GPIO for Engineers and Makers

Linus Walleij, Linaro

GPIOLIB: What has happened since this (2006)?

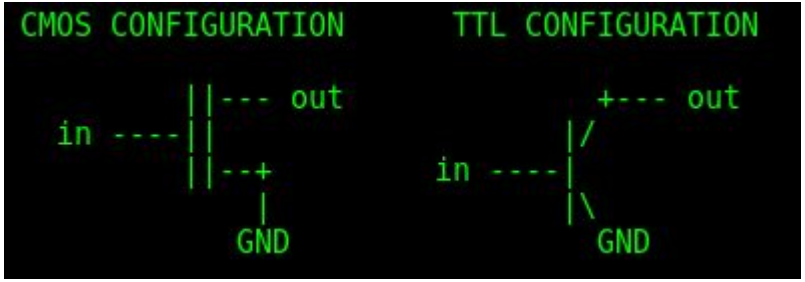
```
int gpio_request(unsigned gpio, const char *label)
int gpio_free(unsigned gpio)
int gpio_direction_input(unsigned gpio)
int gpio_direction_output(unsigned gpio)
int gpio_get_value(unsigned gpio)
int gpio_set_value(unsigned gpio, int value)
unsigned gpio_to_irq(unsigned gpio)
unsigned irq_to_gpio(unsigned irq)
```

Biggest lies:

- <insert favourite biggest lie>
- GPIO is simple

Kernel-internal API changes

- Just select GPIOLIB
- gpiochip_add_data()
- Descriptors
- A real device
- Open drain/open source API
- Pin control back-ends
- CONFIG_GPIOLIB_IRQCHIP
- Hogs



GPIO descriptors

```
#include <linux/gpio/consumer.h>

probe(struct device *dev)
{
    struct gpio_desc *gpiod;

    gpiod = devm_gpiod_get(dev, “reset”, GPIOD_OUT_LOW);
    (...)
}
```

GPIO descriptors

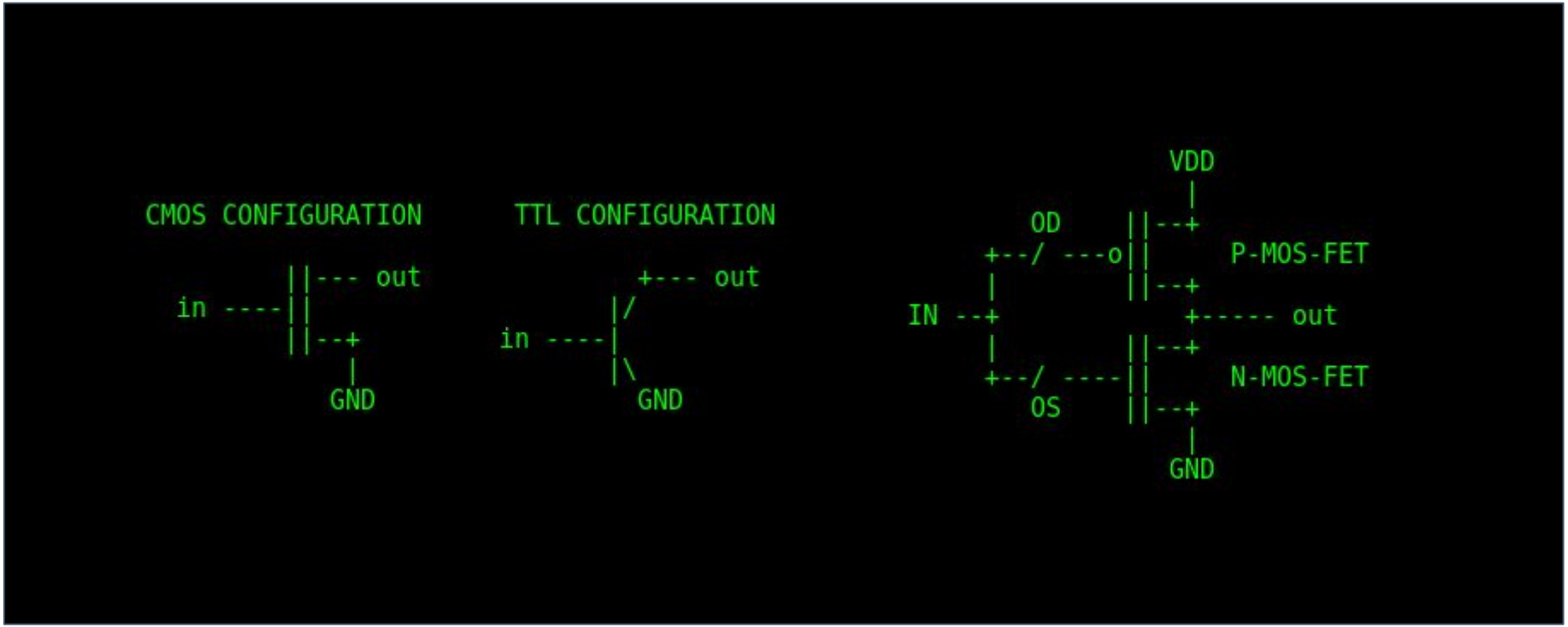
```
gpio_keys {
    compatible = “gpio-keys”;
    (...)
    button@1 {
        wakeup-source;
        linux,code = <KEY_ESC>;
        label = “ESC”;
        gpios = <&gpio0 0 GPIO_ACTIVE_HIGH>;
    };
};
```

The GPIO chip is a real device

```
struct gpio_device {
        int id;
        struct device dev;
        struct cdev chrdev;
(...)
};
```

~~/sys/class/gpio~~ /sys/bus/gpio

Open Drain and Open Source (CMOS)
Open Collector and Open Emitter (TTL)



Pin Control Back-Ends: GPIO Side

```
gpiochip_add_pin_range(struct gpio_chip *gc, const char
 *pinctl_name, unsigned gpio_offset, unsigned pin_offset,
 unsigned npins);
gpiochip_add_pingroup_range();
/* External interface to pin control */
extern int pinctrl_request_gpio(unsigned gpio);
extern void pinctrl_free_gpio(unsigned gpio);
extern int pinctrl_gpio_direction_input(unsigned gpio);
extern int pinctrl_gpio_direction_output(unsigned gpio);
```

Pin Control Back-Ends: Pin Controller Side

```
struct pinmux_ops {
      (...)
       int (*gpio_request_enable) (struct pinctrl_dev *pctldev,
                                   struct pinctrl_gpio_range *range,
                                   unsigned offset);
       void (*gpio_disable_free) (struct pinctrl_dev *pctldev,
                                  struct pinctrl_gpio_range *range,
                                  unsigned offset);
       int (*gpio_set_direction) (struct pinctrl_dev *pctldev,
                                  struct pinctrl_gpio_range *range,
                                  unsigned offset,
                                  bool input);
       bool strict;
};
```

CONFIG_GPIOLIB_IRQCHIP

(A)

```
gpiochip_ [lock|unlock]_as_irq(struct gpio_chip *gc, unsigned int offset);
```

(B)

```
Select GPIOLIB_IRQCHIP

#include <linux/gpio/driver.h>
(...)
gpiolib_irqchip_add(struct gpio_chip *gc, struct irq_chip, *ic, unsigned int
 first_irq, irq_flow_handler_t handler, unsigned int type);
(...)
gpiochip_set_chained_irqchip(struct gpio_chip *gc, struct irq_chip *ic, int
 parent_irq, irq_flow_handler_t parent_handler);
```

Hogs

```
gpio3: gpio@101e7000 {
        /* This hog will bias the MMC/SD card detect line */
        mmcsd-gpio {
                gpio-hog;
                gpios = <16 0x0>;
                output-low;
                line-name = "card detect bias";
        };
};
```

Kernel-external API changes

~~SYSFS~~
~~/sys/class/gpio~~

CHARDEV
/sys/bus/gpiochipN
/dev/gpiochipN

The Rules of Linux Userspace GPIO

1. You do not access GPIOs from userspace
2. YOU DO NOT ACCESS GPIOS FROM USERSPACE
3. Read Documentation/gpio/drivers-on-gpio.txt
4. Use the character device

Why is the sysfs ABI not working? (WorksForMe)

NO EXAMPLES

Features the Character Device Has

Discovery mechanism (not magic numbers)

Cleanup of resources on closing or crashing

Open Drain / Open Source

Get and set multiple lines at once

Good examples: tools/gpio/*

lsgpio

```
GPIO chip: gpiochip0, "pinctrl-bcm2835", 54 GPIO lines
        line  0: "[SDA0]" unused
        line  1: "[SCL0]" unused
(...)
        line 16: "STATUS_LED_N" unused
        line 17: "GPIO_GEN0" unused
        line 18: "GPIO_GEN1" unused
        line 19: "NC" unused
        line 20: "NC" unused
        line 21: "CAM_GPIO" unused
        line 22: "GPIO_GEN3" unused
        line 23: "GPIO_GEN4" unused
        line 24: "GPIO_GEN5" unused
        line 25: "GPIO_GEN6" unused
        line 26: "NC" unused
        line 27: "GPIO_GEN2" unused
(...)
```

Line Naming from the Device Tree

```
 &gpio {
+       /*
+        * Legend:
+        * "NC" = not connected (no rail from the SoC)
+        * "[FOO]" = pin is muxed for peripheral FOO (not GPIO)
+        * "" = No idea
+        * "FOO" = GPIO line named "FOO" on the schematic
+        * "FOO_N" = GPIO line named "FOO" on schematic, active low
+        */
+       gpio-line-names = "[SDA0]", "[SCL0]", "[SDA1]", "[SCL1]",
(...)
+                         "STATUS_LED_N",
+                         "GPIO_GEN0", "GPIO_GEN1",
+                         "NC", "NC", /* GPIO 19 & 20 unused */
+                         "CAM_GPIO",
+                         "GPIO_GEN3", "GPIO_GEN4",
+                         "GPIO_GEN5", "GPIO_GEN6",
```

Open the Character Device

```
#include <linux/gpio.h>
int fd = open(“/dev/gpiochip0”, 0);

struct gpiochip_info cinfo;
ret = ioctl(fd, GPIO_GET_CHIPINFO_IOCTL, &cinfo);
fprintf(stdout, "GPIO chip: %s, \"%s\", %u GPIO lines\n",
      cinfo.name, cinfo.label, cinfo.lines);

struct gpioline_info linfo;
ret = ioctl(fd, GPIO_GET_LINEINFO_IOCTL, &linfo);
fprintf(stdout, "line %2d: %s", linfo.line_offset,
 linfo.name);
```

Line Handles READ

```
struct gpiohandle_request req;
struct gpiohandle_data data;

req.lineoffsets[0] = 4;
req.lines = 1;
req.flags = GPIOHANDLE_REQUEST_INPUT;
strcpy(req.consumer_label, "pushbutton");
int lhfd = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
ret = ioctl(req.fd, GPIOHANDLE_GET_LINE_VALUES_IOCTL, &data);
printf("line 4 is %s\n", data[0] ? "high" : "low");
```

Line Handles WRITE

```
struct gpiohandle_request req;
struct gpiohandle_data data;

req.lineoffsets[0] = 4;
req.lines = 1;
req.flags = GPIOHANDLE_REQUEST_OUTPUT;
strcpy(req.consumer_label, "blinker");
int lhfd = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
data.values[0] = 1;
ret = ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
data.values[0] = 0;
ret = ioctl(req.fd, GPIOHANDLE_SET_LINE_VALUES_IOCTL, &data);
```

Line Events

```
struct gpioevent_request req;
struct gpiohandle_data data;
struct gpioevent_data event;

req.lineoffset = 4;
strcpy(req.consumer_label, "linewatcher");
req.handleflags = GPIOHANDLE_REQUEST_OPEN_DRAIN;
req.eventflags = GPIOEVENT_REQUEST_RISING_EDGE | GPIOEVENT_REQUEST_FALLING_EDGE;
ret = ioctl(fd, GPIO_GET_LINEEVENT_IOCTL, &req);
ret = read(req.fd, &event, sizeof(event));
printf( "GPIO EVENT @%" PRIu64 ": ", event.timestamp);
if (event.id == GPIOEVENT_EVENT_RISING_EDGE)
    printf("RISING EDGE");
else
    printf("FALLING EDGE");
printf ("\n");
```

Thank You!

linus.walleij@linaro.org
linux-gpio@vger.kernel.org