PWM Develop Guide

ID: RK-KF-YF-28

Release Version: V3.0.0

Release Date: 2024-03-26

Security Level: □Top-Secret □Secret □Internal ■Public

DISCLAIMER

THIS DOCUMENT IS PROVIDED “AS IS”. ROCKCHIP ELECTRONICS CO., LTD.(“ROCKCHIP”)DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS,MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip’s registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website： www.rock-chips.com

Customer service Tel： +86-4007-700-590

Customer service Fax： +86-591-83951833

Customer service e-Mail： fae@rock-chips.com

Preface

The Pulse Width Modulation (PWM) function is very common in embedded systems. It is a very effective technology that uses the digital output of a microprocessor to control analog circuits. It is widely used in measurement, Communication into many areas of power control and conversion. This article mainly introduces the basic features, usage and analysis of common problems of Rockchip platform PWM.

Overview

Product Version

Chipset	Kernel Version
RK3036	Linux kernel 4.4 and above
RK312X/PX3SE	Linux kernel 4.4 and above
RK3288	Linux kernel 4.4 and above
RK322X/RK312XH	Linux kernel 4.4 and above
RK3308	Linux kernel 4.4 and above
RK322XH/RK332X	Linux kernel 4.4 and above
RK3326/PX30	Linux kernel 4.4 and above
RK3368/PX5	Linux kernel 4.4 and above
RK3399	Linux kernel 4.4 and above
RK1808	Linux kernel 4.4 and above
RV1109/RV1126	Linux kernel 4.19 and above
RK356X	Linux kernel 4.19 and above
RK3588	Linux kernel 5.10 and above
RV1103/RV1106	Linux kernel 5.10 and above
RK3528	Linux kernel 4.19 and above
RK3562	Linux kernel 5.10 and above
RK3576	Linux kernel 6.1 and above

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Hardware development enginees

Revision History

Version	Author	Date	Change Description
V1.0.0	David Wu	2019-01-28	Initial version
V2.0.0	David Wu	2019-11-14	Support Linux4.19
V2.1.0	Steven Liu	2021-02-24	Add description of Linux4.19
V2.2.0	Steven Liu	2021-12-22	Update version
V2.3.0	Damon Ding	2023-04-03	Add description of Oneshot mode
V3.0.0	Damon Ding	2024-03-26	Optimize the structure of the full text and add a description of PWM v4

Content

PWM Develop Guide

1. Driver

1.1 Kernel Driver

1.1.1 Driver Files

Linux-5.10 and below：

```
drivers/pwm/pwm-rockchip.c
```

Linux-6.1 and above：

```
drivers/pwm/pwm-rockchip.c
drivers/pwm/pwm-rockchip-test.c
```

- Linux-6.1 begins to support the PWM v4 driver. PWM v1-v3 supported by Linux-5.10 and below share the v1 interface, **hereinafter collectively referred to as PWM v1**.
- Linux-6.1 adds a new test driver for testing functions and locating problems. It is also used as an application example of various PWM functions. CONFIG_PWM_ROCKCHIP_TEST needs to be turned on to use it.

1.1.2 DTS Configuration

In DTS, PWM nodes are usually referenced by other drivers, where PWM is configured and used through various interfaces provided by the PWM framework. This section takes a common backlight driver as an example.

PWM v1：

```
backlight: backlight {
    compatible = "pwm-backlight";
    pwms = <&pwm5 0 25000 0>;
    ......
};
```

PWM v4：

```
backlight: backlight {
    compatible = "pwm-backlight";
    pwms = <&pwm1_6ch_1 0 25000 0>;
    ......
};
```

- PWM v1 and PWM v4 nodes are named differently:
  - PWM v1 is pwmX, the actual corresponding controller id is X / 4, and the channel id is X % 4.
  - PWM v4 is pwmX_Ych_Z, X represents the controller id, Y represents the total number of channels supported by the current controller, and Z represents the channel id.
- The number of parameters supported by PWM nodes in Linux-4.4 and above kernels has been increased from 2 in Linux-3.10 to 3. The specific number corresponds to the #pwm-cells attribute of the PWM node. Please refer to the document Documentation/devicetree/bindings/pwm There are detailed instructions in /pwm.txt. Here is only a brief explanation of each parameter:
  - Parameter 1, represents index (per-chip index of the PWM to request), the value is fixed at 0. Each PWM channel of the Rockchip platform corresponds to a PWM device, and each device has only one chip.
  - Parameter 2, represents the period of the PWM output waveform, the unit is ns. The 25000 ns in the example converts to a frequency of 40KHz.
  - Parameter 3 represents the optional parameter polarity, which defaults to 0. If you want to flip the polarity, set it to PWM_POLARITY_INVERTED.

2. Feature support

SOC	PWM version	PWM feature										
		continous	oneshot	capture	global control	output offset	counter	freqency meter	IR output	IR input	wave generator	
RK3036	v1	√	√	√	×	×	×	×	×	×	×	
RK312X/PX3SE	v1	√	√	√	×	×	×	×	×	×	×	
RK3288	v1	√	√	√	×	×	×	×	×	×	×	
RK322X/RK312XH	v1	√	√	√	×	×	×	×	×	×	×	
RK3308	v2	√	√	√	×	×	×	×	×	√	×	
RK322XH/RK332X	v2	√	√	√	×	×	×	×	×	√	×	
RK3326/PX30	v2	√	√	√	×	×	×	×	×	√	×	
RK3368/PX5	v1	√	√	√	×	×	×	×	×	×	×	
RK3399	v1	√	√	√	×	×	×	×	×	×	×	
RK3288	v1	√	√	√	×	×	×	×	×	×	×	
RK1808	v2	√	√	√	×	×	×	×	×	√	×	
RV1109/RV1126	v2	√	√	√	×	×	×	×	×	√	×	
RK356X	v2	√	√	√	×	×	×	×	×	√	×	
RK3588	v2	√	√	√	×	×	×	×	×	√	×	
RV1103/RV1106	v3	√	√	√	×	√	√	×	√	√	×	
RK3528	v3	√	√	√	×	√	√	×	√	√	×	
RK3562	v3	√	√	√	×	√	√	×	√	√	×	
RK3576	v4	√	√	√	√	√	√	√	√	√	√	

3. Application Notes

The application method of PWM kernel and user space has been explained in Documentation/devicetree/bindings/pwm/pwm.txt. This section mainly focuses on further expansion of the PWM features of the Rockchip platform.

3.1 Kernel Driver

If you want to use PWM in the Kernel driver, you can refer to the configuration method of the backlight driver in the "DTS Configuration" chapter, add the pwms attribute under the driver node, and then get/put the PWM device through the following interface:

```
struct pwm_device *pwm_get(struct device *dev, const char *con_id);
void pwm_put(struct pwm_device *pwm);

struct pwm_device *devm_pwm_get(struct device *dev, const char *con_id);
struct pwm_device *devm_fwnode_pwm_get(struct device *dev, struct fwnode_handle *fwnode, const char *con_id);
```

- For detailed implementation and function description, see include/linux/pwm.h and drivers/pwm/core.c.

The interfaces provided by the PWM framework (extracted from Linux-5.10, all interfaces related to legacy drivers have been deleted on Linux-6.1):

```
/**
 * struct pwm_ops - PWM controller operations
 * @request: optional hook for requesting a PWM
 * @free: optional hook for freeing a PWM
 * @capture: capture and report PWM signal
 * @apply: atomically apply a new PWM config
 * @get_state: get the current PWM state. This function is only
 *         called once per PWM device when the PWM chip is
 *         registered.
 * @get_output_type_supported: get the supported output type of this PWM
 * @owner: helps prevent removal of modules exporting active PWMs
 * @config: configure duty cycles and period length for this PWM
 * @set_polarity: configure the polarity of this PWM
 * @enable: enable PWM output toggling
 * @disable: disable PWM output toggling
 */
struct pwm_ops {
    int (*request)(struct pwm_chip *chip, struct pwm_device *pwm);
    void (*free)(struct pwm_chip *chip, struct pwm_device *pwm);
```

```
    int (*capture)(struct pwm_chip *chip, struct pwm_device *pwm,
                   struct pwm_capture *result, unsigned long timeout);
    int (*apply)(struct pwm_chip *chip, struct pwm_device *pwm,
                 const struct pwm_state *state);
    void (*get_state)(struct pwm_chip *chip, struct pwm_device *pwm,
                      struct pwm_state *state);
    int (*get_output_type_supported)(struct pwm_chip *chip,
                struct pwm_device *pwm);
    struct module *owner;

    /* Only used by legacy drivers */
    int (*config)(struct pwm_chip *chip, struct pwm_device *pwm,
                  int duty_ns, int period_ns);
    int (*set_polarity)(struct pwm_chip *chip, struct pwm_device *pwm,
                        enum pwm_polarity polarity);
    int (*enable)(struct pwm_chip *chip, struct pwm_device *pwm);
    void (*disable)(struct pwm_chip *chip, struct pwm_device *pwm);

    ANDROID_KABI_RESERVE(1);
};
```

- Linux-4.4 and above kernels no longer implement interfaces such as config, enable and disable, but implement apply instead.

  The purpose is to use the `int pwm_apply_state(struct pwm_device *pwm, const struct pwm_state *state)` function to atomically change multiple parameters of the PWM device through `struct pwm_state`.

```
/*
 * struct pwm_state - state of a PWM channel
 * @period: PWM period (in nanoseconds)
 * @duty_cycle: PWM duty cycle (in nanoseconds)
 * @polarity: PWM polarity
 * @enabled: PWM enabled status
 * @usage_power: If set, the PWM driver is only required to maintain the power
 *               output but has more freedom regarding signal form.
 *               If supported, the signal can be optimized, for example to
 *               improve EMI by phase shifting individual channels.
 */
struct pwm_state {
    u64 period;
    u64 duty_cycle;
    enum pwm_polarity polarity;
#ifdef CONFIG_PWM_ROCKCHIP_ONESHOT
    u64 oneshot_count;
    u32 oneshot_repeat;
    u64 duty_offset;
#endif /* CONFIG_PWM_ROCKCHIP_ONESHOT */
    bool enabled;
    bool usage_power;
};
```

The basic functions of PWM, including continous, oneshot and caputure, can be applied through the interface provided by the PWM framework. The functions such as frequency meter, counter and wave generator supported by Rockchip platform PWM v4 need to include the header file include/linux/pwm-rockchip.h to use. The following is a detailed introduction to each function and its application. You can also refer to the demo driver drivers/pwm/pwm-rockchip-test.c.

3.1.1 Continous

Continuous output mode supports continuous output of PWM waveform with specified duty cycle.

```
pwm_get_state(pdev, &state);
state.period = period;
state.duty_cycle = duty;
state.polarity = polarity;
state.enabled = enable;
pwm_apply_state(pdev, &state);
```

3.1.2 Oneshot

Single output mode supports outputting a specified number of PWM waveforms. The CONFIG_PWM_ROCKCHIP_ONESHOT configuration needs to be turned on in the Kernel.

```
pwm_get_state(pdev, &state);
state.period = period;
state.duty_cycle = duty;
state.duty_offset = duty_offset;
state.polarity = polarity;
state.oneshot_count = rpt_first;
state.oneshot_repeat = rpt_second;
pwm_apply_state(pdev, &state);
```

- oneshot_count represents the number of waveforms output with a specified duty cycle. The upper limit of the number of waveforms has been extended on PWM v4. The actual number of output waveforms is oneshot_repeat * oneshot_count.
- Oneshot mode will generate an interrupt after the output is completed. Users can add corresponding logic to the drivers/pwm/pwm-rockchip-irq-callbacks.h interrupt processing function as needed:

```
static void rockchip_pwm_oneshot_callback(struct pwm_device *pwm, struct pwm_state *state)
{
    /*
     * If you want to enable oneshot mode again, config and call
     * pwm_apply_state().
     *
     * struct pwm_state new_state;
     *
     * pwm_get_state(pwm, &new_state);
     * new_state.enabled = true;
     * ......
     * pwm_apply_state(pwm, &new_state);
     *
     */
}
```

3.1.3 Capture

Input capture mode supports calculating the duration of high and low levels of the input waveform.

```
pwm_capture(pdev, &cap_res, timeout_ms);
```

- Return the calculated result cap_res after timeout_ms:

```
/**
 * struct pwm_capture - PWM capture data
 * @period: period of the PWM signal (in nanoseconds)
 * @duty_cycle: duty cycle of the PWM signal (in nanoseconds)
 */
struct pwm_capture {
    unsigned int period;
    unsigned int duty_cycle;
};
```

3.1.4 Global control

Global control mode supports synchronous update of multi-channel configurations. Combined with continous/oneshot mode, it can realize output synchronization, complementary output and other functions.

```
// join the global control group
rockchip_pwm_global_ctrl(pdev0, PWM_GLOBAL_CTRL_JOIN);
rockchip_pwm_global_ctrl(pdev1, PWM_GLOBAL_CTRL_JOIN);
rockchip_pwm_global_ctrl(pdev2, PWM_GLOBAL_CTRL_JOIN);
// assign one channel to obtain the permission of global control
rockchip_pwm_global_ctrl(pdev0, PWM_GLOBAL_CTRL_GRANT);
// use pwm_apply_state() to update configurations for each channel
......
// update the configs for all channels in group
rockchip_pwm_global_ctrl(pdev0, PWM_GLOBAL_CTRL_UPDATE);
// enable all channels in group
rockchip_pwm_global_ctrl(pdev0, PWM_GLOBAL_CTRL_ENABLE);
// reclaim the permission of global control
rockchip_pwm_global_ctrl(pdev0, PWM_GLOBAL_CTRL_RECLAIM);
// exit the global control group
rockchip_pwm_global_ctrl(pdev0, PWM_GLOBAL_CTRL_EXIT);
rockchip_pwm_global_ctrl(pdev1, PWM_GLOBAL_CTRL_EXIT);
rockchip_pwm_global_ctrl(pdev2, PWM_GLOBAL_CTRL_EXIT);
```

- Description of each command in global control mode:

```
/**
 * enum rockchip_pwm_global_ctrl_cmd - commands for pwm global ctrl
```

```
 * @PWM_GLOBAL_CTRL_JOIN: join the global control group
 * @PWM_GLOBAL_CTRL_EXIT: exit the global control group
 * @PWM_GLOBAL_CTRL_GRANT: obtian the permission of global control
 * @PWM_GLOBAL_CTRL_RECLAIM: reclaim the permission of global control
 * @PWM_GLOBAL_CTRL_UPDATE: update the configs for all channels in group
 * @PWM_GLOBAL_CTRL_ENABLE: enable all channels in group
 * @PWM_GLOBAL_CTRL_DISABLE: disable all channels in group
 */
enum rockchip_pwm_global_ctrl_cmd {
    PWM_GLOBAL_CTRL_JOIN,
    PWM_GLOBAL_CTRL_EXIT,
    PWM_GLOBAL_CTRL_GRANT,
    PWM_GLOBAL_CTRL_RECLAIM,
    PWM_GLOBAL_CTRL_UPDATE,
    PWM_GLOBAL_CTRL_ENABLE,
    PWM_GLOBAL_CTRL_DISABLE,
};
```

3.1.5 Output offset

Output offset mode supports PWM output waveform offset for a specified time. It is usually used in oneshot mode in combination with global control. It corresponds to the duty offset parameter in `struct pwm_state`. You can refer to oneshot mode description.

3.1.6 Counter

Input counting mode supports counting the number of input waveforms.

```
rockchip_pwm_set_counter(pdev, PWM_COUNTER_INPUT_FROM_IO, true);
msleep(timeout_ms);
rockchip_pwm_set_counter(pdev, 0, false);
rockchip_pwm_get_counter_result(pdev, &counter_res, true);
```

- Close counter after timeout_ms and get the count result counter_res.

3.1.7 Frequency meter

Frequency counter mode supports calculating the frequency of the input waveform.

```
rockchip_pwm_set_freq_meter(pdev, timeout_ms, PWM_COUNTER_INPUT_FROM_IO, &freq_hz);
```

- Returns the calculated result freq_hz after timeout_ms.

3.1.8 IR output

The driver is not supported yet.

3.1.9 IR input

For details, please refer to the document "Rockchip_Developer_Guide_PWM_IR_CN". The corresponding kernel driver is driver/input/remotectl/rockchip_pwm_remotectl.c.

3.1.10 Wave generator

Waveform generator mode supports outputting specified waveforms according to the configuration in the wave table.

```
// setup the duty table
for (i = 0; i < PWM_TABLE_MAX; i++)
    table[i] = i * PWM_WAVE_STEP;
duty_table.table = table;
duty_table.offset = (channel_id % 3) * PWM_TABLE_MAX;
duty_table.len = PWM_TABLE_MAX;

// setup the repeat time for each parameter in table
wave_config.rpt = PWM_WAVE_RPT;

// setup the clk rate
wave_config.clk_rate = 400000;

// If duty_en is true, the wave will get duty config from table each PWM_WAVE_RPT period, and the same to
period_en
wave_config.duty_table = &duty_table;
wave_config.period_table = NULL;
```

```
wave_config.enable = enable;
wave_config.duty_en = true;
wave_config.period_en = false;

// setup the width_mode and update_mode
wave_config.width_mode = PWM_WIDTH_MODE;
wave_config.update_mode = PWM_WAVE_INCREASING_THEN_DECREASING;

// setup the start and end index in duty/period table
wave_config.duty_max = (channel_id % 3 + 1) * PWM_TABLE_MAX - 1;
wave_config.duty_min = (channel_id % 3) * PWM_TABLE_MAX;
wave_config.period_max = 0;
wave_config.period_min = 0;
wave_config.offset = 0;

// setup the middle index to change table config in interrupt if needed.
wave_config.middle = PWM_TABLE_MAX / 2;
rockchip_pwm_set_wave(pdev, &wave_config);

// enable the continous mode
pwm_get_state(pdev, &state);
state.period = period;
state.duty_cycle = duty;
state.polarity = polarity;
state.enabled = enable;
pwm_apply_state(pdev, &state);
```

- The configuration and description related to wave mode are as follows:

```
/**
 * enum rockchip_pwm_wave_table_width_mode - element width of pwm wave table
 * @PWM_WAVE_TABLE_8BITS_WIDTH: each element in table is 8bits
 * @PWM_WAVE_TABLE_16BITS_WIDTH: each element in table is 16bits
 */
enum rockchip_pwm_wave_table_width_mode {
    PWM_WAVE_TABLE_8BITS_WIDTH,
    PWM_WAVE_TABLE_16BITS_WIDTH,
};

/**
 * enum rockchip_pwm_wave_update_mode - update mode of wave generator
 * @PWM_WAVE_INCREASING:
 *     The wave table address will wrap back to minimum address when increase to
 *     maximum and then increase again.
 * @PWM_WAVE_INCREASING_THEN_DECREASING:
 *     The wave table address will change to decreasing when increasing to the maximum
 *     address. it will return to increasing when decrease to the minimum value.
 */
enum rockchip_pwm_wave_update_mode {
    PWM_WAVE_INCREASING,
    PWM_WAVE_INCREASING_THEN_DECREASING,
};

/**
 * struct rockchip_pwm_wave_config - wave generator config object
 * @duty_table: the wave table config of duty
 * @period_table: the wave table config of period
 * @enable: enable or disable wave generator
 * @duty_en: to update duty by duty table or not
 * @period_en: to update period by period table or not
 * @clk_rate: the dclk rate in wave generator mode
 * @rpt: the number of repeated effective periods
 * @width_mode: the width mode of wave table
 * @update_mode: the update mode of wave generator
 * @duty_max: the maximum address of duty table
 * @duty_min: the minimum address of duty table
 * @period_max: the maximum address of period table
 * @period_min: the minimum address of period table
 * @offset: the initial offset address of duty and period
 * @middle: the middle address of duty and period
 * @max_hold: the time to stop at maximum address
 * @min_hold: the time to stop at minimum address
 * @middle_hold: the time to stop at middle address
 */
struct rockchip_pwm_wave_config {
    struct rockchip_pwm_wave_table *duty_table;
    struct rockchip_pwm_wave_table *period_table;
    bool enable;
    bool duty_en;
    bool period_en;
    unsigned long clk_rate;
    u16 rpt;
    u32 width_mode;
```

```
    u32 update_mode;
    u32 duty_max;
    u32 duty_min;
    u32 period_max;
    u32 period_min;
    u32 offset;
    u32 middle;
    u32 max_hold;
    u32 min_hold;
    u32 middle_hold;
};
```

- PWM v4 has 768 * 8bit space in wave generator mode for storing duty/period configuration. After turning on duty_en/period_en, new data will be fetched from the duty_min + offset/period_min + offset index in duty_table/period_table every rpt cycle. Configuration value (unit: ns) until duty_max/period_max. Then it will re-enter the next cycle according to update_mode. If it is oneshot mode, it will stop after oneshot_repeat cycles, while continous mode will continue to output until manually stopped.
- Wave supports width_mode switching (768 * 8bit and 384 * 16bit). Under the same working clock dclk, 16bit mode will support larger duty/period configuration.
- Interrupts will be generated at the configured middle and max indexes. Users can add corresponding logic to the drivers/pwm/pwm-rockchip-irq-callbacks.h interrupt handling function as needed:

```
static void rockchip_pwm_wave_middle_callback(struct pwm_device *pwm)
{
    /*
     * If you want to update the configuration of wave table, set
     * struct rockchip_pwm_wave_table and call rockchip_pwm_set_wave().
     *
     * struct rockchip_pwm_wave_config wave_config;
     * struct rockchip_pwm_wave_table duty_table;
     *
     * //fill the duty table
     * ......
     * wave_config.duty_table = &duty_table;
     * wave_config.enable = true;
     * rockchip_pwm_set_wave(pwm, &wave_config);
     *
     */
}

static void rockchip_pwm_wave_max_callback(struct pwm_device *pwm)
{
    /*
     * If you want to update the configuration of wave table, set
     * struct rockchip_pwm_wave_table and call rockchip_pwm_set_wave().
     *
     * struct rockchip_pwm_wave_config wave_config;
     * struct rockchip_pwm_wave_table duty_table;
     *
     * //fill the duty table
     * ......
     * wave_config.duty_table = &duty_table;
     * wave_config.enable = true;
     * rockchip_pwm_set_wave(pwm, &wave_config);
     *
     */
}
```

3.1.11 Biphasic counter

Bidirectional counter mode supports five counting modes mode0-mode4 (see the description of the PWM chapter in TRM for details). Mode0 can be used as the above counter and freqency meter.

```
biphasic_config.enable = true;
biphasic_config.is_continuous = false;
biphasic_config.mode = biphasic_mode;
biphasic_config.delay_ms = timeout_ms;
rockchip_pwm_set_biphasic(pdev, &biphasic_config, &biphasic_res);
```

- The parameters of biphasic_config are described as follows:

```
/**
 * struct rockchip_pwm_biphasic_config - biphasic counter config object
 * @enable: enable: enable or disable biphasic counter
 * @is_continuous: biphascic counter will not stop at the end of timer in continuous mode
 * @mode: the mode of biphasic counter
 * @delay_ms: time to wait, in milliseconds, before getting biphasic counter result
 */
struct rockchip_pwm_biphasic_config {
    bool enable;
    bool is_continuous;
    u8 mode;
    u32 delay_ms;
};
```

- In non-continous mode, the count result biphasic_res is returned after timeout_ms.
- In continous mode, counting will continue until manual shutdown, and the counting results can be obtained in real time through `int rockchip_pwm_get_biphasic_result(struct pwm_device *pwm, unsigned long *biphasic_res)`.

- biphasic counter mode description:

```
/**
 * enum rockchip_pwm_biphasic_mode - mode of biphasic counter
 * @PWM_BIPHASIC_COUNTER_MODE0: single phase increase mode with A-phase
 * @PWM_BIPHASIC_COUNTER_MODE1: single phase increase/decrease mode with A-phase
 * @PWM_BIPHASIC_COUNTER_MODE2: dual phase with A/B-phase mode
 * @PWM_BIPHASIC_COUNTER_MODE3: dual phase with A/B-phase 2 times frequency mode
 * @PWM_BIPHASIC_COUNTER_MODE4: dual phase with A/B-phase 4 times frequency mode
 */
enum rockchip_pwm_biphasic_mode {
    PWM_BIPHASIC_COUNTER_MODE0,
    PWM_BIPHASIC_COUNTER_MODE1,
    PWM_BIPHASIC_COUNTER_MODE2,
    PWM_BIPHASIC_COUNTER_MODE3,
    PWM_BIPHASIC_COUNTER_MODE4,
    PWM_BIPHASIC_COUNTER_MODE0_FREQ,
};
```

- PWM_BIPHASIC_COUNTER_MODE0 is equivalent to the counter function, and PWM_BIPHASIC_COUNTER_MODE0_FREQ is equivalent to the freqency meter function.

3.2 User space

The PWM framework provides a user layer interface in the /sys/class/pwm/ directory. For details, see drivers/pwm/sysfs.c. After the PWM driver is successfully loaded, the pwmchipX directory will be generated under it, such as pwmchip0, pwmchip1, etc., here The X has nothing to do with the controller or channel ID of the PWM, but is only related to the probe sequence of the PWM device.

```
root@linaro-alip:/# cat /sys/class/pwm/pwmchip0/
device/    export     npwm       power/     subsystem/ uevent     unexport
```

Writing Y to the export node will generate a pwmY directory in the current directory. Since each PWM device on the Rockchip platform has only one chip, the Y value can only be 0. Conversely, writing Y to the unexport node will delete the pwmY directory.

There are the following operable nodes in the pwmY directory:

- enable: Writing 1 enables PWM, writing 0 disables PWM;
- polarity: There are two parameter options: normal or inversed, corresponding to the PWM polarity configuration PWM_POLARITY_NORMAL/PWM_POLARITY_INVERSED;
- duty_cycle: In normal mode, it represents the duration of high level in one cycle (unit: ns). In reversed mode, it represents the duration of low level in one cycle (unit: ns);
- period: represents the period of the PWM waveform (unit: ns);
- oneshot_count: CONFIG_PWM_ROCKCHIP_ONESHOT needs to be turned on, indicating the number of PWM waveforms in oneshot mode;
- oneshot_repeat: CONFIG_PWM_ROCKCHIP_ONESHOT needs to be turned on and is only supported by PWM v4. It indicates the number of times the oneshot mode is repeated. The final number of output waveforms is oneshot_repeat * oneshot_count;
- duty_offset: CONFIG_PWM_ROCKCHIP_ONESHOT needs to be turned on, indicating the offset time of the PWM output waveform (unit: ns);
- capture: Enable capture mode to obtain the duration of the high and low levels of the input waveform (unit: ns).

3.2.1 Continous

```
cd /sys/class/pwm/pwmchip0/
echo 0 > export
cd pwm0
echo 10000 > period
echo 5000 > duty_cycle
echo normal > polarity
echo 1 > enable
```

3.2.2 Oneshot

```
cd /sys/class/pwm/pwmchip0/
echo 0 > export
cd pwm0
echo 10000 > period
echo 5000 > duty_cycle
echo 1000 > duty_offset
echo normal > polarity
echo 100 > oneshot_count
echo 10 > oneshot_repeat
echo 1 > enable
```

3.2.3 Capture

```
cd /sys/class/pwm/pwmchip0/
echo 0 > export
cd pwm0
cat capture
```

4. FAQ

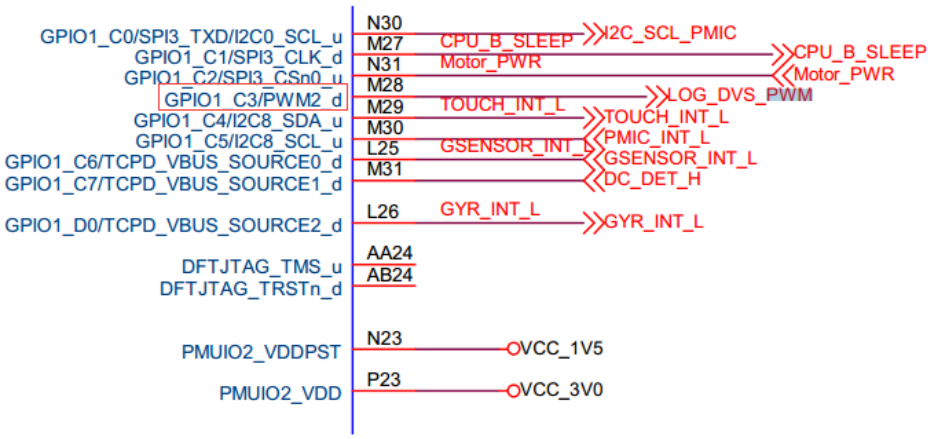
4.1 Connection of PWM Between U-Boot and Kernel

- If U-Boot has the function of PWM voltage regulation, the PWM is still working at the kernel stage, the PWM clock gating count needs to be adjusted to be consistent with the current PWM state according to the current hardware status of the PWM. Otherwise, the clock driver may find that the unused PWM clock, turn off it, which causing the PWM failed to work. The above patch has been modified to ensure the PWM driver: `drivers/pwm/pwm-rockchip.c`, updated to the following submission points:
  1. kernel-4.4: commit e6f2796ef5b660a70102c02d6c15f65ff8701d76
  2. kernel-3.10: commit 5a3d9257d5e379391eb02457ccd70f28a8fb188b
- The frequency of the clock source used by U-Boot and kernel PWM is different, which will also cause switching in the middle, which may cause the PWM duty cycle to change, and similar crashes caused by insufficient PWM voltage regulation will occur. Consistent with the clock source or clock source of the kernel. Make sure that the PWM source clock and source clock frequency of U-Boot PWM is consistent with the kernel.
- Inconsistencies in the polarity and cycle configured by U-Boot and kernel can also lead to middle-state switching, which can lmake the changes for PWM duty cycle, and dead-machine problems such as a lack of PWM voltage control, so keep the U-Boot consistent with kernel's polarity and cycle.

4.2 PWM Pin PULL State As PWM Regulator

When the device rebooting, the registers in the GRF may not reset(second global reset), but the PWM controller reset, which make the PWM pin to be a input state. This will change the default voltage of the PWM Regulator after rebooting by reseting the PWM pin pull state . Therefore, the PWM pin must be configured the same as the default state(pull-up or pull-down) in the kernel, which cannot be configured as "none". This configuration only needs to be modified when the PWM is used as a voltage regulator, as the other functions can be ignored.

- Confirm the default pull-up and pull-down of this PWM pin through the hardware schematic diagram. For example, the RK3399 excavator board PWM2 is used as a voltage regulation function, and the PWM2 pin is found on the schematic diagram: `GPIO1_C3/PWM2_d`, where "d" means down for the default pull-down; if "u" means up for the default pull-up.



- Define the PWM pull down pinctrl in dtsi:

```
        pwm2_pin_pull_down: pwm2-pin-pull-down {
            rockchip,pins =
                <1 19 RK_FUNC_1 &pcfg_pull_down>;
        };
```

- Overwrite pinctrl config at dts:

```
&pwm2 {
    status = "okay";
    pinctrl-names = "active";
    pinctrl-0 = <&pwm2_pin_pull_down>;
};
```

4.3 Oscilloscope Cannot Detect PWM Waveform

If the oscilloscope cannot get the waveform, confirm the following method:

- First check whether the value of the `PWM Counter Register` register is changing. If there is a change, it indicates that the PWM is working. (Note that if you use the io command to read the PWM registers, it need to turn off gating of pclk for RK3328 and the chips later, you can find them in the table of the product documentation , because these chips PWM pclk and the working clock are separated); if the value of this register has not changed, it means that the PWM is irregular. Generally, these exceptions are divided into the following cases:
    1. Clock error
    2. The register configuration problem of the PWM itself, the PWM is not enabled or the value of the duty count is greater than period, etc .;
    3. RK3368 needs to additionally configure `bit12` of the register `GRF_SOC_CON15` in GRF to 1.
- If the read-out value of the `PWM Counter Register` is changing, it means that the PWM is working, but the signal still cannot be measured. It should be a problem with the pinctrl:
    1. iomux error.
    2. io-domain configuration is incorrect;
    3. Interfered by the hardware;