# Timer Driver

Version 1.0.0

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# GPIO_DRIVER

After reading Jacob Beningo's book, Reusable Firmware Development, I've decided to begin the arduous process of building my own easily portable HAL to use for future projects. It feels as if all I've been developing for ages at this point is drivers.

The general principle is as follows:

- A general gpio_interface.h defines the api which will be exposed to applications. It will be this file which is included by the application. It is designed in a way to be 100% non-platform dependent. Changes required may be the modification of uint32_t types to uint16_t to match an architecture.
- The micrcontroller specific gpio_stm32f4xx.c file contains an MCU specific implementation of the peripheral. Accompanying it are a config .c/.h pair. These define a table of init structures for each instance of the peripheral, as well as all relevant typedefs. These files will need to be changed to port the driver. Time to port a gpio driver seems to be less than a day's worth of work.
- To port the driver: simply prepare the MCU specific c and config files and set gpio_interface.h to include the appropriate xxxxx_config.h file, and exchange the source files.

# 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 gpio_config_t Struct Reference

`#include <gpio_stm32f411_config.h>`

### Data Fields

- gpio_pin_t pin
- gpio_mode_t mode
- gpio_resistor_t resistor
- gpio_output_type_t output_type
- gpio_output_speed_t output_speed
- gpio_mux_t mux

### 4.1.1 Detailed Description

Configuration structure holding all values needed to configure a pin.

### 4.1.2 Field Documentation

#### 4.1.2.1 mode

`gpio_mode_t mode`

Selected pin mode

#### 4.1.2.2 mux

`gpio_mux_t mux`

Multiplexer signal used to select alternate function (only relevant in AF mode)

#### 4.1.2.3 output_speed

```
gpio_output_speed_t output_speed
```

Output speed (only relevant in output mode)

#### 4.1.2.4 output_type

```
gpio_output_type_t output_type
```

Output type (only relevant in output mode)

#### 4.1.2.5 pin

```
gpio_pin_t pin
```

Which pin is being configured

#### 4.1.2.6 resistor

```
gpio_resistor_t resistor
```

Pull- up/down selection

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

- gpio_stm32f411_config.h

# Chapter 5

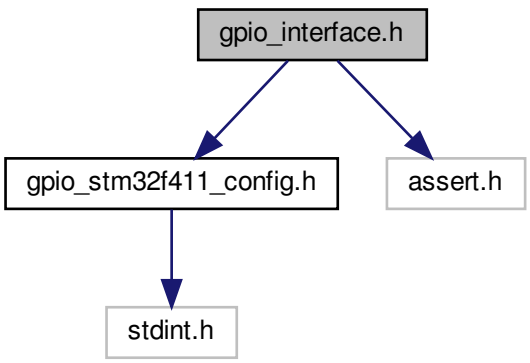
# File Documentation

## 5.1 gpio_interface.h File Reference

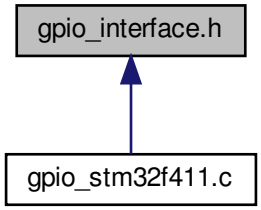
An interface which allows for a level of modularity when using the gpio between different architectures. Usage Notes: To port the driver, change the include config line to point instead to the new config.h file you want. The config file and the interface file together should provide all definitions to implement what you need in the machine specific c implementation file.

```
#include "gpio_stm32f411_config.h"
#include "assert.h"
```

Include dependency graph for gpio_interface.h:



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




## Functions

- void gpio_init (gpio_config_t *config_table)
- gpio_pin_state_t gpio_pin_read (gpio_pin_t pin)
- void gpio_pin_write (gpio_pin_t pin, gpio_pin_state_t value)
- void gpio_pin_toggle (gpio_pin_t pin)
- void gpio_register_write (uint32_t gpio_register, uint32_t value)
- uint32_t gpio_register_read (uint32_t gpio_register)

### 5.1.1 Detailed Description

An interface which allows for a level of modularity when using the gpio between different architectures. Usage Notes: To port the driver, change the include config line to point instead to the new config.h file you want. The config file and the interface file together should provide all definitions to implement what you need in the machine specific c implementation file.

### 5.1.2 Function Documentation

#### 5.1.2.1 gpio_init()

```
void gpio_init (
            gpio_config_t * config_table )
```

**Description:**

This function is used to initialise the gpio based on the configuration table defined in the gpio_stm32f411_config.c

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
PRE-CONDITION: PINS_PER_PORT > 0
PRE-CONDITION: NUMBER_OF_PORTS > 0
PRE-CONDITION: The RCC clocks for all planned ports must be configured and enabled.

POST-CONDITION: The GPIO is ready for use with all active pins set up.

**Parameters**

| | |
|---|---|
| *config_table* | is a pointer to the configuration table that contains the initialisation structures for each planned gpio pin. |

**Returns**

void

**Example:**

```
const gpio_config_t *gpio_config = gpio_config_get();
gpio_init(gpio_config);
```

**See also**

gpio_config_get

**- CHANGE HISTORY -**

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

#### 5.1.2.2 gpio_pin_read()

```
gpio_pin_state_t gpio_pin_read (
            gpio_pin_t pin )
```

**Description:**

This function reads the current state of the selected pin, regardless of whether it is in input or output mode.

PRE-CONDITION: gpio_init() has run successfully with the selected pin configured within the config table

POST-CONDITION: The return value contains the requested pin state in 1/0 form.

**Parameters**

| | |
|---|---|
| *pin* | is a member of the gpio_pin_t enumeration typedef |

**Returns**

gpio_pin_state_t containing the pin's current state

**Example:**

```
gpio_pin_state_t current_state = gpio_pin_read(GPIO_E_4);
```

**See also**

gpio_pin_write
gpio_pin_toggle

**- CHANGE HISTORY -**

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

#### 5.1.2.3 gpio_pin_toggle()

```
void gpio_pin_toggle (
            gpio_pin_t pin )
```

**Description:** Toggles the state of the desired pin operating in output mode.

PRE-CONDITION: gpio_init has been carried out and configured the pin in output mode.

POST-CONDITION: The pin takes on the opposite state.

**Parameters**

| | |
|---|---|
| *pin* | is the pin whose state we wish to change |

**Returns**

void

**Example:**

```
gpio_pin_toggle(GPIO_D_15);
```

**See also**

gpio_pin_read
gpio_pin_write

**- CHANGE HISTORY -**

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

#### 5.1.2.4 gpio_pin_write()

```
void gpio_pin_write (
            gpio_pin_t pin,
            gpio_pin_state_t value )
```

**Description:** Writes the desired state to the pin operating in output mode.

PRE-CONDITION: gpio_init has been carried out and configured the pin in output mode.

POST-CONDITION: The pin takes on the desired state.

**Parameters**

| | |
|---|---|
| *pin* | is the pin whose state we wish to change |
| *value* | is the state which we wish the pin to assume |

**Returns**

void

**Example:**

```
gpio_pin_write(GPIO_C_0, GPIO_PIN_HIGH);
```

**See also**

gpio_pin_read
gpio_pin_toggle

**- CHANGE HISTORY -**

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

#### 5.1.2.5 gpio_register_read()

```
uint32_t gpio_register_read (
            uint32_t gpio_register )
```

**Description:** Reads a the value of the selected GPIO register in the GPIO memory space. This function can be used within a greater super function to access more advanced features of the GPIO, such as the LOCK.

PRE-CONDITION: The value of GPIO_register lies within the memory map defined region dedicated to GPIO

POST-CONDITION: The registers contents are returned

**Parameters**

| | |
|---|---|
| *gpio_register* | whose contents we wish to read |

**Returns**

uint32_t the value within the register

**Example:**

```
uint32_t contents = gpio_register_read(GPIOD_BASE + 0x1CUL);
```

**See also**

gpio_register_write

**- CHANGE HISTORY -**

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

#### 5.1.2.6 gpio_register_write()

```
void gpio_register_write (
            uint32_t gpio_register,
            uint32_t value )
```

**Description:** Writes a desired value to the selected GPIO register in the GPIO memory space. This function can be used within a greater super function to access more advanced features of the GPIO, such as the LOCK.

PRE-CONDITION: The value of GPIO_register lies within the memory map defined region dedicated to GPIO

POST-CONDITION: The register has been modified.

**Parameters**

| | |
|---|---|
| *gpio_register* | is the register whose contents we wish to change |
| *value* | is the state which we wish the pin to assume |

**Returns**

void

**Example:**

```
gpio_register_write(GPIOD_BASE + 0x1CUL, 0xDEADBEEF);
```

**See also**

gpio_register_read

**- CHANGE HISTORY -**

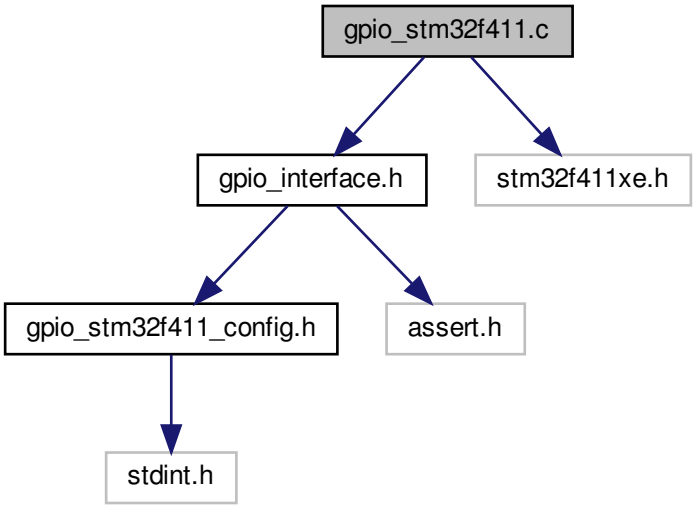
| Date | Software Version | Initials | Description |
|---|---|---|---|

## 5.2 gpio_stm32f411.c File Reference

machine specific implementation of gpio

```
#include "gpio_interface.h"
#include "stm32f411xe.h"
```

Include dependency graph for gpio_stm32f411.c:




### Macros

- #define PINS_PER_PORT 16
- #define NUM_GPIO_PORTS NUM_GPIO_PINS/PINS_PER_PORT
- #define MODERy_WIDTH 0x03UL
- #define OTYPERy_WIDTH 0x01UL
- #define OSPEEDRy_WIDTH 0x03UL
- #define PUPDRy_WIDTH 0x03UL
- #define AFLHRy_WIDTH 0x0FUL

### Functions

- void gpio_init (gpio_config_t *config_table)
- gpio_pin_state_t gpio_pin_read (gpio_pin_t pin)
- void gpio_pin_write (gpio_pin_t pin, gpio_pin_state_t value)
- void gpio_pin_toggle (gpio_pin_t pin)
- void gpio_register_write (uint32_t gpio_register, uint32_t value)
- uint32_t gpio_register_read (uint32_t gpio_register)

### Variables

- const uint32_t ACTIVE_GPIO_PINS

#### 5.2.1 Detailed Description

machine specific implementation of gpio

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 AFLHRy_WIDTH

```
#define AFLHRy_WIDTH 0x0FUL
```

The width of a the bitfield controlling the alternate function MUX

#### 5.2.2.2 MODERy_WIDTH

```
#define MODERy_WIDTH 0x03UL
```

The width of the bitfield controlling a pin's mode

#### 5.2.2.3 NUM_GPIO_PORTS

```
#define NUM_GPIO_PORTS NUM_GPIO_PINS/PINS_PER_PORT
```

Number of letter named ports

#### 5.2.2.4 OSPEEDRy_WIDTH

```
#define OSPEEDRy_WIDTH 0x03UL
```

The width of the bitfield controlling the output speed of a pin

#### 5.2.2.5 OTYPERy_WIDTH

```
#define OTYPERy_WIDTH 0x01UL
```

The width of the bitfield controlling the output type of a pin

#### 5.2.2.6 PINS_PER_PORT

```
#define PINS_PER_PORT 16
```

The number of pins per letter named port

#### 5.2.2.7 PUPDRy_WIDTH

```
#define PUPDRy_WIDTH 0x03UL
```

The width of the bitfield controlling pull up/pull down selection

### 5.2.3 Function Documentation

#### 5.2.3.1 gpio_init()

```
void gpio_init (
            gpio_config_t * config_table )
```

**Description:**

This function is used to initialise the gpio based on the configuration table defined in the gpio_stm32f411_config.c

PRE-CONDITION: Configuration table needs to populated (sizeof > 0)
PRE-CONDITION: PINS_PER_PORT > 0
PRE-CONDITION: NUMBER_OF_PORTS > 0
PRE-CONDITION: The RCC clocks for all planned ports must be configured and enabled.

POST-CONDITION: The GPIO is ready for use with all active pins set up.

**Parameters**

| | |
|---|---|
| *config_table* | is a pointer to the configuration table that contains the initialisation structures for each planned gpio pin. |

**Returns**

void

**Example:**

```
const gpio_config_t *gpio_config = gpio_config_get();
gpio_init(gpio_config);
```

**See also**

gpio_config_get

**- CHANGE HISTORY -**

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

#### 5.2.3.2 gpio_pin_read()

```
gpio_pin_state_t gpio_pin_read (
            gpio_pin_t pin )
```

**Description:**

This function reads the current state of the selected pin, regardless of whether it is in input or output mode.

PRE-CONDITION: gpio_init() has run successfully with the selected pin configured within the config table

POST-CONDITION: The return value contains the requested pin state in 1/0 form.

**Parameters**

| | |
|---|---|
| *pin* | is a member of the gpio_pin_t enumeration typedef |

**Returns**

gpio_pin_state_t containing the pin's current state

**Example:**

```
gpio_pin_state_t current_state = gpio_pin_read(GPIO_E_4);
```

**See also**

gpio_pin_write
gpio_pin_toggle

**- CHANGE HISTORY -**

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

#### 5.2.3.3 gpio_pin_toggle()

```
void gpio_pin_toggle (
            gpio_pin_t pin )
```

**Description:** Toggles the state of the desired pin operating in output mode.

PRE-CONDITION: gpio_init has been carried out and configured the pin in output mode.

POST-CONDITION: The pin takes on the opposite state.

**Parameters**

| | |
|---|---|
| *pin* | is the pin whose state we wish to change |

**Returns**

void

**Example:**

```
gpio_pin_toggle(GPIO_D_15);
```

**See also**

gpio_pin_read
gpio_pin_write

**- CHANGE HISTORY -**

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

#### 5.2.3.4 gpio_pin_write()

```
void gpio_pin_write (
            gpio_pin_t pin,
            gpio_pin_state_t value )
```

**Description:** Writes the desired state to the pin operating in output mode.

PRE-CONDITION: gpio_init has been carried out and configured the pin in output mode.

POST-CONDITION: The pin takes on the desired state.

**Parameters**

| | |
|---|---|
| *pin* | is the pin whose state we wish to change |
| *value* | is the state which we wish the pin to assume |

**Returns**

void

**Example:**

```
gpio_pin_write(GPIO_C_0, GPIO_PIN_HIGH);
```

**See also**

gpio_pin_read
gpio_pin_toggle

**- CHANGE HISTORY -**

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

#### 5.2.3.5 gpio_register_read()

```
uint32_t gpio_register_read (
            uint32_t gpio_register )
```

**Description:** Reads a the value of the selected GPIO register in the GPIO memory space. This function can be used within a greater super function to access more advanced features of the GPIO, such as the LOCK.

PRE-CONDITION: The value of GPIO_register lies within the memory map defined region dedicated to GPIO

POST-CONDITION: The registers contents are returned

**Parameters**

| | |
|---|---|
| *gpio_register* | whose contents we wish to read |

**Returns**

uint32_t the value within the register

**Example:**

```
uint32_t contents = gpio_register_read(GPIOD_BASE + 0x1CUL);
```

**See also**

gpio_register_write

**- CHANGE HISTORY -**

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

#### 5.2.3.6 gpio_register_write()

```
void gpio_register_write (
            uint32_t gpio_register,
            uint32_t value )
```

**Description:** Writes a desired value to the selected GPIO register in the GPIO memory space. This function can be used within a greater super function to access more advanced features of the GPIO, such as the LOCK.

PRE-CONDITION: The value of GPIO_register lies within the memory map defined region dedicated to GPIO

POST-CONDITION: The register has been modified.

**Parameters**

| | |
|---|---|
| *gpio_register* | is the register whose contents we wish to change |
| *value* | is the state which we wish the pin to assume |

**Returns**

void

**Example:**

```
gpio_register_write(GPIOD_BASE + 0x1CUL, 0xDEADBEEF);
```

**See also**

gpio_register_read

**- CHANGE HISTORY -**

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

### 5.2.4 Variable Documentation

#### 5.2.4.1 ACTIVE_GPIO_PINS

```
const uint32_t ACTIVE_GPIO_PINS
```

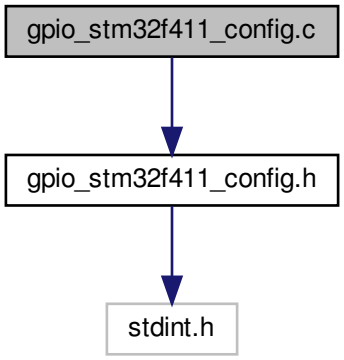
Defined in the appropriate .c config file. Prevents iteration over 64 pins when only a few are used

Prevents iteration over 64 pins when only a few are used

## 5.3 gpio_stm32f411_config.c File Reference

A file defining a config table which contains all information required by gpio_init to initialise the pins with the desired behaviour.

```
#include "gpio_stm32f411_config.h"
```
Include dependency graph for gpio_stm32f411_config.c:




### Functions

- const gpio_config_t * gpio_config_get (void)

### Variables

- const uint32_t ACTIVE_GPIO_PINS = sizeof(gpio_config_table)/sizeof(gpio_config_t)

### 5.3.1 Detailed Description

A file defining a config table which contains all information required by gpio_init to initialise the pins with the desired behaviour.

### 5.3.2 Function Documentation

#### 5.3.2.1 gpio_config_get()

```
const gpio_config_t* gpio_config_get (
            void  )
```

**Description:** Retrieves the config table for the gpio peripheral, normally hidden statically within the config.c file.

PRE-CONDITION: The config table has been populated/exists with a size greater than 0.

POST-CONDITION: The returned value points to the base of the config table

**Returns**

const gpio_config_t *

**Example:**

```
const gpio_config_t gpio_config_table = gpio_config_get(void);
gpio_init(gpio_config_table);
```

**See also**

gpio_init

**- CHANGE HISTORY -**

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

### 5.3.3 Variable Documentation

#### 5.3.3.1 ACTIVE_GPIO_PINS

```
const uint32_t ACTIVE_GPIO_PINS = sizeof(gpio_config_table)/sizeof(gpio_config_t)
```

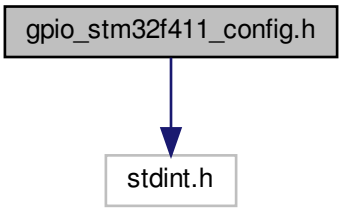
Prevents iteration over 64 pins when only a few are used

## 5.4 gpio_stm32f411_config.h File Reference

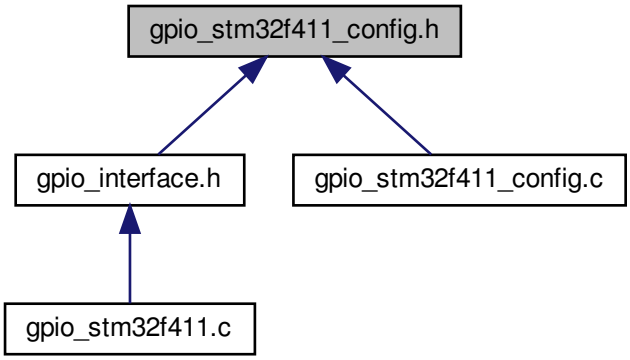
machine specific configuration enumerations and structures

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

Include dependency graph for gpio_stm32f411_config.h:




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




### Data Structures

- struct gpio_config_t

### Enumerations

- enum gpio_pin_t {
  **GPIO_A_0**, **GPIO_A_1**, **GPIO_A_2**, **GPIO_A_3**,
  **GPIO_A_4**, **GPIO_A_5**, **GPIO_A_6**, **GPIO_A_7**,
  **GPIO_A_8**, **GPIO_A_9**, **GPIO_A_10**, **GPIO_A_11**,
  **GPIO_A_12**, **GPIO_A_13**, **GPIO_A_14**, **GPIO_A_15**,
  **GPIO_B_0**, **GPIO_B_1**, **GPIO_B_2**, **GPIO_B_3**,
  **GPIO_B_4**, **GPIO_B_5**, **GPIO_B_6**, **GPIO_B_7**,
  **GPIO_B_8**, **GPIO_B_9**, **GPIO_B_10**, **GPIO_B_11**,
  **GPIO_B_12**, **GPIO_B_13**, **GPIO_B_14**, **GPIO_B_15**,
  **GPIO_C_0**, **GPIO_C_1**, **GPIO_C_2**, **GPIO_C_3**,
  **GPIO_C_4**, **GPIO_C_5**, **GPIO_C_6**, **GPIO_C_7**,
  **GPIO_C_8**, **GPIO_C_9**, **GPIO_C_10**, **GPIO_C_11**,
  **GPIO_C_12**, **GPIO_C_13**, **GPIO_C_14**, **GPIO_C_15**,
  **GPIO_D_0**, **GPIO_D_1**, **GPIO_D_2**, **GPIO_D_3**,
  **GPIO_D_4**, **GPIO_D_5**, **GPIO_D_6**, **GPIO_D_7**,
  **GPIO_D_8**, **GPIO_D_9**, **GPIO_D_10**, **GPIO_D_11**,
  **GPIO_D_12**, **GPIO_D_13**, **GPIO_D_14**, **GPIO_D_15**,
  **GPIO_E_0**, **GPIO_E_1**, **GPIO_E_2**, **GPIO_E_3**,
  **GPIO_E_4**, **GPIO_E_5**, **GPIO_E_6**, **GPIO_E_7**,
  **GPIO_E_8**, **GPIO_E_9**, **GPIO_E_10**, **GPIO_E_11**,
  **GPIO_E_12**, **GPIO_E_13**, **GPIO_E_14**, **GPIO_E_15**,
  **NUM_GPIO_PINS** }
- enum gpio_pin_state_t { **GPIO_PIN_LOW** = 0UL, **GPIO_PIN_HIGH** = 1UL }
- enum gpio_mode_t {
  GPIO_INPUT, GPIO_OUTPUT, GPIO_ALTERNATE_FUNCTION, GPIO_ANALOG,
  GPIO_MAX_MODE_OPTIONS }
- enum gpio_resistor_t { GPIO_NO_RESISTOR, GPIO_PULL_UP, GPIO_PULL_DOWN, GPIO_MAX_RES↩
  ISTOR_OPTIONS }
- enum gpio_output_type_t { GPIO_PUSH_PULL, GPIO_OPEN_DRAIN, GPIO_MAX_OUTPUT_OPTIONS }
- enum gpio_output_speed_t {
  GPIO_LOW_SPEED, GPIO_MED_SPEED, GPIO_FAST_SPEED, GPIO_HIGH_SPEED,
  GPIO_MAX_SPEED_OPTIONS }
- enum gpio_mux_t {
  **GPIO_AF_0**, **GPIO_AF_1**, **GPIO_AF_2**, **GPIO_AF_3**,
  **GPIO_AF_4**, **GPIO_AF_5**, **GPIO_AF_6**, **GPIO_AF_7**,
  **GPIO_AF_8**, **GPIO_AF_9**, **GPIO_AF_10**, **GPIO_AF_11**,
  **GPIO_AF_12**, **GPIO_AF_13**, **GPIO_AF_14**, **GPIO_AF_15**,
  **GPIO_MAX_AF_OPTIONS** }

### Functions

- const gpio_config_t * gpio_config_get (void)

#### 5.4.1 Detailed Description

machine specific configuration enumerations and structures

#### 5.4.2 Enumeration Type Documentation

#### 5.4.2.1 gpio_mode_t

```
enum gpio_mode_t
```

Contains all the modes a specific pin can be in.

**Enumerator**

| | |
|---|---|
| GPIO_INPUT | The pin functions as a digital input |
| GPIO_OUTPUT | The pin functions as a digital output |
| GPIO_ALTERNATE_FUNCTION | The pin is multiplexed to allow another peripheral to control it<br>**See also**<br>gpio_mux_t |
| GPIO_ANALOG | The pin works as an analog, defined by the ADC peripheral |
| GPIO_MAX_MODE_OPTIONS | Redundant extra option. Can be used for assertions in super robust implementations where the strength of enums is in question. |

#### 5.4.2.2 gpio_mux_t

`enum gpio_mux_t`

All alternate function values fed into the 4bit multiplexer. See Figure 17 in RM0383

#### 5.4.2.3 gpio_output_speed_t

`enum gpio_output_speed_t`

Contains speed options for a pin's output. Actual speed is a factor Vdd and capacitor selection. See pages 101-102 in the STM32F411xE datasheet for concrete numbers. All ranges given below are implementation sensitive

**Enumerator**

| | |
|---|---|
| GPIO_LOW_SPEED | Output speed is between 2-8MHz |
| GPIO_MED_SPEED | Output speed is between 12.5-50MHz |
| GPIO_FAST_SPEED | Output speed is between 25-100MHz |
| GPIO_HIGH_SPEED | Output speed is between 50-100MHz |
| GPIO_MAX_SPEED_OPTIONS | Redundant extra option. Can be used for assertions in super robust implementations where the strength of enums is in question. |

#### 5.4.2.4 gpio_output_type_t

`enum gpio_output_type_t`

Defines the electrical behaviour of an output pin.

**Enumerator**

| | |
|---|---|
| GPIO_PUSH_PULL | The pin can drive to electrical defined 1 and 0 (Vdd and GND) |
| GPIO_OPEN_DRAIN | The pin can only drive to GND. Output options are undefined and 0. |
| GPIO_MAX_OUTPUT_OPTIONS | Redundant extra option. Can be used for assertions in super robust implementations where the strength of enums is in question. |

#### 5.4.2.5 gpio_pin_state_t

```
enum gpio_pin_state_t
```

Contains both active states a pin can be in. Actual electrical behaviour depends on push-pull/open drain settings.

#### 5.4.2.6 gpio_pin_t

```
enum gpio_pin_t
```

Contains all of the gpio pins on all of the ports. Intermediary calculations are used to separate port and pins.

#### 5.4.2.7 gpio_resistor_t

```
enum gpio_resistor_t
```

Contains the resistor options over a pin for both input and output modes.

**Enumerator**

| | |
|---|---|
| GPIO_NO_RESISTOR | No resistor. Pin is undefined unless driven actively |
| GPIO_PULL_UP | Pull up resistor over pin. Will default to high unless driven |
| GPIO_PULL_DOWN | Pull down resistor over pin. Will default to low unless driven |
| GPIO_MAX_RESISTOR_OPTIONS | Redundant extra option. Can be used for assertions in super robust where the strength of enums is in question. |

### 5.4.3 Function Documentation

#### 5.4.3.1 gpio_config_get()

```
const gpio_config_t* gpio_config_get (
            void  )
```

**Description:** Retrieves the config table for the gpio peripheral, normally hidden statically within the config.c file.

PRE-CONDITION: The config table has been populated/exists with a size greater than 0.

POST-CONDITION: The returned value points to the base of the config table

**Returns**

const gpio_config_t *

**Example:**

```
const gpio_config_t gpio_config_table = gpio_config_get(void);
gpio_init(gpio_config_table);
```

**See also**

gpio_init

**- CHANGE HISTORY -**

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