触摸屏主机端调试指南

Project name	[Project name]
Document ref	[Document ref]
Version	[Major revision number] .1
Release date	Mar. 23 2011
Owner	Xinming Wang
Classification	[DOCUMENT CLASSIFICATION]
Distribution List	[DISTRIBUTION LIST]
Approval	

R3-B4-A, South Area, Shenzhen Hi-Tech Industrial Park,
Shenzhen, Gungdong, P.R. China
ZIP :518057
T +86 755
F +86 755 26712499
E support@focaltech-systems.com

www.focaltech-systems.com

Revision History

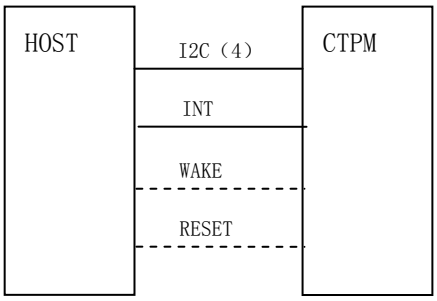
Date	Version	List of changes	Author + Signature
Mar.23 2011	1.0	Initial draft	Wangxm

Table of Contents

1 前言

该文针对 FocalTech 的电容屏触控模组（CTPM）通过 I2C 接口在 Android 系统上的应用，介绍了 Host Driver 的工作原理，CTPM 的交互接口和开发过程的调试步骤以及对常见问题的分析。

2 Host 与 CTPM 连接示意图

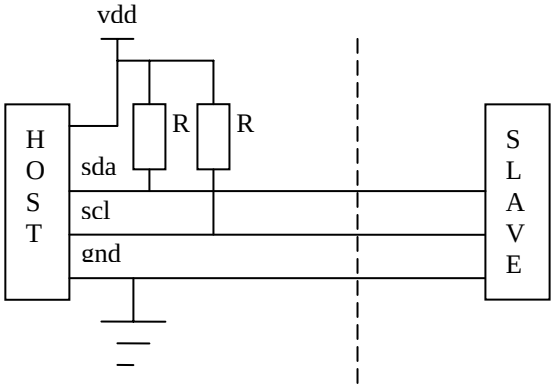


3 接口检查

3.1 硬件接口检查

3.1.1 物理连接

1） Host 端 I2C 连接示意图：



2） I2C 上拉电阻 R：一般为 4.7K；

3） INT 脚（CTPM->Host），电平和 IO 电平一致；

4） Wake/Reset 脚（Host->CTPM）；

a) Wake 脚用于唤醒，但可以省略，转为直接采用 Reset 脚复位；

b) 但 Wake Pin 也必须从 IC 引出并提供一个测试点用于 Download。

3.1.2 电压配置

1） VCC 电压：

标准是 3.3v（±0.3v），但 2.8v 以上都能正常工作；

2） IO 电压：

c) 同 VCC 一样（3.3v）

两者尽量避免有压差，否则将导致漏电；比如：两者都设定为 3v，而避免一个是 3v 而另一个是 2.8v 等；

d) 1.8V（±0.1v）；

e) 该电压配置需与软件设定匹配。

3.1.3 中断触发方式

有 Trigger 和 Polling 两种方式，推荐使用 Trigger 方式。

注意：Host 端的使用方式必须与 CTPM 端的设定一致。

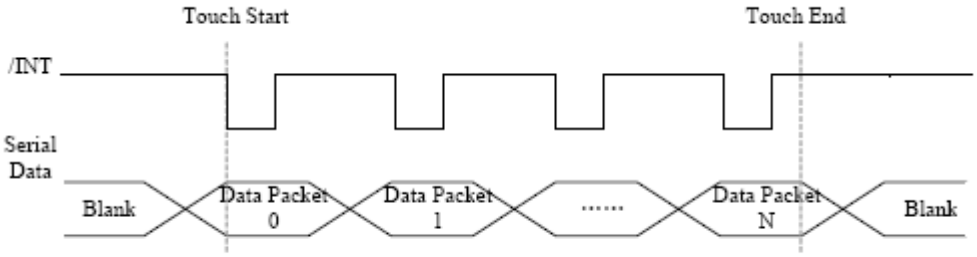
1） Trigger

该方式的示意图如下图所示，假如报点频率为 60Hz，则触摸发生时，CTPM 端约每隔 16ms 发出一次中断，每个中断周期包含所有触摸点的数据；

为了确保不丢点，Host 端需在一个中断周期内完成数据读取；

确认是下降沿触发还是电平触发，推荐采用下降沿触发。

INT 负脉冲的时间约 2-10ms



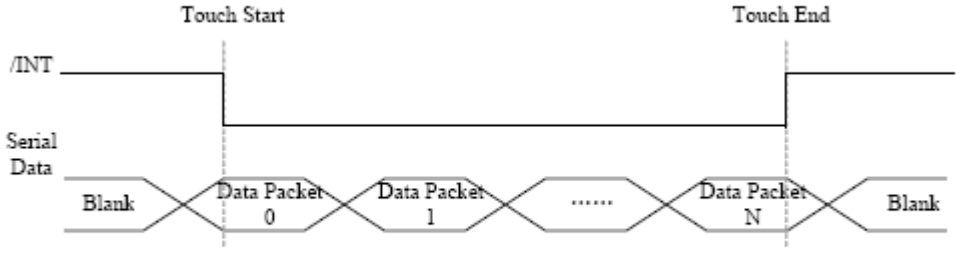
Interrupt trigger mode

2） Polling

该方式的示意图如下图所示，触摸发生时，CTPM 端只发出一次中断并保持 INT 为低电平，触摸结束时 INT 才恢复高电平；

假如报点频率为 60Hz，为了确保不丢点，Host 端需配合 CTPM 端的报点周期读取数据，读快了会重复，读慢了会丢帧；

只能采用下降沿触发。



Interrupt query mode

3.1.4 总线读写接口

I2C 总线的读写函数没有提供可直接编译的 Sample Code，一般由 Host 端去实现：

1） 可以直接操控 I2C 控制芯片的硬件寄存器来实现读写函数；

2） 但一般都是利用 Linux 定义的 I2C 驱动体系结构来实现 I2C 硬件的驱，即通过调用 i2c_transfer(……)实现；

3） Host 一般都会有两个以上的 I2C 设备，有现成的 I2C 访问接口可以复用。

3.2 软件接口检查

3.2.1 I2C 设定

1） I2C 地址

a) 检查地址设定是否与立项表一致；

b) 检查该地址是否与 Host 的其它 I2C 设备的地址相冲突。

c) 注意地址使用习惯，有 7 位（不含读写位）表示和 8 位（含读写位 bit0）表示的差异；比如 8 位地址 0x70，则对应的 7 位地址就是 0x38。

2） I2C 速度

范围应在 100KHz-400KHz。

3） I2C Buffer

检查 I2C 控制芯片设置的 Buffer 长度，它关系到升级时单次可传输的最大数据量。

3.2.2 坐标原点

确认（0，0）点的位置，一般是设定在左上角。

3.2.3 分辨率

确保 Host 端采用的分辨率与 CTPM 设定的分辨率一致，如果有虚拟按键区（Virtual Key），则请明确：

1） 虚拟按键的上报方式:

a) 上报坐标：就如同普通触摸区一样，上报具体的坐标值；

b) 上报特定值：不上报坐标值，而是上报一个与 Host 约定的一个符号，比如上报数字 1，2，3，4 等或字母 a，b，c，d 或其它任何约定的字符。

2） 分辨率是否包含虚拟按键区；

3） 与普通触摸区的界限。

3.2.4 协议

1） 对 FT5x06，推荐使用标准（STD）协议，但也可使用 26bytes 协议（主要是考虑到老客户的兼容性）；

2） 对 FT5202，使用 26bytes 协议，C 版开始增加了标准协议；

3） 两种协议有不同的 sample code 和文档，请注意区分。

3.2.5 屏厂 ID

每个大的屏厂都有对应的 ID 号，分配列表如下：

屏厂名称	Vendor ID
EDT	0x42
凌巨	0x50
欧菲光	0x51
牧东	0x53
EELY	0x54
莱宝	0x55
普容(诺宝)	0x56
超声	0x57
天恩和	0x58
BYD	0x59
TRULY	0x5a
南玻	0x5b
TPK	0x5c
BM（宝明）	0x5d
Topseed	0x5e
宇顺	0x5f
立德	0x60

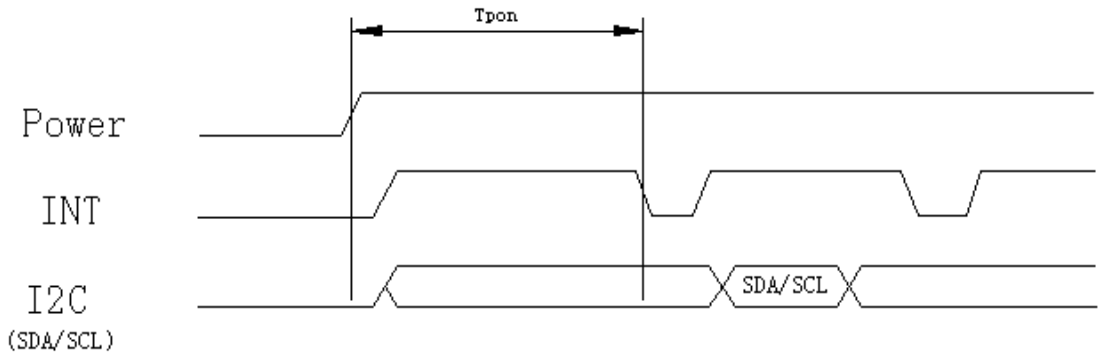
该 ID 号存储于 ID_G_FT5201ID 寄存器。

4 Host 时序检查

4.1 Power On

该时序衡量从设备开机到 CTPM 开始报点的延迟时间，一般为 300ms 左右。

1） 时序图

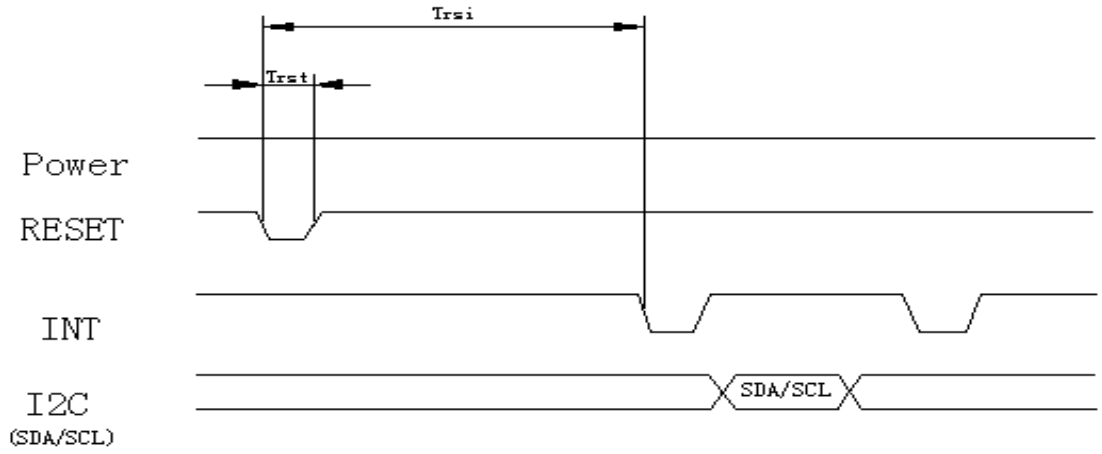


2） Tpon > 300ms

4.2 Reset

该时序衡量当 Host 需要复位 CTPM 时，从发出复位指令到 CTPM 到 CTPM 开始报点的延迟时间，一般为 300ms 左右。

1） 时序图

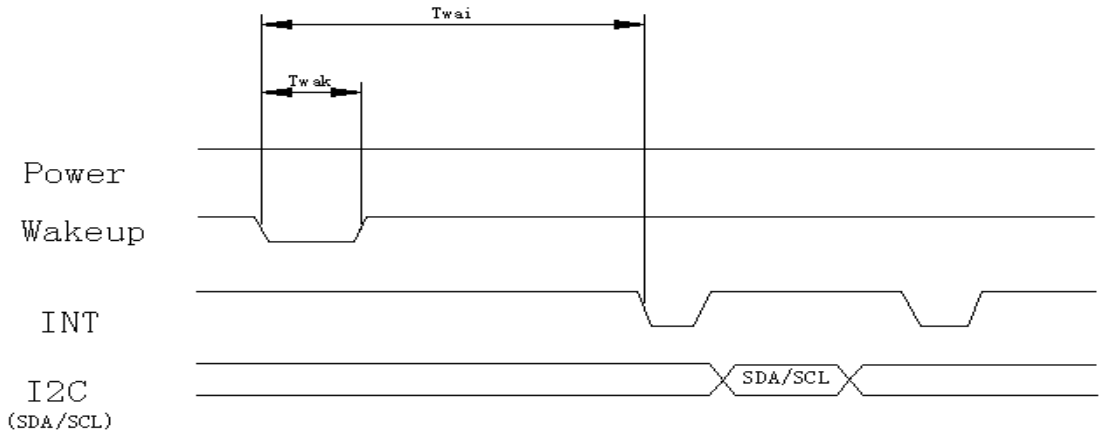


2） Trst > 1ms, Trsi > 300ms

4.3 Wake Up

该时序衡量在 CTPM 处于 Hibernate（或叫 Sleep）状态下，当 Host 设备发出唤醒指令到 CTPM 开始报点的延迟时间，一般为 300ms 左右。

1） 时序图



2） Twak > 1ms, Twai > 300ms

5 功能调试

此小节的特定寄存器地址对应标准 I2C 协议。

5.1 寄存器检查

CTPM 连上 Host 系统后，如有异常，可通过检查以下两个寄存器来查看 CTPM 是否在正常运行：

1） 工作模式寄存器

寄存器地址：0x00；

寄存器值：

a) 0x00：表示工作在 work 模式下；

b) 0x04：表示工作在 factory 模式下；

c) 其它：表示 CTPM 没有工作。

2） FW 版本号寄存器

寄存器地址：0xA6；

寄存器值：格式为 0xXX，从 0x10 开始递增，一般都不会超过 0x30。

5.2 触摸数据获取

数据获取方式有两种：

1） 直接使用 FocalTech 提供的 sample code

Sample code 文件名为 I2C_PP_Std.c, I2C_PP_Std.h 和 I2C_PP_Spec.ppt，调用对应的接口 fts_ctpm_get_touch_info(……)就可以得到所有触摸信息；

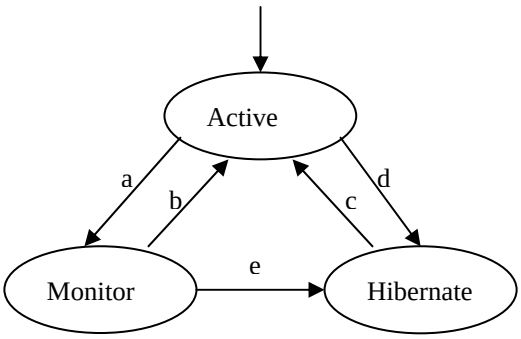
2） 读取相应寄存器

通过阅读 FocalTech 提供的 Application Note 文档中的协议介绍，可直接访问相应寄存器获取触摸信息；

5.3 功耗模式检查

总共有三种功耗模式，分别为 Active，Monitor 和 Hibernate，受寄存器 0xA5 控制；

5.3.1 模式转换图



转换条件如下：

a:无触摸约 60 帧后，帧数目可修改；

b:有触摸时；

c:Host 拉低 Wake 脚；

d:Host 发送指令；

e:同 d。

5.3.2 Active 模式

1） 为正常工作模式，寄存器值为 0x00；

2） 此模式功耗为：约 8-14mA，具体依屏体大小，报点率大小以及触点数目而定。

5.3.3 Monitor 模式

1） 为省电模式，寄存器值为 0x01；

2） 此模式将降低扫描频率（25Hz）；

3） 此模式功耗为：约 2-6mA（与屏体大小相关）。

5.3.4 Hibernate 模式

1） 为休眠模式，寄存器值为 0x03；

2） 此模式功耗为：约 20-50uA

5.4 软件升级（Upgrade）

该功能默认是要支持的，主要是便于开发阶段的调试；

参考 Sample Code 及 PPT 文档，正确调用 fts_ctpm_fw_upgrade(……)即可完成 CTPM 的软件升级，注意事项如下：

1） 使用 FocalTech 提供的目标 FW 文件(.i 文件)按照 I2C_PP_Spec.ppt 的说明调用升级接口；

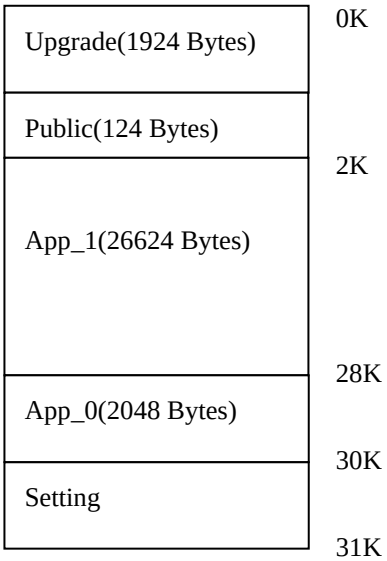
2） Sample Code 不能更改，只能直接调用，因事关时序和逻辑；

3） 升级完后 CTPM 会自动复位；

4） 如果升级失败或升级过程中掉电，可以重新升级。

5.4.1 FT5x06 升级

1） Flash Laypout



Setting 区结构图：

0x6ffa	0x6ffb	0x6ffc	0x6ffd	0x6ffe	0x6fff	0x7800	0x7801	0x7802	0x7803
F/W 的长度		F/W 的长度取反		Ecc	Ecc 取反	Slave_address	Slave_address 取反	I/0 电压	I/0 电压取反

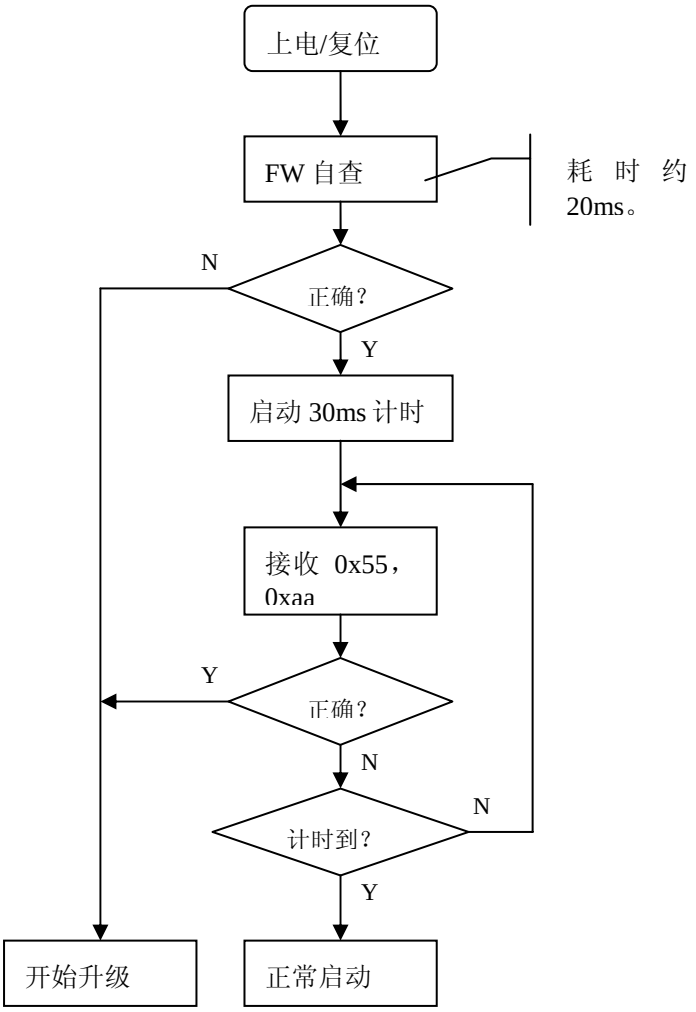
Ecc：表示 F/W 的所有数据的异或校验

I/0 电压：0x00: 3.3V

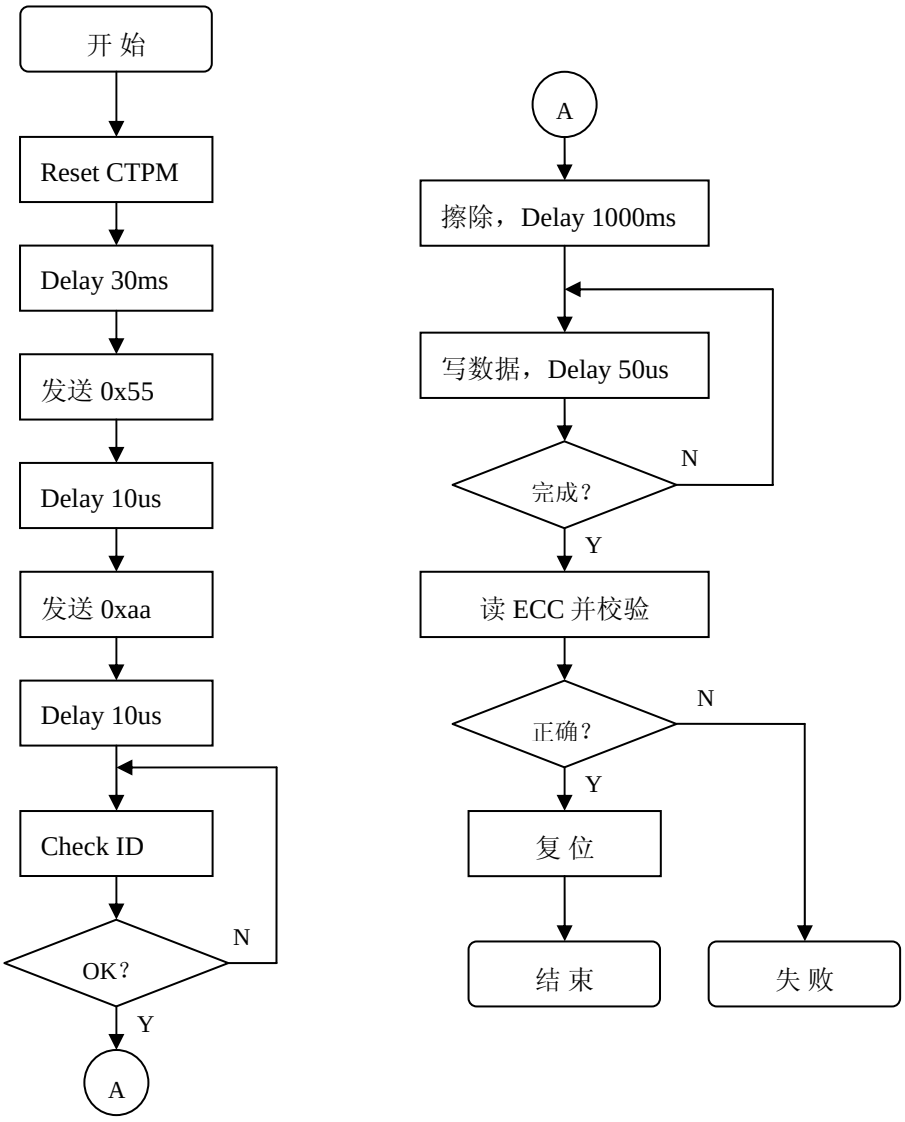
0x01: 1.8V

2） 升级流程图

a) CTPM 端升级启动流程

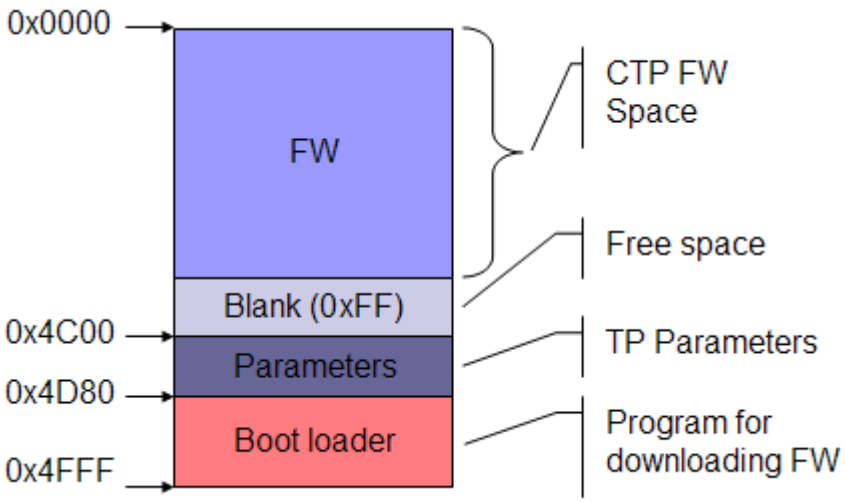


b) Host 端升级流程图



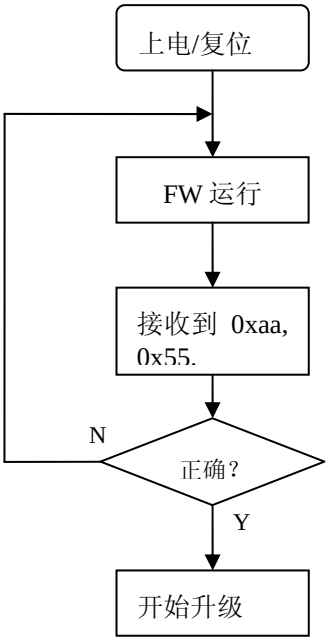
5.4.2 FT5202 升级

1) Flash Layout

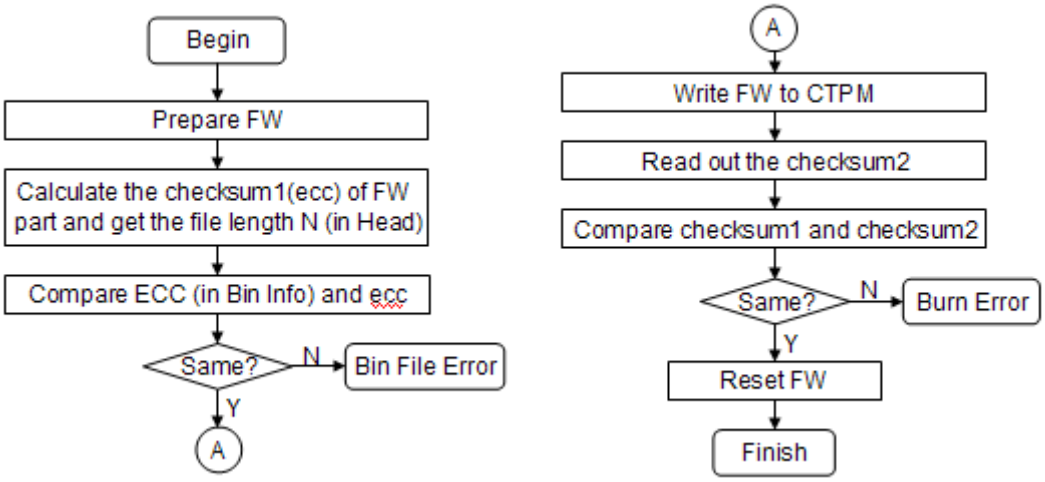


2) 升级流程图

a) CTPM 端升级启动流程



b) Host 端升级流程图



6 Host Driver 编写参考

6.1 代码结构

```
#include <linux/i2c.h>
#include <linux/input.h>
#include <linux/interrupt.h>
#include "ft5x0x_ts.h"
……
//Define some data structure here.
……

static int fts_i2c_rxdata(char *rxdata, int length)
{
        ……
}
static int fts_i2c_txdata(char *txdata, int length)
{
        ……
}
static void fts_work_func()
{
        fts_ctpm_get_touch_info();
}
static irqreturn_t fts_ts_irq(int irq, void *dev_id)
{
   struct FTS_TS_DATA_T *ft5x0x_ts = dev_id;

   if (!work_pending(&ft5x0x_ts->pen_event_work))
   {
      queue_work(ft5x0x_ts->ts_workqueue, &ft5x0x_ts->pen_event_work);
   }

   return IRQ_HANDLED;
}
static int fts_ts_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
        ……
        INIT_WORK(fts_work_func);
        ……
        request_irq(fts_ts_irq);
```

```
        ……
        input_allocate_device();
        ……
        input_register_device();
        ……
}
static int fts_ts_remove(struct i2c_client *client)
{
        ……
}
static struct i2c_driver fts_ts_driver =
{
      .probe   = fts_ts_probe,
      .remove  = fts_ts_remove,
      .id_table  = ft5x0x_ts_id,
      .driver   =
      {
         .name = FT5X0X_NAME,
      },
};
static int __init fts_ts_init(void)
{
        return i2c_add_driver(&fts_ts_driver);
}
static void __exit fts_ts_exit(void)
{
        i2c_del_driver(&fts_ts_driver);
}
module_init(fts_ts_init);
module_exit(fts_ts_exit);
```

6.2 要素说明

1） 代码实现结构不唯一，以上仅为一个参考；

2） input 系统设置

a) 坐标范围设定；

b) 坐标上报；

c) Android 版本差异：有的只支持单点，有的可支持多点，还有的可支持方向，压力等指标。

3） 中断拆分

必须分为上半部和下半部实现，上半部响应硬件中断（把下半部放入任务队列），下半部实现触摸信息读取（可被中断）；

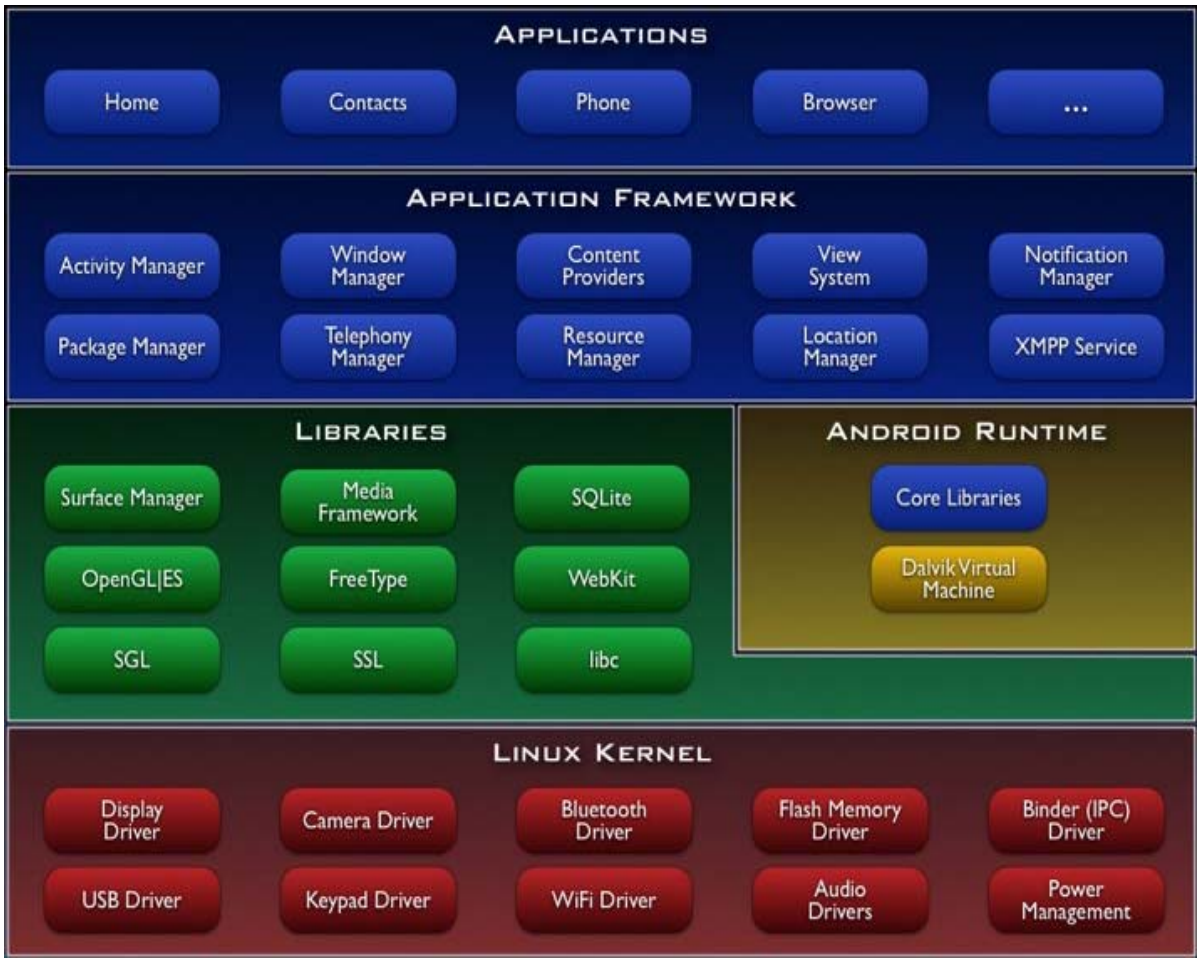
4） 地址设定

一般存放在 i2c_board_info 结构体。

7 Andriod 系统与 Driver 的关系

7.1 Android 系统架构

由 5 部分组成，分别是：Linux Kernel、Android Runtime、Libraries、Application Framework、Applications，如下图所示。



7.2 与 Driver 的关系

1） 无直接关系，Driver 是普通的 Linux Driver；

2） 触摸功能支持

不同版本的 Android 对触摸功能的支持细节有差异；

3） ABS_MT 事件集

a) ABS_MT_POSITION_X：接触面的形心的 X 坐标值；

b) ABS_MT_POSITION_Y：接触面的形心的 Y 坐标值；

c) SYN_MT_REPORT：上报某一坐标的数据；

d) SYN_REPORT：完成一次上报；

e) ABS_MT_TOUCH_MAJOR：接触面的长轴；

f) ABS_MT_TOUCH_MINOR：接触面的短轴，如果是圆形接触面，这个参数可以省略；

g) ABS_MT_WIDTH_MAJOR：接触工具的长轴；

h) ABS_MT_WIDTH_MINOR：接触工具的切面的短轴，如果是圆形，此参数可以省略；

i) ABS_MT_PRESSURE：接触工具对接触面的压力大小；

j) ABS_MT_ORIENTATION：描述随圆的转动趋势；

k) ABS_MT_TOOL_TYPE：触摸工具的类型；

l) ABS_MT_BLOB_ID ：标识多边形的边的集合；

m) ABS_MT_TRACKING_ID ：用来区别一个触摸动作的周期；

n) ABS_MT_APPROACH_X/Y：接触位置和技术工具位置可以被用来派生倾斜。

4） input 系统接口

input 系统负责把由 Driver 层得到的信息传递给上层（Andriod 层），其接口如下：

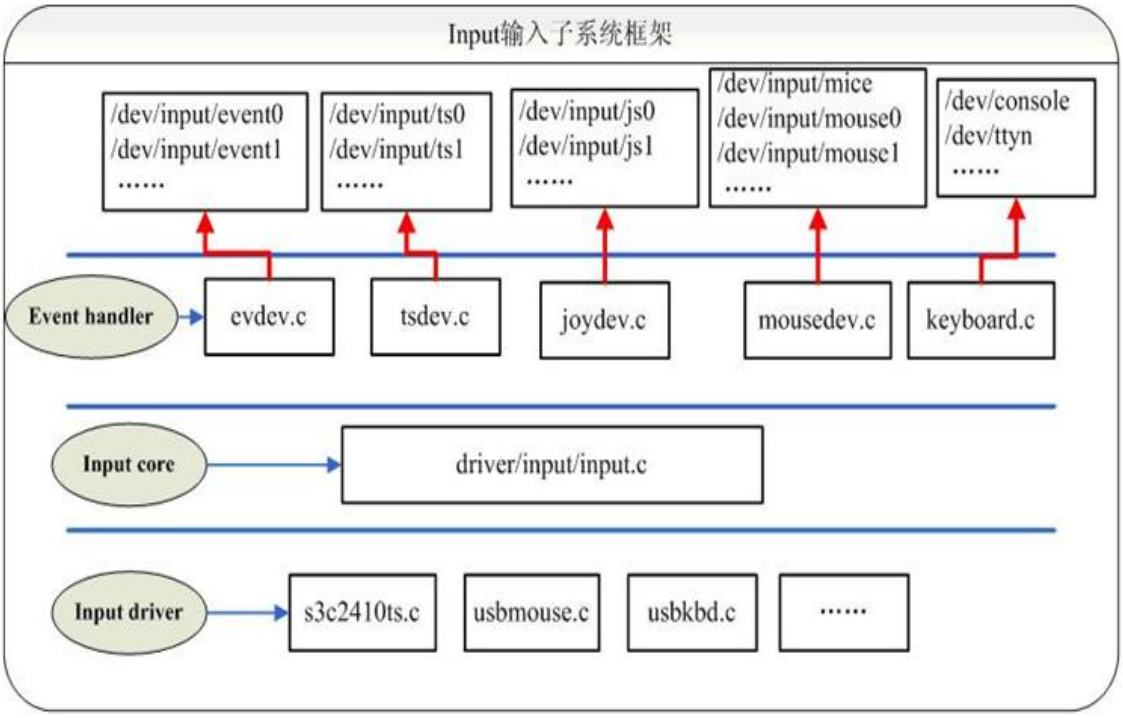
a) set_bit()：告知支持那些事件；

b) input_set_abs_params()：设置参数范围；

c) input_report_abs()：上报具体参数。

5） input 机制

a) input 子系统框架图



8 常见问题解析

8.1 触摸数据异常

1） 地址冲突，确保 Host 端的多个 I2C 设备有不同的地址；

2） 坐标提取不当；

3） 其它系统原因。

8.2 响应延迟

1） 中断响应机制处理不当，数据读取放在了上半部执行；

2） I2C 调度不良，如硬件与 I2C 体检架构衔接实现；

3） 其它系统原因。

8.3 升级失败

1） 设定的单次数据传输长度超出了特定平台设置的 I2C 可支持的最大长度；

2） 时序配合不当；

3） 其它系统原因。

8.4 坐标丢失

1） 中断响应不及时，漏掉了中断；

2） 其它系统原因。

8.5 坐标重复

1） 一个中断响应了两次；

2） MTK 平台只有电平触发；

3） 其它系统原因。

8.6 事件错误

1） 事件提取位置错误；

2） 检查与 ID 的配合；

3） 其它系统原因。

8.7 疑难问题应对

1） 思路：

a) 区分问题归属，明确原因出在 Host 端还是 CTPM 端；

2） 方法：

a) 拆下 CTPM,单独测试；

b) 跟踪总线层数据；

c) 检查 INT 信号；

d) 检查 VDD 电压水平。