GPIO PWM and Capture Linux Module Driver

Example Code Introduction for 64/32-bit NuMicro® Family

file information

Application	This sample code shows how to import GPIO PWM and GPIO Capture driver into the Linux kernel, realize PWM output and detect the change status of pins.
BSP version	Linux-5.10.x
Hardware	NuMaker-IOT-MA35D1 V2.0

1. Overview

This sample code demonstrates how to simulate the PWM signal through the “GPIO PWM” module, and connect the signal to the GPIO which is set by the “GPIO Capture” module as an input pin. When the input pin state is changed, the “GPIO Capture” module will trigger an interrupt to count the number of interrupts and record time tick. User can read the interrupt information through the " Capture " test program and the /sys interface. Figure 1-1 shows GPIO PWM that outputs pulse width modulation through GPIO and GPIO Capture that detects the state change and the time of pin input change.

Considering practicability, the “GPIO Capture” module only supports up to 32 pins in the design. Since the high-precision time in the driver is used to realize the secondary detection function of the input pins, it is necessary to consider whether too many pins to be enabled will affect system performance.

The number of “GPIO PWM” signals is not limited; however, because the module uses high-precision time to calculate the duty cycle, if the number is too large, it will affect the efficiency of the system and will decrease the accuracy of the duty cycle.

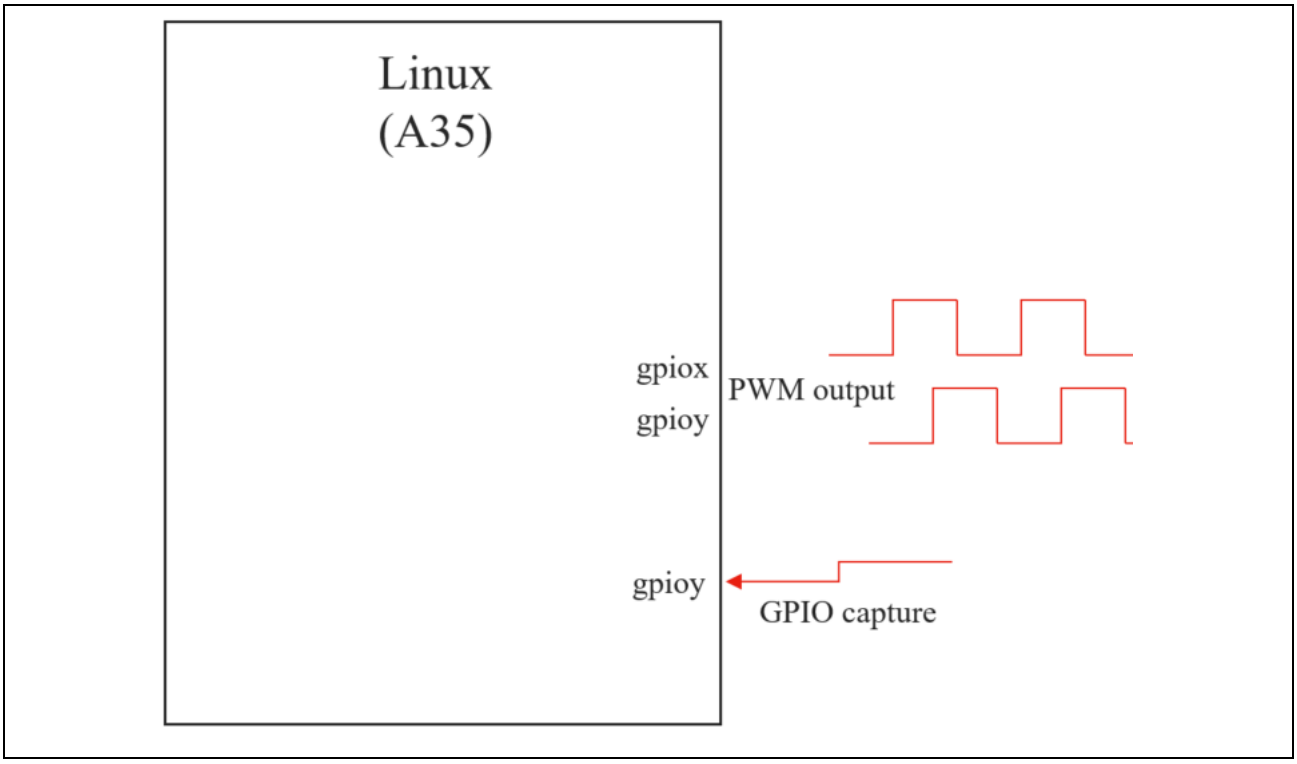

Figure 1-1 Functionality Overview

1.1 Software Architecture

The software architecture is shown in Figure 1-2. Users can modify the device tree to meet their request. The driver will register the PWM and the desired detection function according to the user's settings.

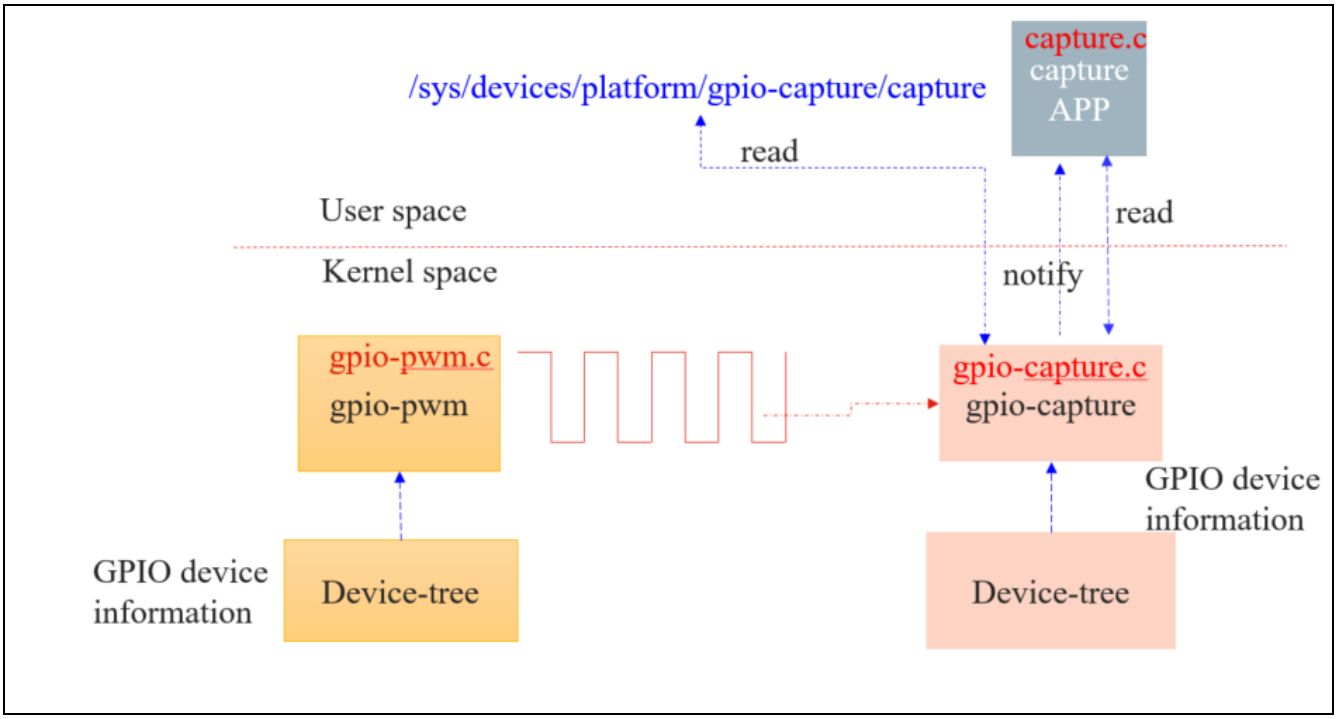

Figure 1-2 Software Architecture

1.2 Driver Settings

1.2.1 GPIO PWM Settings

1.2.1.1 Kernel Configuration Settings

GPIO PWM must enable the Linux PWM compiler configuration in kernel build.

Follow the setting flow below to enable PWM driver:

Device Drivers ---> [*] Pulse-Width Modulation (PWM) Support

1.2.1.2 Device Tree Settings

```
/* 1. Add node in device tree root's configuration */
/ {
    model = "Nuvoton MA35D1-IoT";
    ....
    gpio_pwm {
    compatible = "gpio-pwm";
    status = "okay";
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gpio_pwm>;
    gpios = <&gpioi 12 GPIO_ACTIVE_LOW>,
     <&gpioi 13 GPIO_ACTIVE_LOW>;
    };
};

/* 2. Add module pin configuration to device tree's pinctl as below */
&pinctrl {
....
gpio_pwm {
         pinctrl_gpio_pwm: gpio_pwmgrp{
        nuvoton,pins =
        <SYS_GPI_MFPH_PI12MFP_GPIO &pcfg_default>,
        <SYS_GPI_MFPH_PI13MFP_GPIO &pcfg_default>;
        };
};
....
};
```

- GPIO output pin: PWM output pin selection.
  It should be noted that the device tree is set with port plus pin, but in program it must be converted as formula below.

  pinxy(in device tree) = 16*x+y(in program)

  For example, gpioi12 = 16*8 + 12 = gpio140.

  (port number: a=0, b=1, c=2 ……)

- PWM chip base: The chip index value when registering PWM. The driver uses number 32 as chip index, and will generate /sys/class/pwm/pwmchip32 interface for user operation.

1.2.2 GPIO Capture Settings

```
/* 1. Add node in device tree root's configuration */
/ {
    model = "Nuvoton MA35D1-IoT";
    ....
    gpio_cap {
    compatible = "gpio-capture";
    status = "okay";
    debounce-us = <20>;
    irq-type = <IRQ_TYPE_EDGE_RISING>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_gpio_cap>;
    gpios = <&gpioi 14 GPIO_ACTIVE_LOW>,
     <&gpioi 15 GPIO_ACTIVE_LOW>;
    };
};

/* 2. Add module pin configuration to device tree's pinctl as below */
&pinctrl {
....
gpio_cap {
         pinctrl_gpio_cap: gpio_capgrp{
        nuvoton,pins =
        <SYS_GPI_MFPH_PI14MFP_GPIO &pcfg_default>,
        <SYS_GPI_MFPH_PI15MFP_GPIO &pcfg_default>;
        };
};
....
};
```

- Capture gpio index: capture input detection pin selection.

  It should be noted that the device tree is set with port plus pin, but in program it must be converted as formula below.

  pinxy(in device tree) = 16*x+y(in program)

  For example, gpioi14 = 16*8 + 14 = gpio142.

  (port number: a=0, b=1, c=2 ……)

- Capture type: the trigger type of pin detection, currently there are

  IRQ_TYPE_EDGE_RISING,

  IRQ_TYPE_EDGE_FALLING, IRQ_TYPE_LEVEL_HIGH,

  IRQ_TYPE_LEVEL_LOW, and other settings.

- Filter time

Taking debounce_us=20 as an example, when a state change is detected for the first time, the state of the pin will be s1 first, and the state of the pin will be s2 after 20us . The state changes are valid only when s1 is equal to s2.

1.2.3 Execution Instructions

1.2.3.1 GPIO PWM

After the GPIO PWM module is executed, it will output PWM (Pulse Width Modulation) as shown in Figure 1-3. Users can adjust the "period" and "duty cycle" output through the Linux's /sys interface. The adjustment method is shown in Chapter 5 Example Code Execution.

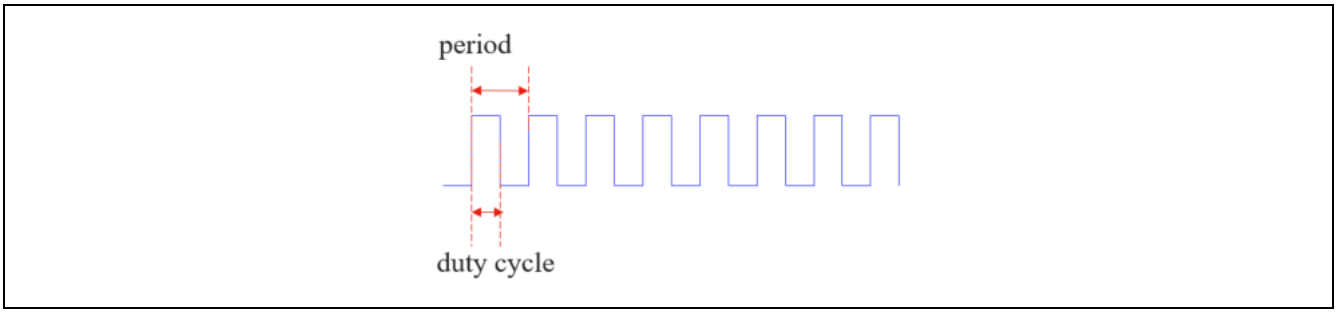

Figure 1-3 PWM Output

1.2.3.2 GPIO Capture

After the “GPIO Capture” module is installed, it will detect the input changes of the pins, and the output can read through /sys and device node (refer to Figure 1-2). The output format is shown in Figure 1-4.

- state: pin detection state
- gpiox: pin number
- count: the number of pin changes
- time: pin change time (jiffies low 32 bits)

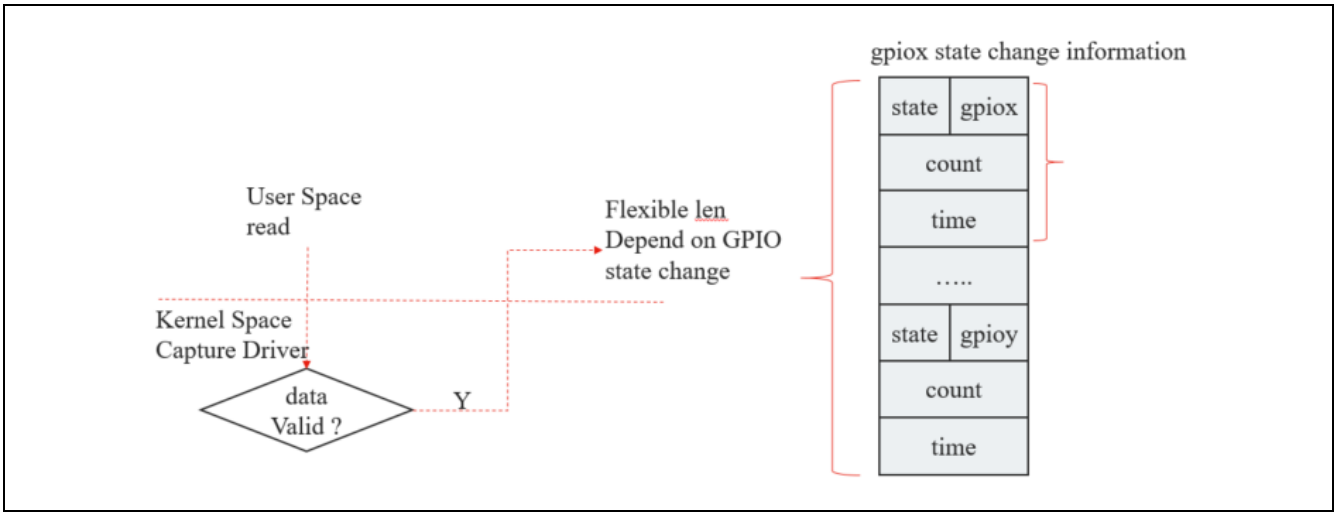

Figure 1-4 Capture Output Format

1.2.3.3 Demo Result

Follow the steps below:

1. Do hardwired (refer to Section 3.2 Hardware Requirements).
2. Install core modules and execute application software (refer to Chapter 5 Example Code Execution).

After the above steps, you can see the following results:

```
gpio142=1 count=1228, time=10375353
gpio142=1  count=1229, time=10375363
gpio142=1  count=1230, time=10389473
gpio142=1  count=1231, time=10389491
```

2. Code Description

The "capture" program reads /dev/gpio_cap (refer to the Section 1.2.3.2 GPIO Capture for output format). The following shows how to read the change status information of the input pin, and the description is shown in the comment.

```
/* 4. If drive got gpio status change will notify user process, this register callback
function will be called */
void sig_event_handler(int sig_id, siginfo_t *sig_info, void *unused)
{
    if ( sig_id == CAP_SIG_ID ) {
        state_change = 1;
    }
}

int main()
{
    int fd;
    struct sigaction act;
    unsigned int read_buf[CAP_MAX_LEN_INT];
    int read_len;
    int data_idx;
    int line_idx;
    int i;

    /* 1. Signal callback function register */
    sigemptyset(&act.sa_mask);
    act.sa_flags = (SA_SIGINFO | SA_RESTART);
    act.sa_sigaction = sig_event_handler;
    sigaction(CAP_SIG_ID, &act, NULL);

    /* 2. Open device & in driver will catch process notify information */
    fd = open("/dev/gpio_cap", O_RDWR);
    if(fd < 0) {
        printf("Open device fail\n");
        return 0;
    }

    int gpio;
    int state;
    unsigned int count;
    unsigned int time;
    int gpio_info_nums;
    unsigned int pre_count;

    while(1) {
        fflush(0);

    /* 3. check if drive got new gpio status */
        if (state_change == 1) {
    /* 5. Got gpio status change */
            read_len = read(fd, read_buf, CAP_MAX_LEN);
            read_len = read_len / 4;
            gpio_info_nums = read_len / GPIO_INFO_SIZE_INT;   /* 3 int gpio info */
```

```
        /* 6. Read all change gpio status information */
             for (i=0; i<gpio_info_nums; i++) {
                 pre_count = count;
                 gpio = read_buf[i*GPIO_INFO_SIZE_INT] & 0x000000ff;
                 state = read_buf[i*GPIO_INFO_SIZE_INT] >> 16;
                 count = read_buf[i*GPIO_INFO_SIZE_INT + 1];
                 time = read_buf[i*GPIO_INFO_SIZE_INT + 2];

                 if ( count - pre_count != 1) {
                     printf("D");
                 }
printf("gpio%d=%d  count=%d, time=%u\n", gpio, state, count, time);
             }
             state_change = 0;
         }
     }

     close(fd);
}
```

3. Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - Linux-5.10.x
- Module names
  - gpio-pwm.ko, gpio-capture.ko
- Application: capture

3.2 Hardware Requirements

- Circuit components
  - NuMaker-IOT-MA35D1 V2.0
- Test schematic

In NuMaker-IOT-MA35D1 V2.0 Board, CON4

Select gpioi12 (gpio140) as PWM output pin

Select gpioi14 (gpio142) as Input capture pin

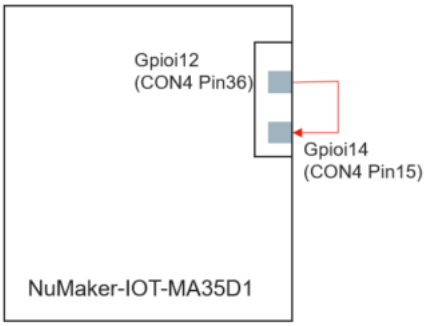

Figure 3-1 Test Schematic

4. Directory Information

The directory structure is shown below.

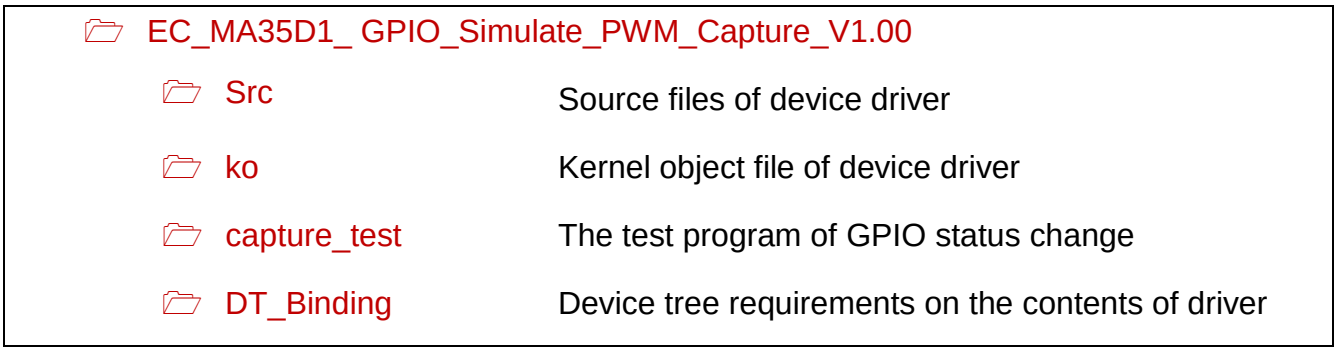

Figure 4-1 Directory Structure

5. Example Code Execution

Perform pin settings (refer to the Section1.2 Driver Settings).

1. Re-compile the kernel and module in the built environment, then download to MA35D1.
2. Power on the MA35D1.
3. Copy the kernel module and test program "capture" to the board side.
4. Install kernel modules.

```
# insmod gpio-pwm.ko
# insmod gpio-capture.ko
```

5. Detect GPIO Capture performs pin status (refer to Chapter 2 Code Description)

```
# ./capture &
```

6. Perform GPIO PWM output and duty cycle settings:
   - Export PWM

```
# echo 0 > /sys/class/pwm/pwmchip32/export
```

   - PWM output period setting in ns unit; gpioi12 will output 100 HZ PWM signal

```
# echo 10000000 > /sys/class/pwm/pwmchip32/pwm0/period
```

   - PWM Duty setting, gpioi12 output duty is 5000000 /10000000 = 0.5 duty

```
# echo 5000000 > /sys/class/pwm/pwmchip32/pwm0/duty_cycle
```

   - PWM enable output

```
# echo 1 >/sys/class/pwm/pwmchip32/pwm0/enable
```

6. Revision History

Date	Revision	Description
2022.09.27	1.00	Initial version.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.