# Application Note

## Document No.: AN_1105

## APM32F407 TMR1 Modulated Breathing LED Application Note

## Version: V1.0

# 1 Introduction

This application note provides users with a guide for configuring and applying timers on the APM32F407 series, including application methods and implementation code.

In this article, the APM32F407 microcontroller TMR1 is used to implement the breathing LED. The APM32F407 microcontroller has 14 built-in timers, including 2 advanced timers (TMR1, TMR8), 2 basic timers (TMR6, TMR7) and 10 general timers. Timers are independent from each other, and synchronization and cascading can be achieved between timers according to the internal design of the microcontroller. The most basic function of the timer is to provide a time base for the microcontroller, and can generate DMA requests through configuration.

# Contents

# 2 Timer Introduction

In a microcontroller system, a timer is an indispensable peripheral. It is used to perform tasks, measure time, generate timing signals and control time-related operations within a specific time interval. Timers are very important for control systems, embedded systems and real-time systems. Their main uses include delay, time measurement, periodic interrupts, PWM (Pulse Width Modulation) generation, counting and time base, etc.

A timer usually consists of a counter register and some control registers. The counter register is used to save the current value of the counter, while the control register is used to configure the timer's working mode, clock source, interrupt enable, etc. The counter increments based on pulses from the clock source, and when the counter reaches a predetermined value, an interrupt can be triggered or other operations performed.

Timers are key components in embedded systems. They provide precise time measurement and control capabilities and can implement a variety of time-related functions, which are very important for control and automation applications.

## 2.1 Timebase unit

The time base unit of the timer usually includes 3 registers:

•16-bit counter register

•16-bit auto-reload register

•16-bit prescaler register

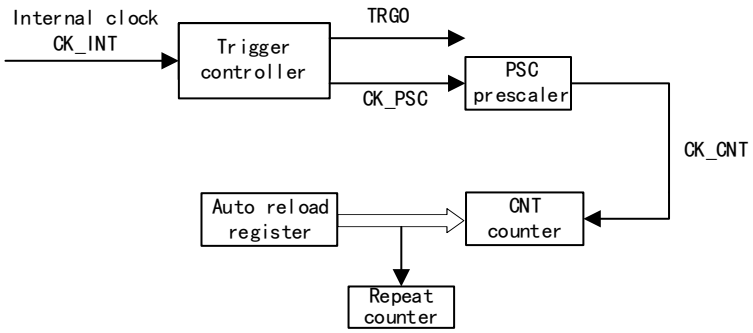


Figure 1 Time base unit block diagram

All timer clock frequency allocations are automatically set by hardware in two different situations:

(1) If the corresponding APB prescaler coefficient is 1, the clock frequency of the timer is consistent with the frequency of the APB bus.

(2) If the corresponding APB prescaler coefficient is not 1, the clock frequency of the timer

is set to 2 times the frequency of the APB bus connected to it.

In the APM32F407 microcontroller, the maximum system clock frequency is 168MHz, the prescaler coefficient of APB1 is 4, and the prescaler coefficient of APB2 is 2. According to the APM32F407 system framework diagram, TMR1/8/9/10/11 is connected to APB2 bus, and TMR2/3/4/5/6/712/13/14 is connected to APB2 bus, so the clock frequencies of the timers are:

TMR1/8/9/10/11:

$$\mathrm{CK\_INT} = \frac{168\mathrm{MHz}}{2} * 2 = 168\mathrm{MHz} \qquad \text{(Equation 2-1)}$$

TMR2/3/4/5/6/712/13/14:

$$\mathrm{CK\_INT} = \frac{168\mathrm{MHz}}{4} * 2 = 84\mathrm{MHz} \qquad \text{(Equation 2-2)}$$

The 16-bit count register stores the count value of the counter. The 16-bit auto-reload register stores the auto-reload value. When the auto-reload value is empty, the counter does not count. The 16-bit prescaler register stores the prescaler value, and the clock frequency of the counter (CK_CNT) can be calculated based on the prescaler value.

$$\mathrm{CK\_CNT} = \frac{\mathrm{CK\_PSC}}{\mathrm{PSC}+1} \qquad \text{(Equation 2-3)}$$

When the slave mode controller is disabled, the clock source CK_PSC of the prescaler is driven by the internal clock CK_INT, then:

$$\mathrm{CK\_CNT} = \frac{\mathrm{CK\_INT}}{\mathrm{PSC}+1} \qquad \text{(Equation 2-4)}$$

In addition, the advanced timer has a unique register: the 8-bit repeat count register. The repeat count register is a unique register of the advanced timer of APM32F407. This register stores the repeat count value (REPCNT). Its main function is to decrease the repeat count value (REPCNT) by 1 when the timer overflows or underflows. Only when the value of the repeat count register is 0 and the timer overflows or underflows, will an update event be generated.

## 2.2 Counting mode

The counter is an important part of the timer, and the counting mode of the counter has an important impact on the function of the timer. In APM32F407, the counting mode of the counter is different depending on the timer. There is only one counting mode for timer TMR6/7/9/10/11/12/13/14: count-up. For timer TMR1/2/3/4/5/8, there are 3 counting modes: count-up mode, count-down mode, and center-aligned mode.

Count-up mode: When the counter is enabled, it starts counting from 0. Each time a CK_CNT

pulse is received, the counter value increases by 1 until the counter value increases from 0 to equal the auto-reload value. At this moment, the timer generates an update event. Then the counter starts counting upward again from 0, and so on.

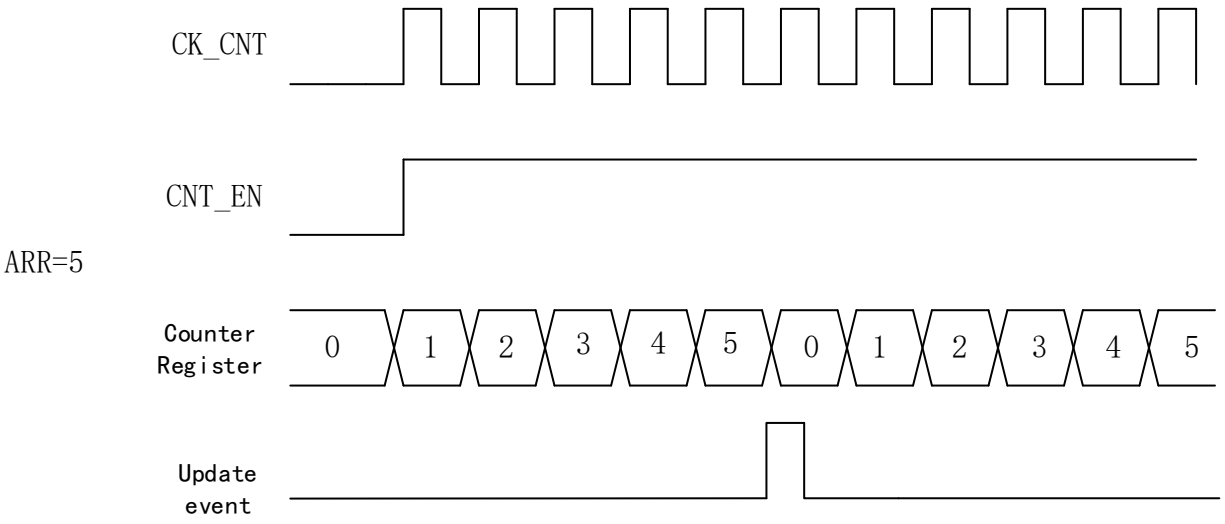


Figure 2 Count-up mode timing diagram

Count-down mode: When the counter is enabled, it starts counting from the automatic reload value. Each time a CK_CNT pulse is received, the counter value decreases by 1 until the counter value decreases from the automatic reload value to 0. At this moment, the timer generates an update event. Then the counter starts counting downwards again from the auto-reload value, and so on.

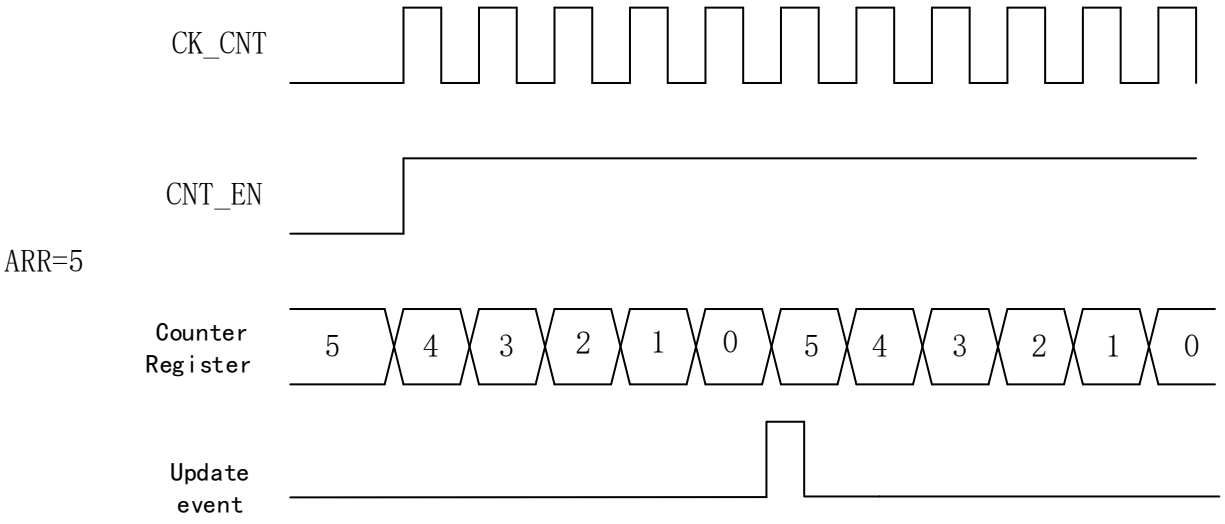


Figure 3 Count-down mode timing diagram

Center-aligned mode: When the counter is enabled, it starts counting from 0. Each time a CK_CNT pulse is received, the counter value increases by 1. When the counter value is equal to (auto-reload value-1), the timer generates an update event. When the counter value

decreases to 0, it increases from 0 to be equal to the auto-reload value. Then the counter counts down from the auto-reload value. Each time a CK_CNT pulse is received, the counter value decreases by 1 until the counter value decreases to 1. At this moment, the timer generates an update event and the counter starts counting upwards from 0, and so on.

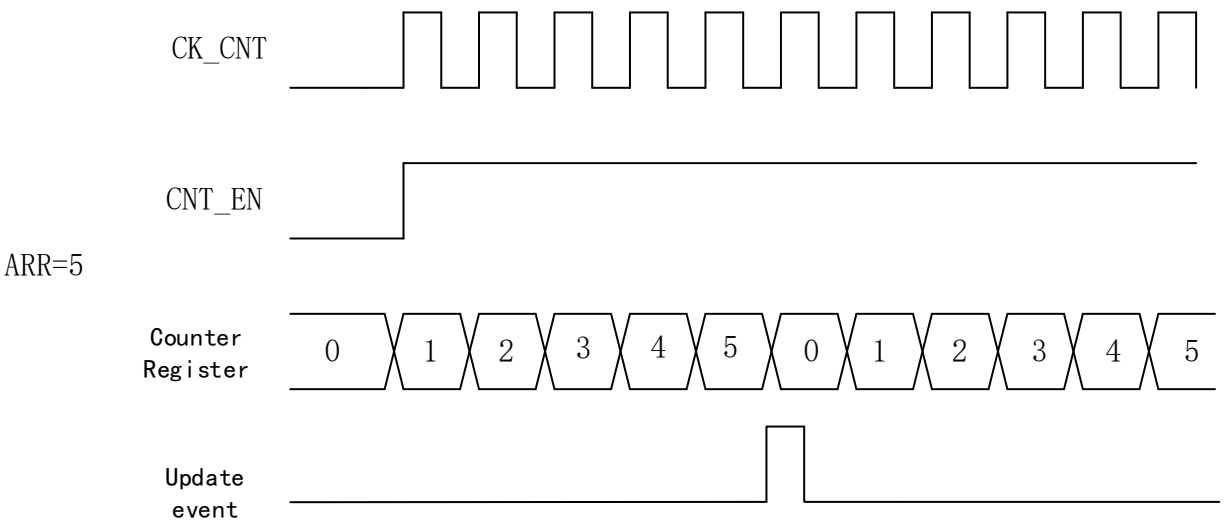


Figure 4 Center-aligned mode timing diagram

In the APM32F407 microcontroller, the center-aligned mode of the timer allows the duty cycle of the PWM signal to vary between the rising and falling edges of the counter, which allows the PWM signal to undergo smooth duty cycle changes throughout the counter cycle. This mode provides precise duty cycle control while also helping to reduce electromagnetic interference and noise. Different center-aligned modes have different timings when the output comparison interrupt flag bit of the output channel is set to 1. The timer counting mode can be controlled by operating the CNTDIR bit and CAMSEL bit of control register 1 (TMRx_CTRL1).

Center-aligned mode 1: The counter alternately counts up and down. If and only when the counter counts down, the output comparison interrupt flag bit of the output channel is set to 1.

Center-aligned mode 2: The counter alternately counts up and down. If and only when the counter counts up, the output comparison interrupt flag bit of the output channel is set to 1.

Center-aligned mode 3: The counter alternately counts up and down. When the counter counts up or down, the output comparison interrupt flag bit of the output channel is set to 1.

# 3 Timer Output Comparison

## 3.1 Introduction

The output comparison function of the timer is: the timer module is compared with external inputs or other internal events, and performs corresponding operations when the comparison conditions are met. This function is usually used for applications such as generating precise time delays, generating PWM signals, and controlling external devices.

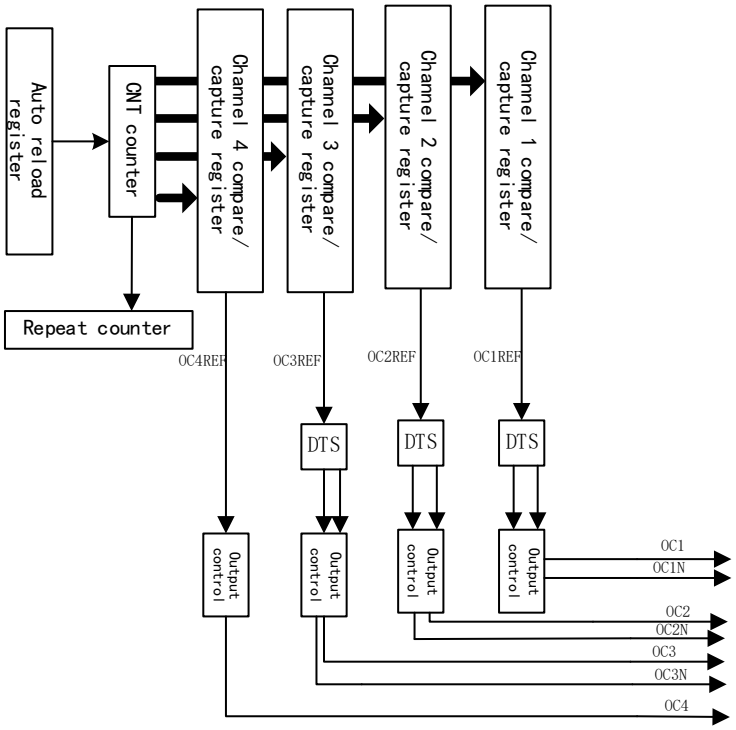


Figure 5 Timer output comparison structure block diagram

APM32F407 microcontroller timer usually provides 4 independent comparison channels, and each comparison channel has an associated comparison register and comparison value. The output comparison function of the timer supports multiple modes. The mode of the output comparison channel is controlled by configuring the OCxMOD of TMRx_CCMx, including the following common ones:

(1) Output comparison mode: The output of the timer is associated with the comparison result. When the value of the counter is equal to the value in the compare register, an output event can be triggered, such as causing the output comparison signal to flip or forcing the output comparison signal to a high (low) level. When the value of the counter is equal to the value in the comparison register, an interrupt request can be triggered, allowing the corresponding operation to be performed in the interrupt service routine.

(2) Forced output mode: Regardless of the comparison result of the output comparison value and count value, the output comparison signal is forced to high (low) level. When the

counter value is compared with the value in the comparison register, it will still be executed and an interrupt request can be triggered.

(3) PWM mode: The output comparison function is usually used to generate PWM signals. By setting the comparison value of the comparison channel to an appropriate value, the duty cycle and frequency of the PWM signal can be controlled.

The dead-time duration of the complementary output channel is set by configuring the DTS bit of TMRx_BDT. The dead-time complementary function is used to ensure that two opposite switching devices will not be turned on at the same time to avoid short circuit and damage. Dead-time is the period of time between two switching devices where both are off. Complementary means that when one switching device is on, the other is off, and vice versa. In the APM32F407, the dead-time complementary function is usually used in conjunction with PWM generation and output comparison channels. The output comparison channel is used to configure the dead-time to ensure that there is sufficient time interval between the two switching devices to avoid conduction conflicts.

## 3.2 PWM output mode

PWM output mode is a type of output comparison. The timer outputs an adjustable pulse signal to the outside. The frequency of the pulse signal is determined by the automatic reload value of the timer, and the duty cycle of the pulse signal is determined by the comparison value. In APM32F407, there are 2 PWM output modes: PWM 1 and PWM 2.

PWM 1 means that when the count value (CNT) is less than the comparison value (CCx), a valid level is output, or else an invalid level is output.

PWM 2 means that when the count value (CNT) is less than the comparison value (CCx), an invalid level is output, or else a valid level is output. The polarity of the active level is controlled by configuring the CCxPOL bit of TMRx_CCEN. Depending on the PWM mode and counting mode, the PWM signal generated is also different.

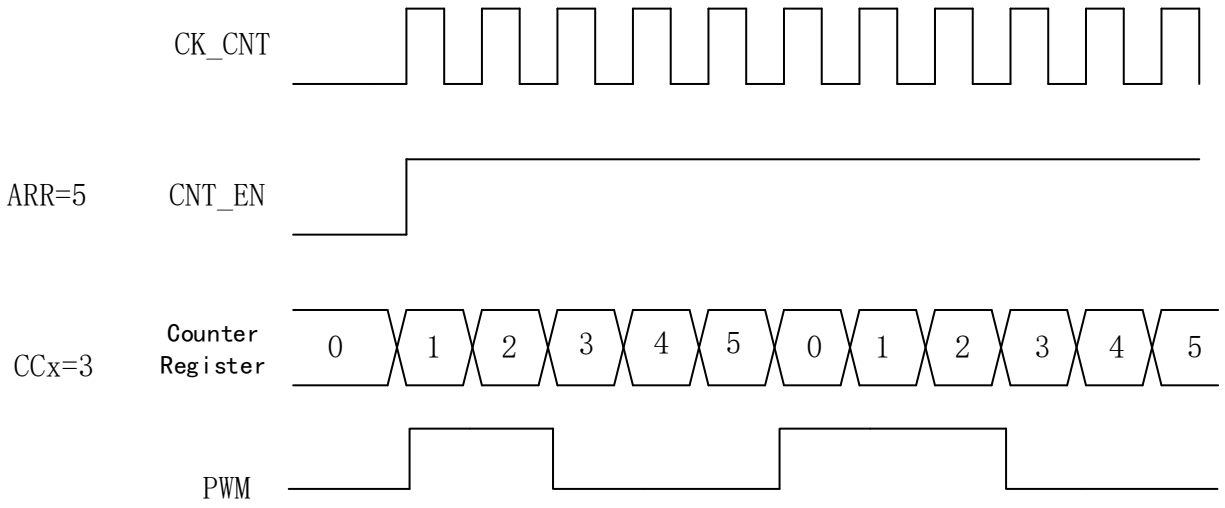


Figure 6 Timing diagram of PWM1 count-up mode

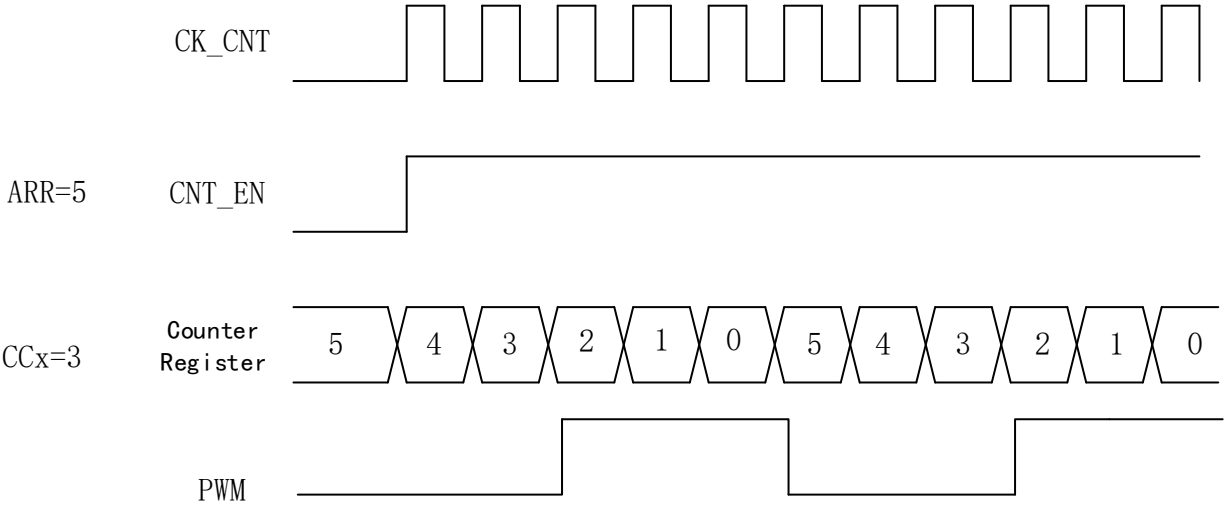


Figure 7 Timing diagram of PWM1 count-down mode

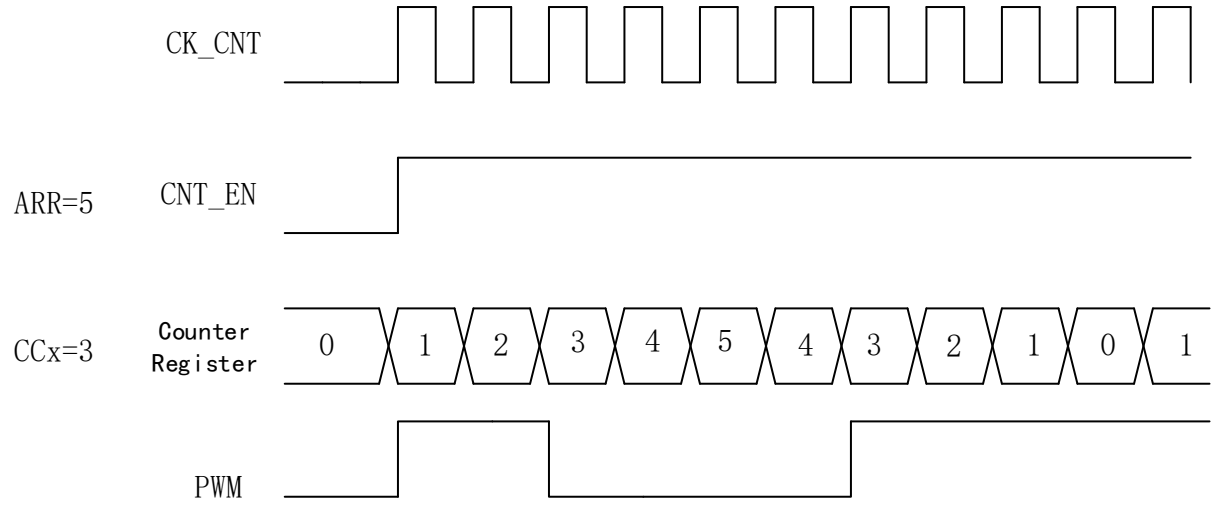

Figure 8 Timing diagram of PWM1 center-aligned mode

PWM output mode controls the output by changing the duty cycle of the pulse (the ratio of high level to total period). As the duty cycle increases, the average output voltage or current also increases, and vice versa. Advantages: Simple and easy to implement, requiring less hardware resources. It is suitable for most basic applications such as motor control and LED brightness adjustment, but the lower resolution restricts fine control.

# 4 APM32F407 TMR1 Modulated Breathing LED Routine

Breathing LED is a common LED lighting mode. Its effect is similar to the breathing rhythm of living organisms, that is, it gradually becomes brighter and then gradually darker, forming a periodic breathing effect. The breathing LED can produce a soft gradient light effect. It is a simple and effective way to improve the appearance and user experience of a product, especially practical in situations where status needs to be indicated or users' attention needs to be attracted.

The brightness of the LED light is determined by the voltage. The greater the voltage across the LED, the greater the brightness is. Using PWM output to control the brightness of the LED to implement a breathing LED is actually to control the analog voltage at both ends of the LED light. At the beginning, set the frequency and duty cycle of PWM, and change the high level time in a cycle by increasing or decreasing the duty cycle, that is, changing the analog voltage across the LED in this cycle. The core of the breathing LED effect is to gradually change the duty cycle of PWM within a certain period of time to control the brightness of the LED.

Initially, the PWM duty cycle gradually increases, making the LED gradually brighter. When the LED reaches maximum brightness, the PWM duty cycle begins to gradually reduce, making the LED gradually dim. Once the brightness decreases to the set minimum value, the operation reverses and the brightness gradually increases again. This creates a cycle. The effect of the breathing LED can be changed by adjusting the PWM frequency, maximum brightness, minimum brightness, breathing cycle and other parameters.

## 4.1 Hardware connection

This routine requires the use of TMR1 channel 1 and LED lights. The MINI Board has connected the TMR1 channel, and relevant IO can be used according to the design requirements. The schematic diagram of the LED lights used is as follows:

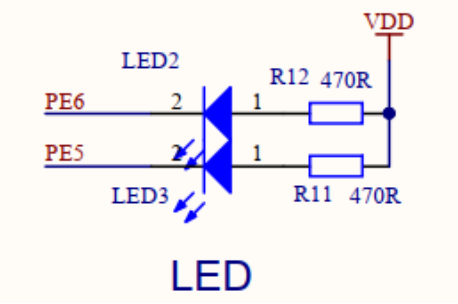


Figure 9 LED schematic diagram

It can be seen from the schematic diagram that when the PE5 and PE6 segment terminals are at low level, the LED light is lit, and when at high level, the LED is turned off. Connect the IO (PA8) corresponding to channel 1 of TMR1 to the corresponding IO of the LED light to complete the hardware connection.

## 4.2 Software design

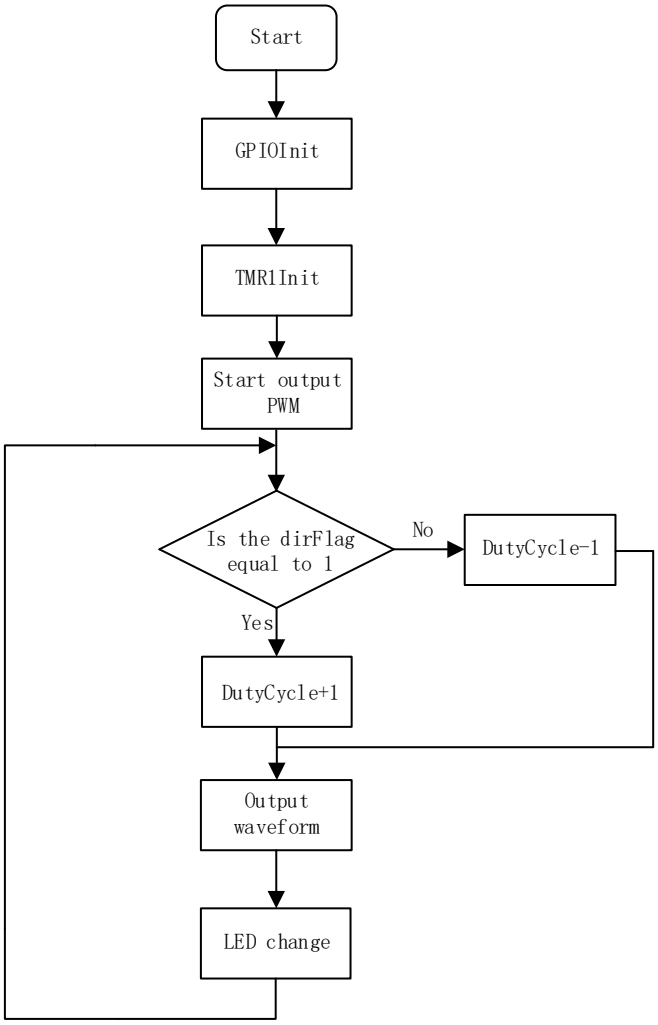


Figure 10 Programming flow chart

Initialize the IO of TMR1 channel 1 and the IO of the LED light, initialize the timer, configure the basic parameters of the timer and channel 1 parameters, and pulse modulate the output PWM wave in the interrupt.

### 4.2.1 Pin configuration

Configure the GPIO used by TMR1 channel 1, and configure GPIOA_PIN_8 to port multiplexing mode. Configure the GPIO of the LED light to push-pull input mode. The reference code is as follows:

```
void GPIO_Init()
{
    GPIO_Config_T GPIO_ConfigStruct;
```

```
    GPIO_ConfigPinAF(GPIOA, GPIO_PIN_SOURCE_8, GPIO_AF_TMR1);

    /* Config PA8 for output PWM */
    GPIO_ConfigStruct.pin = GPIO_PIN_8;
    GPIO_ConfigStruct.mode = GPIO_MODE_AF;
    GPIO_ConfigStruct.otype = GPIO_OTYPE_PP;
    GPIO_ConfigStruct.speed = GPIO_SPEED_100MHz;
    GPIO_Config(GPIOA, &GPIO_ConfigStruct);

    GPIO_ConfigStruct.pin = GPIO_PIN_5;
    GPIO_ConfigStruct.mode = GPIO_MODE_IN;
    GPIO_ConfigStruct.otype = GPIO_OTYPE_PP;
    GPIO_ConfigStruct.speed = GPIO_SPEED_50MHz;
    GPIO_Config(GPIOE, &GPIO_ConfigStruct);
}
```

### 4.2.2 Timer Initialization

#### 4.2.2.1 TMR1 Structure

TMR_BaseConfig_T structure is defined in the document of APM32F4xx_dmc.h. The specific definitions are as follows:

```
typedef struct
{
    TMR_COUNTER_MODE_T   countMode;
    TMR_CLOCK_DIV_T      clockDivision;
    uint32_t             period;
    uint16_t             division;
    uint8_t              repetitionCounter;
} TMR_BaseConfig_T;
```

The meaning of parameters in the structure:

countMode: The counting modes of the timer include count-up, count-down, center-aligned 1, center-aligned 2, and center-aligned 3.

clockDivision: Clock division coefficient selection; the clock used for dead time can be adjusted by setting this bit.

period: Automatic reload value, available in any value from 0x0000 to 0xFFFFFFFF.

division: Prescaler value, available in any value from 0x0000 to 0xFFFF.

repetitionCounter: Repetition count value; only TMR1 and TMR8 can set this bit, available in any value from 0x00 to 0xFF.

#### 4.2.2.2 Timer Configuration

After turning on the GPIO and TMR1 clocks, initialize the GPIO, and then configure the timer time base and channel 1. The specific code is as follows:

```
void TMR_Init()
{
    TMR_BaseConfig_T TMR_TimeBaseStruct;
    TMR_OCConfig_T OCcongigStruct;

    /* config TMR1 */
    TMR_TimeBaseStruct.clockDivision = TMR_CLOCK_DIV_1;
    TMR_TimeBaseStruct.countMode = TMR_COUNTER_MODE_UP;
    TMR_TimeBaseStruct.division = DIV;
    TMR_TimeBaseStruct.period = 999;

    TMR_ConfigTimeBase(TMR1, &TMR_TimeBaseStruct);
```

Configure the counting mode of TMR1 to the count-up, the dead time to $t_{DTS}=t_{CK\_INT}$, the auto-reload value to 999, the prescaler value to 167, and the repeat count value to 0, then the frequency of TMR1 is:

$$f = \frac{CK\_PSC}{PSC+1} * \frac{1}{999+1} = \frac{168MHz}{(167+1)*1000} = 1kHz \quad \text{(Equation 4-1)}$$

In order to make TMR1 output PWM wave, channel 1 needs to be configured. The specific code is as follows:

```
    OCcongigStruct.idleState = TMR_OC_IDLE_STATE_RESET;
    OCcongigStruct.mode = TMR_OC_MODE_PWM2;
    OCcongigStruct.nIdleState = TMR_OC_NIDLE_STATE_RESET;
    OCcongigStruct.nPolarity = TMR_OC_NPOLARITY_HIGH;
    OCcongigStruct.outputNState = TMR_OC_NSTATE_DISABLE;
    OCcongigStruct.outputState = TMR_OC_STATE_ENABLE;
    OCcongigStruct.polarity = TMR_OC_POLARITY_HIGH;
```

```
    OCcongigStruct.pulse = 1;
    TMR_ConfigOC1(TMR1, &OCcongigStruct);

    TMR_Enable(TMR1);
    TMR_EnablePWMOutputs(TMR1);
```

Configure the PWM mode of channel 1 to PWM 2, the channel 1 and the idle point of the complementary channel output of channel 1 to low level, the channel 1 and the complementary channel output polarity of channel 1 to high level, enable channel 1 output, and set channel 1 comparison value to 1.

Enable the TMR1 update interrupt and set the response priority of the TMR1 update interrupt:

```
    TMR_EnableInterrupt(TMR1, TMR1_UP_TMR10_IRQn);
    NVIC_EnableIRQRequest(TMR1_UP_TMR10_IRQn, 0, 0);
}
```

#### 4.2.2.3 Breathing Function

Every time the timer generates an update event, an interrupt service function is executed. In the interrupt service function, the breathing function is called. In the breathing function, the interrupt flag bit is first cleared, and the duty cycle is determined to increase or decrease according to the direction flag bit. When the duty cycle increases to the maximum or decreases to the minimum, change the value of the direction flag. The specific code is as follows:

```
void Breath()
{
    TMR_ClearIntFlag(TMR1, TMR_INT_UPDATE);

    if(dirFlag)
    {
        dutyCycle++;
        TMR_ConfigCompare1(TMR1, dutyCycle);
        if(dutyCycle == 999)
        {
            dirFlag = 0;
        }
    }
    else
```

```
    {
        dutyCycle--;
        TMR_ConfigCompare1(TMR1, dutyCycle);
        if(dutyCycle == 0)
        {
            dirFlag = 1;
        }
    }
}
```

# 5 Summary

In the APM32F407 microcontrol system, the output comparison function of timer 1 is used to modulate the duty cycle, so that the duty cycle of the output PWM wave increases or decreases, and the high-level time gradually increases or decreases. Connect the output PWM signal to the LED light, and the voltage at both ends of the LED light gradually increases or decreases, thereby adjusting the brightness of the LED light and realizing breathing LED.

# 6 Revision history

Table 1 Revision history

| Date | Version | Revision History |
|---|---|---|
| September 27, 2023 | 1.0 | New edition |

# Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Please read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “™” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in thesales order or

the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTIRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDE THE DOCUMENT "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES).

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.