Prática 4

LED PWM Controller

Binary Semaphores

LED PWM Controller (LEDC)

- The LED control (LEDC) peripheral is primarily designed to control the intensity of LEDs, although it can also be used to generate PWM signals for other purposes.
- It has 16 channels which can generate independent waveforms that can be used, for example, to drive RGB LED devices.

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html#ledc-api-configure-timer

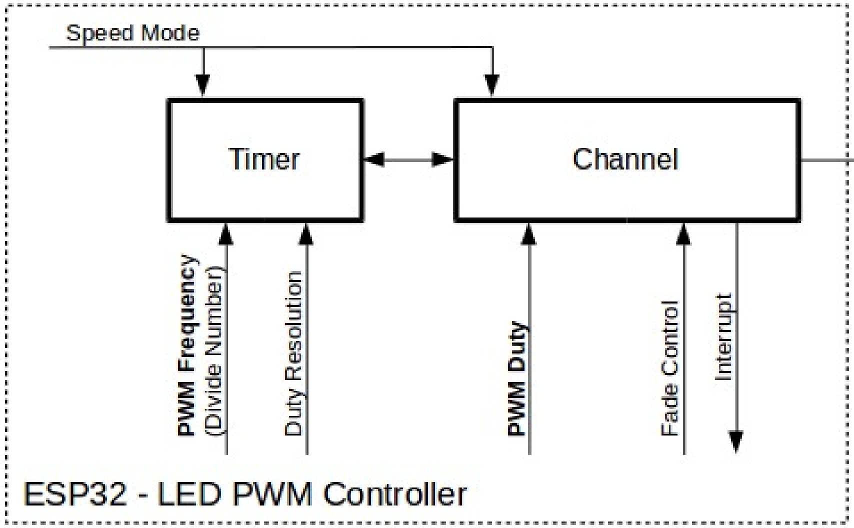
LED PWM Controller (LEDC)

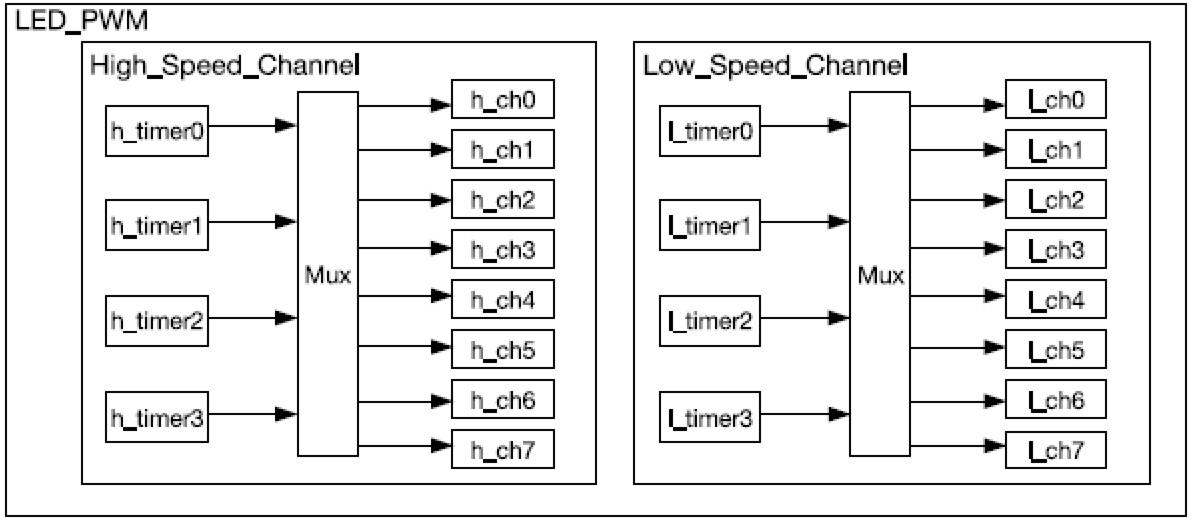
Three steps:

Timer Configuration by specifying the PWM signal's frequency and duty cycle resolution.

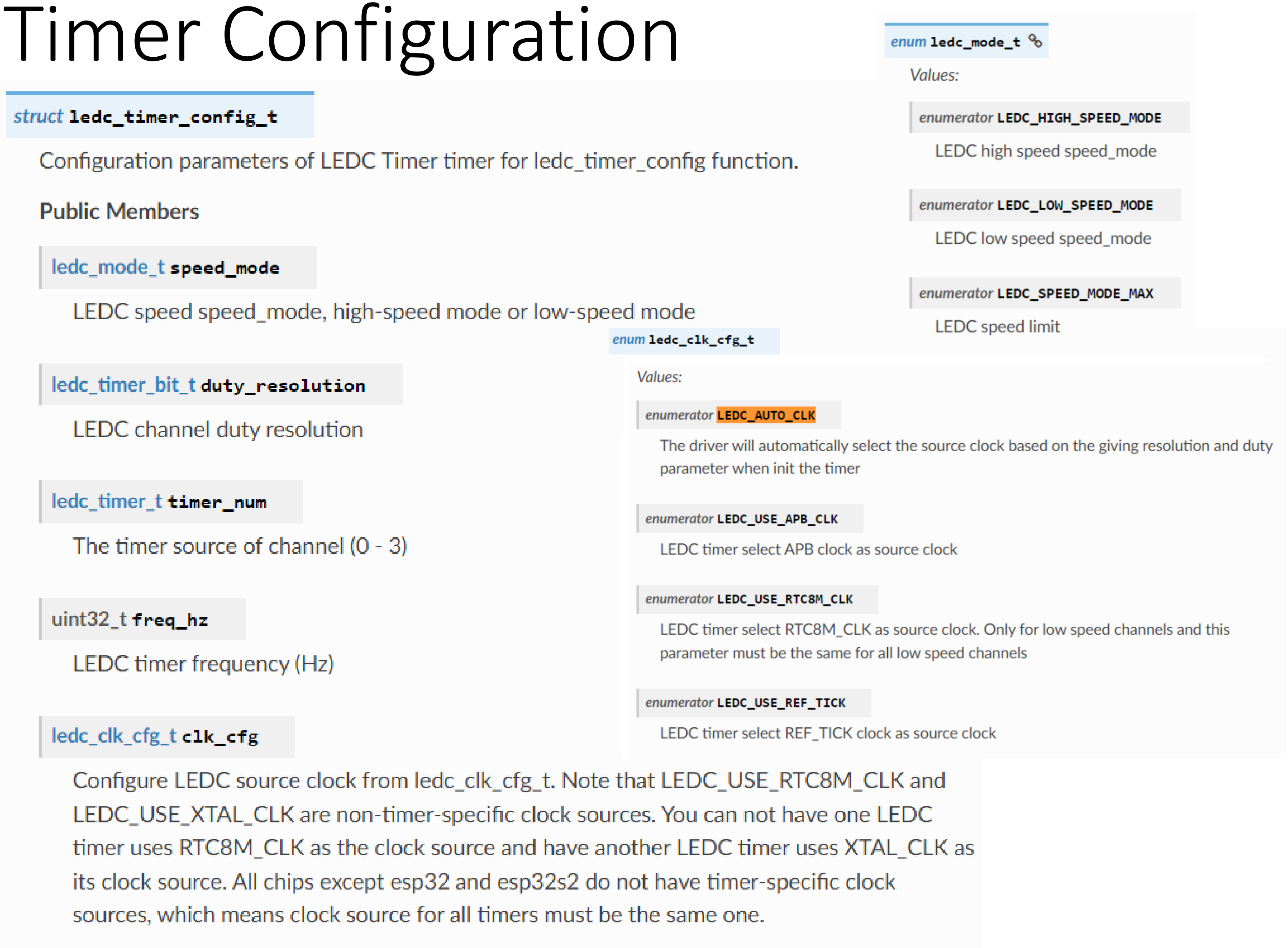
Channel Configuration by associating it with the timer and GPIO to output the PWM signal.

Change PWM Signal that drives the output in order to change LED's intensity. This can be done under the full control of software or with hardware fading functions.





Timer Configuration



struct **ledc_timer_config_t**

Configuration parameters of LEDC Timer timer for ledc_timer_config function.

Public Members

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode or low-speed mode

ledc_timer_bit_t **duty_resolution**

LEDC channel duty resolution

ledc_timer_t **timer_num**

The timer source of channel (0 - 3)

uint32_t **freq_hz**

LEDC timer frequency (Hz)

ledc_clk_cfg_t **clk_cfg**

Configure LEDC source clock from ledc_clk_cfg_t. Note that LEDC_USE_RTC8M_CLK and LEDC_USE_XTAL_CLK are non-timer-specific clock sources. You can not have one LEDC timer uses RTC8M_CLK as the clock source and have another LEDC timer uses XTAL_CLK as its clock source. All chips except esp32 and esp32s2 do not have timer-specific clock sources, which means clock source for all timers must be the same one.

enum **ledc_mode_t**

Values:

enumerator **LEDC_HIGH_SPEED_MODE**

LEDC high speed speed_mode

enumerator **LEDC_LOW_SPEED_MODE**

LEDC low speed speed_mode

enumerator **LEDC_SPEED_MODE_MAX**

LEDC speed limit

enum **ledc_clk_cfg_t**

Values:

enumerator **LEDC_AUTO_CLK**

The driver will automatically select the source clock based on the giving resolution and duty parameter when init the timer

enumerator **LEDC_USE_APB_CLK**

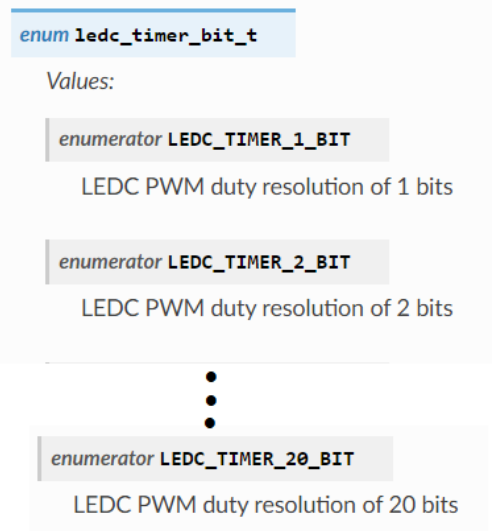
LEDC timer select APB clock as source clock

enumerator **LEDC_USE_RTC8M_CLK**

LEDC timer select RTC8M_CLK as source clock. Only for low speed channels and this parameter must be the same for all low speed channels

enumerator **LEDC_USE_REF_TICK**

LEDC timer select REF_TICK clock as source clock



enum **ledc_timer_bit_t**

Values:

enumerator **LEDC_TIMER_1_BIT**

LEDC PWM duty resolution of 1 bits

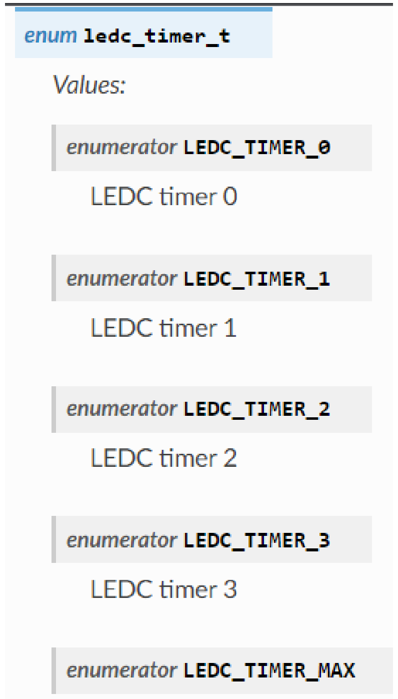
enumerator **LEDC_TIMER_2_BIT**

LEDC PWM duty resolution of 2 bits

⋮

enumerator **LEDC_TIMER_20_BIT**

LEDC PWM duty resolution of 20 bits



enum **ledc_timer_t**

Values:

enumerator **LEDC_TIMER_0**

LEDC timer 0

enumerator **LEDC_TIMER_1**

LEDC timer 1

enumerator **LEDC_TIMER_2**

LEDC timer 2

enumerator **LEDC_TIMER_3**

LEDC timer 3

enumerator **LEDC_TIMER_MAX**

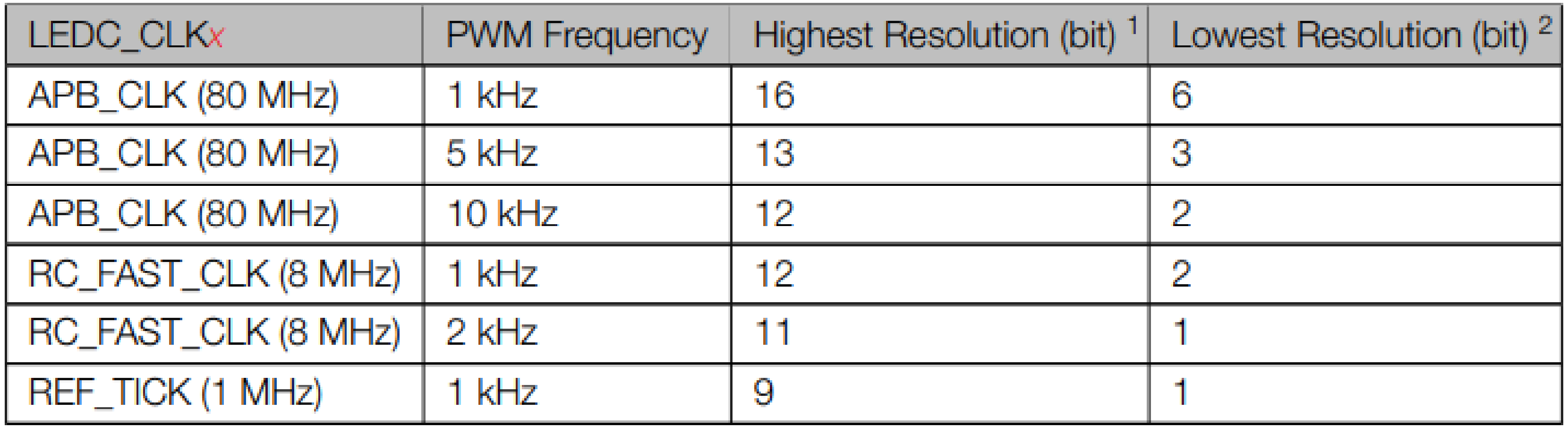
ledc_timer_config()

Exemplo

ledc_timer_config()

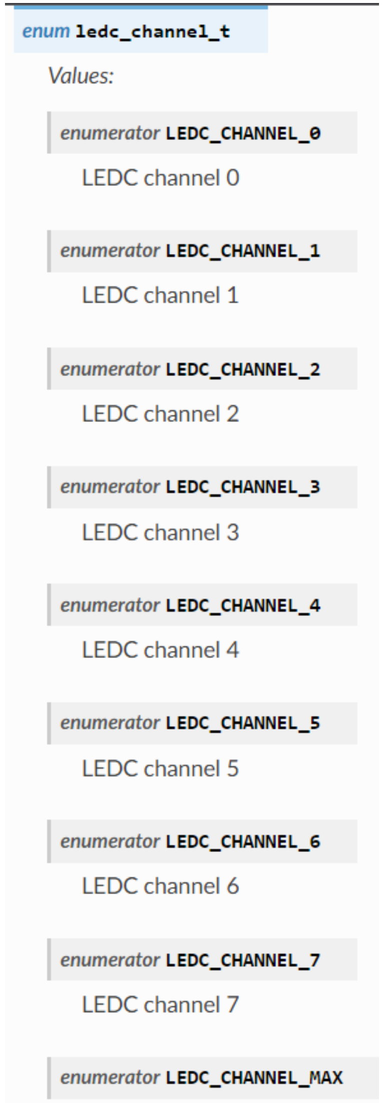
```
#define LEDC_HS_TIMER          LEDC_TIMER_0
#define LEDC_HS_MODE           LEDC_HIGH_SPEED_MODE

ledc_timer_config_t ledc_timer = {
        .duty_resolution = LEDC_TIMER_13_BIT, // resolution of PWM duty
        .freq_hz = 5000,                      // frequency of PWM signal
        .speed_mode = LEDC_HS_MODE,           // timer mode
        .timer_num = LEDC_HS_TIMER,            // timer index
        .clk_cfg = LEDC_AUTO_CLK,              // Auto select the source clock
    };
```



LEDC_CLKx	PWM Frequency	Highest Resolution (bit) [1]	Lowest Resolution (bit) [2]
APB_CLK (80 MHz)	1 kHz	16	6
APB_CLK (80 MHz)	5 kHz	13	3
APB_CLK (80 MHz)	10 kHz	12	2
RC_FAST_CLK (8 MHz)	1 kHz	12	2
RC_FAST_CLK (8 MHz)	2 kHz	11	1
REF_TICK (1 MHz)	1 kHz	9	1

Channel Configuration



enum **ledc_channel_t**

Values:

enumerator **LEDC_CHANNEL_0**

LEDC channel 0

enumerator **LEDC_CHANNEL_1**

LEDC channel 1

enumerator **LEDC_CHANNEL_2**

LEDC channel 2

enumerator **LEDC_CHANNEL_3**

LEDC channel 3

enumerator **LEDC_CHANNEL_4**

LEDC channel 4

enumerator **LEDC_CHANNEL_5**

LEDC channel 5

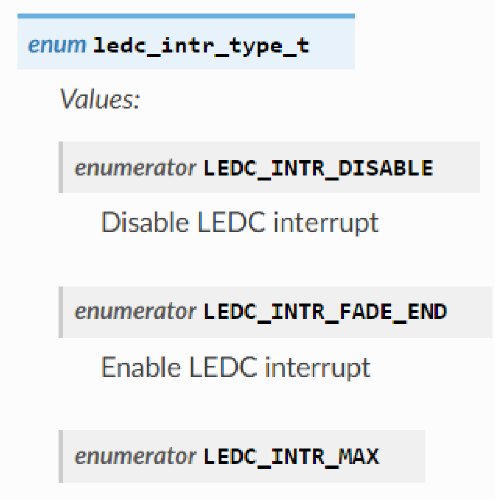
enumerator **LEDC_CHANNEL_6**

LEDC channel 6

enumerator **LEDC_CHANNEL_7**

LEDC channel 7

enumerator **LEDC_CHANNEL_MAX**



enum **ledc_intr_type_t**

Values:

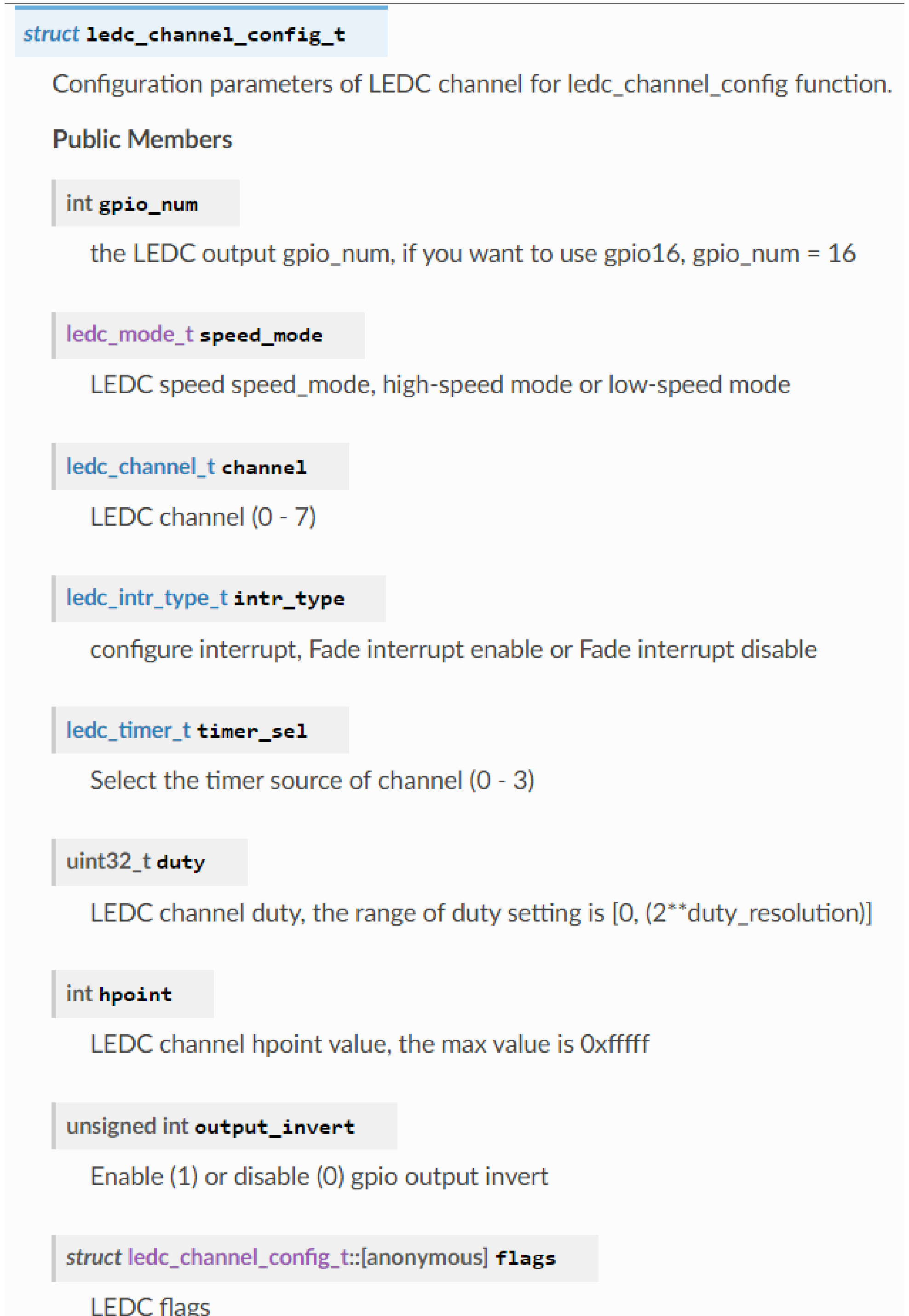
enumerator **LEDC_INTR_DISABLE**

Disable LEDC interrupt

enumerator **LEDC_INTR_FADE_END**

Enable LEDC interrupt

enumerator **LEDC_INTR_MAX**



struct **ledc_channel_config_t**

Configuration parameters of LEDC channel for ledc_channel_config function.

Public Members

int **gpio_num**

the LEDC output gpio_num, if you want to use gpio16, gpio_num = 16

ledc_mode_t **speed_mode**

LEDC speed speed_mode, high-speed mode or low-speed mode

ledc_channel_t **channel**

LEDC channel (0 - 7)

ledc_intr_type_t **intr_type**

configure interrupt, Fade interrupt enable or Fade interrupt disable

ledc_timer_t **timer_sel**

Select the timer source of channel (0 - 3)

uint32_t **duty**

LEDC channel duty, the range of duty setting is [0, (2**duty_resolution)]

int **hpoint**

LEDC channel hpoint value, the max value is 0xfffff

unsigned int **output_invert**

Enable (1) or disable (0) gpio output invert

struct ledc_channel_config_t::[anonymous] **flags**

LEDC flags

ledc_channel_config()

Exemplo

```
#define LEDC_HS_CH0_CHANNEL    LEDC_CHANNEL_0
#define LEDC_HS_CH0_GPIO       (18)
#define LEDC_HS_MODE           LEDC_HIGH_SPEED_MODE
#define LEDC_HS_TIMER          LEDC_TIMER_0
```

```
ledc_channel_config_t ledc_channel =  {
            .channel    = LEDC_HS_CH0_CHANNEL,
            .duty       = 0,
            .gpio_num   = LEDC_HS_CH0_GPIO,
            .speed_mode = LEDC_HS_MODE,
            .hpoint     = 0,
            .timer_sel  = LEDC_HS_TIMER,
            .flags.output_invert = 0
        };
```

ledc_channel_config()

Change PWM Signal

Using Software

To set the duty cycle, use the dedicated function ledc_set_duty(). After that, call ledc_update_duty() to activate the changes. To check the currently set value, use the corresponding _get_ function ledc_get_duty().

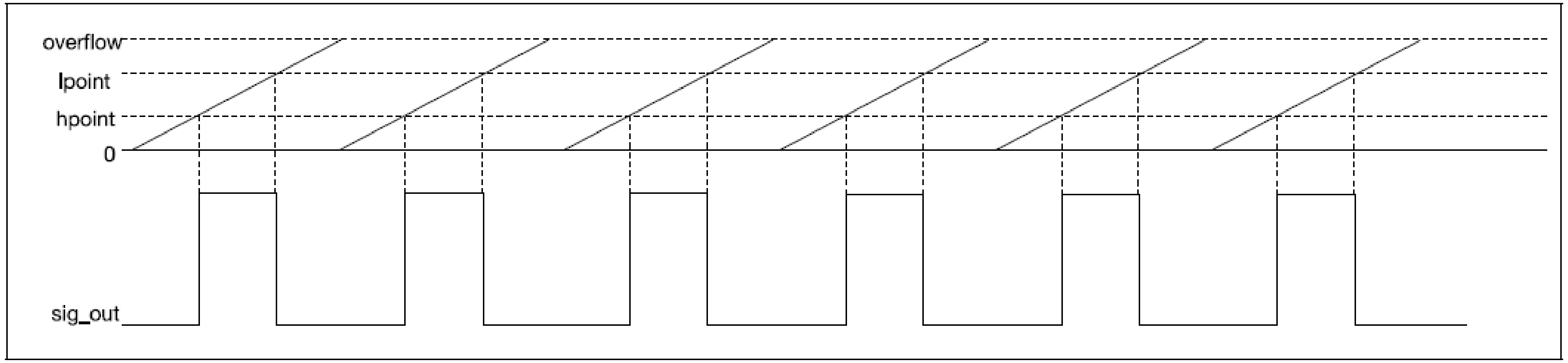
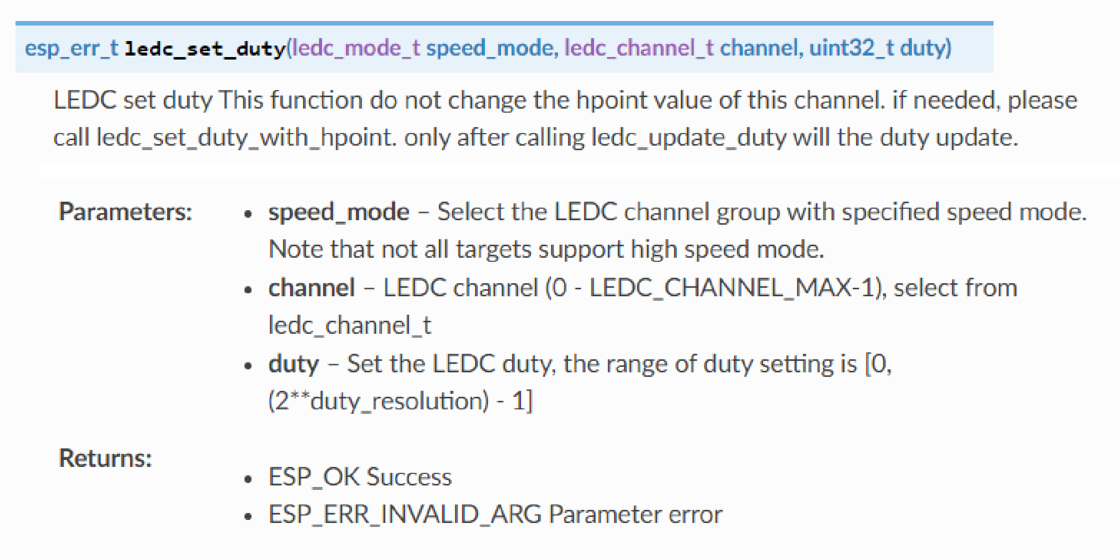


Figure 14-4. LED PWM Output Signal Diagram

Change PWM Signal



esp_err_t **ledc_set_duty**(ledc_mode_t speed_mode, ledc_channel_t channel, uint32_t duty)

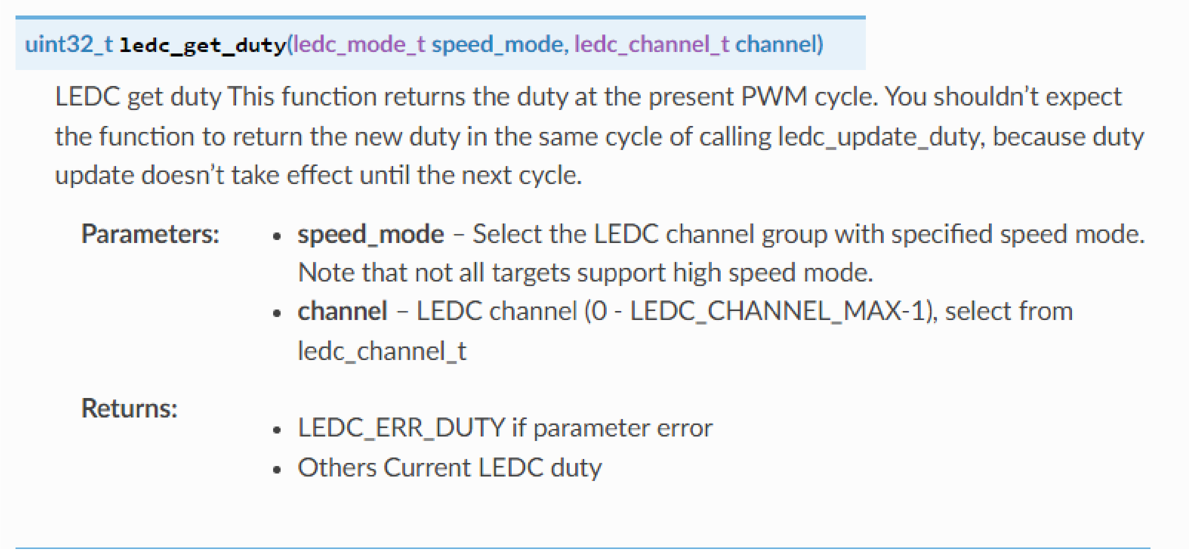
LEDC set duty This function do not change the hpoint value of this channel. if needed, please call ledc_set_duty_with_hpoint. only after calling ledc_update_duty will the duty update.

Parameters:

- **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** – LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- **duty** – Set the LEDC duty, the range of duty setting is [0, (2**duty_resolution) - 1]

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error



uint32_t **ledc_get_duty**(ledc_mode_t speed_mode, ledc_channel_t channel)

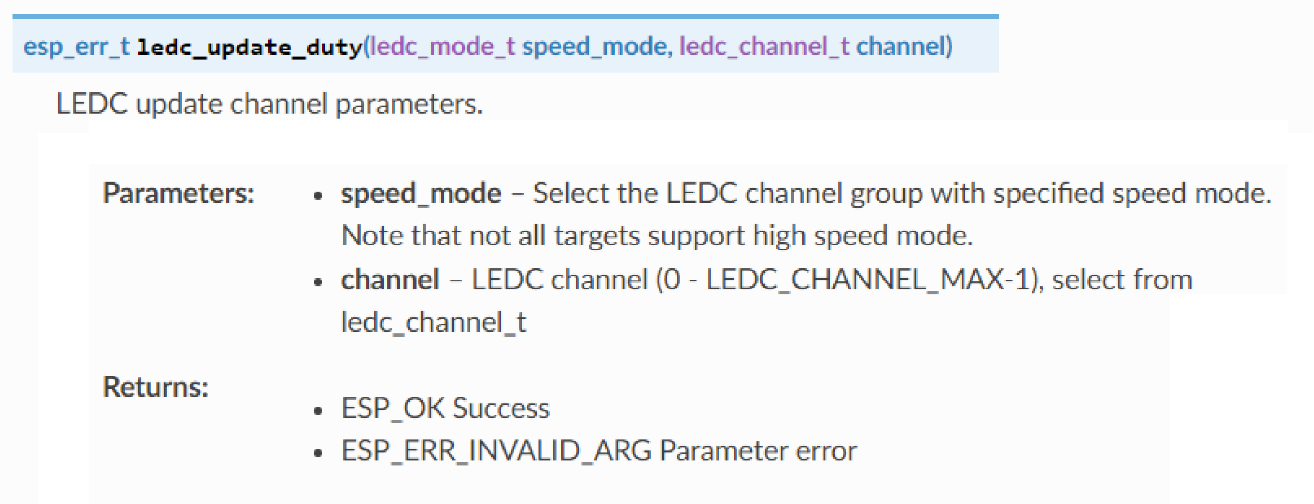
LEDC get duty This function returns the duty at the present PWM cycle. You shouldn't expect the function to return the new duty in the same cycle of calling ledc_update_duty, because duty update doesn't take effect until the next cycle.

Parameters:

- **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** – LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

Returns:

- LEDC_ERR_DUTY if parameter error
- Others Current LEDC duty



esp_err_t **ledc_update_duty**(ledc_mode_t speed_mode, ledc_channel_t channel)

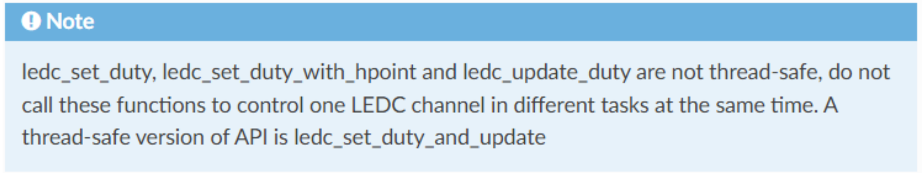
LEDC update channel parameters.

Parameters:

- **speed_mode** – Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- **channel** – LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

Returns:

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error



Note

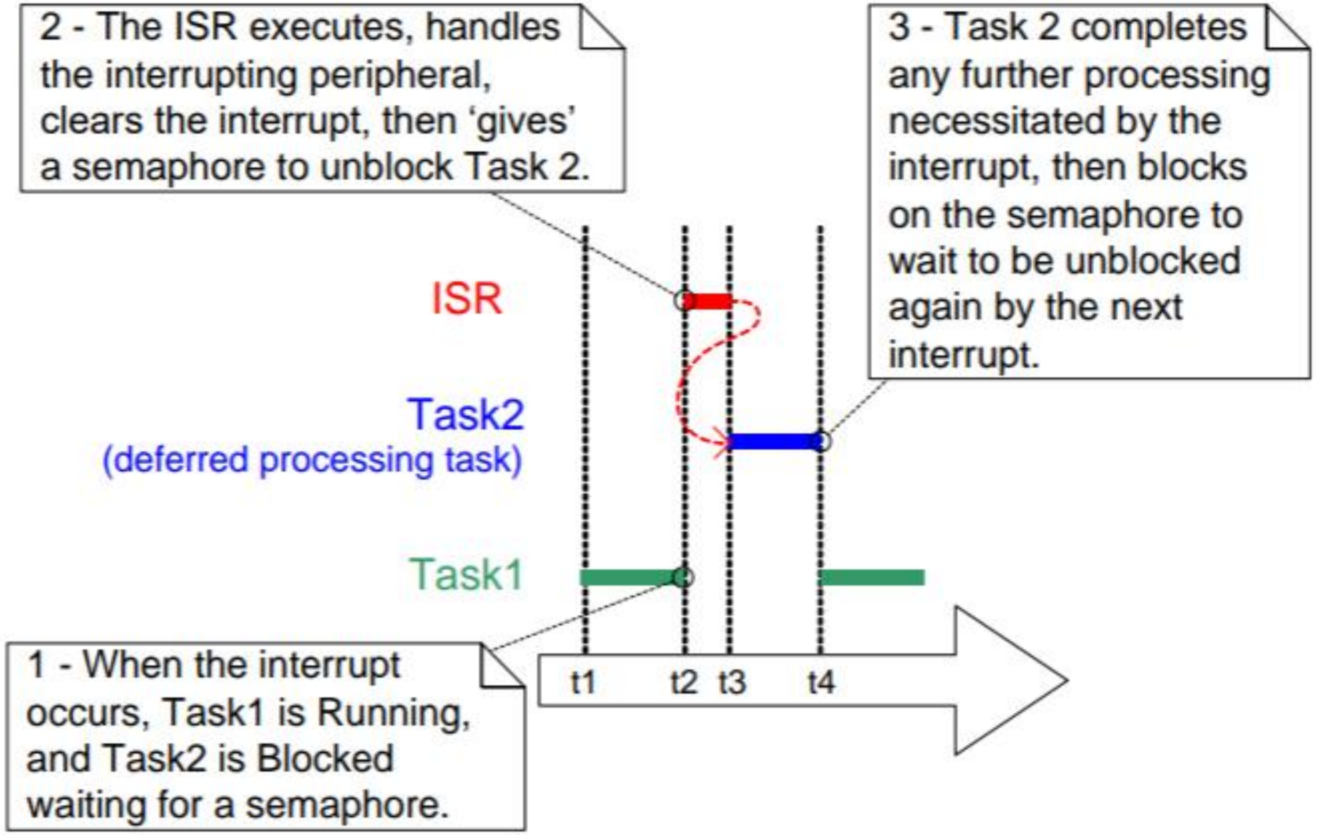
ledc_set_duty, ledc_set_duty_with_hpoint and ledc_update_duty are not thread-safe, do not call these functions to control one LEDC channel in different tasks at the same time. A thread-safe version of API is ledc_set_duty_and_update

Exemplo

```
#define LEDC_TEST_DUTY         (4000)


ledc_set_duty(speed_mode, channel, LEDC_TEST_DUTY);
ledc_update_duty(speed_mode, channel);
```

Semaphore



2 - The ISR executes, handles the interrupting peripheral, clears the interrupt, then ‘gives’ a semaphore to unblock Task 2.

3 - Task 2 completes any further processing necessitated by the interrupt, then blocks on the semaphore to wait to be unblocked again by the next interrupt.

1 - When the interrupt occurs, Task1 is Running, and Task2 is Blocked waiting for a semaphore.

Semaphore

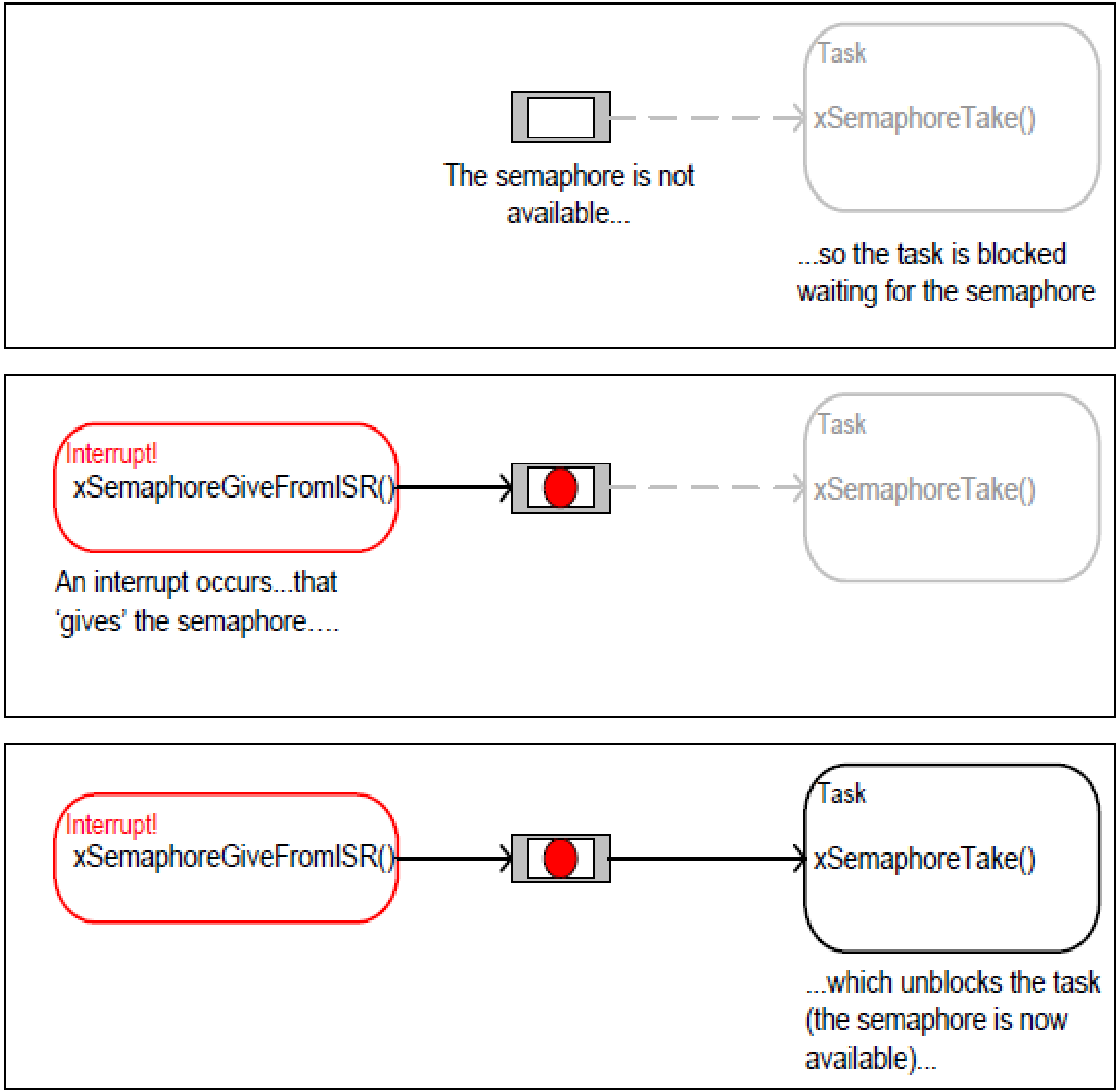
```
static SemaphoreHandle_t semaphore_pwm = NULL; // variável global

semaphore_pwm = xSemaphoreCreateBinary(); // criação do semaphore binario

xSemaphoreGive(semaphore_pwm); //Função na TASK Timer para sincronizar com a task PWM

xSemaphoreTake( semaphore_pwm, portMAX_DELAY )
```

Semaphore



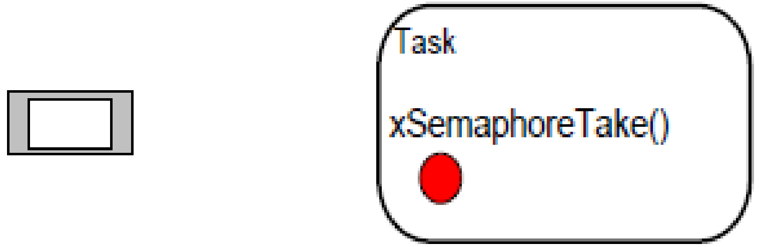
The semaphore is not available...

...so the task is blocked waiting for the semaphore

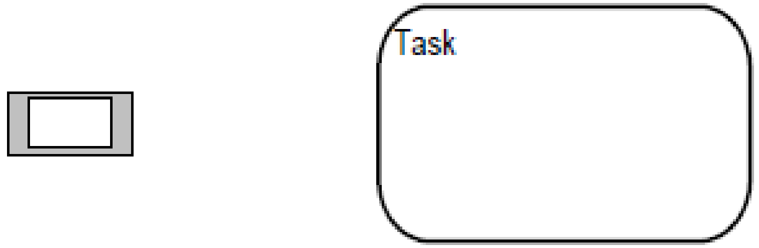
An interrupt occurs...that 'gives' the semaphore....

...which unblocks the task (the semaphore is now available)...

Semaphore



...that now successfully 'takes' the semaphore, so it is unavailable once more.



The task can now perform its action, when complete it will once again attempt to 'take' the semaphore which will cause it to re-enter the Blocked state.

Referências

- Capítulo 14 – Led PWM Controller: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

- Capítulo 6.4 - Semaphores

https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html#ledc-api-configure-timer