

GPIO & RTC GPIO

[\[中文\]](#)

GPIO Summary

The ESP32 chip features 34 physical GPIO pins (GPIO0 ~ GPIO19, GPIO21 ~ GPIO23, GPIO25 ~ GPIO27, and GPIO32 ~ GPIO39). Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. Through IO MUX, RTC IO MUX and the GPIO matrix, peripheral input signals can be from any IO pins, and peripheral output signals can be routed to any IO pins. Together these modules provide highly configurable I/O. For more details, see *ESP32 Technical Reference Manual > IO MUX and GPIO Matrix (GPIO, IO_MUX)* [\[PDF\]](#).

The table below provides more information on pin usage, and please note the comments in the table for GPIOs with restrictions.

GPIO	Analog Function	RTC GPIO	Comments
GPIO0	ADC2_CH1	RTC_GPIO11	Strapping pin
GPIO1			TXD
GPIO2	ADC2_CH2	RTC_GPIO12	Strapping pin
GPIO3			RXD
GPIO4	ADC2_CH0	RTC_GPIO10	
GPIO5			Strapping pin
GPIO6			SPI0/1
GPIO7			SPI0/1
GPIO8			SPI0/1
GPIO9			SPI0/1
GPIO10			SPI0/1
GPIO11			SPI0/1
GPIO12	ADC2_CH5	RTC_GPIO15	Strapping pin; JTAG
GPIO13	ADC2_CH4	RTC_GPIO14	JTAG

GPIO	Analog Function	RTC GPIO	Comments
GPIO14	ADC2_CH6	RTC_GPIO16	JTAG
GPIO15	ADC2_CH3	RTC_GPIO13	Strapping pin; JTAG
GPIO16			SPI0/1
GPIO17			SPI0/1
GPIO18			
GPIO19			
GPIO21			
GPIO22			
GPIO23			
GPIO25	ADC2_CH8	RTC_GPIO6	
GPIO26	ADC2_CH9	RTC_GPIO7	
GPIO27	ADC2_CH7	RTC_GPIO17	
GPIO32	ADC1_CH4	RTC_GPIO9	
GPIO33	ADC1_CH5	RTC_GPIO8	
GPIO34	ADC1_CH6	RTC_GPIO4	GPI
GPIO35	ADC1_CH7	RTC_GPIO5	GPI
GPIO36	ADC1_CH0	RTC_GPIO0	GPI
GPIO37	ADC1_CH1	RTC_GPIO1	GPI
GPIO38	ADC1_CH2	RTC_GPIO2	GPI
GPIO39	ADC1_CH3	RTC_GPIO3	GPI

Note

- Strapping pin: GPIO0, GPIO2, GPIO5, GPIO12 (MTDI), and GPIO15 (MTDO) are strapping pins. For more information, please refer to [ESP32 datasheet](#).
- SPI0/1: GPIO6-11 and GPIO16-17 are usually connected to the SPI flash and PSRAM integrated on the module and therefore should not be used for other purposes.
- JTAG: GPIO12-15 are usually used for inline debug.
- GPI: GPIO34-39 can only be set as input mode and do not have software-enabled pullup or pulldown functions.
- TXD & RXD are usually used for flashing and debugging.

- ADC2: ADC2 pins cannot be used when Wi-Fi is used. So, if you are having trouble getting the value from an ADC2 GPIO while using Wi-Fi, you may consider using an ADC1 GPIO instead, which should solve your problem. For more details, please refer to [Hardware Limitations of ADC Continuous Mode](#) and [Hardware Limitations of ADC Oneshot Mode](#).
- Please do not use the interrupt of GPIO36 and GPIO39 when using ADC or Wi-Fi and Bluetooth with sleep mode enabled. Please refer to [ESP32 ECO and Workarounds for Bugs](#) > Section 3.11 for the detailed description of the issue.

There is also separate “RTC GPIO” support, which functions when GPIOs are routed to the “RTC” low-power and analog subsystem. These pin functions can be used when:

- In Deep-sleep mode
- The [Ultra Low Power co-processor](#) is running
- Analog functions such as ADC/DAC/etc are in use.

Application Example

- GPIO output and input interrupt example: [peripherals/gpio/generic_gpio](#).

API Reference - Normal GPIO

Header File

- [components/driver/gpio/include/driver/gpio.h](#)

Functions

`esp_err_t gpio_config(const gpio_config_t *pGPIOConfig)`

GPIO common configuration.

Configure GPIO's [Mode](#), [pull-up](#), [PullDown](#), [IntrType](#)

Parameters: `pGPIOConfig` – Pointer to GPIO configure struct

Returns:

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

`esp_err_t gpio_reset_pin(gpio_num_t gpio_num)`

Reset an gpio to default state (select gpio function, enable pullup and disable input and output).

! Note

This function also configures the IOMUX for this pin to the GPIO function, and disconnects any other peripheral output configured via GPIO Matrix.

Parameters: **gpio_num** – GPIO number.

Returns: Always return ESP_OK.

esp_err_t gpio_set_intr_type(gpio_num_t gpio_num, gpio_int_type_t intr_type)

GPIO set interrupt trigger type.

Parameters:

- **gpio_num** – GPIO number. If you want to set the trigger type of e.g. of GPIO16, gpio_num should be GPIO_NUM_16 (16);
- **intr_type** – Interrupt type, select from gpio_int_type_t

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_intr_enable(gpio_num_t gpio_num)

Enable GPIO module interrupt signal.

! Note

ESP32: Please do not use the interrupt of GPIO36 and GPIO39 when using ADC or Wi-Fi and Bluetooth with sleep mode enabled. Please refer to the comments of `adc1_get_raw`. Please refer to Section 3.11 of [ESP32 ECO and Workarounds for Bugs](#) for the description of this issue.

Parameters: **gpio_num** – GPIO number. If you want to enable an interrupt on e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_intr_disable(gpio_num_t gpio_num)

Disable GPIO module interrupt signal.

⚠ Note

This function is allowed to be executed when Cache is disabled within ISR context, by enabling `CONFIG_GPIO_CTRL_FUNC_IN_IRAM`

Parameters: `gpio_num` – GPIO number. If you want to disable the interrupt of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);

Returns:

- ESP_OK success
- ESP_ERR_INVALID_ARG Parameter error

`esp_err_t gpio_set_level(gpio_num_t gpio_num, uint32_t level)`

GPIO set output level.

⚠ Note

This function is allowed to be executed when Cache is disabled within ISR context, by enabling `CONFIG_GPIO_CTRL_FUNC_IN_IRAM`

Parameters:

- `gpio_num` – GPIO number. If you want to set the output level of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);
- `level` – Output level. 0: low ; 1: high

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO number error

`int gpio_get_level(gpio_num_t gpio_num)`

GPIO get input level.

⚠ Warning

If the pad is not configured for input (or input and output) the returned value is always 0.

Parameters: `gpio_num` – GPIO number. If you want to get the logic level of e.g. pin GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);

Returns:

- 0 the GPIO input level is 0
- 1 the GPIO input level is 1

`esp_err_t gpio_set_direction(gpio_num_t gpio_num, gpio_mode_t mode)`

GPIO set direction.

Configure GPIO direction, such as output_only, input_only, output_and_input

- Parameters:**
- **gpio_num** – Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
 - **mode** – GPIO direction

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO error

`esp_err_t gpio_set_pull_mode(gpio_num_t gpio_num, gpio_pull_mode_t pull)`

Configure GPIO pull-up/pull-down resistors.

Note

ESP32: Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

- Parameters:**
- **gpio_num** – GPIO number. If you want to set pull up or down mode for e.g. GPIO16, gpio_num should be GPIO_NUM_16 (16);
 - **pull** – GPIO pull up/down mode.

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG : Parameter error

`esp_err_t gpio_wakeup_enable(gpio_num_t gpio_num, gpio_int_type_t intr_type)`

Enable GPIO wake-up function.

- Parameters:**
- **gpio_num** – GPIO number.
 - **intr_type** – GPIO wake-up type. Only GPIO_INTR_LOW_LEVEL or GPIO_INTR_HIGH_LEVEL can be used.

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG Parameter error

`esp_err_t gpio_wakeup_disable(gpio_num_t gpio_num)`

Disable GPIO wake-up function.

Parameters: `gpio_num` – GPIO number

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

`esp_err_t gpio_isr_register(void (*fn)(void*), void *arg, int intr_alloc_flags, gpio_isr_handle_t *handle)`

Register GPIO interrupt handler, the handler is an ISR. The handler will be attached to the same CPU core that this function is running on.

This ISR function is called whenever any GPIO interrupt occurs. See the alternative `gpio_install_isr_service()` and `gpio_isr_handler_add()` API in order to have the driver support per-GPIO ISRs.

To disable or remove the ISR, pass the returned handle to the [interrupt allocation functions](#).

Parameters:

- `fn` – Interrupt handler function.
- `arg` – Parameter for handler function
- `intr_alloc_flags` – Flags used to allocate the interrupt. One or multiple (ORred) `ESP_INTR_FLAG_*` values. See `esp_intr_alloc.h` for more info.
- `handle` – Pointer to return handle. If non-NULL, a handle for the interrupt will be returned here.

Returns:

- ESP_OK Success ;
- ESP_ERR_INVALID_ARG GPIO error
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags

`esp_err_t gpio_pullup_en(gpio_num_t gpio_num)`

Enable pull-up on GPIO.

Parameters: `gpio_num` – GPIO number

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

`esp_err_t gpio_pullup_dis(gpio_num_t gpio_num)`

Disable pull-up on GPIO.

Parameters: `gpio_num` – GPIO number

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_pullup_en(gpio_num_t gpio_num)

Enable pull-up on GPIO.

Parameters: **gpio_num** – GPIO number

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_pullup_dis(gpio_num_t gpio_num)

Disable pull-up on GPIO.

Parameters: **gpio_num** – GPIO number

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

esp_err_t gpio_install_isr_service(int intr_alloc_flags)

Install the GPIO driver's ETS_GPIO_INTR_SOURCE ISR handler service, which allows per-pin GPIO interrupt handlers.

This function is incompatible with `gpio_isr_register()` - if that function is used, a single global ISR is registered for all GPIO interrupts. If this function is used, the ISR service provides a global GPIO ISR and individual pin handlers are registered via the `gpio_isr_handler_add()` function.

Parameters: **intr_alloc_flags** – Flags used to allocate the interrupt. One or multiple (ORred) `ESP_INTR_FLAG_*` values. See `esp_intr_alloc.h` for more info.

Returns:

- ESP_OK Success
- ESP_ERR_NO_MEM No memory to install this service
- ESP_ERR_INVALID_STATE ISR service already installed.
- ESP_ERR_NOT_FOUND No free interrupt found with the specified flags
- ESP_ERR_INVALID_ARG GPIO error

void gpio_uninstall_isr_service(void)

Uninstall the driver's GPIO ISR service, freeing related resources.

esp_err_t gpio_isr_handler_add(gpio_num_t gpio_num, gpio_isr_t isr_handler, void *args)

Add ISR handler for the corresponding GPIO pin.

Call this function after using `gpio_install_isr_service()` to install the driver's GPIO ISR handler service.

The pin ISR handlers no longer need to be declared with `IRAM_ATTR`, unless you pass the `ESP_INTR_FLAG_IRAM` flag when allocating the ISR in `gpio_install_isr_service()`.

This ISR handler will be called from an ISR. So there is a stack size limit (configurable as "ISR stack size" in menuconfig). This limit is smaller compared to a global GPIO interrupt handler due to the additional level of indirection.

Parameters:

- **gpio_num** – GPIO number
- **isr_handler** – ISR handler function for the corresponding GPIO number.
- **args** – parameter for ISR handler.

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_STATE` Wrong state, the ISR service has not been initialized.
- `ESP_ERR_INVALID_ARG` Parameter error

esp_err_t gpio_isr_handler_remove(gpio_num_t gpio_num)

Remove ISR handler for the corresponding GPIO pin.

Parameters: **gpio_num** – GPIO number

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_STATE` Wrong state, the ISR service has not been initialized.
- `ESP_ERR_INVALID_ARG` Parameter error

esp_err_t gpio_set_drive_capability(gpio_num_t gpio_num, gpio_drive_cap_t strength)

Set GPIO pad drive capability.

Parameters:

- **gpio_num** – GPIO number, only support output GPIOs
- **strength** – Drive capability of the pad

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

`esp_err_t gpio_get_drive_capability(gpio_num_t gpio_num, gpio_drive_cap_t *strength)`

Get GPIO pad drive capability.

- Parameters:**
- `gpio_num` – GPIO number, only support output GPIOs
 - `strength` – Pointer to accept drive capability of the pad

- Returns:**
- `ESP_OK` Success
 - `ESP_ERR_INVALID_ARG` Parameter error

`esp_err_t gpio_hold_en(gpio_num_t gpio_num)`

Enable gpio pad hold function.

When a GPIO is set to hold, its state is latched at that moment and will not change when the internal signal or the IO MUX/GPIO configuration is modified (including input enable, output enable, output value, function, and drive strength values). This function can be used to retain the state of GPIOs when the chip or system is reset, for example, when watchdog time-out or Deep-sleep events are triggered.

This function works in both input and output modes, and only applicable to output-capable GPIOs. If this function is enabled: in output mode: the output level of the GPIO will be locked and can not be changed. in input mode: the input read value can still reflect the changes of the input signal.

However, on ESP32/S2/C3/S3/C2, this function cannot be used to hold the state of a digital GPIO during Deep-sleep. Even if this function is enabled, the digital GPIO will be reset to its default state when the chip wakes up from Deep-sleep. If you want to hold the state of a digital GPIO during Deep-sleep, please call `gpio_deep_sleep_hold_en`.

Power down or call `gpio_hold_dis` will disable this function.

- Parameters:** `gpio_num` – GPIO number, only support output-capable GPIOs

- Returns:**
- `ESP_OK` Success
 - `ESP_ERR_NOT_SUPPORTED` Not support pad hold function

`esp_err_t gpio_hold_dis(gpio_num_t gpio_num)`

Disable gpio pad hold function.

When the chip is woken up from Deep-sleep, the gpio will be set to the default mode, so, the gpio will output the default level if this function is called. If you don't want the level changes, the gpio should be configured to a known state before this function is called. e.g. If you hold gpio18 high during Deep-sleep, after the chip is woken up and `gpio_hold_dis` is called,

gpio18 will output low level(because gpio18 is input mode by default). If you don't want this behavior, you should configure gpio18 as output mode and set it to high level before calling `gpio_hold_dis`.

Parameters: `gpio_num` – GPIO number, only support output-capable GPIOs

Returns:

- ESP_OK Success
- ESP_ERR_NOT_SUPPORTED Not support pad hold function

`void gpio_deep_sleep_hold_en(void)`

Enable all digital gpio pads hold function during Deep-sleep.

Enabling this feature makes all digital gpio pads be at the holding state during Deep-sleep. The state of each pad holds is its active configuration (not pad's sleep configuration!).

Note that this pad hold feature only works when the chip is in Deep-sleep mode. When the chip is in active mode, the digital gpio state can be changed freely even you have called this function.

After this API is being called, the digital gpio Deep-sleep hold feature will work during every sleep process. You should call `gpio_deep_sleep_hold_dis` to disable this feature.

`void gpio_deep_sleep_hold_dis(void)`

Disable all digital gpio pads hold function during Deep-sleep.

`void gpio_iomux_in(uint32_t gpio_num, uint32_t signal_idx)`

SOC_GPIO_SUPPORT_HOLD_SINGLE_IO_IN_DSLP.

Set pad input to a peripheral signal through the IOMUX.

Parameters:

- `gpio_num` – GPIO number of the pad.
- `signal_idx` – Peripheral signal id to input. One of the `*_IN_IDX` signals in `soc/gpio_sig_map.h`.

`void gpio_iomux_out(uint8_t gpio_num, int func, bool oen_inv)`

Set peripheral output to an GPIO pad through the IOMUX.

Parameters:

- `gpio_num` – gpio_num GPIO number of the pad.
- `func` – The function number of the peripheral pin to output pin. One of the `FUNC_X_*` of specified pin (X) in `soc/io_mux_reg.h`.
- `oen_inv` – True if the output enable needs to be inverted, otherwise False.

`esp_err_t gpio_sleep_sel_en(gpio_num_t gpio_num)`

Enable SLP_SEL to change GPIO status automatically in lightsleep.

Parameters: `gpio_num` – GPIO number of the pad.

Returns:

- ESP_OK Success

`esp_err_t gpio_sleep_sel_dis(gpio_num_t gpio_num)`

Disable SLP_SEL to change GPIO status automatically in lightsleep.

Parameters: `gpio_num` – GPIO number of the pad.

Returns:

- ESP_OK Success

`esp_err_t gpio_sleep_set_direction(gpio_num_t gpio_num, gpio_mode_t mode)`

GPIO set direction at sleep.

Configure GPIO direction,such as output_only,input_only,output_and_input

Parameters:

- `gpio_num` – Configure GPIO pins number, it should be GPIO number. If you want to set direction of e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);
- `mode` – GPIO direction

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG GPIO error

`esp_err_t gpio_sleep_set_pull_mode(gpio_num_t gpio_num, gpio_pull_mode_t pull)`

Configure GPIO pull-up/pull-down resistors at sleep.

Note

ESP32: Only pins that support both input & output have integrated pull-up and pull-down resistors. Input-only GPIOs 34-39 do not.

Parameters:

- `gpio_num` – GPIO number. If you want to set pull up or down mode for e.g. GPIO16, `gpio_num` should be `GPIO_NUM_16` (16);
- `pull` – GPIO pull up/down mode.

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG : Parameter error

Structures

`struct gpio_config_t`

Configuration parameters of GPIO pad for gpio_config function.

Public Members

`uint64_t pin_bit_mask`

GPIO pin: set with bit mask, each bit maps to a GPIO

`gpio_mode_t mode`

GPIO mode: set input/output mode

`gpio_pullup_t pull_up_en`

GPIO pull-up

`gpio_pulldown_t pull_down_en`

GPIO pull-down

`gpio_int_type_t intr_type`

GPIO interrupt type

Macros

`GPIO_PIN_COUNT`

`GPIO_IS_VALID_GPIO(gpio_num)`

Check whether it is a valid GPIO number.

`GPIO_IS_VALID_OUTPUT_GPIO(gpio_num)`

Check whether it can be a valid GPIO number of output mode.

`GPIO_IS_VALID_DIGITAL_IO_PAD(gpio_num)`

Check whether it can be a valid digital I/O pad.

Type Definitions

```
typedef intr_handle_t gpio_isr_handle_t
```

```
typedef void (*gpio_isr_t)(void *arg)
```

GPIO interrupt handler.

Param arg: User registered data

Header File

- [components/hal/include/hal/gpio_types.h](#)

Macros

```
GPIO_PIN_REG_0
```

```
GPIO_PIN_REG_1
```

```
GPIO_PIN_REG_2
```

```
GPIO_PIN_REG_3
```

```
GPIO_PIN_REG_4
```

```
GPIO_PIN_REG_5
```

```
GPIO_PIN_REG_6
```

```
GPIO_PIN_REG_7
```

```
GPIO_PIN_REG_8
```

```
GPIO_PIN_REG_9
```

```
GPIO_PIN_REG_10
```

GPIO_PIN_REG_11

GPIO_PIN_REG_12

GPIO_PIN_REG_13

GPIO_PIN_REG_14

GPIO_PIN_REG_15

GPIO_PIN_REG_16

GPIO_PIN_REG_17

GPIO_PIN_REG_18

GPIO_PIN_REG_19

GPIO_PIN_REG_20

GPIO_PIN_REG_21

GPIO_PIN_REG_22

GPIO_PIN_REG_23

GPIO_PIN_REG_24

GPIO_PIN_REG_25

GPIO_PIN_REG_26

GPIO_PIN_REG_27

GPIO_PIN_REG_28

GPIO_PIN_REG_29

GPIO_PIN_REG_30

GPIO_PIN_REG_31

GPIO_PIN_REG_32

GPIO_PIN_REG_33

GPIO_PIN_REG_34

GPIO_PIN_REG_35

GPIO_PIN_REG_36

GPIO_PIN_REG_37

GPIO_PIN_REG_38

GPIO_PIN_REG_39

GPIO_PIN_REG_40

GPIO_PIN_REG_41

GPIO_PIN_REG_42

GPIO_PIN_REG_43

GPIO_PIN_REG_44

GPIO_PIN_REG_45

GPIO_PIN_REG_46

GPIO_PIN_REG_47

GPIO_PIN_REG_48

Enumerations

enum **gpio_port_t**

Values:

| *enumerator* **GPIO_PORT_0**

| *enumerator* **GPIO_PORT_MAX**

enum **gpio_num_t**

Values:

| *enumerator* **GPIO_NUM_NC**

Use to signal not connected to S/W

| *enumerator* **GPIO_NUM_0**

GPIO0, input and output

| *enumerator* **GPIO_NUM_1**

GPIO1, input and output

| *enumerator* **GPIO_NUM_2**

GPIO2, input and output

| *enumerator* **GPIO_NUM_3**

GPIO3, input and output

| *enumerator* **GPIO_NUM_4**

GPIO4, input and output

| *enumerator* **GPIO_NUM_5**

GPIO5, input and output

| *enumerator* **GPIO_NUM_6**

GPIO6, input and output

enumerator **GPIO_NUM_7**

GPIO7, input and output

enumerator **GPIO_NUM_8**

GPIO8, input and output

enumerator **GPIO_NUM_9**

GPIO9, input and output

enumerator **GPIO_NUM_10**

GPIO10, input and output

enumerator **GPIO_NUM_11**

GPIO11, input and output

enumerator **GPIO_NUM_12**

GPIO12, input and output

enumerator **GPIO_NUM_13**

GPIO13, input and output

enumerator **GPIO_NUM_14**

GPIO14, input and output

enumerator **GPIO_NUM_15**

GPIO15, input and output

enumerator **GPIO_NUM_16**

GPIO16, input and output

enumerator **GPIO_NUM_17**

GPIO17, input and output

enumerator **GPIO_NUM_18**

GPIO18, input and output

enumerator **GPIO_NUM_19**

GPIO19, input and output

enumerator **GPIO_NUM_20**

GPIO20, input and output

enumerator **GPIO_NUM_21**

GPIO21, input and output

enumerator **GPIO_NUM_22**

GPIO22, input and output

enumerator **GPIO_NUM_23**

GPIO23, input and output

enumerator **GPIO_NUM_25**

GPIO25, input and output

enumerator **GPIO_NUM_26**

GPIO26, input and output

enumerator **GPIO_NUM_27**

GPIO27, input and output

enumerator **GPIO_NUM_28**

GPIO28, input and output

enumerator **GPIO_NUM_29**

GPIO29, input and output

enumerator **GPIO_NUM_30**

GPIO30, input and output

enumerator **GPIO_NUM_31**

GPIO31, input and output

enumerator **GPIO_NUM_32**

GPIO32, input and output

enumerator **GPIO_NUM_33**

GPIO33, input and output

enumerator **GPIO_NUM_34**

GPIO34, input mode only

enumerator **GPIO_NUM_35**

GPIO35, input mode only

enumerator **GPIO_NUM_36**

GPIO36, input mode only

enumerator **GPIO_NUM_37**

GPIO37, input mode only

enumerator **GPIO_NUM_38**

GPIO38, input mode only

enumerator **GPIO_NUM_39**

GPIO39, input mode only

enumerator **GPIO_NUM_MAX**

enum **gpio_int_type_t**

Values:

enumerator **GPIO_INTR_DISABLE**

Disable GPIO interrupt

enumerator **GPIO_INTR_POSEDGE**

GPIO interrupt type : rising edge

enumerator **GPIO_INTR_NEGEDGE**

GPIO interrupt type : falling edge

enumerator **GPIO_INTR_ANYEDGE**

GPIO interrupt type : both rising and falling edge

enumerator **GPIO_INTR_LOW_LEVEL**

GPIO interrupt type : input low level trigger

enumerator **GPIO_INTR_HIGH_LEVEL**

GPIO interrupt type : input high level trigger

enumerator **GPIO_INTR_MAX**

enum **gpio_mode_t**

Values:

enumerator **GPIO_MODE_DISABLE**

GPIO mode : disable input and output

enumerator **GPIO_MODE_INPUT**

GPIO mode : input only

enumerator **GPIO_MODE_OUTPUT**

GPIO mode : output only mode

enumerator **GPIO_MODE_OUTPUT_OD**

GPIO mode : output only with open-drain mode

enumerator **GPIO_MODE_INPUT_OUTPUT_OD**

GPIO mode : output and input with open-drain mode

enumerator **GPIO_MODE_INPUT_OUTPUT**

GPIO mode : output and input mode

enum **gpio_pullup_t**

Values:

enumerator **GPIO_PULLUP_DISABLE**

Disable GPIO pull-up resistor

enumerator **GPIO_PULLUP_ENABLE**

Enable GPIO pull-up resistor

enum **gpio_pulldown_t**

Values:

enumerator **GPIO_PULLDOWN_DISABLE**

Disable GPIO pull-down resistor

enumerator **GPIO_PULLDOWN_ENABLE**

Enable GPIO pull-down resistor

enum **gpio_pull_mode_t**

Values:

enumerator **GPIO_PULLUP_ONLY**

Pad pull up

enumerator **GPIO_PULLDOWN_ONLY**

Pad pull down

enumerator **GPIO_PULLUP_PULLDOWN**

Pad pull up + pull down

enumerator **GPIO_FLOATING**

Pad floating

enum **gpio_drive_cap_t**

Values:

enumerator **GPIO_DRIVE_CAP_0**

Pad drive capability: weak

enumerator **GPIO_DRIVE_CAP_1**

Pad drive capability: stronger

enumerator **GPIO_DRIVE_CAP_2**

Pad drive capability: medium

enumerator **GPIO_DRIVE_CAP_DEFAULT**

Pad drive capability: medium

enumerator **GPIO_DRIVE_CAP_3**

Pad drive capability: strongest

enumerator **GPIO_DRIVE_CAP_MAX**

enum **gpio_hys_ctrl_mode_t**

Available option for configuring hysteresis feature of GPIOs.

Values:

enumerator **GPIO_HYS_CTRL_EFUSE**

Pad input hysteresis ctrl by efuse

enumerator **GPIO_HYS_SOFT_ENABLE**

Pad input hysteresis enable by software

enumerator **GPIO_HYS_SOFT_DISABLE**

Pad input hysteresis disable by software

API Reference - RTC GPIO

Header File

- [components/driver/gpio/include/driver/rtc_io.h](#)

Functions

bool **rtc_gpio_is_valid_gpio**(*gpio_num_t* gpio_num)

Determine if the specified GPIO is a valid RTC GPIO.

Parameters: `gpio_num` – GPIO number

Returns: true if GPIO is valid for RTC GPIO use. false otherwise.

`int rtc_io_number_get(gpio_num_t gpio_num)`

Get RTC IO index number by gpio number.

Parameters: `gpio_num` – GPIO number

Returns: `>=0`: Index of rtcio. `-1` : The gpio is not rtcio.

`esp_err_t rtc_gpio_init(gpio_num_t gpio_num)`

Init a GPIO as RTC GPIO.

This function must be called when initializing a pad for an analog function.

Parameters: `gpio_num` – GPIO number (e.g. `GPIO_NUM_12`)

Returns:

- `ESP_OK` success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

`esp_err_t rtc_gpio_deinit(gpio_num_t gpio_num)`

Init a GPIO as digital GPIO.

Parameters: `gpio_num` – GPIO number (e.g. `GPIO_NUM_12`)

Returns:

- `ESP_OK` success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

`uint32_t rtc_gpio_get_level(gpio_num_t gpio_num)`

Get the RTC IO input level.

Parameters: `gpio_num` – GPIO number (e.g. `GPIO_NUM_12`)

Returns:

- 1 High level
- 0 Low level
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

`esp_err_t rtc_gpio_set_level(gpio_num_t gpio_num, uint32_t level)`

Set the RTC IO output level.

- Parameters:**
- **gpio_num** – GPIO number (e.g. GPIO_NUM_12)
 - **level** – output level

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_set_direction(gpio_num_t gpio_num, rtc_gpio_mode_t mode)

RTC GPIO set direction.

Configure RTC GPIO direction, such as output only, input only, output and input.

- Parameters:**
- **gpio_num** – GPIO number (e.g. GPIO_NUM_12)
 - **mode** – GPIO direction

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_set_direction_in_sleep(gpio_num_t gpio_num, rtc_gpio_mode_t mode)

RTC GPIO set direction in deep sleep mode or disable sleep status (default). In some application scenarios, IO needs to have another states during deep sleep.

NOTE: ESP32 support INPUT_ONLY mode. ESP32S2 support INPUT_ONLY, OUTPUT_ONLY, INPUT_OUTPUT mode.

- Parameters:**
- **gpio_num** – GPIO number (e.g. GPIO_NUM_12)
 - **mode** – GPIO direction

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

esp_err_t rtc_gpio_pullup_en(gpio_num_t gpio_num)

RTC GPIO pullup enable.

This function only works for RTC IOs. In general, call gpio_pullup_en, which will work both for normal GPIOs and RTC IOs.

- Parameters:** **gpio_num** – GPIO number (e.g. GPIO_NUM_12)

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

`esp_err_t rtc_gpio_pulldown_en(gpio_num_t gpio_num)`

RTC GPIO pulldown enable.

This function only works for RTC IOs. In general, call `gpio_pulldown_en`, which will work both for normal GPIOs and RTC IOs.

Parameters: `gpio_num` – GPIO number (e.g. `GPIO_NUM_12`)

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

`esp_err_t rtc_gpio_pullup_dis(gpio_num_t gpio_num)`

RTC GPIO pullup disable.

This function only works for RTC IOs. In general, call `gpio_pullup_dis`, which will work both for normal GPIOs and RTC IOs.

Parameters: `gpio_num` – GPIO number (e.g. `GPIO_NUM_12`)

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

`esp_err_t rtc_gpio_pulldown_dis(gpio_num_t gpio_num)`

RTC GPIO pulldown disable.

This function only works for RTC IOs. In general, call `gpio_pulldown_dis`, which will work both for normal GPIOs and RTC IOs.

Parameters: `gpio_num` – GPIO number (e.g. `GPIO_NUM_12`)

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` GPIO is not an RTC IO

`esp_err_t rtc_gpio_set_drive_capability(gpio_num_t gpio_num, gpio_drive_cap_t strength)`

Set RTC GPIO pad drive capability.

Parameters:

- `gpio_num` – GPIO number, only support output GPIOs
- `strength` – Drive capability of the pad

Returns:

- `ESP_OK` Success
- `ESP_ERR_INVALID_ARG` Parameter error

`esp_err_t rtc_gpio_get_drive_capability(gpio_num_t gpio_num, gpio_drive_cap_t *strength)`

Get RTC GPIO pad drive capability.

- Parameters:**
- **gpio_num** – GPIO number, only support output GPIOs
 - **strength** – Pointer to accept drive capability of the pad

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG Parameter error

`esp_err_t rtc_gpio_hold_en(gpio_num_t gpio_num)`

Enable hold function on an RTC IO pad.

Enabling HOLD function will cause the pad to latch current values of input enable, output enable, output value, function, drive strength values. This function is useful when going into light or deep sleep mode to prevent the pin configuration from changing.

- Parameters:** **gpio_num** – GPIO number (e.g. GPIO_NUM_12)

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

`esp_err_t rtc_gpio_hold_dis(gpio_num_t gpio_num)`

Disable hold function on an RTC IO pad.

Disabling hold function will allow the pad receive the values of input enable, output enable, output value, function, drive strength from RTC_IO peripheral.

- Parameters:** **gpio_num** – GPIO number (e.g. GPIO_NUM_12)

- Returns:**
- ESP_OK Success
 - ESP_ERR_INVALID_ARG GPIO is not an RTC IO

`esp_err_t rtc_gpio_isolate(gpio_num_t gpio_num)`

Helper function to disconnect internal circuits from an RTC IO This function disables input, output, pullup, pulldown, and enables hold feature for an RTC IO. Use this function if an RTC IO needs to be disconnected from internal circuits in deep sleep, to minimize leakage current.

In particular, for ESP32-WROVER module, call `rtc_gpio_isolate(GPIO_NUM_12)` before entering deep sleep, to reduce deep sleep current.

- Parameters:** **gpio_num** – GPIO number (e.g. GPIO_NUM_12).

Returns:

- ESP_OK on success
- ESP_ERR_INVALID_ARG if GPIO is not an RTC IO

esp_err_t rtc_gpio_force_hold_en_all(void)

Enable force hold signal for all RTC IOs.

Each RTC pad has a “force hold” input signal from the RTC controller. If this signal is set, pad latches current values of input enable, function, output enable, and other signals which come from the RTC mux. Force hold signal is enabled before going into deep sleep for pins which are used for EXT1 wakeup.

esp_err_t rtc_gpio_force_hold_dis_all(void)

Disable force hold signal for all RTC IOs.

esp_err_t rtc_gpio_wakeup_enable(gpio_num_t gpio_num, gpio_int_type_t intr_type)

Enable wakeup from sleep mode using specific GPIO.

- Parameters:**
- **gpio_num** – GPIO number
 - **intr_type** – Wakeup on high level (GPIO_INTR_HIGH_LEVEL) or low level (GPIO_INTR_LOW_LEVEL)

- Returns:**
- ESP_OK on success
 - ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO, or intr_type is not one of GPIO_INTR_HIGH_LEVEL, GPIO_INTR_LOW_LEVEL.

esp_err_t rtc_gpio_wakeup_disable(gpio_num_t gpio_num)

Disable wakeup from sleep mode using specific GPIO.

- Parameters:** **gpio_num** – GPIO number

- Returns:**
- ESP_OK on success
 - ESP_ERR_INVALID_ARG if gpio_num is not an RTC IO

Macros

RTC_GPIO_IS_VALID_GPIO(gpio_num)

Header File

- [components/hal/include/hal/rtc_io_types.h](#)

Enumerations

***enum* rtc_gpio_mode_t**

RTCIO output/input mode type.

Values:

***enumerator* RTC_GPIO_MODE_INPUT_ONLY**

Pad input

***enumerator* RTC_GPIO_MODE_OUTPUT_ONLY**

Pad output

***enumerator* RTC_GPIO_MODE_INPUT_OUTPUT**

Pad input + output

***enumerator* RTC_GPIO_MODE_DISABLED**

Pad (output + input) disable

***enumerator* RTC_GPIO_MODE_OUTPUT_OD**

Pad open-drain output

***enumerator* RTC_GPIO_MODE_INPUT_OUTPUT_OD**

Pad input + open-drain output

[Provide feedback about this document](#)