<http://blog.csdn.net/wh_19910525/article/details/10828025>

[android 电容屏（四）：驱动调试之驱动程序分析篇 -- FocalTech](http://blog.csdn.net/wh_19910525/article/details/10828025)

2013-09-01 15:56 6906人阅读 [评论](http://blog.csdn.net/wh_19910525/article/details/10828025#comments)(3) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/wh_19910525/article/details/10828025#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

touch（5） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg linux（132） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

        本人用的触摸屏IC是FocalTech公司的ft5306，是一款i2c的电容屏多点触控芯片。对于它的整体驱动官方已经给了，我们就触摸屏和按键部分的代码做相关说明。说明其中应该注意的地方。

对于所有的input设备，报告input事件时候都分这么几部分，首先在probe文件中设置设备发送的事件类型、按键类型、设置设备一些属性信息。然后在发送事件时候要根据probe的设置来发送事件，否则就会被判为无效忽略掉。

一、触摸屏部分

1.设备配置

对于触摸屏，必须支持的事件类型有以下这么三个

\_\_set\_bit(EV\_SYN, input\_dev->evbit);  //设备同步，每次触摸完成以后都要发送一个同步事件，来表明这次触摸已经完成  
\_\_set\_bit(EV\_ABS, input\_dev->evbit); //绝对坐标事件，触摸屏每次发送的坐标都是绝对坐标，不同于鼠标的相对坐标  
\_\_set\_bit(EV\_KEY, input\_dev->evbit); //按键事件，每次触摸都有一个BTN\_TOUCH的按键事件

触摸屏必须支持的按键类型

\_\_set\_bit(BTN\_TOUCH, input\_dev->keybit);//touch类型按键  
触摸屏属性设置  
input\_mt\_init\_slots(input\_dev, CFG\_MAX\_TOUCH\_POINTS);//报告最大支持的点数  
input\_set\_abs\_params(input\_dev,ABS\_MT\_TOUCH\_MAJOR, 0, PRESS\_MAX, 0, 0);//将触摸点看成一个椭圆，它的长轴长度。这个是可选项，并不影响正常使用。  
input\_set\_abs\_params(input\_dev, ABS\_MT\_POSITION\_X, 0, ft5x0x\_ts->x\_max, 0, 0);//x坐标取值范围  
input\_set\_abs\_params(input\_dev, ABS\_MT\_POSITION\_Y, 0, ft5x0x\_ts->y\_max, 0, 0);//y坐标取值范围

2.事件发送

  我们知道每次触摸完成后都必须发送一个同步事件（EV\_SYN）来表明这次触摸的完成。 那么对于多点触控的屏幕事件发送分为两种方法，一是每次事件同步前包括多个点，一是每次事件同步前仅包含一个点。

先来看包含多个点的

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** **void** ft5x0x\_report\_value(**struct** ft5x0x\_ts\_data \*data)
2. {
3. **struct** ts\_event \*event = &data->event;
4. **int** i;
5. **int** uppoint = 0;    //已经抬起的点数
7. **for** (i = 0; i < event->touch\_point; i++)  //循环处理 缓存中的所有点
8. {
9. input\_mt\_slot(data->input\_dev, event->au8\_finger\_id[i]);  //发送点的ID
11. **if** (event->au8\_touch\_event[i]== 0 || event->au8\_touch\_event[i] == 2)  //如果点按下
12. {
13. input\_mt\_report\_slot\_state(data->input\_dev, MT\_TOOL\_FINGER,  **true**);  //手指按下
14. input\_report\_abs(data->input\_dev,ABS\_MT\_POSITION\_X,event->au16\_x[i]); //x坐标
15. input\_report\_abs(data->input\_dev, ABS\_MT\_POSITION\_Y,event->au16\_y[i]);    //y坐标
16. input\_report\_abs(data->input\_dev,ABS\_MT\_TOUCH\_MAJOR,event->pressure); //触摸点长轴长度
17. }
18. **else**
19. {
20. uppoint++;                              //没有按下，则表明这个手指已经抬起
21. input\_mt\_report\_slot\_state(data->input\_dev, MT\_TOOL\_FINGER,**false**);   //报告手指抬起
22. }

25. }
27. **if**(event->touch\_point == uppoint)
28. {
29. input\_report\_key(data->input\_dev, BTN\_TOUCH, 0); //所有手指都抬起了 发送BTN\_TOUCH 抬起事件
31. }
32. **else**
33. {
34. input\_report\_key(data->input\_dev, BTN\_TOUCH, event->touch\_point > 0);//还有手指没抬起，发送BTN\_TOUCH 按下的事件
36. }
37. input\_sync(data->input\_dev); //sync 设备同步
39. }

然后是每次同步仅发送一个点

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** ft5x0x\_report\_value(**struct** ft5x0x\_ts\_data \*data)
3. {
4. **struct** ts\_event \*event = &data->event;
5. **int** i;
7. **for** (i = 0; i < event->touch\_point; i++)  //循环处理 缓存中的所有点
8. {
9. input\_mt\_slot(data->input\_dev, event->au8\_finger\_id[i]);  //发送点的ID
10. **if** (event->au8\_touch\_event[i]== 0 || event->au8\_touch\_event[i] == 2)  //如果点按下
11. {
12. input\_mt\_report\_slot\_state(data->input\_dev, MT\_TOOL\_FINGER,  **true**);  //手指按下
13. input\_report\_abs(data->input\_dev,ABS\_MT\_POSITION\_X,event->au16\_x[i]); //x坐标
14. input\_report\_abs(data->input\_dev, ABS\_MT\_POSITION\_Y,event->au16\_y[i]);    //y坐标
15. input\_report\_abs(data->input\_dev,ABS\_MT\_TOUCH\_MAJOR,event->pressure); //触摸点长轴长度
16. }
17. **else**
18. {
19. input\_mt\_report\_slot\_state(data->input\_dev, MT\_TOOL\_FINGER,**false**);   //手指抬起
20. }

23. input\_mt\_report\_pointer\_emulation(input\_dev, **true**);//用模拟点的方法，来告知此次触摸已经完成。
24. input\_sync(data->input\_dev); //sync 设备同步
26. }
27. }

这两种方法都可以，但是建议选择上面那种，效率比较高。

二、触摸按键部分

对于触摸按键的发送可以分为两种方法，一是android提供的 virtualkey's 架构方法，一种是直接报告key event的方法。我们一一来看

1.报告key event方法

在probe中添加所支持的按键类型，本人用的触摸屏上有三个按键因此

报告支持事件类型

\_\_set\_bit(EV\_SYN, input\_dev->evbit);

\_\_set\_bit(EV\_KEY, input\_dev->evbit);   
报告支持的按键

\_\_set\_bit(KEY\_HOME, input\_dev->keybit);     
\_\_set\_bit(KEY\_BACK, input\_dev->keybit);    
\_\_set\_bit(KEY\_MENU, input\_dev->keybit);

触摸屏上的三个按键对应的坐标

(KEY\_BACK)  120:1400   (KEY\_HOME) 360:1400(KEY\_MENU)  500:1400

key event的报告方法很简单只要报告相应的key 和设备同步sync就可以了

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** **void** ft5x0x\_report\_value(**struct** ft5x0x\_ts\_data \*data)
2. {
3. **struct** ts\_event \*event = &data->event;
4. **int** i;
5. **for** (i = 0; i < event->touch\_point; i++)
6. {
7. **if** (event->au16\_y[i]==1400)
8. {
9. **if**(event->au8\_touch\_event[i]== 0 || event->au8\_touch\_event[i] == 2)
10. {
12. **switch**(event->au16\_x[i])
13. {
14. **case** 120:
15. input\_report\_key(data->input\_dev, KEY\_BACK, 1);
16. **break**;
17. **case** 360:
18. input\_report\_key(data->input\_dev, KEY\_HOME, 1);
19. **break**;
20. **case** 500:
21. input\_report\_key(data->input\_dev, KEY\_MENU, 1);
22. **break**;
23. **default**: **break**;
24. }
26. }
27. **else**
28. {
30. **switch**(event->au16\_x[i])
31. {
32. **case** 120:
33. input\_report\_key(data->input\_dev, KEY\_BACK, 0);
34. **break**;
35. **case** 360:
36. input\_report\_key(data->input\_dev, KEY\_HOME, 0);
37. **break**;
38. **case** 500:
39. input\_report\_key(data->input\_dev, KEY\_MENU, 0);
40. **break**;
41. **default**: **break**;
42. }
44. }
45. input\_sync(data->input\_dev);
46. **return**;
47. }
48. }

对于这种方法有一个bug，就是事件发送上去，系统并不认为是触摸屏发送的按键，系统的 触屏震动反馈 并不起作用。这并不符合标准的android触摸设备标准。具体怎么破本人比较菜没有找到方法，大神们谁知道 求破。

2.virtualkeys方法

virtualkeys是android提供的架构使用起来简单方便，推荐大家使用。直接上代码

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** ssize\_t ft5x06\_virtual\_keys\_show(**struct** kobject \*kobj,       //按键的配置
2. **struct** kobj\_attribute \*attr, **char** \*buf)
3. {
4. **return** sprintf(buf,
5. \_\_stringify(EV\_KEY) ":" \_\_stringify(KEY\_BACK) ":120:1400:8:8"   //键类型:键值:按键区域中心x坐标:按键区域中心y坐标:按键区域宽:按键区域高
6. ":" \_\_stringify(EV\_KEY) ":"
7. \_\_stringify(KEY\_HOME) ":360:1400:8:8"
8. ":" \_\_stringify(EV\_KEY) ":"
9. \_\_stringify(KEY\_MENU) ":500:1400:8:8"
10. "\n");
11. }
13. **static** **struct** kobj\_attribute ft5x06\_virtual\_keys\_attr = {
14. .attr = {
15. .name = "virtualkeys.Ft5x0x\_Touch\_Screen",  //这里的名字必须为virtualkeys.设备名字  否则系统不会识别
16. .mode = S\_IRUGO,
17. },
18. .show = &ft5x06\_virtual\_keys\_show,
19. };
21. **static** **struct** attribute \*ft5x06\_properties\_attrs[] = {
22. &ft5x06\_virtual\_keys\_attr.attr,
23. NULL,
24. };
26. **static** **struct** attribute\_group ft5x06\_properties\_attr\_group = {
27. .attrs = ft5x06\_properties\_attrs,
28. };
30. **static** **void** ft5x06\_virtual\_keys\_init(**void**)
31. {
32. **struct** kobject \*properties\_kobj;
33. **int** ret;
35. properties\_kobj = kobject\_create\_and\_add("board\_properties", NULL);//添加目录board\_properties
37. **if** (properties\_kobj)
38. ret = sysfs\_create\_group(properties\_kobj,//生成/sys/board\_properties/virtualkeys.Ft5x0x\_Touch\_Screen虚拟按键配置文件
39. &ft5x06\_properties\_attr\_group); //可以使用 cat /sys/board\_properties/virtualkeys.Ft5x0x\_Touch\_Screen命令来查看配置是否正确
40. **if** (!properties\_kobj || ret)
41. pr\_err("failed to create board\_properties\n");
42. }

然后将ft5x06\_virtual\_keys\_init()加入到 触摸屏的init 或者probe 函数中，这样触摸键就可以使用了。

三、触摸屏驱动流程

i2c中加入平台初始化代码

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** **struct** ft5x0x\_platform\_data  ft5x0x\_platform\_i2c\_data = {
2. .x\_max=540,
3. .y\_max=960,
4. .irq= SABRESD\_CHARGE\_FLT\_1\_B,<span style="white-space:pre"> </span>//中断引脚
5. .reset=SABRESD\_DISP0\_RST\_B,<span style="white-space:pre">   </span>//复位引脚
6. };

触摸屏驱动初始化

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** **int** \_\_init ft5x0x\_ts\_init(**void**)
2. {
3. **int** ret;
4. ret = i2c\_add\_driver(&ft5x0x\_ts\_driver);
5. **if** (ret) {
6. printk(KERN\_WARNING "Adding ft5x0x driver failed "
7. "(errno = %d)\n", ret);
8. } **else** {
9. pr\_info("Successfully added driver %s\n",
10. ft5x0x\_ts\_driver.driver.name);
11. }
12. **return** ret;
13. }

probe函数

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. #define VIRTUAL\_LI      0
2. #define EVENT\_LI        1
3. #define TOUCH\_KEY       VIRTUAL\_LI
4. **static** **int** ft5x0x\_ts\_probe(**struct** i2c\_client \*client,
5. **const** **struct** i2c\_device\_id \*id)
6. {
7. 。。。。。。。。。。
8. ft5x0x\_ts = kzalloc(**sizeof**(**struct** ft5x0x\_ts\_data), GFP\_KERNEL);//分配参数内存
10. ..........
11. i2c\_set\_clientdata(client, ft5x0x\_ts);参数地址传给i2c 内核
13. 初始化一些参数
14. ft5x0x\_ts->irq = client->irq;
15. ft5x0x\_ts->client = client;
16. ft5x0x\_ts->pdata = pdata;
17. ft5x0x\_ts->x\_max = pdata->x\_max - 1;
18. ft5x0x\_ts->y\_max = pdata->y\_max - 1;
19. ft5x0x\_ts->pdata->reset = FT5X0X\_RESET\_PIN;
20. ft5x0x\_ts->pdata->irq = ft5x0x\_ts->irq;
21. .....................
22. err = request\_threaded\_irq(client->irq, NULL, ft5x0x\_ts\_interrupt,   //注册读取数据中断
23. IRQF\_TRIGGER\_FALLING, client->dev.driver->name,
24. ft5x0x\_ts);
25. 。、、、、、、、、、、、、
26. input\_dev = input\_allocate\_device();//分配设备
27. ...........................................
28. \_\_set\_bit(EV\_SYN, input\_dev->evbit);  //注册设备支持event类型
29. \_\_set\_bit(EV\_ABS, input\_dev->evbit);
30. \_\_set\_bit(EV\_KEY, input\_dev->evbit);
31. \_\_set\_bit(BTN\_TOUCH, input\_dev->keybit);
32. #if TOUCH\_KEY == EVENT\_LI               //如果使用event key的方法
33. \_\_set\_bit(KEY\_HOME, input\_dev->keybit);
34. \_\_set\_bit(KEY\_BACK, input\_dev->keybit);
35. \_\_set\_bit(KEY\_MENU, input\_dev->keybit);
36. #endif
37. input\_mt\_init\_slots(input\_dev, CFG\_MAX\_TOUCH\_POINTS);   //设备属性
38. input\_set\_abs\_params(input\_dev,ABS\_MT\_TOUCH\_MAJOR,
39. 0, PRESS\_MAX, 0, 0);
40. input\_set\_abs\_params(input\_dev, ABS\_MT\_POSITION\_X,
41. 0, ft5x0x\_ts->x\_max, 0, 0);
42. input\_set\_abs\_params(input\_dev, ABS\_MT\_POSITION\_Y,
43. 0, ft5x0x\_ts->y\_max, 0, 0);


47. input\_dev->name ="Ft5x0x\_Touch\_Screen";//lijianzhang
48. err = input\_register\_device(input\_dev);         //注册这个input设备
49. 。。。。。。。。。。。
50. #if TOUCH\_KEY == VIRTUAL\_LI     //如果使用虚拟键盘设定
52. ft5x06\_virtual\_keys\_init();
53. #endif
54. 。。。。。。。。。。。。。。。。。。。。。
55. }

中断处理

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** irqreturn\_t ft5x0x\_ts\_interrupt(**int** irq, **void** \*dev\_id)
2. {
3. **struct** ft5x0x\_ts\_data \*ft5x0x\_ts = dev\_id;
4. **int** ret = 0;
5. disable\_irq\_nosync(ft5x0x\_ts->irq);
6. ret = ft5x0x\_read\_Touchdata(ft5x0x\_ts); //读取数据
7. **if** (ret == 0)
8. ft5x0x\_report\_value(ft5x0x\_ts);//报告数据
10. enable\_irq(ft5x0x\_ts->irq);
12. **return** IRQ\_HANDLED;
13. }

报告事件

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. **static** **void** ft5x0x\_report\_value(**struct** ft5x0x\_ts\_data \*data)
2. {
3. **struct** ts\_event \*event = &data->event;
4. **int** i;
5. **int** uppoint = 0;
7. /\*protocol B\*/
8. **for** (i = 0; i < event->touch\_point; i++)
9. {
10. #if TOUCH\_KEY == EVENT\_LI       //如果使用 key event方法
11. **if** (event->au16\_y[i]==1400)
12. {
13. **if**(event->au8\_touch\_event[i]== 0 || event->au8\_touch\_event[i] == 2)
14. {
16. **switch**(event->au16\_x[i])
17. {
18. **case** 120:
19. input\_report\_key(data->input\_dev, KEY\_BACK, 1);
20. **break**;
21. **case** 360:
22. input\_report\_key(data->input\_dev, KEY\_HOME, 1);
23. **break**;
24. **case** 500:
25. input\_report\_key(data->input\_dev, KEY\_MENU, 1);
26. **break**;
27. **default**: **break**;
28. }
30. }
31. **else**
32. {
34. **switch**(event->au16\_x[i])
35. {
36. **case** 120:
37. input\_report\_key(data->input\_dev, KEY\_BACK, 0);
38. **break**;
39. **case** 360:
40. input\_report\_key(data->input\_dev, KEY\_HOME, 0);
41. **break**;
42. **case** 500:
43. input\_report\_key(data->input\_dev, KEY\_MENU, 0);
44. **break**;
45. **default**: **break**;
46. }
47. uppoint++;
49. }
50. input\_sync(data->input\_dev);
51. **return**;
52. }
54. #endif
56. input\_mt\_slot(data->input\_dev, event->au8\_finger\_id[i]);

59. **if** (event->au8\_touch\_event[i]== 0 || event->au8\_touch\_event[i] == 2)
60. {
61. input\_mt\_report\_slot\_state(data->input\_dev, MT\_TOOL\_FINGER,  **true**);
62. input\_report\_abs(data->input\_dev,ABS\_MT\_POSITION\_X,event->au16\_x[i]);     //lijianzhang
63. input\_report\_abs(data->input\_dev, ABS\_MT\_POSITION\_Y,event->au16\_y[i]);
64. input\_report\_abs(data->input\_dev,ABS\_MT\_TOUCH\_MAJOR,event->pressure);
65. }
66. **else**
67. {
68. uppoint++;
69. input\_mt\_report\_slot\_state(data->input\_dev, MT\_TOOL\_FINGER,  **false**);
70. }

73. }
75. **if**(event->touch\_point == uppoint)
76. {
77. input\_report\_key(data->input\_dev, BTN\_TOUCH, 0);
78. }
79. **else**
80. {
81. input\_report\_key(data->input\_dev, BTN\_TOUCH, event->touch\_point > 0);
82. }
83. input\_sync(data->input\_dev);
85. }

这里驱动流程做了简略的说明，关键的代码都已经贴出来了。与设备相关代码都是厂商给的没有太实际参考价值.

从android input的流程分析我们知道，驱动编译完成以后，要使触摸屏工作，还需要三个文件：触摸屏配置文件 （idc文件，用来配置触摸屏的一些属性）、keylayout文件（kl文件，安卓层面的按键映射文件）、characterMap文件（kcm文件，安卓层面的字符映射文件）  
我们一一来看这三个文件

1.触摸屏配置文件

文件所在目录访问顺序：

首先ANDROID\_ROOT/usr/idc目录下去找相应名字的文件并返回完整的路径名，如果找不到就从ANDROID\_DATA/system/devices/idc下面去找,这里ANDROID\_ROOT一般指的是/system目录，ANDROID\_DATA一般指/data目录.

文件名称的查找顺序首先是Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.idc，然后是Vendor\_XXXX\_Product\_XXXX.idc最后是DEVICE\_NAME.idc

总结来看安卓为输入设备打开配置文件依次会访问  
/system/usr/idc/Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.idc  
/system/usr/idc/Vendor\_XXXX\_Product\_XXXX.idc  
/system/usr/idc/DEVICE\_NAME.idc  
/data/system/devices/idc/Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.idc  
/data/system/devices/idc/Vendor\_XXXX\_Product\_XXXX.idc  
/data/system/devices/idc/DEVICE\_NAME.idc

我们驱动里并没有写版本号等这些信息，因此我们设备访问的idc文件会是/system/usr/idc/DEVICE\_NAME.idc。因此我们在这个目录下增加文件Ft5x0x\_Touch\_Screen.idc.对于idc文件的内容，下面是我使用的idc文件的具体内容，仅供参考

**[cpp]** [view plaincopy](http://blog.csdn.net/dkleikesa/article/details/9415023)

1. touch.deviceType = touchScreen
2. touch.orientationAware = 1
4. touch.size.calibration = none
5. touch.orientation.calibration = none

2.key layout文件

key layout文件是android层面的按键映射文件，通过这个文件，用户可以对kernel发送上来的按键功能进行重新定义。也就是说，kernel发送上来一个home键，你可以在这里把它映射成一个back键或者其他的。一般情况下不会修改这个文件，因此我么完全可以使用默认的配置文件

这个文件访问顺序

/system/usr/keylayout/Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.kl  
/system/usr/keylayout/Vendor\_XXXX\_Product\_XXXX.kl  
/system/usr/keylayout/DEVICE\_NAME.kl  
/data/system/devices/keylayout/Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.kl  
/data/system/devices/keylayout/Vendor\_XXXX\_Product\_XXXX.kl  
/data/system/devices/keylayout/DEVICE\_NAME.kl  
/system/usr/keylayout/Generic.kl  
/data/system/devices/keylayout/Generic.kl

这里不用修改因此不用做改变

3.characterMap文件

characterMap文件是android层面的字符映射文件，比如：你摁下了一个'e'键，平时代表'e'，shift+'e'代表'E',casplk+'e'代表'E',alt+'e'可能代表别的意思，这个配置文件就是，做这些映射的。一般情况下这个文件也不用修改。使用默认的就可以。这个文件的访问顺序：

/system/usr/keychars/Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.kcm  
/system/usr/keychars/Vendor\_XXXX\_Product\_XXXX.kcm  
/system/usr/keychars/DEVICE\_NAME.kcm  
/data/system/devices/keychars/Vendor\_XXXX\_Product\_XXXX\_Version\_XXXX.kcm  
/data/system/devices/keychars/Vendor\_XXXX\_Product\_XXXX.kcm  
/data/system/devices/keychars/DEVICE\_NAME.kcm  
/system/usr/keychars/Generic.kcm  
/data/system/devices/keychars/Generic.kcm  
/system/usr/keychars/Virtual.kcm  
/data/system/devices/keychars/Virtual.kcm

到了这里 我们的触摸屏已经完成了，烧写以后应该可以正常使用了。

在这里分享一个小技巧，getevent 这个工具，在/dev/input/目录下使用这个命令，会首先得到系统中所有input设备的描述，然后会得到，kernel发送的所有input事件，当我们写完驱动以后，可以用这个命令将发送的事件打印出来，看驱动写的是否正确。