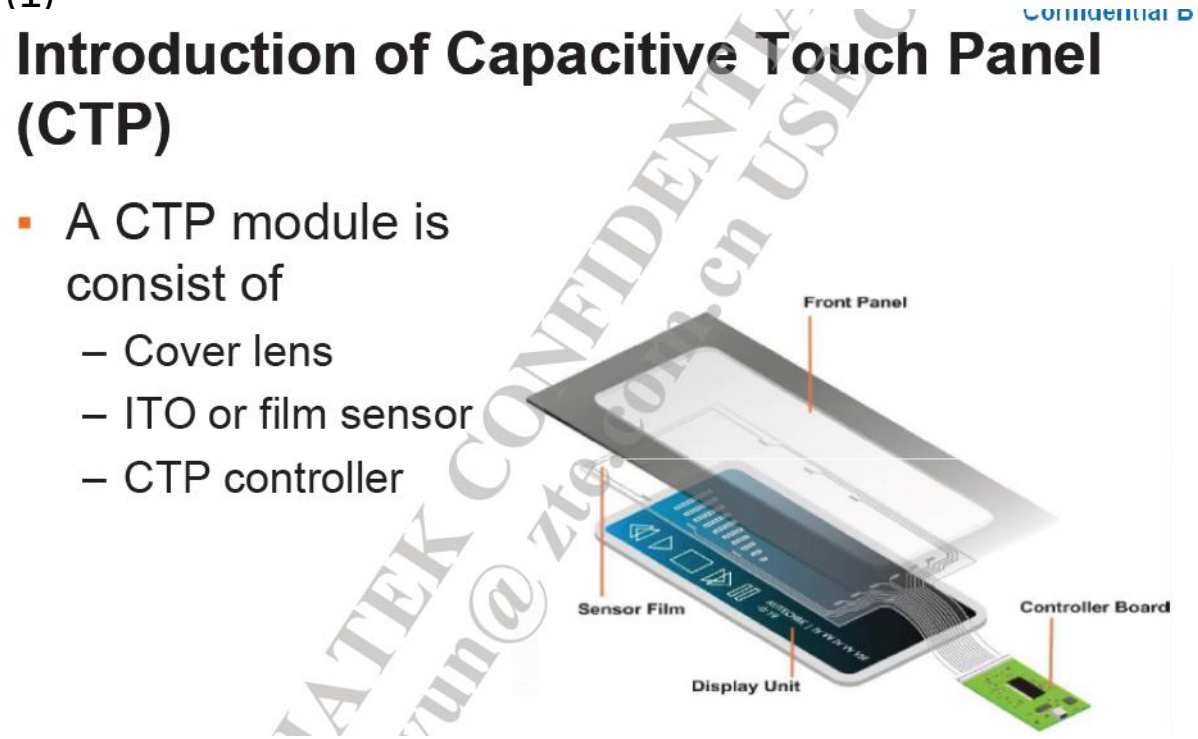
CTP

笔记本： td
创建时间： 2013/9/2 16:20 更新时间： 2014/2/13 15:52

一.CTP模组简介

(1)



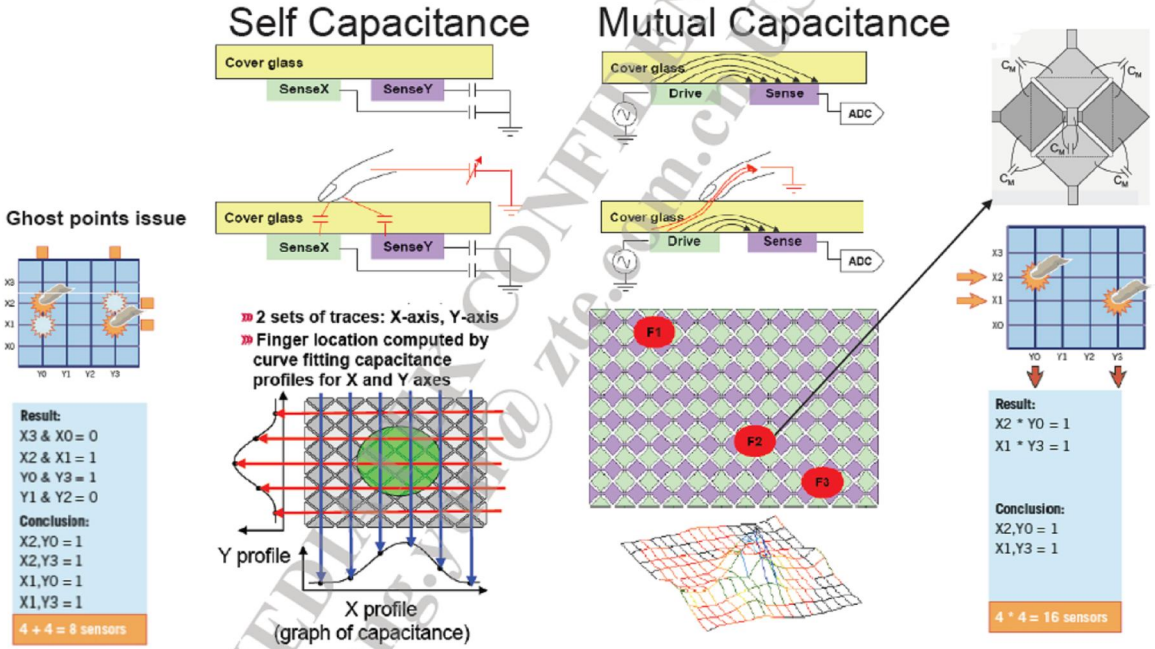
ps:ITO 掺锡氧化铟（IndiumTinOxide）是一种n型半导体材料，具有高的导电率、高的可见光透过率、高的机械硬度和良好的化学稳定性

Types of CTP Controller

- Two types of CTP controller:
  - **Self capacitance:** measure self capacitance change on each sensing line
  - **Mutual capacitance:** measure mutual capacitance change between driving lines and sensing line

	Self capacitance	Mutual capacitance
Performance	Good	Excellent
Scan rate	Fast	Average
Power consumption	Low	Average
Multi-touch capability	Yes, but can only support two-finger touch with ghost point issue	Yes, may support 2~10 fingers.
Chip Cost	Lower	Higher

Comparison of Controller



(2) IC厂商：

Synaptics、Atmel与Cypress等三家厂商为主

国内常用的：GOODIX/FT/MSTAR/HIMAX/

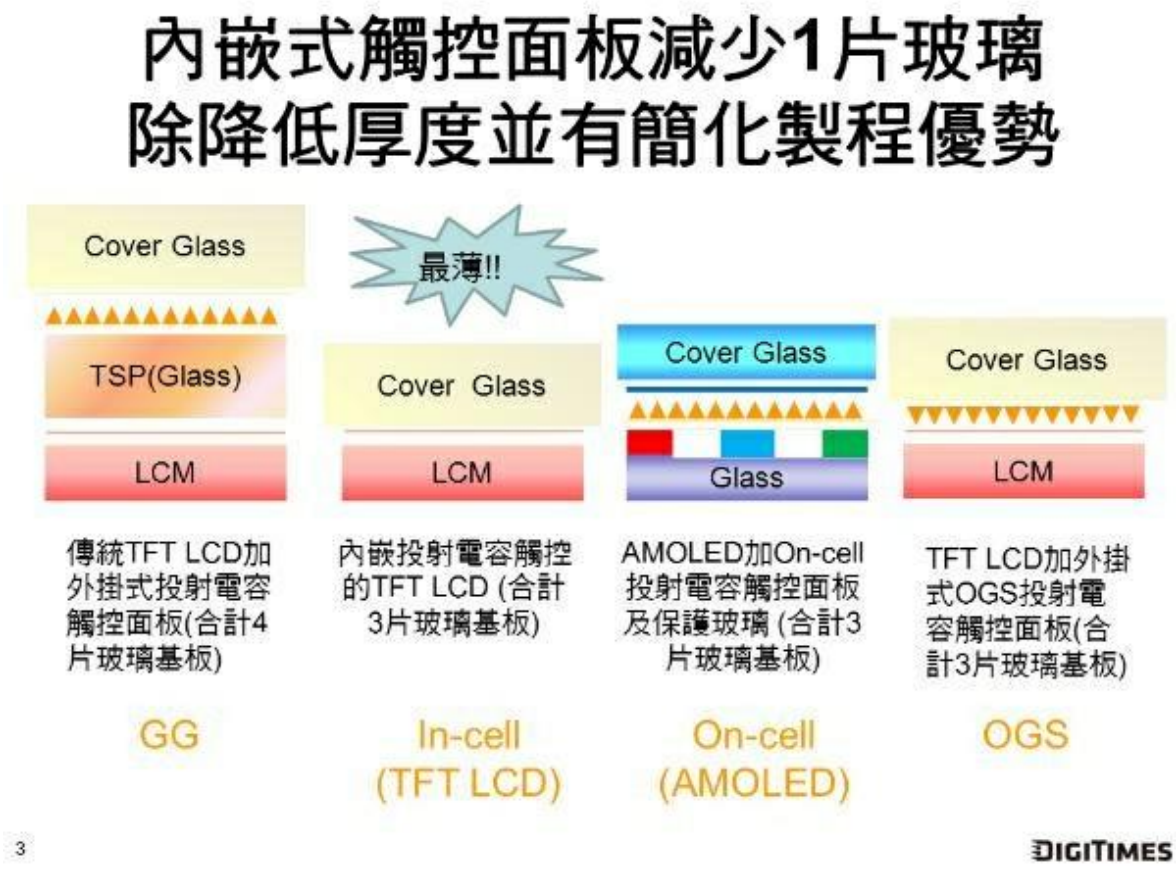
(3) 关于玻璃贴合及触控屏整合工艺:

单玻璃贴合技术。这些技术都是把触控部分整合到内层玻璃或者是显示屏上面，实现减少厚度、简化工艺制程、增加屏幕 的通透程度、减少反光、不进灰等目的。目前这一类的技术主要有以触控屏厂商为主导的One Glass/Touch on Lens方案，以及由面板厂商主导的On-Cell和In-Cell方案。

One Glass/Touch on Lens通过在保护玻璃内侧镀上ITO导电层，把触控屏与保护玻璃集成在一起，代表产品有魅族MX2、小米手机2，使用这一方案的手机屏幕如果摔碎的话，触控也随之失效；

On-Cell是将触摸屏嵌入到显示屏的彩色滤光片基板和偏光片之间，代表产品有三星的Galaxy S3；

In-Cell则是将触摸传感器嵌入到液晶像素中，代表产品有苹果的iPhone 5。



要彻底了解In-Cell/On-Cell/OGS等等屏幕，就得先知道屏幕的基本结构组成。

从上到下，**屏幕的基本结构分为三层，保护玻璃（最上面橙黄色标注了Cover glass的部分），触控层（图中一点点淡蓝色虚线，标注了X、Y的部分），显示面板。**

保护玻璃没什么好说的，康宁大猩猩玻璃就是。触控层的话，就是由ITO触控薄膜和ITO玻璃基板组成。显示面板可细分的程度高，这里只大致排列下：从上到下，分别是上玻璃基板（粉红色标注了Color filiter的区域，即彩色滤光基板），液晶层（蓝条），下玻璃基板（粉红色标注了Array的区域，即薄膜电晶体基板）。最后，还需要指出的是，保护玻璃/触控层与显示面板之间，一般贴合技术会形成一层空气（即图中标注了Bonding的金黄色区域），如果采用全贴合技术去除这层空气，屏幕反光会大大减少，点亮屏幕时就显得更为通透，熄屏时更加黑沉，没有灰白的观感。

传统的G/G、GFF屏幕，都是标准的保护玻璃+触控层+显示面板层的结构，不同之处在于触控层。G/G屏幕的触控层是由1层ITO玻璃基板+1层ITO触控薄膜组成，GFF屏幕的触控层则有2层ITO玻璃基板+2层ITO触控薄膜（ITO：X和ITO：Y）。显然，G/G屏幕更薄一些。

今天各厂家标榜的OGS屏幕、In Cell/On Cell屏幕，为何值得拿出来吹嘘，是因为它们都是保护玻璃层+显示面板层的结构，少了一层触控层，更加轻薄。那中间的触控层哪去了呢？这正好是区分In Cell/On Cell屏幕和OGS屏幕的关键。In Cell/On Cell屏幕是将触控层和显示面板整合在了一起，OGS屏幕是将保护玻璃层和触控层整合在了一起。为了争夺产业话语权，**触控模块厂商力推OGS，而显示面板厂商如LG、三星、夏普等则力推In-Cell/On-Cell。**

可能还有人追根究底，既然In Cell屏幕和On Cell屏幕都是将触摸层和显示面板整合在一起，那如何区分In Cell和On Cell？这跟ITO薄膜嵌入显示面板的位置有关。所谓On Cell，就是将ITO触控薄膜放在了显示面板的上玻璃基板之上，三星几代Galaxy旗舰就是这么做的；至于In-Cell，则是ITO触控薄膜放在了显示面板的上玻璃基板之下，一般是与液晶层融合在一起，代表机型是苹果的iPhone 5。

In-Cell、On-Cell、OGS因为不同的结构，屏幕的轻薄程度、显示效果、制造成本、良品率都有所不同。下面是一个基本总结：

1、屏幕的通透程度和视觉效果方面，OGS是最好的，In-Cell和On-Cell则次之。所以，吹嘘In-Cell/On-Cell如何如何通透，视觉效果如何好，可以歇歇了。无论是iPhone5还是三星Galaxy S4，单纯的屏幕通透度其实还不如一些采用了OGS屏幕的国产手机，比如魅族MX3。

2、轻薄程度，一般来说In-Cell最轻最薄，这也是iPhone使用了金属机身，还能做到极致轻薄的原因之一。OGS则次之，On-Cell比前两者稍差。

3、屏幕强度（抗冲击、抗摔），On-Cell最好，OGS次之，In-Cell最差。需要指出的是，OGS则因为直接将康宁保护玻璃与触控层整合在一起，加工过程削弱了玻璃的强度，屏幕也很脆弱。至于In-Cell屏幕为何强度差，笔者也不甚清楚其中的原因，这可能跟目前苹果自家的In-Cell追求极致轻薄有关，Lumia 920同样是In-Cell屏幕，但有很好的辅助设计/措施来增强屏幕的抗冲击、抗摔能力。另，需要指出的是，因为In-Cell将触控层与液晶层融合在一起，一旦触摸屏出了问题，需要连同显示面板一起更换。

4、触控方面，OGS的触控灵敏度比On-Cell/In-Cell屏幕都要好，但这有时候也不是什么好事。OGS技术不过关，超高的灵敏度很容易发生“跳屏”的现象，细微的灰尘、汗液、水汽都能引发触控屏的“误操作”，比如MX3最早一批机器这个问题就比较严重。对多点触控、手指、Stylus触控笔的支持上，其实OGS也是好于In-Cell/On-Cell的。另外，还是因为In-Cell屏幕直接将触控层和液晶层融合在一起，感测杂讯较大，需要有专门的触控芯片进行过滤和校正处理。OGS屏幕对于触控芯片的依赖则没那么高。

5、技术要求，In-Cell/On-Cell都比OGS要复杂，生产控制上，难度也更高。

6、良品率方面，之前In-Cell屏幕的良品率较低，很大程度上影响了iPhone 5等产品的供货，但随着厂商不断投入，技术走向成熟，In-Cell/On-Cell屏幕的良品率与OGS已经处于同一水平，大规模出货不成问题。

表1 In-cell與Out-cell技術比較

Project Capacitive		In-Cell		Out-Cell		
				G/G	OGS	GFF
Display Mode		VA/IPS/TN mode		Any	Any	Any
Touch Panel Stick		CL+LCM	LCM	CL/Sensor+LCM	CL+LCM	CL/Sensor+LCM
Transparency/Opt. performance		O/◎	O/◎	O/◎	◎/◎	O-/O
Column Spacer Life Damage		◎	O	◎	◎	◎
Array Design, Process & Material		Complex	Complex	Easy	Easy	Easy
Process Yield Control		Δ	Δ	O+	O+	Δ
ME	Thickness (Weight)	O	◎+	O	◎	O+
	Strength	◎	Δ	◎	O+	◎
ID	2D design	◎	O	◎	◎	◎
	2.5D design	Development	NA	O	Development	◎
	3D design	Development	NA	O	Development	X
Touch function	Multi-Touch	O	O	◎	◎	◎
	Noise effect	High	High	Low	Low	Low
Input Device	Finger	O	O	◎	◎	◎
	Stylus	Δ	Δ	O	O	O
Customer Request	Different outline	◎	Δ	◎	◎	◎
	TP sensitivity	O	O	◎	◎	O
	TP supply feasibility	Δ	Δ	◎	◎	◎
	Non-AA Touch(Virtual key)	O	X	◎	◎	◎
	IC supply feasibility	O	O	◎	◎	◎
	1st pixel accuracy	◎	◎	◎	◎	O
	OSlim boarder	O	O	◎	◎	Δ
	Quality	◎	◎	◎	◎	Δ
Manufacture Efficiency		◎	◎	◎	◎	O
Touch Panel Support Size		<10.1"		2.X"~30"		2.X"~10.1"
Market Application		Mobile/Tablet (少樣多量)	NB/DSC/PND (少樣多量)	Mobile/Tablet/EPD/NB/AiO/DSC/PND (多樣市場)		
Technology Stage		R&D Pilot run		MP	MP	MP

Δ：bad O：good ◎：better

~~二.多点触控协议~~

~~https://www.kernel.org/doc/Documentation/input/multi-touch-protocol.txt~~

1， 两种多点触摸协议：

1）A类： 处理无关联的接触： 用于直接发送原始数据；

B类： 处理跟踪识别类的接触： 通过事件slot发送相关联的独立接触更新。

2， 触摸协议的使用：

A类协议：

A类协议在每发送完一个接触数据包后会调用 input_mt_sync() 声明 一次数据的结束；input_mt_sync() 会发出一个 SYN_MT_REPORT

提示接收器接收数据并准备下一次数据的接收。

B类协议：

与A类协议不同的是， B类在使用input_mt_slot()的时候会带有一个slot的参数，在每个数据包开始时 产生一个ABS_MT_SLOT事件，提示接收器更新数据。

最终A，B类协议均会调用 input_sync();

A类与B类协议不同的在于： B类协议 通过B类协议 slot 协议需要是用到 ABS_MT_TRACKING_ID------ 可以从硬件上获取，或者从原始数据中计算。

3, B类协议： ABS_MT_TRACKING_ID 表示一次接触； -1 代表一个不用的slot；

使用参考例子：

释放事件：

```
input_mt_slot(data->input_dev, i);
input_mt_report_slot_state(data->input_dev, MT_TOOL_FINGER, false);        ----------释放
```

点击事件：

```
input_mt_slot(data->input_dev, i);
input_mt_report_slot_state(data->input_dev, MT_TOOL_FINGER, true);
input_report_abs(data->input_dev, ABS_MT_TOUCH_MAJOR, 1);
```

input_report_abs(data->input_dev, ABS_MT_POSITION_X, current_events[i].x);

input_report_abs(data->input_dev, ABS_MT_POSITION_Y, current_events[i].y)

为了使用功能强大的多点触控设备，就需要一种方案去上报用户层所需的详细的手指触摸数据。这个文档所描述的多点触控协议可以让内核驱动程序向用户层上报任意多指的数据信息。

使用说明

单点触摸信息是以ABS承载并按一定顺序发送，如BTN_TOUCH、ABS_X、ABS_Y、SYNC。而多点触摸信 息则是以ABS_MT承载并按一定顺序发送，如ABS_MT_POSITION_X、ABS_MT_POSITION_Y，然后通过调用 input_mt_sync()产生一个 SYN_MT_REPORT event来标记一个点的结束，告诉接收方接收当前手指的信息并准备接收其它手指的触控信息。最后调用 input_sync()函数上报触摸信息开始动作并告诉接收方开始接收下一系列多点触摸信息。

协议定义了一系列ABS_MT事件，这些事件被分为几大类，充许只应用其中的一部份，多点触摸最小的事件集中应包括 ABS_MT_TOUCH_MAJOR、ABS_MT_POSITION_X和 ABS_MT_POSITION_Y，以此来实现多点触摸。如果设备支持ABS_MT_WIDTH_MAJOR这个事件，那么此事件可以提供手指触摸接触 面积大小。触摸方向等信息可以由ABS_MT_TOUCH_MINOR, ABS_MT_WIDTH_MINOR and ABS_MT_ORIENTATION提供。ABS_MT_TOOL_TYPE提供触摸设备的类别，如手或是笔或是其它。最后有些设备可能会支持 ABS_MT_TRACKING_ID，用来支持硬件跟踪多点信息，即该点属于哪一条线等。

下面是两点触摸支持的最小事件集序列：

ABS_MT_TOUCH_MAJOR
ABS_MT_POSITION_X
ABS_MT_POSITION_Y
SYN_MT_REPORT //上报第一个点
ABS_MT_TOUCH_MAJOR
ABS_MT_POSITION_X
ABS_MT_POSITION_Y
SYN_MT_REPORT //上报第二个点
SYN_REPORT //开始动作

Event 原语

“接触”一词用来描述一个物体直接碰到另一个物体的表面。

ABS_MT_TOUCH_MAJOR描述了主接触面的长轴，它和X，Y同一个单位，如果一个面的分辨率为X*Y，则ABS_MT_TOUCH_MAJOR的最大值为sqrt(X^2+Y^2)

ABS_MT_TOUCH_MINOR描述了接触面的短轴，如果接触面是圆形，它可以不用。

ABS_MT_WIDTH_MAJOR描述了接触工具的长轴

ABS_MT_WIDTH_MINOR描述了接触工具的短轴

ABS_MT_TOUCH_MAJOR := max(X, Y)

ABS_MT_TOUCH_MINOR := min(X, Y)

ABS_MT_ORIENTATION := bool(X > Y)

以上四个参数可以用来生成额外的触摸信息，如ABS_MT_TOUCH_MAJOR/ABS_MT_WIDTH_MAJOR的比率可以用来描述压力。

ABS_MT_ORIENTATION

ABS_MT_POSITION_X接触面的中心点X坐标

ABS_MT_POSITION_Y接触面的中心点Y坐标

ABS_MT_TOOL_TYPE描述接触工具类型，很多内核驱动无法区分此参数如手指及笔，如果是这样，该参数可以不用，协议目前支持MT_TOOL_FINGER和MT_TOOL_PEN两种类型。

ABS_MT_BLOB_ID形状集ID，集合几个点以描述一个形状，很多驱动没有形状属性，此参数可以不

用。

ABS_MT_TRACKING_ID描述了从接触开始到释放的整个过程的集合，如果设备不支持，此参数可是不用。

触摸轨迹

仅有少数设备可以明触的标识真实的 trackingID，多数情况下 trackingID只能来标识一次触摸动作的过程。

手势

多点触摸指定的应用是创建手势动作， TOUCH和 WIDTH参数经常用来区别手指的压力和手指间的距离，另外 MINOR类的参数可以用来区别设备的接触面的大小(点接触还是面接触)，ORIENTATION可以产生旋转事件。

==========

在Linux内核支持的基础上，Android在其2.0源码中加入多点触摸功能。由此触摸屏在Android的frameworks被完全分为2种实现途径：单点触摸屏的单点方式，多点触摸屏的单点和多点方式。

在Linux的input.h中，多点触摸功能依赖于以下几个主要的软件位：

```
……
#define SYN_REPORT0
#define SYN_CONFIG1
#define SYN_MT_REPORT2
……
#define ABS_MT_TOUCH_MAJOR0x30
#define ABS_MT_TOUCH_MINOR0x31
#define ABS_MT_WIDTH_MAJOR0x32
#define ABS_MT_WIDTH_MINOR0x33
#define ABS_MT_ORIENTATION0x34
#define ABS_MT_POSITION_X0x35
#define ABS_MT_POSITION_Y0x36
#define ABS_MT_TOOL_TYPE0x37
#define ABS_MT_BLOB_ID0x38
……
```

在Android中对应的软件位定义在RawInputEvent.java中：

```
……
public class RawInputEvent {
……
public static final int CLASS_TOUCHSCREEN_MT = 0x00000010;
……
public static final int ABS_MT_TOUCH_MAJOR = 0x30;
public static final int ABS_MT_TOUCH_MINOR = 0x31;
public static final int ABS_MT_WIDTH_MAJOR = 0x32;
public static final int ABS_MT_WIDTH_MINOR = 0x33;
public static final int ABS_MT_ORIENTATION = 0x34;
public static final int ABS_MT_POSITION_X = 0x35;
public static final int ABS_MT_POSITION_Y = 0x36;
public static final int ABS_MT_TOOL_TYPE = 0x37;
public static final int ABS_MT_BLOB_ID = 0x38;
……
public static final int SYN_REPORT = 0;
public static final int SYN_CONFIG = 1;
public static final int SYN_MT_REPORT = 2;
……
```

在Android中，多点触摸的实现方法在具体的代码实现中和单点是完全区分开的。在Android代码的EventHub.cpp中，单点屏和多点屏由如下代码段来判定：

```
int EventHub::open_device(const char *deviceName)
{
……
if (test_bit(ABS_MT_TOUCH_MAJOR, abs_bitmask)
&& test_bit(ABS_MT_POSITION_X, abs_bitmask)
&& test_bit(ABS_MT_POSITION_Y, abs_bitmask)) {
device->classes |= CLASS_TOUCHSCREEN | CLASS_TOUCHSCREEN_MT;
//LOGI("It is a multi-touch screen!");
}
//single-touch?
else if (test_bit(BTN_TOUCH, key_bitmask)
&& test_bit(ABS_X, abs_bitmask)
&& test_bit(ABS_Y, abs_bitmask)) {
device->classes |= CLASS_TOUCHSCREEN;
//LOGI("It is a single-touch screen!");
}
……
}
```

我们知道，在触摸屏驱动中，通常在probe函数中会调用input_set_abs_params给设备的input_dev结构体初始化， 这些input_dev的参数会在Android的EventHub.cpp中被读取。如上可知，如果我们的触摸屏想被当成多点屏被处理，只需要在驱动中 给input_dev额外增加以下几个参数即可：

```
input_set_abs_params(mcs_data.input, ABS_MT_POSITION_X, pdata->abs_x_min, pdata->abs_x_max, 0, 0);
input_set_abs_params(mcs_data.input, ABS_MT_POSITION_Y, pdata->abs_y_min, pdata->abs_y_max, 0, 0);
input_set_abs_params(mcs_data.input, ABS_MT_TOUCH_MAJOR, 0, 15, 0, 0);
//相当于单点屏的ABX_PRESSURE
input_set_abs_params(mcs_data.input, ABS_MT_WIDTH_MAJOR, 0, 15, 0, 0);
//相当于单点屏的ABS_TOOL_WIDTH
```

由于多点触摸技术需要采集到多个点，然后再一起处理这些点，所以在软件实现中需要保证每一波点的准确性和完整性。因此，Linux内核提供了 input_mt_sync(struct input_dev * input)函数。在每波的每个点上报后需要紧跟一句input_mt_sync()， 当这波所有点上报后再使用input_sync()进行同步。例如一波要上报3个点：

```
……
input_mt_sync(input);
……
input_mt_sync(input);
……
input_mt_sync(input);
input_sync(input);
```

注：即使是仅上报一个点的单点事件，也需要一次input_my_sync。

三.linux 输入子系统

1. Input子系统是分层结构的，总共分为三层： 硬件驱动层，子系统核心层，事件处理层。

（1）其中硬件驱动层负责操作具体的硬件设备，这层的代码是针对具体的驱动程序的，需要驱动程序的作者来编写。

（2）子系统核心层是链接其他两个层之间的纽带与桥梁，向下提供驱动层的接口，向上提供事件处理层的接口。

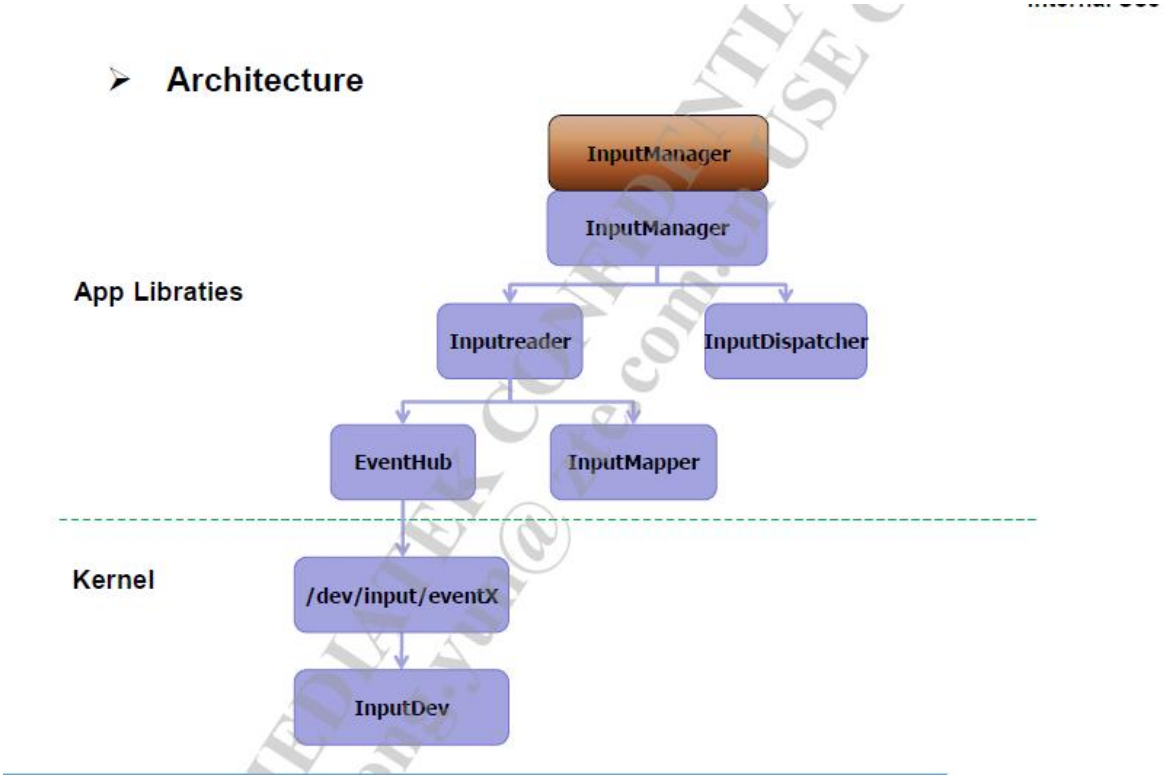
（3）事件处理层负责与用户程序打交道，将硬件驱动层传来的事件报告给用户程序。

2. 各层之间通信的基本单位就是事件，任何一个输入设备的动作都可以抽象成一种事件，如键盘的按下，触摸屏的按下，鼠标的移动等。事件有三种属性：类型（type），编码(code)，值(value)，Input子系统支持的所有事件都定义在input.h中，包括所有支持的类型，所属类型支持的编码等。事件传送的方向是 硬件驱动层-->子系统核心-->事件处理层-->用户空间

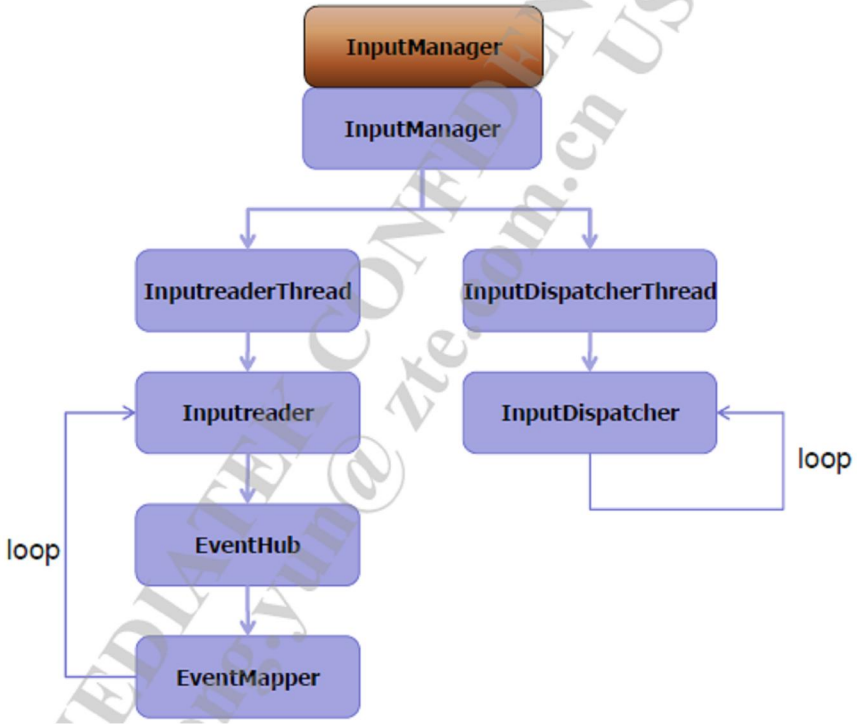
3. input_register_device　向内核注册一个input设备

input_register_handle　向内核注册一个handle结构

input_register_handler　注册一个事件处理器

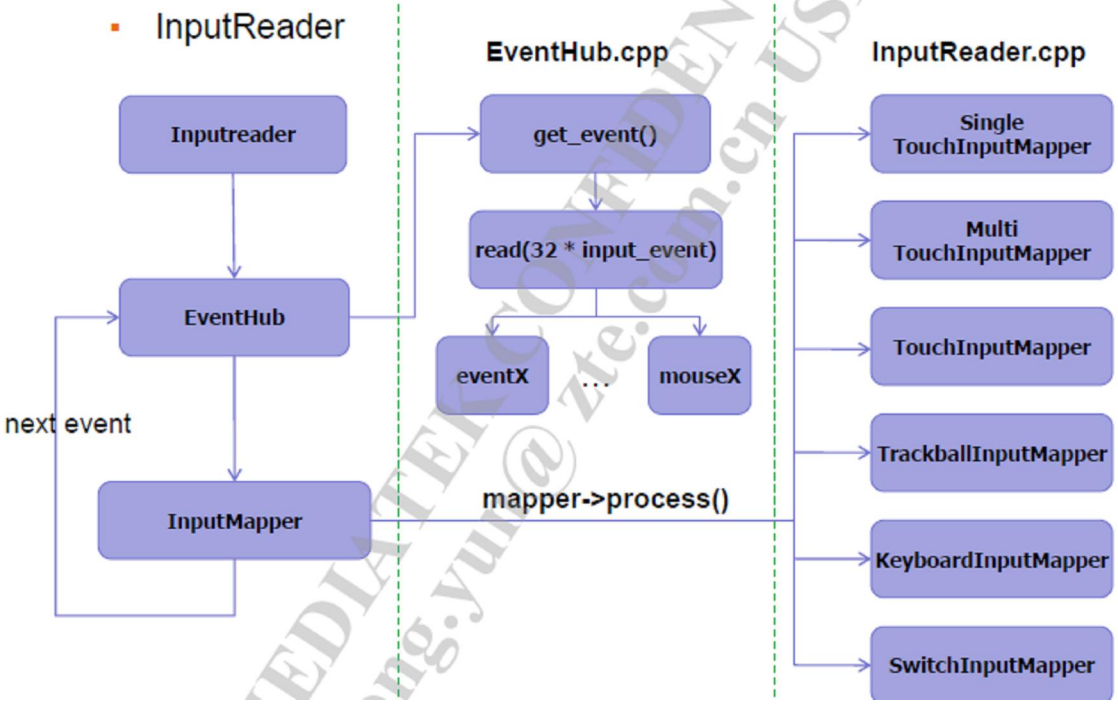


App Libraries



App Libraries

- InputReader



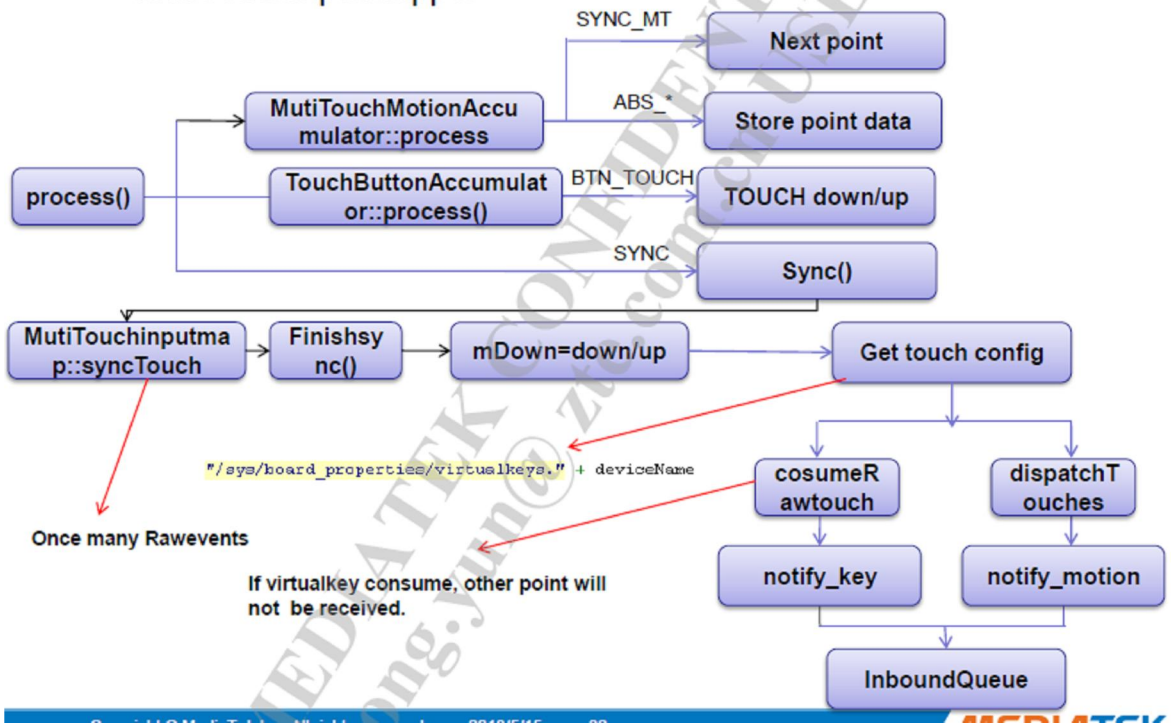
App Libraries

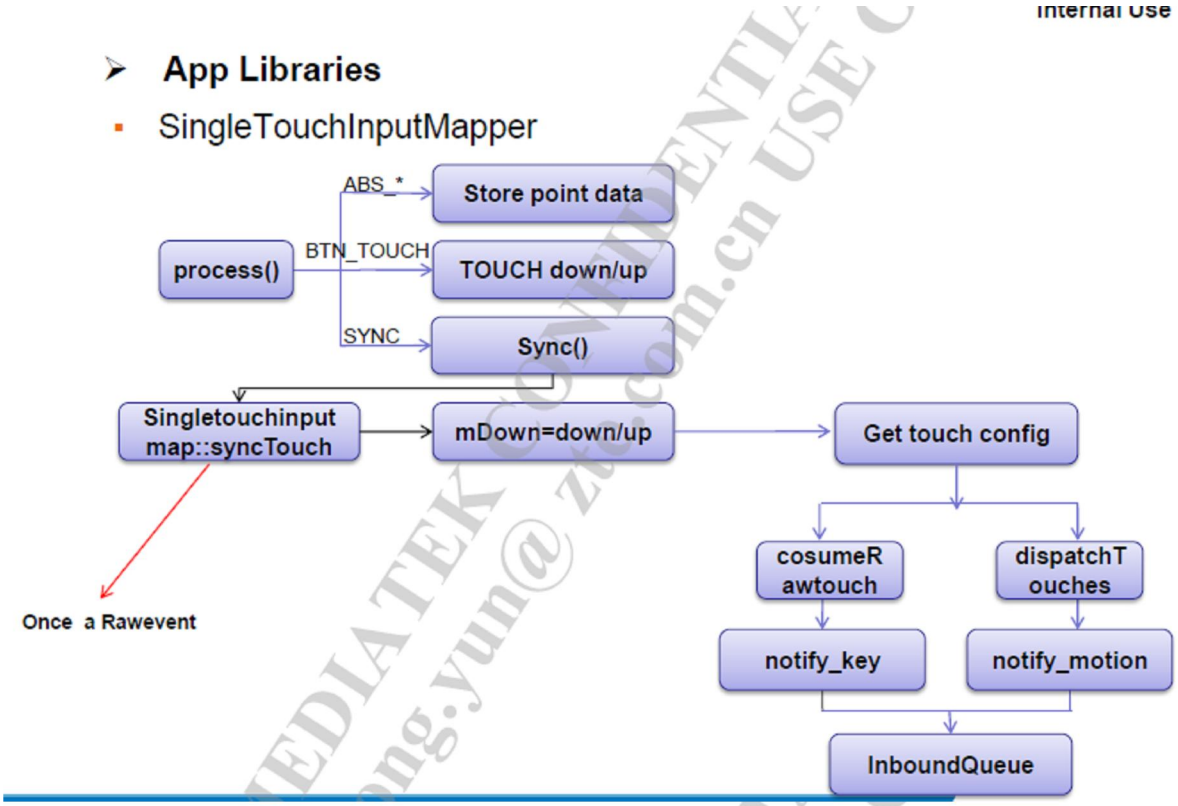
- InputMapper

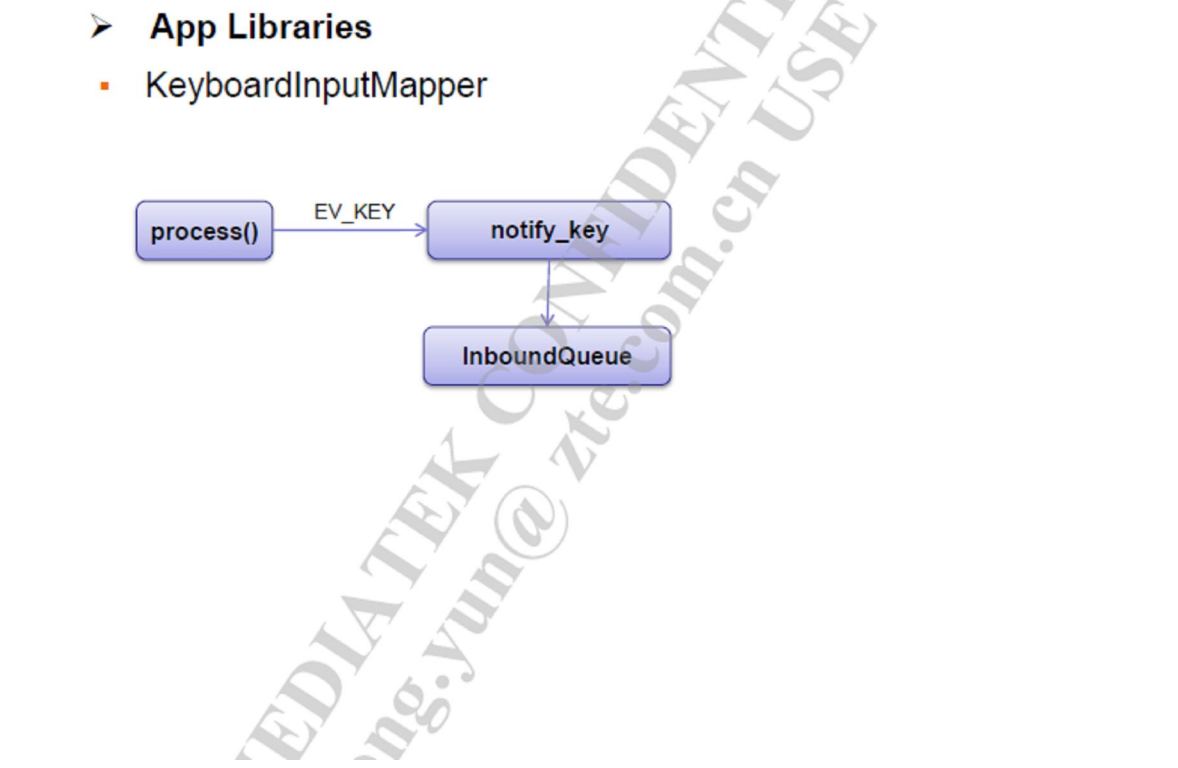
Mapper Type	Event Type	Event Code	Introduce
MultiTouchInputMapper	EV_ABS	ABS_MT_POSITION_X ABS_MT_POSITION_y	The input device is a muti-touch touchscreen
SingleTouchInputMapper	EV_ABS EV_KEY	ABS_X/ABS_Y BTN_TOUCH	The input device is a single-touch touchscreen
KeyBoardInputMapper	EV_KEY	KEY_*	The input device is a keyboard
TrackballInputMapper	EV_KEY EV_REL	BTN_MOUSE REL_X/REL_Y	The input device is a trackball
SwitchInputMapper	EV_SW	SW_*	The input device has switchs

App Libraries

- MutiTouchInputMapper







四.MTK平台触摸屏驱动架构

(1).mtk_tpd.c

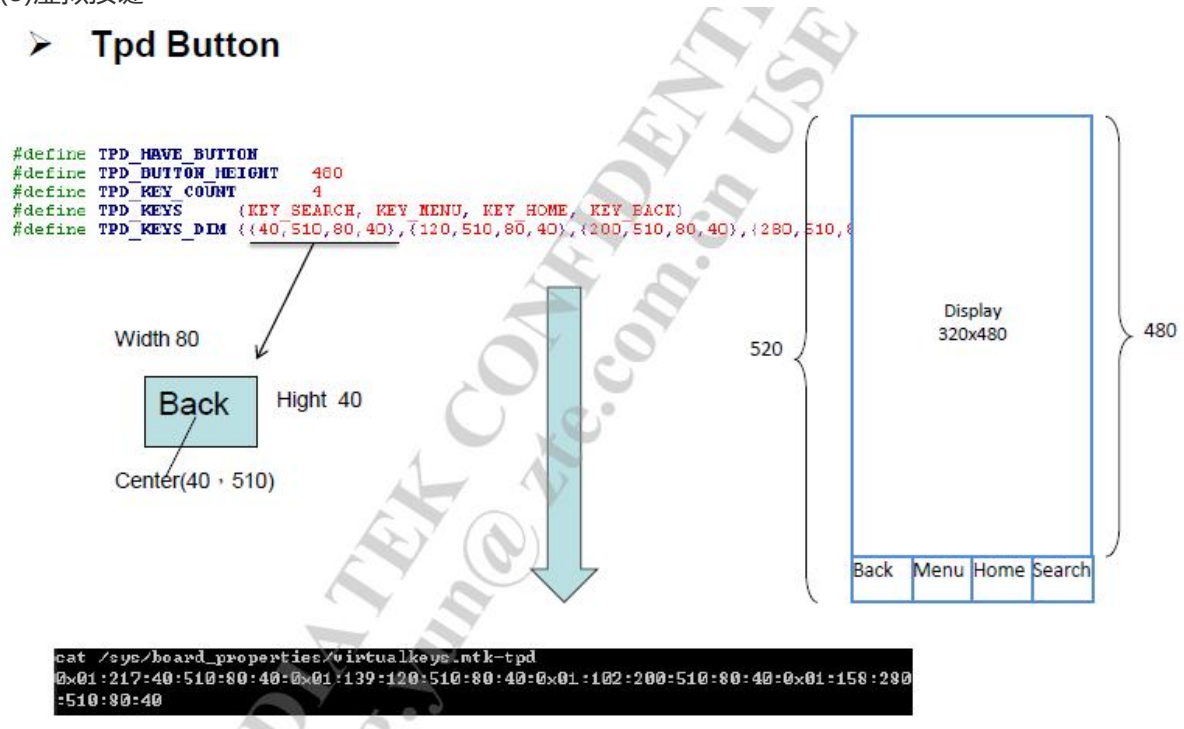
```
struct tpd_device
{
    struct input_dev *dev;
    struct input_dev *kpd;
    struct timer_list timer;
    struct tasklet_struct tasklet;
    int btn_state;
};

struct tpd_device  *tpd = 0;
```

probe中设置设备支持的事件

然后留下接口

(2)具体在驱动文件中I2C设备的注册以及控制

(3)虚拟按键



meiy