



An OS to build, deploy and securely manage billions of devices

Latest News:

Apache Mynewt 1.10.0, Apache NimBLE 1.5.0 (/download) released (May 6, 2022)

Docs (/documentation/) / OS User Guide (../../../../os_user_guide.html) / Hardware Abstraction Layer (../hal.html) / GPIO

[Edit on GitHub \(https://github.com/apache/mynewt-core/edit/master/docs/os/modules/hal/hal_gpio/hal_gpio.rst\)](https://github.com/apache/mynewt-core/edit/master/docs/os/modules/hal/hal_gpio/hal_gpio.rst)

Search documentation

Version: latest



Introduction (../../../../index.html)

Setup & Get Started (../../../../get_started/index.html)

Concepts (../../../../concepts.html)

Tutorials (../../../../tutorials/tutorials.html)

Third-party Resources (../../../../external_links.html)

OS User Guide (../../../../os_user_guide.html)

Kernel (../../../../core_os/mynewt_os.html)

System (../../../../system_modules.html)

Hardware Abstraction (../hal.html)

Timer (../hal_timer/hal_timer.html)

GPIO



[UART \(../hal_uart/hal_uart.html\)](#)

[SPI \(../hal_spi/hal_spi.html\)](#)

[I2C \(../hal_i2c/hal_i2c.html\)](#)

[Flash \(../hal_flash/hal_flash.html\)](#)

[System \(../hal_system/hal_sys.html\)](#)

[Watchdog \(../hal_watchdog/hal_watchdog.html\)](#)

[BSP \(../hal_bsp/hal_bsp.html\)](#)

[OS Tick \(../hal_os_tick/hal_os_tick.html\)](#)

[Creating New HAL Interfaces \(../hal_creation.html\)](#)

[Using HAL in Your Libraries \(../hal_in_libraries.html\)](#)

[Secure Bootloader \(../../bootloader/bootloader.html\)](#)

[Split Images \(../../split/split.html\)](#)

[Porting Guide \(../../core_os/porting/port_os.html\)](#)

[Baselibc \(../../baselibc.html\)](#)

[Drivers \(../../drivers/driver.html\)](#)

[Device Management with Newt Manager \(../../devmgt/newtmgr.html\)](#)

[Device Management with MCUmgr \(../../mcumgr/mcumgr.html\)](#)

[Image Manager \(../../imgmgr/imgmgr.html\)](#)

[Compile-Time Configuration \(../../sysinitconfig/sysinitconfig.html\)](#)

[System Initialization and Shutdown \(../../sysinitdown/sysinitdown.html\)](#)

[Build-Time Hooks \(../../extcmd/extcmd.html\)](#)

[File System \(../../fs/fs.html\)](#)

[Flash Circular Buffer \(../../fcb/fcb.html\)](#)

[Sensor Framework \(../../sensor_framework/sensor_framework.html\)](#)

[Test Utilities \(../../testutil/testutil.html\)](#)

[JSON \(../../json/json.html\)](#)

[Manufacturing support \(../../mfg/mfg.html\)](#)

[Board support \(../../bsp/index.html\)](#)

[BLE User Guide \(../../network/index.html\)](#)

[Newt Tool Guide \(../../newt/index.html\)](#)

Newt Manager Guide ([../.../newtmgr/index.html](#))

Mynewt FAQ ([../.../mynewt_faq/index.html](#))

Appendix ([../.../misc/index.html](#))

GPIO

This is the hardware independent GPIO (General Purpose Input Output) Interface for Mynewt.

Description

Contains the basic operations to set and read General Purpose Digital I/O Pins within a Mynewt system.

Individual GPIOs are referenced in the APIs as `pins`. However, in this interface the `pins` are virtual GPIO pins. The MCU header file maps these virtual `pins` to the physical GPIO ports and pins.

Typically, the BSP code may define named I/O pins in terms of these virtual `pins` to describe the devices attached to the physical pins.

Here's a brief example so you can get the gist of the translation.

Suppose my product uses the stm32F4xx processor. There already exists support for this processor within Mynewt. The processor has N ports (A,B,C..) of 16 GPIO pins per port. The MCU `hal_gpio` driver maps these to a set of virtual pins 0-N where port A maps to 0-15, Port B maps to 16-31, Port C maps to 32-47 and so on. The exact number of physical port (and virtual port pins) depends on the specific variant of the stm32F4xx.

So if I want to turn on port B pin 3, that would be virtual pin $1 \times 16 + 3 = 19$. This translation is defined in the MCU implementation of `hal_gpio.c` (https://github.com/apache/mynewt-core/blob/master/hw/mcu/stm/stm32_common/src/hal_gpio.c) for the stm32. Each MCU will typically have a different translation method depending on its GPIO architecture.

Now, when writing a BSP, it's common to give names to the relevant port pins that you are using. Thus, the BSP may define a mapping between a function and a virtual port pin in the `bsp.h` header file for the BSP. For example,

```
#define SYSTEM_LED           (37)
#define FLASH_SPI_CHIP_SELECT (3)
```

would map the system indicator LED to virtual pin 37 which on the stm32F4xx would be Port C pin 5 and the chip select line for the external SPI flash to virtual pin 3 which on the stm32F4xx is port A pin 3.

Said another way, in this specific system we get

```
SYSTEM_LED --> hal_gpio virtual pin 37 --> port C pin 5 on the stm34F4xx
```

API

enum hal_gpio_mode_e

The “mode” of the gpio.

The gpio is either an input, output, or it is “not connected” (the pin specified is not functioning as a gpio)

Values:

enumerator HAL_GPIO_MODE_NC

Not connected.

enumerator HAL_GPIO_MODE_IN

Input.

enumerator HAL_GPIO_MODE_OUT

Output.

enum hal_gpio_pull

Values:

enumerator HAL_GPIO_PULL_NONE

Pull-up/down not enabled.

enumerator HAL_GPIO_PULL_UP

Pull-up enabled.

enumerator HAL_GPIO_PULL_DOWN

Pull-down enabled.

enum hal_gpio_irq_trigger

Values:

enumerator HAL_GPIO_TRIG_NONE

enumerator HAL_GPIO_TRIG_RISING

IRQ occurs on rising edge.

enumerator HAL_GPIO_TRIG_FALLING

IRQ occurs on falling edge.

enumerator HAL_GPIO_TRIG_BOTH

IRQ occurs on either edge.

enumerator HAL_GPIO_TRIG_LOW

IRQ occurs when line is low.

enumerator HAL_GPIO_TRIG_HIGH

IRQ occurs when line is high.

typedef enum hal_gpio_mode_e hal_gpio_mode_t

typedef enum hal_gpio_pull hal_gpio_pull_t

typedef enum hal_gpio_irq_trigger hal_gpio_irq_trig_t

typedef void (*hal_gpio_irq_handler_t)(void *arg)

int hal_gpio_init_in(int pin, hal_gpio_pull_t pull)

Initializes the specified pin as an input.

Return

int 0: no error; -1 otherwise.

Parameters

- `pin` : Pin number to set as input
- `pull` : pull type

int hal_gpio_init_out(int pin, int val)

Initialize the specified pin as an output, setting the pin to the specified value.

Return

int 0: no error; -1 otherwise.

Parameters

- `pin` : Pin number to set as output
- `val` : Value to set pin

int hal_gpio_deinit(int pin)

Deinitialize the specified pin to revert the previous initialization.

Return

int 0: no error; -1 otherwise.

Parameters

- `pin` : Pin number to unset

void hal_gpio_write(int pin, int val)

Write a value (either high or low) to the specified pin.

Parameters

- `pin` : Pin to set
- `val` : Value to set pin (0:low 1:high)

int hal_gpio_read(int pin)

Reads the specified pin.

Return

int 0: low, 1: high

Parameters

- `pin`: Pin number to read

int hal_gpio_toggle(int pin)

Toggles the specified pin.

Return

current gpio state int 0: low, 1: high

Parameters

- `pin`: Pin number to toggle

int hal_gpio_irq_init(int pin, hal_gpio_irq_handler_t handler, void *arg, hal_gpio_irq_trig_t trig, hal_gpio_pull_t pull)

Initialize a given pin to trigger a GPIO IRQ callback.

Return

0 on success, non-zero error code on failure.

Parameters

- `pin`: The pin to trigger GPIO interrupt on
- `handler`: The handler function to call
- `arg`: The argument to provide to the IRQ handler
- `trig`: The trigger mode (e.g. rising, falling)
- `pull`: The mode of the pin (e.g. pullup, pulldown)

void hal_gpio_irq_release(int pin)

Release a pin from being configured to trigger IRQ on state change.

Parameters

- `pin`: The pin to release

`void hal_gpio_irq_enable(int pin)`

Enable IRQs on the passed pin.

Parameters

- `pin`: The pin to enable IRQs on

`void hal_gpio_irq_disable(int pin)`

Disable IRQs on the passed pin.

Parameters

- `pin`: The pin to disable IRQs on

⬅ Previous: Timer (../hal_timer/hal_timer.html)

Next: UART ➡ (../hal_uart/hal_uart.html)

Apache Mynewt is available under Apache License, version 2.0.



Apache Mynewt, Mynewt, Apache, the Apache feather logo, and the Apache Mynewt project logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and other countries.

