fabiobaltieri

Fabio Firmware?

Linux LED Subsystem

2011/09/21 6 COMMENTS (HTTP://FABIOBALTIERI.COM/2011/09/21/LINUX-LED-SUBSYSTEM/#COMMENTS)

LEDs… Everyone likes that! Those little shiny electronic devices are mounted on any well-made electronic equipment to indicate at a glance its working status. They tell you when your network has activity, when your laptop battery is empty, when your hard-drive is working, when your amplifier is overloading… they may even light up your bedroom (http://www.ikea.com/us/en/catalog/products/30150984)!

In embedded systems the proper design of the front panel, with the right LED illuminated icons, is an essential feature and if you are familiar with network troubleshooting you can understand why!

Well-made devices should have a panel that instantly gives you an idea of what's working and what's not just by looking at it.

If you are using Linux as your kernel on a SoC design, you'll be glad to know that it has an entire subsystem dedicated to LEDs!

In this post I'll show how you to check if your system has some controllable LEDs, and how to use that from userspace applications and kernel drivers.

Host Drivers

Kernel controllable LEDs are commonly found on embedded systems, where the CPU usually has some LEDs controlled by its GPIO pins or by some external peripheral, such as I2C/SPI port expander.

There are many LED-enabled device drivers in the kernel sources, some of which are dedicated drivers, such as the one for the LEDs in PC Engines ALIX (http://lxr.linux.no/#linux/drivers/leds/leds-alix2.c) boards, while others are just drivers for devices with integrated LEDs, such as the one for RaLink wireless network cards (http://lxr.linux.no/#linux/drivers/net/wireless/rt2x00/rt2x00leds.c).

Just run a search on the kernel code for the *led_classdev_register* function to see if you find something to play with.

The most common driver you'll find in modern ARM-based SoC is "leds-gpio (http://lxr.linux.no/#linux/drivers/leds/leds-gpio.c)", which is used to control LEDs connected to GPIO of any gpiolib enabled CPU, including most ARM SoC.

Registering a GPIO with the leds-gpio driver is easy, take a look at this code from the nslu2-setup.c (http://lxr.linux.no/#linux/arch/arm/mach-ixp4xx/nslu2-setup.c) driver:

```
#include <linux/leds.h>
...
static struct gpio_led nslu2_led_pins[] = {
        {
                .name           = "nslu2:green:ready",
                .gpio           = NSLU2_LED_GRN_GPIO,
        },
...
};

```

```
static struct gpio_led_platform_data nslu2_led_data = {
        .num_leds               = ARRAY_SIZE(nslu2_led_pins),
        .leds                   = nslu2_led_pins,
};

static struct platform_device nslu2_leds = {
        .name                   = "leds-gpio",
        .id                     = -1,
        .dev.platform_data      = &nslu2_led_data,
};

static struct platform_device *nslu2_devices[] __initdata = {
...
        &nslu2_leds,
...
};
```

On these platforms, when all the structures are in place, if you want to add a new GPIO controlled LED just add the appropriate structure with the name, pin and properties and light it up! If it's not working, check that all the necessary modules are compiled in and be sure that your pin is configured as a GPIO, since in many CPU you have to do that explicitly.

If you're a lucky owner of an embedded PowerPC board, you might like to know that the leds-gpio driver is openfirmware enabled!

Check out this (http://lxr.linux.no/#linux/arch/powerpc/boot/dts/mpc8315erdb.dts) configuration for the MPC8315E-RDB board:

```
leds {
        compatible = "gpio-leds";
        pwr {
                gpios = <&mcu_pio 0 0>;
                default-state = "on";
        };
        hdd {
                gpios = <&mcu_pio 1 0>;
                linux,default-trigger = "ide-disk";
        };
};
```

Note that while the platform driver registers as "leds-gpio", the openfirmware one matches with "gpio-leds".

Also, when you are naming your LEDs, you might want to follow the convention suggested in the official documentation (http://lxr.linux.no/#linux/Documentation/leds/leds-class.txt), which tells you to name your led as: "devicename:colour:function".

Controlling LEDs from Userspace

So, you have your kernel configured and the system booted. Now what?

First, be sure that the kernel registered the LEDs, as in:

```
balto@balto-mpc:~# dmesg | grep "led device"
Registered led device: pwr
Registered led device: hdd
```

Now, go to the "/sys/class/leds/" directory and look at the content:

```
balto@balto-mpc:~# cd /sys/class/leds/hdd
balto@balto-mpc:/sys/class/leds/hdd# ls
brightness  device  max_brightness  power  subsystem  trigger
uevent
```

You should see what's coming: to turn on the LED, just use:

```
balto-mpc:/sys/class/leds/hdd# echo 1 > brightness
```

and to turn it off use:

```
balto-mpc:/sys/class/leds/hdd# echo 0 > brightness
```

Also note that, as suggested by the content of the “max_brightness” file, the brightness parameter goes from 0 to 255, and on some platforms you may have a PWM controlled LED with variable brightness.

Now, what’s that “trigger” file for?

Triggers

Triggers are the API used to link a LED to an event in kernel space.

That’s the idea: the platform registers some LED device, the drivers register some LED trigger, you configure the device to use a certain trigger and the led will be controlled from the appropriate driver.

To see the list of the available triggers, just read the “trigger” file:

```
balto@balto-mpc:/sys/class/leds/hdd# cat trigger
[none] nand-disk timer heartbeat gpio rfkill0 phy0rx
phy0tx phy0assoc phy0radio
```

you see that the trigger is disabled. To associate the LED with a trigger, just write its name in the “trigger” file:

```
balto@balto-mpc:/sys/class/leds/hdd# echo heartbeat > trigger
balto@balto-mpc:/sys/class/leds/hdd# cat trigger
none nand-disk timer [heartbeat] gpio rfkill0 phy0rx
phy0tx phy0assoc phy0radio
```

Now the LED should pulse like an heart with a period proportional to the system load average.

When you change trigger, check again the content of the LED directory, as many triggers also register some parameter files.

Hacking the kernel to add new triggers is pretty easy and good fun, just take some time and read a couple of trigger (http://lxr.linux.no/#linux/drivers/leds/ledtrig-heartbeat.c) source (http://lxr.linux.no/#linux/drivers/leds/ledtrig-timer.c) files (http://lxr.linux.no/#linux/drivers/leds/ledtrig-ide-disk.c) to see how they work!

Also, you can find the official documentation in the kernel’s “Documentation/leds” directory.

Happy blinkin’!

FILED UNDER LINUX TAGGED WITH BLINKING, EMBEDDED, LED, LINUX

About these ads (http://wordpress.com/about-these-ads/)

6 Responses to *Linux LED Subsystem*

O'Ray says:
2011/11/21 at 01:47
very interesting!

Reply

fabiobaltieri says:
2011/11/21 at 19:01
Thanks mate!

Reply

Denver says:
2013/01/08 at 05:43
Thanks for this article! It really helped me understand what the heck was going on in an Android app that I wrote that controls the brightness of the capacitive buttons backlight on the HTC One X. I’ve got a bunch of bug reports about weird behaviour and I definitely have a better understanding of what might be going on now. https://code.google.com/p/hox-cap-butn-brightness

Reply

taubo says:
2013/02/14 at 09:56
cercando “gpio leds ethernet activity” su google indovina qual’era il primo risultato? Credo che tu sia il maggior esperto mondiale al mondo di gpio leds al mondo

Reply

fabiobaltieri says:
2013/02/14 at 10:11
Mah ho paura dipenda da chi lo cerca… se lo googlo io sono il secondo. E io odio essere il secondo.

Reply

taubo says:
2013/02/14 at 11:23
looooooooooool