

---

# **halmcu**

***Release 0.0.1***

**Kyunghwan Kwon**

**Oct 08, 2021**



# CONTENTS

<b>1</b>	<b>Getting Started</b>	<b>3</b>
<b>2</b>	<b>Architecture &amp; Design</b>	<b>7</b>
<b>3</b>	<b>API Reference</b>	<b>9</b>
<b>4</b>	<b>Examples</b>	<b>53</b>
<b>5</b>	<b>Supported Devices</b>	<b>55</b>
<b>6</b>	<b>Changelog</b>	<b>57</b>
<b>7</b>	<b>Indices and tables</b>	<b>59</b>
	<b>Index</b>	<b>61</b>



*halmcu* provides a developing environment for microcontrollers by including low level hardware drivers, libraries, and examples for peripherals.

See online documentation at [halmcu.readthedocs.io](http://halmcu.readthedocs.io)



## GETTING STARTED

### 1.1 Installing prerequisites

#### 1.1.1 GNU Arm Embedded Toolchain

1. Download the [GNU Arm Embedded Toolchain](#).
2. Locate it where desired
3. Add path to environment variable
4. Then you can check that ARM GCC is in your path:

```
$ arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Arm Embedded Toolchain 10-2020-q4-major) 10.2.1 20201103 (release)
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Alternatively you can do it with package manager:

- [MacOSX](#)
  - \$ brew install --cask gcc-arm-embedded
- Linux
  - \$ sudo apt-get install -y gcc-arm-none-eabi
- Windows
  - Download and run [the installer](#)

### 1.2 Setting up the development environment

#### 1.2.1 Visual Studio Code

1. Install [Visual Studio Code](#)
2. Install Visual Studio Code Plugins
  - [C/C++](#)
3. Press F1 to display the command palette
4. At the command palette prompt, enter `git cl`, select the **Git: Clone (Recursive)**

5. When prompted for the **Repository URL**, enter `https://github.com/onkwon/libabov-example.git`
6. And press F1 again. At the command palette prompt enter task, select the **Tasks: Run Build Task**

## 1.2.2 Embedded Studio

## 1.2.3 IAR

## 1.2.4 Keil

## 1.2.5 Other

### 1. Get *halmcu* into your project

```
$ cd ${YOUR_PROJECT_DIR}
$ git submodule add https://github.com/onkwon/halmcu.git ${YOUR_THIRD_PARTY_DIR}/halmcu
```

### 2. Add *halmcu* to your existing build system

#### Make

Please refer to a [build template](#).

#### Other

Please refer to a [custom build example](#).

1. Add the sources under `${YOUR_THIRD_PARTY_DIR}/halmcu/drivers/*.c` to your project
2. Add the sources under `${YOUR_THIRD_PARTY_DIR}/halmcu/devices/common/*.c` to your project
3. Add the sources under `${YOUR_THIRD_PARTY_DIR}/halmcu/devices/${VENDOR}/${DEVICE}/*.c` to your project
4. Add cpu architecture specific sources under `${YOUR_THIRD_PARTY_DIR}/halmcu/arch/${YOUR_ARCH}` to your project
5. Add `${YOUR_THIRD_PARTY_DIR}/halmcu/include` to the include paths for your project
6. Add CMSIS include path to the include paths for your project
7. Pass `DEVICE`, `ARCH`, and `HSE` definitions to your compiler



## 1.3 Running an example

## 1.4 Supported devices and peripherals

- Supported
- Not supported
- Unavailable in device
- Planned

	ABOV		STM32		
Peripheral	A31G1x	A33G	F1 <sup>1</sup>	F4	F7
ADC					
CAN					
CLK					
CRC					
DAC					
DMA					
ETH					
FLASH					
FSMC					
GPIO					
I2C					
PWR					
RTC					
SDIO					
SPI					
Timer					
UART					
USB					
WDT					

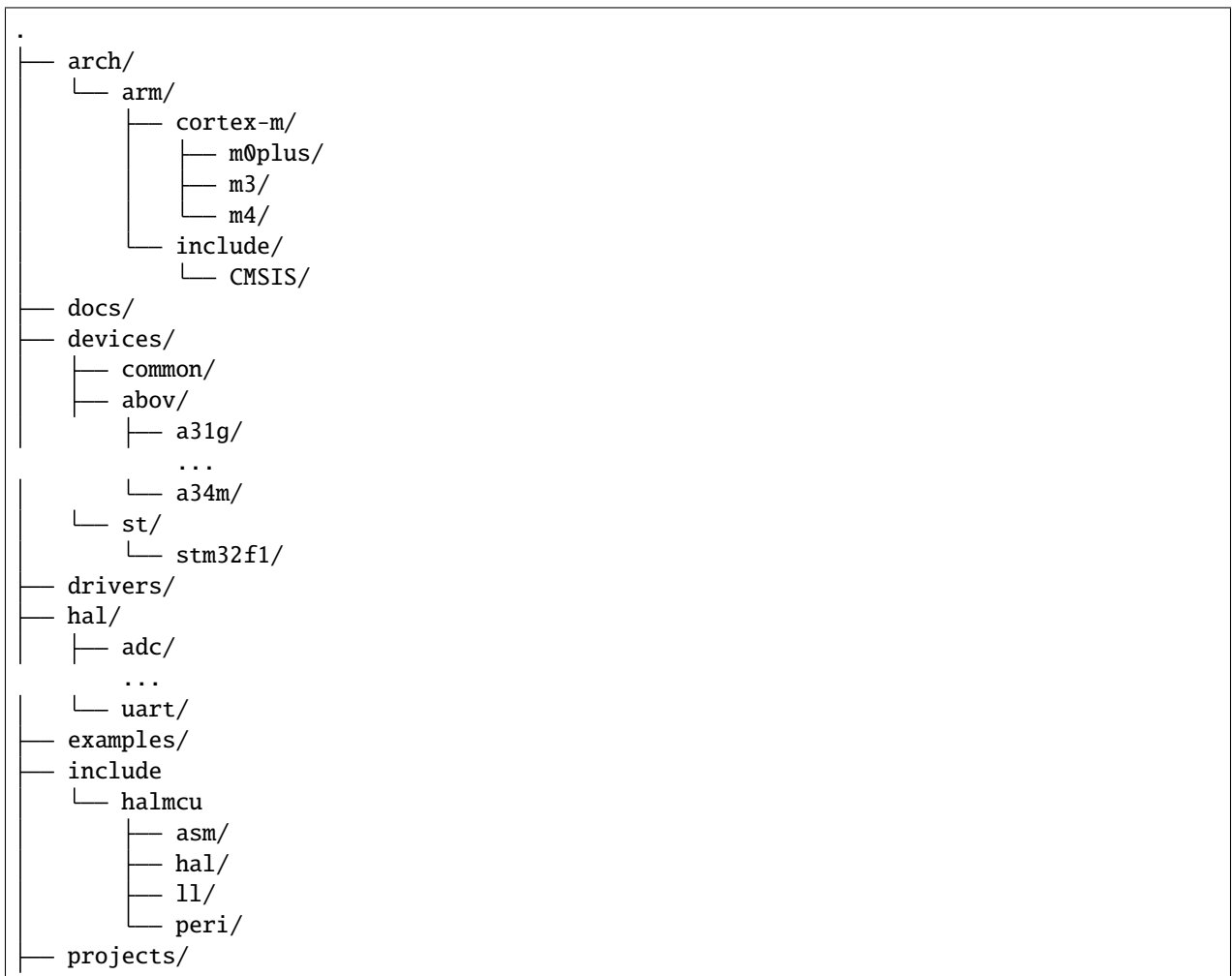
<sup>1</sup> Only high-density devices are supported at the moment.



## ARCHITECTURE & DESIGN

### 2.1 API

### 2.2 Directory Structure



(continues on next page)

└─

devices/  
tests/

## 2.3 Build System

`Make` build automation tool is used.

Additional tools like `Kbuild` or scripts are intentionally avoided not to increase complexity.

1. You should specify device you want to build for, something like: `make DEVICE=a33g`
2. It will automatically include a project Makefile according to the device. e.g. `projects/devices/a33g.mk`
3. In the project Makefile, all drivers it supports are included by `devices/a33g/devices.mk`
4. Then it decides what cpu architecture is of the device
5. And build based on the architecture with its Makefile, `arch/arm/cortex-m/m3/m3.mk`
  - `HALMCU_CFLAGS`
  - `HALMCU_WARNING_CFLAGS`
  - `CROSS_COMPILE`
  - `DEVICE`
  - `HSE`
  - `HSI`
  - `NDEBUG`

## 2.4 Software Layers

### 2.4.1 Low Level Layer

### 2.4.2 Hardwar Abstraction Layer

### 2.4.3 Driver Layer

## 2.5 Porting

- `HALMCU_PREFIX`

## API REFERENCE

### 3.1 Peripheral

#### 3.1.1 ADC

##### Examples

```
#include "abov/system.h"
#include "abov/hal/gpio.h"
#include "abov/hal/adc.h"

static void myadc_gpio_init(void)
{
    struct gpio_cfg cfg = {
        .mode = GPIO_MODE_ANALOG,
        .altfunc = true,
        .altfunc_number = 3,
    };
    gpio_open(PERIPH_GPIOA, 1, &cfg);
}

static void myadc_init(void)
{
    myadc_gpio_init();

    adc_enable(PERIPH_ADC);
    adc_set_mode(PERIPH_ADC, ADC_MODE_SINGLE_CONVERSION);
#ifdef DEFAULT_CONFIGURATION
    adc_set_clock_frequency(PERIPH_ADC, 1000000,
        clk_get_peripheral_clock_source_frequency(PERIPH_ADC));
    adc_set_trigger(PERIPH_ADC, ADC_TRIGGER_MANUAL);
#endif
    adc_select_channel(PERIPH_ADC, ADC_CHANNEL_1);
}

int main(void)
{
```

(continues on next page)

(continued from previous page)

```
myadc_init();

adc_start(PERIPH_ADC);

while (!adc_is_completed(PERIPH_ADC)) {
    /* waiting */
}

uint32_t adc_result = adc_get_measurement(PERIPH_ADC);

adc_clear_event(PERIPH_ADC, ADC_EVENT_COMPLETE);

return 0;
}
```

## HAL

### Functions

void **adc\_enable**(*periph\_t* adc)

void **adc\_disable**(*periph\_t* adc)

## LL

### Enums

enum **adc\_mode\_t**

*Values:*

enumerator **ADC\_MODE\_SINGLE\_CONVERSION**

enumerator **ADC\_MODE\_SINGLE\_CONVERSION\_MULTI\_CHANNEL**

enumerator **ADC\_MODE\_CONTINUOUS\_CONVERSION**

enumerator **ADC\_MODE\_CONTINUOUS\_CONVERSION\_MULTI\_CHANNEL**

enumerator **ADC\_MODE\_IDLE**

enum **adc\_channel\_t**

*Values:*

enumerator **ADC\_CHANNEL\_0**

enumerator **ADC\_CHANNEL\_1**

enumerator **ADC\_CHANNEL\_2**

enumerator **ADC\_CHANNEL\_3**

enumerator **ADC\_CHANNEL\_4**

enumerator **ADC\_CHANNEL\_5**  
enumerator **ADC\_CHANNEL\_6**  
enumerator **ADC\_CHANNEL\_7**  
enumerator **ADC\_CHANNEL\_8**  
enumerator **ADC\_CHANNEL\_9**  
enumerator **ADC\_CHANNEL\_10**  
enumerator **ADC\_CHANNEL\_11**  
enumerator **ADC\_CHANNEL\_12**  
enumerator **ADC\_CHANNEL\_13**  
enumerator **ADC\_CHANNEL\_14**  
enumerator **ADC\_CHANNEL\_15**  
enumerator **ADC\_CHANNEL\_MAX**  
enumerator **ADC\_CHANNEL\_MASK**

enum **adc\_trigger\_t**

*Values:*

enumerator **ADC\_TRIGGER\_MANUAL**  
enumerator **ADC\_TRIGGER\_TIMER0\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER1\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER2\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER3\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER4\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER5\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER6\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER7\_CC0**  
enumerator **ADC\_TRIGGER\_TIMER1\_CC1**  
enumerator **ADC\_TRIGGER\_TIMER1\_CC2**  
enumerator **ADC\_TRIGGER\_TIMER1\_CC3**  
enumerator **ADC\_TRIGGER\_TIMER2\_CC2**  
enumerator **ADC\_TRIGGER\_TIMER3\_CC1**  
enumerator **ADC\_TRIGGER\_TIMER3\_TRG0**  
enumerator **ADC\_TRIGGER\_EXTI11**  
enumerator **ADC\_TRIGGER\_MAX**

enum **adc\_event\_t**

*Values:*

enumerator **ADC\_EVENT\_NONE**  
enumerator **ADC\_EVENT\_BUSY**

enumerator **ADC\_EVENT\_COMPLETE**

enumerator **ADC\_EVENT\_MASK**

## Functions

void **adc\_reset**(*periph\_t* adc)

Reset ADC.

This function makes the given ADC the reset default state.

**Parameters** **adc** – [in] a peripheral enumerated in *periph\_t*

void **adc\_enable\_clock**(*periph\_t* adc)

Activate the ADC.

**Parameters** **adc** – [in] a peripheral enumerated in *periph\_t*

void **adc\_disable\_clock**(*periph\_t* adc)

Deactivate the ADC.

**Parameters** **adc** – [in] a peripheral enumerated in *periph\_t*

void **adc\_set\_clock\_frequency**(*periph\_t* adc, uint32\_t hz, uint32\_t pclk)

uint32\_t **adc\_get\_frequency**(*periph\_t* adc, uint32\_t pclk)

void **adc\_set\_mode**(*periph\_t* adc, *adc\_mode\_t* mode)

void **adc\_start**(*periph\_t* adc)

Start the ADC conversion.

**Parameters** **adc** – [in] a peripheral enumerated in *periph\_t*

void **adc\_stop**(*periph\_t* adc)

Stop the ADC conversion.

**Parameters** **adc** – [in] a peripheral enumerated in *periph\_t*

void **adc\_select\_channel**(*periph\_t* adc, *adc\_channel\_t* channel)

void **adc\_set\_trigger**(*periph\_t* adc, *adc\_trigger\_t* trigger)

uint32\_t **adc\_get\_measurement**(*periph\_t* adc)

void **adc\_enable\_irq**(*periph\_t* adc)

void **adc\_disable\_irq**(*periph\_t* adc)

*adc\_event\_t* **adc\_get\_event**(*periph\_t* adc)

void **adc\_clear\_event**(*periph\_t* adc, *adc\_event\_t* events)



bool **adc\_is\_busy**(*periph\_t* adc)

bool **adc\_is\_completed**(*periph\_t* adc)

void **adc\_set\_sample\_time**(*periph\_t* adc, *adc\_channel\_t* channel, uint32\_t cycle)

void **adc\_calibrate**(*periph\_t* adc)

### 3.1.2 Clock

#### Examples

```
#include "halmcu/hal/clk.h"
#include "halmcu/compiler.h"

int main(void)
{
    clk_init(CLK_HSI, 16000000);

    uint32_t hclk = clk_get_hclk_frequency();
    uint32_t pclk = clk_get_pclk_frequency();
    unused(hclk);
    unused(pclk);

    while (1) {
        /* hang */
    }

    return 0;
}
```

## HAL

### Functions

bool **clk\_init**(*clk\_source\_t* clock\_source, uint32\_t target\_hz)

## LL

### Enums

enum **clk\_source\_t**

Clock source type

*Values:*

enumerator **CLK\_LSI**

Low-speed internal oscillator

enumerator **CLK\_HSI**

High-speed internal oscillator

enumerator **CLK\_LSE**

Low-speed external oscillator

enumerator **CLK\_HSE**

High-speed external oscillator

enumerator **CLK\_PLL**

PLL

enumerator **CLK\_PLL\_BYPASS**

PLL bypass

### Functions

void **clk\_reset**(void)

Reset CLK unit.

This function makes CLK unit the reset state.

void **clk\_enable\_peripheral**(*periph\_t* peri)

Enable peripheral clock.

**Parameters** *peri* – [in] enumerated in *periph\_t*

void **clk\_disable\_peripheral**(*periph\_t* peri)

Disable peripheral clock.

**Parameters** *peri* – [in] enumerated in *periph\_t*

void **clk\_enable\_source**(*clk\_source\_t* clk)

Enable clock source.

**Parameters** *clk* – [in] Clock source.

void **clk\_disable\_source**(*clk\_source\_t* clk)

Disable clock source.

**Parameters** *clk* – [in] Clock source.

void **clk\_set\_source**(*clk\_source\_t* clk)

Select main clock source.

```

clk_source_t clk_get_source(void)
    Get main clock source.

bool clk_set_pll_frequency(clk_source_t clk, clk_source_t clkin, uint32_t hz)
    Set frequency.

void clk_start_pll(void)
    Enable PLL.

void clk_stop_pll(void)
    Disable PLL.

bool clk_is_pll_locked(void)
    Check if PLL is locked.

uint32_t clk_get_hclk_frequency(void)
    Get processor clock frequency in Hz.

uint32_t clk_get_pclk_frequency(void)
    Get PCLK frequency in Hz.

uint32_t clk_get_frequency(clk_source_t clk)
    Get clock frequency in Hz.

clk_source_t clk_get_peripheral_clock_source(periph_t peri)
    Get peripheral clock source.

void clk_set_peripheral_clock_source(periph_t peri, clk_source_t clk)

uint32_t clk_get_peripheral_clock_source_frequency(periph_t peri)

void clk_enable_output(void)

void clk_disable_output(void)

void clk_set_output_prescaler(uint32_t div_factor)

void clk_set_output_source(clk_source_t clk)

```

### 3.1.3 GPIO

#### Examples

```

#include "abov/hal/gpio.h"
#include "abov/delay.h"

#define LED_PORT          PERIPH_GPIOD
#define LED_PIN           1

int main(void)

```

(continues on next page)

(continued from previous page)

```

{
  #if 0
    gpio_open_output(LED_PORT, LED_PIN, GPIO_MODE_PUSHPULL);
  #else
    struct gpio_cfg cfg = { GPIO_MODE_PUSHPULL, };
    gpio_open(LED_PORT, LED_PIN, &cfg);
  #endif

    while (1) {
        gpio_write(LED_PORT, LED_PIN, 1);
        udelay(5000000);
        gpio_write(LED_PORT, LED_PIN, 0);
        udelay(5000000);
    }

    return 0;
}

```

## HAL

### Functions

void **gpio\_open**(*periph\_t* port, uint32\_t pin, const struct *gpio\_cfg* \*cfg)  
Initialize the given GPIO pin to the specified mode.

#### Parameters

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0
- **cfg** – [in] configuration

**Returns** true on success

void **gpio\_open\_output**(*periph\_t* port, uint32\_t pin, *gpio\_mode\_t* mode)  
Initialize the given GPIO pin to the specified mode.

#### Parameters

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0
- **mode** – [in] sets gpio operation mode

**Returns** true on success

void **gpio\_close**(*periph\_t* port, uint32\_t pin)  
Deinitialize the given GPIO pin.

#### Parameters

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0

**Returns** true on success

struct **gpio\_cfg**  
*#include <gpio.h>* GPIO configuration

### Public Members

*gpio\_mode\_t* **mode**  
*gpio\_irq\_t* **irq**  
*gpio\_speed\_t* **speed**  
 bool **altfunc**  
 int **altfunc\_number**  
 uint32\_t **debounce**

## LL

### Enums

enum **gpio\_mode\_t**  
 GPIO mode

*Values:*

enumerator **GPIO\_MODE\_PUSH\_PULL**  
 Output only mode

enumerator **GPIO\_MODE\_INPUT**  
 Input only mode

enumerator **GPIO\_MODE\_INPUT\_PULLUP**  
 Input only mode with pull-up function

enumerator **GPIO\_MODE\_INPUT\_PULLDOWN**  
 Input only mode with pull-down function

enumerator **GPIO\_MODE\_OPENDRAIN**  
 Input/output open-drain mode

enumerator **GPIO\_MODE\_OPENDRAIN\_PULLUP**  
 Input/output open-drain mode with pull-up function

enumerator **GPIO\_MODE\_OPENDRAIN\_PULLDOWN**  
 Input/output open-drain mode with pull-down function

enumerator **GPIO\_MODE\_ANALOG**  
 Analog function

enum **gpio\_irq\_t**  
 GPIO interrupt type

*Values:*

enumerator **GPIO\_IRQ\_NONE**

enumerator **GPIO\_IRQ\_EDGE\_RISING**  
Rising edge interrupt

enumerator **GPIO\_IRQ\_EDGE\_FALLING**  
Falling edge interrupt

enumerator **GPIO\_IRQ\_EDGE\_ANY**  
Both rising and falling edge interrupts

enumerator **GPIO\_IRQ\_LEVEL\_HIGH**  
Logic level high interrupt

enumerator **GPIO\_IRQ\_LEVEL\_LOW**  
Logic level low interrupt

enum **gpio\_speed\_t**

*Values:*

enumerator **GPIO\_SPEED\_LOW**

enumerator **GPIO\_SPEED\_MID**

enumerator **GPIO\_SPEED\_HIGH**

## Functions

void **gpio\_reset**(*periph\_t* port)  
Reset GPIO port.

This function makes GPIO port the reset state.

**Parameters** **port** – [in] GPIO port enumerated in *periph\_t*

void **gpio\_enable\_port**(*periph\_t* port)  
Enable the given port.

**Parameters** **port** – [in] GPIO port enumerated in *periph\_t*

void **gpio\_disable\_port**(*periph\_t* port)  
Disable the given port.

**Parameters** **port** – [in] GPIO port enumerated in *periph\_t*

void **gpio\_set\_mode**(*periph\_t* port, uint32\_t pin, *gpio\_mode\_t* mode)

void **gpio\_set\_altfunc**(*periph\_t* port, uint32\_t pin, int altfunc)  
Select GPIO alternate function.

Call this function after :c:func:gpio\_set\_mode

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0

- **altfunc** – [in] alternate function number

void **gpio\_set\_speed**(*periph\_t* port, uint32\_t pin, *gpio\_speed\_t* speed)

---

**Note:** Call this function after *gpio\_set\_altfunc*

---

void **gpio\_set\_debouncer**(*periph\_t* port, uint32\_t pin, uint32\_t pclk\_clocks)

**Parameters** **pclk\_clocks** – [in] pass 0 clock cycle to disable

void **gpio\_enable\_irq**(*periph\_t* port, uint32\_t pin, *gpio\_irq\_t* irq\_type)

Enable interrupt on the given GPIO pin.

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0
- **irq\_type** – [in] sets interrupt trigger type

void **gpio\_disable\_irq**(*periph\_t* port, uint32\_t pin)

Disable interrupt on the given GPIO pin.

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0

void **gpio\_clear\_event**(*periph\_t* port, uint32\_t pin)

Clear interrupt flag on the given GPIO pin.

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – GPIO number starting from 0

void **gpio\_write\_pin**(*periph\_t* port, uint32\_t pin, int value)

Write output level to the given GPIO pin.

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0
- **value** – [out] logic output level. Only 0(low) and 1(high) are possible

int **gpio\_read\_pin**(*periph\_t* port, uint32\_t pin)

Read the current input level from the given GPIO pin.

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **pin** – [in] GPIO number starting from 0

**Returns** 0 when the logic level is low. 1 when the logic level is high

void **gpio\_write\_port**(*periph\_t* port, int value)

Write a value to the given GPIO port.

**Parameters**

- **port** – [in] GPIO port enumerated in *periph\_t*
- **value** – to be written to the GPIO port

int **gpio\_read\_port**(*periph\_t* port)

Read the current value of the given GPIO port.

**Parameters** **port** – [in] GPIO port enumerated in *periph\_t*

**Returns** the value read from the specified GPIO port

### 3.1.4 I2C

#### Examples

```
#include "abov/system.h"
#include "abov/hal/uart.h"
#include "abov/hal/gpio.h"
#include "abov/irq.h"
#include "abov/delay.h"

#include "abov/ll/clk.h"

#define UART0_RX_PIN          (GPIOC + 8)
#define UART0_TX_PIN          (GPIOC + 9)

static uart_handle_t uart0_handle;

#include "printf.h"
void _putchar(char character)
{
    uart_write(PERI_UART0, &character, 1);
}

#include "libmcu/shell.h"
static size_t shell_read(void *buf, size_t bufsize)
{
    size_t result = uart_read(PERI_UART0, buf, bufsize);
    return result;
}

static size_t shell_write(const void *data, size_t datasize)
{
    size_t result = uart_write(PERI_UART0, data, datasize);
    return result;
}

#include "abov/hal/i2c.h"
#include "abov/ll/pwr.h"
#define I2C_SCL                (GPIOB + 14)
#define I2C_SDA                (GPIOB + 15)
static void myi2c_gpio_init(void)
```

(continues on next page)



(continued from previous page)

```

{
    gpio_open(I2C_SCL, GPIO_MODE_OPENDRAIN);
    gpio_open(I2C_SDA, GPIO_MODE_OPENDRAIN);
    gpio_set_altfunc(I2C_SCL, 1);
    gpio_set_altfunc(I2C_SDA, 1);
}
static void myi2c_init(void)
{
    myi2c_gpio_init();
}

uint32_t bccr, ccr, mr, wdt, pcsr;
static void system_clock_init(void)
{
    bccr = *(volatile uint32_t *)0x4000003c;
    ccr = *(volatile uint32_t *)0x40000030;
    mr = *(volatile uint32_t *)0x40000004;
    wdt = *(volatile uint32_t *)0x40000408;
    pcsr = *(volatile uint32_t *)0x40000040;

#if 1
    clk_enable_source(CLK_HSI);
#endif
#if 1
    clk_set_pll_frequency(CLK_PLL, CLK_HSI, 16000000);
#else
    gpio_open(GPIOC + 14, GPIO_MODE_ANALOG); // XTALO
    gpio_set_altfunc(GPIOC + 14, 1);
    gpio_open(GPIOC + 15, GPIO_MODE_ANALOG); // XTALI
    gpio_set_altfunc(GPIOC + 15, 1);
    clk_enable_source(CLK_HSE);
    clk_set_pll_frequency(CLK_PLL, CLK_HSE, 16000000);
#endif
    clk_enable_pll();
    clk_set_source(CLK_PLL);
    while (!clk_is_pll_locked());
#endif
}

static void system_init(void)
{
    system_clock_init();
}

static void uart_gpio_init(void)
{
    gpio_open(UART0_RX_PIN, GPIO_MODE_INPUT_PULLUP);
    gpio_open(UART0_TX_PIN, GPIO_MODE_PUSH_PULL);
    gpio_set_altfunc(UART0_RX_PIN, 1);
    gpio_set_altfunc(UART0_TX_PIN, 1);
}

int _sbrk_r;

```

(continues on next page)

(continued from previous page)

```

int main(void)
{
    system_init();

    uart_gpio_init();
    uart_init(PERI_UART0, &(struct uart_cfg) {
        .wordsize = UART_WORDSIZE_8,
        .stopbit = UART_STOPBIT_1,
        .parity = UART_PARITY_NONE,
        .baudrate = 115200, },
        &uart0_handle);

    uart_write(PERI_UART0, "Hello, World!\r\n", 15);
    printf("hclk %u, pclk %u\r\n", clk_get_hclk_frequency(), clk_get_pclk_
↪ frequency());

    myi2c_init();

    pwr_enable_peripheral(PERI_I2C0);
    clk_enable_peripheral(PERI_I2C0);
    i2c_reset(PERI_I2C0);
    i2c_set_frequency(PERI_I2C0, 100000, clk_get_pclk_frequency());

    //i2c_enable_irq(PERI_I2C0);
    //irq_enable(PERI_TO_IRQ(PERI_I2C0));

    i2c_start(PERI_I2C0, 0x69, 0);
    i2c_write_byte(PERI_I2C0, 0xf);
    i2c_start(PERI_I2C0, 0x69, 1);
    int rxd = i2c_read_byte(PERI_I2C0, false);
    i2c_stop(PERI_I2C0);

#if 0
    i2c_start(PERI_I2C0, 0x69, 0);
    i2c_write_byte(PERI_I2C0, 0x20);
    i2c_write_byte(PERI_I2C0, 0xf);
    i2c_stop(PERI_I2C0);
#endif

    i2c_start(PERI_I2C0, 0x69, 0);
    i2c_write_byte(PERI_I2C0, 0x20);
    i2c_start(PERI_I2C0, 0x69, 1);
    //int rxdata = i2c_read_byte(PERI_I2C0, false);
    int rxdata = i2c_read_byte(PERI_I2C0, true);
    rxdata = i2c_read_byte(PERI_I2C0, true);
    rxdata = i2c_read_byte(PERI_I2C0, false);
    i2c_stop(PERI_I2C0);

    printf("Recieved : %x\r\n", rxd);
    printf("Recieved : %x\r\n", rxdata);

    const shell_io_t io = {

```

(continues on next page)

(continued from previous page)

```

        .read = shell_read,
        .write = shell_write,
    };
    shell_run(&io);

    while (1) {
        uint8_t ch;
        size_t received = uart_read(PERI_UART0, &ch, sizeof(ch));
        if (received > 0) {
            uart_write(PERI_UART0, &ch, sizeof(ch));
        }
    }

    return 0;
}

void ISR_I2C0(void)
{
    printf("INT\r\n");
    #if 0
        switch (i2c_run_fsm()) {
        case I2C_FSM_ERROR:
            i2c_stop(PERI_I2C0);
            break;
        case I2C_FSM_START_SEND:
            i2c_set_txd(PERI_I2C0, 0xf);
            break;
        case I2C_FSM_START_RECV:
            break;
        case I2C_FSM_SENDING:
            i2c_set_txd(PERI_I2C0, 0xd3);
            i2c_disable_ack(PERI_I2C0);
            i2c_set_start(PERI_I2C0);
            break;
        case I2C_FSM_RECEIVING:
            printf("@gcall>tend/receiving: %x\r\n", i2c_read_byte(PERI_I2C0));
            i2c_stop(PERI_I2C0);
            break;
        default:
            break;
        }
    #endif
    #if 0
        uint32_t event = i2c_get_event(PERI_I2C0);
        if (event & 0x80) { //GCALL
            if (!(event & 1)) {
                printf("@gcall>noack\r\n");
                i2c_stop(PERI_I2C0);
            } else if (event & 2) { //
                printf("@gcall>tmode\r\n");
            }
        }
        i2c_set_txd(PERI_I2C0, 0xf);
        } else { //

```

(continues on next page)

(continued from previous page)

```

    }
    } else if (event & 0x40) { //TEND
        if (event & 0x2) { //
            // if no ack, stop
            printf("@gcall>tend|tmode\r\n");
i2c_set_txd(PERI_I2C0, 0xd3);
//i2c_enable_ack(PERI_I2C0);
i2c_set_start(PERI_I2C0);
        } else { //
            // nak
            printf("@gcall>tend|receiving: %x\r\n", i2c_read_byte(PERI_
↪ I2C0));
i2c_stop(PERI_I2C0);
        }
    }
#endif

    volatile uint32_t *sr = (volatile uint32_t *)0x40000a08;
    *sr = 0xff;
}

```

## HAL

**Warning:** doxygenfile: Cannot find file “halmcu/hal/i2c.h

## LL

## Enums

enum **i2c\_event\_t**

*Values:*

enumerator **I2C\_EVENT\_NONE**

enumerator **I2C\_EVENT\_STOP**

enumerator **I2C\_EVENT\_BUSY**

enumerator **I2C\_EVENT\_RX**

enumerator **I2C\_EVENT\_TX**

enumerator **I2C\_EVENT\_SLAVE**

enumerator **I2C\_EVENT\_COLLISION**

enumerator **I2C\_EVENT\_MASK**

## Functions

`void i2c_reset(periph_t i2c)`

`void i2c_enable(periph_t i2c)`

`void i2c_disable(periph_t i2c)`

`void i2c_set_frequency(periph_t i2c, uint32_t hz, uint32_t pclk)`

`void i2c_send_start(periph_t i2c)`

`void i2c_send_stop(periph_t i2c)`

`void i2c_enable_ack(periph_t i2c)`

`void i2c_disable_ack(periph_t i2c)`

`void i2c_write_byte(periph_t i2c, uint8_t value)`

`uint8_t i2c_read_byte(periph_t i2c)`

`bool i2c_is_busy(periph_t i2c)`

`bool i2c_has_started(periph_t i2c)`

`bool i2c_has_address_set(periph_t i2c)`

`bool i2c_has_transfer_completed(periph_t i2c)`

`bool i2c_has_received(periph_t i2c)`

`void i2c_enable_interrupt(periph_t i2c, i2c_event_t events)`

`void i2c_disable_interrupt(periph_t i2c, i2c_event_t events)`

`void i2c_start(periph_t i2c)`

`void i2c_stop(periph_t i2c)`

`void i2c_set_slave_address(periph_t i2c, uint16_t slave_addr)`

`void i2c_enable_irq(periph_t i2c)`

void **i2c\_disable\_irq**(*periph\_t* i2c)

*i2c\_event\_t* **i2c\_get\_event**(*periph\_t* i2c)

void **i2c\_clear\_event**(*periph\_t* i2c, *i2c\_event\_t* events)

### 3.1.5 Power

#### Examples

#### HAL

**Warning:** doxygenfile: Cannot find file “halmcu/hal/pwr.h

#### LL

#### Enums

enum **pwr\_mode\_t**

Power mode enumeration

*Values:*

enumerator **PWR\_MODE\_RUN**

Normal running mode

enumerator **PWR\_MODE\_SLEEP**

CPU clock gets stopped while core peripherals are kept running

enumerator **PWR\_MODE\_DEEP\_SLEEP**

Not only CPU clock but also most of peripheral clocks get stopped. External clock sources are also get off

enumerator **PWR\_MODE\_BLACKOUT**

RTC is the only one running. SRAM and peripheral registers are not preserved

#### Functions

void **pwr\_reset**(void)

Reset PWR unit.

This function makes PWR unit the reset state.

void **pwr\_reboot**(void)

Software system reset

```

uint32_t pwr_get_reboot_source(void)
    Get reboot source

void pwr_clear_reboot_source(uint32_t bitmask)
    Clear reboot source

void pwr_set_mode(pwr_mode_t sleep_mode)
    Set power(sleep) mode

void pwr_set_wakeup_source(periph_t peri)
    Set wakeup source

void pwr_clear_wakeup_source(periph_t peri)
    Clear wakeup source

uint32_t pwr_get_wakeup_source(void)
    Get wakeup source

void pwr_enable_peripheral(periph_t peri)
    Enable peripheral

void pwr_disable_peripheral(periph_t peri)
    Disable peripheral

```

### 3.1.6 SPI

#### Examples

```

#include "abov/system.h"
#include "abov/hal/gpio.h"
#include "abov/irq.h"

#include "abov/ll/pwr.h"
#include "abov/ll/clk.h"
#include "abov/ll/spi.h"

static void myspi_gpio_init(void)
{
    struct gpio_cfg cfg = {
        .mode = GPIO_MODE_PUSH_PULL,
        .altfunc = true,
        .altfunc_number = 1,
    };
    gpio_open(PERIPH_GPIOB, 10, &cfg); // SS
    gpio_open(PERIPH_GPIOB, 11, &cfg); // SCK
    gpio_open(PERIPH_GPIOB, 12, &cfg); // MOSI
    gpio_open(PERIPH_GPIOB, 13, &cfg); // MISO
}

static void myspi_init(void)
{
    myspi_gpio_init();
}

```

(continues on next page)

(continued from previous page)

```

    pwr_enable_peripheral(PERIPH_SPI0);
    clk_enable_peripheral(PERIPH_SPI0);

    spi_reset(PERIPH_SPI0);
    spi_set_mode(PERIPH_SPI0, SPI_MODE_MASTER);
    spi_set_frequency(PERIPH_SPI0, 10000000, clk_get_pclk_frequency());
    spi_set_clock_phase(PERIPH_SPI0, 0);
    spi_set_clock_polarity(PERIPH_SPI0, 0);
    spi_set_bitorder(PERIPH_SPI0, false);
    spi_set_data_width(PERIPH_SPI0, 8);

    spi_disable_chip_select(PERIPH_SPI0);
    spi_set_loopback(PERIPH_SPI0, true);
}

int main(void)
{
    myspi_init();
    spi_start(PERIPH_SPI0);

    while (!(spi_get_event(PERIPH_SPI0) & SPI_EVENT_TX_COMPLETE)) { /* waiting */ }
    spi_set_txd(PERIPH_SPI0, 0xA5);

    while (!(spi_get_event(PERIPH_SPI0) & SPI_EVENT_RX)) { /* waiting */ }

    uint32_t received = spi_get_rxd(PERIPH_SPI0);
    (void)received;

    return 0;
}

```

## HAL

### Functions

void **spi\_enable**(*periph\_t* spi)

void **spi\_disable**(*periph\_t* spi)

bool **spi\_init**(*periph\_t* spi, const struct *spi\_cfg* \*cfg)

void **spi\_deinit**(*periph\_t* spi)

void **spi\_start**(*periph\_t* spi)

void **spi\_stop**(*periph\_t* spi)



```
void spi_write(periph_t spi, uint32_t value)
```

```
uint32_t spi_read(periph_t spi)
```

```
uint32_t spi_write_read(periph_t spi, uint32_t value)
```

```
struct spi_cfg  
    #include <spi.h>
```

## Public Members

```
spi_mode_t mode  
uint32_t frequency  
int cpol  
int cpha  
uint32_t data_width  
bool lsb_first  
bool auto_chip_select  
bool interrupt
```

## LL

## Enums

```
enum spi_mode_t  
    Values:
```

```
    enumerator SPI_MODE_MASTER  
    enumerator SPI_MODE_SLAVE
```

```
enum spi_irq_t  
    Values:
```

```
    enumerator SPI_IRQ_NONE  
    enumerator SPI_IRQ_RX  
    enumerator SPI_IRQ_TX  
    enumerator SPI_IRQ_EDGE_CHAGNE  
    enumerator SPI_IRQ_OVERRUN  
    enumerator SPI_IRQ_FRAME_ERROR  
    enumerator SPI_IRQ_ERROR  
    enumerator SPI_IRQ_MASK
```

enum **spi\_event\_t**

*Values:*

enumerator **SPI\_EVENT\_NONE**

enumerator **SPI\_EVENT\_TX\_COMPLETE**

enumerator **SPI\_EVENT\_RX**

enumerator **SPI\_EVENT\_BUSY**

enumerator **SPI\_EVENT\_OVERRUN**

enumerator **SPI\_EVENT\_UNDERRUN**

enumerator **SPI\_EVENT\_CHIP\_SELECTED**

enumerator **SPI\_EVENT\_CHIP\_DESELECTED**

enumerator **SPI\_EVENT\_MODE\_FAULT**

enumerator **SPI\_EVENT\_CRC\_ERROR**

enumerator **SPI\_EVENT\_MASK**

## Functions

void **spi\_reset**(*periph\_t* spi)

Reset SPI

This function makes the given SPI the reset default state.

**Parameters** **spi** – [in] a peripheral enumerated in *periph\_t*

void **spi\_enable\_clock**(*periph\_t* spi)

void **spi\_disable\_clock**(*periph\_t* spi)

*spi\_event\_t* **spi\_get\_event**(*periph\_t* spi)

void **spi\_clear\_event**(*periph\_t* spi, *spi\_event\_t* events)

uint32\_t **spi\_get\_rxd**(*periph\_t* spi)

void **spi\_set\_txd**(*periph\_t* spi, uint32\_t value)

bool **spi\_is\_busy**(*periph\_t* spi)

bool **spi\_is\_tx\_completed**(*periph\_t* spi)

bool **spi\_has\_rx**(*periph\_t* spi)

void **spi\_clear\_rx\_buffer**(*periph\_t* spi)

void **spi\_clear\_tx\_buffer**(*periph\_t* spi)

void **spi\_enable\_irq**(*periph\_t* spi, *spi\_irq\_t* irqs)

void **spi\_disable\_irq**(*periph\_t* spi, *spi\_irq\_t* irqs)

void **spi\_enable\_chip\_select**(*periph\_t* spi)

void **spi\_disable\_chip\_select**(*periph\_t* spi)

void **spi\_set\_chip\_select\_mode**(*periph\_t* spi, bool manual)

void **spi\_set\_chip\_select\_level**(*periph\_t* spi, int level)

void **spi\_set\_chip\_select\_polarity**(*periph\_t* spi, int level)

void **spi\_set\_loopback**(*periph\_t* spi, bool enable)

void **spi\_set\_mode**(*periph\_t* spi, *spi\_mode\_t* mode)

void **spi\_set\_clock\_phase**(*periph\_t* spi, int cpha)

Set clock phase

#### Parameters

- **spi** – [in] a peripheral enumerated in *periph\_t*
- **cpha** – [in] 0 for the first clock edge or 1 for the second clock edge

void **spi\_set\_clock\_polarity**(*periph\_t* spi, int cpol)

Set clock polarity

#### Parameters

- **spi** – [in] a peripheral enumerated in *periph\_t*
- **cpol** – [in] 0 for low, 1 for high

void **spi\_set\_data\_width**(*periph\_t* spi, uint32\_t data\_width)

void **spi\_set\_bitorder**(*periph\_t* spi, bool lsb\_first)

void **spi\_set\_frequency**(*periph\_t* spi, uint32\_t hz, uint32\_t pclk)

void **spi\_set\_start\_delay**(*periph\_t* spi, uint32\_t nsck)

void **spi\_set\_stop\_delay**(*periph\_t* spi, uint32\_t nsck)

void **spi\_set\_burst\_delay**(*periph\_t* spi, uint32\_t nsck)

```
void spi_enable_crc(periph_t spi)
```

```
void spi_disable_crc(periph_t spi)
```

### 3.1.7 Timer

#### Examples

```
#include "abov/hal/gpio.h"
#include "abov/hal/timer.h"
#include "abov/irq.h"

#define MHZ                                1000000U
#define TIMER_SOURCE_CLOCK_MHZ            16U
#define TIMER_COUNTER_WIDTH               (1U << 16)

static volatile uint32_t measured_hz;

static void target_timer_gpio_init(void)
{
    struct gpio_cfg cfg = {
        .mode = GPIO_MODE_PUSH_PULL,
        .altfunc = true,
        .altfunc_number = 1,
    };
    gpio_open(PERIPH_GPIOB, 0, &cfg);
}

static void capture_timer_gpio_init(void)
{
    struct gpio_cfg cfg = {
        .mode = GPIO_MODE_INPUT,
        .altfunc = true,
        .altfunc_number = 1,
    };
    gpio_open(PERIPH_GPIOA, 7, &cfg);
}

static void set_target_timer_clock_source(void)
{
    timer_ll_set_clock_divider(PERIPH_TIMER0, 3); // 16MHz/64 = 250KHz
}

static void target_timer_init(void)
{
    target_timer_gpio_init();

    timer_init(PERIPH_TIMER0, &(struct timer_cfg) {
```

(continues on next page)

(continued from previous page)

```

        .mode = TIMER_MODE_NORMAL,
        .frequency = 1000,
        .set_clock_source = set_target_timer_clock_source, });
timer_set_reload(PERIPH_TIMER0, 1000 - 1); // every 1-sec

timer_start(PERIPH_TIMER0);
}

static void capture_timer_init(void)
{
    capture_timer_gpio_init();

    timer_init(PERIPH_TIMER1, &(struct timer_cfg) {
        .mode = TIMER_MODE_CAPTURE,
        .irq = (timer_event_t)(TIMER_EVENT_CC_0 |
                                TIMER_EVENT_CC_1 |
                                TIMER_EVENT_OVERFLOW),
        .irq_priority = 3, });
    timer_set_edge(PERIPH_TIMER1, TIMER_RISING_EDGE);

    timer_start(PERIPH_TIMER1);
}

int main(void)
{
    target_timer_init();
    capture_timer_init();

    while (1) {
        uint32_t hz = measured_hz / MHZ;
        uint32_t dec = measured_hz % MHZ;
        (void)hz;
        (void)dec;
    }

    return 0;
}

void ISR_TIMER1(void)
{
    static uint32_t ovf = 0;
    timer_event_t event = timer_get_event(PERIPH_TIMER1);

    if (event & TIMER_EVENT_OVERFLOW) {
        ovf++;
    }
    if (event & TIMER_EVENT_CC_0) {
        uint32_t captured1 =
            TIMER_COUNTER_WIDTH - timer_get_cc(PERIPH_TIMER1, 0);
        uint32_t captured2 = timer_get_cc(PERIPH_TIMER1, 1) + 1;
        uint32_t ticks =
            ovf * TIMER_COUNTER_WIDTH + captured1 + captured2;

```

(continues on next page)

(continued from previous page)

```

        uint32_t hz = ticks / TIMER_SOURCE_CLOCK_MHZ;

        measured_hz = hz;
        ovf = 0;
    }
    if (event & TIMER_EVENT_CC_1) {
    }

    timer_clear_event(PERIPH_TIMER1, (timer_event_t)
        (TIMER_EVENT_OVERFLOW |
         TIMER_EVENT_CC_0 |
         TIMER_EVENT_CC_1));
}

```

## HAL

### Functions

bool **timer\_init**(*periph\_t* timer, const struct *timer\_cfg* \*cfg)  
Initialize the timer.

#### Parameters

- **timer** – [in] a peripheral enumerated in *periph\_t*
- **cfg** – [in] pointer to the structure with the initial configuration

**Returns** true on success

void **timer\_deinit**(*periph\_t* timer)  
Deinitialize the timer.

**Parameters** **timer** – [in] a peripheral enumerated in *periph\_t*

struct **timer\_cfg**  
#include <timer.h> Timer configuration

### Public Members

*timer\_mode\_t* **mode**

uint32\_t **frequency**

*timer\_event\_t* **irq**

int **irq\_priority**

void (\***set\_clock\_source**)(void)

## LL

### Enums

enum **timer\_mode\_t**  
Timer mode type

*Values:*

enumerator **TIMER\_MODE\_NORMAL**

enumerator **TIMER\_MODE\_PWM**

enumerator **TIMER\_MODE\_ONESHOT**

enumerator **TIMER\_MODE\_CAPTURE**

enum **timer\_cc\_t**  
Timer pin type

*Values:*

enumerator **TIMER\_CC\_0**

enumerator **TIMER\_CC\_1**

enumerator **TIMER\_CC\_2**

enumerator **TIMER\_CC\_3**

enumerator **TIMER\_CC\_4**

enumerator **TIMER\_CC\_1N**

enumerator **TIMER\_CC\_2N**

enumerator **TIMER\_CC\_3N**

enumerator **TIMER\_CC\_4N**

enum **timer\_cc\_mode\_t**  
Timer CC mode type

*Values:*

enumerator **TIMER\_CC\_MODE\_NONE**

enumerator **TIMER\_CC\_MODE\_ACTIVE\_HIGH**

enumerator **TIMER\_CC\_MODE\_ACTIVE\_LOW**

enumerator **TIMER\_CC\_MODE\_TOGGLE**

enumerator **TIMER\_CC\_MODE\_LOW**

enumerator **TIMER\_CC\_MODE\_HIGH**

enumerator **TIMER\_CC\_MODE\_PWM\_ACTIVE\_HIGH**

enumerator **TIMER\_CC\_MODE\_PWM\_ACTIVE\_LOW**

enum **timer\_event\_t**  
Timer event type

*Values:*

enumerator **TIMER\_EVENT\_NONE**  
enumerator **TIMER\_EVENT\_OVERFLOW**  
enumerator **TIMER\_EVENT\_UNDERFLOW**  
enumerator **TIMER\_EVENT\_CC\_0**  
Capture/Compare Channel interrupt

enumerator **TIMER\_EVENT\_CC\_1**  
enumerator **TIMER\_EVENT\_CC\_2**  
enumerator **TIMER\_EVENT\_CC\_3**  
enumerator **TIMER\_EVENT\_CC\_4**  
enumerator **TIMER\_EVENT\_UPDATE**

enum **timer\_direction\_t**  
*Values:*

enumerator **TIMER\_DIRECTION\_UP**  
enumerator **TIMER\_DIRECTION\_DOWN**

## Functions

void **timer\_reset**(*periph\_t* peri)  
Reset the timer interface.

This function makes the given timer in the reset default state.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

void **timer\_set\_mode**(*periph\_t* peri, *timer\_mode\_t* mode)  
Set the timer mode.

### Parameters

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **mode** – [in] one of *timer\_mode\_t*

void **timer\_enable\_irq**(*periph\_t* peri, *timer\_event\_t* events)  
Enable interrupts on events for a timer.

### Parameters

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **events** – [in] to be enabled

void **timer\_disable\_irq**(*periph\_t* peri, *timer\_event\_t* events)  
Disable interrupts on events for a timer.

### Parameters

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **events** – [in] to be disabled

void **timer\_set\_clock\_divider**(*periph\_t* peri, uint32\_t div\_factor)  
Set the timer clock divider.



**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **div\_factor** – [in] clock divider

void **timer\_set\_counter**(*periph\_t* peri, uint32\_t value)

Set the timer counter.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **value** – [in] to be written

uint32\_t **timer\_get\_counter**(*periph\_t* peri)

Get the timer counter.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

**Returns** timer counter

uint32\_t **timer\_get\_frequency**(*periph\_t* peri, uint32\_t tclk)

void **timer\_start**(*periph\_t* peri)

Start the timer.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

void **timer\_stop**(*periph\_t* peri)

Stop the timer.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

void **timer\_clear\_event**(*periph\_t* peri, *timer\_event\_t* events)

Clear event flags.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **events** – [in] to be cleared

*timer\_event\_t* **timer\_get\_event**(*periph\_t* peri)

Read event flags.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

**Returns** events

void **timer\_set\_prescaler**(*periph\_t* peri, uint32\_t div\_factor)

Set the timer prescaler.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **div\_factor** – [in] prescaler values

uint32\_t **timer\_get\_prescaler**(*periph\_t* peri)

Get the timer prescaler.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

void **timer\_set\_reload**(*periph\_t* peri, uint32\_t value)

Set the timer period.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **value** – [in] to be written

uint32\_t **timer\_get\_reload**(*periph\_t* peri)

Get the timer period.

**Parameters** **peri** – [in] a peripheral enumerated in *periph\_t*

**Returns** timer period

void **timer\_set\_cc**(*periph\_t* peri, *timer\_cc\_t* cc, uint32\_t value)

Set Capture/Compare register.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel
- **value** – [in] to be written to the capture/compare register

uint32\_t **timer\_get\_cc**(*periph\_t* peri, *timer\_cc\_t* cc)

Get Capture/Compare register.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel

**Returns** capture/compare value

void **timer\_enable\_cc\_pin**(*periph\_t* peri, *timer\_cc\_t* cc)

Enable Capture/Compare pin.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel *timer\_cc\_t*

void **timer\_disable\_cc\_pin**(*periph\_t* peri, *timer\_cc\_t* cc)

Disable Capture/Compare pin.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel *timer\_cc\_t*

void **timer\_set\_cc\_pin**(*periph\_t* peri, *timer\_cc\_t* cc, uint32\_t value)

Set Capture/Compare pin map.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel *timer\_cc\_t*
- **value** – [in] 0 for output, 1 for TI1, 2 for TI2, and 3 for TRC

void **timer\_set\_cc\_pin\_mode**(*periph\_t* peri, *timer\_cc\_t* cc, *timer\_cc\_mode\_t* mode)

Set Capture/Compare pin mode.

**Parameters**

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel *timer\_cc\_t*
- **mode** – [in] capture/compare output mode in *timer\_cc\_mode\_t*

void **timer\_set\_cc\_pin\_polarity**(*periph\_t* peri, *timer\_cc\_t* cc, bool active\_high)  
Set Capture/Compare polarity.

#### Parameters

- **peri** – [in] a peripheral enumerated in *periph\_t*
- **cc** – [in] a number of capture/compare channel *timer\_cc\_t*
- **active\_high** – [in] active high on true

void **timer\_enable\_cc\_preload**(*periph\_t* peri, *timer\_cc\_t* cc)

void **timer\_disable\_cc\_preload**(*periph\_t* peri, *timer\_cc\_t* cc)

void **timer\_enable\_cc\_fastmode**(*periph\_t* peri, *timer\_cc\_t* cc)

void **timer\_disable\_cc\_fastmode**(*periph\_t* peri, *timer\_cc\_t* cc)

void **timer\_set\_cc\_prescaler**(*periph\_t* peri, *timer\_cc\_t* cc, uint32\_t value)

void **timer\_set\_cc\_filter**(*periph\_t* peri, *timer\_cc\_t* cc, uint32\_t value)

void **timer\_set\_counter\_direction**(*periph\_t* peri, *timer\_direction\_t* dir)

void **timer\_set\_counter\_alignment\_mode**(*periph\_t* peri, uint32\_t align)

void **timer\_set\_slave\_mode**(*periph\_t* peri, uint32\_t value)

## 3.1.8 UART

### Examples

```
#include "halmcu/system.h"
#include "halmcu/irq.h"
#include "halmcu/hal/uart.h"
#include "halmcu/hal/gpio.h"
#include "halmcu/hal/clk.h"

// #define POLLING

static uart_handle_t uart0_handle;
```

(continues on next page)

(continued from previous page)

```

static void uart_rx_handler(uint32_t flags)
{
    (void)flags;
    uint8_t c;
    uart_read(PERIPH_UART0, &c, 1);
    uart_write(PERIPH_UART0, "Received!\r\n", 11);
}

static void system_clock_init(void)
{
    clk_enable_source(CLK_HSI);
    clk_set_pll_frequency(CLK_PLL, CLK_HSI, 160000000);
    clk_start_pll();
    clk_set_source(CLK_PLL);
    while (!clk_is_pll_locked()) ;
}

static void uart_gpio_init(void)
{
    struct gpio_cfg cfg = {
        .mode = GPIO_MODE_PUSH_PULL,
        .altfunc = true,
        .altfunc_number = 1,
    };
    gpio_open(PERIPH_GPIOC, 9, &cfg); // TX

    cfg.mode = GPIO_MODE_INPUT_PULLUP;
    gpio_open(PERIPH_GPIOC, 8, &cfg); // RX
}

int main(void)
{
    system_clock_init();

    uart_gpio_init();
    uart_init(PERIPH_UART0, &(struct uart_cfg) {
        .wordsize = UART_WORDSIZE_8,
        .stopbit = UART_STOPBIT_1,
        .parity = UART_PARITY_NONE,
        .baudrate = 115200,
#ifdef POLLING
        .rx_interrupt = true,
#endif
    },
    &uart0_handle);
    uart_register_rx_handler(&uart0_handle, uart_rx_handler);

    uart_write(PERIPH_UART0, "Hello, World!\r\n", 15);

    while (1) {
        uint8_t ch;

```

(continues on next page)

(continued from previous page)

```

        size_t received = uart_read(PERIPH_UART0, &ch, sizeof(ch));
        if (received > 0) {
            uart_write(PERIPH_UART0, &ch, sizeof(ch));
        }
    }

    return 0;
}

void ISR_UART0(void)
{
    uart_default_isr(PERIPH_UART0, &uart0_handle);
}

```

## HAL

### Typedefs

typedef void (\***uart\_irq\_callback\_t**)(uint32\_t flags)  
UART handler type

### Functions

bool **uart\_init**(*periph\_t* uart, const struct *uart\_cfg* \*cfg, *uart\_handle\_t* \*handle)  
Initialize UART port with given configuration.

#### Parameters

- **uart** – [in] a peripheral enumerated in *periph\_t*
- **cfg** – [in] configuration
- **handle** – [in] handle of uart port

**Returns** true on success

void **uart\_deinit**(*periph\_t* uart)  
Deinitialize UART port.

**Parameters** **uart** – [in] a peripheral enumerated in *periph\_t*

size\_t **uart\_read**(*periph\_t* uart, void \*buf, size\_t bufsize)  
Read bytes from UART port.

#### Parameters

- **uart** – [in] a peripheral enumerated in *periph\_t*
- **buf** – [out] receive buffer address
- **bufsize** – [in] buffer size

size\_t **uart\_write**(*periph\_t* uart, const void \*data, size\_t datasize)  
Write data to UART port.

#### Parameters

- **uart** – [in] a peripheral enumerated in *periph\_t*

- **data** – [in] data buffer address
- **datasize** – [in] data size to send

int **uart\_read\_byte**(*periph\_t* port)

Read a byte from UART.

**Parameters** **port** – [in] a peripheral enumerated in *periph\_t*

**Returns** received byte

int **uart\_read\_byte\_nonblock**(*periph\_t* port)

Read a byte from UART.

---

**Note:** This function is non-blocking.

---

**Parameters** **port** – [in] a peripheral enumerated in *periph\_t*

**Returns** received bytes on success

**Returns** -1 – when no received data

void **uart\_write\_byte**(*periph\_t* port, uint8\_t val)

Write a byte to UART.

This function will block until the byte gets written into the hold register.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **val** – [in] value to write

void **uart\_register\_rx\_handler**(*uart\_handle\_t* \*handle, *uart\_irq\_callback\_t* handler)

Register rx interrupt handler.

**Parameters**

- **handle** – [in] handle of uart port
- **handler** – [in] rx interrupt handler

void **uart\_register\_tx\_handler**(*uart\_handle\_t* \*handle, *uart\_irq\_callback\_t* handler)

Register tx ready interrupt handler.

**Parameters**

- **handle** – [in] handle of uart port
- **handler** – [in] tx ready interrupt handler

void **uart\_register\_error\_handler**(*uart\_handle\_t* \*handle, *uart\_irq\_callback\_t* handler)

Register error interrupt handler.

**Parameters**

- **handle** – handle of uart port
- **handler** – error interrupt handler

void **uart\_default\_isr**(*periph\_t* uart, const *uart\_handle\_t* \*handle)

The default UART interrupt handler.

**Parameters**

- **uart** – [in] a peripheral enumerated in *periph\_t*
- **handle** – handle of uart port

struct **uart\_cfg**  
*#include <uart.h>* UART configuration

### Public Members

*uart\_wordsize\_t* **wordsize**

*uart\_stopbit\_t* **stopbit**

*uart\_parity\_t* **parity**

unsigned int **baudrate**

bool **rx\_interrupt**

bool **tx\_interrupt**

union **uart\_handle\_t**  
*#include <uart.h>* UART handle type

### Public Members

char **\_size**[sizeof(struct *uart\_cfg*) + 12]

long **\_align**

## LL

### Enums

enum **uart\_parity\_t**  
 UART parity enumeration  
*Values:*

enumerator **UART\_PARITY\_NONE**

enumerator **UART\_PARITY\_ODD**

enumerator **UART\_PARITY\_EVEN**

enum **uart\_stopbit\_t**  
 UART stopbit enumeration  
*Values:*

enumerator **UART\_STOPBIT\_1**

enumerator **UART\_STOPBIT\_1\_5**

enumerator **UART\_STOPBIT\_2**

enum **uart\_wordsize\_t**

UART wordsize enumeration

*Values:*

enumerator **UART\_WORDSIZE\_8**

enumerator **UART\_WORDSIZE\_9**

enumerator **UART\_WORDSIZE\_7**

enumerator **UART\_WORDSIZE\_6**

enumerator **UART\_WORDSIZE\_5**

enum **uart\_irq\_t**

UART irq enumeration

*Values:*

enumerator **UART\_IRQ\_NONE**

enumerator **UART\_IRQ\_RX**

enumerator **UART\_IRQ\_TX\_READY**

enumerator **UART\_IRQ\_MASK**

enum **uart\_event\_t**

UART event enumeration

*Values:*

enumerator **UART\_EVENT\_BIT**

enumerator **UART\_EVENT\_RX**

enumerator **UART\_EVENT\_TX\_READY**

enumerator **UART\_EVENT\_ERROR**

enumerator **UART\_EVENT\_MASK**

## Functions

void **uart\_reset**(*periph\_t* port)

Reset UART interface.

This function makes the given UART the reset default state.

**Parameters** **port** – [in] a peripheral enumerated in *periph\_t*

bool **uart\_has\_rx**(*periph\_t* port)

bool **uart\_is\_tx\_ready**(*periph\_t* port)

int **uart\_get\_rxd**(*periph\_t* port)

void **uart\_set\_txd**(*periph\_t* port, uint32\_t value)



void **uart\_enable\_irq**(*periph\_t* port, *uart\_irq\_t* irqs)  
Enable UART interrupts.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **irqs** – [in] a mix enum of *uart\_event\_t*

void **uart\_disable\_irq**(*periph\_t* port, *uart\_irq\_t* irqs)  
Disable UART interrupts.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **irqs** – [in] a mix enum of *uart\_event\_t*

void **uart\_start**(*periph\_t* port)

void **uart\_stop**(*periph\_t* port)

void **uart\_set\_baudrate**(*periph\_t* port, uint32\_t baudrate, uint32\_t pclk)  
Set UART baudrate.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **baudrate** – [in] baudrate
- **pclk** – [in] pclk

*uart\_event\_t* **uart\_get\_event**(*periph\_t* port)  
Read UART event flag.

**Parameters** **port** – [in] a peripheral enumerated in *periph\_t*

**Returns** event *uart\_event\_t*

void **uart\_clear\_event**(*periph\_t* port, *uart\_event\_t* events)  
Clear UART event flag.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **events** – [in] a mix enum of *uart\_event\_t*

void **uart\_set\_parity**(*periph\_t* port, *uart\_parity\_t* parity)  
Set UART parity.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **parity** – [in] a enum of *uart\_parity\_t*

void **uart\_set\_stopbits**(*periph\_t* port, *uart\_stopbit\_t* stopbit)  
Set UART stopbits.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **stopbit** – [in] a enum of *uart\_stopbit\_t*

void **uart\_set\_wordsize**(*periph\_t* port, *uart\_wordsize\_t* wordsize)  
Set UART data bit length.

**Parameters**

- **port** – [in] a peripheral enumerated in *periph\_t*
- **wordsize** – [in] a enum of *uart\_wordsize\_t*

### 3.1.9 Watchdog

#### Examples

```
#include "abov/hal/wdt.h"
#include "abov/irq.h"
#include "abov/delay.h"

#define OPTIONAL

int main(void)
{
    wdt_enable();
    #if defined(OPTIONAL)
        wdt_set_clock_source(CLK_LSI);
    #endif
    #if defined(ENABLE_WATCHDOG_INTERRUPT)
        wdt_set_interrupt(true);
        irq_enable(IRQ_WDT);
    #endif
    wdt_set_reload_ms(1000);
    wdt_start();

    for (int i = 0; i < 10; i++) {
        wdt_feed();
        udelay(500000);
    }

    while (1) {
        /* waiting for watchdog event */
    }

    return 0;
}

#if defined(ENABLE_WATCHDOG_INTERRUPT)
void ISR_WDT(void)
{
    wdt_feed();
}
#endif
```

## HAL

### Functions

void **wdt\_enable**(void)

void **wdt\_disable**(void)

uint32\_t **wdt\_get\_clock\_frequency**(void)

## LL

### Functions

void **wdt\_reset**(void)

Reset WDT unit.

This function makes WDT unit the reset state.

uint32\_t **wdt\_get\_count**(void)

Get the current watchdog count

void **wdt\_set\_prescaler**(uint32\_t div\_factor)

uint32\_t **wdt\_get\_prescaler**(void)

void **wdt\_set\_reload\_ms**(uint32\_t period\_ms)

void **wdt\_set\_reload**(uint32\_t timeout)

uint32\_t **wdt\_get\_reload**(void)

void **wdt\_feed**(void)

void **wdt\_start**(void)

void **wdt\_stop**(void)

void **wdt\_set\_debug\_stop\_mode**(bool enable)

Stop watchdog counter clock when core is halted.

**Parameters** **enable** – [in] counter clock stopped when true. Otherwise continues

void **wdt\_set\_interrupt**(bool enable)

Set watchdog interrupt.

The default behavior is to reset the system if not interrupt set

**Parameters** **enable** – [in] enables interrupt when true. disables when false

bool **wdt\_is\_event\_raised**(void)

void **wdt\_set\_clock\_source**(*clk\_source\_t* clk)

*clk\_source\_t* **wdt\_get\_clock\_source**(void)

## 3.2 System

### 3.2.1 Errata

### 3.2.2 IRQ

*IRQ\_FIXED*

#### Defines

**PERIPH\_TO\_IRQ**(periph)

**DEFINE\_IRQ**(n, name)

**RESERVE\_IRQ**(n)

**DEFINE\_IRQ**(n, name)

**RESERVE\_IRQ**(n)

#### Enums

enum **irq\_t**

*Values:*

enumerator **IRQ\_FIXED**

enumerator **IRQ\_MAX**

enumerator **IRQ\_UNDEFINED**

## Functions

void **irq\_enable**(*irq\_t* irq)

Enable an interrupt.

**Parameters** **irq** – [in] a enum of *irq\_t*

void **irq\_disable**(*irq\_t* irq)

Disable an interrupt.

**Parameters** **irq** – [in] a enum of *irq\_t*

void **irq\_clear\_pending**(*irq\_t* irq)

Clear pending bit on the given interrupt.

**Parameters** **irq** – [in] a enum of *irq\_t*

void **irq\_set\_priority**(*irq\_t* irq, int priority)

Set the priority for an interrupt.

**Parameters**

- **irq** – [in] a enum of *irq\_t*
- **priority** – [in] supports 0 to 192. The smaller number the higher priority

void **irq\_raise**(*irq\_t* irq)

Generate software interrupt.

**Parameters** **irq** – [in] a enum of *irq\_t*

void **irq\_default\_handler**(void)

void **ISR\_reset**(void)

void **ISR\_nmi**(void)

void **ISR\_hardfault**(void)

void **ISR\_memmanage**(void)

void **ISR\_busfault**(void)

void **ISR\_usagefault**(void)

void **ISR\_svc**(void)

void **ISR\_debugmonitor**(void)

void **ISR\_pendsv**(void)

void **ISR\_systick**(void)

### 3.2.3 Peripheral

#### Defines

**DEFINE\_IRQ**(n, name)

**RESERVE\_IRQ**(n)

#### Enums

enum **periph\_t**  
Peripherals enumeration  
*Values:*  
  
enumerator **PERIPH\_MAX**  
enumerator **PERIPH\_UNDEFINED**

### 3.2.4 System

#### Defines

**mb**()  
Insert a hardware full memory barrier.  
  
**rmb**()  
Insert a hardware read access memory barrier.  
  
**wmb**()  
Insert a hardware write access memory barrier.  
  
**interrupt\_unlock**()  
Enable IRQ Interrupts  
  
**interrupt\_lock**()  
Disable IRQ Interrupts

#### Functions

void **early\_init**(void)  
Early low-level initialization can be done depending on the core implementation such as co-processor configuration, initializing external memory, enabling caches and so on.  
  
void **pre\_main**(void)  
Device specific initialization can be done before jumping into the main application code.

## 3.3 Other

### 3.3.1 SysTick

*SYSTICK\_PRESCALER*

#### Examples

```
#include "abov/asm/arm/systick.h"
#include "abov/irq.h"
#include "abov/hal/gpio.h"

#define LED_PORT          PERIPH_GPIOD
#define LED_PIN           1

int main(void)
{
    gpio_open_output(LED_PORT, LED_PIN, GPIO_MODE_PUSH_PULL);

    systick_set_frequency(1);
    systick_clear();
    systick_start();

    while (1) {
        /* hang */
    }

    return 0;
}

void ISR_systick(void)
{
    static int led = 0;
    led ^= 1;
    gpio_write(LED_PORT, LED_PIN, led);
}
```

#### API

##### Functions

void **systick\_start**(void)

Start SysTick

void **systick\_stop**(void)

Stop SysTick

void **systick\_clear**(void)

Clear SysTick counter value

uint32\_t **systick\_set\_frequency**(uint32\_t hz)

Set SysTick clock frequency

uint32\_t **systick\_get\_frequency**(void)

Get SysTick clock frequency

uint32\_t **systick\_get\_counter**(void)

void **systick\_set\_counter**(uint32\_t value)



## EXAMPLES

<https://github.com/onkwon/libabov-example.git>

## 4.1 Blinky

### 4.1.1 Source

```
#include "abov/hal/gpio.h"
#include "abov/delay.h"

#define LED_PORT          PERIPH_GPIOD
#define LED_PIN           1

int main(void)
{
    #if 0
        gpio_open_output(LED_PORT, LED_PIN, GPIO_MODE_PUSHPULL);
    #else
        struct gpio_cfg cfg = { GPIO_MODE_PUSHPULL, };
        gpio_open(LED_PORT, LED_PIN, &cfg);
    #endif

    while (1) {
        gpio_write(LED_PORT, LED_PIN, 1);
        udelay(5000000);
        gpio_write(LED_PORT, LED_PIN, 0);
        udelay(5000000);
    }

    return 0;
}
```



## SUPPORTED DEVICES

### 5.1 ABOV

#### 5.1.1 next

- A31G
- A31L



## CHANGELOG

### 6.1 v0.0.1

@2021-04-06

#### 6.1.1 New Features

Initial version

#### 6.1.2 Bug Fixes

Initial version



## INDICES AND TABLES

- `genindex`
- `search`





## A

- `adc_calibrate` (C++ *function*), 13
- `adc_channel_t` (C++ *enum*), 10
- `adc_channel_t::ADC_CHANNEL_0` (C++ *enumerator*), 10
- `adc_channel_t::ADC_CHANNEL_1` (C++ *enumerator*), 10
- `adc_channel_t::ADC_CHANNEL_10` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_11` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_12` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_13` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_14` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_15` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_2` (C++ *enumerator*), 10
- `adc_channel_t::ADC_CHANNEL_3` (C++ *enumerator*), 10
- `adc_channel_t::ADC_CHANNEL_4` (C++ *enumerator*), 10
- `adc_channel_t::ADC_CHANNEL_5` (C++ *enumerator*), 10
- `adc_channel_t::ADC_CHANNEL_6` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_7` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_8` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_9` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_MASK` (C++ *enumerator*), 11
- `adc_channel_t::ADC_CHANNEL_MAX` (C++ *enumerator*), 11
- `adc_clear_event` (C++ *function*), 12
- `adc_disable` (C++ *function*), 10
- `adc_disable_clock` (C++ *function*), 12
- `adc_disable_irq` (C++ *function*), 12
- `adc_enable` (C++ *function*), 10
- `adc_enable_clock` (C++ *function*), 12
- `adc_enable_irq` (C++ *function*), 12
- `adc_event_t` (C++ *enum*), 11
- `adc_event_t::ADC_EVENT_BUSY` (C++ *enumerator*), 11
- `adc_event_t::ADC_EVENT_COMPLETE` (C++ *enumerator*), 12
- `adc_event_t::ADC_EVENT_MASK` (C++ *enumerator*), 12
- `adc_event_t::ADC_EVENT_NONE` (C++ *enumerator*), 11
- `adc_get_event` (C++ *function*), 12
- `adc_get_frequency` (C++ *function*), 12
- `adc_get_measurement` (C++ *function*), 12
- `adc_is_busy` (C++ *function*), 12
- `adc_is_completed` (C++ *function*), 13
- `adc_mode_t` (C++ *enum*), 10
- `adc_mode_t::ADC_MODE_CONTINUOUS_CONVERSION` (C++ *enumerator*), 10
- `adc_mode_t::ADC_MODE_CONTINUOUS_CONVERSION_MULTI_CHANNEL` (C++ *enumerator*), 10
- `adc_mode_t::ADC_MODE_IDLE` (C++ *enumerator*), 10
- `adc_mode_t::ADC_MODE_SINGLE_CONVERSION` (C++ *enumerator*), 10
- `adc_mode_t::ADC_MODE_SINGLE_CONVERSION_MULTI_CHANNEL` (C++ *enumerator*), 10
- `adc_reset` (C++ *function*), 12
- `adc_select_channel` (C++ *function*), 12
- `adc_set_clock_frequency` (C++ *function*), 12
- `adc_set_mode` (C++ *function*), 12
- `adc_set_sample_time` (C++ *function*), 13
- `adc_set_trigger` (C++ *function*), 12
- `adc_start` (C++ *function*), 12
- `adc_stop` (C++ *function*), 12
- `adc_trigger_t` (C++ *enum*), 11
- `adc_trigger_t::ADC_TRIGGER_EXTI11` (C++ *enumerator*), 11
- `adc_trigger_t::ADC_TRIGGER_MANUAL` (C++ *enumerator*), 11
- `adc_trigger_t::ADC_TRIGGER_MAX` (C++ *enumerator*), 11

tor), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER0\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER1\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER1\_CC1 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER1\_CC2 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER1\_CC3 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER2\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER2\_CC2 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER3\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER3\_CC1 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER3\_TRGO (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER4\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER5\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER6\_CC0 (C++  
 enumerator), 11  
 adc\_trigger\_t::ADC\_TRIGGER\_TIMER7\_CC0 (C++  
 enumerator), 11

## C

clk\_disable\_output (C++ function), 15  
 clk\_disable\_peripheral (C++ function), 14  
 clk\_disable\_source (C++ function), 14  
 clk\_enable\_output (C++ function), 15  
 clk\_enable\_peripheral (C++ function), 14  
 clk\_enable\_source (C++ function), 14  
 clk\_get\_frequency (C++ function), 15  
 clk\_get\_hclk\_frequency (C++ function), 15  
 clk\_get\_pclk\_frequency (C++ function), 15  
 clk\_get\_peripheral\_clock\_source (C++ function),  
 15  
 clk\_get\_peripheral\_clock\_source\_frequency  
 (C++ function), 15  
 clk\_get\_source (C++ function), 14  
 clk\_init (C++ function), 13  
 clk\_is\_pll\_locked (C++ function), 15  
 clk\_reset (C++ function), 14  
 clk\_set\_output\_prescaler (C++ function), 15  
 clk\_set\_output\_source (C++ function), 15  
 clk\_set\_peripheral\_clock\_source (C++ function),  
 15  
 clk\_set\_pll\_frequency (C++ function), 15  
 clk\_set\_source (C++ function), 14

clk\_source\_t (C++ enum), 14  
 clk\_source\_t::CLK\_HSE (C++ enumerator), 14  
 clk\_source\_t::CLK\_HSI (C++ enumerator), 14  
 clk\_source\_t::CLK\_LSE (C++ enumerator), 14  
 clk\_source\_t::CLK\_LSI (C++ enumerator), 14  
 clk\_source\_t::CLK\_PLL (C++ enumerator), 14  
 clk\_source\_t::CLK\_PLL\_BYPASS (C++ enumerator),  
 14  
 clk\_start\_pll (C++ function), 15  
 clk\_stop\_pll (C++ function), 15

## D

DEFINE\_IRQ (C macro), 48, 50

## E

early\_init (C++ function), 50

## G

gpio\_cfg (C++ struct), 16  
 gpio\_cfg::altfunc (C++ member), 17  
 gpio\_cfg::altfunc\_number (C++ member), 17  
 gpio\_cfg::debounce (C++ member), 17  
 gpio\_cfg::irq (C++ member), 17  
 gpio\_cfg::mode (C++ member), 17  
 gpio\_cfg::speed (C++ member), 17  
 gpio\_clear\_event (C++ function), 19  
 gpio\_close (C++ function), 16  
 gpio\_disable\_irq (C++ function), 19  
 gpio\_disable\_port (C++ function), 18  
 gpio\_enable\_irq (C++ function), 19  
 gpio\_enable\_port (C++ function), 18  
 gpio\_irq\_t (C++ enum), 17  
 gpio\_irq\_t::GPIO\_IRQ\_EDGE\_ANY (C++ enumera-  
 tor), 18  
 gpio\_irq\_t::GPIO\_IRQ\_EDGE\_FALLING (C++ enu-  
 mator), 18  
 gpio\_irq\_t::GPIO\_IRQ\_EDGE\_RISING (C++ enumera-  
 tor), 18  
 gpio\_irq\_t::GPIO\_IRQ\_LEVEL\_HIGH (C++ enumera-  
 tor), 18  
 gpio\_irq\_t::GPIO\_IRQ\_LEVEL\_LOW (C++ enumera-  
 tor), 18  
 gpio\_irq\_t::GPIO\_IRQ\_NONE (C++ enumerator), 18  
 gpio\_mode\_t (C++ enum), 17  
 gpio\_mode\_t::GPIO\_MODE\_ANALOG (C++ enumera-  
 tor), 17  
 gpio\_mode\_t::GPIO\_MODE\_INPUT (C++ enumerator),  
 17  
 gpio\_mode\_t::GPIO\_MODE\_INPUT\_PULLDOWN (C++  
 enumerator), 17  
 gpio\_mode\_t::GPIO\_MODE\_INPUT\_PULLUP (C++ enu-  
 mator), 17  
 gpio\_mode\_t::GPIO\_MODE\_OPENDRAIN (C++ enumer-  
 ator), 17

gpio\_mode\_t::GPIO\_MODE\_OPENDRAIN\_PULLDOWN (C++ enumerator), 17  
 gpio\_mode\_t::GPIO\_MODE\_OPENDRAIN\_PULLUP (C++ enumerator), 17  
 gpio\_mode\_t::GPIO\_MODE\_PUSHPULL (C++ enumerator), 17  
 gpio\_open (C++ function), 16  
 gpio\_open\_output (C++ function), 16  
 gpio\_read\_pin (C++ function), 19  
 gpio\_read\_port (C++ function), 20  
 gpio\_reset (C++ function), 18  
 gpio\_set\_altfunc (C++ function), 18  
 gpio\_set\_debouncer (C++ function), 19  
 gpio\_set\_mode (C++ function), 18  
 gpio\_set\_speed (C++ function), 19  
 gpio\_speed\_t (C++ enum), 18  
 gpio\_speed\_t::GPIO\_SPEED\_HIGH (C++ enumerator), 18  
 gpio\_speed\_t::GPIO\_SPEED\_LOW (C++ enumerator), 18  
 gpio\_speed\_t::GPIO\_SPEED\_MID (C++ enumerator), 18  
 gpio\_write\_pin (C++ function), 19  
 gpio\_write\_port (C++ function), 19  
 |  
 i2c\_clear\_event (C++ function), 26  
 i2c\_disable (C++ function), 25  
 i2c\_disable\_ack (C++ function), 25  
 i2c\_disable\_interrupt (C++ function), 25  
 i2c\_disable\_irq (C++ function), 25  
 i2c\_enable (C++ function), 25  
 i2c\_enable\_ack (C++ function), 25  
 i2c\_enable\_interrupt (C++ function), 25  
 i2c\_enable\_irq (C++ function), 25  
 i2c\_event\_t (C++ enum), 24  
 i2c\_event\_t::I2C\_EVENT\_BUSY (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_COLLISION (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_MASK (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_NONE (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_RX (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_SLAVE (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_STOP (C++ enumerator), 24  
 i2c\_event\_t::I2C\_EVENT\_TX (C++ enumerator), 24  
 i2c\_get\_event (C++ function), 26  
 i2c\_has\_address\_set (C++ function), 25  
 i2c\_has\_received (C++ function), 25  
 i2c\_has\_started (C++ function), 25  
 i2c\_has\_transfer\_completed (C++ function), 25  
 i2c\_is\_busy (C++ function), 25  
 i2c\_read\_byte (C++ function), 25  
 i2c\_reset (C++ function), 25  
 i2c\_send\_start (C++ function), 25  
 i2c\_send\_stop (C++ function), 25  
 i2c\_set\_frequency (C++ function), 25  
 i2c\_set\_slave\_address (C++ function), 25  
 i2c\_start (C++ function), 25  
 i2c\_stop (C++ function), 25  
 i2c\_write\_byte (C++ function), 25  
 interrupt\_lock (C macro), 50  
 interrupt\_unlock (C macro), 50  
 irq\_clear\_pending (C++ function), 49  
 irq\_default\_handler (C++ function), 49  
 irq\_disable (C++ function), 49  
 irq\_enable (C++ function), 49  
 irq\_raise (C++ function), 49  
 irq\_set\_priority (C++ function), 49  
 irq\_t (C++ enum), 48  
 irq\_t::IRQ\_FIXED (C++ enumerator), 48  
 irq\_t::IRQ\_MAX (C++ enumerator), 48  
 irq\_t::IRQ\_UNDEFINED (C++ enumerator), 48  
 ISR\_busfault (C++ function), 49  
 ISR\_debugmonitor (C++ function), 49  
 ISR\_hardfault (C++ function), 49  
 ISR\_memmanage (C++ function), 49  
 ISR\_nmi (C++ function), 49  
 ISR\_pendsv (C++ function), 49  
 ISR\_reset (C++ function), 49  
 ISR\_svc (C++ function), 49  
 ISR\_systick (C++ function), 49  
 ISR\_usagefault (C++ function), 49  
 M  
 mb (C macro), 50  
 P  
 periph\_t (C++ enum), 50  
 periph\_t::PERIPH\_MAX (C++ enumerator), 50  
 periph\_t::PERIPH\_UNDEFINED (C++ enumerator), 50  
 PERIPH\_TO\_IRQ (C macro), 48  
 pre\_main (C++ function), 50  
 pwr\_clear\_reboot\_source (C++ function), 27  
 pwr\_clear\_wakeup\_source (C++ function), 27  
 pwr\_disable\_peripheral (C++ function), 27  
 pwr\_enable\_peripheral (C++ function), 27  
 pwr\_get\_reboot\_source (C++ function), 26  
 pwr\_get\_wakeup\_source (C++ function), 27  
 pwr\_mode\_t (C++ enum), 26  
 pwr\_mode\_t::PWR\_MODE\_BLACKOUT (C++ enumerator), 26  
 pwr\_mode\_t::PWR\_MODE\_DEEP\_SLEEP (C++ enumerator), 26

pwr\_mode\_t::PWR\_MODE\_RUN (C++ *enumerator*), 26  
 pwr\_mode\_t::PWR\_MODE\_SLEEP (C++ *enumerator*), 26  
 pwr\_reboot (C++ *function*), 26  
 pwr\_reset (C++ *function*), 26  
 pwr\_set\_mode (C++ *function*), 27  
 pwr\_set\_wakeup\_source (C++ *function*), 27

## R

RESERVE\_IRQ (C *macro*), 48, 50  
 rmb (C *macro*), 50

## S

spi\_cfg (C++ *struct*), 29  
 spi\_cfg::auto\_chip\_select (C++ *member*), 29  
 spi\_cfg::cpha (C++ *member*), 29  
 spi\_cfg::cpol (C++ *member*), 29  
 spi\_cfg::data\_width (C++ *member*), 29  
 spi\_cfg::frequency (C++ *member*), 29  
 spi\_cfg::interrupt (C++ *member*), 29  
 spi\_cfg::lsb\_first (C++ *member*), 29  
 spi\_cfg::mode (C++ *member*), 29  
 spi\_clear\_event (C++ *function*), 30  
 spi\_clear\_rx\_buffer (C++ *function*), 30  
 spi\_clear\_tx\_buffer (C++ *function*), 30  
 spi\_deinit (C++ *function*), 28  
 spi\_disable (C++ *function*), 28  
 spi\_disable\_chip\_select (C++ *function*), 31  
 spi\_disable\_clock (C++ *function*), 30  
 spi\_disable\_crc (C++ *function*), 32  
 spi\_disable\_irq (C++ *function*), 31  
 spi\_enable (C++ *function*), 28  
 spi\_enable\_chip\_select (C++ *function*), 31  
 spi\_enable\_clock (C++ *function*), 30  
 spi\_enable\_crc (C++ *function*), 31  
 spi\_enable\_irq (C++ *function*), 31  
 spi\_event\_t (C++ *enum*), 29  
 spi\_event\_t::SPI\_EVENT\_BUSY (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_CHIP\_DESELECTED (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_CHIP\_SELECTED (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_CRC\_ERROR (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_MASK (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_MODE\_FAULT (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_NONE (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_OVERRUN (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_RX (C++ *enumerator*), 30

spi\_event\_t::SPI\_EVENT\_TX\_COMPLETE (C++ *enumerator*), 30  
 spi\_event\_t::SPI\_EVENT\_UNDERRUN (C++ *enumerator*), 30  
 spi\_get\_event (C++ *function*), 30  
 spi\_get\_rxd (C++ *function*), 30  
 spi\_has\_rx (C++ *function*), 30  
 spi\_init (C++ *function*), 28  
 spi\_irq\_t (C++ *enum*), 29  
 spi\_irq\_t::SPI\_IRQ\_EDGE\_CHAGNE (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_ERROR (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_FRAME\_ERROR (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_MASK (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_NONE (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_OVERRUN (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_RX (C++ *enumerator*), 29  
 spi\_irq\_t::SPI\_IRQ\_TX (C++ *enumerator*), 29  
 spi\_is\_busy (C++ *function*), 30  
 spi\_is\_tx\_completed (C++ *function*), 30  
 spi\_mode\_t (C++ *enum*), 29  
 spi\_mode\_t::SPI\_MODE\_MASTER (C++ *enumerator*), 29  
 spi\_mode\_t::SPI\_MODE\_SLAVE (C++ *enumerator*), 29  
 spi\_read (C++ *function*), 29  
 spi\_reset (C++ *function*), 30  
 spi\_set\_bitorder (C++ *function*), 31  
 spi\_set\_burst\_delay (C++ *function*), 31  
 spi\_set\_chip\_select\_level (C++ *function*), 31  
 spi\_set\_chip\_select\_mode (C++ *function*), 31  
 spi\_set\_chip\_select\_polarity (C++ *function*), 31  
 spi\_set\_clock\_phase (C++ *function*), 31  
 spi\_set\_clock\_polarity (C++ *function*), 31  
 spi\_set\_data\_width (C++ *function*), 31  
 spi\_set\_frequency (C++ *function*), 31  
 spi\_set\_loopback (C++ *function*), 31  
 spi\_set\_mode (C++ *function*), 31  
 spi\_set\_start\_delay (C++ *function*), 31  
 spi\_set\_stop\_delay (C++ *function*), 31  
 spi\_set\_txd (C++ *function*), 30  
 spi\_start (C++ *function*), 28  
 spi\_stop (C++ *function*), 28  
 spi\_write (C++ *function*), 28  
 spi\_write\_read (C++ *function*), 29  
 systick\_clear (C++ *function*), 51  
 systick\_get\_counter (C++ *function*), 52  
 systick\_get\_frequency (C++ *function*), 51  
 systick\_set\_counter (C++ *function*), 52  
 systick\_set\_frequency (C++ *function*), 51  
 systick\_start (C++ *function*), 51  
 systick\_stop (C++ *function*), 51

## T

timer\_cc\_mode\_t (C++ enum), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_ACTIVE\_HIGH (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_ACTIVE\_LOW (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_HIGH (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_LOW (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_NONE (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_PWM\_ACTIVE\_HIGH (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_PWM\_ACTIVE\_LOW (C++ enumerator), 35  
 timer\_cc\_mode\_t::TIMER\_CC\_MODE\_TOGGLE (C++ enumerator), 35  
 timer\_cc\_t (C++ enum), 35  
 timer\_cc\_t::TIMER\_CC\_0 (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_1 (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_1N (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_2 (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_2N (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_3 (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_3N (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_4 (C++ enumerator), 35  
 timer\_cc\_t::TIMER\_CC\_4N (C++ enumerator), 35  
 timer\_cfg (C++ struct), 34  
 timer\_cfg::frequency (C++ member), 34  
 timer\_cfg::irq (C++ member), 34  
 timer\_cfg::irq\_priority (C++ member), 34  
 timer\_cfg::mode (C++ member), 34  
 timer\_cfg::set\_clock\_source (C++ member), 34  
 timer\_clear\_event (C++ function), 37  
 timer\_deinit (C++ function), 34  
 timer\_direction\_t (C++ enum), 36  
 timer\_direction\_t::TIMER\_DIRECTION\_DOWN (C++ enumerator), 36  
 timer\_direction\_t::TIMER\_DIRECTION\_UP (C++ enumerator), 36  
 timer\_disable\_cc\_fastmode (C++ function), 39  
 timer\_disable\_cc\_pin (C++ function), 38  
 timer\_disable\_cc\_preload (C++ function), 39  
 timer\_disable\_irq (C++ function), 36  
 timer\_enable\_cc\_fastmode (C++ function), 39  
 timer\_enable\_cc\_pin (C++ function), 38  
 timer\_enable\_cc\_preload (C++ function), 39  
 timer\_enable\_irq (C++ function), 36  
 timer\_event\_t (C++ enum), 35  
 timer\_event\_t::TIMER\_EVENT\_CC\_0 (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_CC\_1 (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_CC\_2 (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_CC\_3 (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_CC\_4 (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_NONE (C++ enumerator), 35  
 timer\_event\_t::TIMER\_EVENT\_OVERFLOW (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_UNDERFLOW (C++ enumerator), 36  
 timer\_event\_t::TIMER\_EVENT\_UPDATE (C++ enumerator), 36  
 timer\_get\_cc (C++ function), 38  
 timer\_get\_counter (C++ function), 37  
 timer\_get\_event (C++ function), 37  
 timer\_get\_frequency (C++ function), 37  
 timer\_get\_prescaler (C++ function), 37  
 timer\_get\_reload (C++ function), 38  
 timer\_init (C++ function), 34  
 timer\_mode\_t (C++ enum), 35  
 timer\_mode\_t::TIMER\_MODE\_CAPTURE (C++ enumerator), 35  
 timer\_mode\_t::TIMER\_MODE\_NORMAL (C++ enumerator), 35  
 timer\_mode\_t::TIMER\_MODE\_ONESHOT (C++ enumerator), 35  
 timer\_mode\_t::TIMER\_MODE\_PWM (C++ enumerator), 35  
 timer\_reset (C++ function), 36  
 timer\_set\_cc (C++ function), 38  
 timer\_set\_cc\_filter (C++ function), 39  
 timer\_set\_cc\_pin (C++ function), 38  
 timer\_set\_cc\_pin\_mode (C++ function), 38  
 timer\_set\_cc\_pin\_polarity (C++ function), 39  
 timer\_set\_cc\_prescaler (C++ function), 39  
 timer\_set\_clock\_divider (C++ function), 36  
 timer\_set\_counter (C++ function), 37  
 timer\_set\_counter\_alignment\_mode (C++ function), 39  
 timer\_set\_counter\_direction (C++ function), 39  
 timer\_set\_mode (C++ function), 36  
 timer\_set\_prescaler (C++ function), 37  
 timer\_set\_reload (C++ function), 37  
 timer\_set\_slave\_mode (C++ function), 39  
 timer\_start (C++ function), 37  
 timer\_stop (C++ function), 37

## U

uart\_cfg (C++ struct), 43  
 uart\_cfg::baudrate (C++ member), 43  
 uart\_cfg::parity (C++ member), 43  
 uart\_cfg::rx\_interrupt (C++ member), 43



uart\_cfg::stopbit (C++ member), 43  
uart\_cfg::tx\_interrupt (C++ member), 43  
uart\_cfg::wordsize (C++ member), 43  
uart\_clear\_event (C++ function), 45  
uart\_default\_isr (C++ function), 42  
uart\_deinit (C++ function), 41  
uart\_disable\_irq (C++ function), 45  
uart\_enable\_irq (C++ function), 44  
uart\_event\_t (C++ enum), 44  
uart\_event\_t::UART\_EVENT\_BIT (C++ enumerator), 44  
uart\_event\_t::UART\_EVENT\_ERROR (C++ enumerator), 44  
uart\_event\_t::UART\_EVENT\_MASK (C++ enumerator), 44  
uart\_event\_t::UART\_EVENT\_RX (C++ enumerator), 44  
uart\_event\_t::UART\_EVENT\_TX\_READY (C++ enumerator), 44  
uart\_get\_event (C++ function), 45  
uart\_get\_rxd (C++ function), 44  
uart\_handle\_t (C++ union), 43  
uart\_handle\_t::\_align (C++ member), 43  
uart\_handle\_t::\_size (C++ member), 43  
uart\_has\_rx (C++ function), 44  
uart\_init (C++ function), 41  
uart\_irq\_callback\_t (C++ type), 41  
uart\_irq\_t (C++ enum), 44  
uart\_irq\_t::UART\_IRQ\_MASK (C++ enumerator), 44  
uart\_irq\_t::UART\_IRQ\_NONE (C++ enumerator), 44  
uart\_irq\_t::UART\_IRQ\_RX (C++ enumerator), 44  
uart\_irq\_t::UART\_IRQ\_TX\_READY (C++ enumerator), 44  
uart\_is\_tx\_ready (C++ function), 44  
uart\_parity\_t (C++ enum), 43  
uart\_parity\_t::UART\_PARITY\_EVEN (C++ enumerator), 43  
uart\_parity\_t::UART\_PARITY\_NONE (C++ enumerator), 43  
uart\_parity\_t::UART\_PARITY\_ODD (C++ enumerator), 43  
uart\_read (C++ function), 41  
uart\_read\_byte (C++ function), 42  
uart\_read\_byte\_nonblock (C++ function), 42  
uart\_register\_error\_handler (C++ function), 42  
uart\_register\_rx\_handler (C++ function), 42  
uart\_register\_tx\_handler (C++ function), 42  
uart\_reset (C++ function), 44  
uart\_set\_baudrate (C++ function), 45  
uart\_set\_parity (C++ function), 45  
uart\_set\_stopbits (C++ function), 45  
uart\_set\_txd (C++ function), 44  
uart\_set\_wordsize (C++ function), 45  
uart\_start (C++ function), 45

uart\_stop (C++ function), 45  
uart\_stopbit\_t (C++ enum), 43  
uart\_stopbit\_t::UART\_STOPBIT\_1 (C++ enumerator), 43  
uart\_stopbit\_t::UART\_STOPBIT\_1\_5 (C++ enumerator), 43  
uart\_stopbit\_t::UART\_STOPBIT\_2 (C++ enumerator), 43  
uart\_wordsize\_t (C++ enum), 43  
uart\_wordsize\_t::UART\_WORDSIZE\_5 (C++ enumerator), 44  
uart\_wordsize\_t::UART\_WORDSIZE\_6 (C++ enumerator), 44  
uart\_wordsize\_t::UART\_WORDSIZE\_7 (C++ enumerator), 44  
uart\_wordsize\_t::UART\_WORDSIZE\_8 (C++ enumerator), 44  
uart\_wordsize\_t::UART\_WORDSIZE\_9 (C++ enumerator), 44  
uart\_write (C++ function), 41  
uart\_write\_byte (C++ function), 42

## W

wdt\_disable (C++ function), 47  
wdt\_enable (C++ function), 47  
wdt\_feed (C++ function), 47  
wdt\_get\_clock\_frequency (C++ function), 47  
wdt\_get\_clock\_source (C++ function), 48  
wdt\_get\_count (C++ function), 47  
wdt\_get\_prescaler (C++ function), 47  
wdt\_get\_reload (C++ function), 47  
wdt\_is\_event\_raised (C++ function), 47  
wdt\_reset (C++ function), 47  
wdt\_set\_clock\_source (C++ function), 48  
wdt\_set\_debug\_stop\_mode (C++ function), 47  
wdt\_set\_interrupt (C++ function), 47  
wdt\_set\_prescaler (C++ function), 47  
wdt\_set\_reload (C++ function), 47  
wdt\_set\_reload\_ms (C++ function), 47  
wdt\_start (C++ function), 47  
wdt\_stop (C++ function), 47  
wmb (C macro), 50