ESP8266

PWM Interface

Version 1.1

Espressif Systems IOT Team

http://bbs.espressif.com

Copyright © 2015

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The WiFi Alliance Member Logo is a trademark of the WiFi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2015 Espressif Systems. All rights reserved.

Table of Contents

1. Preambles

1.1. Features

PWM (Pulse Width Modulation) can be implemented on Frame Rate Control 1 (FRC1) via software programming, achieving multi-channelled PWM with the same frequency but different duty ratio. It can be used to control devices such as colour lights, buzzer, and electric machines, etc.

> **Note:**
>
> FRC1 is a 23-bit hardware timer.

Features of PWM are listed below:

- Apply NMI (Non Maskable Interrupt) to interrupt, more precise.
- Can be extended to 8 channels of PWM signal.
- Resolution ratio higher than 14 bit, the minimum resolution can reach 45 ns.
- Configuration can be completed by call interface functions, without set the register.

1.2. Implementation

An optimized software algorithm provided by ESP8266 system enable the transmission of multi-channel PWM signals via GPIO (General Purpose Input Output) interface by way of mounting NMI on FRC1 timer.

The clock of PWM is provided by high-speed system clock, the frequency speed of which can reach as high as 80MHz. Through pre-frequency divider, the clock source can be divided into 16 separated frequencies, the input clock frequency of which is 5MHz. PWM can issue coarse tuning timing via FRC1, which combined with fine tuning issued by the high-speed system clock, can improve the resolution to as much as 45 ns.

> **Note:**
>
> The highest priority level of interrupt owned by NMI ensures the precision of PWM output waveform.

1.3. Configuration

- In timing interrupt, to exist the program as soon as possible, timing parameters of the next period of PWM waveform can be loaded when PWM period started.

- After the duty ratios of every channel have been configured, the system will call function `pwm_start()` to calculate timing cycle. Before that, parameters of all current channels will be stored and protected by the system, calculation completion bits will be cleared, too. When PWM period comes, parameters stored by the system will be invoked.
- When PWM period is discontinued new parameters will be applied, and flags should be set when the calculation of timing cycle is completed, so that cycles between different colour shade with each new frame and simulate an intermediate shade, achieving higher quality colour. The control of RGB colour lights is an good example of PWM control.
- The specific GPIO used can be configured in `user_light.h`. In our demo SDK, 5 channels of PWM is applied, however, it can be extended to 16 channels. Details on how to extend the channels of PWM is explained in Chapter 3. The minimum resolution can reach 45 ns at 1KHz refresh rate, while the minimum duty ratio can reach 1/22222.

1.4. Parameter Specification

- Minimum resolution: 45 ns (approximately speaking, the PWM input clock frequency is 22.72 MHz): >14 bit PWM @ 1 KHz
- PWM period: 1000 μs (1 KHz) ~ 10000 μs (100 Hz)

2. Details on pwm.h

2.1. Sample Codes

```
#ifndef __PWM_H__
#define __PWM_H__
#define PWM_CHANNEL_NUM_MAX 8              //8 channels PWM at most
struct pwm_single_param {                  //define the structure of a single PWM parameter
    uint16 gpio_set;                       //GPIO needs to be set
    uint16 gpio_clear;                     //GPIO needs to be cleared
    uint32 h_time;                         //time needs to be written into FRC1_LOAD
};
struct pwm_param {                        //define the structure of PWM parameter
    Uint32 period;                        //PWM period
    Uint32 freq;                          //PWM frequency
    uint32 duty[PWM_CHANNEL_NUM_MAX];     //PWM duty ratio
};
void pwm_init(uint32 period, uint32 *duty,uint32 pwm_channel_num,uint32
(*pin_info_list)[3]);
void pwm_start(void);
void pwm_set_duty(uint32 duty, uint8 channel);
uint32 pwm_get_duty(uint8 channel);
void pwm_set_freq(uint32 period);
uint32 pwm_get_freq(void);
```

2.2. Interface Specifications

2.2.1. pwm_init

Function Name	pwm_init
Definition	PWM initialization.
Sample code	`pwm_init (uint32 freq, uint32 *duty, uint32 pwm_channel_num,uint32 (*pin_info_list)[3]);`
Description	PWM GPIO, initializing parameters and timer.

Parameters	• uint32 freq: PWM period.
	• uint32 *duty: duty ratio of each PWM channel.
	• uint32 pwm_channel_num: the number of PWM channels.
	• uint32 (*pin_info_list)[3]: This parameter, which is made up of a n x 3 array pointer, defines the GPIO hardware parameter of each PWM channel. Registers of GPIO, pin multiplexing of IO, and the serial number of each GPIO are defined in the array. Take the initialization of a 3-channel PWM for example:
	`uint32 io_info[][3] =`
	`{{PWM_0_OUT_IO_MUX,PWM_0_OUT_IO_FUNC,PWM_0_OUT_IO_NUM},`
	`{PWM_1_OUT_IO_MUX,PWM_1_OUT_IO_FUNC,PWM_1_OUT_IO_NUM},`
	`{PWM_2_OUT_IO_MUX,PWM_2_OUT_IO_FUNC,PWM_2_OUT_IO_NUM}};`
	`pwm_init(light_param.pwm_period,light_param.pwm_duty,3,io_info);`

Call	Call the function when the system is been initialized. Currently the function can be called only once.
Returned Value	Null

2.2.2. pwm_set_period

Function Name	pwm_set_period
Definition	Set PWM period.
Sample code	`pwm_set_period (uint32 period)`
Description	Set PWM period, unit: μs. For example, PWM period at 1KHz is1000 μs.
Parameters	uint32 period: PWM period.
Call	Call `pwm_start()` after the parameters has been set.
Returned Value	Null

2.2.3. pwm_set_duty

Function Name	pwm_set_duty
Definition	Set the duty ratio of PWM signal at a certain channel
Sample code	`pwm_set_duty (uint32 duty, uint8 channel)`
Description	Set PWM duty ratio. Set the time period of PWM signal when the voltage is high. The value of duty ratio change with PWM period. PWM duty ratio can reach period*1000/45 at most. For example, the range of duty ratio is between 0 and 22222 at 1kHz refresh rate.
Parameters	• uint32 duty: set the time parameter when the voltage is high. Duty ratio is (duty*45)/ (period*1000). • uint8 channel: PWM channel that needs to be set at present. This parameter is defined in PWM_CHANNEL.
Call	Call `pwm_start()` after the parameters has been set.
Returned Value	Null

2.2.4. pwm_get_period

Function Name	pwm_get_period
Description	Get the current PWM period.
Sample code	`pwm_get_period (void)`
Description	None.
Returned Value	PWM period, unit: μs.

2.2.5. pwm_get_duty

Function Name	pwm_get_duty
Description	Get the duty ratio of current PWM signal at a certain channel.
Sample code	`pwm_get_duty (uint8 channel)`
Parameter	uint8 channel: get the current PWM channel. This parameter is defined in PWM_CHANNEL.
Call	Call `pwm_start()` after the parameters has been set.
Returned Value	Duty ratio of a certain PWM channel, the value returned is (duty*45)/ (period*1000).

2.2.6. pwm_start

Function Name	pwm_start
Description	Update PWM parameters.
Sample code	`pwm_start (void)`
Parameter	None.
Call	Call `pwm_start()` when PWM related parameters have been set.
Returned Value	Null.

3. Custom Channels

Users can customise PWM channels. Below is a detailed instruction on how to set GPIO4 as the forth channel for PWM signal output.

1. Modify initialization parameters.

```
uint32 io_info[][3]={
    {PWM_0_OUT_IO_MUX,PWM_0_OUT_IO_FUNC,PWM_0_OUT_IO_NUM},
    {PWM_1_OUT_IO_MUX,PWM_1_OUT_IO_FUNC,PWM_1_OUT_IO_NUM},
    {PWM_2_OUT_IO_MUX,PWM_2_OUT_IO_FUNC,PWM_2_OUT_IO_NUM},
    {PWM_3_OUT_IO_MUX,PWM_3_OUT_IO_FUNC,PWM_3_OUT_IO_NUM},
    {PWM_4_OUT_IO_MUX,PWM_4_OUT_IO_FUNC,PWM_4_OUT_IO_NUM},
    };
pwm_init(light_param.pwm_period, light_param.pwm_duty, PWM_CHANNEL,io_info);
```

2. Modify user_light.h.

```
#define PWM_0_OUT_IO_MUX PERIPHS_IO_MUX_MTDI_U
#define PWM_0_OUT_IO_NUM 12
#define PWM_0_OUT_IO_FUNC FUNC_GPIO12
#define PWM_1_OUT_IO_MUX PERIPHS_IO_MUX_MTDO_U
#define PWM_1_OUT_IO_NUM 15
#define PWM_1_OUT_IO_FUNC FUNC_GPIO15
#define PWM_2_OUT_IO_MUX PERIPHS_IO_MUX_MTCK_U
#define PWM_2_OUT_IO_NUM 13
#define PWM_2_OUT_IO_FUN CFUNC_GPIO13
#define PWM_3_OUT_IO_MUX PERIPHS_IO_MUX_GPIO4_U
#define PWM_3_OUT_IO_NUM 4
#define PWM_3_OUT_IO_FUNC FUNC_GPIO4
#define PWM_4_OUT_IO_MUX PERIPHS_IO_MUX_GPIO5_U
#define PWM_4_OUT_IO_NUM 5
#define PWM_4_OUT_IO_FUNC FUNC_GPIO5
#define PWM_CHANNEL 5
```