# SPI Driver

Version 1.0.0

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Portable SPI driver

Written as per the standards of Beningo's book. Marks the final driver I'll probably be making for the hal. TODO for all communication drivers: implement systick for timeouts

# Chapter 2

# Data Structure Index

## 2.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

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

# Chapter 4

# Data Structure Documentation

## 4.1 spi_config_t Struct Reference

`#include <spi_stm32f411_config.h>`

### Data Fields

- spi_enable_t spi_enable
- spi_master_slave_t master_slave
- spi_slave_mgmt_t slave_management
- spi_bidir_t bidirectional_mode
- spi_baud_rate_t baud_rate

### 4.1.1 Detailed Description

An initialisation config struct which configures the global parameters for a spi device

### 4.1.2 Field Documentation

#### 4.1.2.1 baud_rate

`spi_baud_rate_t baud_rate`

Communication rate of the spi

#### 4.1.2.2 bidirectional_mode

`spi_bidir_t bidirectional_mode`

Configured based upon physical topology of the spi

#### 4.1.2.3 master_slave

```
spi_master_slave_t master_slave
```

Decides whether the spi is in master or slave mode

#### 4.1.2.4 slave_management

```
spi_slave_mgmt_t slave_management
```

Determines method of (self) slave management

#### 4.1.2.5 spi_enable

```
spi_enable_t spi_enable
```

Decides whether the spi will be used at all

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

- /home/marko/Documents/embedded_workspace/spi_driver/spi_stm32f411_config.h

## 4.2 spi_transfer_t Struct Reference

```
#include <spi_interface.h>
```

### Data Fields

- spi_channel_t channel
- gpio_pin_t slave_pin
- spi_ss_polarity_t ss_polarity
- uint16_t * tx_buffer
- uint32_t tx_length
- uint16_t * rx_buffer
- uint32_t rx_length
- spi_data_format_t data_format
- spi_bit_format_t bit_format
- spi_clock_polarity_t clock_polarity
- spi_clock_phase_t clock_phase
- spi_bidir_dir_t bidir_direction

### 4.2.1 Detailed Description

Struct containing implementation agnostic transfer information.

### 4.2.2 Field Documentation

#### 4.2.2.1 bidir_direction

```
spi_bidir_dir_t bidir_direction
```

Direction of the single data line transfer

#### 4.2.2.2 bit_format

```
spi_bit_format_t bit_format
```

MSB or LSB first

#### 4.2.2.3 channel

```
spi_channel_t channel
```

The on-chip spi device to manage the transfer

#### 4.2.2.4 clock_phase

```
spi_clock_phase_t clock_phase
```

Edge sensitivity on sampling and shifts

#### 4.2.2.5 clock_polarity

```
spi_clock_polarity_t clock_polarity
```

Selection of the clock's active and idle states

#### 4.2.2.6 data_format

```
spi_data_format_t data_format
```

Data size of the transfer elements

#### 4.2.2.7 rx_buffer

```
uint16_t* rx_buffer
```

Pointer to the data buffer for reception

#### 4.2.2.8 rx_length

```
uint32_t rx_length
```

Length of the reception buffer

#### 4.2.2.9 slave_pin

```
gpio_pin_t slave_pin
```

The slave's ss pin

#### 4.2.2.10 ss_polarity

```
spi_ss_polarity_t ss_polarity
```

The polarity of slave_pin

#### 4.2.2.11 tx_buffer

```
uint16_t* tx_buffer
```

Pointer to the data buffer for transfers

#### 4.2.2.12 tx_length

```
uint32_t tx_length
```

Length of the transfer buffer

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

- /home/marko/Documents/embedded_workspace/spi_driver/spi_interface.h

# Chapter 5

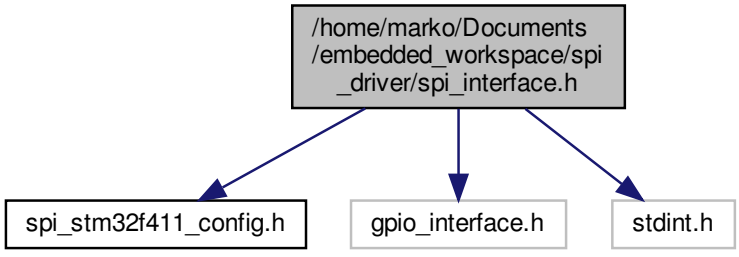
# File Documentation

## 5.1 /home/marko/Documents/embedded_workspace/spi_driver/spi_interface.h File Reference

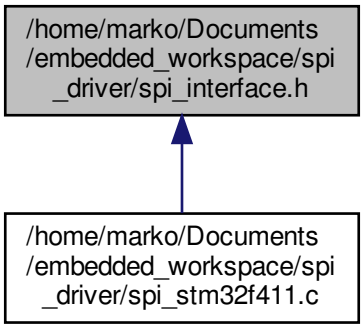
General interface covering user accesses to the spi communication bus.

```
#include "spi_stm32f411_config.h"
#include "gpio_interface.h"
#include <stdint.h>
```

Include dependency graph for spi_interface.h:



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




### Data Structures

- struct spi_transfer_t

### Enumerations

- enum spi_ss_polarity_t { SS_ACTIVE_LOW, SS_ACTIVE_HIGH }
- enum spi_clock_polarity_t { ACTIVE_HIGH, ACTIVE_LOW }
- enum spi_clock_phase_t { FIRST_EDGE, SECOND_EDGE }
- enum spi_bit_format_t { MSB_FIRST, LSB_FIRST }
- enum spi_data_format_t { SPI_DATA_8BIT, SPI_DATA_16BIT }
- enum spi_bidir_dir_t { BIDIR_RECEIVE, BIDIR_TRANSMIT }

### Functions

- void spi_init (spi_config_t *config_table)
- void spi_transfer (spi_transfer_t *transfer)
- void spi_transfer_it (spi_transfer_t *transfer)
- void spi_irq_handler (spi_channel_t channel)
- void spi_register_write (uint32_t spi_register, uint16_t value)
- uint16_t spi_register_read (uint32_t spi_register)

## 5.1.1 Detailed Description

General interface covering user accesses to the spi communication bus.

## 5.1.2 Enumeration Type Documentation

### 5.1.2.1 spi_bidir_dir_t

```
enum spi_bidir_dir_t
```

Contains the direction selections when using single data wire (Bidirectional) spi

**Enumerator**

| | |
|---|---|
| BIDIR_RECEIVE | The data line is being used for data reception |
| BIDIR_TRANSMIT | The data line is being used for data transmission |

#### 5.1.2.2 spi_bit_format_t

`enum spi_bit_format_t`

Contanins the options for the bit order of transfers

**Enumerator**

| | |
|---|---|
| MSB_FIRST | The most significant bit is sent first |
| LSB_FIRST | The least significant bit is sent first |

#### 5.1.2.3 spi_clock_phase_t

`enum spi_clock_phase_t`

Contains the options for the clock's phase

**Enumerator**

| | |
|---|---|
| FIRST_EDGE | Data is clocked out/sampled on the first edge |
| SECOND_EDGE | Data is clocked out/sampled on the second edge |

#### 5.1.2.4 spi_clock_polarity_t

`enum spi_clock_polarity_t`

Contains the options for the clock's polarity

**Enumerator**

| | |
|---|---|
| ACTIVE_HIGH | The clock's idle state is low and ticks upwards |
| ACTIVE_LOW | The clock's idle state is high and ticks downwards |

#### 5.1.2.5 spi_data_format_t

```
enum spi_data_format_t
```

Contains the options for payload size

**Enumerator**

| | |
|---|---|
| SPI_DATA_8BIT | The data is in 8 bit frames |
| SPI_DATA_16BIT | The data is in 16 bit frames |

#### 5.1.2.6 spi_ss_polarity_t

```
enum spi_ss_polarity_t
```

Contains the options for slave select

**Enumerator**

| | |
|---|---|
| SS_ACTIVE_LOW | A slave is selected by pulling its select pin low |
| SS_ACTIVE_HIGH | A slave is selected by pulling its select pin high |

### 5.1.3 Function Documentation

#### 5.1.3.1 spi_init()

```
void spi_init (
            spi_config_t * config_table )
```

**Description:**

```
Carries out the initialisation of the spi 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 the spi combination have been configured correctly with gpio_init PRE-CONDITION: The appropriate peripheral clocks have been activated

POST-CONDITION: The selected spi channels have been activated and ready to be used

**Returns**

void

**Example:**

```
const spi_config_t *config_table = spi_config_get();
spi_init(config_table);
```

**See also**

spi_config_get
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.1.3.2 spi_irq_handler()

```
void spi_irq_handler (
             spi_channel_t channel )
```

**Description:**

Calls the appropriate callback function (registered during spi_transfer_it) and feeds it a safe copy of the desired transfer (made during spi_transfer_it).

PRE-CONDITION: spi_transfer_it has been called and set up on the desired channel POST-CONDITION: The callback has been called and has handled a single reception/transfer/end of transfer

**Returns**

void

**Example:** Called from withn the hardware defined irqs defined in the vector table

```
SPI3_IRQHandler()
{
    spi_irq_handler(SPI_3);
}
```

**See also**

spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.1.3.3 spi_register_read()

```
uint16_t spi_register_read (
            uint32_t spi_register )
```

**Description:**

Reads the current value from the register in spi address space

PRE-CONDITION: the spi_register is within spi address space POST-CONDITION: Returns the value within the register

**Returns**

uint16_t

**Example:**

```
uin16_t current_value = spi_register_write(SPI1_BASE + 0x20UL);
```

**See also**

spi_register_write
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.1.3.4 spi_register_write()

```
void spi_register_write (
            uint32_t spi_register,
            uint16_t value )
```

**Description:**

Write the desired value into the register in spi address space

PRE-CONDITION: the spi_register is within spi address space POST-CONDITION: the spi_register contains the desired value

**Returns**

void

**Example:**

```
uint16_t new_value = 0xFF2A;
spi_register_write(SPI1_BASE + 0x20UL, new_value);
```

**See also**

spi_register_read
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.1.3.5 spi_transfer()

```
void spi_transfer (
            spi_transfer_t * transfer )
```

**Description:**

```
Carries out a blocking spi transfer according to the specifications of the
 transfer parameter.
```

PRE-CONDITION: spi_init() has been successfully carried out for the required spi channel PRE-CONDITION↩ : gpio_init() has been called for the slave select pin to configure it as an output/input, depending on desired direction PRE-CONDITION: The proper data buffers and lengths are non-null/non-zero PRE-CONDITION: The transfer pointer is non-null

POST-CONDITION: The desired transfer has been successfully carried out

**Returns**

void

**Example:**

```
spi_transfer_t flash_transfer;
flash_transfer.channel = SPI_3;
flash_transfer.slave_pin = GPIO_C_3;
flash_transfer.ss_polarity = ACTIVE_LOW;
flash_transfer.tx_buffer = &data_out;
flash_transfer.tx_length = sizeof(data_out);
flash_transfer.rx_buffer = &dummy_data;
flash_transfer.rx_length = sizeof(data_out);
flash_transfer.data_format = SPI_DATA_8BIT;
flash_transfer.bit_format = MSB_FIRST;
flash_transfer.clock_polarity = ACTIVE_HIGH;
flash_transfer.clock_phase = SECOND_EDGE;
spi_transfer(&flash_transfer);
```

**See also**

spi_init
spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.1.3.6 spi_transfer_it()

```
void spi_transfer_it (
            spi_transfer_t * transfer )
```

**Description:**

```
Sets up an interrupt based spi transfer according to the specifications of
 the transfer parameter. Makes a safe copy of the transfer structure and
 calls a function to map the correct callback
```

PRE-CONDITION: spi_init() has been successfully carried out for the required spi channel PRE-CONDITION↩ : gpio_init() has been called for the slave select pin to configure it as an output/input, depending on desired direction PRE-CONDITION: The proper data buffers and lengths are non-null/non-zero PRE-CONDITION: The transfer pointer is non-NULL

POST-CONDITION: The irq handler will now handle the rest of the transfer POST-CONDITION: A safe copy of the transfer structure has been placed in the static file-wide buffer

**Returns**

void

**Example:**

```
spi_transfer_t flash_transfer;
flash_transfer.channel = SPI_3;
flash_transfer.slave_pin = GPIO_C_3;
flash_transfer.ss_polarity = ACTIVE_LOW;
flash_transfer.tx_buffer = &data_out;
flash_transfer.tx_length = sizeof(data_out);
flash_transfer.rx_buffer = &dummy_data;
flash_transfer.rx_length = sizeof(data_out);
flash_transfer.data_format = SPI_DATA_8BIT;
flash_transfer.bit_format = MSB_FIRST;
flash_transfer.clock_polarity = ACTIVE_HIGH;
flash_transfer.clock_phase = SECOND_EDGE;
spi_transfer_it(&flash_transfer);
```

**See also**

spi_init
spi_transfer
spi_irq_handler
**- CHANGE HISTORY -**

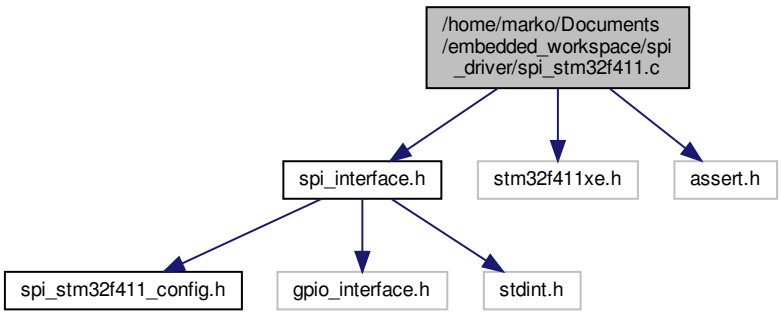
| Date | Software Version | Initials | Description |
|---|---|---|---|

## 5.2 /home/marko/Documents/embedded_workspace/spi_driver/spi_stm32f411.c File Reference

Chip specific implementation for spi communication.

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

Include dependency graph for spi_stm32f411.c:



### Macros

- #define NULL (void*) 0

### Typedefs

- typedef void(* spi_interrupt_callback_t) (spi_transfer_t *)

### Functions

- static void spi_select_slave (spi_transfer_t *transfer)
- static void spi_release_slave (spi_transfer_t *transfer)
- static void spi_configure_clock (spi_transfer_t *transfer)
- static void spi_configure_data_frame (spi_transfer_t *transfer)
- static void spi_transfer_bidir (spi_transfer_t *transfer)
- static void spi_transfer_bidir_transmit (spi_transfer_t *transfer)
- static void spi_transfer_bidir_receive (spi_transfer_t *transfer)
- static void spi_transfer_full_duplex_rxonly (spi_transfer_t *transfer)
- static void spi_transfer_full_duplex (spi_transfer_t *transfer)
- static void spi_transfer_full_duplex_master (spi_transfer_t *transfer)
- static void spi_transfer_full_duplex_slave (spi_transfer_t *transfer)
- static void spi_transfer_it_bidir (spi_transfer_t *transfer)
- static void spi_transfer_it_full_duplex_rxonly (spi_transfer_t *transfer)
- static void spi_transfer_it_full_duplex (spi_transfer_t *transfer)
- void spi_init (spi_config_t *config_table)
- void spi_transfer (spi_transfer_t *transfer)
- void spi_transfer_it (spi_transfer_t *transfer)
- void spi_irq_handler (spi_channel_t channel)
- void spi_register_write (uint32_t spi_register, uint16_t value)
- uint16_t spi_register_read (uint32_t spi_register)
- static void spi_transfer_it_bidir_transmit_callback (spi_transfer_t *transfer)
- static void spi_transfer_it_bidir_receive_callback (spi_transfer_t *transfer)
- static void spi_transfer_it_full_duplex_rxonly_callback (spi_transfer_t *transfer)
- static void spi_transfer_it_full_duplex_callback (spi_transfer_t *transfer)

### Variables

- static volatile uint16_t *const SPI_CR1 [NUM_SPI]
- static volatile uint16_t *const SPI_CR2 [NUM_SPI]
- static volatile uint16_t *const SPI_SR [NUM_SPI]
- static volatile uint16_t *const SPI_DR [NUM_SPI]
- static volatile uint16_t *const SPI_CRCPR [NUM_SPI]
- static volatile uint16_t *const SPI_RXCRCR [NUM_SPI]
- static volatile uint16_t *const SPI_TXCRCR [NUM_SPI]
- static spi_transfer_t spi_interrupt_transfers [NUM_SPI]
- static spi_interrupt_callback_t spi_interrupt_callbacks [NUM_SPI]

#### 5.2.1 Detailed Description

Chip specific implementation for spi communication.

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 NULL

```
#define NULL (void*) 0
```

Redefinition of NULL macro in case stdlib isn't used by the rest of the project

### 5.2.3 Typedef Documentation

#### 5.2.3.1 spi_interrupt_callback_t

```
typedef void(* spi_interrupt_callback_t) (spi_transfer_t *)
```

Callback typedef for interrupt callbacks

### 5.2.4 Function Documentation

#### 5.2.4.1 spi_configure_clock()

```
static void spi_configure_clock (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Static function used to configure the clock phase and polarity to be used during the spi communication

PRE-CONDITION: the clock_polarity member of the transfer structure is valid PRE-CONDITION: the clock_phase member of the transfer structure is valid

POST-CONDITION: The CR1 registers now contain the desired clock configuration

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer and spi_transfer_it and callbacks

**See also**

spi_configure_data_frame
spi_transfer
spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.2 spi_configure_data_frame()

```
static void spi_configure_data_frame (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Static function used to configure the the data size and bit format to be used during the current transfer

PRE-CONDITION: the data_format member of the transfer structure is valid PRE-CONDITION: the bit_format member of the transfer structure is valid

POST-CONDITION: The CR1 registers now contain the desired data format configuration

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer and spi_transfer_it and callbacks

**See also**

spi_configure_clock
spi_transfer
spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.3 spi_init()

```
void spi_init (
            spi_config_t * config_table )
```

**Description:**

```
Carries out the initialisation of the spi 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 the spi combination have been configured correctly with gpio_init PRE-CONDITION: The appropriate peripheral clocks have been activated

POST-CONDITION: The selected spi channels have been activated and ready to be used

**Returns**

void

**Example:**

```
const spi_config_t *config_table = spi_config_get();
spi_init(config_table);
```

**See also**

spi_config_get
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.4 spi_irq_handler()

```
void spi_irq_handler (
            spi_channel_t channel )
```

**Description:**

Calls the appropriate callback function (registered during spi_transfer_it) and feeds it a safe copy of the desired transfer (made during spi_transfer_it).

PRE-CONDITION: spi_transfer_it has been called and set up on the desired channel POST-CONDITION: The callback has been called and has handled a single reception/transfer/end of transfer

**Returns**

void

**Example:** Called from withn the hardware defined irqs defined in the vector table

```
SPI3_IRQHandler()
{
    spi_irq_handler(SPI_3);
}
```

**See also**

spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.5 spi_register_read()

```
uint16_t spi_register_read (
             uint32_t spi_register )
```

**Description:**

Reads the current value from the register in spi address space

PRE-CONDITION: the spi_register is within spi address space POST-CONDITION: Returns the value within the register

**Returns**

uint16_t

**Example:**

```
uin16_t current_value = spi_register_write(SPI1_BASE + 0x20UL);
```

**See also**

spi_register_write
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.6 spi_register_write()

```
void spi_register_write (
            uint32_t spi_register,
            uint16_t value )
```

**Description:**

Write the desired value into the register in spi address space

PRE-CONDITION: the spi_register is within spi address space POST-CONDITION: the spi_register contains the desired value

**Returns**

void

**Example:**

```
uint16_t new_value = 0xFF2A;
spi_register_write(SPI1_BASE + 0x20UL, new_value);
```

**See also**

spi_register_read
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.7 spi_release_slave()

```
static void spi_release_slave (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Static function used to release a slave (whether it is active high or low) from within transfer functions

PRE-CONDITION: The GPIO pin for controlling the slave has been correctly configured

POST-CONDITION: The GPIO output is at the correct level to release the slave

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer and spi_transfer_it and callbacks in master mode

**See also**

spi_select_slave
spi_transfer
spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.8 spi_select_slave()

```
static void spi_select_slave (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Static function used to select a slave (whether it is active high or low) from within transfer functions

PRE-CONDITION: The GPIO pin for controlling the slave has been correctly configured

POST-CONDITION: The GPIO output is at the correct level to select the slave

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer and spi_transfer_it in master mode before transmission begins

**See also**

spi_release_slave
spi_transfer
spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.9 spi_transfer()

```
void spi_transfer (
            spi_transfer_t * transfer )
```

**Description:**

```
Carries out a blocking spi transfer according to the specifications of the
 transfer parameter.
```

PRE-CONDITION: spi_init() has been successfully carried out for the required spi channel PRE-CONDITION↩ : gpio_init() has been called for the slave select pin to configure it as an output/input, depending on desired direction PRE-CONDITION: The proper data buffers and lengths are non-null/non-zero PRE-CONDITION: The transfer pointer is non-null

POST-CONDITION: The desired transfer has been successfully carried out

**Returns**

void

**Example:**

```
spi_transfer_t flash_transfer;
flash_transfer.channel = SPI_3;
flash_transfer.slave_pin = GPIO_C_3;
flash_transfer.ss_polarity = ACTIVE_LOW;
flash_transfer.tx_buffer = &data_out;
flash_transfer.tx_length = sizeof(data_out);
flash_transfer.rx_buffer = &dummy_data;
flash_transfer.rx_length = sizeof(data_out);
flash_transfer.data_format = SPI_DATA_8BIT;
flash_transfer.bit_format = MSB_FIRST;
flash_transfer.clock_polarity = ACTIVE_HIGH;
flash_transfer.clock_phase = SECOND_EDGE;
spi_transfer(&flash_transfer);
```

**See also**

spi_init
spi_transfer_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.10 spi_transfer_bidir()

```
static void spi_transfer_bidir (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called auomatically by spi_transfer when appropriate which subsequently simply calls the appropriate transfer subroutine based on transfer direction

PRE-CONDITION: (Soft assertion) The pointer to the data buffer to be transferred is non-NULL. Failure of this check results in an uneventful return

POST-CONDITION: The data has been transferred as specified in the transfer parameter by the subroutine.

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when BIDIMODE == 1

**See also**

spi_transfer_bidir_transmit
spi_transfer_bidir_receive
spi_transfer_full_duplex_rxonly
spi_transfer_full_duplex
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.11 spi_transfer_bidir_receive()

```
static void spi_transfer_bidir_receive (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called automatically by spi_transfer_bidir when the single data line is being used to receive information

PRE-CONDITION: (Soft assertion) The length of the data buffer is non-NULL. Failure of this check simply skips the function

POST-CONDITION: Places received spi data in the rx_buffer

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when BIDIMODE == 1 && BIDIOE == 0

**See also**

spi_transfer_bidir
spi_transfer_bidir_transmit
spi_transfer_bidir_it
spi_transfer_bidir_transmit_it
spi_transfer_bidir_receive_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.12 spi_transfer_bidir_transmit()

```
static void spi_transfer_bidir_transmit (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called automatically by spi_transfer_bidir when the single data line is being used to send information

PRE-CONDITION: (Soft assertion) The length of the data buffer is non-NULL. Failure of this check simply ends the function

POST-CONDITION: Transmits the data in tx_buffer to the selected slave

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when BIDIMODE == 1 && BIDIOE != 0

**See also**

spi_transfer_bidir
spi_transfer_bidir_receive
spi_transfer_bidir_it
spi_transfer_bidir_transmit_it
spi_transfer_bidir_receive_it
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.13 spi_transfer_full_duplex()

```
static void spi_transfer_full_duplex (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called automatically by spi_transfer when the spi is configured with two data lines. It boots a subroutine dependent on whether or not the spi is configured in master or slave mode

PRE-CONDITION: (Soft assertion) The address and length of the data buffer are non-NULL. Failure of this check simply results in a meaningless return

POST-CONDITION: All data in the tx_buffer has been sent to the selected slave POST-CONDITION: All received data has been placed in the rx_buffer

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when no other special transfer modes are valid

**See also**

spi_transfer
spi_transfer_bidir
spi_transfer_full_duplex_rxonly

**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.14 spi_transfer_full_duplex_master()

```
static void spi_transfer_full_duplex_master (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called automatically by spi_transfer_full_duplex when the spi is configured as a master with two data lines.

PRE-CONDITION: The tx_buffer is non-NULL and of non-zero length PRE-CONDITION: The rx_buffer is non-NULL and of non-zero length

POST-CONDITION: All data in the tx_buffer has been sent to the selected slave POST-CONDITION: All received data has been placed in the rx_buffer

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when no other special transfer modes are valid

**See also**

spi_transfer
spi_transfer_full_duplex
spi_transfer_full_duplex_slave

**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.15 spi_transfer_full_duplex_rxonly()

```
static void spi_transfer_full_duplex_rxonly (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called automatically by spi_transfer when only the MISO line is being used to receive data

PRE-CONDITION: (Soft assertion) The address and length of the data buffer are non-NULL. Failure of this check simply results in a meaningless return

POST-CONDITION: The received data is placed in rx_buffer

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when RX_ONLY == 1

**See also**

spi_transfer
spi_transfer_bidir
spi_transfer_full_duplex

**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.16 spi_transfer_full_duplex_slave()

```
static void spi_transfer_full_duplex_slave (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static function called automatically by spi_transfer_full_duplex when the spi is configured as a slave with two data lines.

PRE-CONDITION: The tx_buffer is non-NULL and of non-zero length PRE-CONDITION: The rx_buffer is non-NULL and of non-zero length

POST-CONDITION: All data in the tx_buffer has been sent to the master POST-CONDITION: All received data has been placed in the rx_buffer

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called automatically by spi_transfer when no other special transfer modes are valid

**See also**

spi_transfer
spi_transfer_full_duplex
spi_transfer_full_duplex_master

**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.17 spi_transfer_it()

```
void spi_transfer_it (
            spi_transfer_t * transfer )
```

**Description:**

```
Sets up an interrupt based spi transfer according to the specifications of
 the transfer parameter. Makes a safe copy of the transfer structure and
 calls a function to map the correct callback
```

PRE-CONDITION: spi_init() has been successfully carried out for the required spi channel PRE-CONDITION↩ : gpio_init() has been called for the slave select pin to configure it as an output/input, depending on desired direction PRE-CONDITION: The proper data buffers and lengths are non-null/non-zero PRE-CONDITION: The transfer pointer is non-NULL

POST-CONDITION: The irq handler will now handle the rest of the transfer POST-CONDITION: A safe copy of the transfer structure has been placed in the static file-wide buffer

**Returns**

void

**Example:**

```
spi_transfer_t flash_transfer;
flash_transfer.channel = SPI_3;
flash_transfer.slave_pin = GPIO_C_3;
flash_transfer.ss_polarity = ACTIVE_LOW;
flash_transfer.tx_buffer = &data_out;
flash_transfer.tx_length = sizeof(data_out);
flash_transfer.rx_buffer = &dummy_data;
flash_transfer.rx_length = sizeof(data_out);
flash_transfer.data_format = SPI_DATA_8BIT;
flash_transfer.bit_format = MSB_FIRST;
flash_transfer.clock_polarity = ACTIVE_HIGH;
flash_transfer.clock_phase = SECOND_EDGE;
spi_transfer_it(&flash_transfer);
```

**See also**

spi_init
spi_transfer
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.18 spi_transfer_it_bidir()

```
static void spi_transfer_it_bidir (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Maps the appropriate bidir callback and makes a safe copy of the transfer structure.

PRE-CONDITION: (Soft Assert) The tx_buffer is non-NULL and of non-zero length OR PRE-CONDITION: (Soft Assert) The rx_buffer is non-NULL and of non-zero length

POST-CONDITION: The correct callback function has been mapped to the file-scope callback array POST-CO↩ NDITION: The correct buffer interrupt (RXNEIE or TXEIE) has been enabled POST-CONDITION: The SPI Enable (SPE) has been switched on

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer_it when BIDIMODE == 1

**See also**

spi_transfer_it
spi_transfer_it_bidir_transmit_callback
spi_transfer_it_bidir_receive_callback
spi_transfer_it_full_duplex
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.19 spi_transfer_it_bidir_receive_callback()

```
static void spi_transfer_it_bidir_receive_callback (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static callback mapped to the irq handler by spi_transfer_bidir_it and called by the irq handler. Handles the reception of a single data unit or ends the communication and releases the slave.

PRE-CONDITION: The rx_buffer is of non-zero length

POST-CONDITION: A single data unit has been received from the target OR POST-CONDITION: The communication has been ended and the slave released

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Registered when appropriate by spi_transfer_it_bidir and called by the spi_irq_handler

**See also**

spi_transfer_it
spi_transfer_it_bidir
spi_transfer_it_full_duplex
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.20 spi_transfer_it_bidir_transmit_callback()

```
static void spi_transfer_it_bidir_transmit_callback (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A static callback mapped to the irq handler by spi_transfer_bidir_it and called by the irq handler. Handles the transmission of a single data unit or ends the communication and releases the slave.

PRE-CONDITION: The tx_buffer is of non-zero length

POST-CONDITION: A single data unit has been sent to the target OR POST-CONDITION: The communication has been ended and the slave released

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Registered when appropriate by spi_transfer_it_bidir and called by the spi_irq_handler

**See also**

spi_transfer_it
spi_transfer_it_bidir
spi_irq_handler
spi_transfer_it_full_duplex
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.21 spi_transfer_it_full_duplex()

```
static void spi_transfer_it_full_duplex (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Maps the correct callback to the callback array and enables the correct interrupt flags

PRE-CONDITION: None

POST-CONDITION: The reception and transmission (RXNEIE and TXEIE) interrupts have been enabled

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer_it when full duplex configuration is selected

**See also**

spi_transfer_it
spi_transfer_it_bidir
spi_transfer_it_full_duplex_rxonly
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.22 spi_transfer_it_full_duplex_callback()

```
static void spi_transfer_it_full_duplex_callback (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A callback function called by the irq handler which manages the reception and transmission of a single data unit when the spi has two data lines.

PRE-CONDITION: (Soft Assert) The rx_buffer is of non-zero length PRE-CONDITION: (Soft Assert) The tx_buffer is of non-zero length

POST-CONDITION: A single data unit has been received and another sent OR POST-CONDITION: The communication has been shut down and the slave released

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by the irq_handler if it's mapped

**See also**

spi_transfer_it
spi_transfer_it_bidir
spi_transfer_it_full_duplex_rxonly
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.23 spi_transfer_it_full_duplex_rxonly()

```
static void spi_transfer_it_full_duplex_rxonly (
            spi_transfer_t * transfer )  [static]
```

**Description:**

Registers the rxonly callback and makes a safe copy of the transfer structure.

PRE-CONDITION: The rx buffer is non-NULL and of non-zero length

POST-CONDITION: The rxonly callback has been mapped to the right spi device POST-CONDITION: The RXNEIE interrupt has been enabled

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by spi_transfer_it when RXONLY == 1

**See also**

spi_transfer_it
spi_transfer_it_bidir
spi_transfer_it_full_duplex
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

#### 5.2.4.24 spi_transfer_it_full_duplex_rxonly_callback()

```
static void spi_transfer_it_full_duplex_rxonly_callback (
            spi_transfer_t * transfer )  [static]
```

**Description:**

A callback function called by the irq handler which manages the reception of a single data unit when the spi has two data lines.

PRE-CONDITION: (Soft Assert) The rx_buffer is of non-zero length

POST-CONDITION: A single data unit has been received OR POST-CONDITION: The communication has been shut down and the slave released

**Parameters**

| | |
|---|---|
| *transfer* | a pointer to the transfer structure containing all relevant information for the transmission |

**Returns**

void

**Example:** Called by the irq_handler if it's mapped

**See also**

spi_transfer_it
spi_transfer_it_bidir
spi_transfer_it_full_duplex
spi_irq_handler
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

### 5.2.5 Variable Documentation

#### 5.2.5.1 SPI_CR1

```
volatile uint16_t* const SPI_CR1[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE,  (uint16_t *)SPI2_BASE,
    (uint16_t *)SPI3_BASE,  (uint16_t *)SPI4_BASE,
    (uint16_t *)SPI5_BASE
}
```

Array of pointers to Control Register 1 registers

#### 5.2.5.2 SPI_CR2

```
volatile uint16_t* const SPI_CR2[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE + 0x04UL, (uint16_t *)SPI2_BASE + 0x04UL,
    (uint16_t *)SPI3_BASE + 0x04UL, (uint16_t *)SPI4_BASE + 0x04UL,
    (uint16_t *)SPI5_BASE + 0x04UL
}
```

Array of pointers to Control Register 2 registers

#### 5.2.5.3 SPI_CRCPR

```
volatile uint16_t* const SPI_CRCPR[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE + 0x10UL, (uint16_t *)SPI2_BASE + 0x10UL,
    (uint16_t *)SPI3_BASE + 0x10UL, (uint16_t *)SPI4_BASE + 0x10UL,
    (uint16_t *)SPI5_BASE + 0x10UL
}
```

Array of pointers to CRC Polynomial registers

#### 5.2.5.4 SPI_DR

```
volatile uint16_t* const SPI_DR[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE + 0x0CUL, (uint16_t *)SPI2_BASE + 0x0CUL,
    (uint16_t *)SPI3_BASE + 0x0CUL, (uint16_t *)SPI4_BASE + 0x0CUL,
    (uint16_t *)SPI5_BASE + 0x0CUL
}
```

Array of pointers to Data registers

#### 5.2.5.5 spi_interrupt_callbacks

```
spi_interrupt_callback_t spi_interrupt_callbacks[NUM_SPI]  [static]
```

Static array of callback functions mapped to each spi device

#### 5.2.5.6 spi_interrupt_transfers

```
spi_transfer_t spi_interrupt_transfers[NUM_SPI]  [static]
```

Static array which holds safe copies of transfers for interrupt routines, mapped to spi devices

#### 5.2.5.7 SPI_RXCRCR

```
volatile uint16_t* const SPI_RXCRCR[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE + 0x14UL, (uint16_t *)SPI2_BASE + 0x14UL,
    (uint16_t *)SPI3_BASE + 0x14UL, (uint16_t *)SPI4_BASE + 0x14UL,
    (uint16_t *)SPI5_BASE + 0x14UL
}
```

Array of pointers to reception CRC registers

#### 5.2.5.8 SPI_SR

```
volatile uint16_t* const SPI_SR[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE + 0x08UL, (uint16_t *)SPI2_BASE + 0x08UL,
    (uint16_t *)SPI3_BASE + 0x08UL, (uint16_t *)SPI4_BASE + 0x08UL,
    (uint16_t *)SPI5_BASE + 0x08UL
}
```

Array of pointers to Status registers

#### 5.2.5.9 SPI_TXCRCR

```
volatile uint16_t* const SPI_TXCRCR[NUM_SPI]  [static]
```

**Initial value:**

```
=
{
    (uint16_t *)SPI1_BASE + 0x18UL, (uint16_t *)SPI2_BASE + 0x18UL,
    (uint16_t *)SPI3_BASE + 0x18UL, (uint16_t *)SPI4_BASE + 0x18UL,
    (uint16_t *)SPI5_BASE + 0x18UL
}
```

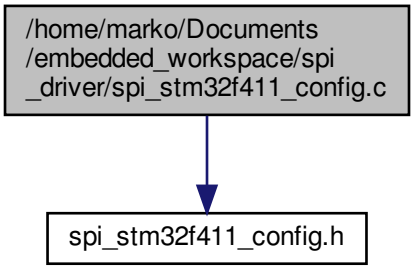
Array of pointers to transmission CRC registers

## 5.3 /home/marko/Documents/embedded_workspace/spi_driver/spi_stm32f411_config.c File Reference

Contains the configuration information for each spi channel.

```
#include "spi_stm32f411_config.h"
```

Include dependency graph for spi_stm32f411_config.c:



## Functions

- const spi_config_t * spi_config_get (void)

## Variables

- static const spi_config_t config_table [NUM_SPI]

### 5.3.1 Detailed Description

Contains the configuration information for each spi channel.

### 5.3.2 Function Documentation

#### 5.3.2.1 spi_config_get()

```
const spi_config_t* spi_config_get (
            void  )
```

**Description:**

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

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

**Returns**

*spi_config_t

**Example:**

```
const spi_config_t *config_table = spi_config_get();
spi_init(config_table);
```

**See also**

spi_init
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|

### 5.3.3 Variable Documentation

#### 5.3.3.1 config_table

```
const spi_config_t config_table[NUM_SPI]  [static]
```

**Initial value:**

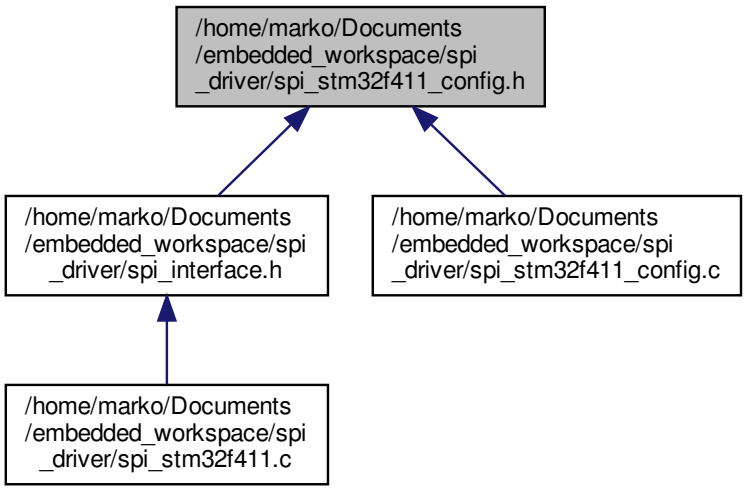
```
=
{

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

Config table containing peripheral wide options for each spi device on chip

## 5.4 /home/marko/Documents/embedded_workspace/spi_driver/spi_stm32f411_config.h File Reference

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

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



### Data Structures

- struct spi_config_t

### Enumerations

- enum spi_channel_t { **SPI_1** = 0x00UL, **SPI_2** = 0x01UL, **SPI_3** = 0x02UL, **SPI_4** = 0x03UL, **SPI_5** = 0x04UL, **NUM_SPI** = 0x05UL }
- enum spi_master_slave_t { SPI_SLAVE, SPI_MASTER }
- enum spi_baud_rate_t { **PCLK_DIV_2**, **PCLK_DIV_4**, **PCLK_DIV_8**, **PCLK_DIV_16**, **PCLK_DIV_32**, **PCLK_DIV_64**, **PCLK_DIV_128**, **PCLK_DIV_256** }
- enum spi_enable_t { **SPI_DISABLE**, **SPI_ENABLE** }
- enum spi_slave_mgmt_t { **HARDWARE_SMM**, **SOFTWARE_SMM** }
- enum spi_bidir_t { UNIDIR_FULL_DUPLEX, UNIDIR_RXONLY, BIDIR_MODE }
- enum spi_crc_en_t { **CRC_DISABLE**, **CRC_ENABLE** }
- enum spi_rx_dma_t { **RX_DMA_REQ_DISABLE**, **RX_DMA_REQ_ENABLE** }
- enum spi_tx_dma_t { **TX_DMA_REQ_DISABLE**, **TX_DMA_REQ_ENABLE** }
- enum spi_ssoe_t { **SS_OUTPUT_DISABLE**, **SS_OUTPUT_ENABLE** }
- enum spi_frame_format_t { **SPI_MOTOROLA**, **SPI_TI** }
- enum spi_interrupt_t { ERROR_INTERRUPT, RXNE_INTERRUPT, TXE_INTERRUPT }

### Functions

- const spi_config_t * spi_config_get (void)

## 5.4.1 Detailed Description

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

## 5.4.2 Enumeration Type Documentation

#### 5.4.2.1 spi_baud_rate_t

`enum spi_baud_rate_t`

Contains all of the prescaler options for the master clock generation

#### 5.4.2.2 spi_bidir_t

`enum spi_bidir_t`

Contains options determining the topology of the bus system

**Enumerator**

| | |
|---|---|
| UNIDIR_FULL_DUPLEX | Two data wire spi, with simultaneous data reception and transmission |
| UNIDIR_RXONLY | Two data wire spi, but with reception only |
| BIDIR_MODE | Single data wire spi, must select transfer direction upon call |

#### 5.4.2.3 spi_channel_t

`enum spi_channel_t`

Contains all of the spi devices found on chip

#### 5.4.2.4 spi_crc_en_t

`enum spi_crc_en_t`

Contains options for enabling the hardware CRC calculation of received data

#### 5.4.2.5 spi_enable_t

`enum spi_enable_t`

Spi devices which are disabled are ignored during init

#### 5.4.2.6 spi_frame_format_t

`enum spi_frame_format_t`

Options for motorola's vs TI's spi format. Motorola is the defualt and should suffice

#### 5.4.2.7 spi_interrupt_t

`enum spi_interrupt_t`

Contains all the possible interrupts a spi device can handle

**Enumerator**

| | |
|---|---|
| ERROR_INTERRUPT | Interrupt generated upon a communication error |
| RXNE_INTERRUPT | Interrupt generated upon having non-read received data |
| TXE_INTERRUPT | Interrupt generated when no data is in the transfer buffer |

#### 5.4.2.8 spi_master_slave_t

```
enum spi_master_slave_t
```

Contains the options for the spi device's functioning mode

**Enumerator**

| | |
|---|---|
| SPI_SLAVE | The spi is functioning as a slave and will await selection and a clock |
| SPI_MASTER | The spi is functioning as a master and will select slaves and generate a clock |

#### 5.4.2.9 spi_rx_dma_t

```
enum spi_rx_dma_t
```

Contains options to enable DMA requests upon data reception

#### 5.4.2.10 spi_slave_mgmt_t

```
enum spi_slave_mgmt_t
```

Contains options to enable the self selection of the spi (as a slave) through software

#### 5.4.2.11 spi_ssoe_t

```
enum spi_ssoe_t
```

Contains options for slave select output control. Allows work in multi-master mode when disabled

#### 5.4.2.12 spi_tx_dma_t

```
enum spi_tx_dma_t
```

Contains options to enable DMA requests upon data transmission

### 5.4.3 Function Documentation

#### 5.4.3.1 spi_config_get()

```
const spi_config_t* spi_config_get (
            void  )
```

**Description:**

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

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

**Returns**

*spi_config_t

**Example:**

```
const spi_config_t *config_table = spi_config_get();
spi_init(config_table);
```

**See also**

spi_init
**- CHANGE HISTORY -**

| Date | Software Version | Initials | Description |
|---|---|---|---|