency" function used to enable or disable a desired i2c channel.
I2C channels that have been enabled through the config table are enabled
automatically after initialisation.
Mainly used to change configs at runtime.
```

PRE-CONDITION: The i2c_init function has been carried out successfully

POST-CONDITION: The desired i2c device has been activated/disabled

Returns

void

Example:

```
i2c_control(I2C_2, I2C_DISABLED);
```

See also

i2c_init
i2c_interrupt_control
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.2 i2c_init()

```
void i2c_init (
            const i2c_config_t * config_table )
```

Description:

```
Carries out the initialisation of the I2C channels as per the information
in the config table
```

PRE-CONDITION: The config table has been obtained and is non-null PRE-CONDITION: The required GPIO pins for i2c combination have been configured correctly with gpio_init PRE-CONDITION: The appropriate peripheral clocks have been activated

POST-CONDITION: The selected i2c channels have been activated anda ready to be used

Returns

void

Example:

```
const i2c_config_t *config_table = i2c_config_get();
i2c_init(config_table);
```

See also

i2c_config_get
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.3 i2c_interrupt_control()

```
void i2c_interrupt_control (
            i2c_channel_t channel,
            i2c_interrupt_dma_t interrupt,
            i2c_interrupt_control_t signal )
```

Description:

```
Enabled or disables the selected interrupt on the selected channel. Caled both by users
and within the driver itself
```

PRE-CONDITION: The i2c_init function has been carried out successfully

POST-CONDITION: The desired interrupt on the selected device has been activated/disabled

Returns

void

Example:

```
i2c_interrupt_control(I2C_2, IT_BUF, INTERRUPT_ENABLED);
```

See also

i2c_init
i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.4 i2c_irq_handler()

```
void i2c_irq_handler (
            i2c_channel_t channel )
```

Description:

```
User level function, to be placed within IRQ handlers at the main.c level.
When called, it checks callback table for the appropriate i2c. If it isn't
null, it feeds a pointer to the appropriate transfer to said callback.
```

PRE-CONDITION: None

POST-CONDITION: The previously mapped callback has been called.

Parameters

channel	refers to any i2c device on chip

Returns

void

Example:

```
called automatically within IRQ handlers, e.g.
I2C1_EV_IRQHandler(void)
{
    i2c_irq_handler(I2C_1);
}
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.5 i2c_master_receive()

```
void i2c_master_receive (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking reception in master mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been received from the slave

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t accelerometer_comm;
accelerometer_comm.channel = I2C_2;
accelerometer_comm.buffer = &data_from_accel;
accelerometer_comm.length = 2;
accelerometer_comm.address = ACCELEROMETER_ADDRESS;
i2c_master_receive(&accelerometer_comm);
```

See also

i2c_init
i2c_master_transmit
i2c_slave_transmit
i2c_slave_receive
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.6 i2c_master_receive_it()

```
uint32_t i2c_master_receive_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based reception.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, enables interrupts, and sends the slave address.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_master_receive_it(&accelerometer_comm);
```

See also

i2c_master_transmit_it
i2c_master_receive_it_callback
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.7 i2c_master_transmit()

```
void i2c_master_transmit (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking transmission in master mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been sent to the slave

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t motor_controller_comm;
motor_controller_comm.channel = I2C_2;
motor_controller_comm.buffer = &data_to_motor;
motor_controller_comm.length = 4;
motor_controller_comm.address = MOTOR_CONTROLLER_ADDRESS;
i2c_master_transmit(&motor_controller_comm);
```

See also

i2c_init
i2c_master_receive
i2c_slave_transmit
i2c_slave_receive
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.8 i2c_master_transmit_it()

```
uint32_t i2c_master_transmit_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based transmission.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, enables interrupts, and sends the slave address.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t motor_controller_comm;
....
  i2c_master_transmit_it(&motor_controller_comm);
```

See also

i2c_master_transmit_it_callback
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.9 i2c_register_read()

```
uint32_t i2c_register_read (
            uint32_t i2c_register )
```

Description:

```
Read the current value of the register in i2c address space. It is the
user's own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside i2c_register_write() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The register's current contents are returned

Parameters

i2c_register	is a uint32_t which is cast as a 16bit address

Returns

uint16_t timer_register's contents

Example:

```
uint32_t cr1_i2c3 = i2c_register_read(I2C3_BASE + 0x00UL); //get current value
cr1_i2c3 &= ~(0x01UL << I2C_CR1_ACK_Pos); //clear the DMA request on CC3 bit
timer_register_write(I2C3_BASE + 0x00UL, cr1_i2c3);
```

See also

i2c_register_write
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.10 i2c_slave_receive()

```
void i2c_slave_receive (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking reception in master mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been received by the master from the slave

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t accelerometer_comm;
accelerometer_comm.channel = I2C_2;
accelerometer_comm.buffer = &data_from_accel;
accelerometer_comm.length = 2;
i2c_slave_receive(&accelerometer_comm);
```

See also

i2c_init
i2c_master_transmit
•
i2c_master_receive
i2c_slave_transmit
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.11 i2c_slave_receive_it()

```
uint32_t i2c_slave_receive_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based transmission.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, and enables interrupts.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_slave_receive_it(&accelerometer_comm);
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it_callback
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.12 i2c_slave_transmit()

```
void i2c_slave_transmit (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking transmission in slave mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been sent to the master

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t motor_controller_comm;
motor_controller_comm.channel = I2C_2;
motor_controller_comm.buffer = &data_to_motor;
motor_controller_comm.length = 4;
i2c_slave_transmit(&motor_controller_comm);
```

See also

i2c_init
i2c_master_transmit
i2c_master_receive
i2c_slave_receive
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.1.2.13 i2c_slave_transmit_it()

```
uint32_t i2c_slave_transmit_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based transmission.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, and enables interrupts.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t motor_controller_comm;
....
  i2c_slave_transmit_it(&motor_controller_comm);
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it_callback
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

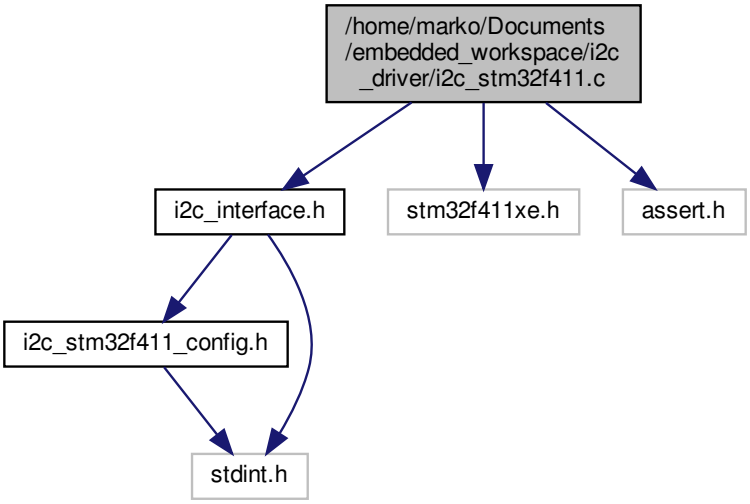
Date	Software Version	Initials	Description

4.2 /home/marko/Documents/embedded_workspace/i2c_driver/i2c_stm32f411.c File Reference

Chip specific implementation for i2c communication.

```
#include "i2c_interface.h"
#include "stm32f411xe.h"
#include <assert.h>
```

Include dependency graph for i2c_stm32f411.c:



Macros

- #define SM_RISE_TIME_MAX 1000
- #define FM_RISE_TIME_MAX 300
- #define **NULL** (void *)0

Typedefs

- typedef void(* i2c_interrupt_callback_t) (i2c_transfer_t *)

Functions

- static uint32_t i2c_calculate_ccr (i2c_config_t *config_entry)
- static uint32_t i2c_calculate_trise (i2c_config_t *config_entry)
- static void i2c_clear_addr_bit (i2c_channel_t channel)
- static void i2c_handle_stopf_flag (i2c_channel_t channel)
- static void i2c_one_byte_reception (i2c_transfer_t *i2c_transfer)
- static void i2c_two_byte_reception (i2c_transfer_t *i2c_transfer)
- static void i2c_n_byte_reception (i2c_transfer_t *i2c_transfer)
- void i2c_init (const i2c_config_t *config_table)
- void i2c_control (i2c_channel_t channel, i2c_control_t signal)
- void i2c_interrupt_control (i2c_channel_t channel, i2c_interrupt_dma_t interrupt, i2c_interrupt_control_t signal)
- void i2c_master_transmit (i2c_transfer_t *i2c_transfer)
- void i2c_master_receive (i2c_transfer_t *i2c_transfer)
- void i2c_slave_transmit (i2c_transfer_t *i2c_transfer)
- void i2c_slave_receive (i2c_transfer_t *i2c_transfer)
- static void i2c_master_transmit_it_callback (i2c_transfer_t *i2c_transfer)

- static void i2c_master_receive_it_callback (i2c_transfer_t *i2c_transfer)
- static void i2c_slave_transmit_it_callback (i2c_transfer_t *i2c_transfer)
- static void i2c_slave_receive_it_callback (i2c_transfer_t *i2c_transfer)
- uint32_t i2c_master_transmit_it (i2c_transfer_t *i2c_transfer)
- uint32_t i2c_master_receive_it (i2c_transfer_t *i2c_transfer)
- uint32_t i2c_slave_transmit_it (i2c_transfer_t *i2c_transfer)
- uint32_t i2c_slave_receive_it (i2c_transfer_t *i2c_transfer)
- void i2c_irq_handler (i2c_channel_t channel)
- void i2c_register_write (uint32_t i2c_register, uint16_t value)
- uint16_t i2c_register_read (uint32_t i2c_register)

Variables

- static volatile uint16_t *const I2C_CR1 [NUM_I2C]
- static volatile uint16_t *const I2C_CR2 [NUM_I2C]
- static volatile uint16_t *const I2C_OAR1 [NUM_I2C]
- static volatile uint16_t *const I2C_OAR2 [NUM_I2C]
- static volatile uint16_t *const I2C_DR [NUM_I2C]
- static volatile uint16_t *const I2C_SR1 [NUM_I2C]
- static volatile uint16_t *const I2C_SR2 [NUM_I2C]
- static volatile uint16_t *const I2C_CCR [NUM_I2C]
- static volatile uint16_t *const I2C_TRISE [NUM_I2C]
- static volatile uint16_t *const I2C_FLTR [NUM_I2C]
- static i2c_transfer_t i2c_interrupt_transfers [NUM_I2C]
- static i2c_interrupt_callback_t i2c_interrupt_callbacks [NUM_I2C]

4.2.1 Detailed Description

Chip specific implementation for i2c communication.

4.2.2 Macro Definition Documentation

4.2.2.1 FM_RISE_TIME_MAX

```
#define FM_RISE_TIME_MAX 300
```

Maximum rise time for a fast mode pulse in ns.

4.2.2.2 SM_RISE_TIME_MAX

```
#define SM_RISE_TIME_MAX 1000
```

Rise times obtained from the phillips i2c spec sheetMaximum rise time for a stanard mode pulse in ns.

4.2.3 Typedef Documentation

4.2.3.1 i2c_interrupt_callback_t

```
typedef void(* i2c_interrupt_callback_t) (i2c_transfer_t *)
```

Callback typedef for interrupt callbacks

4.2.4 Function Documentation

4.2.4.1 i2c_calculate_ccr()

```
static uint32_t i2c_calculate_ccr (
            i2c_config_t * config_entry )  [inline], [static]
```

Description:

```
Static inline function called from within the driver to carry out the calculation of
the required pulse length for a given frequency.
```

PRE-CONDITION: The config table has been obtained and is non-null

POST-CONDITION: The appropriate cc value has been calculated and placed in the register

Returns

uint32_t

Example: Automatically called within i2c_init

See also

i2c_init
i2c_calculate_trise
- CHANGE HISTORY -

Date	Software Version	Initials	Description

Equation obtained from RM0383 18.6.8: $CCR = \frac{T_{high}}{T_{pclk}}$, where $T_{high} = \frac{1}{2 \times T_{opfreq}}$ (kHz) and $T_{opfreq} = \frac{1}{f_{opfreq}}$ and $T_{pclk} = \frac{1}{f_{pclk}}$ leading to $CCR = \frac{f_{pclk}(MHz)}{2 \times f_{opfreq}(khZ)} = \frac{f_{pclk}}{2000 \times f_{opfreq}}$

Equation obtained from RM0383 18.6.8: $CCR = \frac{T_{high}}{T_{pclk}}$, where $T_{high} = \frac{1}{3 \times T_{opfreq}}$ (kHz) and $T_{opfreq} = \frac{1}{f_{opfreq}}$ and $T_{pclk} = \frac{1}{f_{pclk}}$ leading to $CCR = \frac{f_{pclk}(MHz)}{3 \times f_{opfreq}(khZ)} = \frac{f_{pclk}}{3000 \times f_{opfreq}}$

Equation obtained from RM0383 18.6.8: $CCR = \frac{T_{high}}{T_{pclk}}$, where $T_{high} = \frac{25}{9 \times T_{opfreq}}$ (kHz) and $T_{opfreq} = \frac{1}{f_{opfreq}}$ and $T_{pclk} = \frac{1}{f_{pclk}}$ leading to $CCR = \frac{9 \times f_{pclk}(MHz)}{25 \times f_{opfreq}(khZ)} = \frac{9 \times f_{pclk}}{25000 \times f_{opfreq}}$

4.2.4.2 i2c_calculate_trise()

```
static uint32_t i2c_calculate_trise (
            i2c_config_t * config_entry )  [inline], [static]
```

Description:

```
Static inline function called from within the driver to carry out the calculation of
the required rise time
```

PRE-CONDITION: The config table has been obtained and is non-null

POST-CONDITION: The appropriate trise value has been calculated and placed in the register

Returns

uint32_t

Example: Automatically called within i2c_init

See also

i2c_init
i2c_calculate_ccr
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.3 i2c_clear_addr_bit()

```
static void i2c_clear_addr_bit (
            i2c_channel_t channel )  [static]
```

Description:

```
Static function called during transfer functions to clear the address bit
through a read to the SR1 register.
```

PRE-CONDITION: None

POST-CONDITION: The addr bit in SR1 for the appropriate channel has been cleared.

Parameters

channel	points to the appropriate i2c channel

Returns

void

Example:

```
called automatically by various transmission functions
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.4 i2c_control()

```
void i2c_control (
            i2c_channel_t channel,
            i2c_control_t signal )
```

Description:

```
"Emergency" function used to enable or disable a desired i2c channel.
I2C channels that have been enabled through the config table are enabled
automatically after initialisation.
Mainly used to change configs at runtime.
```

PRE-CONDITION: The i2c_init function has been carried out successfully

POST-CONDITION: The desired i2c device has been activated/disabled

Returns

void

Example:

```
i2c_control(I2C_2, I2C_DISABLED);
```

See also

i2c_init
i2c_interrupt_control
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.5 i2c_handle_stopf_flag()

```
static void i2c_handle_stopf_flag (
            i2c_channel_t channel )  [static]
```

Description:

```
Static function called during transfer functions to clear the stopf bit
through a read to the SR1 register, and then initiating a stop.
```

PRE-CONDITION: None

POST-CONDITION: The stopf bit in SR1 for the appropriate channel has been cleared, and the communication stopped with a write to CR1

Parameters

channel	points to the appropriate i2c channel

Returns

void

Example:

```
called automatically by various transmission functions
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.6 i2c_init()

```
void i2c_init (
            const i2c_config_t * config_table )
```

Description:

```
Carries out the initialisation of the I2C channels as per the information
in the config table
```

PRE-CONDITION: The config table has been obtained and is non-null PRE-CONDITION: The required GPIO pins for i2c combination have been configured correctly with gpio_init PRE-CONDITION: The appropriate peripheral clocks have been activated

POST-CONDITION: The selected i2c channels have been activated anda ready to be used

Returns

void

Example:

```
const i2c_config_t *config_table = i2c_config_get();
i2c_init(config_table);
```

See also

i2c_config_get
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.7 i2c_interrupt_control()

```
void i2c_interrupt_control (
            i2c_channel_t channel,
            i2c_interrupt_dma_t interrupt,
            i2c_interrupt_control_t signal )
```

Description:

```
Enabled or disables the selected interrupt on the selected channel. Caled both by users
and within the driver itself
```

PRE-CONDITION: The i2c_init function has been carried out successfully

POST-CONDITION: The desired interrupt on the selected device has been activated/disabled

Returns

void

Example:

```
i2c_interrupt_control(I2C_2, IT_BUF, INTERRUPT_ENABLED);
```

See also

i2c_init
i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.8 i2c_irq_handler()

```
void i2c_irq_handler (
            i2c_channel_t channel )
```

Description:

```
User level function, to be placed within IRQ handlers at the main.c level.
When called, it checks callback table for the appropriate i2c. If it isn't
null, it feeds a pointer to the appropriate transfer to said callback.
```

PRE-CONDITION: None

POST-CONDITION: The previously mapped callback has been called.

Parameters

channel	refers to any i2c device on chip

Returns

void

Example:

```
called automatically within IRQ handlers, e.g.
I2C1_EV_IRQHandler(void)
{
    i2c_irq_handler(I2C_1);
}
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.9 i2c_master_receive()

```
void i2c_master_receive (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking reception in master mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been received from the slave

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t accelerometer_comm;
accelerometer_comm.channel = I2C_2;
accelerometer_comm.buffer = &data_from_accel;
accelerometer_comm.length = 2;
accelerometer_comm.address = ACCELEROMETER_ADDRESS;
i2c_master_receive(&accelerometer_comm);
```

See also

i2c_init
i2c_master_transmit
i2c_slave_transmit
i2c_slave_receive
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.10 i2c_master_receive_it()

```
uint32_t i2c_master_receive_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based reception.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, enables interrupts, and sends the slave address.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_master_receive_it(&accelerometer_comm);
```

See also

i2c_master_transmit_it
i2c_master_receive_it_callback
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.11 i2c_master_receive_it_callback()

```
static void i2c_master_receive_it_callback (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
This callback function is mapped to an i2c device through calling the
master_receive_it function. The function is then called by i2c_irq_handler
whenever the i2c generates an interrupt
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The interrupt has been processed. One or two bytes have been received, or the communication has been halted and interrupts disabled. POST-CONDITION: When the communication is over, the function unmaps itself from the i2c channel

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

void

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_master_receive_it(&accelerometer_comm);

  ....
  //automatically called by
  i2c_irq_handler()
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.12 i2c_master_transmit()

```
void i2c_master_transmit (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking transmission in master mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been sent to the slave

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t motor_controller_comm;
motor_controller_comm.channel = I2C_2;
motor_controller_comm.buffer = &data_to_motor;
motor_controller_comm.length = 4;
motor_controller_comm.address = MOTOR_CONTROLLER_ADDRESS;
i2c_master_transmit(&motor_controller_comm);
```

See also

i2c_init
i2c_master_receive
i2c_slave_transmit
i2c_slave_receive
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.13 i2c_master_transmit_it()

```
uint32_t i2c_master_transmit_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based transmission.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, enables interrupts, and sends the slave address.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t motor_controller_comm;
....
  i2c_master_transmit_it(&motor_controller_comm);
```

See also

i2c_master_transmit_it_callback
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.14 i2c_master_transmit_it_callback()

```
static void i2c_master_transmit_it_callback (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
This callback function is mapped to an i2c device through calling the
master_transmit_it function. The function is then called by i2c_irq_handler
whenever the i2c generates an interrupt
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The interrupt has been processed. One or two bytes have been sent, or the communication has been halted and interrupts disabled. POST-CONDITION: When the communication is over, the function un-maps itself from the i2c channel

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

void

Example:

```
  i2c_init(config_table);
  i2c_transfer_t motor_controller_comm;
....
  i2c_master_transmit_it(&motor_controller_comm);

  ....
  //automatically called by
  i2c_irq_handler()
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.15 i2c_n_byte_reception()

```
static void i2c_n_byte_reception (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
Static function called during transfer functions to handle the reception
of n bytes, as per RM 00383 18.3.3
```

PRE-CONDITION: None

POST-CONDITION: The ACK has been disabled, ADDR bit cleared, the bytes have been received, STOP has been set.

Parameters

i2c_transfer	contains all the information required to carry out any kind of transfer

Returns

void

Example:

```
called automatically by blocking reception functions
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.16 i2c_one_byte_reception()

```
static void i2c_one_byte_reception (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
Static function called during transfer functions to handle the reception
of single bytes, as per RM 00383 18.3.3
```

PRE-CONDITION: None

POST-CONDITION: The ACK has been disabled, ADDR bit cleared, the byte has been received, STOP has been set.

Parameters

i2c_transfer	contains all the information required to carry out any kind of transfer

Returns

void

Example:

```
called automatically by blocking reception functions
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.17 i2c_register_read()

```
uint16_t i2c_register_read (
             uint32_t i2c_register )
```

Description:

```
Read the current value of the register in i2c address space. It is the
user's own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside i2c_register_write() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The register's current contents are returned

Parameters

i2c_register	is a uint32_t which is cast as a 16bit address

Returns

uint16_t timer_register's contents

Example:

```
uint32_t cr1_i2c3 = i2c_register_read(I2C3_BASE + 0x00UL); //get current value
cr1_i2c3 &= ~(0x01UL << I2C_CR1_ACK_Pos); //clear the DMA request on CC3 bit
timer_register_write(I2C3_BASE + 0x00UL, cr1_i2c3);
```

See also

i2c_register_write
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.18 i2c_register_write()

```
void i2c_register_write (
            uint32_t i2c_register,
            uint16_t value )
```

Description:

```
Write the value into the register in i2c address space. It is the
user's own responisibility to consult the RM0383 to ensure that no reserved bits are
overwritten, etc.
Intended to be used alongside i2c_register_read() to create composite advanced user functions
```

PRE-CONDITION: The address does in fact lie in the address space of any timer.

POST-CONDITION: The register now contains the value

Parameters

i2c_register	is a uint32_t which is cast as a 16bit address
value	is a 16 bit value

Example:

```
uint32_t cr1_i2c3 = i2c_register_read(I2C3_BASE + 0x00UL); //get current value
cr1_i2c3 &= ~(0x01UL << I2C_CR1_ACK_Pos); //clear the DMA request on CC3 bit
timer_register_write(I2C3_BASE + 0x00UL, cr1_i2c3);
```

See also

i2c_register_read
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.19 i2c_slave_receive()

```
void i2c_slave_receive (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking reception in master mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been received by the master from the slave

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t accelerometer_comm;
accelerometer_comm.channel = I2C_2;
accelerometer_comm.buffer = &data_from_accel;
accelerometer_comm.length = 2;
i2c_slave_receive(&accelerometer_comm);
```

See also

i2c_init
i2c_master_transmit

-

i2c_master_receive
i2c_slave_transmit
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.20 i2c_slave_receive_it()

```
uint32_t i2c_slave_receive_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based transmission.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, and enables interrupts.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_slave_receive_it(&accelerometer_comm);
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it_callback
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.21 i2c_slave_receive_it_callback()

```
static void i2c_slave_receive_it_callback (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
This callback function is mapped to an i2c device through calling the
slave_receive_it function. The function is then called by i2c_irq_handler
whenever the i2c generates an interrupt
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The interrupt has been processed. One or two bytes have been received, or the communication has been halted and interrupts disabled. POST-CONDITION: When the communication is over, the function unmaps itself from the i2c channel

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

void

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_slave_receive_it(&accelerometer_comm);

  ....
  //automatically called by
  i2c_irq_handler()
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.22 i2c_slave_transmit()

```
void i2c_slave_transmit (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
Initiates a blocking transmission in slave mode using the parameters specified in transfer
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The data has been sent to the master

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer.

Returns

void

Example:

```
i2c_init(config_table);
i2c_transfer_t motor_controller_comm;
motor_controller_comm.channel = I2C_2;
motor_controller_comm.buffer = &data_to_motor;
motor_controller_comm.length = 4;
i2c_slave_transmit(&motor_controller_comm);
```

See also

i2c_init
i2c_master_transmit
i2c_master_receive
i2c_slave_receive
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.23 i2c_slave_transmit_it()

```
uint32_t i2c_slave_transmit_it (
            i2c_transfer_t * i2c_transfer )
```

Description:

```
User level function used to initiate an interrupt based transmission.
Creates a safe (static) copy of the transmission data, maps the appropriate
callback to the i2c channel, and enables interrupts.
The rest of the communication is managed by the callback.
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero. PRE-CONDITION: No other interrupt based transmission is currently running on the same channel

POST-CONDITION: The interrupt transmission is ready to continue upon the next interrupt

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

uint32_t returns 0 if all is good, 1 if another transfer is ongoing.

Example:

```
  i2c_init(config_table);
  i2c_transfer_t motor_controller_comm;
....
  i2c_slave_transmit_it(&motor_controller_comm);
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it_callback
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.24 i2c_slave_transmit_it_callback()

```
static void i2c_slave_transmit_it_callback (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
This callback function is mapped to an i2c device through calling the
slave_transmit_it function. The function is then called by i2c_irq_handler
whenever the i2c generates an interrupt
```

PRE-CONDITION: i2c_init has been carried out properly PRE-CONDITION: The data buffer points to non-null location and the transfer length is non-zero.

POST-CONDITION: The interrupt has been processed. One or two bytes have been sent, or the communication has been halted and interrupts disabled. POST-CONDITION: When the communication is over, the function un-maps itself from the i2c channel

Parameters

i2c_transfer	is a pointer to a struct which contains all the information required to carry out a transfer. In this case it points to the i2c_interrupt_transfers copy, which is safe from the application layer.

Returns

void

Example:

```
  i2c_init(config_table);
  i2c_transfer_t accelerometer_comm;
....
  i2c_slave_transmit_it(&accelerometer_comm);

  ....
  //automatically called by
  i2c_irq_handler()
```

See also

i2c_master_transmit_it
i2c_master_receive_it
i2c_slave_transmit_it
i2c_slave_receive_it
i2c_irq_handler
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.4.25 i2c_two_byte_reception()

```
static void i2c_two_byte_reception (
            i2c_transfer_t * i2c_transfer )  [static]
```

Description:

```
Static function called during transfer functions to handle the reception
of two bytes, as per RM 00383 18.3.3
```

PRE-CONDITION: None

POST-CONDITION: The ACK has been disabled, the POS bit is set, ADDR bit cleared, the bytes have been received, STOP has been set.

Parameters

i2c_transfer	contains all the information required to carry out any kind of transfer

Returns

void

Example:

```
called automatically by blocking reception functions
```

- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.2.5 Variable Documentation

4.2.5.1 I2C_CCR

```
volatile uint16_t* const I2C_CCR[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x1CUL, (uint16_t *)I2C2_BASE + 0x1CUL,
    (uint16_t *)I2C3_BASE + 0x1CUL
}
```

Array of pointers to the clock control registers

4.2.5.2 I2C_CR1

```
volatile uint16_t* const I2C_CR1[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE, (uint16_t *)I2C2_BASE, (uint16_t *)I2C3_BASE
}
```

Array of pointers to the Control Register 1 registers

4.2.5.3 I2C_CR2

```
volatile uint16_t* const I2C_CR2[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x04UL, (uint16_t *)I2C2_BASE + 0x04UL,
    (uint16_t *)I2C3_BASE + 0x04UL

}
```

Array of pointers to the Control Register 2 registers

4.2.5.4 I2C_DR

```
volatile uint16_t* const I2C_DR[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x10UL, (uint16_t *)I2C2_BASE + 0x10UL,
    (uint16_t *)I2C3_BASE + 0x10UL
}
```

Array of pointers to the Data registers

4.2.5.5 I2C_FLTR

```
volatile uint16_t* const I2C_FLTR[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x24UL, (uint16_t *)I2C2_BASE + 0x24UL,
    (uint16_t *)I2C3_BASE + 0x24UL
}
```

Array of pointers to the filter registers

4.2.5.6 i2c_interrupt_callbacks

```
i2c_interrupt_callback_t i2c_interrupt_callbacks[NUM_I2C]  [static]
```

Static array containing interrupt callbacks currently mapped to each i2c channel

4.2.5.7 i2c_interrupt_transfers

```
i2c_transfer_t i2c_interrupt_transfers[NUM_I2C]  [static]
```

Static array which holds copies of requested interrupt based transfers

4.2.5.8 I2C_OAR1

```
volatile uint16_t* const I2C_OAR1[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x08UL, (uint16_t *)I2C2_BASE + 0x08UL,
    (uint16_t *)I2C3_BASE + 0x08UL
}
```

Array of pointers to the Own Address 1 registers

4.2.5.9 I2C_OAR2

```
volatile uint16_t* const I2C_OAR2[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x0CUL, (uint16_t *)I2C2_BASE + 0x0CUL,
    (uint16_t *)I2C3_BASE + 0x0CUL
}
```

Array of pointers to the Own Address 2 registers

4.2.5.10 I2C_SR1

```
volatile uint16_t* const I2C_SR1[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x14UL, (uint16_t *)I2C2_BASE + 0x14UL,
    (uint16_t *)I2C3_BASE + 0x14UL
}
```

Array of pointers to the Status Register 1 registers

4.2.5.11 I2C_SR2

```
volatile uint16_t* const I2C_SR2[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x18UL, (uint16_t *)I2C2_BASE + 0x18UL,
    (uint16_t *)I2C3_BASE + 0x18UL
}
```

Array of pointers to the Status Register 2 registers

4.2.5.12 I2C_TRISE

```
volatile uint16_t* const I2C_TRISE[NUM_I2C]  [static]
```

Initial value:

```
=
{
    (uint16_t *)I2C1_BASE + 0x20UL, (uint16_t *)I2C2_BASE + 0x20UL,
    (uint16_t *)I2C3_BASE + 0x20UL
}
```

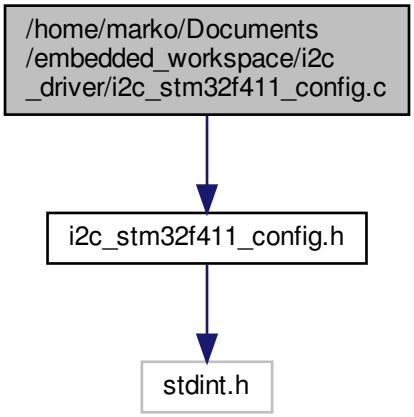
Array of pointers to the rise time registers

4.3 /home/marko/Documents/embedded_workspace/i2c_driver/i2c_stm32f411_config.c File Reference

Contains the configuration information for each I2C channel.

```
#include "i2c_stm32f411_config.h"
```

Include dependency graph for i2c_stm32f411_config.c:



Functions

- const i2c_config_t * i2c_config_get (void)

Variables

- static const i2c_config_t i2c_config_table [NUM_I2C]

4.3.1 Detailed Description

Contains the configuration information for each I2C channel.

4.3.2 Function Documentation

4.3.2.1 i2c_config_get()

```
const i2c_config_t* i2c_config_get (
            void  )
```

Description:

```
Returns a pointer to the base of the configuration table for i2c peripherals
```

PRE-CONDITION: The config table has been filled out and is non-null

Returns

*i2c_config_t

Example:

```
const i2c_config_t *config_table = i2c_config_get();
i2c_init(config_table);
```

See also

i2c_init
- CHANGE HISTORY -

Date	Software Version	Initials	Description

4.3.3 Variable Documentation

4.3.3.1 i2c_config_table

```
const i2c_config_t i2c_config_table[NUM_I2C]  [static]
```

Initial value:

```
=
{

        {},
        {},
        {}
}
```

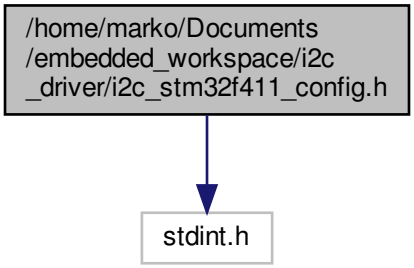
The configuration table that must be filled out by the user and is used by i2c_init to initialise the separate i2c channels

4.4 /home/marko/Documents/embedded_workspace/i2c_driver/i2c_stm32f411_config.h File Reference

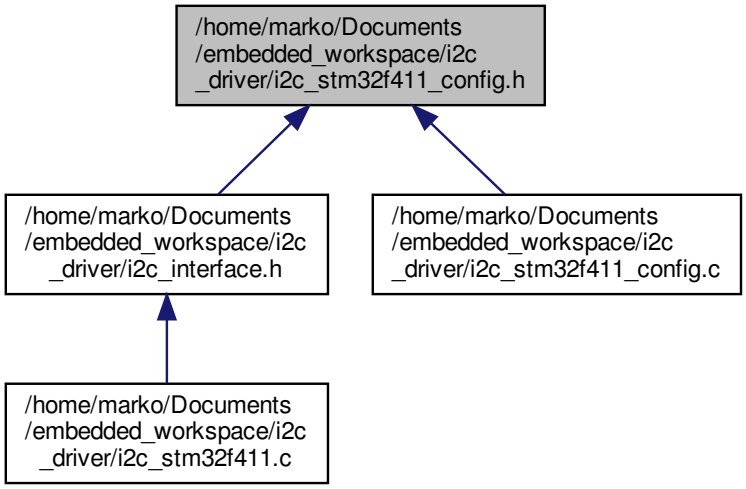
Contains the definitions and structures required to configure the i2c peripherals on an stm32f411.

```
#include <stdint.h>
```

Include dependency graph for i2c_stm32f411_config.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct i2c_config_t

Macros

- #define **DISABLED** 0
- #define **ENABLED** 1

Enumerations

- enum i2c_control_t { **I2C_DISABLED**, **I2C_ENABLED** }
- enum i2c_ack_en_t { **I2C_ACK_DISABLED**, **I2C_ACK_ENABLED** }
- enum i2c_interrupt_dma_t { IT_ERR, IT_EVT, IT_BUF, DMA_REQ }
- enum i2c_channel_t { **I2C_1** = 0x00UL, **I2C_2** = 0x01UL, **I2C_3** = 0x02UL, **NUM_I2C** }
- enum i2c_fast_slow_t { I2C_SM = 0x00UL, I2C_FM = 0x01UL }
- enum i2c_fm_duty_cycle_t { FM_MODE_2 = 0x00UL, FM_MODE_16_9 = 0x01UL }

Functions

- const i2c_config_t * i2c_config_get (void)

4.4.1 Detailed Description

Contains the definitions and structures required to configure the i2c peripherals on an stm32f411.

4.4.2 Enumeration Type Documentation

4.4.2.1 i2c_ack_en_t

`enum i2c_ack_en_t`

Options which decided whether the I2C returns an ACK pulse upon data reception or address match

4.4.2.2 i2c_channel_t

`enum i2c_channel_t`

Contains all of the I2C devices on chip

4.4.2.3 i2c_control_t

`enum i2c_control_t`

Options for enabling or disabling an I2C channel

4.4.2.4 i2c_fast_slow_t

`enum i2c_fast_slow_t`

Decides the maximum frequency with which the i2c may work

Enumerator

I2C_SM	Up to 100kHz
I2C_FM	Up to 400kHz

4.4.2.5 i2c_fm_duty_cycle_t

```
enum i2c_fm_duty_cycle_t
```

Determines the ratio of low to high periods per I2C pulse

Enumerator

FM_MODE_2	T_low/T_high = 2
FM_MODE_16_9	T_low/T_high = 16/9

4.4.2.6 i2c_interrupt_dma_t

```
enum i2c_interrupt_dma_t
```

Lists all the possible interrupts (and the dma request mode) available to the I2C channel

Enumerator

IT_ERR	The I2C raises an interrupt upon an error flag being raised
IT_EVT	The I2C raises an interrupt upon events: Start Bit, Address Matching, STOPF, BTF
IT_BUF	The I2C raises an interrupt when TxE or RxNE = 1 (if IT_EVT is also enabled)
DMA_REQ	Th2 I2C issues a DMA request upon TxE or TxNE = 1

4.4.3 Function Documentation

4.4.3.1 i2c_config_get()

```
const i2c_config_t* i2c_config_get (
            void  )
```

Description:

```
Returns a pointer to the base of the configuration table for i2c peripherals
```

PRE-CONDITION: The config table has been filled out and is non-null

Returns

∗i2c_config_t

Example:

```
const i2c_config_t *config_table = i2c_config_get();
i2c_init(config_table);
```

See also

i2c_init
- CHANGE HISTORY -

Date	Software Version	Initials	Description