

# 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:

<a href="#">spi_config_t</a>	.....	??
<a href="#">spi_transfer_t</a>	.....	??





## Chapter 3

# File Index

### 3.1 File List

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

/home/marko/Documents/embedded_workspace/spi_driver/ <a href="#">spi_interface.h</a>	
General interface covering user accesses to the spi communication bus . . . . .	??
/home/marko/Documents/embedded_workspace/spi_driver/ <a href="#">spi_stm32f411.c</a>	
Chip specific implementation for spi communication . . . . .	??
/home/marko/Documents/embedded_workspace/spi_driver/ <a href="#">spi_stm32f411_config.c</a>	
Contains the configuration information for each spi channel . . . . .	??
/home/marko/Documents/embedded_workspace/spi_driver/ <a href="#">spi_stm32f411_config.h</a>	
Contains the definitions and structures required to configure the spi peripherals on an stm32f411	??



## 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`

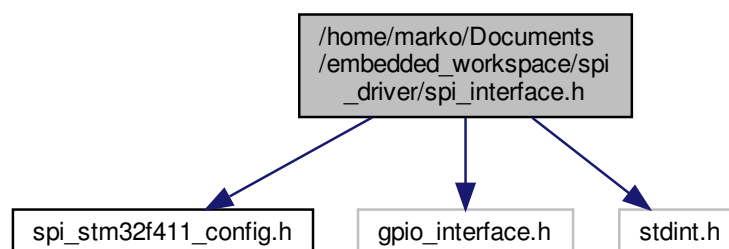
## Chapter 5

# File Documentation

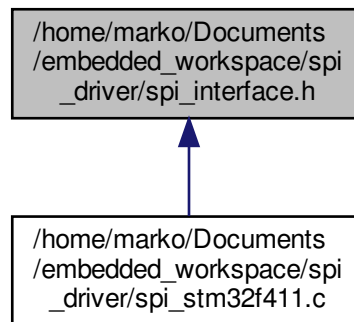
### 5.1 /home/marko/Documents/embedded\_workspace/spi\_driver/spi\_interface.h File Reference

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
```

Contains 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 within 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:

```
uint16_t current_value = spi_register_read(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

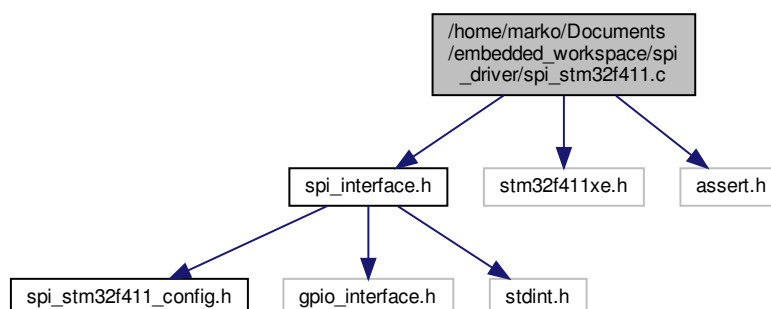
Chip specific implementation for spi communication.

```
#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\\_RXCR](#) [NUM\_SPI]
- static volatile uint16\_t \*const [SPI\\_TXCR](#) [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 Typedef Documentation

### 5.2.2.1 spi\_interrupt\_callback\_t

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

Callback typedef for interrupt callbacks

## 5.2.3 Function Documentation

### 5.2.3.1 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

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.2 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

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.3 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
------	------------------	----------	-------------

#### 5.2.3.4 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 within the hardware defined irq's defined in the vector table

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

#### See also

[spi\\_transfer\\_it](#)

- CHANGE HISTORY -

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

#### 5.2.3.5 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:

```
uint16_t current_value = spi_register_read(SPI1_BASE + 0x20UL);
```

##### See also

[spi\\_register\\_write](#)

- CHANGE HISTORY -

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

#### 5.2.3.6 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
------	------------------	----------	-------------

#### 5.2.3.7 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

<i>transfer</i>	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
------	------------------	----------	-------------

**5.2.3.8 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**

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.9 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
------	------------------	----------	-------------

### 5.2.3.10 spi\_transfer\_bidir()

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

**Description:**

A static function called automatically 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**

<code>transfer</code>	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
------	------------------	----------	-------------

**5.2.3.11 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

<i>transfer</i>	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
------	------------------	----------	-------------

#### 5.2.3.12 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

<i>transfer</i>	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
------	------------------	----------	-------------

**5.2.3.13 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**

<i>transfer</i>	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
------	------------------	----------	-------------

#### 5.2.3.14 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

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.15 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

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.16 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**

<i>transfer</i>	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
------	------------------	----------	-------------

**5.2.3.17 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
------	------------------	----------	-------------

**5.2.3.18 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-CONDITION: The correct buffer interrupt (RXNEIE or TXEIE) has been enabled POST-CONDITION: The SPI Enable (SPE) has been switched on

**Parameters**

<i>transfer</i>	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
------	------------------	----------	-------------

**5.2.3.19 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**

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.20 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**

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.21 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

<i>transfer</i>	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
------	------------------	----------	-------------

### 5.2.3.22 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**

<i>transfer</i>	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
------	------------------	----------	-------------

**5.2.3.23 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**

<i>transfer</i>	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
------	------------------	----------	-------------

**5.2.3.24 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**

<i>transfer</i>	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
------	------------------	----------	-------------

## 5.2.4 Variable Documentation

### 5.2.4.1 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

### 5.2.4.2 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

### 5.2.4.3 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

### 5.2.4.4 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

### 5.2.4.5 spi\_interrupt\_callbacks

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

Static array of callback functions mapped to each spi device

### 5.2.4.6 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

### 5.2.4.7 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

## 5.2.4.8 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

## 5.2.4.9 SPI\_TXCRCR

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

**Initial value:**

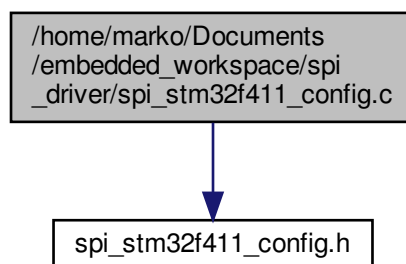
```
=
{
    (uint16_t *)SPI1_BASE + 0x18UL, (uint16_t *)SPI2_BASE + 0x18UL,
    (uint16_t *)SPI3_BASE + 0x18UL, (uint16_t *)SPI4_BASE + 0x18UL,
    (uint16_t *)SPI5_BASE + 0x18UL
}
```

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]
```

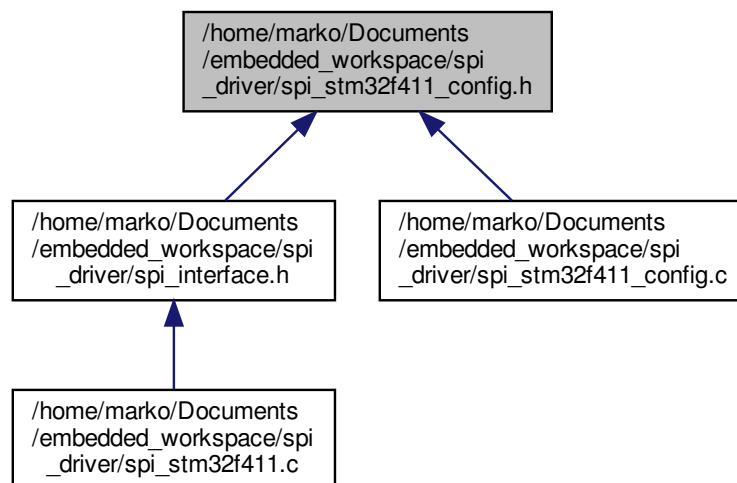
**Initial value:**

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

## 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

<code>UNIDIR_FULL_DUPLEX</code>	Two data wire spi, with simultaneous data reception and transmission
<code>UNIDIR_RXONLY</code>	Two data wire spi, but with reception only
<code>BIDIR_MODE</code>	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 default 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

<code>ERROR_INTERRUPT</code>	Interrupt generated upon a communication error
<code>RXNE_INTERRUPT</code>	Interrupt generated upon having non-read received data
<code>TXE_INTERRUPT</code>	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
------	------------------	----------	-------------

