halmcu

Release 0.0.1

Kyunghwan Kwon

Oct 08, 2021

CONTENTS

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

CHAPTER

ONE

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 `gitcl`, 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[1]	F4	F7
ADC					
CAN					
CLK					
CRC					
DAC					
DMA					
ETH					
FLASH					
FSMC					
GPIO					
I2C					
PWR					
RTC					
SDIO					
SPI					
Timer					
UART					
USB					
WDT					

[1] Only high-density devices are supported at the moment.

CHAPTER

TWO

ARCHITECTURE & DESIGN

2.1 API

2.2 Directory Structure

```
.
├── arch/
│   └── arm/
│       ├── cortex-m/
│       │   ├── m0plus/
│       │   ├── m3/
│       │   └── m4/
│       └── include/
│           └── CMSIS/
├── docs/
├── devices/
│   ├── common/
│   ├── abov/
│       ├── a31g/
│           ...
│       └── a34m/
│   └── st/
│       └── stm32f1/
├── drivers/
├── hal/
│   ├── adc/
│       ...
│   └── uart/
├── examples/
├── include
│   └── halmcu
│       ├── asm/
│       ├── hal/
│       ├── ll/
│       └── peri/
├── projects/
```

(continues on next page)

(continued from previous 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 shoud 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`

CHAPTER

THREE

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);
#if defined(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_TRGO**

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

```c
#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(500000);
                gpio_write(LED_PORT, LED_PIN, 0);
                udelay(500000);
        }

        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_PUSHPULL**
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);
#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_PUSHPULL);
        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_PUSHPULL,
                .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, 1000000, 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_PUSHPULL,
                .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, 16000000);
        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_PUSHPULL,
                .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,
#if !defined(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 sizie

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 bariier.

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_PUSHPULL);

        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)

CHAPTER

FOUR

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(500000);
                gpio_write(LED_PORT, LED_PIN, 0);
                udelay(500000);
        }

        return 0;
}
```

CHAPTER

FIVE

SUPPORTED DEVICES

5.1 ABOV

5.1.1 next

- A31G
- A31L

CHAPTER

SIX

CHANGELOG

6.1 v0.0.1

@2021-04-06

6.1.1 New Features

Initial version

6.1.2 Bug Fixes

Initial version

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- search

INDEX

A

C

D

E

G

I

M

P

R

S

T

U

W