Prática 3

General Purpose Timer (GPTimer)

General Purpose Timer (GPTimer)

- The ESP32 chip contains two hardware timer groups. Each group has two general-purpose hardware timers.
- 4 general-purpose timers embedded in the ESP32.
- 64-bit generic timers based on 16-bit prescalers
- 64-bit auto-reload-capable up/down counters.

https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32/api-reference/peripherals/timer.html

GPTimer

The timers feature:

A 16-bit clock prescaler, from 2 to 65536

A 64-bit time-base counter

Configurable up/down time-base counter: incrementing or decrementing

Halt and resume of time-base counter

Auto-reload at alarm

Software-controlled instant reload

Level and edge interrupt generation.

Fonte de clock APB

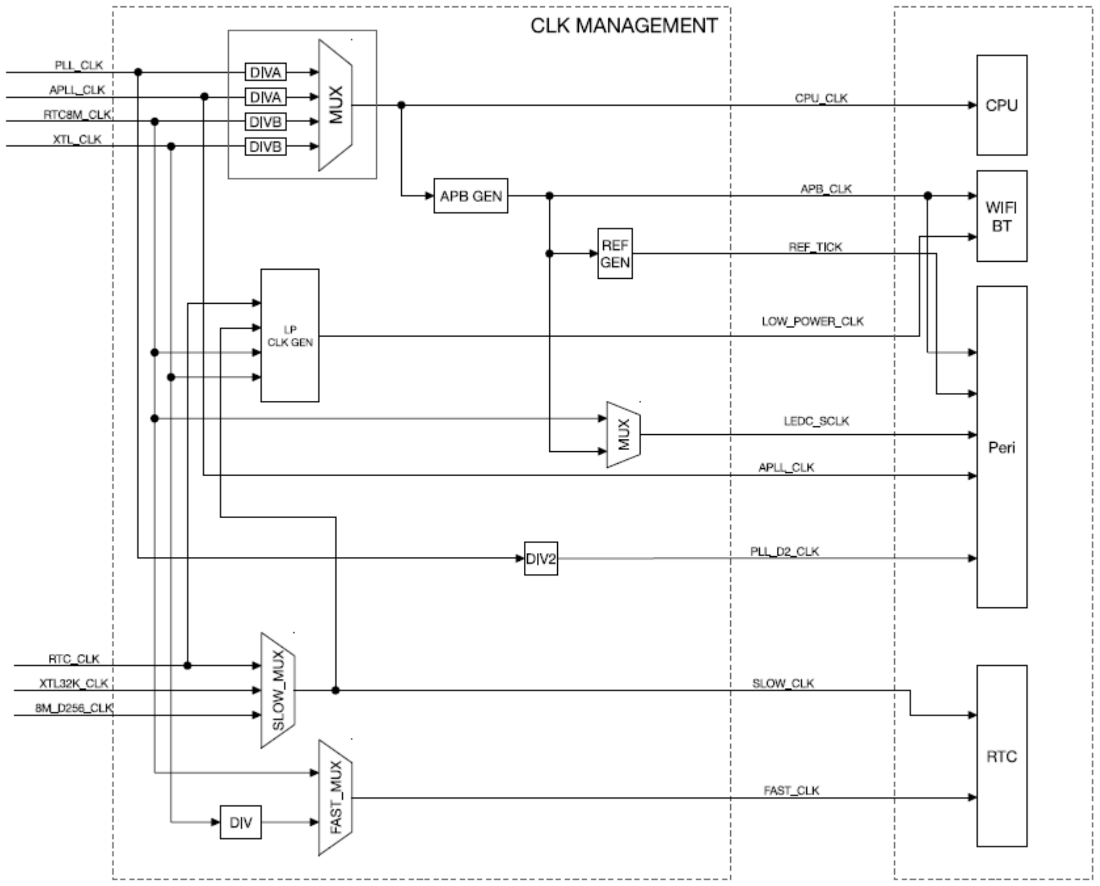


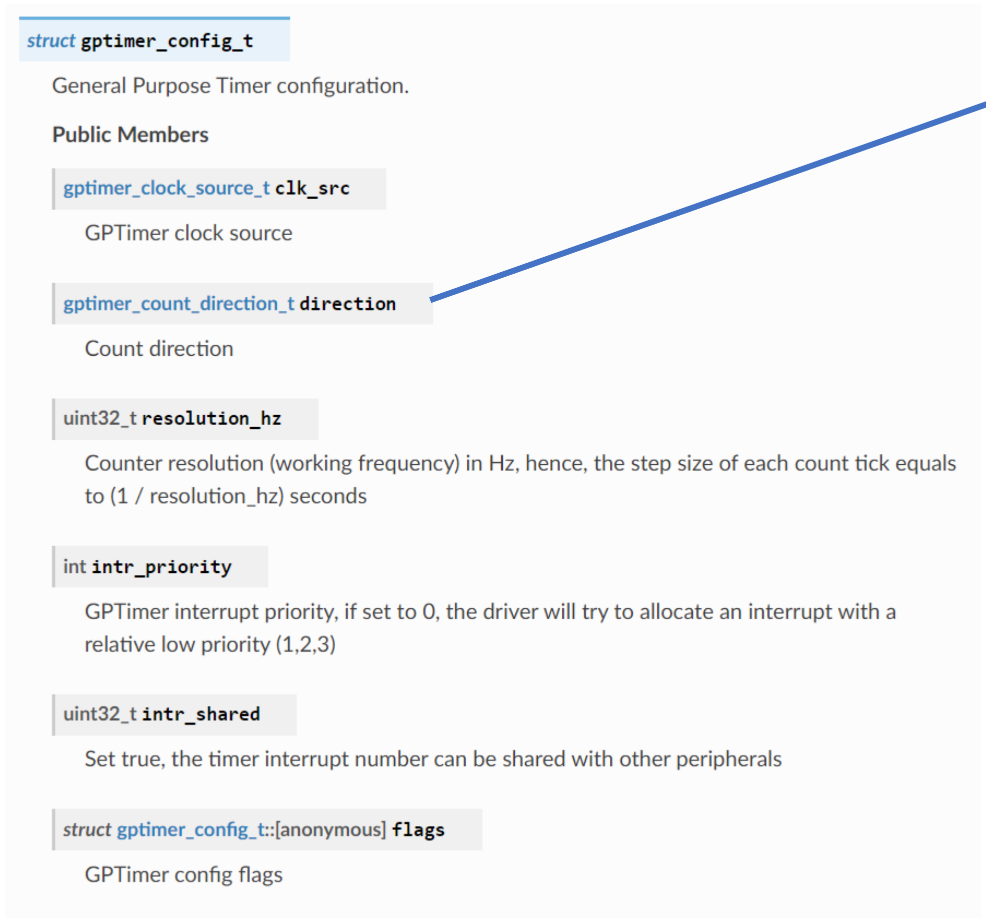
Table 3-4. Peripheral Clock Usage

Peripherals	APB_CLK	REF_TICK	LEDC_SCLK	APLL_CLK	PLL_D2_CLK
EMAC	Y	N	N	Y	N
TIMG	Y	N	N	N	N
I2S	Y	N	N	Y	Y
UART	Y	Y	N	N	N
RMT	Y	Y	N	N	N
LED PWM	Y	Y	Y	N	N
PWM	Y	N	N	N	N
I2C	Y	N	N	N	N
SPI	Y	N	N	N	N
PCNT	Y	N	N	N	N
eFuse Controller	Y	N	N	N	N
SDIO Slave	Y	N	N	N	N
SDMMC	Y	N	N	N	N

Table 3-5. APB_CLK Derivation

CPU_CLK Source	APB_CLK
PLL_CLK	80 MHz
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

Initialization



struct **gptimer_config_t**

General Purpose Timer configuration.

Public Members

gptimer_clock_source_t **clk_src**

GPTimer clock source

gptimer_count_direction_t **direction**

Count direction

uint32_t **resolution_hz**

Counter resolution (working frequency) in Hz, hence, the step size of each count tick equals to (1 / resolution_hz) seconds

int **intr_priority**

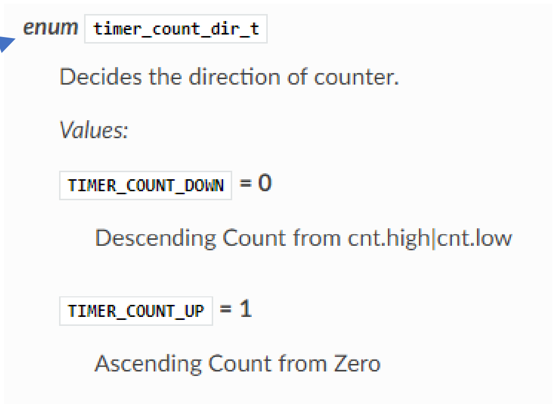
GPTimer interrupt priority, if set to 0, the driver will try to allocate an interrupt with a relative low priority (1,2,3)

uint32_t **intr_shared**

Set true, the timer interrupt number can be shared with other peripherals

struct gptimer_config_t::[anonymous] **flags**

GPTimer config flags



enum `timer_count_dir_t`

Decides the direction of counter.

Values:

`TIMER_COUNT_DOWN` = 0

Descending Count from cnt.high|cnt.low

`TIMER_COUNT_UP` = 1

Ascending Count from Zero

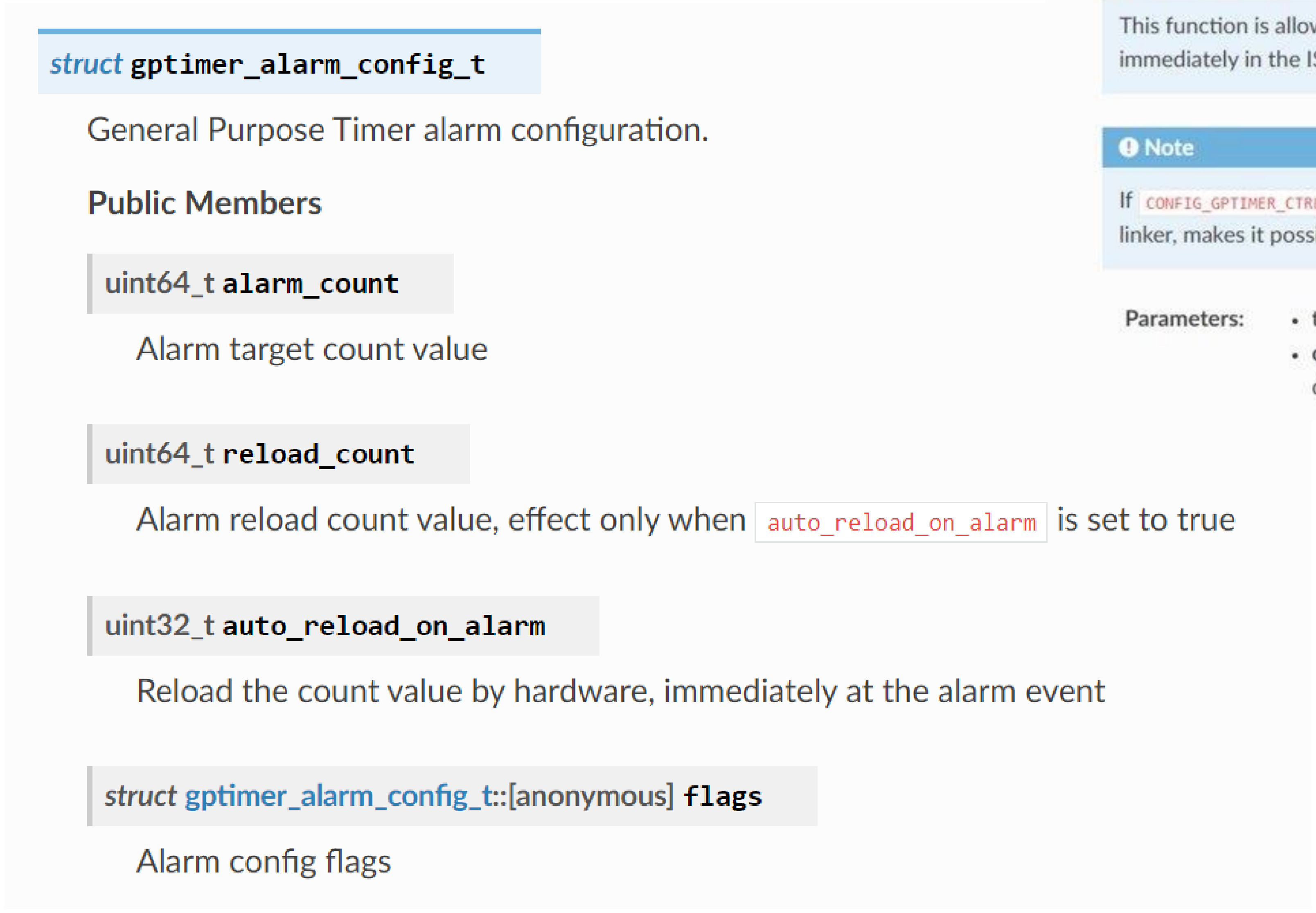
Initialization - example

```
gptimer_handle_t gptimer = NULL;
gptimer_config_t timer_config = {
    .clk_src = GPTIMER_CLK_SRC_DEFAULT,
    .direction = GPTIMER_COUNT_UP,
    .resolution_hz = 1 * 1000 * 1000, // 1MHz, 1 tick = 1us
};
ESP_ERROR_CHECK(gptimer_new_timer(&timer_config, &gptimer));
```

Set and Get Count Value

- When the GPTimer is created, the internal counter will be reset to zero by default
- The counter value can be updated asynchronously by gptimer_set_raw_count()
- Count value can be retrieved by gptimer_get_raw_count(), at any time.

Set up Alarm Action



struct **gptimer_alarm_config_t**

General Purpose Timer alarm configuration.

Public Members

uint64_t **alarm_count**

Alarm target count value

uint64_t **reload_count**

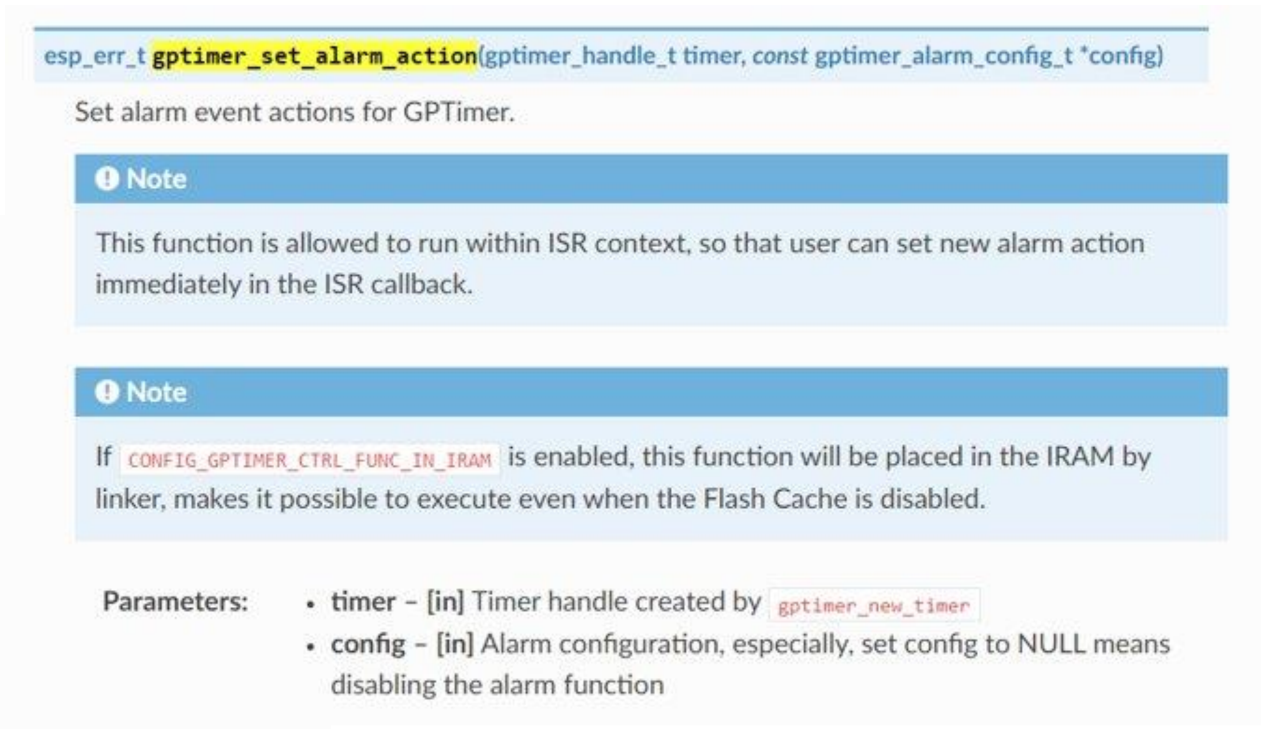
Alarm reload count value, effect only when `auto_reload_on_alarm` is set to true

uint32_t **auto_reload_on_alarm**

Reload the count value by hardware, immediately at the alarm event

struct gptimer_alarm_config_t::[anonymous] **flags**

Alarm config flags



esp_err_t **gptimer_set_alarm_action**(gptimer_handle_t timer, *const* gptimer_alarm_config_t *config)

Set alarm event actions for GPTimer.

Note

This function is allowed to run within ISR context, so that user can set new alarm action immediately in the ISR callback.

Note

If `CONFIG_GPTIMER_CTRL_FUNC_IN_IRAM` is enabled, this function will be placed in the IRAM by linker, makes it possible to execute even when the Flash Cache is disabled.

Parameters:

- **timer** – [in] Timer handle created by `gptimer_new_timer`
- **config** – [in] Alarm configuration, especially, set config to NULL means disabling the alarm function

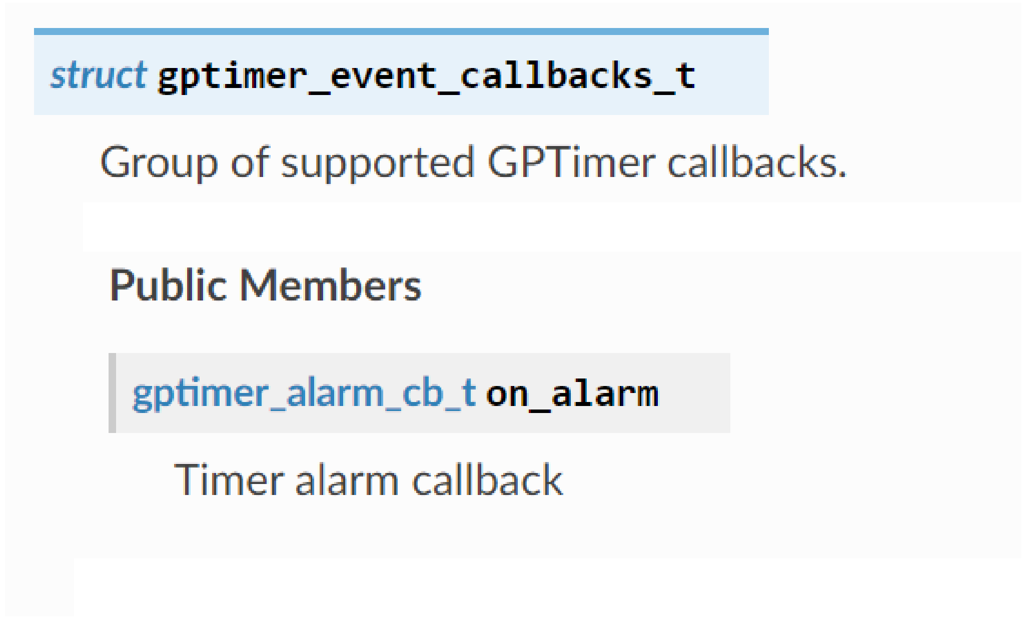
Set up Alarm Action - Example

```
gptimer_alarm_config_t alarm_config2 = {
        .reload_count = 0,
        .alarm_count = 1000000,
        .flags.auto_reload_on_alarm = true,
    };

ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config2));
```

Register Event Callbacks

- hook your function to the interrupt service routine by calling gptimer_register_event_callbacks()
- The user data will be directly passed to the callback function.



struct `gptimer_event_callbacks_t`

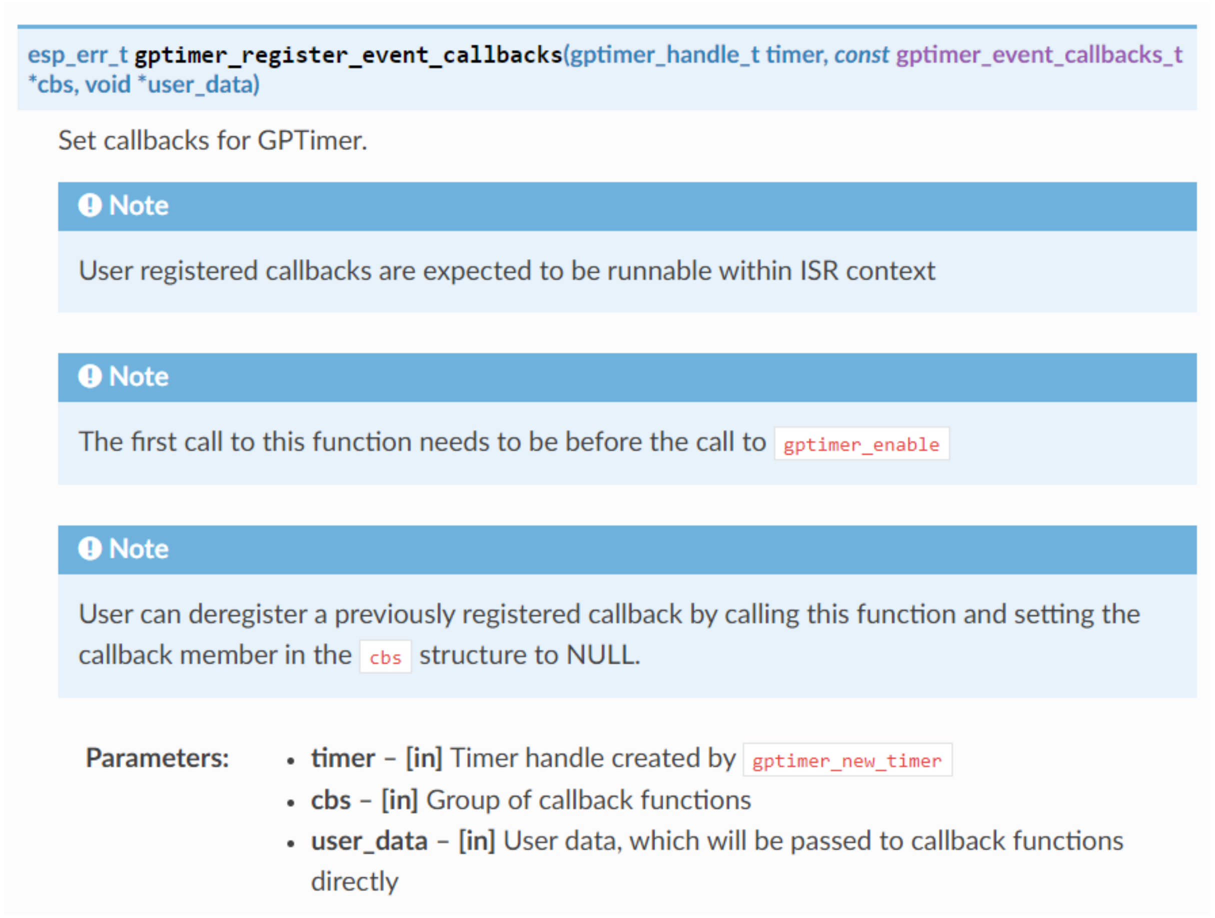
Group of supported GPTimer callbacks.

Public Members

gptimer_alarm_cb_t `on_alarm`

Timer alarm callback

Register Event Callbacks



esp_err_t **gptimer_register_event_callbacks**(gptimer_handle_t timer, *const* gptimer_event_callbacks_t *cbs, void *user_data)

Set callbacks for GPTimer.

Note

User registered callbacks are expected to be runnable within ISR context

Note

The first call to this function needs to be before the call to `gptimer_enable`

Note

User can deregister a previously registered callback by calling this function and setting the callback member in the `cbs` structure to NULL.

Parameters:

- **timer** – **[in]** Timer handle created by `gptimer_new_timer`
- **cbs** – **[in]** Group of callback functions
- **user_data** – **[in]** User data, which will be passed to callback functions directly

Register Event Callbacks

```
static bool IRAM_ATTR example_timer_on_alarm_cb_v1(gptimer_handle_t timer, const gptimer_alarm_event_data_t
*edata, void *user_data)
{
  ------
}


  gptimer_event_callbacks_t cbs = {
        .on_alarm = example_timer_on_alarm_cb_v1,
    };
    ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
```

Enable and Disable Timer

- Before doing IO control to the timer, you needs to enable the timer first, by calling gptimer_enable().
  - Switch the timer driver state from init to enable.
  - Enable the interrupt service if it has been lazy installed by gptimer_register_event_callbacks().
  - Acquire a proper power management lock if a specific clock source (e.g. APB clock) is selected.

Start and Stop Timer

- gptimer_start() can make the internal counter work,
- gptimer_stop() can make the counter stop working.

Example

```
gptimer_config_t timer_config = {
        .clk_src = GPTIMER_CLK_SRC_DEFAULT,
        .direction = GPTIMER_COUNT_UP,
        .resolution_hz = 1000000, // 1MHz, 1 tick=1us
    };
    ESP_ERROR_CHECK(gptimer_new_timer(&timer_config, gptimer1));

gptimer_event_callbacks_t cbs = {
      .on_alarm = example_timer_on_alarm_cb_v1,
  };
 ESP_ERROR_CHECK(gptimer_register_event_callbacks(gptimer, &cbs, queue));
```

ESP_ERROR_CHECK(gptimer_enable(gptimer));

```
 gptimer_alarm_config_t alarm_config = {
   .reload_count = 0,
   .alarm_count = 1000000,
   .flags.auto_reload_on_alarm = false,
 };
 ESP_ERROR_CHECK(gptimer_set_alarm_action(gptimer, &alarm_config));
```

ESP_ERROR_CHECK(gptimer_start(gptimer));

Referências

- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gptimer.html