GPIO LEDS

From ArmadeusWiki

How to use *gpio-leds* driver to manage states of the LEDs connected to your Armadeus Dev board.

Contents

Introduction

You can manage a LED connected to a GPIO pin with a sysfs interface very similar to the standard GPIO sysfs driver, but you will have some new features like triggers (e.g. "heartbeat" trigger will make LED blink like a heart at the rate of the CPU load) . Here are the GPIO used for the user LED on each APF board:

- APF9328: PORT A / bit 2
- APF27: GPIO_PORTF | 14
- APF28: PINID_GPMI_RDY1 (Bank 0 - pin 21)
- APF51: GPIO_PORTA | 2
- APF6: GPIO7 | 12
- OPOS6UL: GPIO3 | 4

Configuration

Note: Following configuration instructions are now done by default in BSP > 5.3, so you can skip them if you use a recent armadeus software

You need to enable the leds-gpio driver in your kernel and some triggers like the "heartbeat".

- Configure Linux kernel:

```
$ make linux-menuconfig
```

```
Device Drivers  --->
     --- LED support
         [*] LED Class Support
              *** LED drivers ***
         <*> LED Support for GPIO connected LEDs
              [*] Platform device bindings for GPIO LEDs
              *** LED Triggers ***
         [*]   LED Trigger support
         <*>     LED Timer Trigger
         <*>     LED Heartbeat Trigger
         <*>     LED backlight Trigger
         <*>     LED Default ON Trigger
```

Then, in your *apfXX-dev.c*, you would need to define your LED **before** the variable *platform_devices[]*. This code is already implemented for the APF27, APF28 and APF51 so the source code hereafter (for the APF27) is only present as a reference sample to understand how to activate a GPIO LED driver.

```
#include <linux/leds.h>

/* GPIO LED */
#if defined(CONFIG_LEDS_GPIO) || defined(CONFIG_LEDS_GPIO_MODULE)
static struct gpio_led apf27dev_led[] = {
        {
                .name = "apfdev:green:user",
                .default_trigger = "heartbeat",
                .gpio = (GPIO_PORTF | 14),
                .active_low = 1,
        },
};

static struct gpio_led_platform_data apf27dev_led_data = {
        .num_leds       = ARRAY_SIZE(apf27dev_led),
        .leds           = apf27dev_led
};

static struct platform_device apf27dev_led_dev = {
        .name           = "leds-gpio",
        .id             = -1,
        .dev            = {
                .platform_data  = &apf27dev_led_data,
        },
};
#endif /* CONFIG_LEDS_GPIO */
```

Add the LED to get it managed by the kernel.

```
static struct platform_device *platform_devices[] __initdata = {
#if defined(CONFIG_LEDS_GPIO) || defined(CONFIG_LEDS_GPIO_MODULE)
        &apf27dev_led_dev,
#endif
        ALSA_SOUND

};
```

Then rebuild and update your bard with the new kernel. Upon the next kernel boot you should see the LED flash like a heartbeat (if you have activated the "heartbeat" trigger)

Usage

- In following instructions replace $LED with "*apfdev\:green\:user*", except on OPOS6ULDev where you will use "*User*":

```
# ls /sys/class/leds/$LED/
brightness      max_brightness  subsystem       uevent
device          power           trigger
```

You can change the trigger behaviors. By default, Heartbeat is selected:

- "heatbeat": led blinks like a heart and blink frequency will change according o the CPU activity.
- "nand-disk": the led will blink each time nand access occur (try with *sync* command to see it blinking).

```
# cat /sys/class/leds/$LED/trigger
none nand-disk mmc0 timer [heartbeat] backlight gpio default-on

# echo none > /sys/class/leds/$LED/trigger
```

Switch on and off the LED

```
# cat /sys/class/leds/$LED/max_brightness > /sys/class/leds/$LED/brightness

# echo 0 > /sys/class/leds/$LED/brightness
```

It is possible to switch LED state using the APF28Dev/OPOS6ULDev User button. This button is seen as gpio17 (as seen under APF28Dev datasheet) under Linux; gpio43 on OPOS6ULDev. Configure the LED trigger as gpio :

```
# echo "gpio" > /sys/class/leds/$LED/trigger
```

New config files are available :

```
# ls /sys/class/leds/$LED/
brightness          gpio                power               uevent
desired_brightness  inverted            subsystem
device              max_brightness      trigger
```

Then the trigger gpio can be configured with *gpio* file (replace 17 with 43 on OPOS6ULDev) :

```
# echo 17 > /sys/class/leds/$LED/gpio
```

Then pushing the user switch will now commute the LED state.

Links

- http://www.kernel.org/doc/Documentation/leds/leds-class.txt