HAL_V2 PWM

开发指南

版本号: 1.0

发布日期: 2025.7.15

版本历史

版本号	日期	制/修订人	内容描述
1.0	2025.7.15	AWA2351	初始版本

目　录

15

1 前言

1.1 文档简介

介绍 HAL_V2 中 PWM 驱动的接口及使用方法，为 PWM 使用者提供参考。

1.2 目标读者

PWM 驱动的开发/维护人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
T536	FreeRTOS	hal_pwm.c
T153	FreeRTOS	hal_pwm.c

1.4 文档约定

1 标志说明

⚠ 注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.5 相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台
频率	1 秒钟内信号从高电平到低电平再回到高电平的次数
占空比	一个信号周期中，信号高电平的时间与信号周期的比例，即一个周期内的平均电压
极性	决定了高电平有效或者低电平有效
开关	控制 PWM 信号是否输出

2 模块介绍

2.1 模块功能介绍

脉冲宽度调制（PWM）是一种对模拟信号电平进行数字编码的方法。通过高分辨率计数器的使用，方波的占空比被调制用来对一个具体模拟信号的电平进行编码。PWM 具有以下特点：

支持脉冲（脉冲个数可配）、周期和互补对输出；

（2）支持捕捉输入；

（3）带可编程死区发生器，死区时间可控；

（4）0-24M/100M 输出频率范围、0%-100% 占空比可调、最小分辨率 1/65536；

（5）支持 PWM 输出和捕捉输入产生中断；

（6）支持 PWM 组模式，同组内各个通道起始相位可配置。

模块配置介绍

menuconfig 配置项

```
DRIVERS_V2_PWM  //打开驱动
HAL_TEST_PWM   //打开测试，可不开
```

2.3 模块源码结构

动源码

```
hal_v2/hal/source/pwm/
├── hal_pwm.c
├── common_pwm.h
├── Kconfig
├── Makefile
├── platform
│   ├── pwm_sun8iw22.h
│   └── pwm_sun55iw6.h
└── platform_pwm.h
```

2. 驱动 APIs 声明头文件

```
hal_v2/include/hal/
└── hal_pwm.h
```

3. 驱动 APIs 测试代码

```
hal_v2/hal/examples/pwm/
├── pwm_capture_mode
│   ├── pwm_capture_mode.c
│   └── Makefile
├── pwm_cycle_mode
│   ├── pwm_cycle_mode.c
│   └── Makefile
└── pwm_pulse_mode
    ├── pwm_pulse_mode.c
    └── Makefile
```

3 模块接口说明

表 3-1: 模块接口列表

API	解释说明
hal_pwm_init	PWM 模块初始化
hal_pwm_deinit	PWM 模块反初始化
hal_pwm_channel_enable	PWM 通道输出使能
hal_pwm_channel_disable	PWM 通道输出除能
hal_pwm_cap_enable	PWM 通道捕获使能
hal_pwm_cap_disable	PWM 通道捕获除能
hal_pwm_cap_count_enable	PWM 通道捕获计数使能
hal_pwm_cap_count_disable	PWM 通道捕获计数除能
hal_pwm_channel_pulse_mode_start	PWM 启动 pulse 模式
hal_pwm_pulse_irq_enable	PWM 通道 pulse 模式中断使能
hal_pwm_pulse_irq_disable	PWM 通道 pulse 模式中断除能
hal_pwm_get_rise_cnt	PWM 通道捕获上升沿
hal_pwm_get_fall_cnt	PWM 通道捕获下降沿

接口使用说明

3.1.1 hal_pwm_init

- 作用：PWM 模块初始化，初始化对应的 pwm，通道，配置
- 参数：
  - port 代表 pwm 端口号
  - channel 代表通道

- pwm_config 代表该通道的配置参数

值：

- 0 代表成功
- -1 代表失败

3.1.2 hal_pwm_deinit

- 作用：PWM 模块解初始化
- 参数：

号port 代表 pwm

- 返回值：
  - 0 代表成功
  - -1 代表失败

3.1.3 hal_pwm_channel_enable

- 作用：使能 PWM 模块某个通道
- 参数：

端口号port 代表 pwm

- channel_in 代表通道号

- 返回值：无

3.1.4 hal_pwm_channel_disable

- 作用：除能 PWM 模块某个通道
- 参数：
  - port 代表 pwm 端口号

channel_in 代表通道号

- 返回值：无

3.1.5 hal_pwm_cap_enable

- 作用：使能 PWM 模块某个通道捕获模式
- 参数：

  - port 代表 pwm 端口号
- channel_in 代表通道号

- 返回值：
  - 0 代表成功
  - -1 代表失败

3.1.6 hal_pwm_cap_disable

- 作用：除能 PWM 模块某个通道捕获模式
- 参数：

端口号port 代表 pwm
  - channel_in 代表通道号

- 返回值：
  - 0 代表成功
  - -1 代表失败

3.1.7 hal_pwm_cap_count_enable

：使能 PWM 模块某个通道捕获计数模式
：

  - port 代表 pwm 端口号
  - channel_in 代表通道号

- 返回值：
  - 0 代表成功
  - -1 代表失败

3.1.8 hal_pwm_cap_count_disable

- 作用：除能 PWM 模块某个通道捕获计数模式
- 参数：
  - port 代表 pwm 端口号
  - channel_in 代表通道号

- 返回值：
  - 0 代表成功
  - -1 代表失败

3.1.9 hal_pwm_channel_pulse_mode_start

- 作用：PWM 模块启动 pulse 模式
- 参数：
  - port 代表 pwm 端口号
  - channel_in 代表通道号
- 返回值：无

3.1.10 hal_pwm_pulse_irq_enable

- 作用：使能PWM模块某个通道pulse模式中断
- 参数：
  - port 代表 pwm 端口号
  - channel_in 代表通道号
- 返回值：无

3.1.11 hal_pwm_pulse_irq_disable

：除能PWM模块某个通道pulse模式中断
：

  - port 代表 pwm 端口号
  - channel_in 代表通道号
- 返回值：无

3.1.12 hal_pwm_get_rise_cnt

- 作用：获取某个通道的捕获的上升沿数量
：

  - port 代表 pwm 端口号
  - channel_in 代表通道号
- 返回值：捕获数量

3.1.13 hal_pwm_get_fall_cnt

- 作用：获取某个通道的捕获的下降沿数量
- 参数：
  - port 代表 pwm 端口号
  - channel_in 代表通道号
- 返回值：捕获数量

4 模块使用范例

可参考驱动 APIs 测试代码（hal_v2/hal/examples/pwm/pwm_capture/pwm_capture.c），具体实现如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <hal_clk.h>
#include <hal_reset.h>
#include <hal_interrupt.h>
#include <hal_pwm.h>

#if  defined(CONFIG_KERNEL_FREERTOS)
#include <hal_log.h>
#include <hal_cmd.h>
#include <hal_thread.h>
#include <hal_timer.h>
#elif  defined(CONFIG_KERNEL_BAREMETAL)
#include "shell.h"
#endif

#if  defined(CONFIG_ARCH_SUN55IW6)
#define TEST_RST_BUS_PWM  RST_BUS_PWM0
#define TEST_CLK_PWM_BUS  CLK_PWM0
#define PWM_TEST_BASE     HAL_PWM0
#define PWM_OUT_CHANNEL   PWM_CHANNEL_2
#define PWM_OUT_GPIO_PORT  SUNXI_GPIO_B
#define PWM_OUT_GPIO_PIN   PIN_11
#define PWM_OUT_GPIO_SEL   GPB_PWM0
#define PWM_CAPT_CHANNEL   PWM_CHANNEL_4
#define PWM_CAPT_GPIO_PORT  SUNXI_GPIO_B
#define PWM_CAPT_GPIO_PIN   PIN_13
#define PWM_CAPT_GPIO_SEL   GPB_PWM0
#define PWM_TEST_IRQ      SUNXI_CPUX_IRQ_PWM0
#elif  defined(CONFIG_ARCH_SUN8IW22)
#define TEST_RST_BUS_PWM   RST_BUS_PWM2
#define TEST_CLK_PWM_BUS   CLK_PWM2
#define PWM_TEST_BASE     HAL_PWM2
#define PWM_OUT_CHANNEL PWM_CHANNEL_0
#define PWM_OUT_GPIO_PORT  SUNXI_GPIO_J
#define PWM_OUT_GPIO_PIN   PIN_0
#define PWM_OUT_GPIO_SEL   GPJ_PWM2_0
#define PWM_CAPT_CHANNEL   PWM_CHANNEL_2
#define PWM_CAPT_GPIO_PORT  SUNXI_GPIO_J
#define PWM_CAPT_GPIO_PIN   PIN_2
#define PWM_CAPT_GPIO_SEL   GPJ_PWM2_2
#define PWM_TEST_IRQ      SUNXI_CPUX_IRQ_PWM2
#endif

#define PWM_CAPTURE_RETRYS  20
```

```
#define TIME_1_SECOND 1000000000
#define CLK_SRC1 100000000

static  unsigned int pwm_retrys;

static  int  pwm_clk_init(void)
{
    int  ret;
    hal_rst_t  rst;
    rst_number_t rst_num;
    hal_clk_t clk;
    clk_number_t clk_num;

    rst_num = MAKE_RSTn(AW_SYS_CCU, TEST_RST_BUS_PWM);
    clk_num = MAKE_CLKn(AW_SYS_CCU, TEST_CLK_PWM_BUS);

    printf("reset info, rst_num: %u, RC_ID: %u, RST_ID: %u\n", rst_num, RC_ID(rst_num), RST_ID(rst_num));
    printf("clk info, clk_num: %u, CC_ID: %u, CLK_ID: %u\n", clk_num, CC_ID(clk_num), CLK_ID(clk_num));

    ret  = hal_rst_get_by_rst_num(rst_num, &rst);
    if  (ret)  {
        printf("get reset  handle failed,  rst_num: %u, ret: %d\n", rst_num, ret);
        return -1;
    }

    ret  = hal_n_clk_get_by_clk_num(clk_num, &clk);
    if  (ret)  {
        printf("get clk(%u) failed,  ret:  %d\n", clk_num, ret);
        return -1;
    }

    ret  = hal_rst_deassert(rst);

        printf("deassert reset failed ,  rst_num: %u, ret: %d\n", rst_num, ret);
        ret  = -2;
        goto exit_with_put_rst;
    }

    ret  = hal_n_clk_enable(clk);
    if  (ret)  {
        printf("enable clk(%u) failed,  ret:  %d\n", clk_num, ret);
        ret  = -2;
        goto exit_with_put_clk;
    }

    printf("deassert reset success, rst_num: %u\n", rst_num);
    printf("enable clk(%u) success\n", clk_num);



exit_with_put_clk:
    hal_n_clk_put(clk);

exit_with_put_rst:
    hal_rst_put(rst);
    return ret;
}

static  hal_irqreturn_t pwm_capture_handler(void *data)
```

```
{
    long long data;
    unsigned int base_addr = (
    unsigned int regval;

    regval = readl(base_addr + PWM_PISR);
    writel(regval, base_addr + PWM_PISR);

    hal_pwm_port_t port = PWM_TEST_BASE;
    uint32_t rise  = 0, fall   = 0;
    uint32_t period = 0, duty = 0;

    uint32_t pisr_val  = 0;
    uint32_t cisr_val  = 0;
    uint32_t capture_channel = 0;
    unsigned long long pwm_clk;
    unsigned int reg_offset, pwm_div, temp;

2 pre_scal[][2] ={
        /* reg_value clk_pre_div */
        {0,  1},
        {1,  2},
        {2,  4},
        {3,  8},
        {4,  16},
        {5,  32},
        {6,  64},
        {7,  128},
        {8,  256},
    };

    /* PWM IRQ Status, counter reaches Entire Cycle Value, channel bit is set 1 by hardware.
     * PWM Capture interrupts do not trigger set 1.
     */
sr_val = readl(base_addr + PWM_PISR);
    /* PWM Capture IRQ Ststus */
    cisr_val  = readl(base_addr + PWM_CISR);

    /*
     * calculate the pwm channel according to the interrupt status bit, avoid using for().
     * 0 , 1 --> 0/2
     * 2 , 3 --> 2/2
     * 4 , 5 --> 4/2
     * 6 , 7 --> 6/2
     */
    capture_channel = ((ffs(cisr_val) - 1) & (~(0x01))) / 2;

    if (cisr_val & (0x1 << (capture_channel << 1))) {
        rise  = hal_pwm_get_rise_cnt(port, capture_channel) + 1;
        writel(0x1 << (capture_channel << 1), base_addr + PWM_CISR); /* clean interrupt status */
cisr_val &= ~(0x1 << (capture_channel << 1));

        /* clean capture CRLF and enabled fail interrupt */
        writel(0x1E, base_addr + PWM_CCR + capture_channel * REG_CHN_OFFSET);
    }

    if (cisr_val & (0x2 << (capture_channel << 1))) {
        fall   = hal_pwm_get_fall_cnt(port, capture_channel) + 1;
        writel(0x2 << (capture_channel << 1), base_addr + PWM_CISR); /* clean interrupt status */
        cisr_val &= ~(0x2 << (capture_channel << 1)); /* clean cisr_val flag */
```

```
        /* clean capture CFLF and disabled fail interrupt */
        writel(0x1E, base_addr + PWM_CCR + capture_channel * REG_CHN_OFFSET);


    if (pwm_retrys >= PWM_CAPTURE_RETRYS) {
        reg_offset = hal_get_pccr_reg_offset(capture_channel);
        temp = readl(base_addr + reg_offset);
        pwm_div = pre_scal[temp & (0xf)][1];
        if (temp & (0x1 << PWM_CLK_SRC_SHIFT))
            pwm_clk = CLK_SRC1;
        else
            pwm_clk = CLK_SRC0;

        /* period = (1s / clk_src * period_cycles). */
        /* period = (rise + fall ) * 1000 / 40; */
        period = (rise + fall ) * (TIME_1_SECOND / pwm_clk) * pwm_div;
        duty = fall * (TIME_1_SECOND / pwm_clk) * pwm_div;

finish, print("pwm channel:%d, period:%d ns, duty:%d ns\n", capture_channel, period, duty);
        writel(0x1 << capture_channel, base_addr + PWM_PISR);/* clean interrupt status */
        pisr_val &= ~(0x1 << capture_channel); /* clean cisr_val flag */
        pwm_retrys = 0;

        /* clean capture CRLF and disabled rise interrupt */
        /* hal_writel(0x18, para->reg_base + PWM_CCR + capture_channel * REG_CHN_OFFSET); */

        writel(0x1 << (capture_channel << 1), base_addr + PWM_CISR); /* clean interrupt status */
        cisr_val &= ~(0x1 << (capture_channel << 1)); /* clean cisr_val flag */

        hal_pwm_cap_disable(port, capture_channel);
    }
    pwm_retrys++;

*/  /* clean other status
    writel(cisr_val, base_addr + PWM_CISR); /* clean interrupt status */
    writel(pisr_val, base_addr + PWM_PISR); /* clean interrupt status */

    return 0;
}

void hal_v2_pwm_capture_mode(void)
{
    /* pwm0 clk init */
    pwm_clk_init();

    /* pwm0_2 PB11 init */
    gpio_init_t gpio_initstruct;
    gpio_initstruct.port = PWM_OUT_GPIO_PORT;
    gpio_initstruct.port_num = PWM_OUT_GPIO_PIN;
    gpio_initstruct.mul_sel = PWM_OUT_GPIO_SEL;
    hal_gpio_init(&gpio_initstruct);

    pwm_config_t pwm_initstruct;
    pwm_initstruct.output_mode = PWM_MODE_SINGLE;
    pwm_initstruct.period_ns = 25000;//25us
    pwm_initstruct.duty_ns = 7125;//28.5%
    pwm_initstruct.pulse_num = 0;
    pwm_initstruct.polarity = PWM_POLARITY_NORMAL;
    hal_pwm_init(PWM_TEST_BASE, PWM_OUT_CHANNEL, &pwm_initstruct);
```

```
    /*  enable pwm0_2 and ouput waveform */
    hal_pwm_channel_enable(PWM_TEST_BASE, PWM_OUT_CHANNEL);

    /*  config PB12 as pwm0_4 */
    gpio_initstruct. port =  PWM_CAPT_GPIO_PORT;
    gpio_initstruct. port_num = PWM_CAPT_GPIO_PIN;
    gpio_initstruct. mul_sel = PWM_CAPT_GPIO_SEL;
    hal_gpio_init(&gpio_initstruct);

    //  /*  config pwm0_4 as capture function */
    hal_pwm_cap_enable(PWM_TEST_BASE, PWM_CAPT_CHANNEL, NULL);

    hal_irq_init_t  intc_initstruct ;
    intc_initstruct . irqn  =  PWM_TEST_IRQ;
    intc_initstruct . preemptionpriority = 0;
    intc_initstruct . subpriority =  0;
    intc_initstruct . cmd = true;
c_initstruct . isr_info . func = pwm_capture_handler;
    intc_initstruct . isr_info . parg = (void *) PWM_BASE(PWM_TEST_BASE);
    hal_v2_intc_irq_config(&intc_initstruct);

    /*  connect pwm0_2(PB11) to pwm0_4(PB13), and the capture result will be printed in the interrup handler*/
}

#if  defined(CONFIG_KERNEL_FREERTOS)
FINSH_FUNCTION_EXPORT_CMD(hal_v2_pwm_capture_mode, pwm_capture_mode_test, pwm hal_v2 APIs tests)
#elif  defined(CONFIG_KERNEL_BAREMETAL)
SHELL_EXPORT_CMD(
SHELL_CMD_PERMISSION(0)|SHELL_CMD_TYPE(SHELL_TYPE_CMD_FUNC)|SHELL_CMD_DISABLE_RETURN,
pwm_capture_mode_test, hal_v2_pwm_capture_mode, test for pwm capture mode);
#endif
```

5 FAQ

无

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。