Linux LEDC
开发指南

版本号: 1.1
发布日期: 2025.03.28

版本历史

版本号	日期	制/修订人	内容描述
1.0	2024.2.21	XAA0311	添加初版
1.1	2025.03.28	AWA2231	根据最新模块刷新文档

目　录

1 前言 1
1.1 文档简介 1
1.2 目标读者 1
1.3 适用范围 1
1.4 相关术语介绍 1

2 模块介绍 2
2.1 模块功能介绍 2
2.2 结构框图介绍 2
2.3 LED 数据输入码型 4
2.4 模块配置介绍 4
2.4.1 设备树配置 4
2.4.2 kernel menuconfig 配置 7
2.5 源代码结构介绍 7

3 模块接口说明 8
3.1 内部接口 8
3.1.1 sunxi_ledc_set_led_brightness 8
3.1.2 sunxi_ledc_trans_data 8
3.1.3 sunxi_ledc_set_time 8
3.1.4 sunxi_ledc_set_output_mode 9
3.1.5 sunxi_ledc_set_cpu_mode 9
3.1.6 sunxi_ledc_set_dma_mode 9
3.1.7 sunxi_ledc_set_length 9
3.2 外部接口 10
3.2.1 brightness 调节说明 10
3.2.2 led trigger 使用说明 10

4 调试方法 11
4.1 debugfs 接口 11

5 用户层使用 demo 12

ALLWINNER®

1 前言

1.1 文档简介

介绍 LEDC 模块的使用方法，方便开发人员使用。

1.2 目标读者

LEDC 模块的驱动开发/维护人员。

1.3 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-5.10 及以上	{sdk}/bsp/drivers/ledc/ledc-sunxi.c

1.4 相关术语介绍

表 1-2: 术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台。
LED	Light Emitting Diode, 发光二极管。
LEDC	Light Emitting Diode Controller，发光二极管控制器。
sdk	指 tina 的根目录

2 模块介绍

2.1 模块功能介绍

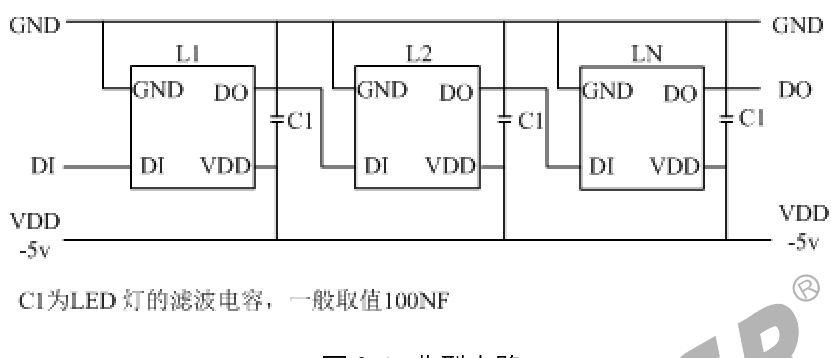


图 2-1: 典型电路

LED 典型电路如上图所示，其中 DI 表示控制数据输入脚，DO 表示控制数据输出脚。DI 端接受从控制器传输过来的数据，每个 LED 内部的数据锁存器会存储 24bit (分别对应 R,G,B 三种颜色) 数据，剩余的数据经过内部整形处理电路整形放大后通过 DO 端口开始转发输出给下一个级联的 LED 。因此，每经过一个 LED ，数据减少 24bit。

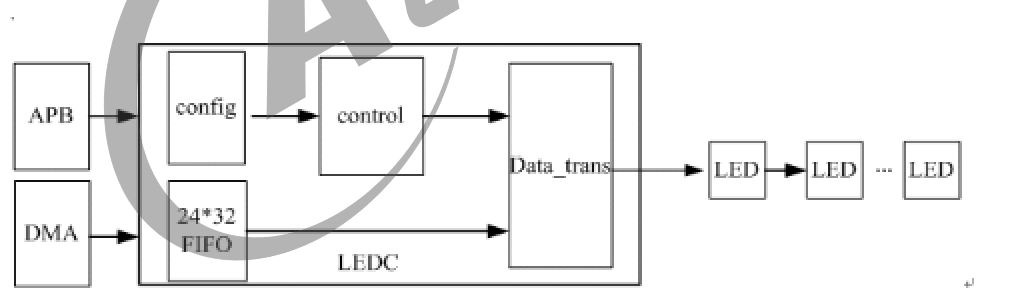


图 2-2: 硬件结构

CPU 通过 APB 总线操作 LEDC 寄存器来控制 LEDC 。当 CPU 配置好 LEDC 的相关寄存器之后，且通过 CPU 或 DMA 将 R、G、B 数据从 DRAM 搬到 LEDC FIFO 中，启动 LEDC 之后就可以向外部的 LED 发送数据了。

2.2 结构框图介绍

根据 LED 子系统的设计，只能对一个 led_classdev 进行 brightness 调节而不能做颜色调节，因此在设计 LEDC 驱动时，我们将一个 LED 抽象为 3 个 led_classdev，分别对应 R、G 和 B，这 3 个

版权所有 © 珠海全志科技股份有限公司。保留一切权利

led_classdev 称为一个 led_classdev group。因此，led_classdev group 的数量应和 LED 的数量保持一致。

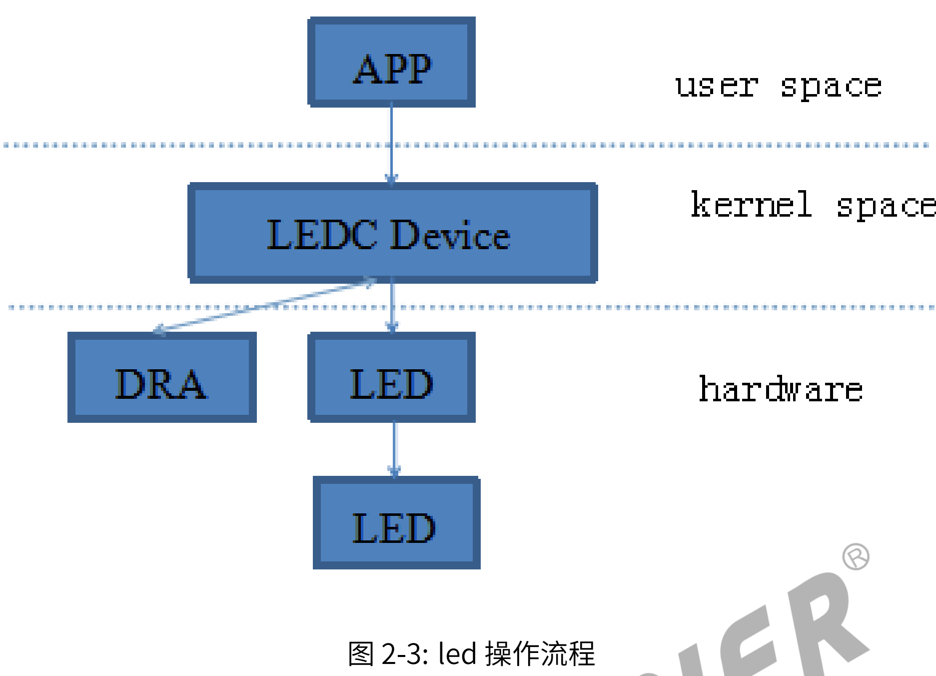


图 2-3: led 操作流程

应用通过 LED 子系统透给应用层的节点来对 LED 进行操作，应用根据需要调节 LED 的 R、G、B 颜色对应 led_classdev 设备的亮度，从而达到调节 LED 颜色的目的。LEDC 驱动在初始化时会根据实际的 LED 的数量申请一块相应大小的内存，这块内存用来保存所有 LED 的数据，当应用层通过 LED 子系统透给上层的节点调节 LED 的亮度时，LEDC 驱动就会检查当前的 led_classdev 设备对应的 LED 的编号和颜色分量，然后将应用层传下来的亮度数据写入到该内存中，然后再通过 CPU 或 DMA 将内存中的 LED 数据写入到 LEDC FIFO 中，启动 LEDC 之后即可输出数据信号用来控制 LED 的颜色。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.3 LED 数据输入码型

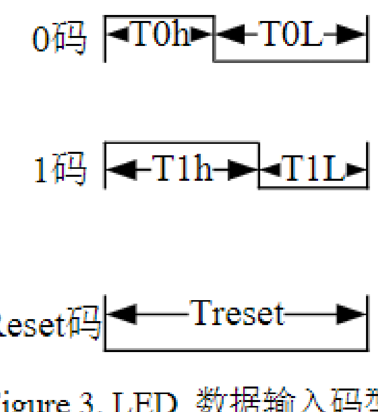


图 2-4: 数据输入码型

LED 数据传输时间：

T0H	0 码，高电平时间	220ns-380ns
T0L	0 码，低电平时间	580ns-1.6us
T1H	1 码，高电平时间	580ns-1.6us
T1L	1 码，低电平时间	220ns-420ns
RESET	帧单位，低电平时间	280ns 以上

2.4 模块配置介绍

2.4.1 设备树配置

在 soc 级的 dtsi 文件中提炼了内存基地址、中断控制、时钟等共性信息，是该类芯片所有平台的模块配置，soc 级的 dtsi 文件的路径为：sdk/bsp/configs/{linux-ver}/{CHIP}.dtsi(CHIP 为研发代号, 如 sun50iw10p1 等) 下面为 ledc 配置：

```
ledc: ledc@2008000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "allwinner,sunxi-leds";
    reg = <0x0 0x02008000 0x0 0x400>;
    interrupts = <GIC_SPI 28 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&ccu CLK_LEDC>, <&ccu CLK_BUS_LEDC>;
    clock-names = "clk_ledc", "clk_cpuapb";
    resets = <&ccu RST_BUS_LEDC>;
    reset-names = "ledc_reset";
    dmas = <&dma 42>;
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    dma-names = "tx";
    status = "disabled";
};
```

LEDC 相关的时序参数，引脚参数需要在 sdk/device/config/chips/{IC}/config/{BOARD}/{linux-ver}/board.dts 里面配置相关参数：

```
&pio {
...
  ledc_pins_active: ledc@0 {
        pins = "PE5";
        function = "ledc";
        drive-strength = <20>;
  };

  ledc_pins_sleep: ledc@1 {
        pins = "PE5";
        function = "gpio_in";
  };
...
}

&ledc {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&ledc_pins_active>;
    pinctrl-1 = <&ledc_pins_sleep>;
    led_count = <34>;
    output_mode = "GRB";
    reset_ns = <84>;            //设置LED的reset时间
    t1h_ns = <800>;             //1码高电平时间
    t1l_ns = <320>;             //1码低电平时间
    t0h_ns = <300>;             //0码高电平时间
    t0l_ns = <800>;             //0码低电平时间
    wait_time0_ns = <84>;       //相邻两个LED数据之间等待的时间
    wait_time1_ns = <84>;       //相邻两帧数据之间等待的时间
    wait_data_time_ns = <600000>;   //LEDC内部FIFO等待数据的时间容忍度
    status = "disabled";
};
```

dts 的配置含义如下所示：

- pinctrl-names: 用于表示 0 和 1 的 pinctrl 哪个是默认和休眠状态。
- pinctrl-0: 引脚配置，这里是默认使用的时候配置。
- pinctrl-1: 同上，这里是休眠时的配置。
- led_count: LED 灯的数目，根据硬件配置, 当数目超过 32 个使用 dma 模式进行数据搬运。
- output_mode: 设置 LED 灯输出模式用来控制 LED 灯的颜色，通过该节点可设置和读取当前输出的模式，输出模式有 GRB、GBR、RGB、RBG、BGR 和 BRG 等
- reset_ns: 通过该节点可设置和读取 LED 的 reset 时间控制 (当 LEDC 一个数据包传输完毕，会自动插入设定的 reset 时间的低电平)，默认约 300ns，范围为 80ns-327us。调节范围 42ns*(N+1) 单位 cyle(24Mhz) N 范围：1~1FFF
- t1h_ns：通过该节点可设置和读取 1 码高电平时间，范围为 80ns-2560ns。
- t1l_ns：通过该节点可设置和读取 1 码低电平时间，范围为 80ns-1280ns。
- t0h_ns：通过该节点可设置和读取 0 码高电平时间，范围为 80ns-1280ns。
- t0l_ns：通过该节点可设置和读取 1 码低电平时间，范围为 80ns-2560ns。

- wait_time0_ns：通过该节点可设置和读取相邻两个 LED 数据之间等待的时间，范围为 80ns-10us。
- wait_time1_ns：通过该节点可设置和读取相邻两帧数据之间等待的时间，范围为 80ns-85s。
- wait_data_time_ns：通过该节点可设置和读取 LEDC 内部 FIFO 等待数据的时间容忍度，当超过这个设定的时间 LEDC 将发 waitdata_timeout_int 中断，此时属于异常情况，软件需将 LEDC 进行 soft reset，wait_data_time_ns 范围为 80ns-655us。调节范围 42ns*(N+1) 单位 cyle(24Mhz) N 范围：1~1FFF
- status: 设备状态。通常，如果想要使用一款新的 LEDC 灯，需要确认上述全部配置项都配置正确，比如说引脚配置以及 LED 灯的参数配置（包括 01 码高低电平时间、reset 时间以及 wait 时间），全部配置正确才能成功点亮。

注意：

1. 设置的时间必须在所说明的时间范围内，否则不会做任何操作。
2. 最终设置寄存器之后得到的时间均为 42ns 的整数倍，若通过节点设置的时间不遵循 42ns 的整数倍，则实际所设置的时间为小于该值的最大能够被 42ns 整除的数。例如通过 reset_ns 设置 90ns，则设置成功之后的 LED reset 时间为 84ns。
3. 上述参数中，参数 reset_ns 概念往往容易理解出现偏差：进入 reset 后，如果使能了 wait time1，那么 reset 完成后，会进入到 wait time1 这个状态，之后等 wait time1 时间到了再开始传输数据；要是 ledc 灯带不需要 wait time1 可以不使能，那 reset 完成了就直接下一笔数据。

dts 的配置属性应用场景说明如下

led_count

- 作用：定义连接的 LED 灯数量。
- 应用场景：大型 LED 显示屏或装饰照明系统，其中可能需要控制大量的 LED 灯。需要高效的数据传输机制来管理大量 LED 灯的状态变化。

output_mode

- 作用：设置 LED 灯的颜色输出顺序。不同的硬件可能采用不同的颜色顺序，如 GRB、GBR 等。
- 应用场景：在多色 LED 应用中确保正确的颜色显示。例如，在 RGB LED 灯带中正确地映射红、绿、蓝三色信号。适应不同制造商的 LED 模块，它们可能使用不同的颜色通道排列。

reset_ns

- 作用：在 LEDC 完成一个数据包的传输后，会插入一段设定长度的低电平时间（reset）
- 应用场景：确保 LED 驱动芯片能够正确识别每个数据包的结束和下一个数据包的开始，特别是在连续发送多个数据包时避免混淆。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

t1h_ns t1l_ns t0h_ns t0l_ns

- 作用：分别设置 1 码和 0 码的高低电平时间，用于定义如何编码二进制信息到 LED 上。
- 应用场景：实现自定义的协议或优化现有协议的性能。

wait_time0_ns、wait_time1_ns

- 作用：分别为相邻两个 LED 数据之间和两帧数据之间的等待时间。
- 应用场景：控制 LED 灯显示内容的变化速度和平滑度

wait_data_time_ns

- 作用：设置 LEDC 内部 FIFO 等待数据的最大容忍时间，超出该时间将触发中断，提示软件执行软复位。
- 应用场景：在实时性要求较高的场合，比如舞台灯光控制或动态广告牌，保证数据流的持续性和稳定性。

2.4.2 kernel menuconfig 配置

在根目录中执行./build.sh menuconfig，选择 Allwinner BSP 选项进入下一级配置，如下图所示：

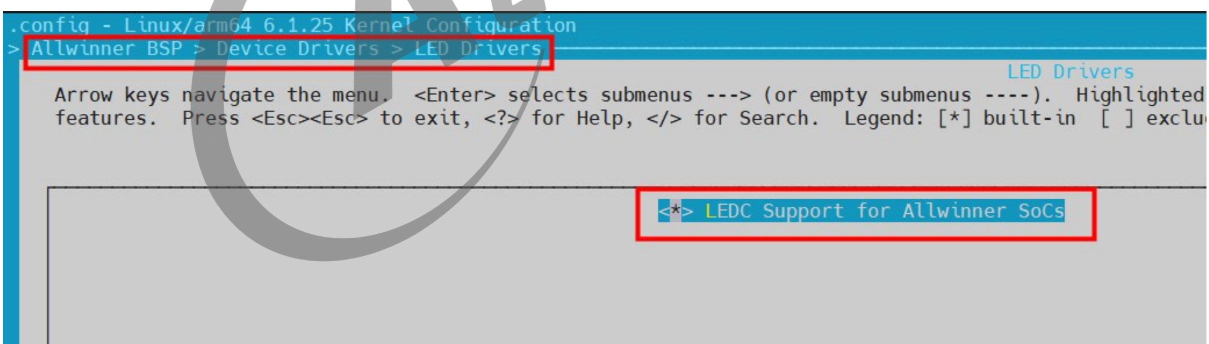


图 2-5: LEDC 的 menuconfig 配置

2.5 源代码结构介绍

LEDC 驱动的源代码位于内核在 drivers/ledc 目录下，具体的路径如下所示：

```
drivers/ledc/
├── ledc-sunxi.c  // sunxi平台的LEDC驱动代码
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 模块接口说明

3.1 内部接口

LEDC 驱动主要内部接口如下：

3.1.1 sunxi_ledc_set_led_brightness

- 函数原型：static int sunxi_ledc_set_led_brightness(struct led_classdev *led_cdev, enum led_brightness value)；
- 功能描述：设置 LEDC 的亮度;
- 参数说明：

（1）led_cdev：led_cdev 结构体；

（2）value：设置的亮度数值 (0-255)；

- 返回值：成功返回 0，失败返回负数。

3.1.2 sunxi_ledc_trans_data

- 函数原型：static void sunxi_ledc_trans_data(struct sunxi_led *led)；
- 功能描述：设置 LEDC 相关寄存器；将 RGB 数据搬到 LEDC FIFO 中；低于 128byte 通过 cpu 模式搬运，超过 128 byte 启动 dma 搬运；启动 LEDC;
- 参数说明：

（1）led，led 结构体。

3.1.3 sunxi_ledc_set_time

- 函数原型：static void sunxi_ledc_set_time(struct sunxi_led *led)；
- 功能描述：模块初始化时设置 reset_ns、t1h_ns、t1l_ns 等的时间;

- 参数说明：

（1）led，led 结构体；

3.1.4 sunxi_ledc_set_output_mode

- 函数原型：static void sunxi_ledc_set_output_mode(struct sunxi_led led, const char str)；
- 功能描述：设置 LEDC 的输出模式（R、G、B 的排布顺序）；
- 参数说明：

（1）led，led 结构体;

（2）str，输出模式（GRB、GBR、RGB、RBG、BGR、BRG）；

3.1.5 sunxi_ledc_set_cpu_mode

- 函数原型：static void sunxi_ledc_set_cpu_mode(struct sunxi_led *led)；
- 功能描述：配置 LED 控制器进入 CPU 模式; 开启 fifo 中断;
- 参数说明：

（1）led，led 结构体;

3.1.6 sunxi_ledc_set_dma_mode

- 函数原型：static void sunxi_ledc_set_cpu_mode(struct sunxi_led *led)；
- 功能描述：配置 LED 控制器进入 DMA 模式; 关闭 FIFO 中断;
- 参数说明：

（1）led，led 结构体;

3.1.7 sunxi_ledc_set_length

- 函数原型：static void sunxi_ledc_set_length(struct sunxi_led *led)；
- 功能描述：设置 LED 的数量;
- 参数说明：

（1）led，led 结构体。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.2 外部接口

3.2.1 brightness 调节说明

每个 LED 在/sys/class/leds 目录下对应有 3 个 led_classdev 设备目录，分别如下：

```
/sys/class/leds/sunxi_led<n>r
/sys/class/leds/sunxi_led<n>g
/sys/class/leds/sunxi_led<n>b
```

其中 n 表示 LED 的编号，n 最小值为 0。

注意：从 LEDC PIN 端开始数，第一个 LED 的编号为 0，沿着远离 PIN 端的方向 LED 的编号依次递增。然后进入上述相应的目录。其中 brightness 文件是指调节相应 LED 的亮度，而 trigger 是调节相应 LED 的亮灭节点。

需要调节第 0 个 LED 的颜色为白光且最亮，操作如下：

```
echo 255 > /sys/class/leds/sunxi_led0r/brightness
echo 255 > /sys/class/leds/sunxi_led0g/brightness
echo 255 > /sys/class/leds/sunxi_led0b/brightness
```

3.2.2 led trigger 使用说明

led 的触发方式的选择，在驱动中，内核已经有着现成的框架为我们做了，$sdk/kernel/${linux-ver}/drivers/leds/led-triggers.c 负责实现触发方式的选择，并在 trigger 目录中有着众多的触发方式具体的实现代码：

用户可以通过 /sys/class/leds/trigger 来设置 trigger 类型。trigger 类型有：backlight、camera、cpu、default-on、disk、gpio、heartbeat、mtd、oneshot、panic、timer、transient.

例如设置 trigger 类型为 timer，操作如下：

```
echo timer > /sys/class/leds/sunxi_led0r/trigger
```

注意：触发模式是 timer 类型的触发，默认亮 500ms，灭 500ms。可通过以下节点设置亮和灭持续的时间。

```
/sys/class/leds/<device>/delay_on
/sys/class/leds/<device>/delay_off
```

注意：触发模式是 default-on 类型的触发，默认一直亮。可通过以下节点设置触发器。

```
echo default-on > /sys/class/leds/<device>/trigger
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4 调试方法

4.1 debugfs 接口

LEDC 相关的 debugfs 文件节点所在目录为/sys/kernel/debug/sunxi_leds，节点说明如下：

- reset_ns：通过该节点可设置和读取 LED 的 reset 时间，范围为 80ns-327us。
- t1h_ns：通过该节点可设置和读取 1 码高电平时间，范围为 80ns-2560ns。
- t1l_ns：通过该节点可设置和读取 1 码低电平时间，范围为 80ns-1280ns。
- t0h_ns：通过该节点可设置和读取 0 码高电平时间，范围为 80ns-1280ns。
- t0l_ns：通过该节点可设置和读取 0 码低电平时间，范围为 80ns-2560ns。
- wait_time0_ns：通过该节点可设置和读取相邻两个 LED 数据之间等待的时间，范围为 80ns-10us。
- wait_time1_ns：通过该节点可设置和读取相邻两帧数据之间等待的时间，范围为 80ns-85s。
- wait_data_time_ns：通过该节点可设置和读取 LEDC 内部 FIFO 等待数据的时间容忍度，范围为 80ns-655us。
- data：通过该节点可读取 data buffer 中的数据，即所有 LED 对应的数据。
- output_mode：通过该节点可设置和读取当前输出的模式，输出模式有 GRB、GBR、RGB、RBG、BGR 和 BRG。
- trans_mode：通过该节点可设置和读取当前的数据传输模式（CPU 或 DMA）。
- hwversion：通过该节点可查看当前 LEDC 的硬件版本。

注意：

1. 设置的时间必须在所说明的时间范围内，否则不会做任何操作。
2. 最终设置寄存器之后得到的时间均为 42ns 的整数倍，若通过节点设置的时间不遵循 42ns 的整数倍，则实际所设置的时间为小于该值的最大能够被 42ns 整除的数。例如通过 reset_ns 设置 90ns，则设置成功之后的 LED reset 时间为 84ns。

debugfs 使用举例如下：

```
echo 84 > /sys/kernel/debug/sunxi_leds/reset_ns
cat /sys/kernel/debug/sunxi_leds/data
echo RGB > /sys/kernel/debug/sunxi_leds/output_mode
echo DMA > /sys/kernel/debug/sunxi_leds/trans_mode
cat /sys/kernel/debug/sunxi_leds/hwversion
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5 用户层使用 demo

```
#include <linux/input.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/time.h>

int main(int argc, char *argv[])
{
    int fd;
    int led_counts = atoi(argv[1]); /* 将参数转换为整数 */
    unsigned char led_raw_buf[led_counts * 3];
    struct timeval start, end;
    long s, us;
    long elapsed;
    int time_count = 1000;
    float fps;

    if ((fd = open("/sys/class/led/light", O_RDWR)) < 0) {
        printf("can't open %s(%s)\n", "/sys/class/led/light", strerror(errno));
        return -1;
    }

    gettimeofday(&start, NULL);

    for (int i = 0; i < time_count; i++ ) {
        memset(led_raw_buf, ((i * 2)% 255), led_counts * 3);
        write(fd, led_raw_buf, sizeof(led_raw_buf));
    }

    gettimeofday(&end, NULL);

    s = end.tv_sec - start.tv_sec;
    us = end.tv_usec - start.tv_usec;
    elapsed = s * 1000000 + us;
    fps = time_count / ((float)elapsed / 1000000);

    printf("test count %d, elapsed %ld us, fps %f\n", time_count, elapsed, fps);

    sleep (1);
    close(fd);
    printf("%s","test finished\n");

    return 0;
}
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

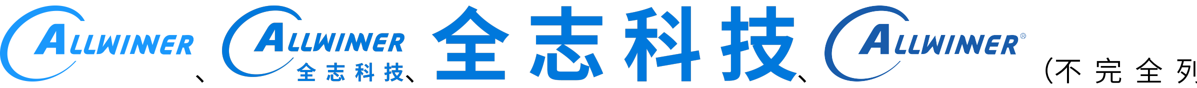
著作权声明

版权所有 ©2025 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明



（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。