EVB335X-II（3.14 内核）实现 gpio-key 功能手册

Date：2015/10/30　　公开资料

类别	内容
关键词	EVB335X-II　3.14 内核　　gpio-key
摘要	EVB335X-II 3.14 添加轮询式，中断式 GPIO-KEY 功能，及按键超过 5s 向用户空间发送事件功能

深圳市盈鹏飞科技有限公司 www.embedall.com
电邮：info@embedall.com　　传真：0755-82523090
电话：0086-0755 – 82523090

目录

修订历史:

版本	日期	原因
V1.00	2014-10-30	创建文档

前言

本文主要讲解 EVB335X-II emmc 3.14 内核实现 gpio-key 功能，以 GPIO1_16(gpmc_a0)为例进行讲解。
实现以下 3 种方式：
1）轮询方式的 gpio 按键
2）中断方式的 gpio 按键
3）按下超过 5s 时，向用户空间发送事件功能。

1 轮询方式gpio按键的实现

1.1 修改.dts 文件

修改内核源码：arch/arm/boot/dts/evb335x-ii-emmc.dts ，如下红色字体所示：

```
#include "am33xx.dtsi"

/ {
    model = "EAC EVB335X-II";
    compatible = "ti,com335x", "ti,am33xx";

    cpus {
        cpu@0 {
            cpu0-supply = <&dcdc2_reg>;  /* mpu supply 1.1V */
        };
    };
……

    watchdog: watchdog {
        pinctrl-names = "default";
        pinctrl-0 = <&wdt_pins>;

        compatible = "wdt,isl88013";
        gpios = <&gpio3 15 GPIO_ACTIVE_LOW>;
    };

    backlight: backlight {
        pinctrl-names = "default";
        pinctrl-0 = <&backlight_pins>;

        compatible = "gpio-backlight";
        gpios = <&gpio3 17 GPIO_ACTIVE_HIGH>;
        default-on;
    };

/* gpio-keys-poll */
   gpio_keys_polled {
        compatible = "gpio-keys-polled";
        #address-cells = <1>;
        #size-cells = <0>;
```

```
        poll-interval = <100>;
        autorepeat;
        button@48{
            label = "GPIO Key UP";
            linux,code = <103>;
            gpios = <&gpio1 16 0>;
        };

    };

};
```

1.2 修改内核配置

修改内核配置如下所示：

Device Drivers ---->

Input device support ---->

-*- Polled input device skeleton

[*] Keyboards ---->

<*> AT keyboard

<*> GPIO Buttons

<*> Polled GPIO buttons

<*> TI TWL4030/TWL5030/TPS659x0 keypad support

重新编译内核，及.dts 文件，至此 evb335xii 添加轮询方式 gpio 按键完成；

1.3 测试按键功能

将光盘：emmc\customize\gpio_key\poll\test\ keybutton-poll 拷贝到开发板，输入命令：./ keybutton-poll，

按下按键即可观察键值等信息。

(注：开发板 CN21 的第 20 引脚为 gpio1_16 引脚，接高电平 3.3V 时模拟按键按下)

2 中断方式gpio按键的实现

2.1 修改.dts 文件

修改内核源码：arch/arm/boot/dts/evb335x-ii-emmc.dts ，如下红色字体所示：

```
#include "am33xx.dtsi"

/ {
    model = "EAC EVB335X-II";
    compatible = "ti,com335x", "ti,am33xx";

    cpus {
        cpu@0 {
            cpu0-supply = <&dcdc2_reg>;  /* mpu supply 1.1V */
        };
    };
……

    watchdog: watchdog {
        pinctrl-names = "default";
        pinctrl-0 = <&wdt_pins>;

        compatible = "wdt,isl88013";
        gpios = <&gpio3 15 GPIO_ACTIVE_LOW>;
    };

    backlight: backlight {
        pinctrl-names = "default";
        pinctrl-0 = <&backlight_pins>;

        compatible = "gpio-backlight";
        gpios = <&gpio3 17 GPIO_ACTIVE_HIGH>;
        default-on;
    };

   gpio_keys {
        compatible = "gpio-keys";
        #address-cells = <1>;
        #size-cells = <0>;
        autorepeat;
        button@21 {
```

```
            label = "GPIO Key UP";
            linux,code = <103>;
            gpios = <&gpio1 16 0>;
        };
    };

};
```

2.2 修改内核配置

修改内核配置如下所示：

Device Drivers --->

Input device support --->

-*- Polled input device skeleton

[*] Keyboards --->

<*> AT keyboard

<*> GPIO Buttons

<*> TI TWL4030/TWL5030/TPS659x0 keypad support

重新编译内核，及.dts 文件，至此 evb335xii 添加轮询方式 gpio 按键完成；

2.3 测试按键功能

将光盘：emmc\customize\gpio_key\interrupt\test\ keybutton-interrupt 拷贝到开发板，

输入命令：./ keybutton-interrupt，按下按键即可观察键值等信息。

(注：开发板 CN21 的第 20 引脚为 gpio1_16 引脚，接高电平 3.3V 时模拟按键按下)

3 按键按下 5s向用户空间发送事件功能实现

3.1 修改.dts 文件

修改内核源码：arch/arm/boot/dts/evb335x-ii-emmc.dts ，如下红色字体所示：

```
#include "am33xx.dtsi"

/ {
    model = "EAC EVB335X-II";
    compatible = "ti,com335x", "ti,am33xx";

    cpus {
        cpu@0 {
            cpu0-supply = <&dcdc2_reg>;  /* mpu supply 1.1V */
        };
    };
……

    watchdog: watchdog {
        pinctrl-names = "default";
        pinctrl-0 = <&wdt_pins>;

        compatible = "wdt,isl88013";
        gpios = <&gpio3 15 GPIO_ACTIVE_LOW>;
    };

    backlight: backlight {
        pinctrl-names = "default";
        pinctrl-0 = <&backlight_pins>;

        compatible = "gpio-backlight";
        gpios = <&gpio3 17 GPIO_ACTIVE_HIGH>;
        default-on;
    };

   gpio_keys {
        compatible = "gpio-keys";
        #address-cells = <1>;
        #size-cells = <0>;
        autorepeat;
        button@21 {
```

```
            label = "GPIO Key UP";
            linux,code = <103>;
            gpios = <&gpio1 16 0>;
        };
    };

};
```

3.2 修改内核配置

修改内核配置如下所示：

Device Drivers ---->

Input device support ---->

-*- Polled input device skeleton

[*] Keyboards ---->

<*> AT keyboard

<*> GPIO Buttons

<*> TI TWL4030/TWL5030/TPS659x0 keypad support

重新编译内核，及.dts 文件，至此 evb335xii 添加轮询方式 gpio 按键完成；

3.3 修改驱动代码

修改 drivers/input/keyboard/gpio-key.c 代码如下所示：

1）在 gpio_keys_drvdata 结构体后面添加如下代码：

```
struct gpio_keys_drvdata {
    const struct gpio_keys_platform_data *pdata;
    struct input_dev *input;
    struct mutex disable_lock;
    struct gpio_button_data data[0];
};
#include <linux/semaphore.h>
#include <linux/kthread.h>
static struct semaphore key_release;
static struct semaphore key_press;
```

```
//DECLARE_MUTEX(key_press);
//DECLARE_MUTEX(key_release);
static int prev_state = 0;
static int user_thread_handler(void *arg)
{
    int ret;
    char event_string[20];
    char *envp[] = { event_string, NULL };
    struct device *dev = (struct device *)arg;
    while(1)
    {
        down_interruptible(&key_press); // acquire the semaphore unless interrupted
        ret = down_timeout(&key_release, HZ*2 );// acquire the semaphore within 2s
        if(ret == 0)     /*  如果  2s 内接收到信号，则判定是短按键  */
        {
            sprintf(event_string, "EVENT=short-press");
            kobject_uevent_env(&dev->kobj, KOBJ_CHANGE, envp);
        }
        else
        {     /* we are waiting for long-press now */
            //  上面 2s+3s=5s，如果用户需要 10s,则将此处改为 HZ*8 即可
            ret = down_timeout(&key_release, HZ * 3);
            if(ret == -ETIME)   /* key always pressed */
            {
                sprintf(event_string, "EVENT=long-press");
                kobject_uevent_env(&dev->kobj, KOBJ_CHANGE, envp);
                if(prev_state)  //if key pressed,wait key release
                down_interruptible(&key_release);
            }
```

```
            }
        }
        return 0;
}
```

2）修改 gpio_keys_gpio_report_event 函数如下红色字体所示：

```
static void gpio_keys_gpio_report_event(struct gpio_button_data *bdata)
{
        const struct gpio_keys_button *button = bdata->button;
        struct input_dev *input = bdata->input;
        unsigned int type = button->type ?: EV_KEY;
        int state = (gpio_get_value_cansleep(button->gpio) ? 1 : 0) ^ button->active_low; //记录按键状态
        if(!state) /* key pressed */
        {
                up(&key_press);
                prev_state = 1;
        }else
        {
                if(prev_state)
                {
                        up(&key_release);
                        prev_state = 0;
                }
        }
        if (type == EV_ABS) {
                if (state)
                        input_event(input, type, button->code, button->value);
        } else {
                input_event(input, type, button->code, !!state);
        }
```

```
    input_sync(input);
}
```

3）在函数 gpio_keys_probe 中添加如下红色字体，如下所示：

```
static int gpio_keys_probe(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    const struct gpio_keys_platform_data *pdata = dev_get_platdata(dev);
    struct gpio_keys_drvdata *ddata;
    struct input_dev *input;
    int i, error;
    int wakeup = 0;
…….
    if (error) {
        dev_err(dev, "Unable to register input device, error: %d\n",
            error);
        goto fail3;
    }
    device_init_wakeup(&pdev->dev, wakeup);
    kthread_run(user_thread_handler, &input->dev/*NULL*/, "system button");
    return 0;
 fail3:
    sysfs_remove_group(&pdev->dev.kobj, &gpio_keys_attr_group);
 fail2:
    while (--i >= 0)
        gpio_remove_key(&ddata->data[i]);
 fail1:
    input_free_device(input);
    kfree(ddata);
```

```
    /* If we have no platform data, we allocated pdata dynamically. */
    if (!dev_get_platdata(&pdev->dev))
        kfree(pdata);
    return error;
}
```

4)修改 gpio_keys_init 函数如下红色字体所示：

```
static int __init gpio_keys_init(void)
{
    sema_init(&key_press,1);
    sema_init(&key_release,1);
    return platform_driver_register(&gpio_keys_device_driver);
}
```

重新编译内核及 dts 文件，至此，实现 5s 向用户空间发送事件功能完成；

3.4 测试按键功能

将光盘：emmc\customize\gpio_key\interrupt-5s\test\ main_key 拷贝到开发板，

输入命令：./ main_key，按下按键即可发送到的事件信息情况。主要观察是 EVENT=short-press,还是 EVENT=long-press

(注：开发板 CN21 的第 20 引脚为 gpio1_16 引脚，接高电平 3.3V 时模拟按键按下)

4 免责声明

本手册所陈述的产品文本及相关软件版权均属深圳盈鹏飞科技有限公司所有，其产权受国家法律保护，未经本公司授权，其它公司、单位、代理商及个人不得非法使用和拷贝。

您若需要我公司产品及相关信息，请及时与我们联系，我们将热情接待。

深圳盈鹏飞科技有限公司将会不断地完善本手册的相关技术内容，请客户适时从公司网站下载最新版本，不再另行通知。