iTOP-4412 开发板的触摸屏驱动

大家好，今天我们来学习下触摸屏驱动，iTOP-4412 开发板 9.7 寸和 7 寸的屏幕使用的是电容触摸屏，触摸 IC 是 ft5406 ,对应的驱动文件是 drivers/input/touchscreen/ft5x06_ts.c ,现在我们来分析下这个文件，首先是 ft5x0x_ts_init 函数，它是驱动加载首先执行的函数，代码如下：

```
static int __init ft5x0x_ts_init(void)
{
    int ret;

#if 1
    printk("==%s: reset==\n", __FUNCTION__);
        ret = gpio_request(EXYNOS4_GPX0(3), "GPX0_3");
        if (ret) {
                gpio_free(EXYNOS4_GPX0(3));

                ret = gpio_request(EXYNOS4_GPX0(3), "GPX0_3");
                if(ret)
                {
                        printk("ft5xox: Failed to request GPX0_3 \n");
                }
        }
```

```
        gpio_direction_output(EXYNOS4_GPX0(3), 0);
        mdelay(200);
        gpio_direction_output(EXYNOS4_GPX0(3), 1);

        s3c_gpio_cfgpin(EXYNOS4_GPX0(3), S3C_GPIO_OUTPUT);
        gpio_free(EXYNOS4_GPX0(3));
        msleep(10);
#endif

    return i2c_add_driver(&ft5x0x_ts_driver);
}
```

在这个函数里面主要实现的功能是通过 cpu 的一个 GPIO 引脚给触摸芯片复位，然后就是向内核注册 i2c 驱动，接下来在系统在枚举设备的时候会执行到 ft5x0x_ts_probe 函数，这个函数是整个驱动的初始化函数，下面我们来分析下这个函数，首先是

```
if (!i2c_check_functionality(client->adapter, I2C_FUNC_I2C)) {
        err = -ENODEV;
        goto exit_check_functionality_failed;
    }
```

这个函数的作用是检查一下设备是不是 i2c 设备，然后是

```
ts = kzalloc(sizeof(*ts), GFP_KERNEL);
    if (!ts) {
```

```
        err = -ENOMEM;
        goto exit_alloc_data_failed;
    }

    pdata = client->dev.platform_data;
    if (!pdata) {
        dev_err(&client->dev, "failed to get platform data\n");
        goto exit_no_pdata;
    }

    ts->screen_max_x = pdata->screen_max_x;
    ts->screen_max_y = pdata->screen_max_y;
    ts->pressure_max = pdata->pressure_max;
```

定义了一个 ft5x0x_ts_data 结构的变量 ts，并为其分配内存，并且给这个结构体赋值。

然后是：

```
ts->gpio_irq = pdata->gpio_irq;
    if (ts->gpio_irq != -EINVAL) {
        client->irq = gpio_to_irq(ts->gpio_irq);
    } else {
        goto exit_no_pdata;
    }
```

```
    if (pdata->irq_cfg) {
        s3c_gpio_cfgpin(ts->gpio_irq, pdata->irq_cfg);
        s3c_gpio_setpull(ts->gpio_irq, S3C_GPIO_PULL_NONE);
    }

    ts->gpio_wakeup = pdata->gpio_wakeup;
    ts->gpio_reset = pdata->gpio_reset;
```

获取到触摸的中断引脚，并配置这个引脚为中断模式。

接下来是：

```
INIT_WORK(&ts->work, ft5x0x_ts_pen_irq_work);
    this_client = client;
    i2c_set_clientdata(client, ts);

    ts->queue = create_singlethread_workqueue(dev_name(&client->dev));
    if (!ts->queue) {
        err = -ESRCH;
        goto exit_create_singlethread;
    }
```

创建了一个工作队列，用于读取触摸的数据，并上报给内核。

接下来是：

```
input_dev = input_allocate_device();
```

```
    if (!input_dev) {
        err = -ENOMEM;
        dev_err(&client->dev, "failed to allocate input device\n");
        goto exit_input_dev_alloc_failed;
    }

    ts->input_dev = input_dev;
```

上面是分配了一个 input_dev 结构.

接下来是：

```
set_bit(EV_SYN, input_dev->evbit);
    set_bit(EV_ABS, input_dev->evbit);
    set_bit(EV_KEY, input_dev->evbit);

#ifdef CONFIG_FT5X0X_MULTITOUCH
    set_bit(ABS_MT_TRACKING_ID, input_dev->absbit);
    set_bit(ABS_MT_TOUCH_MAJOR, input_dev->absbit);
    set_bit(ABS_MT_WIDTH_MAJOR, input_dev->absbit);
    set_bit(ABS_MT_POSITION_X, input_dev->absbit);
    set_bit(ABS_MT_POSITION_Y, input_dev->absbit);

    input_set_abs_params(input_dev, ABS_MT_POSITION_X, 0, ts->screen_max_x, 0, 0);
```

```
    input_set_abs_params(input_dev, ABS_MT_POSITION_Y, 0, ts->screen_max_y, 0, 0);
    input_set_abs_params(input_dev, ABS_MT_TOUCH_MAJOR, 0, ts->pressure_max, 0, 0);
    input_set_abs_params(input_dev, ABS_MT_WIDTH_MAJOR, 0, 200, 0, 0);
    input_set_abs_params(input_dev, ABS_MT_TRACKING_ID, 0, FT5X0X_PT_MAX, 0, 0);
#else
    set_bit(ABS_X, input_dev->absbit);
    set_bit(ABS_Y, input_dev->absbit);
    set_bit(ABS_PRESSURE, input_dev->absbit);
    set_bit(BTN_TOUCH, input_dev->keybit);

    input_set_abs_params(input_dev, ABS_X, 0, ts->screen_max_x, 0, 0);
    input_set_abs_params(input_dev, ABS_Y, 0, ts->screen_max_y, 0, 0);
    input_set_abs_params(input_dev, ABS_PRESSURE, 0, ts->pressure_max, 0 , 0);
#endif

    input_dev->name = FT5X0X_NAME;
    input_dev->id.bustype = BUS_I2C;
    input_dev->id.vendor = 0x12FA;
    input_dev->id.product = 0x2143;
    input_dev->id.version = 0x0100;
```

上面是设置 input_dev 的一些参数，用于通知输入子系统上报的数据类型以及方式。

接下来：

```
err = ft5x0x_read_fw_ver(&val);
    if (err < 0) {
        dev_err(&client->dev, "chip not found\n");
        goto exit_irq_request_failed;
    }
```

上面是读取触摸芯片的硬件信息。

接下来是：

```
err = request_irq(client->irq, ft5x0x_ts_interrupt,
            IRQ_TYPE_EDGE_FALLING /*IRQF_TRIGGER_FALLING*/, "ft5x0x_ts", ts);
    if (err < 0) {
        dev_err(&client->dev, "Request IRQ %d failed, %d\n", client->irq, err);
        goto exit_irq_request_failed;
    }


    disable_irq(client->irq);


dev_info(&client->dev, "Firmware version 0x%02x\n", val);
```

上面是向内核注册中断，然后先关闭中断。

接下来是：

```
#ifdef CONFIG_HAS_EARLYSUSPEND
```

```
    ts->early_suspend.level = EARLY_SUSPEND_LEVEL_DISABLE_FB + 1;
    ts->early_suspend.suspend = ft5x0x_ts_suspend;
    ts->early_suspend.resume = ft5x0x_ts_resume;
    register_early_suspend(&ts->early_suspend);
#endif

    enable_irq(client->irq);

    //cym 4412_set_ctp(CTP_FT5X06);
    dev_info(&client->dev, "FocalTech ft5x0x TouchScreen initialized\n");
    return 0;
```

上面代码是如果内核支持休眠唤醒功能，就注册休眠唤醒处理函数，然后使能中断。

到这里驱动的初始化函数就完成了，现在我们再来回顾一下初始化函数完成的主要功能：首先是把连接触摸芯片的 GPIO 设置成中断模式，然后向系统注册一个输入设备，接着注册中断处理。

下面我们再来看看中断处理：

```
static irqreturn_t ft5x0x_ts_interrupt(int irq, void *dev_id) {
    struct ft5x0x_ts_data *ts = dev_id;

    //printk("%s(%d)\n", __FUNCTION__, __LINE__);

    disable_irq_nosync(this_client->irq);
```

```
    if (!work_pending(&ts->work)) {
        queue_work(ts->queue, &ts->work);
    }

    return IRQ_HANDLED;
}
```

中断处理函数里面会启动工作队列，然后就返回了，这样的好处就是中断的处理时间短，具体的处理交给工作队列去完成。

接着我们在来看看工作队列函数：

```
static void ft5x0x_ts_pen_irq_work(struct work_struct *work) {
    struct ft5x0x_ts_data *ts = container_of(work, struct ft5x0x_ts_data, work);

    if (!ft5x0x_read_data(ts)) {
        ft5x0x_ts_report(ts);
    }

    enable_irq(this_client->irq);
}
```

这个函数的作用是读取触摸的数据，然后上报给内核的输入子系统。

其中函数 ft5x0x_read_data(ts)就是读取触摸的数据，下面来看下这个函数：

```
static int ft5x0x_read_data(struct ft5x0x_ts_data *ts) {
    struct ft5x0x_event *event = &ts->event;
    //u8 buf[32] = { 0 };
    u8 buf[64] = { 0 };
    int ret;

#ifdef CONFIG_FT5X0X_MULTITOUCH
    //ret = ft5x0x_i2c_rxdata(buf, 31);
    ret = ft5x0x_i2c_rxdata(buf, 63);
#else
    ret = ft5x0x_i2c_rxdata(buf, 7);
#endif
    if (ret < 0) {
        printk("%s: read touch data failed, %d\n", __func__, ret);
        return ret;
    }

    memset(event, 0, sizeof(struct ft5x0x_event));

    //event->touch_point = buf[2] & 0x07;
    event->touch_point = buf[2] & 0x0F;
```

```
    if (!event->touch_point) {
        ft5x0x_ts_release(ts);
        return 1;
    }
    //printk("point = %d\n", event->touch_point);
#ifdef CONFIG_FT5X0X_MULTITOUCH
    switch (event->touch_point) {
        case 10:
            event->x[9] = (s16)(buf[57] & 0x0F)<<8 | (s16)buf[58];
            event->y[9] = (s16)(buf[59] & 0x0F)<<8 | (s16)buf[60];
        case 9:
            event->x[8] = (s16)(buf[51] & 0x0F)<<8 | (s16)buf[52];
            event->y[8] = (s16)(buf[53] & 0x0F)<<8 | (s16)buf[54];
        case 8:
            event->x[7] = (s16)(buf[45] & 0x0F)<<8 | (s16)buf[46];
            event->y[7] = (s16)(buf[47] & 0x0F)<<8 | (s16)buf[48];
        case 7:
            event->x[6] = (s16)(buf[39] & 0x0F)<<8 | (s16)buf[40];
            event->y[6] = (s16)(buf[41] & 0x0F)<<8 | (s16)buf[42];
        case 6:
```

```
            event->x[5] = (s16)(buf[33] & 0x0F)<<8 | (s16)buf[34];
            event->y[5] = (s16)(buf[35] & 0x0F)<<8 | (s16)buf[36];
        case 5:
            event->x[4] = (s16)(buf[0x1b] & 0x0F)<<8 | (s16)buf[0x1c];
            event->y[4] = (s16)(buf[0x1d] & 0x0F)<<8 | (s16)buf[0x1e];
        case 4:
            event->x[3] = (s16)(buf[0x15] & 0x0F)<<8 | (s16)buf[0x16];
            event->y[3] = (s16)(buf[0x17] & 0x0F)<<8 | (s16)buf[0x18];
            //printk("x:%d, y:%d\n", event->x[3], event->y[3]);
        case 3:
            event->x[2] = (s16)(buf[0x0f] & 0x0F)<<8 | (s16)buf[0x10];
            event->y[2] = (s16)(buf[0x11] & 0x0F)<<8 | (s16)buf[0x12];
            //printk("x:%d, y:%d\n", event->x[2], event->y[2]);
        case 2:
            event->x[1] = (s16)(buf[0x09] & 0x0F)<<8 | (s16)buf[0x0a];
            event->y[1] = (s16)(buf[0x0b] & 0x0F)<<8 | (s16)buf[0x0c];
            //printk("x:%d, y:%d\n", event->x[1], event->y[1]);
        case 1:
            event->x[0] = (s16)(buf[0x03] & 0x0F)<<8 | (s16)buf[0x04];
            event->y[0] = (s16)(buf[0x05] & 0x0F)<<8 | (s16)buf[0x06];
            //printk("x:%d, y:%d\n", event->x[0], event->y[0]);
```

```
                break;
        default:
                printk("%s: invalid touch data, %d\n", __func__, event->touch_point);
                return -1;
    }
#else
    if (event->touch_point == 1) {
        event->x[0] = (s16)(buf[0x03] & 0x0F)<<8 | (s16)buf[0x04];
        event->y[0] = (s16)(buf[0x05] & 0x0F)<<8 | (s16)buf[0x06];
    }
#endif

    event->pressure = 200;

    return 0;
}
```

通过看这个函数的代码，我们可以知道 CPU 先通过 i2c 读出有几组数据（有几点按下）然后在根据有几点按下，通过 i2c 把这几点的数据读上来，并保存到全局变量 ts->event 里面。 然后是 ft5x0x_ts_report 函数：

```
static void ft5x0x_ts_report(struct ft5x0x_ts_data *ts) {
    struct ft5x0x_event *event = &ts->event;
```

```
    int x, y;
    int i;

#ifdef CONFIG_FT5X0X_MULTITOUCH
    for (i = 0; i < event->touch_point; i++) {

        event->x[i] = ts->screen_max_x - event->x[i];
        //event->y[i] = ts->screen_max_y - event->y[i];
#ifdef CONFIG_PRODUCT_SHENDAO
        event->y[i] = ts->screen_max_y - event->y[i];
#endif

        if (swap_xy) {
            x = event->y[i];
            y = event->x[i];
        } else {
            x = event->x[i];
            y = event->y[i];
        }

        if (scal_xy) {
```

```
                x = (x * ts->screen_max_x) / TOUCH_MAX_X;
                y = (y * ts->screen_max_y) / TOUCH_MAX_Y;
        }
        //printk("x = %d, y = %d\n", x, y);
        input_report_abs(ts->input_dev, ABS_MT_POSITION_X, x);
        input_report_abs(ts->input_dev, ABS_MT_POSITION_Y, y);

        input_report_abs(ts->input_dev, ABS_MT_PRESSURE, event->pressure);
        input_report_abs(ts->input_dev, ABS_MT_TOUCH_MAJOR, event->pressure);
        input_report_abs(ts->input_dev, ABS_MT_TRACKING_ID, i);

        input_mt_sync(ts->input_dev);
    }
#else
    if (event->touch_point == 1) {

        //event->x[0] = ts->screen_max_x - event->x[0];
        //event->y[0] = ts->screen_max_y - event->y[0];

        if (swap_xy) {
                x = event->y[0];
```

```
                y = event->x[0];
        } else {
                x = event->x[0];
                y = event->y[0];
        }

        if (scal_xy) {
                x = (x * ts->screen_max_x) / TOUCH_MAX_X;
                y = (y * ts->screen_max_y) / TOUCH_MAX_Y;
        }

        input_report_abs(ts->input_dev, ABS_X, x);
        input_report_abs(ts->input_dev, ABS_Y, y);
        input_report_abs(ts->input_dev, ABS_PRESSURE, event->pressure);
    }

    input_report_key(ts->input_dev, BTN_TOUCH, 1);
#endif

    input_sync(ts->input_dev);
}
```

该函数的作用就是把触摸的数据传递到内核的输入子系统，我们可以在这个函数里面加条打印信息，把触摸的坐标打印出来，以方便调试，例如上面代码里面红色的语句就是打印坐标的，把这个语句前面的“//”去掉，当我们点触摸屏的时候就会在串口看到坐标数据的输出，有的时候屏幕的坐标和触摸的坐标对不上，导致触摸不准，我们就可以在这个函数里对读上来的坐标进行转换以达到和屏幕坐标一致。

关于触摸的驱动大体主要就是完成初始化，在初始化的时候向内核注册输入设备，向内核注册中断，然后就是中断服务了。中断服务里面完成触摸数据的读取，上报触摸数据给内核的输入子系统。

至于其他的 i2c 的触摸驱动，大体框架也是这个流程，大家通过上面的学习主要是掌握触摸驱动的整个流程和框架，至于代码里面的具体细节，可以通过添加打印信息来调试。