LED Control (LEDC)

[中文]

Introduction

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.

LEDC channels are divided into two groups of 8 channels each. One group of LEDC channels operates in high speed mode. This mode is implemented in hardware and offers automatic and glitch-free changing of the PWM duty cycle. The other group of channels operate in low speed mode, the PWM duty cycle must be changed by the driver in software. Each group of channels is also able to use different clock sources.

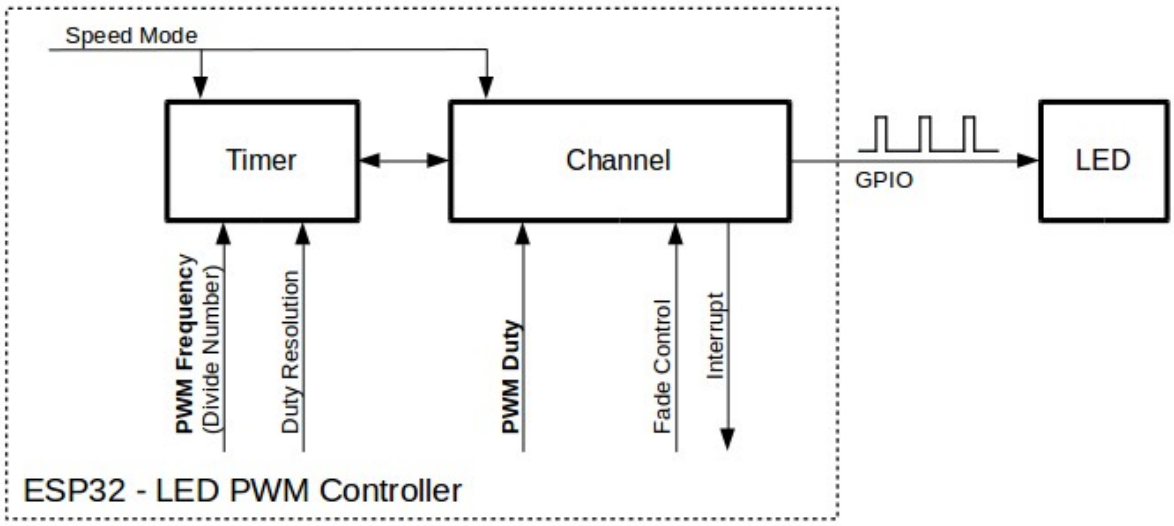
The PWM controller can automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference.

Functionality Overview

Setting up a channel of the LEDC in either high or low speed mode is done in three steps:

1. Timer Configuration by specifying the PWM signal's frequency and duty cycle resolution.
2. Channel Configuration by associating it with the timer and GPIO to output the PWM signal.
3. 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.

As an optional step, it is also possible to set up an interrupt on fade end.



Key Settings of LED PWM Controller's API

Timer Configuration

Setting the timer is done by calling the function `ledc_timer_config()` and passing the data structure `ledc_timer_config_t` that contains the following configuration settings:

- Speed mode `ledc_mode_t`
- Timer number `ledc_timer_t`
- PWM signal frequency
- Resolution of PWM duty
- Source clock `ledc_clk_cfg_t`

The frequency and the duty resolution are interdependent. The higher the PWM frequency, the lower the duty resolution which is available, and vice versa. This relationship might be important if you are planning to use this API for purposes other than changing the intensity of LEDs. For more details, see Section Supported Range of Frequency and Duty Resolutions.

The source clock can also limit the PWM frequency. The higher the source clock frequency, the higher the maximum PWM frequency can be configured.

Characteristics of ESP32 LEDC source clocks

Clock name	Clock freq	Speed mode	Clock capabilities
APB_CLK	80 MHz	High / Low	/
REF_TICK	1 MHz	High / Low	Dynamic Frequency Scaling compatible

Clock name	Clock freq	Speed mode	Clock capabilities
RTC8M_CLK	8.5 MHz	SLOW	Dynamic Frequency Scaling compatible, Ligh

Channel Configuration

When the timer is set up, configure the desired channel (one out of `ledc_channel_t`). This is done by calling the function `ledc_channel_config()`.

Similar to the timer configuration, the channel setup function should be passed a structure `ledc_channel_config_t` that contains the channel's configuration parameters.

At this point, the channel should start operating and generating the PWM signal on the selected GPIO, as configured in `ledc_channel_config_t`, with the frequency specified in the timer settings and the given duty cycle. The channel operation (signal generation) can be suspended at any time by calling the function `ledc_stop()`.

Change PWM Signal

Once the channel starts operating and generating the PWM signal with the constant duty cycle and frequency, there are a couple of ways to change this signal. When driving LEDs, primarily the duty cycle is changed to vary the light intensity.

The following two sections describe how to change the duty cycle using software and hardware fading. If required, the signal's frequency can also be changed; it is covered in Section Change PWM Frequency.

Change PWM Duty Cycle 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()`.

Another way to set the duty cycle, as well as some other channel parameters, is by calling `ledc_channel_config()` covered in Section Channel Configuration.

The range of the duty cycle values passed to functions depends on selected `duty_resolution` and should be from `0` to `(2 ** duty_resolution) - 1`. For example, if the selected duty resolution is 10, then the duty cycle values can range from 0 to 1023. This provides the resolution of ~0.1%.

Change PWM Duty Cycle using Hardware

The LEDC hardware provides the means to gradually transition from one duty cycle value to another. To use this functionality, enable fading with `ledc_fade_func_install()` and then configure it by calling one of the available fading functions:

- `ledc_set_fade_with_time()`
- `ledc_set_fade_with_step()`
- `ledc_set_fade()`

Start fading with `ledc_fade_start()`. A fade can be operated in blocking or non-blocking mode, please check `ledc_fade_mode_t` for the difference between the two available fade modes. Note that with either fade mode, the next fade or fixed-duty update will not take effect until the last fade finishes. Due to hardware limitations, there is no way to stop a fade before it reaches its target duty.

To get a notification about the completion of a fade operation, a fade end callback function can be registered for each channel by calling `ledc_cb_register()` after the fade service being installed.

If not required anymore, fading and an associated interrupt can be disabled with `ledc_fade_func_uninstall()`.

Change PWM Frequency

The LEDC API provides several ways to change the PWM frequency “on the fly”:

- Set the frequency by calling `ledc_set_freq()`. There is a corresponding function `ledc_get_freq()` to check the current frequency.
- Change the frequency and the duty resolution by calling `ledc_bind_channel_timer()` to bind some other timer to the channel.
- Change the channel’s timer by calling `ledc_channel_config()`.

More Control Over PWM

There are several lower level timer-specific functions that can be used to change PWM settings:

- `ledc_timer_set()`
- `ledc_timer_rst()`
- `ledc_timer_pause()`
- `ledc_timer_resume()`

The first two functions are called “behind the scenes” by `ledc_channel_config()` to provide a startup of a timer after it is configured.

Use Interrupts

When configuring an LEDC channel, one of the parameters selected within `ledc_channel_config_t` is `ledc_intr_type_t` which triggers an interrupt on fade completion.

For registration of a handler to address this interrupt, call `ledc_isr_register()`.

LEDC High and Low Speed Mode

High speed mode enables a glitch-free changeover of timer settings. This means that if the timer settings are modified, the changes will be applied automatically on the next overflow interrupt of the timer. In contrast, when updating the low-speed timer, the change of settings should be explicitly triggered by software. The LEDC driver handles it in the background, e.g., when `ledc_timer_config()` or `ledc_timer_set()` is called.

For additional details regarding speed modes, see *ESP32 Technical Reference Manual > LED PWM Controller (LEDC)* [PDF].

Supported Range of Frequency and Duty Resolutions

The LED PWM Controller is designed primarily to drive LEDs. It provides a large flexibility of PWM duty cycle settings. For instance, the PWM frequency of 5 kHz can have the maximum duty resolution of 13 bits. This means that the duty can be set anywhere from 0 to 100% with a resolution of ~0.012% (2 ** 13 = 8192 discrete levels of the LED intensity). Note, however, that these parameters depend on the clock signal clocking the LED PWM Controller timer which in turn clocks the channel (see timer configuration and the *ESP32 Technical Reference Manual > LED PWM Controller (LEDC)* [PDF]).

The LEDC can be used for generating signals at much higher frequencies that are sufficient enough to clock other devices, e.g., a digital camera module. In this case, the maximum available frequency is 40 MHz with duty resolution of 1 bit. This means that the duty cycle is fixed at 50% and cannot be adjusted.

The LEDC API is designed to report an error when trying to set a frequency and a duty resolution that exceed the range of LEDC’s hardware. For example, an attempt to set the frequency to 20 MHz and the duty resolution to 3 bits will result in the following error reported on a serial monitor:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try reducing freq_hz or duty_resolution. div_param=128
```

In such a situation, either the duty resolution or the frequency must be reduced. For example, setting the duty resolution to 2 will resolve this issue and will make it possible to set the duty cycle at 25% steps, i.e., at 25%, 50% or 75%.

The LEDC driver will also capture and report attempts to configure frequency / duty resolution combinations that are below the supported minimum, e.g.:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try increasing freq_hz or duty_resolution. div_param=128000000
```

The duty resolution is normally set using `ledc_timer_bit_t`. This enumeration covers the range from 10 to 15 bits. If a smaller duty resolution is required (from 10 down to 1), enter the equivalent numeric values directly.

Application Example

The LEDC change duty cycle and fading control example: peripherals/ledc/ledc_fade.

The LEDC basic example: peripherals/ledc/ledc_basic.

API Reference

Header File

- components/driver/include/driver/ledc.h

Functions

esp_err_t **ledc_channel_config**(*const* ledc_channel_config_t * *ledc_conf*)

LEDC channel configuration Configure LEDC channel with the given channel/output gpio_num/interrupt/source timer/frequency(Hz)/LEDC duty resolution.

> **Return**

- ESP_OK Success

- ESP_ERR_INVALID_ARG Parameter error

Parameters

- `ledc_conf`: Pointer of LEDC channel configure struct

---

esp_err_t **ledc_timer_config**(*const* ledc_timer_config_t * *timer_conf*)

LEDC timer configuration Configure LEDC timer with the given source timer/frequency(Hz)/duty_resolution.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error
- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.

Parameters

- `timer_conf`: Pointer of LEDC timer configure struct

---

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

LEDC update channel parameters.

Note

Call this function to activate the LEDC updated parameters. After ledc_set_duty, we need to call this function to update the settings. And the new LEDC parameters don't take effect until the next PWM cycle.

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

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

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

---

esp_err_t **ledc_set_pin**(int *gpio_num*, ledc_mode_t *speed_mode*, ledc_channel_t *ledc_channel*)

Set LEDC output gpio.

> **Return**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

> **Parameters**

- `gpio_num`: The LEDC output gpio
- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `ledc_channel`: LEDC channel (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t

---

esp_err_t **ledc_stop**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *idle_level*)

LEDC stop. Disable LEDC output, and set idle level.

> **Return**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

> **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
- `idle_level`: Set output idle level after LEDC stops.

---

esp_err_t **ledc_set_freq**(ledc_mode_t *speed_mode*, ledc_timer_t *timer_num*, uint32_t *freq_hz*)

LEDC set channel frequency (Hz)

> **Return**

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

- ESP_FAIL Can not find a proper pre-divider number base on the given frequency and the current duty_resolution.

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `timer_num`: LEDC timer index (0-3), select from ledc_timer_t
- `freq_hz`: Set the LEDC frequency

---

uint32_t **ledc_get_freq**(ledc_mode_t *speed_mode*, ledc_timer_t *timer_num*)

LEDC get channel frequency (Hz)

Return

- 0 error
- Others Current LEDC frequency

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `timer_num`: LEDC timer index (0-3), select from ledc_timer_t

---

esp_err_t **ledc_set_duty_with_hpoint**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *duty*, uint32_t *hpoint*)

LEDC set duty and hpoint value Only after calling ledc_update_duty will the duty update.

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

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

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]
- `hpoint`: Set the LEDC hpoint value(max: 0xfffff)

---

int **ledc_get_hpoint**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*)

LEDC get hpoint value, the counter value when the output is set high level.

Return

- LEDC_ERR_VAL if parameter error
- Others Current hpoint value of LEDC channel

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

---

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.

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.

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

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]

---

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.

Return

- LEDC_ERR_DUTY if parameter error
- Others Current LEDC duty

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

---

esp_err_t **ledc_set_fade**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *duty*, ledc_duty_direction_t *fade_direction*, uint32_t *step_num*, uint32_t *duty_cycle_num*, uint32_t *duty_scale*)

LEDC set gradient Set LEDC gradient, After the function calls the ledc_update_duty function, the function can take effect.

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Parameter error

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 start of the gradient duty, the range of duty setting is [0, (2**duty_resolution) - 1]
- `fade_direction`: Set the direction of the gradient
- `step_num`: Set the number of the gradient
- `duty_cycle_num`: Set how many LEDC tick each time the gradient lasts
- `duty_scale`: Set gradient change amplitude

---

esp_err_t **ledc_isr_register**(void (*_fn_)(void *), void *_arg_, int _intr_alloc_flags_, ledc_isr_handle_t * _handle_,)

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

Return

- ESP_OK Success
- ESP_ERR_INVALID_ARG Function pointer error.

Parameters

- `fn`: Interrupt handler function.
- `arg`: User-supplied argument passed to the 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.

---

esp_err_t **ledc_timer_set**(ledc_mode_t _speed_mode_, ledc_timer_t _timer_sel_, uint32_t _clock_divider_, uint32_t _duty_resolution_, ledc_clk_src_t _clk_src_)

Configure LEDC settings.

Return

- (-1) Parameter error
- Other Current LEDC duty

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `timer_sel`: Timer index (0-3), there are 4 timers in LEDC module
- `clock_divider`: Timer clock divide value, the timer clock is divided from the selected clock source
- `duty_resolution`: Resolution of duty setting in number of bits. The range of duty values is [0, (2**duty_resolution)]
- `clk_src`: Select LEDC source clock.

---

esp_err_t **ledc_timer_rst**(ledc_mode_t *speed_mode*, ledc_timer_t *timer_sel*)

Reset LEDC timer.

> **Return**
>
> - ESP_ERR_INVALID_ARG Parameter error
> - ESP_OK Success
>
> **Parameters**
>
> - `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
> - `timer_sel`: LEDC timer index (0-3), select from ledc_timer_t

---

esp_err_t **ledc_timer_pause**(ledc_mode_t *speed_mode*, ledc_timer_t *timer_sel*)

Pause LEDC timer counter.

> **Return**
>
> - ESP_ERR_INVALID_ARG Parameter error
> - ESP_OK Success
>
> **Parameters**
>
> - `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
> - `timer_sel`: LEDC timer index (0-3), select from ledc_timer_t

---

esp_err_t **ledc_timer_resume**(ledc_mode_t *speed_mode*, ledc_timer_t *timer_sel*)

Resume LEDC timer.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `timer_sel`: LEDC timer index (0-3), select from ledc_timer_t

---

esp_err_t **ledc_bind_channel_timer**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, ledc_timer_t *timer_sel*)

Bind LEDC channel with the selected timer.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel`: LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- `timer_sel`: LEDC timer index (0-3), select from ledc_timer_t

---

esp_err_t **ledc_set_fade_with_step**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *target_duty*, uint32_t *scale*, uint32_t *cycle_num*)

Set LEDC fade function.

Note

Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

Note

ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start 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_fade_step_and_start

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode. ,
- `channel`: LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- `target_duty`: Target duty of fading [0, (2**duty_resolution) - 1]
- `scale`: Controls the increase or decrease step scale.
- `cycle_num`: increase or decrease the duty every cycle_num cycles

---

esp_err_t **ledc_set_fade_with_time**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *target_duty*, int *max_fade_time_ms*)

Set LEDC fade function, with a limited time.

Note

Call ledc_fade_func_install() once before calling this function. Call ledc_fade_start() after this to start fading.

Note

ledc_set_fade_with_step, ledc_set_fade_with_time and ledc_fade_start 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_fade_step_and_start

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode. ,
- `channel`: LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- `target_duty`: Target duty of fading [0, (2**duty_resolution) - 1]
- `max_fade_time_ms`: The maximum time of the fading (ms).

---

esp_err_t **ledc_fade_func_install**(int *intr_alloc_flags*)

Install LEDC fade function. This function will occupy interrupt of LEDC module.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function already installed.

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.

---

void **ledc_fade_func_uninstall**(void)

Uninstall LEDC fade function.

---

esp_err_t **ledc_fade_start**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, ledc_fade_mode_t *fade_mode*)

Start LEDC fading.

Note

Call ledc_fade_func_install() once before calling this function. Call this API right after ledc_set_fade_with_time or ledc_set_fade_with_step before to start fading.

Note

Starting fade operation with this API is not thread-safe, use with care.

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_ERR_INVALID_ARG Parameter error.

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel`: LEDC channel number
- `fade_mode`: Whether to block until fading done. See ledc_types.h ledc_fade_mode_t for more info. Note that this function will not return until fading to the target duty if LEDC_FADE_WAIT_DONE mode is selected.

---

esp_err_t **ledc_set_duty_and_update**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *duty*, uint32_t *hpoint*)

A thread-safe API to set duty for LEDC channel and return when duty updated.

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

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]
- `hpoint`: Set the LEDC hpoint value(max: 0xfffff)

---

esp_err_t **ledc_set_fade_time_and_start**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *target_duty*, uint32_t *max_fade_time_ms*, ledc_fade_mode_t *fade_mode*)

A thread-safe API to set and start LEDC fade function, with a limited time.

Note

Call ledc_fade_func_install() once, before calling this function.

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel`: LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- `target_duty`: Target duty of fading [0, (2**duty_resolution) - 1]
- `max_fade_time_ms`: The maximum time of the fading (ms).
- `fade_mode`: choose blocking or non-blocking mode

---

esp_err_t **ledc_set_fade_step_and_start**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, uint32_t *target_duty*, uint32_t *scale*, uint32_t *cycle_num*, ledc_fade_mode_t *fade_mode*)

A thread-safe API to set and start LEDC fade function.

Note

Call ledc_fade_func_install() once before calling this function.

Note

For ESP32, hardware does not support any duty change while a fade operation is running in progress on that channel. Other duty operations will have to wait until the fade operation has finished.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel`: LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- `target_duty`: Target duty of fading [0, (2**duty_resolution) - 1]
- `scale`: Controls the increase or decrease step scale.
- `cycle_num`: increase or decrease the duty every cycle_num cycles
- `fade_mode`: choose blocking or non-blocking mode

---

esp_err_t **ledc_cb_register**(ledc_mode_t *speed_mode*, ledc_channel_t *channel*, ledc_cbs_t * *cbs*, void **user_arg*)

LEDC callback registration function.

Note

The callback is called from an ISR, it must never attempt to block, and any FreeRTOS API called must be ISR capable.

Return

- ESP_ERR_INVALID_ARG Parameter error
- ESP_OK Success
- ESP_ERR_INVALID_STATE Fade function not installed.
- ESP_FAIL Fade function init error

Parameters

- `speed_mode`: Select the LEDC channel group with specified speed mode. Note that not all targets support high speed mode.
- `channel`: LEDC channel index (0 - LEDC_CHANNEL_MAX-1), select from ledc_channel_t
- `cbs`: Group of LEDC callback functions

- `user_arg`: user registered data for the callback function

Structures

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** : 1

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

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

LEDC flags

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_bit_t **bit_num**

Deprecated in ESP-IDF 3.0. This is an alias to 'duty_resolution' for backward compatibility with ESP-IDF 2.1

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. For low speed channels and high speed channels, you can specify the source clock using LEDC_USE_REF_TICK, LEDC_USE_APB_CLK or LEDC_AUTO_CLK. For low speed channels, you can also specify the source clock using LEDC_USE_RTC8M_CLK, in this case, all low speed channel's source clock must be RTC8M_CLK

struct **ledc_cb_param_t**

LEDC callback parameter.

Public Members

ledc_cb_event_t **event**

Event name

uint32_t **speed_mode**

Speed mode of the LEDC channel group

uint32_t **channel**

LEDC channel (0 - LEDC_CHANNEL_MAX-1)

uint32_t **duty**

LEDC current duty of the channel, the range of duty is [0, (2**duty_resolution) - 1]

struct **ledc_cbs_t**

Group of supported LEDC callbacks.

Note

The callbacks are all running under ISR environment

Public Members

ledc_cb_t **fade_cb**

LEDC fade_end callback function

Macros

`LEDC_APB_CLK_HZ`

`LEDC_REF_CLK_HZ`

`LEDC_ERR_DUTY`

`LEDC_ERR_VAL`

Type Definitions

typedef intr_handle_t **ledc_isr_handle_t**

typedef bool (***ledc_cb_t**)(*const* ledc_cb_param_t *param, void *user_arg)

Type of LEDC event callback.

Parameters

- `param`: LEDC callback parameter
- `user_arg`: User registered data

Enumerations

enum `ledc_cb_event_t`

LEDC callback event type.

Values:

`LEDC_FADE_END_EVT`

LEDC fade end event

Header File

- components/hal/include/hal/ledc_types.h

Enumerations

enum `ledc_mode_t`

Values:

`LEDC_HIGH_SPEED_MODE` = 0

LEDC high speed speed_mode

`LEDC_LOW_SPEED_MODE`

LEDC low speed speed_mode

`LEDC_SPEED_MODE_MAX`

LEDC speed limit

enum `ledc_intr_type_t`

Values:

`LEDC_INTR_DISABLE` = 0

Disable LEDC interrupt

`LEDC_INTR_FADE_END`

Enable LEDC interrupt

`LEDC_INTR_MAX`

enum `ledc_duty_direction_t`

Values:

`LEDC_DUTY_DIR_DECREASE` = 0

LEDC duty decrease direction

`LEDC_DUTY_DIR_INCREASE` = 1

LEDC duty increase direction

`LEDC_DUTY_DIR_MAX`

enum `ledc_slow_clk_sel_t`

Values:

`LEDC_SLOW_CLK_RTC8M` = 0

LEDC low speed timer clock source is 8MHz RTC clock

`LEDC_SLOW_CLK_APB`

LEDC low speed timer clock source is 80MHz APB clock

enum `ledc_clk_cfg_t`

In theory, the following enumeration shall be placed in LEDC driver's header. However, as the next enumeration, `ledc_clk_src_t`, makes the use of some of these values and to avoid mutual inclusion of the headers, we must define it here.

Values:

`LEDC_AUTO_CLK` = 0

The driver will automatically select the source clock(REF_TICK or APB) based on the giving resolution and duty parameter when init the timer

`LEDC_USE_APB_CLK`

LEDC timer select APB clock as source clock

`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

`LEDC_USE_REF_TICK`

LEDC timer select REF_TICK clock as source clock

enum `ledc_clk_src_t`

Values:

`LEDC_REF_TICK` = **LEDC_USE_REF_TICK**

LEDC timer clock divided from reference tick (1Mhz)

`LEDC_APB_CLK` = **LEDC_USE_APB_CLK**

LEDC timer clock divided from APB clock (80Mhz)

`LEDC_SCLK` = **LEDC_USE_APB_CLK**

Selecting this value for LEDC_TICK_SEL_TIMER let the hardware take its source clock from LEDC_APB_CLK_SEL

enum `ledc_timer_t`

Values:

`LEDC_TIMER_0` = **0**

LEDC timer 0

`LEDC_TIMER_1`

LEDC timer 1

`LEDC_TIMER_2`

LEDC timer 2

`LEDC_TIMER_3`

LEDC timer 3

`LEDC_TIMER_MAX`

enum `ledc_channel_t`

Values:

`LEDC_CHANNEL_0` = **0**

LEDC channel 0

`LEDC_CHANNEL_1`

LEDC channel 1

`LEDC_CHANNEL_2`

LEDC channel 2

`LEDC_CHANNEL_3`

LEDC channel 3

`LEDC_CHANNEL_4`

LEDC channel 4

`LEDC_CHANNEL_5`

LEDC channel 5

`LEDC_CHANNEL_6`

LEDC channel 6

`LEDC_CHANNEL_7`

LEDC channel 7

`LEDC_CHANNEL_MAX`

enum `ledc_timer_bit_t`

Values:

`LEDC_TIMER_1_BIT` **= 1**

LEDC PWM duty resolution of 1 bits

`LEDC_TIMER_2_BIT`

LEDC PWM duty resolution of 2 bits

`LEDC_TIMER_3_BIT`

LEDC PWM duty resolution of 3 bits

`LEDC_TIMER_4_BIT`

LEDC PWM duty resolution of 4 bits

`LEDC_TIMER_5_BIT`

LEDC PWM duty resolution of 5 bits

`LEDC_TIMER_6_BIT`

LEDC PWM duty resolution of 6 bits

`LEDC_TIMER_7_BIT`

LEDC PWM duty resolution of 7 bits

`LEDC_TIMER_8_BIT`

LEDC PWM duty resolution of 8 bits

`LEDC_TIMER_9_BIT`

LEDC PWM duty resolution of 9 bits

`LEDC_TIMER_10_BIT`

LEDC PWM duty resolution of 10 bits

`LEDC_TIMER_11_BIT`

LEDC PWM duty resolution of 11 bits

`LEDC_TIMER_12_BIT`

LEDC PWM duty resolution of 12 bits

`LEDC_TIMER_13_BIT`

LEDC PWM duty resolution of 13 bits

`LEDC_TIMER_14_BIT`

LEDC PWM duty resolution of 14 bits

`LEDC_TIMER_15_BIT`

LEDC PWM duty resolution of 15 bits

`LEDC_TIMER_16_BIT`

LEDC PWM duty resolution of 16 bits

`LEDC_TIMER_17_BIT`

LEDC PWM duty resolution of 17 bits

`LEDC_TIMER_18_BIT`

LEDC PWM duty resolution of 18 bits

`LEDC_TIMER_19_BIT`

LEDC PWM duty resolution of 19 bits

`LEDC_TIMER_20_BIT`

LEDC PWM duty resolution of 20 bits

`LEDC_TIMER_BIT_MAX`

enum `ledc_fade_mode_t`

Values:

`LEDC_FADE_NO_WAIT` **= 0**

LEDC fade function will return immediately

`LEDC_FADE_WAIT_DONE`

LEDC fade function will block until fading to the target duty

`LEDC_FADE_MAX`