Fremont Micro Devices

AN-22018

FT62F13XX TOUCH_LIB

使用说明

Rev1.0.1
www.fremontmicro.com

日期	版本	描述
2022-03-25	V1.0.0	初版
2022-05-10	V1.0.1	修改：普通 Touch 外设资源占用 T1,不可使用;

目录

1. 简介 6

2. TOUCH_lib 资源使用 6

2.1 带算法库 6

2.1.1 内存资源 6

2.1.2 外设资源 6

2.2 无算法库(输出原始值) 6

2.2.1 内存资源 6

2.2.2 外设资源 7

3. 软件函数接口说明 7

3.1 `void TOUCH_INITIAL (void);` 7

3.2 `void TSC_Start (void);` 7

3.3 `unsigned char TSC_DataProcessing (void);` （带算法库有效） 7

3.4 `unsigned char TSC_GetStrongest(void);` （带算法库有效） 7

3.5 `unsigned char TSC_GetSingle(void);` （带算法库有效） 7

3.6 `unsigned int TSC_GetPrevData(unsigned char KeyNum);` （带算法库有效） 8

3.7 `unsigned char TSC_GetDelta(unsigned char KeyNum);` （带算法库有效） 8

3.8 `void TSC_Reset(void);` （带算法库有效） 8

3.9 `unsigned char TSC_Sampling(void);` （无算法库有效） 8

3.10 `unsigned int TSC_GetSmpleData(unsigned char keyNum);` （无算法库有效） 8

3.11 `unsigned char TSC_SleepPrcoessing(void);` （低功耗使能有效） 8

4. 头文件参数说明 8

4.1 按键定义 8

4.2 按键数据处理定义（带算法库） 9

4.3 防干扰选项定义（带算法库） 10

4.3.1 扫描频率微调定义 10

4.3.2 被动调频定义 10

4.4 低功耗定义（带算法库） 10

5. 电容取值注意事项 11

6. 应用举例 11

6.1 带算法举例 11

6.2 无算法(输出原始值)举例 14
7. 硬件设计说明 14
7.1 PCB 设计的基本原则 14
7.1.1 电源 14
7.1.2 地线 14
7.1.3 布局 16
7.2 感应按键 16
7.2.1 形状 16
7.2.2 尺寸 17
7.2.3 按键的间距 17
7.3 走线 17
7.3.1 基本走线原则 17
7.4 弹簧应用注意事项 17
7.5 绝缘材料 18
7.5.1 覆盖层材料及厚度 18
7.6 其他注意事项 18
7.6.1 PCB 板的清洁 18
联系信息 19

表格索引

Figure 1 网格大小 15
Figure 2 The distance between the touch plate and the bottom plate 15
Figure 3 走线包覆一圈地线..... 16
Figure 4 PCB 布局..... 16
Figure 5 感应按键形状 17
Figure 6 PCB 走线..... 17

1. 简介

FT62F13XX 具有触摸传感控制器，提供了电容式触摸传感功能的简单解决方案。为方便应用程序开发人员便捷稳定的使用触摸功能，FMD 提供触摸库文件,应用程序开发人员通过调用库函数，可以迅速的完成触摸功能的应用。

本文档提供了 FT62F13XX 触摸库的使用说明，介绍了 TOUCH_LIB 所占用的芯片资源，方便评估芯片选型；详述了软件函数接口，帮助开发应用程序；并提供了硬件设计相关注意事项，尽可能的减少来自电源和 pcb 的干扰。

2. TOUCH_lib 资源使用

2.1 带算法库

2.1.1 内存资源

- ROM、SRAM 使用说明

  功能选项不同，按键个数不同其资源使用都不相同，各库资源使用如下表所示：

库类型	描述	ROM*14		SRAM*8	
lpp	功能选项	ROM0	ROM1	SRAM0	SRAM1
touch_soft.lpp	普通Touch	1,151	3*K	52	8*K
touch.lpp	增强Touch、无防干扰	1,262	3*K	55	8*K
touch_cs_ac.lpp	增强Touch、主动调频防干扰	1,400	4*K	62	8*K
touch_cs_uac.lpp	增强Touch、被动调频防干扰	1,473	4*K	65	8*K
touch_lp.lpp	增强Touch、无防干扰、低功耗	1,650	3*K	59	8*K+2*Ks
touch_cs_lp_ac.lpp	增强Touch、主动调频防干扰 、低功耗	1,790	4*K	64	8*K+2*Ks
touch_cs_lp_uac.lpp	增强Touch、被动调频防干扰 、低功耗	1,844	4*K	67	8*K+2*Ks

- SRAM 的指定地址

  0x20，0x21，0x72~0x79，0x120~0x15F 已用于按键处理，不能使用；其中 0x120~0x15F 的占用长度与按键个数 K 有关：

  占用长度 = 8 * K

  例如：K= 1 按键，占用长度=8Byte，占用 0x120~0x128 ；K=8 按键，占用长度=64Byte，占用 0x120~0x15F；

- 扫键占用的 MCU 周期(单位：us)

  占用的 MCU 周期会随项目设置改变，具体数值由 Touch Tools 测试输出。

- 扫键时间

  MCU 扫键时间会随项目设置、线路参数改变，具体数值由 Touch Tools 测试输出。

2.1.2 外设资源

- 普通 Touch 库

  TIM1、LVD 全占用，不能用作其它功能；

- 增强通 Touch 库

  TIM0、TIM1、LVD 全占用，不能用作其它功能；

2.2 无算法库(输出原始值)

2.2.1 内存资源

- ROM、SRAM 使用说明

库类型	描述	ROM*14		SRAM*8	
lpp	功能选项	ROM0	ROM1	SRAM0	SRAM1
touch.lpp	输出原始值	397	3*k	15	2*K

- 扫键占用的 MCU 周期(单位：us)

  占用的 MCU 周期会随项目设置改变，具体数值由 Touch Tools 测试输出。

- 扫键时间

  MCU 扫键时间会随项目设置、线路参数改变，具体数值由 Touch Tools 测试输出。

2.2.2 外设资源

TIM0、TIM1、LVD 全占用，不能用作其它功能；

3. 软件函数接口说明

3.1 `void TOUCH_INITIAL (void);`

功能：配置 touch 相关寄存器；
参数：None；
返回值：None；
说明：此函数为 Touch 相关寄存器的初始配置和触摸按键相关的外设初始化（IO,TIM0,TIM1），在系统初始化位置调用。

3.2 `void TSC_Start (void);`

功能：启动触摸扫描并获取原始扫描数据；
参数：None；
返回值：None；
说明：此函数用于启动触摸，放在程序主循环中循环调用，调用越频繁效果越好；

3.3 `unsigned char TSC_DataProcessing (void);` （带算法库有效）

功能：按键数据处理；
参数：None；
返回值：返回 1 或 0；
说明：此函数用于处理 touch 数据并生成最强键与有效键，处理完成返回 1 否则返回 0，放在程序主循环中循环调用；

3.4 `unsigned char TSC_GetStrongest(void);` （带算法库有效）

功能：返回当前按压信号最强的按键通道号；
参数：None；
返回值：最强按键通道号（取值范围 1-8）；
说明：调用函数获取当前按下的最强按键值 1-8，如果 key8 为最强，则返回 8；

3.5 `unsigned char TSC_GetSingle(void);` （带算法库有效）

功能：返回当前按下的按键通道号；
参数：None；
返回值：当前按下的按键通道号；

说明： 8 位数据，按键通道号对应数据位，如 0X01 表示按键 1 按下，0X03 表示按键 1、2 按下；**特别说明：在打开防干扰选项时，如果检测到干扰环境，此函数返为 0 则为不输出多键。**

3.6 unsigned int TSC_GetPrevData(unsigned char KeyNum)；（带算法库有效）

功能：获取对应按键扫描数据；
参数：KeyNum（取值范围 1-8）；
返回值：对应按键扫描数据；
说明：通过参数，获取对应按键扫描数据；

3.7 unsigned char TSC_GetDelta(unsigned char KeyNum)；（带算法库有效）

功能：读取所选按键相对于基线的变化值；
参数：KeyNum，按键通道（取值范围 1-8）；
返回值：按键变化值；
说明：读取对应的按键变化值 ，KeyNum 不在 1-8 的范围内时， 返回按压信号最强的按键对应的变化值；

3.8 void TSC_Reset(void)；（带算法库有效）

功能：复位 touch 功能；
参数：None；
返回值：None；
说明：复位基线，清除按键值；

3.9 unsigned char TSC_Sampling(void)；（无算法库有效）

功能：查询是否采样完成；
参数：None；
返回值：返回 1 或 0；
说明：当所有按键采样完成返回 1 否则返回 0；

3.10 unsigned int TSC_GetSmpleData(unsigned char keyNum)；（无算法库有效）

功能：获取按键采样值；
参数：None；
返回值：返回对应按键数据；
说明：此函数返回的按键值为未做任何处理的值；

3.11 unsigned char TSC_SleepPrcoessing(void)；（低功耗使能有效）

功能：低功耗的处理；
参数：None；
返回值：返回 1 或 0；
说明：当有按键唤醒时返回 1，否则返回 0；

4. 头文件参数说明

4.1 按键定义

- 对应芯片 IO 引脚的映射定义

```
#define    KEY0_INDEX_MAP          7
```

```
#define    KEY1_INDEX_MAP        6
#define    KEY2_INDEX_MAP        5
#define    KEY3_INDEX_MAP        4
#define    KEY4_INDEX_MAP        2
#define    KEY5_INDEX_MAP        3
#define    KEY6_INDEX_MAP        0
#define    KEY7_INDEX_MAP        1
```

其中 KEYx_INDEX_MAP 对应的值为芯片触摸引脚的 IO 列表下标值，其列表为 KEY1~KEY8 的顺序定义，

IO 列表[8]=

{

KEY1/PC1,//下标 0

KEY2/PC0,//下标 1

KEY3/PB7,//下标 2

……

KEY8/PB2,//下标/7

}

例如：
物理按键 KEY0_INDEX_MAP 对接到芯片的 KEY8/PB2，那么就定义 KEY0_INDEX_MAP 为 7；
物理按键 KEY6_INDEX_MAP 对接到芯片的 KEY1/PC1，那么就定义 KEY6_INDEX_MAP 为 0；

- 按键个数

```
#define    KEY_NUMBER            8
```

4.2 按键数据处理定义（带算法库）

- 放大倍数，算法所需的数值，与实际应用电容相关，建议在 80(取值范围 30~250) ，可参考电容取值注意事项说明；

```
#define MULTIPLE_1                                    80
```

- 最强键滤波次数，建议在 4 (取值范围 1~6)

```
#define STRONG_FILTER_1                               4
```

- 单键滤波次数，建议在 4 (取值范围 1~6)

```
#define SINGLE_FILTER_1                               4
```

- 基线向下更新速度，数值设置越大，更新速度越慢(取值范围 1~127)

```
#define BASE_LINE_DOWN_SPEED_1                        64
```

- 基线向上更新速度，数值设置越大，更新速度越慢(取值范围 1~127)

```
#define BASE_LINE_UP_SPEED_1                          64
```

- 基线快速向上更新，数值设置越大，更新速度越慢(取值范围 1~255)

#define BASE_LINE_EQUAL_UP_SPEED_1 100

- 基线快速向下更新，数值设置越大，更新速度越慢(取值范围 1~255)

#define BASE_LINE_EQUAL_DOWN_SPEED_1 200

- 整体信噪比，用于判断最强键的条件之一，建议在 15-40 范围内的取值(取值范围 0-80)

#define SIGNAL_NOISE_RATIO_1 30

- 基线重新扫描更新，用于当同时有多个按键变化量大于有效阈值的一半时，基线重新扫描更新(取值范围 1~ KEY_NUMBER)。

#define MEANTIME_MAX_KEY_NUM_1 KEY_NUMBER

4.3 防干扰选项定义（带算法库）

4.3.1 扫描频率微调定义

为了提高触摸在特定干扰下的稳定性，代码具有跳频功能；通过设置三个频率参数，支持三个频点跳变；建议频率设置为 4M，并在固定频率范围跳频；

```
#define FREQ_CHANGER_1     0X20     //扫描频率 1，固定值，不建义修改
#define FREQ_CHANGER_2     0X00     //扫描频率 2，固定值，不建义修改
#define FREQ_CHANGER_3     0XE0     //扫描频率 3，固定值，不建义修改
const unsigned char ucFreachooseArray[3] = {
    FREQ_CHANGER_1,
    FREQ_CHANGER_2,
    FREQ_CHANGER_3
};
```

4.3.2 被动调频定义

选择被动调频时，即生成该定义；正常状态为非调频，如果检测到有干扰将转入三个频率的调频切换，调频时间结束则会退回正常状态。

- 调频时长定义，单位是次数。建议控制时间在 12 秒~15 秒 = 扫描周期*次数；

#define RESONANCE_CHANGER_FREQ_COUNT_FILTER 1000

- 检测干扰条件；**该值由 Touch 工具根据所有按键的值生成的，建议不要修改；**

#define RESONANCE_DATA_FILTER 10

4.4 低功耗定义（带算法库）

- 由唤醒状态进入低功耗状态的计数，扫描处理所有按键完成并且无按键按下计 1 次，n 次无按键按下就会进入低功耗；建议 20 以上（取值范围 1~255）

#define SLEEP_MODE_WAIT_NUMBER 250

- 低功耗状态下看门狗唤醒频率；**由 TouchTool 选择产生，建议不要修改；**

```
#define SLEEP_RATE_SET0     0B00001111
#define SLEEP_RATE_SET1     0B00010001
```

- 低功耗下做全按键扫描的时间计数，与看门狗唤醒频率有关，建议每 2s 扫一次；**由 TouchTool 选择产生，建议不要修改；**

#define SLEEP_UPDATA_BASE_WAIT_NUMBER 15

- 低功耗下做按键扫描的基线更新计数，值越大更新越慢；（取值范围 1~255）

#define SLEEP_MODE_BASE_LINE_COUNT 10

- 低功耗下按键唤醒的有效阈值；**由 TouchTool 测试产生，建议不要修改；**

#define SLEEP_KEY_ON_VALUE 4

- 低功耗下扫描的按键组数，组数越多唤醒时间越长，产生功耗越大，最多可分 4 组；**由 TouchTool 测试产生，建议不要修改**；

#define SLEEP_SCAN_KEY_GROUP 4

- 低功耗下扫描的按键组配置定义，与组数 SLEEP_SCAN_KEY_GROUP 相对应；**由 TouchTool 测试产生，建议不要修改；**

```
#define TouchIoMaskB      0x03
#define TouchIoMaskC      0xFC

#define SleepTrisB_0      0x00
#define SleepTrisC_0      0x03
#define SleepTK_0         0x03
#define SleepTrisB_1      0xC0
#define SleepTrisC_1      0x00
#define SleepTK_1         0x0C
#define SleepTrisB_2      0x30
#define SleepTrisC_2      0x00
#define SleepTK_2         0x30
#define SleepTrisB_3      0x0C
#define SleepTrisC_3      0x00
#define SleepTK_3         0xC0
```

5. 电容取值注意事项

- 取值范围根据 PCB 特性会有所区别，建议在 2.2nf～10nf 间选取。
- 取值和放大倍数有关联性，建议放大倍数=80，此时键值输出码应要在 730～1100 间。
- 当应用需要选取比较大的电容时，放大倍数需要同步调整，保证输出值和放大倍数的关联，键值输出码应在下面公式计算所得范围内;
  假设 P=放大倍数
  公式：键值=P*3413/平方根(P*4096/2.328) ~ P*5120/平方根(P*4096/2.328)

6. 应用举例

6.1 带算法举例

- 无低功耗

```
int main(void)
{
    SYS_INITIAL(); //系统时钟、IO 初始化
```

```
    TOUCH_INITIAL();  //触摸按键初始化
    WDT_INITIAL(); //看门狗初始化
    PEIE = 1;  //使能外设中断
    GIE = 1;   //使能全局中断
    LED(); //led.c  led 提示灯初始化
    while (1)
    {
        CLRWDT();      //清看门狗
        TSC_Start();  //在主循环中频繁调用，检测各个按键的数据
        if(TSC_DataProcessing()==1) //在主循环中频繁调用，处理检测到的数据
        {
            strongest = TSC_GetStrongest(); //获取最强按键值
            if(Sav_strongest != strongest)
            {
                LED_Scan(); //led.c  按键按下点亮对应的 led
            }
            Sav_strongest = strongest;
        }
    }
}
```

- 低功耗

```
void main(void )
{
    SYS_INITIAL ();//系统时钟、IO 初始化
    ////////////////////////////////////////////////
    TOUCH_INITIAL();//触摸按键初始化
    WDT_INITIAL(); //看门狗初始化低功耗定时，请勿关闭
    ////////////////////////////////////////////////
    PEIE = 1;  //使能外设中断
    GIE = 1;  //使能全局中断
    LED(); //led.c  led 提示灯初始化
    ////触摸按键低功耗处理标志////////////////////
    bSleepUserFlag = 1; //为 1 表示可进入低功耗，用户可以使用这个标志来控制是否要进入低功耗状态
    bSleepUserWakeFlag = 0; //设置为 1 表示强制退出低功耗，用户可以使用这个标志来控制是否强制退出低耗
    ////触摸按键低功耗处理标志////////////////////
    //////demo_test////////////////////
    TRISA |= 0B00000001; //PA0 设为输入
    //////demo_test////////////////////
    while (1)
    {
          CLRWDT();  //清看门狗
          ///////////触摸按键低功耗处理，请勿修改//////////////////////
```

```
    if(bSleepScanEnable == 1)
    {
        //////demo_test////////////////////
        if(PA0==1)//检测到 PA0 高电平将 bSleepUserWakeFlag 置 1
        {
            bSleepUserWakeFlag = 1; //设置为 1 表示强制退出低功耗，用户可以使
            用这个标志来控制是否强制退出低耗
        }
        //////demo_test////////////////////
        //被按键唤醒退出低功耗模式或 bSleepUserWakeFlag=1 强制退出低功耗
        if(TSC_SleepPrcoessing()==1)
        {
            GIE = bIEBackupsFlag;
            bSleepUserWakeFlag = 0; //清除强制退出低功耗标志
            //////--start--退出低功耗模式用户自定义处理/////
            //用户自定义处理
            //////--end---退出低功耗模式用户自定义处理/////
        }
    }
    else
    //////////触摸按键低功耗处理，请勿修改//////////////////////
    {
        TSC_Start(); //按键扫描
        if(TSC_DataProcessing()== 1) //返回 1 表示所有按键处理完成一次。
        {
            strongest = TSC_GetStrongest(); //获取最强按键值
            if(Sav_strongest != strongest)
            {
                LED_Scan(); //led.c  按键按下点亮对应的 led
            }
            Sav_strongest = strongest;

            if(bSleepMode == 1) //1 为表示要进入低功耗
            {
                bIEBackupsFlag = GIE;
                GIE = 0;
                ////start--进入低功耗用户自定义处理//////////////////

                ////end---进入低功耗用户自定义处理//////////////////
            }
        }
}
 //////////触摸按键低功耗处理，请勿修改///////////////
Sleep_Enable();
```

```
            //////////触摸按键低功耗处理，请勿修改///////
        }
    }
```

6.2 无算法(输出原始值)举例

```
void main(void )
{
    SYS_INITIAL ();//系统时钟、IO 初始化
    TOUCH_INITIAL();//触摸按键初始化
    WDT_INITIAL(); //看门狗初始化
    PEIE = 1;  //使能外设中断
    GIE = 1;  //使能全局中断
    while(1)
    {
        CLRWDT();//清看门狗
        TSC_Start();//按键扫描
        if(TSC_Sampling()== 1) //采样完成
        {
            for(key = 0; key < KEY_NUMBER; key++)
            {
                uiTempBuffer[key]= TSC_GetSmpleData(key); //读取对应按键的键值
            }
        }
    }
}
```

7. 硬件设计说明

7.1 PCB 设计的基本原则

7.1.1 电源

- 较高稳定度
  触摸 IC 测量的是电容的微小变化，对电源的稳定度要求较高，电源的纹波和噪声要小，应注意避免由电源串入外界强干扰，优先采用线性电源。根据实际的电源纹波情况，决定是否单独用 LDO 稳压。
- 星形取电
  触摸芯片最好用一根独立的走线从板子的供电点/LDO 取电，不要和其他电路共用电源回路。如果做不到完全独立，也应该保证供电的电源线先到触摸芯片的电源然后再引到其它的电路的电源，这样可以减小其他电路在电源上产生的噪声对触摸芯片的影响
- 滤波处理
  在触摸芯片的每组 VDD 与 VSS 以及 VDDA 与 VSSA 之间并联一个滤波电容(如 1uF 贴片电容)，并尽量靠近 IC，另外也可根据实际情况加装滤波电路，以滤除噪声，保持供电稳定，提高系统抗干扰能力。

7.1.2 地线

- 星形接地

触摸芯片的地线不要和其他电路公用，应该单独连到板子电源输入的接地点。

- 走线

  电源与地平行走线并尽量拉等宽与等距的线，并先经过滤波电容再到 MCU 的 VDD 和 VSS。尽量在电源端保留一块铺地，同时 GND 走线面积 > VDD 走线面积。

- 铺地布局

  单面铺地，触摸盘与铺地的距离以 1mm 左右为宜，建议网格铺地。网格大小如 Figure 2 所示

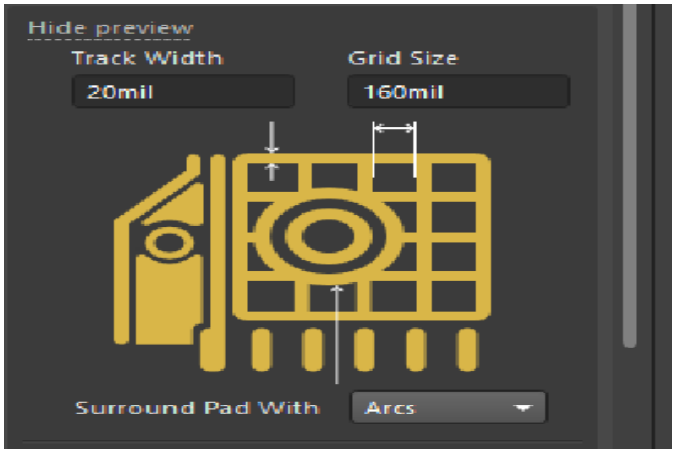

Figure 1　网格大小

以弹簧按键为例，如图 Figure 3 所示触，摸盘与铺地的距离为 1mm。

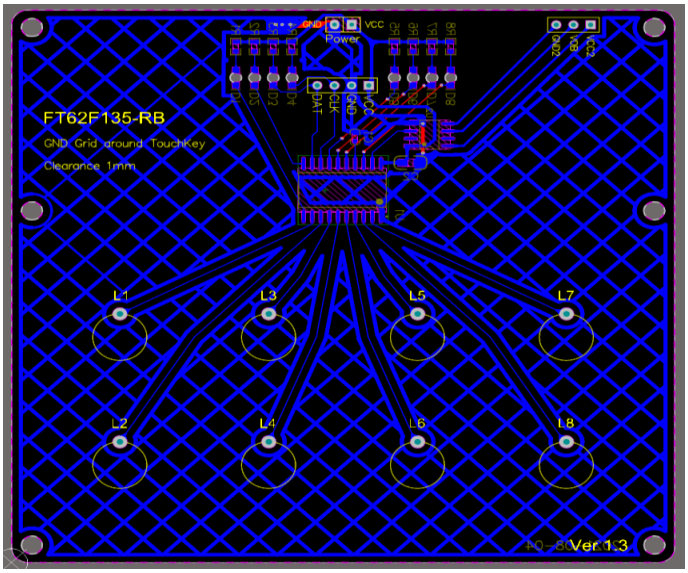

Figure 2　The distance between the touch plate and the bottom plate

如果无法网格铺地，建议在走线周围包覆地线，如图 Figure 4 所示

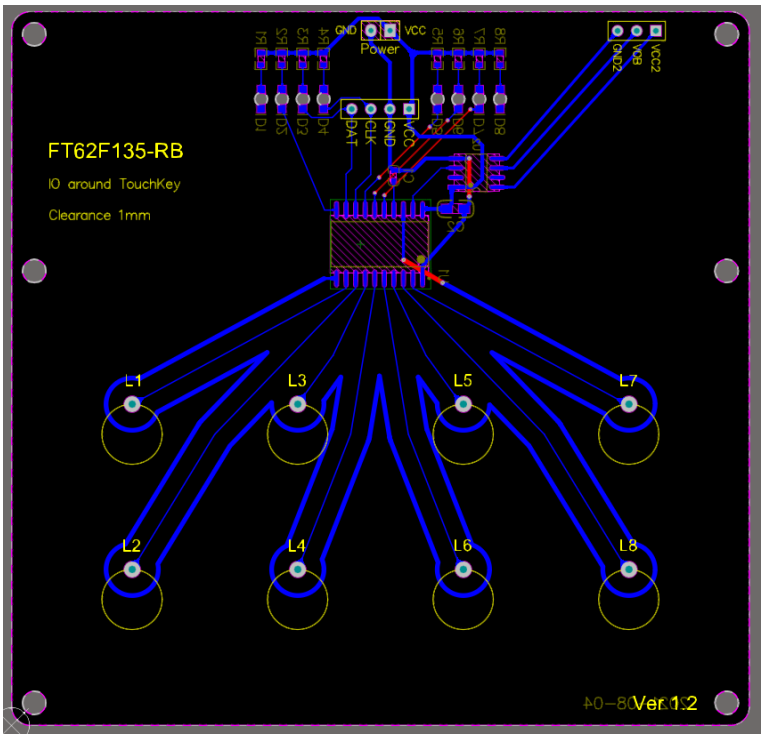

Figure 3　走线包覆一圈地线

7.1.3　布局

- 芯片的位置：
  如图 Figure 6 所示，触摸芯片和触摸盘应放置在同一块 PCB 板上，在 PCB 板空间允许的情况下，应尽量将触摸芯片放置在触摸板的中间位置，使触摸芯片的每个触摸通道的引脚到感应盘的距离差异最小。

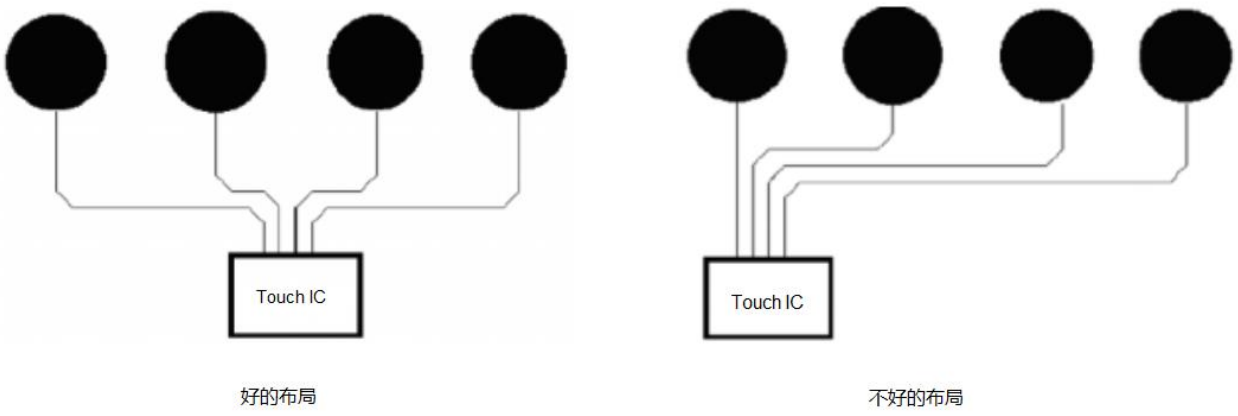

Figure 4　PCB 布局

7.2　感应按键

7.2.1　形状

如图 Figure 7 所示，原则上可以做成任意形状，但尽量集中在正方形、长方形、圆形等比较规则的形状以确保良好的触摸效果，避免将触摸按键设计成窄长的形状（规则形状的触摸效果要比不规则的好得多）。推荐做成边缘圆滑的形状，如圆形或六角形。

Figure 5　感应按键形状

7.2.2　尺寸

根据经验，建议感应按键的直径为 10mm~20mm 为宜(typ:13mm)。

7.2.3　按键的间距

各个感应按键间的距离要尽可能的大一些。

7.3　走线

7.3.1　基本走线原则

保证走线尽量细，到触摸 IC 的距离尽量短。建议走线宽度不大于 10mil (0.25mm)，长度越短越佳，以确保信号的稳定。为避免干扰源增加，请在路由电容感应输入时不使用或者使用最小的过孔数量(最多为 2 个)，最佳走线及铺地如图 Figure 8 所示，PAD 直径 13mm， PAD 于 GND 之间间距为 1mm，PAD 之间距离为 9mm，PAD 与 MCU 走线宽度为 0.25mm。

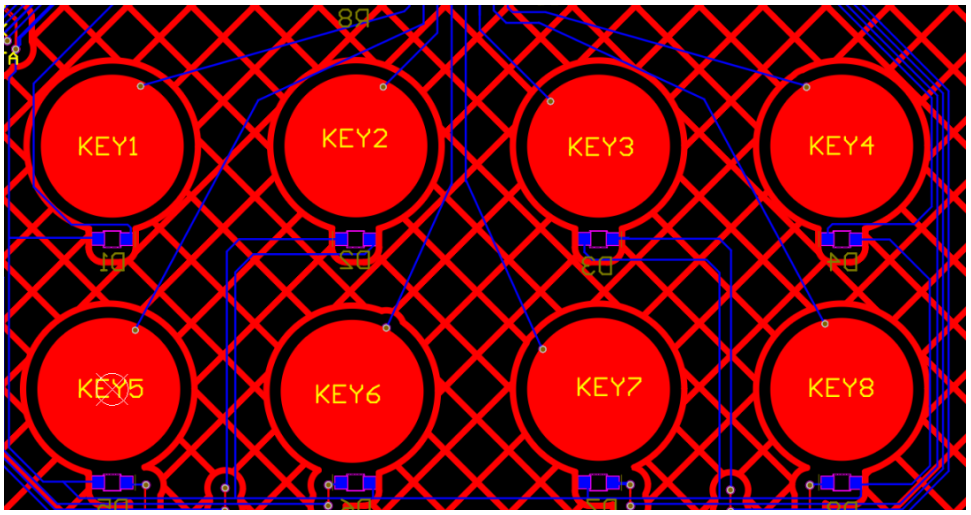

Figure 6　PCB 走线

7.4　弹簧应用注意事项

- 使用带弹簧的感应盘，将感应盘顶在面板上。
- 因为弹簧具有较高的侧面灵敏度，因此要尽可能地使相邻的弹簧传感器彼此远离，以防止检测错误。
- 与感应盘类似，外覆层厚度越厚，灵敏度越低。
- 弹簧的直径越大，灵敏度越高。
- 弹簧的金属丝厚度越大，灵敏度越高。

7.5 绝缘材料

7.5.1 覆盖层材料及厚度

- 面板必须选用绝缘材料，可以是玻璃、聚苯乙烯、聚氯乙烯（pvc）、尼龙、树脂玻璃等。在生产过程中，要保持面板的材质和厚度不变，面板的表面喷涂必须使用绝缘的油漆。
  在电极不变的情况下，面板的厚度和材质决定灵敏度。建议实际应用时选用合适的介质材质和厚度。
- 通常，在厚度、面积相同的情况下，介电常数越大，灵敏度越高。但在正常应用中， 推荐使用介电常数适中的材质，比如树脂玻璃等。介电常数过小，会导致灵敏度差；介电常数过大，发生误动作的几率会变大。
- 触摸感应面板的灵敏度与介质的厚度有关，同一介质厚度越薄，灵敏度越高，厚度越厚，灵敏度越低。建议触摸按键的面板厚度在 1~5mm 之间。

7.6 其他注意事项

7.6.1 PCB 板的清洁

残留的助焊剂和污物，在恶劣的温度和湿度环境下会严重影响芯片工作的稳定性。

联系信息

Fremont Micro Devices Corporation

#5−8, 10/F, Changhong Building
Ke-Ji Nan 12 Road, Nanshan District,
Shenzhen, Guangdong, PRC 518057

Tel: (+86 755) 8611 7811
Fax: (+86 755) 8611 7810

Fremont Micro Devices (HK) Corporation

#16, 16/F, Block B, Veristrong Industrial Centre,
34−36 Au Pui Wan Street, Fotan, Shatin, Hong Kong SAR

Tel: (+852) 2781 1186
Fax: (+852) 2781 1144

http://www.fremontmicro.com