

# Linux I<sup>2</sup>C Touch Device Driver

Version: V0.0.6  
Document: ILITEK\_LINUX\_I2C\_DRIVER.pdf

**ILI TECHNOLOGY CORP.**

8F, No.1, Taiyuan 2nd-St., Jhubei City, Hsinchu County 302, Taiwan, R.O.C.  
Tel.886-3-5600099; Fax.886-3-5600055  
<http://www.ilitek.com>

**目录**

<b>章节</b>	<b>页码</b>
1. 总体描述 .....	3
2. 头文件中相关宏的说明 .....	4
3. 部分代码说明 .....	7
4. 相关功能说明 .....	9
5. 常见问题 .....	14

## 1. 总体描述

- A. 这份文件对ILITEK\_LINUX\_I2C\_DRIVER进行说明。此版驱动将不同平台整合在一起，其实就是将不同平台能够共用的就共用，不能共用的会进行区分，因此写Makefile选择编译哪些文件时要注意，若MTK平台不是使用dts的方式需要开启NO\_USE\_MTK\_ANDROID\_SDK\_6\_UPWARD这个宏。同时在ilitek\_ts.h头文件中设定选择的平台（没有的可以直接设定ILITEK\_PLAT\_QCOM），以对应各平台的差异！设定方式如下：

```
000064:
00065: #define ILITEK_PLAT_QCOM
00066: #define ILITEK_PLAT_MTK
00067: #define ILITEK_PLAT_ROCKCHIP
00068: #define ILITEK_PLAT_ALLWIN
00069: #define ILITEK_PLAT_AMLOGIC
00070:
00071: #define ILITEK_PLAT
00072:
00073:
```

1  
2  
3  
4  
5

ILITEK\_PLAT\_QCOM

支持芯片型号	ILI230X、ILI231X、ILI251X、ILI2120
I2C 设备地址（7位）	0x41
开机自动升级	头文件包含ili档，或者使用bin档（针对ILI2120）
支持平台	Qcom、Rockchip、MTK(dts方式)、Allwinner、Amlogic，没有对应平台就以Qcom为base调试

- B. 各文件说明如下：

ilitek\_ts.h: 驱动头文件，包含驱动中要用到的一些头文件以及宏和函数的声明等。

ilitek\_platform\_init.c: 平台加载初始化需要的文件。

ilitek\_main.c: 驱动主文档，完成probe的具体实现，GPIO注册、读取TP信息、注册输入设备信息，报点，休眠唤醒处理等

ilitek\_update.c: 驱动升级IC固件功能的具体实现

ilitek\_tool.c: 用于支持TouchUtility apk 以及创建用命令升级固件、sensor test、查看固件版本的设备节点以及一些调试命令的使用等等

- C. 驱动移植说明：

1. 将 ilitek\_lim 文件夹复制到 kernel/drivers/input/touchscreen/（一般是放在这里，如果平台有指定则放到指定路径下）下面，
2. 在 kernel/drivers/input/touchscreen/Makefile里面添加一行obj-y += ilitek\_lim/，如果需要Kconfig,可自行写Kconfig文件
3. 根据具体平台选择所需编译的文件如下：

```
obj-y += ilitek_main.o \
        ilitek_platform_init.o \
        ilitek_update.o \
        ilitek_tool.o
```

4. 添加 I2C 设备：

对于使用 board file的方式，找到 kernel中初始化 I2C 总线的板级文件，若本驱动测试过的Tiny 4412

此文件为 linux-3.5/arch/arm/mach-exynos/mach-tiny4412.c, 添加内容如下:

```
00986: static struct i2c_board info initdata i2c_tpd={
00987:     I2C_BOARD_INFO("ilitek i2c", 0x41),
00988:     //.platform_data = &ilitek_pdata,
00989:
00990: };
                                名称      地址

02243:     i2c_register_board_info(2, &i2c_tpd, 1);
02244:
                                总线号
```

对于使用dts注册的方式, 可在dts文件对应I2C总线节点下加入如下参考内容:

```
ilitek@41 {
    compatible = "tchip,ilitek";
    reg = <0x41>;
    interrupt-parent = <&msm_gpio>;
    interrupts = <13 0x0>;
    vdd-supply = <&pm8916_l17>;
    vcc_i2c-supply = <&pm8916_l6>;
    ilitek,irq-gpio = <&msm_gpio 13 0x0>;
    ilitek,reset-gpio = <&msm_gpio 12 0x0>;
    ilitek,vbus = "vcc_i2c";
    ilitek,vdd = "vdd";
    ilitek,name = "ilitek_i2c";
};
```

## 2. 头文件中相关宏的说明

驱动版本信息, 不要修改第一码 (DERVER\_VERSION\_MAJOR)

//driver information

```
#define DERVER_VERSION_MAJOR      5
#define DERVER_VERSION_MINOR      0
#define CUSTOMER_ID               0
#define MODULE_ID                 0
#define PLATFORM_ID               0
#define PLATFORM_MODULE           0
#define ENGINEER_ID               0
```

平台设定, 选择对应的平台赋给 **ILITEK\_PLAT** 这个宏, 没有可选择 **QCOM**, 对应编译 **ilitek\_platform\_init.c** 这个文件, 如果使用平台有特定的设定, 则代码内相关部分按照使用平台的使用方式修改

```
#define ILITEK_PLAT_QCOM          1
#define ILITEK_PLAT_MTK          2
#define ILITEK_PLAT_ROCKCHIP     3
```

```
#define ILITEK_PLAT_ALLWIN 4
#define ILITEK_PLAT_AMLOGIC 5

#define ILITEK_PLAT ILITEK_PLAT_QCOM

#define ILITEK_TOOL
供调试工具所用，默认开启

#define ILITEK_TUNING_MESSAGE
供 FW 调试时丢出 debug 信息所用，默认开启

//define ILITEK_GLOVE
手套功能开关，主要针对 ILI2120，默认关闭

//define ILITEK_CHARGER_DETECTION
#define POWER_SUPPLY_BATTERY_STATUS_PATCH "/sys/class/power_supply/battery/status"
充电检测开关，主要针对 ILI2120，默认关闭

//define ILITEK_ESD_PROTECTION
ESD 保护开关，默认关闭

#define ILITEK_TOUCH_PROTOCOL_B
报点协议使用 B 类报点，默认开启

//define ILITEK_USE_LCM_RESOLUTION
使用 LCM 的分辨率，默认关闭

//define NO_USE_MTK_ANDROID_SDK_6_UPWARD //no use dts and for mtk old version
MTK 平台使用非 dts 方式需要开启此宏

#define ILITEK_ROTATE_FLAG 0
报点将 X、Y 调换，默认设为 0，启用时设为非 0 即可

#define ILITEK_REVERT_X 0
报点将 X 做镜像，即最大变最小，最小变最大，默认设 0，启用时设为非 0 即可

#define ILITEK_REVERT_Y 0
报点将 Y 做镜像，即最大变最小，最小变最大，默认设 0，启用时设为非 0 即可

#define ILITEK_ENABLE_REGULATOR_POWER_ON
```

使用 **regulator** 方式上电，当需要我们驱动来控制上电且是这种方式时可开启此宏，默认打开

### #define ILITEK\_GET\_GPIO\_NUM

当我们驱动能够解析获取 **reset**、**irq** 对应 **pin** 脚时，开启此宏，目前只有写用 **dtb** 方式解析，当不开此宏时需要设定 **ILITEK\_RESET\_GPIO** 和 **ILITEK\_IRQ\_GPIO** 为对应的值

#define ILITEK\_CLICK\_WAKEUP 0

单机唤醒（**driver** 实现），主要用于大尺寸

#define ILITEK\_DOUBLE\_CLICK\_WAKEUP 1

双击唤醒（**driver** 实现），主要用于大尺寸

#define ILITEK\_GESTURE\_WAKEUP 2

手势唤醒（**FW** 实现），主要用于小尺寸

//define ILITEK\_GESTURE ILITEK\_CLICK\_WAKEUP

设定手势唤醒方式，默认关闭此功能

//define ILITEK\_UPDATE\_FW

开机升级功能，前期调试时建议关闭，调试好后再开启测试，默认关闭

#define ILI\_UPDATE\_BY\_CHECK\_INT

升级时通过检测 **INT** 的状态来看 **IC** 是否 **ready** 好继续写下一笔数据，开此宏升级速度较快，针对大尺寸 **ILI2302** 和 **ILI2312**，默认开启

#define ILITEK\_UPGRADE\_WITH\_BIN 0

升级档案用 **bin** 档，且是放在系统对应 **firmware** 文件夹下（一般为 **/system/etc/firmware/**，或自己定义的路径），针对 **ILI2120**

#define ILITEK\_FW\_FILENAME "ilitek\_i2c.bin"

升级档案用 **bin** 档的档案名设定

#define TOUCH\_SCREEN\_X\_MAX (1080) //LCD\_WIDTH

#define TOUCH\_SCREEN\_Y\_MAX (1920) //LCD\_HEIGHT

LCD 的分辨率设定，当 **MTK** 平台且我们代码有开启 **ILITEK\_USE\_MTK\_INPUT\_DEV** 或有开启

**ILITEK\_USE\_LCM\_RESOLUTION** 这个宏时要设定正确

#define ILITEK\_USE\_MTK\_INPUT\_DEV

针对 **MTK** 平台使用平台内的 **tpd->dev**，默认开启，当报点有异常时可以关掉此宏看看，默认开启

#if defined ILITEK\_GET\_GPIO\_NUM

#undef ILITEK\_GET\_GPIO\_NUM

#endif

针对 **MTK** 平台不需要解析 **reset**、**irq** 对应 **pin** 脚时关掉 **ILITEK\_GET\_GPIO\_NUM** 这个宏，注意设定

ILITEK\_RESET\_GPIO 和 ILITEK\_IRQ\_GPIO 为对应的值

```
#define ILITEK_ERR_LOG_LEVEL (1)
```

```
#define ILITEK_INFO_LOG_LEVEL (3)
```

```
#define ILITEK_DEBUG_LOG_LEVEL (4)
```

```
#define ILITEK_DEFAULT_LOG_LEVEL (3)
```

```
#define debug_level(level, fmt, arg...) do {\n    if (level <= ilitek_log_level_value) {\n        if (level == ILITEK_ERR_LOG_LEVEL) {\n            printk(" %s ERR   line = %d %s : "fmt, "ILITEK", __LINE__, __func__, ##arg);\n        }\n        else if (level == ILITEK_INFO_LOG_LEVEL) {\n            printk(" %s INFO line = %d %s : "fmt, "ILITEK", __LINE__, __func__, ##arg);\n        }\n        else if (level == ILITEK_DEBUG_LOG_LEVEL) {\n            printk(" %s DEBUG line = %d %s : "fmt, "ILITEK", __LINE__, __func__, ##arg);\n        }\n    }\n}\n\n} while (0)
```

```
#define tp_log_err(fmt, arg...) debug_level(ILITEK_ERR_LOG_LEVEL, fmt, ##arg)
```

```
#define tp_log_info(fmt, arg...) debug_level(ILITEK_INFO_LOG_LEVEL, fmt, ##arg)
```

```
#define tp_log_debug(fmt, arg...) debug_level(ILITEK_DEBUG_LOG_LEVEL, fmt, ##arg)
```

log 打印相关，等级设定与 tp\_log\_err、tp\_log\_info、tp\_log\_debug 对应，ILITEK\_DEFAULT\_LOG\_LEVEL 默认打印等级，低于或等于此等级的都会打印，log 关键字 “ILITEK”

### 3. 部分代码说明

```
int ilitek_power_on(bool status)
```

当有开启 ILITEK\_ENABLE\_REGULATOR\_POWER\_ON 这个宏时才有具体实现，对应的 ilitek\_data->vdd 或 ilitek\_data->vdd\_i2c 在对应的平台初始化代码内有实现，若是其他方式按具体方式修改

```
int ilitek_get_gpio_num(void)
```

会去获取对应的 reset、irq gpio，若不需要获取则注意设定 ILITEK\_RESET\_GPIO 和 ILITEK\_IRQ\_GPIO 为对应的值

```
int ilitek_request_gpio(void)
```

申请 gpio，当申请失败时会先 free 然后再 try 一次，申请成功后会设定 reset 输出高，irq 为输入，关于 GPIO 的操作请以具体平台操作方式为准

reset 函数，delay 为从拉低到拉高后的延时时间，需要大于 IC 的初始化时间，之前有发现 MTK 平台用 tpd\_gpio\_output 这个接口不能正常拉高拉低，可以改为 gpio\_direction\_output 测试看看

```
void ilitek_reset(int delay) {
    tp_log_info("delay = %d\n", delay);
    if (ilitek_data->reset_gpio > 0) {
        #if ILITEK_PLAT != ILITEK_PLAT_MTK
            gpio_direction_output(ilitek_data->reset_gpio, 1);
            mdelay(10);
            gpio_direction_output(ilitek_data->reset_gpio, 0);
            mdelay(10);
            gpio_direction_output(ilitek_data->reset_gpio, 1);
            mdelay(delay);
        #else
            tpd_gpio_output(ilitek_data->reset_gpio, 1);
            mdelay(10);
            tpd_gpio_output(ilitek_data->reset_gpio, 0);
            mdelay(10);
            tpd_gpio_output(ilitek_data->reset_gpio, 1);
            mdelay(delay);
        #endif
    }
    else {
        tp_log_err("reset pin is invalid\n");
    }
    return;
}
```

```
int ilitek_read_tp_info(void)
```

1.此函数一开始会下 **0x61** 命令来查看是哪颗 IC，从而选择不同的流程，主要区分 **ILI2120** 和非 **ILI2120**，若发现此处得到的信息没有正确区分可手动改为需要的值，即设定 **ilitek\_data->ic\_2120** 为 **true** 或 **false**

2.当判断到 IC 为 **ILI2511** 时会将 **ilitek\_repeat\_start** 置为 **false**

3.对于 **ILI2120** 若 **0x10** 命令读到的第二 **byte** 数据低于 **0x80** 则会将强制升级的 **flag** 置起来，对于大尺寸 IC 当 **0xC0** 命令读到数据为 **0x55**（即 **BL** 模式）是会将强制升级的 **flag** 置起来，强制升级 **flag**：

**ilitek\_data->force\_update**

4.存放按键信息的 **keyinfo** 这个数组设定大小为 **10**，当按键数大于 **10** 时需在 **struct ilitek\_ts\_data** 结构体内修改成员 **keyinfo** 的大小

```
static int ilitek_request_irq(void)
```

```
#if ILITEK_PLAT != ILITEK_PLAT_MTK
```

```
    ilitek_data->client->irq = gpio_to_irq(ilitek_data->irq_gpio);
```

```
#else
```

```
    node = of_find_matching_node(NULL, touch_of_match);
```



```
#endif
```

## 4. 相关功能说明

### A. 开机升级功能

- ```
ilitek data->force update
```

- #### 4. 是否升级判定方式

```
tp_log_info("ilitek dt_startaddr=0x%X, dt_endaddr=0x%X, dt_checksum=0x%X, dt_len = 0x%d\n", dt_startaddr, dt_endaddr, dt_checksum, dt_len);
if (!ilitek_data->force_update) {
    for (i = 0; i < 8; i++) {
        tp_log_info("ilitek_data.firmware_ver[%d] = %d, firmware_ver[%d] = %d\n", i, ilitek_data->firmware_ver[i], i, firmware_ver[i]);
        if (!ilitek_data->ic_2120) {
            if (firmware_ver[i] < ilitek_data->firmware_ver[i]) {
                i = 8;
                break;
            }
            if (firmware_ver[i] > ilitek_data->firmware_ver[i]) {
                break;
            }
        } else {
            if (firmware_ver[i] != ilitek_data->firmware_ver[i]) {
                break;
            }
        }
    }
}
if (i >= 8) {
    if (!ilitek_data->ic_2120) {
        tp_log_info("firmware version is older so not upgrade\n");
    }
    else {
        tp_log_info("firmware version is same so not upgrade\n");
        if (ILITEK_UPGRADE_WITH_BIN) {
            if (CTPM_FW_BIN) {
                vfree(CTPM_FW_BIN);
                CTPM_FW_BIN = NULL;
            }
        }
    }
}
return 1;
}
} ? end if !ilitek_data->force ... ?
```

- ## 5. 升级流程选择

```

if (!ilitek_data->ic_2120) {
    ret = ilitek_upgrade_bigger_size_ic(df_startaddr, df_endaddr, df_c
} else {
    ret = ilitek_upgrade_2120(CTPM_FW);
}

if ((ilitek_data->mcu_ver[0] == 0x11 || ilitek_data->mcu_ver[0] == 0x10) && ilitek_data->mcu_ver[1] == 0x25) {
    df_startaddr = 0xF000;
    ret = ilitek_upgrade_2511(df_startaddr, df_endaddr, ap_startaddr, ap_endaddr, CTPM_FW);
    if (ret < 0) {
        tp_log_err("ilitek_upgrade_2511 err ret = %d\n", ret);
        goto Retry;
    }
} else {
    df_startaddr = 0x1F000;
    if (df_startaddr < df_endaddr) {
        ilitek_data->has_df = true;
    } else {
        ilitek_data->has_df = false;
    }
    ret = ilitek_upgrade_2302or2312(df_startaddr, df_endaddr, df_checksum, ap_startaddr, ap_endaddr, ap_checksum, CTPM_FW);
    if (ret < 0) {
        tp_log_err("ilitek_upgrade_2302or2312 err ret = %d\n", ret);
        goto Retry;
    }
}
}

```

## B. 手势、单双击唤醒功能

**#define ILITEK\_CLICK\_WAKEUP** 0 //单击唤醒

**#define ILITEK\_DOUBLE\_CLICK\_WAKEUP** 1 //双击唤醒

**#define ILITEK\_GESTURE\_WAKEUP** 2 //手势唤醒，针对 FW 内部做手势判断的方式

开启此功能要确保休眠时 TP 没有下电

大尺寸有些客户需要点击唤醒或双击唤醒功能时需要如下设置

**#define ILITEK\_GESTURE** ILITEK\_CLICK\_WAKEUP //点击唤醒

**#define ILITEK\_GESTURE** ILITEK\_DOUBLE\_CLICK\_WAKEUP //双击唤醒

点击唤醒功能，必须在点击抬起后执行唤醒动作

双击唤醒参数说明：

**#define DOUBLE\_CLICK\_DISTANCE** 1000 //两次点击坐标的最大距离

**#define DOUBLE\_CLICK\_ONE\_CLICK\_USED\_TIME** 800 //一次点击所用最长时间，单位 ms

**#define DOUBLE\_CLICK\_NO\_TOUCH\_TIME** 1000 //两次点击中间间隔时间，单位 ms

**#define DOUBLE\_CLICK\_TOTAL\_USED\_TIME** (DOUBLE\_CLICK\_NO\_TOUCH\_TIME + (DOUBLE\_CLICK\_ONE\_CLICK\_USED\_TIME \* 2)) //双击总时间

对于系统上层进行设置手势功能是否启用的接口在/sys/touchscreen/gesture，具体实现如下：

```

00191: #ifndef ILITEK_GESTURE
00192: static ssize_t ilitek_gesture_show(struct device *dev,
00193: struct device_attribute *attr, char *buf) {
00194:     if (ilitek_data->enbale_gesture) {
00195:         return sprintf(buf, "gesture: on\n");
00196:     }
00197:     else {
00198:         return sprintf(buf, "gesture: off\n");
00199:     }
00200: }
00201: static ssize_t ilitek_gesture_store(struct device *dev,
00202: struct device_attribute *attr, const char *buf, size_t size) {
00203:     if (buf[0]) {
00204:         ilitek_data->enbale_gesture = true;
00205:     }
00206:     else {
00207:         ilitek_data->enbale_gesture = false;
00208:     }
00209:     return size;
00210: }
00211: static DEVICE_ATTR(gesture, S_IRWXUGO, ilitek_gesture_show, ilitek_gesture_store);
00212: #endif

```

若与客户使用接口不对应，可修改此处以及节点路径

## C. 手套功能

宏开关

#define ILITEK\_GLOVE //默认关闭

具体接口实现

```
00233: static ssize_t ilitek_glove_show(struct device *dev,
00234:     struct device_attribute *attr, char *buf) {
00235:     if (ilitek_data->enbale_glove) {
00236:         return sprintf(buf, "glove: on\n");
00237:     }
00238:     else { 读/sys/touchscreen/glove节点即可获取手套开关状态
00239:         return sprintf(buf, "glove: off\n");
00240:     }
00241: }
00242:
00243: static ssize_t ilitek_glove_store(struct device *dev,
00244:     struct device_attribute *attr, const char *buf, size_t size) {
00245:     if (buf[0]) {
00246:         ilitek_data->enbale_glove = true;
00247:     }
00248:     else { 系统上层手套开关节点/sys/touchscreen/glove
00249:           写入第一byte数据为非0即开启手套功能, 否则关闭手套功能
00250:         ilitek_data->enbale_glove = false;
00251:     }
00252:     ilitek_into_glovemode(ilitek_data->enbale_glove);
00253:     return size;
00254: }
00255: static DEVICE_ATTR(glove, S_IRWXUGO, ilitek_glove_show, ilitek_glove_store);
```

唤醒时若手套功能是开启的则会下命令进入手套模式

若与客户使用接口不对应, 可修改此处以及节点路径

#### D. 充电检测

宏开关

#define ILITEK\_CHARGER\_DETECTION //默认关闭

开启此宏后会创建一个工作队列, 每间隔按照设定的时间来看是否有在进行充电从而下命令让 IC 进入不同模式, 可修改 ilitek\_data->charge\_delay 来改变间隔时间, 唤醒时会去检测

检测是否有充电的方式为读取如下文件内容来判断是否有充电, 若此方式不能正常获取充电状态则按具体情况处理

#define POWER\_SUPPLY\_BATTERY\_STATUS\_PATCH "/sys/class/power\_supply/battery/status"

#### E. ESD 检测

宏开关

#define ILITEK\_ESD\_PROTECTION //默认关闭

开启此宏后会创建一个工作队列, 每间隔按照设定的时间来检测 IC 是否有异常, 若有异常则进行 reset (或者加入上下电), 可修改 ilitek\_data->esd\_delay 来改变间隔时间

预设检测方式为: 下命令读取数据 (预设 0x42 获取 protocol), 会 retry 3 次, 若 3 次都失败则 reset

具体实现如下:

## F. 通过命令升级固件

- upgrade successfull ilitek firmware version is 6.0.0.0.1.2.255.255

[illegible]

同时会将数据保存在默认路径下面"/data/local/tmp/"，文件名以"ilitek\_sensortest"开头

## 2. 设定卡控范围、数据保存路径，数据显示 flag

```

if (! (iitek_data > ic_2120)) {
    ret = sscanf(buf, "%d,%d,%d,%d,%d,%d,%d,%d,%s", &iitek_short_threshold, &iitek_open_threshold, &iitek_open_txdatathreshold, &iitek_allnode_max_threshold, &iitek_allnode_min_threshold, &iitek_allnodetestw1, &iitek_allnodetestw2, &iitek_allnodemodule, &iitek_printsensortestdata, sensor_test_data_path);
    tp_log_info("iitek_short_threshold = %d\n", iitek_short_threshold);
    tp_log_info("iitek_open_threshold = %d\n", iitek_open_threshold);
    tp_log_info("iitek_open_txdatathreshold = %d\n", iitek_open_txdatathreshold);
    tp_log_info("iitek_allnode_max_threshold = %d\n", iitek_allnode_max_threshold);
    tp_log_info("iitek_allnode_min_threshold = %d\n", iitek_allnode_min_threshold);
    tp_log_info("iitek_allnodetestw1 = %d\n", iitek_allnodetestw1);
    tp_log_info("iitek_allnodetestw2 = %d\n", iitek_allnodetestw2);
    tp_log_info("iitek_allnodemodule = %d\n", iitek_allnodemodule);
    tp_log_info("iitek_allnodetx = %d\n", iitek_allnodetx);
}
else {
    ret = sscanf(buf, "%d,%d,%d,%d,%s", &iitek_short_threshold, &iitek_open_threshold, &iitek_allnode_max_threshold, &iitek_allnode_min_threshold, &iitek_printsensortestdata, sensor_test_data_path);
    tp_log_info("iitek_short_threshold = %d\n", iitek_short_threshold);
    tp_log_info("iitek_open_threshold = %d\n", iitek_open_threshold);
    tp_log_info("iitek_allnode_max_threshold = %d\n", iitek_allnode_max_threshold);
    tp_log_info("iitek_allnode_min_threshold = %d\n", iitek_allnode_min_threshold);
}
tp_log_info("iitek_printsensortestdata = %d\n", iitek_printsensortestdata);
tp_log_info("sensor_test_data_path = %s\n", sensor_test_data_path);
}

```

非 ILI2120 卡控范围设定方式

ILI2120卡控范围设定方式

打印及显示flag

保存sensor test 数据的路径, 推荐/data/local/tmp/

参考如下：

非 ILI2120

```
echo 7,400,120,120,4800,3700,120,120,20,20,1,/data/local/tmp/ > /proc/ilitek/sensor_test_data
```

ILI2120

```
echo 7,3500,3500,1800,1,/data/local/tmp/ > /proc/ilitek/sensor_test_data
```

3. 对于客户上层调用的方式，上层可以只读取 5 bytes 内容，来看读到的是 pass 还是 fail

代码会判断测试是否 pass 来丢出 pass 或 fail

```
if (short_test_result == 0 && open_test_result == 0 && allnode_test_result == 0) {
    seq_printf(m, "pass\n");
}
else {
    seq_printf(m, "fail\n");
}
```

## H. NoiseFre 功能

1. 直接 `cat /proc/ilitek/noisefre_data` 这个节点即可（使用默认的参数），丢出数据如下

```
shell@msm8916_64:/ # cd /proc/ilitek
cd /proc/ilitek
shell@msm8916_64:/proc/ilitek # cat no*
cat no*
0300, 0305, 0310, 0315, 0320, 0325, 0330, 0335, 0340, 0345, 0350, 0355, 0360, 0365, 0370, 0375, 0380, 0385, 0390, 0395,
0005, 0005, 0005, 0005, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0005, 0005,
0400, 0405, 0410, 0415, 0420, 0425, 0430, 0435, 0440, 0445, 0450, 0455, 0460, 0465, 0470, 0475, 0480, 0485, 0490, 0495,
0005, 0005, 0005, 0004, 0004, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0004, 0005, 0004, 0005, 0005, 0005, 0005,
0500, 0505, 0510, 0515, 0520, 0525, 0530, 0535, 0540, 0545, 0550, 0555, 0560, 0565, 0570, 0575, 0580, 0585, 0590, 0595,
0005, 0005, 0005, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0004, 0005, 0005, 0005, 0004, 0004, 0004, 0005,
0600, 0605, 0610, 0615, 0620, 0625, 0630, 0635, 0640, 0645, 0650, 0655, 0660, 0665, 0670, 0675, 0680, 0685, 0690, 0695,
0005, 0004, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0004, 0005, 0005, 0004, 0005, 0004, 0005,
0700, 0705, 0710, 0715, 0720, 0725, 0730, 0735, 0740, 0745, 0750, 0755, 0760, 0765, 0770, 0775, 0780, 0785, 0790, 0795,
0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0005, 0005,
0800, 0805, 0810, 0815, 0820, 0825, 0830, 0835, 0840, 0845, 0850, 0855, 0860, 0865, 0870, 0875, 0880, 0885, 0890, 0895,
0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0004, 0004, 0004, 0005, 0005, 0005, 0005, 0005, 0005, 0004, 0005, 0005,
0900, 0905, 0910, 0915, 0920, 0925, 0930, 0935, 0940, 0945, 0950, 0955, 0960, 0965, 0970, 0975, 0980, 0985, 0990, 0995,
0005, 0004, 0005, 0005, 0004, 0005, 0005, 0005, 0005, 0005, 0004, 0004, 0005, 0004, 0005, 0005, 0004, 0004, 0005, 0005,
1000, 1005, 1010, 1015, 1020, 1025, 1030, 1035, 1040, 1045, 1050, 1055, 1060, 1065, 1070, 1075, 1080, 1085, 1090, 1095,
0004, 0005, 0005, 0005, 0004, 0005, 0004, 0005, 0005, 0004, 0005, 0005, 0004, 0005, 0004, 0005, 0005, 0005, 0005, 0004,
1100, 1105, 1110, 1115, 1120, 1125, 1130, 1135, 1140, 1145, 1150, 1155, 1160, 1165, 1170, 1175, 1180, 1185, 1190, 1195,
0004, 0004, 0005, 0005, 0004, 0004, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005, 0005,
1200,
0005,
```

2. 修改起始和结束频率以及跳动大小方式，操作如下：

```
echo 30,120,5,/data/local/tmp/ > /proc/ilitek/noisefre_data
```

30,120,5,/data/local/tmp/ → 起始频率，结束频率，跳动大小，数据保存路径

- I. 针对中大尺寸 debug 信息是 ASCII 的方式使用 debug 节点直接看 debug 信息

1. echo dbg\_flag > /proc/ilitek\_debug //此命令用来打开或关闭此功能的 flag，每下一次是上一次的非动作
2. cat /proc/ilitek\_debug 这命令执行下去后若有 debug 信息就会印出来
3. 当不看时记得再下一次 echo dbg\_flag > /proc/ilitek\_debug

- J. 获取固件版本

直接下命令 cat /proc/ilitek/firmware\_version 即可丢出如下信息

```
ilitek firmware version is 6.0.0.0.1.2.255.255
```

对于客户上层获取版本号的需求可直接读取此节点或/sys/touchscreen/firmware\_version

## 5. 常见问题

- A. 驱动不会进入 probe 函数

对于使用 board file 方式注册的查看 ILITEK\_TS\_NAME 和注册的 I2C 设备名称是否一致，此处必须一致

对于使用 dts 方式注册的查看.of\_match\_table = ilitek\_touch\_match\_table 中

ilitek\_touch\_match\_table 内 compatible 是否和 dts 注册中的 compatible 匹配，此处必须匹配

## B. 通信不通

1. 软件上的只有 I2C 总线号及地址会影响到通信，软件配置确保这两项是 OK 的
2. 硬件上首先确认 IC 电这块是否 OK
3. 抓取波形确认是否满足通信协议
4. 用其他器件通信是否 OK，可尝试把此总线上的其他设备都卸掉测试

## C. 报点问题

1. 有触摸效果，只是坐标 mapping 问题
  - i. X、Y 需要交换 → 将 ILITEK\_ROTATE\_FLAG 设定值由 0 改为 1 或由 1 改为 0
  - ii. X、Y 值要做镜像变化即最大变最小 → 将 ILITEK\_REVERT\_X 或 ILITEK\_REVERT\_Y 的设定值由 0 改为 1 或由 1 改为 0
  - iii. 若需要使用显示屏的分辨率，则开启#define ILITEK\_USE\_LCM\_RESOLUTION 这个宏，同时将 TOUCH\_SCREEN\_X\_MAX 和 TOUCH\_SCREEN\_Y\_MAX 设为正确值
2. 触摸没反应
  - i. 确认中断是否注册成功，同步确认中断号是否正确
  - ii. 通过 log 确认触摸时是否有中断响应，若有中断响应，则可将 ilitek\_read\_data\_and\_report\_2120 或者 ilitek\_read\_data\_and\_report\_3XX 内收到的数据打印出来看数据是否正确
  - iii. 抓取触摸时 INT 的波形确认是否有正常拉高拉低的动作

## Revision History

| Version No. | Date       | Page | Description                                                                                                                                                            |
|-------------|------------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0.0.1       | 2011/03/07 | All  | Firstly release                                                                                                                                                        |
| 0.0.2       | 2011/05/12 | 3    | Modified driver file name.                                                                                                                                             |
| 0.0.3       | 2011/09/30 | 3    | Modified version id                                                                                                                                                    |
| 0.0.4       | 2012/11/26 | 3    | Method of adding idc files                                                                                                                                             |
| 0.0.5       | 2017/07/14 | 15   | Modified driver structure                                                                                                                                              |
| 0.0.6       | 2017/09/12 | 16   | <ol style="list-style-type: none"> <li>1. MTK 平台支援非 dts 方式</li> <li>2. 针对中大尺寸添加 debug 信息节点</li> <li>3. 升级前查看 hex 档是否匹配</li> <li>4. 针对 Intel 平台添加对应 match 方式</li> </ol> |
|             |            |      |                                                                                                                                                                        |
|             |            |      |                                                                                                                                                                        |