StarFive 赛昉科技

Software SDK Developer Guide for PWM

VisionFive 2
Version: 1.0
Date: 2022/11/10
Doc ID: JH7110-DGEN-010

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Information in this document is provided "as is," with all faults. Contents may be periodically updated or revised due to product development. Shanghai StarFive Technology Co., Ltd. (hereinafter "StarFive") reserves the right to make changes without further notice to any products herein.

StarFive expressly disclaims all warranties, representations, and conditions of any kind, whether express or implied, including, but not limited to, the implied warranties or conditions of merchantability, fitness for a particular purpose, and non-infringement.

StarFive does not assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation indirect, incidental, special, exemplary, or consequential damages.

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Contents

List of Tables....4
List of Figures....5
Legal Statements....ii
Preface....vi
1. Introduction....7
1.1. Function Introduction....7
1.2. Block Diagram....7
1.3. Device Tree Overview....8
1.4. Source Code Structure....8
2. Configuration....9
2.1. Kernel Menu Configuration....9
2.2. Device Tree Source Code....10
2.3. Device Tree Configuration....11
2.4. VisionFive 2 Board Level Configuration....12
3. Interface Description....13
3.1. pwmchip_add....13
3.2. pwm_get....13
3.3. pwm_put....13
3.4. pwm_apply_state....13
4. General Use Example....15
4.1. Sysfs Interface Example....15

List of Tables

Table 0-1 Revision History.....vi

List of Figures

Figure 1-1 Block Diagram....7
Figure 1-2 Device Tree Workflow....8
Figure 2-1 Device Drivers....9
Figure 2-2 PWM Support....10
Figure 2-3 StarFive PWM PTC Support....10

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the PWM of the StarFive next generation SoC platform - JH7110.

Audience

This document mainly serves the PWM relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0		First official release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

Pulse Width Modulation (PWM) provides a way of controlling certain analog quantities, by varying the pulse width of a fixed frequency rectangular waveform. PWM is used in many applications, ranging from communications to power control and conversion.

1.1. Function Introduction

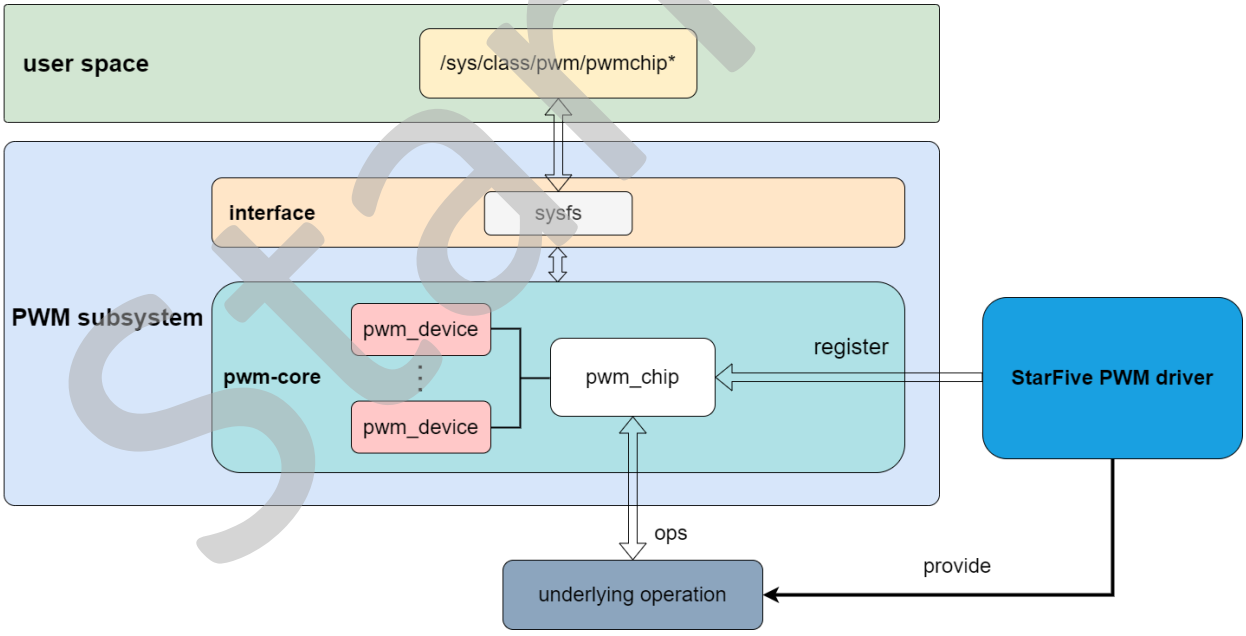
The JH7110 SoC Platform supports the following features and specifications on the PWM module.

- 8 PWM channels
- 32-bit counter/timer facility
- Single-run or continuous run of PTC counter
- Programmable PWM mode
- System clock and external clock sources for timer functionality
- HI/LO Reference and Capture registers
- Three-state control for PWM output driver
- PWM/Timer/Counter functionalities can cause an interrupt to the CPU

1.2. Block Diagram

The following figure shows the block diagram of the PWM driver for JH7110.

Figure 1-1 Block Diagram



As in the above diagram, besides the general layer of user space for all Linux modules, the **PWM subsystem** has the following sub-modules.

- **interface**: The sub-module provides the **sysfs** interface to interact with **pwm-core**.
- **pwm-core**: The sub-module builds the core functions of PWM subsystem. It provides PWM control APIs. **pwm-core** has the following two key components.

- **pwm_chip** represents the module which controls PWM generation.
- **pwm_device** represents a single physical PWM channel.

The **PWM subsystem** communicates with the **StarFive PWM driver** to achieve the PWM functions. The driver registers a **pwm_chip** through API and provide operation functions (ops) for the **pwm-core**, which implements the underlying operations such as reading or writing register values and generating PWM signals within the specified time period.

1.3. Device Tree Overview

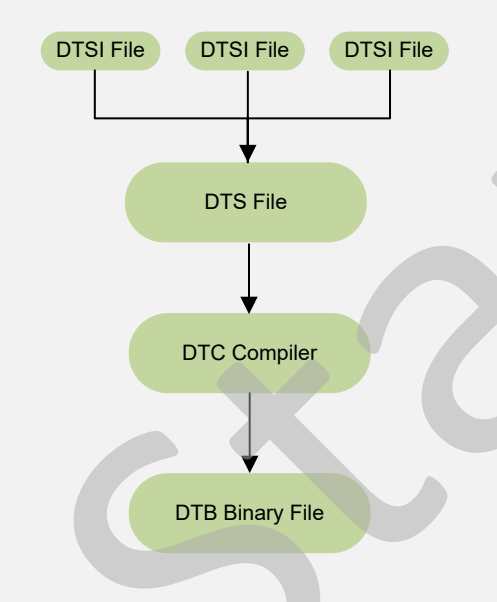
Since Linux 3.x, device tree is introduced as a data structure and language to describe hardware configuration. It is a system-readable description of hardware settings so that the operating system doesn't have to hard code details of the machine.

A device tree is primarily represented in the following forms.

- *Device Tree Compiler (DTC)*: The tool used to compile device tree into system-readable binaries.
- *Device Tree Source (DTS)*: The human-readable device tree description file. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Source Information (DTSI)*: The human-readable header file which you can include in device tree description. You can locate the target parameters and modify hardware configuration in this file.
- *Device Tree Blob (DTB)*: The system-readable device tree binary blob files which is burned in system for execution.

The following diagram shows the relationship (workflow) of the above forms.

Figure 1-2 Device Tree Workflow



1.4. Source Code Structure

The following code block shows the source code structure for the PWM subsystem.

```
linux-5.15.0
├── drivers
|   ├── pwm
|   |   ├── core.c
|   |   ├── sysfs.c
|   |   └── pwm-starfive-ptc.c
```

2. Configuration

2.1. Kernel Menu Configuration

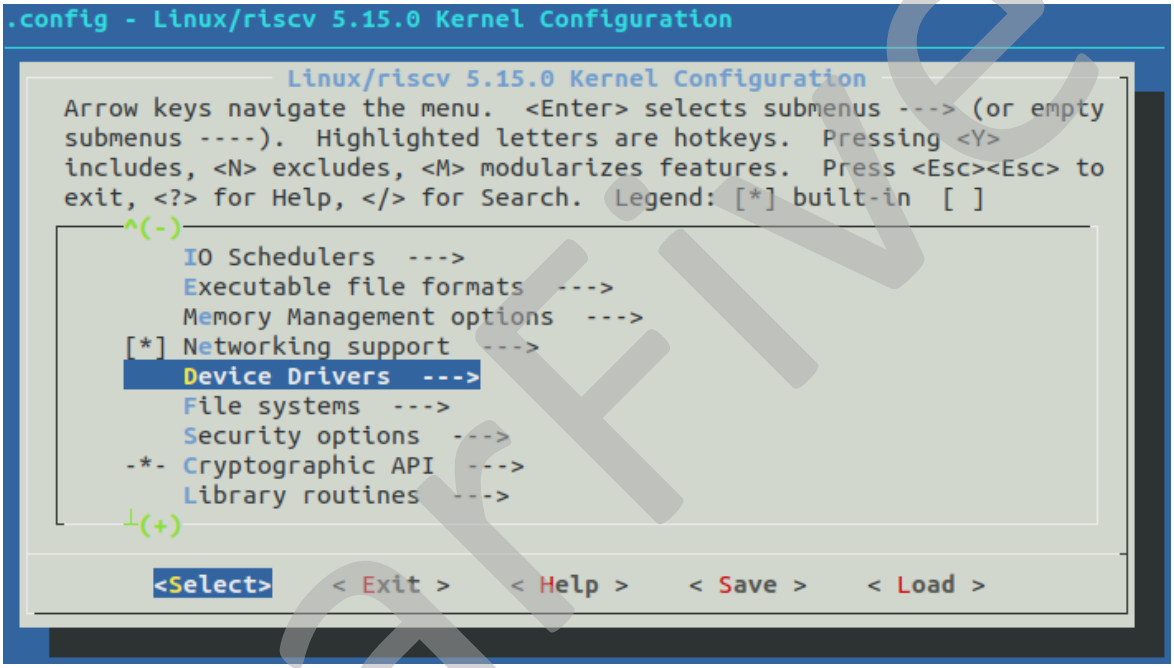
Follow the steps below to enable the kernel configuration for PWM.

1. Under the root directory of `freelight-u-sdk`, type the following command to enter the kernel menu configuration GUI.

```
make linux-menuconfig
```

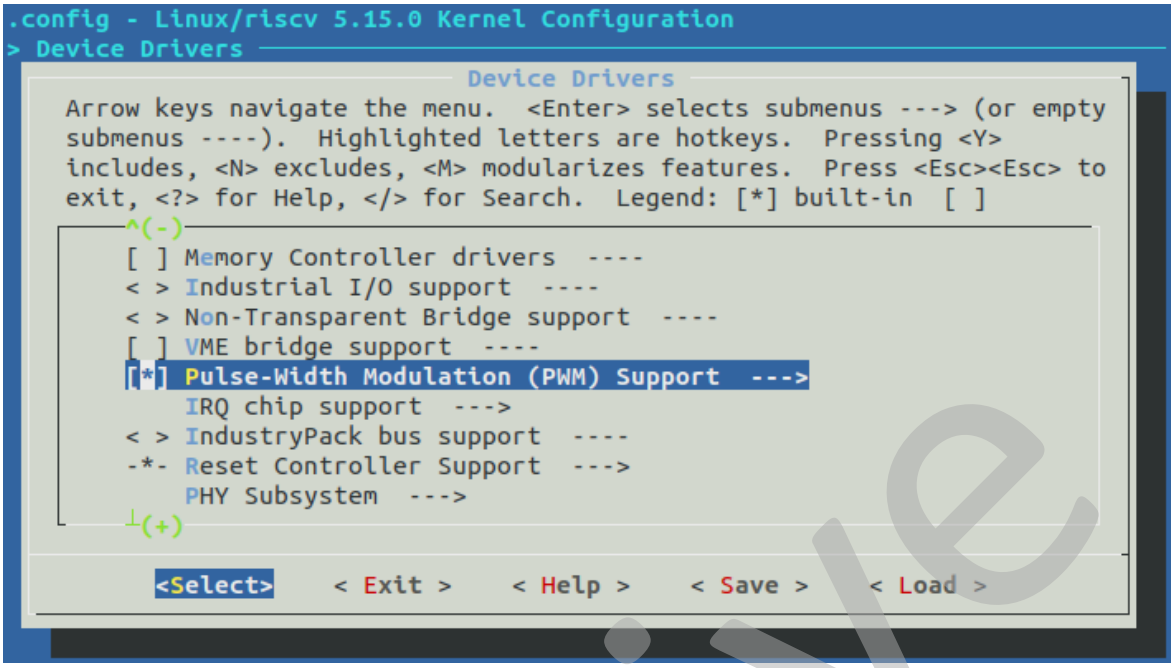
2. Enter the **Device Drivers** menu.

Figure 2-1 Device Drivers



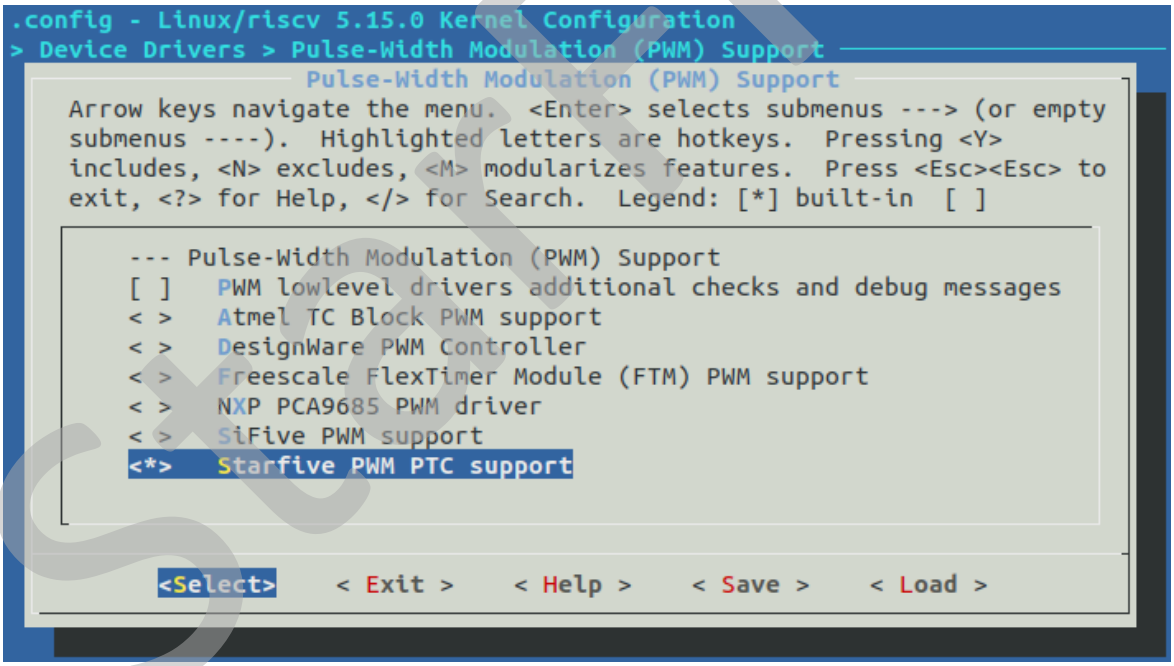
3. Enter the **PWM support** menu.

Figure 2-2 PWM Support



4. Select the **StarFive PWM PTC Support** option.

Figure 2-3 StarFive PWM PTC Support



5. Save your change before you exit the kernel configuration dialog.

2.2. Device Tree Source Code

Overview Structure

The device tree source code of JH7110 is listed as follows:

```
linux
├── arch
|   ├── riscv
|   |   ├── boot
```

```
|   |   |   ├── dts
|   |   |   |   └── starfive
|   |   |   |       ├── codecs
|   |   |   |       |   ├── sf_pdm.dtsi
|   |   |   |       |   ├── sf_pwmdac.dtsi
|   |   |   |       |   ├── sf_spdif.dtsi
|   |   |   |       |   ├── sf_tdm.dtsi
|   |   |   |       |   └── sf_wm8960.dtsi
|   |   |   |       ├── evb-overlay
|   |   |   |       |   ├── jh7110-evb-overlay-can.dts
|   |   |   |       |   ├── jh7110-evb-overlay-rgb2hdmi.dts
|   |   |   |       |   ├── jh7110-evb-overlay-sdio.dts
|   |   |   |       |   ├── jh7110-evb-overlay-spi.dts
|   |   |   |       |   ├── jh7110-evb-overlay-uart4-emmc.dts
|   |   |   |       |   ├── jh7110-evb-overlay-uart5-pwm.dts
|   |   |   |       |   └── Makefile
|   |   |   |       ├── jh7110-clk.dtsi
|   |   |   |       ├── jh7110-common.dtsi
|   |   |   |       ├── jh7110.dtsi
|   |   |   |       ├── jh7110-evb-can-pdm-pwmdac.dts
|   |   |   |       ├── jh7110-evb.dts
|   |   |   |       ├── jh7110-evb.dtsi
|   |   |   |       ├── jh7110-evb-dvp-rgb2hdmi.dts
|   |   |   |       ├── jh7110-evb-pcie-i2s-sd.dts
|   |   |   |       ├── jh7110-evb-pinctrl.dtsi
|   |   |   |       ├── jh7110-evb-spi-uart2.dts
|   |   |   |       ├── jh7110-evb-uart1-rgb2hdmi.dts
|   |   |   |       ├── jh7110-evb-uart4-emmc-spdif.dts
|   |   |   |       ├── jh7110-evb-uart5-pwm-i2c-tdm.dts
|   |   |   |       ├── jh7110-fpga.dts
|   |   |   |       ├── jh7110-visionfive-v2.dts
|   |   |   |       ├── Makefile
|   |   |   |       └── vf2-overlay
|   |   |   |           ├── Makefile
|   |   |   |           └── vf2-overlay-uart3-i2c.dts
```

SoC Platform

The device tree source code of the JH7110 SoC platform is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

VisionFive 2

The device tree source code of the VisionFive 2 *Single Board Computer (SBC)* is in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-visionfive-v2.dts
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-common.dtsi
-- freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110.dtsi
```

2.3. Device Tree Configuration

The PWM device tree source code is stored under the **ptc** node in the file of `jh7110.dts`.

The following code block provides more detail.

```
        ptc: pwm@120d0000 {
            compatible = "starfive,pwm";
            reg = <0x0 0x120d0000 0x0 0x10000>;
            reg-names = "control";
            clocks = <&clkgen JH7110_PWM_CLK_APB>;
            resets = <&rstgen RSTN_U0_PWM_8CH_APB>;
            starfive,approx-freq = <2000000>;
            #pwm-cells=<3>;
            starfive,npwm = <8>;
            status = "disabled";
        };
```

The following list provides more descriptions for the configuration bits.

- **compatible**: Compatibility information, used to associate the driver and its target device.
- **reg**: Register base address "`0x120d0000`" and range "`0x10000`".
- **reg-names**: The names of the registers used by the PWM module.
- **clocks**: The clocks used by the PWM module.
- **resets**: The reset signals used by the PWM module.
- **interrupts**: Hardware interrupt ID.
- **approx-frequency**: Edit this bit to configure the approximate frequency of PWM.
- **status**: The work status of the PWM module. To enable the module, set this bit as "`okay`" or to disable the module, set this bit as "`disabled`".

2.4. VisionFive 2 Board Level Configuration

The board level device tree file (DTSI file) stores all differentiation information for each board-level device. (For example, `common.dtsi`, `jh7110-visionfive-v2.dts`, and `jh7110-visionfive-v2.dtsi`, etc.)

The `common.dtsi` file is stored in the following path:

```
freelight-u-sdk/linux/arch/riscv/boot/dts/starfive/jh7110-common.dtsi
```

For example, the pin configuration of PWM on the VisionFive 2 SBC is listed under the **pwm_pins** node in the file `jh7110-visionfive-v2.dtsi`.

The following code block provides an example of the pins used by PWM including **dout** (Data Output), **doen** (Data Output Enable), and so on.

```
pwm_pins: pwm-pins {
                pwm_ch0-pins {
                        sf,pins = <PAD_GPIO46>;
                        sf,pinmux = <PAD_GPIO46_FUNC_SEL 0>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
                        sf,pin-gpio-dout = <GPO_PTC0_PWM_0>;
                        sf,pin-gpio-doen = <OEN_PTC0_PWM_0_OE_N>;
                };

                pwm_ch1-pins {
                        sf,pins = <PAD_GPIO59>;
                        sf,pinmux = <PAD_GPIO59_FUNC_SEL 0>;
                        sf,pin-ioconfig = <IO(GPIO_IE(1))>;
                        sf,pin-gpio-dout = <GPO_PTC0_PWM_1>;
                        sf,pin-gpio-doen = <OEN_PTC0_PWM_1_OE_N>;
                };
};
```

3. Interface Description

3.1. pwmchip_add

The interface has the following parameters.

- **Synopsis**:

```
int pwmchip_add(struct pwm_chip *chip)
```

- **Description**: The interface is used to register a new PWM chip.
- **Parameter**:
  - **chip**: The PWM chip to add.
- **Return**:
  - **Success**: 0.
  - **Failure**: Error code.

3.2. pwm_get

The interface has the following parameters.

- **Synopsis**:

```
struct pwm_device *pwm_get(struct device *dev, const char *con_id)
```

- **Description**: The interface is used to look up and request a PWM device.
- **Parameter**:
  - **dev**: The device for the PWM consumer.
  - **con_id**: The consumer ID (name).
- **Return**:
  - **Success**: The pointer to the requested PWM device.
  - **Failure**: Error code.

3.3. pwm_put

The interface has the following parameters.

- **Synopsis**:

```
void pwm_put(struct pwm_device *pwm)
```

- **Description**: The interface is used to release a PWM device.
- **Parameter**:
  - **pwm**: The PWM device to release.
- **Return**: None.

3.4. pwm_apply_state

The interface has the following parameters.

- **Synopsis**:

```
int pwm_apply_state(struct pwm_device *pwm, const struct pwm_state *state)
```

- **Description**: The interface is used to apply a new state to a PWM device atomically.
- **Parameter**:
    - **pwm**: The PWM device to change status.
    - **state**: The new state to apply to the PWM device.
- **Return**:
    - **Success**: 0.
    - **Failure**: Error code.

4. General Use Example

4.1. Sysfs Interface Example

In the user space, you can easily control PWM through **sysfs** interface supported by Linux.

The following procedure provides an example for it.

1. Enter the following directory, you can see all the registered PWM chips.

```
# cd /sys/class/pwm/
# ls
pwmchip0
```

2. Enter the directory `pwmchip*`, run command `echo X > export`.

Note:
In the command, the parameter **X** is the PWM channel which you want to control.

Result: Then the directory `pwmX` is generated.

```
# cd pwmchip0/
# ls
device     export     npwm       power      subsystem  uevent     unex-port
# echo 0 > export
# ls
device     npwm       pwm0       uevent
export     power      subsystem  unexport
# echo 1 > export
# ls
device     npwm       pwm0       subsystem  unexport
export     power      pwm1       uevent
```

3. Enter the directory `pwmX`, you can see all states of this PWM channel.

   The following list provides descriptions for the parameters.

   - **enable**: Enable or disable the PWM channel. `1` for enable, `0` for disable.
   - **polarity**: Invert polarity or not. `1` for invert polarity, `0` for normal (do not invert).
   - **period**: Enter the period of PWM signal (unit: ns)
   - **duty_cycle**: Enter the duty cycle of PWM signal (unit: ns)

4. Now, you can change the states of PWM channel using commands of echo and cat.

```
# cd pwm0/
# ls
capture     enable      polarity    uevent
duty_cycle  period      power
# echo 10000 > period
# echo 5000 > duty_cycle
# cat period
10000
# cat duty_cycle
5000
# cat polarity
normal
# cat enable
1
```