目录

1 赛元 TOUCHKEY MCU 应用指南总体描述

本文档是赛元 Touchkey MCU 触控的应用指南，主要介绍如何使用赛元提供的触摸按键库文件以及触控上位机如何调试参数。其特点如下：

- ●高灵敏度模式可适应普通触摸按键、隔空触摸按键、滑轮滑条、接近感应等对灵敏度要求较高的触控应用
- ●高灵敏度模式具有很强的抗干扰能力
- ●可实现多路触摸按键及衍生功能
- ●高灵活度开发软件库支持，低开发难度
- ●自动化调试软件支持，智能化开发

Touchkey MCU 具有触控低功耗应用模式，其中，12 个触摸按键 500ms 唤醒时芯片功耗最低可至 9uA@5V。用户通过使用赛元提供的触控按键库文件，可选择触控模式并快速简单实现所需的触摸功能。用户可以通过下表的信息选择最适合当前应用的触控模式：

说明	高灵敏度模式
特点	●超强抗干扰能力，可通过 10V 动态 CS ●超高灵敏度
适用的应用	●弹簧触摸按键应用 ●隔空触摸按键应用 ●接近感应应用 ●滑轮滑条应用 ●对灵敏度要求较高的触控应用
如何进入模式	根据触控应用选择赛元所提供的对应触控库类型进行使用
库体说明	4.3.1 C51 高灵敏触控软件库移植 4.3.4 ARM 高灵敏触控软件库移植
对应的库文件	C51 系列： “SC9XF8XXX_HighSensitive_Lib_Tn_Vx.x.x.LIB” ARM 系列： KEIL 平台：“SC32F1XXX_HighSensitive_Lib_Tn_Vx.x.LIB” IAR:平台：“SC32F1XXX_HighSensitive_Lib_Tn_Vx.x.a”
注意事项	●T1 库适用于弹簧，导电膜，TP 膜等覆盖面板紧贴的触控应用，应用于邻键干扰小的场景，支持组合按键 ●T2 库应用于隔空类型的应用，且按键个数至少 3 个以上，应用于邻键干扰大的场景，不支持组合按键
选择说明	通常状况下建议使用此高灵敏度模式，将会获得更佳的使用体验。

赛元系列Touchkey MCU 触摸芯片供电电源注意事项：

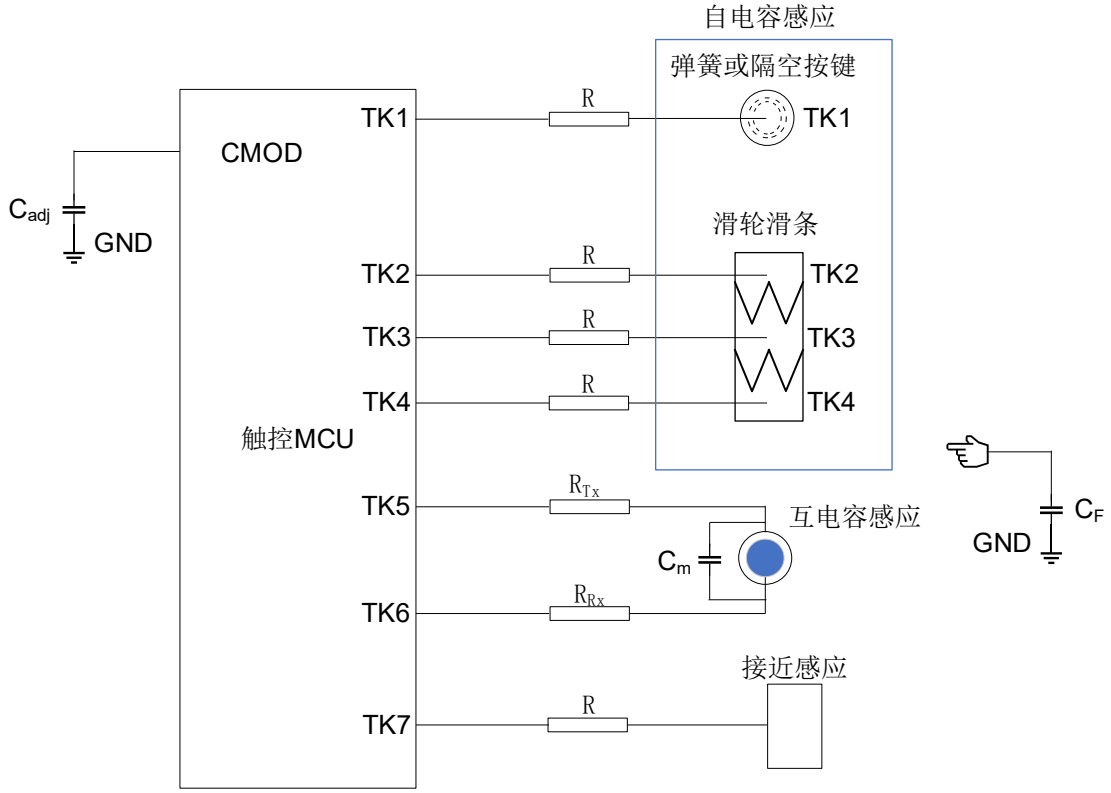
- 触摸芯片供电电源范围：参考不同芯片规格书要求范围使用
- 触摸芯片供电电源纹波：推荐触摸芯片工作电源电压纹波应控制小于 100mv，最大不超过 200mv。

2 赛元触控技术

2.1触摸按键基本类型

赛元触控 MCU 内置电容检测电路，具有高达 100：1 的信噪比和 0.1pF 的分辨率，配备完善的触控库算法，兼具灵敏度和可靠性，同时触摸感应按键相对于传统的机械按键有寿命长、占用空间少、易于操作等诸多优点。

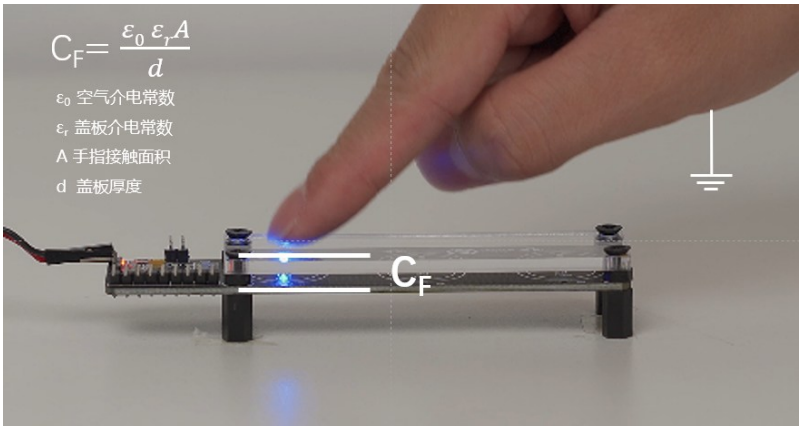
赛元触控 MCU 支持多种触控按键类型，弹簧按键、隔空按键、滑轮滑条、接近感应、 防水功能这些触控功能类型。其中也支持自电容感应和互电容感应两种感应类型，而普通的触控按键和滑轮滑条这两种按键类型则是自电容感应类型。下图为触控 MCU 的各种按键类型的基本电路图。



2.2电容感应类型

2.2.1 自电容感应

以普通的触控单按键为例，PCB 板上的触摸感应按键感应盘就是电容的一个极板，而手指构成了另一个极板，PCB 材料本身或者 PCB 板上覆盖的介质就算电容中间的绝缘体，因而构成了一个电容器。



如上图所示的操作，此时就构成了一个平行板电容器。而电容式触控则是基于手指电容(C_F)实现的。感应盘

与手指之间的电容值可通过公式 2-1 计算

公式 2-1 手指电容

$$C_F = \frac{\varepsilon_0 \varepsilon_r S}{d}$$

其中：

ε_0 = 空气介电常数

ε_r = 盖板介电常数

S = 手指与盖板的接触面积

d = 盖板厚度

C_F 为手指电容。手指电容与电荷传导介质材料的介电常数、手指接触面积成正比，与电荷传导介质材料厚度成反比。同时按键感应盘与 PCB 之间有寄生电容 C_P，则当手指触摸按键感应盘的时候，此时的总感应电容为

公式 2-2 触摸感应盘时的总感应电容

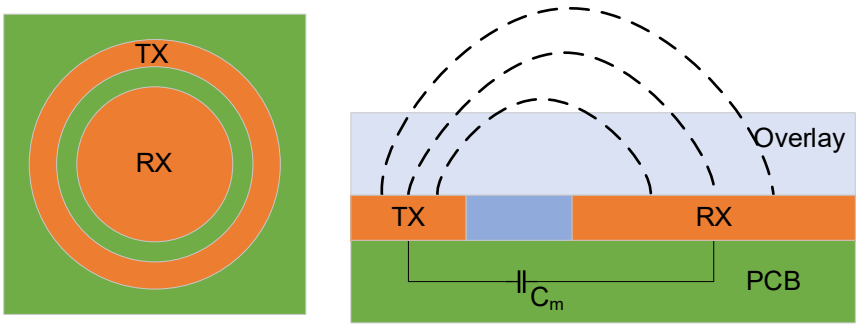
$$C = C_F + C_P$$

如果没有进行触摸动作，那么 C= C_P。根据总感应电容的变化来判断触摸/无触摸状态

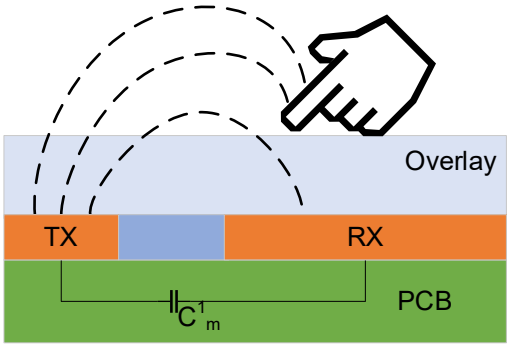
2.2.2 互电容感应

在互电容感应中，使用两个触控 IO 口，一个为发送电极 TX，一个为接收电极，TX 引脚提供激励，电荷在 C_m 中积累，在 RX 电极上接收到的电荷与两个电极之间的互电容 C_m 成正比。

俯视图　　边视图



当手指接近电极时，手指与电极之间会产生寄生电容 C_F，而 C_m 和 C_F 并联。由于驱动 TX 的能量是恒定的，电荷量恒定，手指接近后，C_m 电荷量减少，变为 C^1_m，RX 电极上接收到电荷量减少，由此通过检测 RX 电极上的电荷检测触摸/无触摸状态。

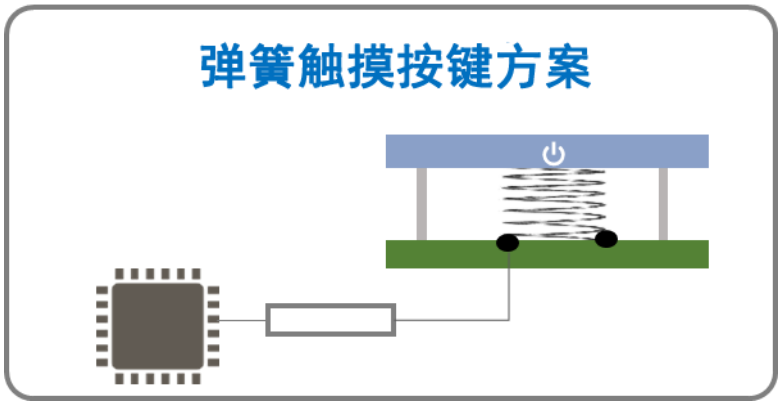


2.3弹簧与隔空按键

2.3.1 弹簧触摸按键

赛元触控 MCU 在弹簧触摸按键方案中使用弹簧或者 PCB 铜箔作为电容式感应盘，具体实际应用请用户根据场景需要进行选择。弹簧触摸按键具有灵敏的电容感应和抗干扰能力，因此是当前电容式触控的主流方案。

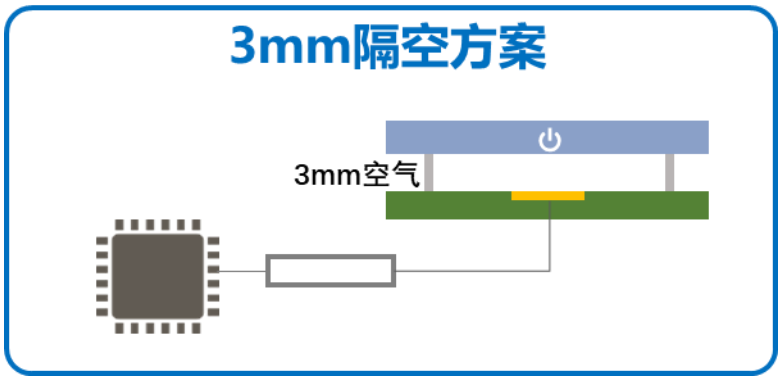
弹簧触摸按键是基于自电容感应技术进行实现的，具体软件库的调用可参考 4.4 完成用户程序和赛元触控软件库的融合。



2.3.2 隔空触摸按键

赛元触控 MCU 内置高灵敏触控电路，信噪比高达 100：1，可以轻松实现隔空触控，以替代传统弹簧按键方式。PCB 上覆铜作为电容式感应盘，感应盘与按键面板之间允许 3mm 以内的空气间隔，可满足超薄触控面板、微型曲面、超厚面板、手套操作等应用场景，优化产品结构与外观，大幅度降低产品生产工艺难度，提升生产良率。这类触摸按键可用于白电、小家电、厨电、工业控制、IOT、消费类等触控类产品。

隔空触摸按键是基于自电容感应技术进行实现的，与弹簧触摸按键有些不同，在应用于实际场景中，隔空触摸按键覆盖面板的时候，按键感应盘不与面板直接接触。具体软件库的调用可参考 4.4 完成用户程序和赛元触控软件库的融合



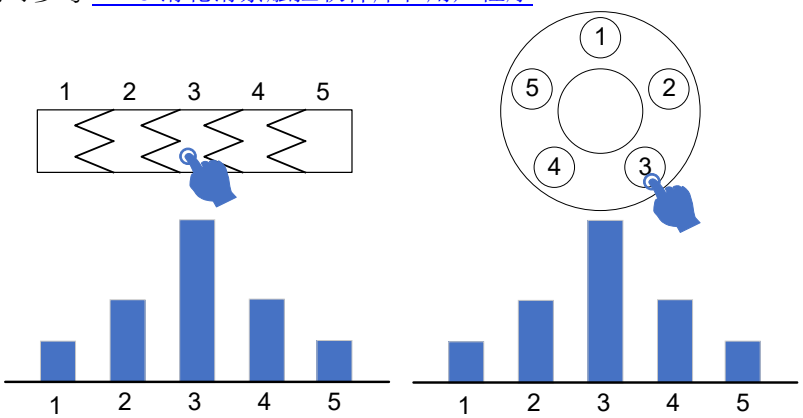
2.4滑轮滑条

当使用滑轮滑条触摸按键时，至少需要占用 3 个 TK 口，现已应用于灯光调节、智能音箱音量调节等应用。

用滑条进行举例，滑条触摸按键是由多个相邻放置的电容感应 PAD 组成的，当触摸其中某一个滑条感应段的时候会让相邻的滑条段有一定程度的电容变化，通过软件算法来进行处理被触摸的滑条感应段和相邻的滑条感应段的变化值，从而得到此时手指触摸的几何中心位置，输出相应的档位值。

下图为手指触摸某一滑条感应段的时候，对应的滑条感应段的电容变化量，滑轮也是同理。

具体软件库的调用可参考 4.4.3 滑轮滑条触控软件库和用户程序

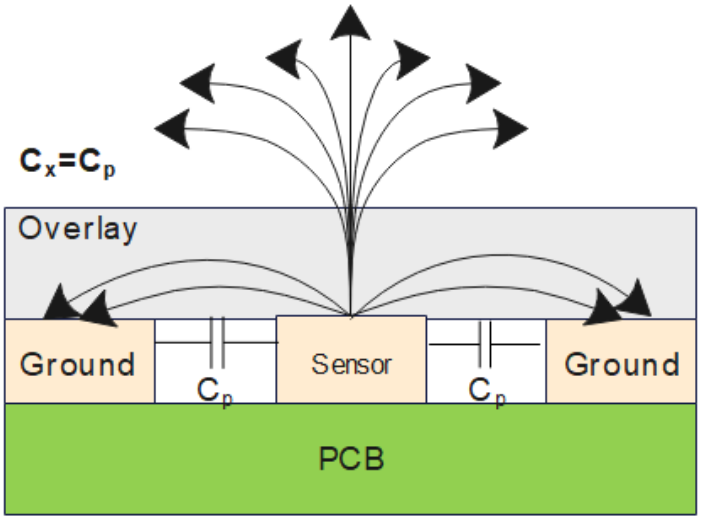


2.5接近感应

接近感应技术主要用于生物体的近距离无接触感应控制，常见的接近感应技术有红外反射、雷达探测等，可以应用于公共设备、电工开关、卫浴产品、智能家电等。

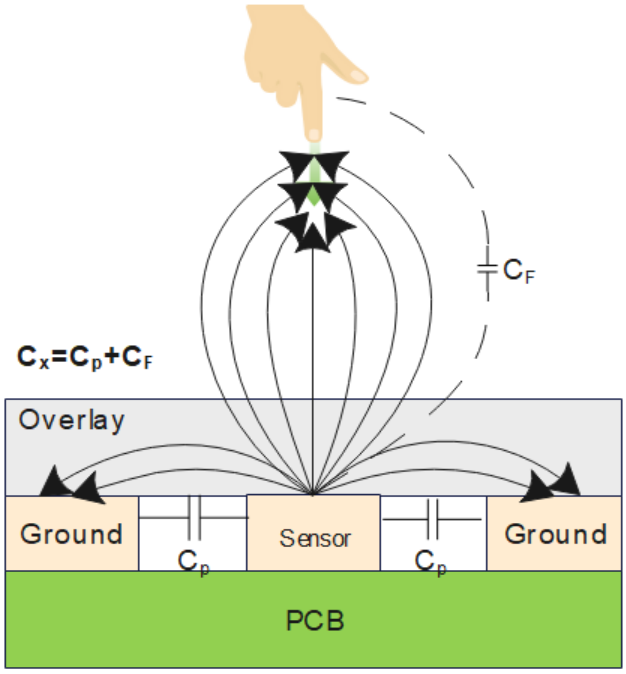
电容式接近感应技术涉及测量目标物体接近传感器时的电容变化，目标可以是人体或者任何导电物体。电容式接近感应可以使用导电物体做传感器(通常是铜和锡等)，电容式接近传感器通常位于 PCB 上。

当电容式接近传感器受到激励时，传感器周围会产生电场，少量的电场线与附近的地面耦合，而大部分电场线则投射到附近的空间。



空置状态接近感应电场分布

当物体靠近接近感应传感器时，一些电场线与目标耦合并形成电容。电容的这种变化由芯片触控电路进行测量，以检测接近的目标物体。



手接近状态接近感应电场分布

为保证远距离情况下接近感应通道的性能，应当减小接近感应通道上的寄生电容 C_P。推荐使用线宽为 2-3mm 的环形闭合走线作为接近感应 TK 通道的感应线圈，感应线圈的面积要尽量大。而具体的 PCB 设计建议可参考 5.5 接近感应感应按键设计。

3 赛元触控库介绍

3.1触控库应用类型

赛元 Touchkey MCU 提供一个可以供用户调用的库文件，以降低用户触摸按键部分的开发难度。
大体分为以下几类库体类型：

- 普通触摸按键库（T1/T2 库在 1 赛元 TOUCHKEY MCU 应用指南总体描述章节已介绍差异）
  - 弹簧触控库（简称 T1 库）
  - 隔空触控库（简称 T2 库）
- 滑轮滑条触控库
- 接近感应触控库
- 低功耗触控库（包含普通低功耗触控库，滑轮滑条低功耗触控库）

本文将介绍触摸按键库体使用以及触控调试上位机 SOC TouchKey Tool Menu 软件使用，

用户仅仅需要经过以下几个步骤，便可实现触摸按键的功能，并将赛元的触控软件库跟用户的软件完美结合，实现最终的产品功能。

3.2触控项目开发简要步骤

完整触控项目开发分以下几个步骤：

1. **安装开发工具并配置参数、导出配置参数**
   赛元提供了专门的触控调试上位机软件 SOC TouchKey Tool Menu，方便用户能通过一系列的人机交互完成调试工作，用户需要安装此软件，并配合 DPT52/SC-LINK/SC-LINK PRO 在线烧录器使用。用户可通过软件界面配置参数来找到用户 PCB 最合适的触摸按键关键参数，并将最终的相关参数导出生成头文件加入到用户工程中使用。

2. **实现赛元软件库的功能测试**
   将步骤 1 生成的配置文件加入到赛元触控软件库中，将整个库相关文件加入到用户项目工程进行编译。赛元提供简单的测试程序，可供用户完成按键部分功能的测试。

3. **完成用户程序和赛元触控软件库的融合**
   用户自行写好除触摸按键以外的其它部分软件，并将赛元的软件库嵌套进用户程序中，从而完成整个产品的整体功能。
   详细开发操作流程请至相关章节查看：4 触控开发流程

3.3赛元触控库文件介绍

赛元触控库包括以下几个文件：

3.3.1 C51 系列触控库文件

SensorMethod.h： 该文件是触控库对外的接口函数声明。用户需要在主程序引用该头文件。
SC9XF8XXX_X_X_Vx.x.x.LIB：该文件是触控库算法部分，用户需要将该文件加入工程进行编译
S_TOUCHKEYCFG.H：该文件是触控相关参数的配置文件。（用户通过 SOC TouchKey Tool Menu 软件调试后生成）
S_TouchKeyCFG.C：该文件包含触控参数头文件与触控库交互的相关接口，用户需要将文件加入工程编译。普通触控库该文件无需修改，请知悉。

3.3.2 ARM 系列触控库文件

SC32F1XXX_TK_FunctionResource.h：该文件是触控库对外的接口函数声明。用户需要在主程序引用该头文件。

SC32F1XXX_X_X_Vx.x.x.LIB(KEIL 平台) / SC32F1XXX_X_X_Vx.x.x.a(IAR 平台)：
该文件是触控库算法部分，用户需要将该文件加入工程进行编译。
S_TOUCHKEYCFG.H：该文件是触控相关参数的配置文件。（用户通过 SOC TouchKey Tool Menu 软件调试后生成）
SC32F1XXX_TK_ParameterAnalysis.c：该文件包含触控参数头文件与触控库交互的相关接口，用户需要将文件加入工程编译。普通触控库该文件无需修改，请知悉。
而滑轮滑条触控库及接近感应触控库需要进行参数配置，详细修改项请移至特殊应用指南进行查阅：
《赛元 TK 触控特殊应用说明》。

4 触控开发流程

本章节主要介绍的是如何进行完整的触控项目开发。**需要注意的是在项目开发之前，完整的触控板 PCBA 也是项目调试中不可或缺的部分，关于触控 MCU Layout 请参考 5 触摸按键 MCU PCB 设计要点。保证触控项目硬件符合要求再进行开发，将减少开发过程中所遇到的问题。**

4.1安装开发工具

4.1.1 C51 系列工具

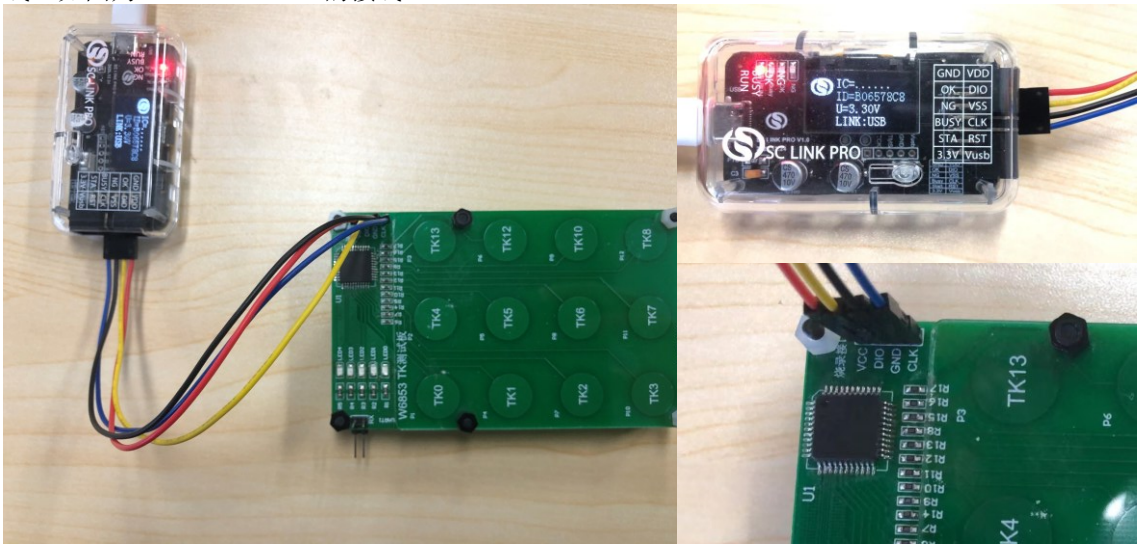
1. Setup SOC Pro51/SOC Programming Tool Enhance
安装赛元软件 SOC Pro51 Vx.xx.exe/ SOC Programming Tool Enhance (请从赛元网站找最新版本)。

2. Setup SOC TouchKey Tool Menu
安装赛元触控调试软件 SOC TouchKey Tool Menu (请从赛元网站找最新版本)。

3. 升级 SC-LINK/SC-LINK PRO 固件,更新 MCU 库
在线烧写器 SC-LINK/SC-LINK PRO 的固件和 SOC Pro51/ SOC Programming Tool Enhance 的 MCU 库文件需升级到赛元官网最新版本。

4. 安装 SOC_KEIL 插件；
请将赛元 MCU 的插件安装文件版本更新到官网最新版本。安装方法及注意事项如下：
a.安装 SOC_KEIL 插件，此插件可自动查找系统中安装的 KEIL（C51 版本）的安装目录，并将所有文件安装到 KEIL C 安装目录下 C51 目录内 SinOne_Chip/SinOne_Chip_SClinkPRO 目录。
b.SinOne_Chip /SinOne_Chip_SClinkPRO 目录内所有文件如下：
CDB：赛元 MCU 开发库文件
DEMO：赛元 MCU 示例程序
INC：赛元 MCU 头文件
PDF：赛元 SOC 烧录仿真工具 SC-LINK PRO 使用说明
SOC_Debug_Driver/SCLINK_PRO_Debug_Driver: 赛元仿真插件
c.赛元 SOC_KEIL 插件会新建一个赛元 MCU 专用列表，不会覆盖掉 KEIL C 原有的 MCU 列表。
d.如果无法安装 SOC_KEIL 插件，请检查您的 KEIL 是否是 C51 版本。

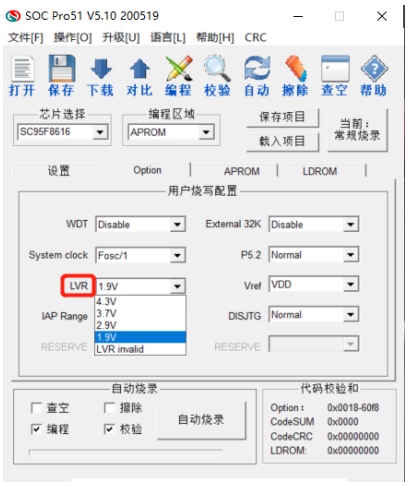
5. 硬件连接顺序：电脑 USB-->SC-LINK/SC-LINK PRO(VCC/GND/CLK/DIO)-->用户 PCB(VCC/GND/tCK/tDIO),并测试连接正常。调试过程需要用到硬件 UART 资源，请 PCB 预留接线。如图为 SC-LINK PRO 的接线。



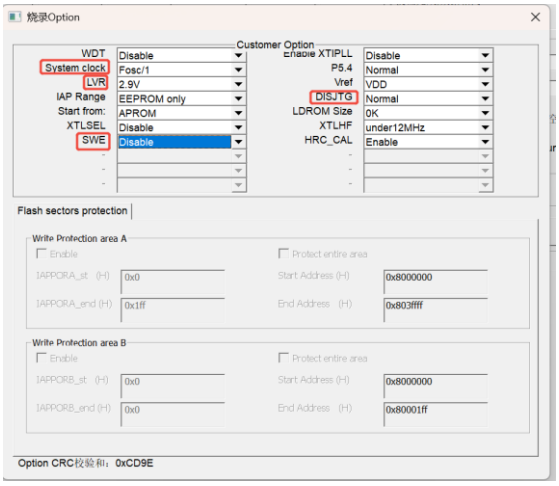
6. 烧录静态调试代码 hex 文件到用户 PCB 上的 SC9XF8XXX IC 中
打开 SOC Pro51/SOC Programming Tool Enhance 软件，选择项目使用的 MCU 型号，载入 静态调试代

码 hex 文件，点击“编程”，完成后关闭 SOC Pro51/ SOC Programming Tool Enhance 软件，重新拔插 USB 上电。**(注意:LVR 设置必须低于供电电压,如供电为 3.3V,则 Option 中 LVR 必须选择 3.3V 以下的档位，系统分频则选择 Option 中的默认分频。若有 SWE 和 JTAG 功能则设置为 Normal 模式，否则影响静态调试功能)**

见以下 SOC Pro51/SOC Programming Tool Enhance 软件所示图：



SOC Pro51 软件



SOC Programming Tool Enhance 软件(C51 操作)

高灵敏调试见以下操作步骤：4.2.1 高灵敏度调试触控参数
部分有并联通道的触控型号低功耗应用调试见以下操作步骤：4.2.2 并联通道调试触控参数

4.1.2 ARM 系列工具

1. Setup SOC Programming Tool /SOC Programming Tool Enhance

安装赛元软件 SOC Programming Tool / SOC Programming Tool Enhance (请从赛元网站找最新版本)。

2. Setup SOC TouchKey Tool Menu

安装赛元触控调试软件 SOC TouchKey Tool Menu (请从赛元网站找最新版本)。

3. 升级 SC-LINK PRO 固件,更新 MCU 库

在线烧写器 SC-LINK PRO 的固件和 SOC Programming Tool / SOC Programming Tool Enhance 的 MCU 库文件需升级到赛元官网最新版本。

4. 安装 SOC_KEIL / SOC_IAR 插件；

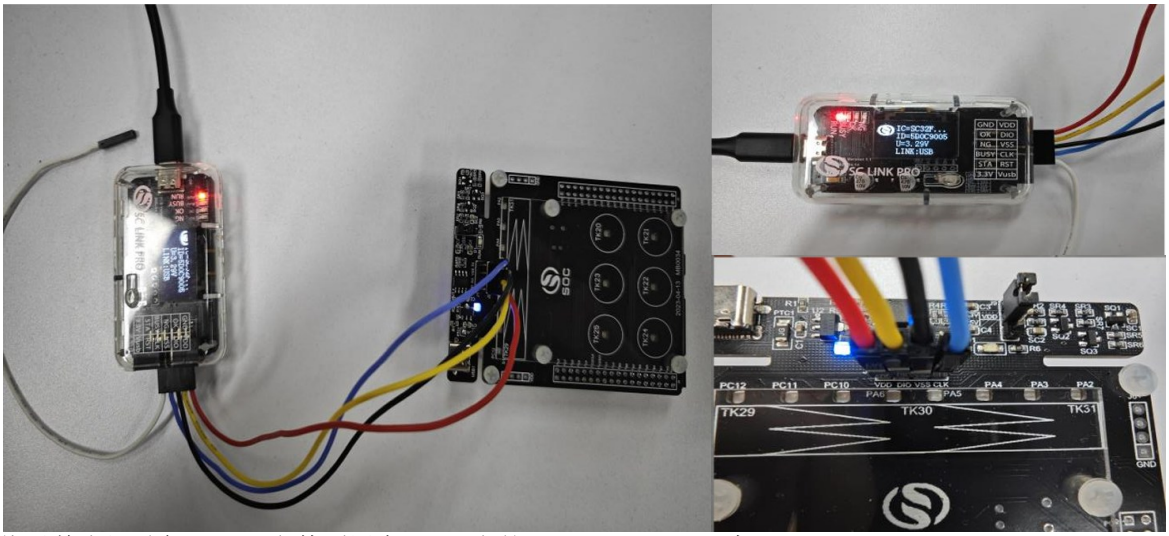
请将赛元 MCU 的插件安装文件版本更新到官网最新版本。安装方法及注意事项如下：

a.安装 SOC_KEIL / SOC_IAR 插件，此插件可自动查找系统中安装的 KEIL / IAR（ARM 版本）的安装目录，并将所有文件安装到对应软件安装目录下的 ARM 目录内。

b.赛元 SOC_KEIL / SOC_IAR 插件会新建一个赛元 MCU 专用列表，不会覆盖掉原有的 MCU 列表。

c.如果无法安装 SOC_KEIL 插件，请检查您的 KEIL 是否是 ARM 版本。

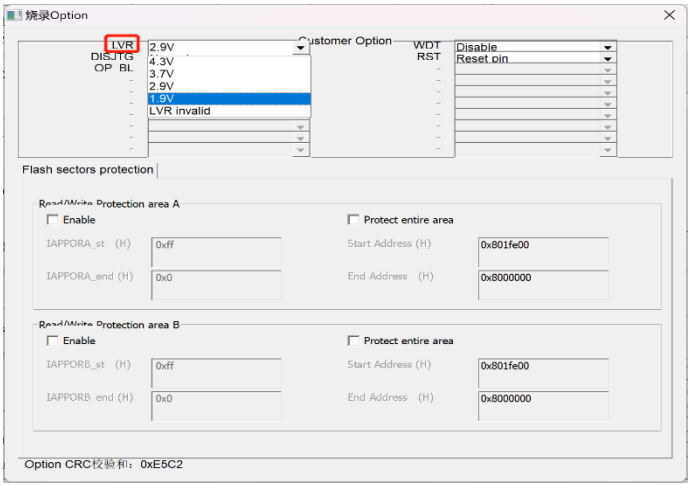
5. 硬件连接顺序：电脑 USB-->SC-LINK PRO(VCC/GND/CLK/DIO)-->用户 PCB(VCC/GND/tCK/tDIO),并测试连接正常。调试过程需要用到硬件 UART 资源，请 PCB 预留接线。如图为 SC-LINK PRO 的接线。



6. 烧录静态调试代码 hex 文件到用户 PCB 上的 SC32F1XXX IC 中

打开 SOC Programming Tool/ SOC Programming Tool Enhance 软件，选择项目使用的 MCU 型号，载入静态调试代码 hex 文件，点击“编程”，完成后关闭 SOC Programming Tool/ SOC Programming Tool Enhance 软件，重新拔插 USB 上电。(**注意:LVR 设置必须低于供电电压,如供电为 3.3V,则 Option 中 LVR 必须选择 3.3V 以下的档位，系统分频则选择 Option 中的默认分频。其中 JTAG 功能设置为 Normal 模式，否则影响静态调试功能**)

见以下 SOC Programming Tool/ SOC Programming Tool Enhance 软件所示图：



SOC Programming Tool/SOC Programming Tool Enhance 软件

高灵敏调试见以下操作步骤：4.2.1 高灵敏度调试触控参数

低功耗应用调试见以下操作步骤：4.2.2 并联通道调试触控参数

4.2调试触控参数

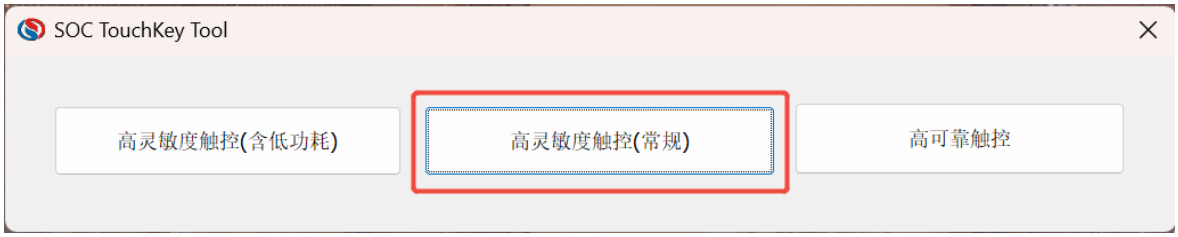
触摸静态调试主要用于触控参数的确定，在进行触控参数调试时烧入赛元提供的静态调试代码，配合赛元触控调试软件便可实现触控参数的确定。而触摸动态调试是用于在程序在 PCB 上运行时进行动态观察触控数据的，观察是否需要微调触控参数。

4.2.1 高灵敏度调试触控参数

1. **需提前烧录好对应芯片的静态调试码，具体操作请看** **4.1 安装开发工具**
2. **打开 Touch Key Tool Menu，选择高灵敏度触控**

常规：普通触摸按键调试，滑轮滑条按键调试，接近感应按键调试。

含低功耗：普通触控低功耗按键调试，滑轮滑条低功耗调试。



3. 参数配置，进入触控参数调试

1) 选择项目使用的 MCU 型号以及勾选使用的 TK 通道，如图所示：

设置应用的基本信息如下：

应用类型：选择弹簧按键/隔空按键/接近感应，根据项目需要选择(暂时矩阵按键无具体应用支持)。

按键类型：**针对弹簧按键应用需要设置。其他应用无需设置。**

选择单按键或者组合按键(双键)，根据项目需要选择。

设置应用的基本信息如下：

应用类型：选择弹簧按键/隔空按键/接近感应，根据项目需要选择(暂时矩阵按键无具体应用支持)。

弹簧按键：适用于弹簧，导电膜，TP 膜等覆盖面板紧贴类的应用场景，且可适用于邻键干扰小的场景，支持组合按键，但其局限性在于弹簧按键的感应面积相对较小，对按压位置的精准度有一定要求。

隔空按键：用于邻键干扰大的场景，不支持组合按键，需要至少 3 个以上的按键个数，感应面积大，用户无需精准按压在某个点上，只要在感应区域内操作即可，使用起来更方便。但其局限性在于对环境要求相对较高，如温度、湿度变化可能影响电容值，导致感应灵敏度不稳定。而且隔空按键需要与操作面板保持一定间隙，若生产误差或长时间使用导致间隙变化，可能会出现触摸不稳定、不灵敏甚至触发失灵的情况。

调试电压选择：与项目中赛元 MCU 芯片 VDD 供电电压有关。

5V 项目选择 5V 调试，3.3V 项目选择 3.3V 调试。

2) 配置触控算法运算的相关参数（保持默认参数不改动，以下为各个参数相关介绍）

按键确认次数：建议保持默认。该参数决定触控算法运行的出键速度，出键速度与一轮按键扫描时间有关，若扫描一轮按键需要 12MS，按键确认次数为 5 次，则按键需要的响应时间为 5*12MS=60MS。

按键最长输出：建议保持默认。该参数决定了按键持续响应的时间，单位为轮数。时间大概为 N*2ms*3000，N 为按键个数（扫描时间一般为 1ms）按键时间到达指定次数，则该按键的标志会被清除。

滤波 K 值：建议保持默认。**注：当芯片拥有内外置 CMOD 电容功能时，此设置项将显示为 CMOD 电容，默认 0：内嵌。其他调试流程和触控库调用方式，和原来保持一致。**

抗干扰设置：用于扫描时钟变频，有助于通过 EMI 测试，当项目有 EMI 测试要求，需要选择打开 1:12bit。**注意：1.用于低功耗应用时，禁止开启抗干扰设置。2. 当有多颗芯片在同一个 PCB 上，同时不同芯片的 TK 通道 PAD/弹簧或者走线距离过近（≤10mm），优先做法是将距离过近的 TK 通道修改为由同一颗芯片控制。如果不能修改走线，则需要针对相邻芯片其中一颗芯片开启抗干扰设置 1:12bit，另外一颗不开启（在进行触摸上位机静态调试操作时请留意设置）。**

参考电压：建议保持默认。

调试模式选择：建议保持默认。静态调试为确定触控参数，动态调试为应用中采集数据，这里选择静态调试，后续章节会介绍动态调试。

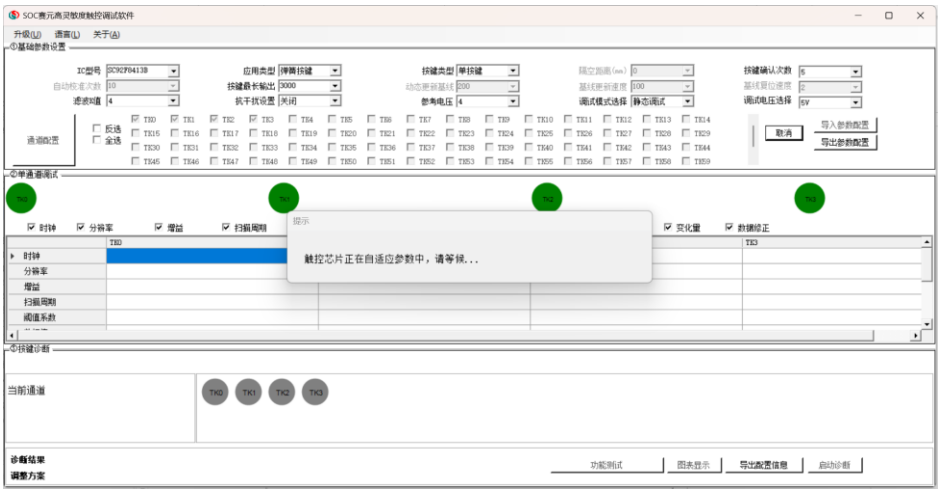
3) 请确保勾选的项目所需的完整 TK 通道口，待以上步骤完成后点击“确定”按钮，此时通道选择上锁，不能进行设置。若需要更改通道，需要点击“取消”按钮。

注意：由于调试触摸需要用到烧录口上的 UART 资源，部分型号烧录口也具有 TK 功能，因此在

进行触摸调试时无法调试这两路的参数。若用户需要用到这两个 TK 口，请联系赛元的工程师协助。

4. 触摸按键参数自适应

用户点击“确定”按键后会进入按键参数自适应阶段，此时需要等待几十秒到几分钟的时间，具体时间和按键的个数有关，直到弹出的提示窗口关闭，自适应完成。**在此过程，需要用户安装好整机，请勿对面板以及面板周围进行任何操作。**



5. 进行单通道调试

1) 在通道调试区点击对应通道绿色的按钮，进行单通道调试界面

2) 设置触控相关参数

一般情况下，按键经过自适应过程，用户无需修改以上参数，直接点击启动调试。

时钟：保持默认，不进行改动

分辨率：保持默认，不进行改动 增益：保持默认，不进行改动

扫描周期：设置范围 1-32，单位为 128us。数值越大，该键扫描时间越长，变化量越大。

阈值设置：设置范围 1-8，数值越大，灵敏度越低，反之亦然。如设置值为 5，即阈值设置为变化量的 50%，当数据变化超过阈值认为有键。建议设置为 5。

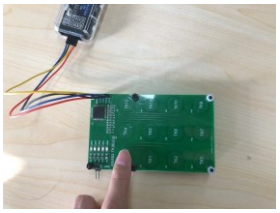
3) 点击“启动调试”按钮进行调试

调试分两个过程：无触摸过程以及触摸过程。

请按照界面的提示相应进行操作。该过程大约需要 15 秒。

不触摸过程：

触摸过程：



普通按键调试阶段将手垂直紧贴于按键感应面上方

滑轮滑条调试阶段将手垂直紧贴于锯齿中心感应面上方，如左图所示白色区域

注：软件显示的 TK 通道与 MCU 规格书一致，请根据实际 PCB 的 layout 布局，操作对应的按键，否则得到的结果将会错误！

单通道调试结束:若调试通过，则下图界面内显示绿色图标：

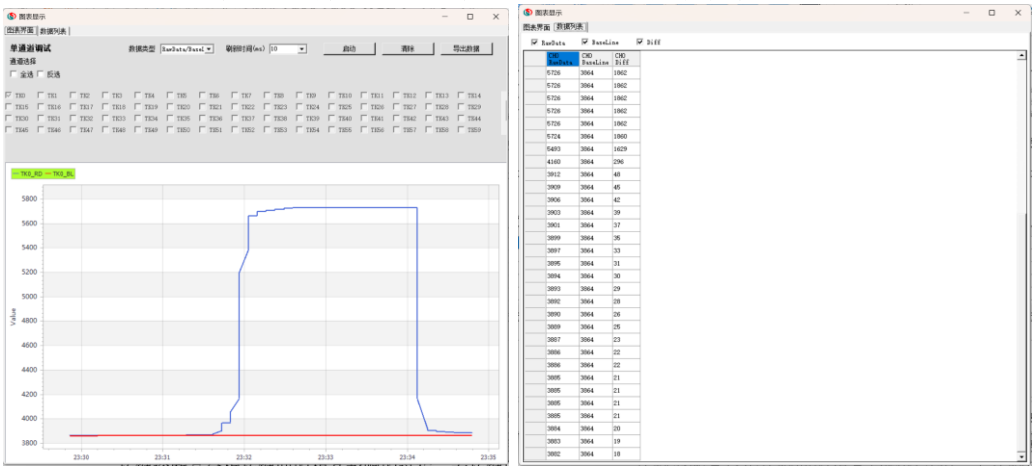
若调试不通过，则显示红色图标。

不通过的项目相应会红色字体标出。

依次调试每个按键，调至按键均通过。

注：附加观察项（调试非必要流程）

点击“图表显示”按钮，再按“启动”按钮可以实时的观察数据变化



4) 进行按键诊断（只针对普通触摸按键进行诊断，滑轮滑条不需要诊断，过程中遇到滑轮滑条按键诊断无需操作，静待切换至普通按键在进行诊断指示操作）

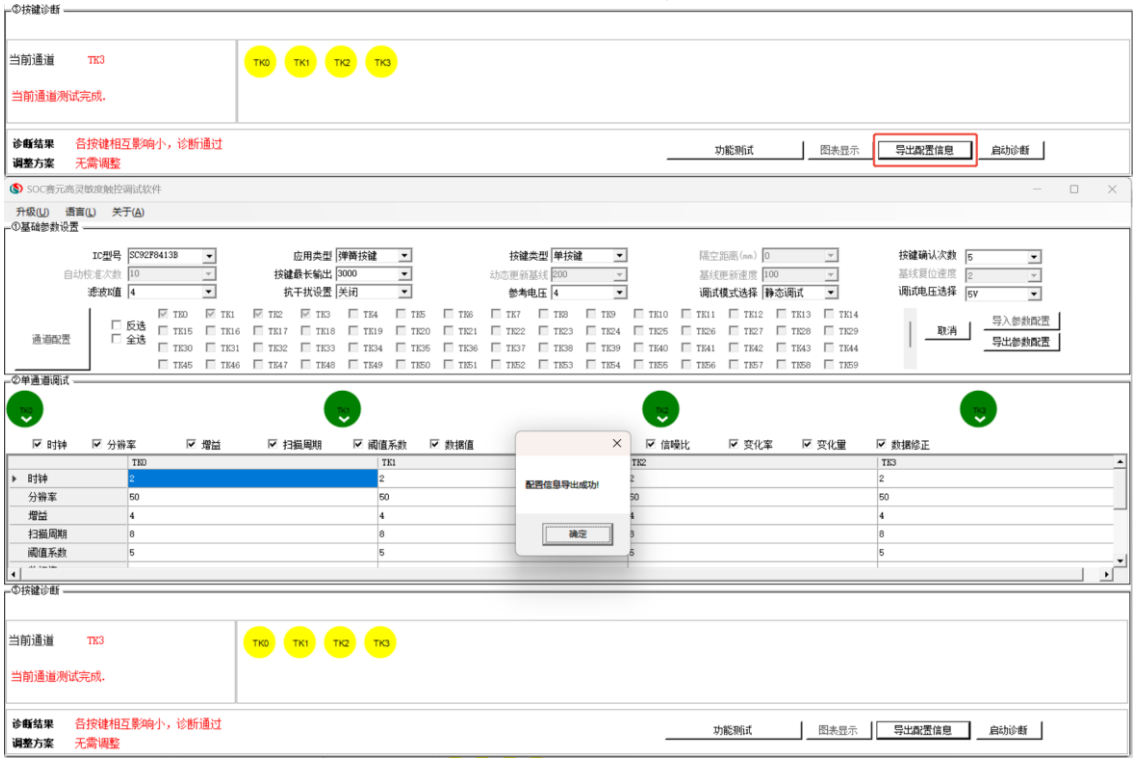
按键诊断是分析按键间的相互影响的过程。若按键间的相互影响比较大，会影响到按键的性能。

点击“启动诊断”按钮

注：软件显示的 TK 通道与 MCU 规格书一致，请根据实际 PCB 的 layout 布局，操作对应的按键，否则得到的结果将会错误！

若诊断不通过，请根据诊断结果和调整方案，调整硬件 Layout。如下图是诊断不通过的提示语：

5) **完成按键诊断并且测试通过后，点击“导出配置信息”按钮生成配置文件 S_TOUCHKEYCFG.H** 将生成的配置文件保存（**后续触控软件库移植融合步骤会用到，请保存好**）。



C51 系列：
S_TOUCHKEYCFG.H 内容如下：

S_TouchKeyCFG.h

```
//*****************************************************************************************************
//Copyright (c) 深圳市赛元微电子有限公司
//文件名称： S_TouchKeyCFG.h
//作者:
//模块功能： 触控键配置文件
//版本： V0.2
//更改记录:
//*****************************************************************************************************
#ifndef __S_TOUCHKEYCFG_H__
#define __S_TOUCHKEYCFG_H__
#define         SOCAPI_SET_TOUCHKEY_TOTAL                   4
#define         SOCAPI_SET_TOUCHKEY_CHANNEL                 0x0000000F
#define         SOCAPI_SET_TOUCHKEY_CHANNEL2                0x00000000
unsigned int  code TKCFG[17] = {0,0,0,5,10,3000,200,100,2,0,0,4,0,1,65535,65535,49};
unsigned char code TKChannelCfg[4][8]={
0x02,0x32,0x04,0x08,0x18,0x05,0x03,0xb6,
0x02,0x32,0x04,0x08,0x1b,0x05,0x03,0x6c,
0x02,0x32,0x04,0x08,0x1d,0x05,0x03,0x2b,
0x02,0x32,0x04,0x08,0x1e,0x05,0x03,0xf4,
};
#endif

```

配置文件的定义如下：

数据类型	说明	范围
SOCAPI_SET_TOUCHKEY_TOTAL	通道个数	1-31
SOCAPI_SET_TOUCHKEY_CHANNEL	通道对应数据位	0x00000001-0xffffffff
TKCFG[0]	应用类型	0-3 0 为弹簧 1 为隔空 3 为接近感应
TKCFG[1]	按键类型	0-1 0 为单键 1 为双键
TKCFG[2]		保持默认 0 不改动
TKCFG[3]	按键确认次数	3-50
TKCFG[4]		保持默认 10 不改动
TKCFG[5]	按键最长输出	0-5000
TKCFG[6]		保持默认 200 不改动
TKCFG[7]		保持默认 100 不改动
TKCFG[8]		保持默认 2 不改动
TKCFG[9]		保持默认 0 不改动
TKCFG[10]		保持默认不改动
TKCFG[11]		保持默认不改动
TKCFG[12]		保持默认不改动
TKCFG[13]		保持默认不改动
TKCFG[14]		保持默认 65535 不改动
TKCFG[15]		保持默认 65535 不改动
TKCFG[16]	噪声值	3-50
TKChannelCfg[][0]		保持默认不改动
TKChannelCfg[][1]		保持默认不改动
TKChannelCfg[][2]		保持默认不改动
TKChannelCfg[][3]	扫描周期	0x01-0x20
TKChannelCfg[][4]		保持默认不改动
TKChannelCfg[][5]		保持默认不改动
TKChannelCfg[][6]	阈值高 8 位	0x00-0xff
TKChannelCfg[][7]	阈值低 8 位	0x01-0xff

至此触摸按键的调试过程结束。

若用户调试完成后，需要微调灵敏度，可以改变 TKChannelCfg[][6] 和 TKChannelCfg[][7] 的数值， TKChannelCfg[][6]是阈值的高 8 位，TKChannelCfg[][7]是阈值的低 8 位，值越小，灵敏度越高，反之亦然。 建议多调试机台整机，以便取到折中效果的参数来去除材料对一致性的影响。

ARM 系列：

S_TOUCHKEYCFG.H 内容如下：

```
//*****************************************************************************************
//Copyright (c) 深圳市赛元微电子有限公司
//文件名称: S_TouchKeyCFG.h
//作者:
//模块功能: 触控键配置文件
//版本: V0.2
//更改记录:
//*****************************************************************************************
#ifndef __S_TOUCHKEYCFG_H__
#define __S_TOUCHKEYCFG_H__
#define             SOCAPI_SET_TOUCHKEY_TOTAL                                   6
#define                 SOCAPI_SET_TOUCHKEY_CHANNEL                                     0x03F00000
#define                 SOCAPI_SET_TOUCHKEY_CHANNEL2                        0x00000000
const unsigned int TKCFG[17] = {0,0,0,5,10,3000,200,100,2,1,0,4,0,1,65535,65535,26};
const unsigned char TKChannelCfg[6][10]={
0x03,0x02,0x46,0x04,0x08,0x03,0x1d,0x05,0x02,0x0b,
0x03,0x02,0x46,0x04,0x08,0x03,0x21,0x05,0x02,0x77,
0x03,0x02,0x46,0x04,0x08,0x03,0x21,0x05,0x02,0x64,
0x03,0x02,0x46,0x04,0x08,0x03,0x1b,0x05,0x02,0x4d,
0x03,0x02,0x46,0x04,0x08,0x03,0x21,0x05,0x02,0x2e,
0x03,0x02,0x46,0x04,0x08,0x03,0x1b,0x05,0x02,0x36,
};
#endif

```

配置文件的定义如下：

数据类型	说明	范围
SOCAPI_SET_TOUCHKEY_TOTAL	通道个数	1-64
SOCAPI_SET_TOUCHKEY_CHANNEL	通道对应数据位 TK0-TK31	0x00000001-0xffffffff
SOCAPI_SET_TOUCHKEY_CHANNEL2	通道对应数据位 TK31-TK63	0x00000001-0xffffffff
TKCFG[0]	应用类型	0-3 0 为弹簧 1 为隔空 3 为接近感应
TKCFG[1]	按键类型	0-1 0 为单键 1 为双键
TKCFG[2]		保持默认 0 不改动
TKCFG[3]	按键确认次数	3-50
TKCFG[4]		保持默认 10 不改动
TKCFG[5]	按键最长输出	0-5000
TKCFG[6]		保持默认 200 不改动
TKCFG[7]		保持默认 100 不改动
TKCFG[8]		保持默认 2 不改动
TKCFG[9]		保持默认 0 不改动
TKCFG[10]		保持默认不改动
TKCFG[11]		保持默认不改动
TKCFG[12]		保持默认不改动
TKCFG[13]		保持默认不改动
TKCFG[14]		保持默认 65535 不改动
TKCFG[15]		保持默认 65535 不改动
TKCFG[16]	噪声值	3-50
TKChannelCfg[][0]		保持默认不改动
TKChannelCfg[][1]		保持默认不改动
TKChannelCfg[][2]		保持默认不改动
TKChannelCfg[][3]		保持默认不改动
TKChannelCfg[][4]	扫描周期	0x01-0x20
TKChannelCfg[][5]		保持默认不改动
TKChannelCfg[][6]		保持默认不改动
TKChannelCfg[][7]		保持默认不改动
TKChannelCfg[][8]	阈值高 8 位	0x00-0xff
TKChannelCfg[][9]	阈值低 8 位	0x01-0xff

至此触摸按键的调试过程结束。

若用户调试完成后，需要微调灵敏度，可以改变 TKChannelCfg[][8] 和 TKChannelCfg[][9] 的数值， TKChannelCfg[][8]是阈值的高 8 位，TKChannelCfg[][9]是阈值的低 8 位，值越小，灵敏度越高，反之亦然。 建议多调试机台整机，以便取到折中效果的参数来去除材料对一致性的影响。

6. 附加功能-按键/滑轮/滑条手感模拟功能测试：

主要功能：在“启动诊断”及 “导出配置信息”操作后，可直接在上位机测试按键参数手感，观察参数是否适应整机。

模拟功能测试按键类型包含：按键，滑条，滑轮。

以下为测试流程：

1) 在“启动诊断”及 “导出配置信息”操作后，选择某一种按键类型：以下以滑轮为例。

2) 按键/滑条/滑轮测试：选择滑轮测试

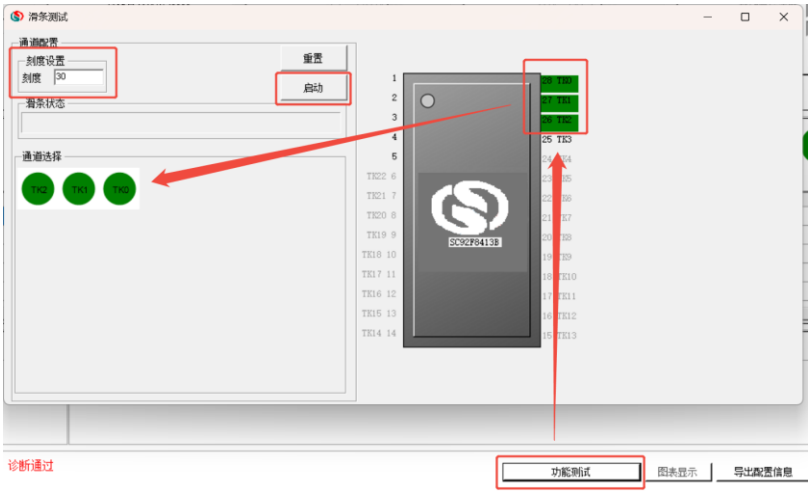
① 设置刻度值：对应滑轮最大刻度

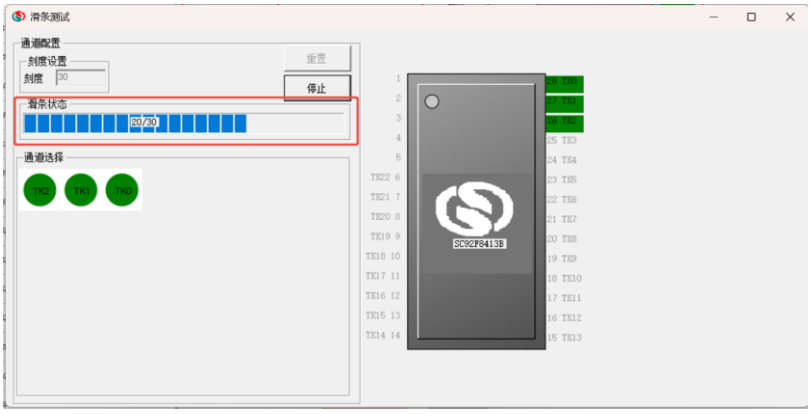
② 设置滑轮通道顺序：注意选择的顺序，需按着硬件上滑轮 TK 通道顺序设置，例如以下图片滑轮按着 TK2->TK1->TK0 的顺序排布

③ 点击启动，滑动硬件上滑轮按键，可在上位机上观察到滑轮效果和手感。

④ 测试完点击停止按钮即可，随后关闭该界面。

补充：重置按钮是在启动前，需重新设置，可点击重置按钮。



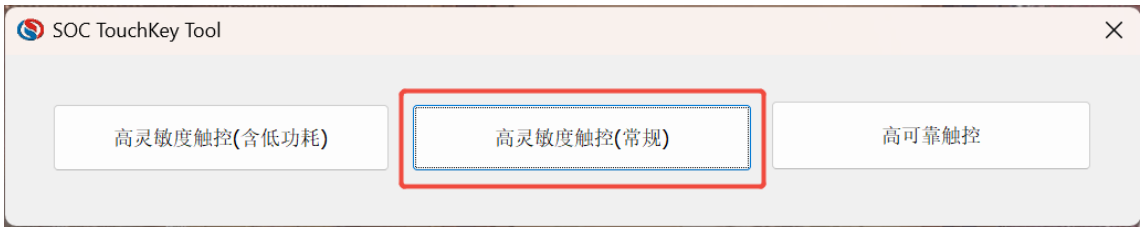


注意点：

- “滑条/滑轮测试”和“按键测试”设置的按键不能重复，且不能共用。
- 先设置完滑条/滑轮的按键测试后，再次选择该 TK 通道“按键测试”，无法测试单个按键功能。
- 要体验滑条/滑轮/按键测试功能请更新最新的高灵敏静态调试文件。
- 按键测试项，不需要设置刻度参数，直接点击启动即可，可在点击对应 TK 按键在上位机观察按键情况。

4.2.2 并联通道调试触控参数

1. 需提前烧录好对应芯片的静态调试码，具体操作请看 4.1 安装开发工具
2. 打开 Touch Key Tool Menu，选择高灵敏度触控，选择高灵敏度触控。



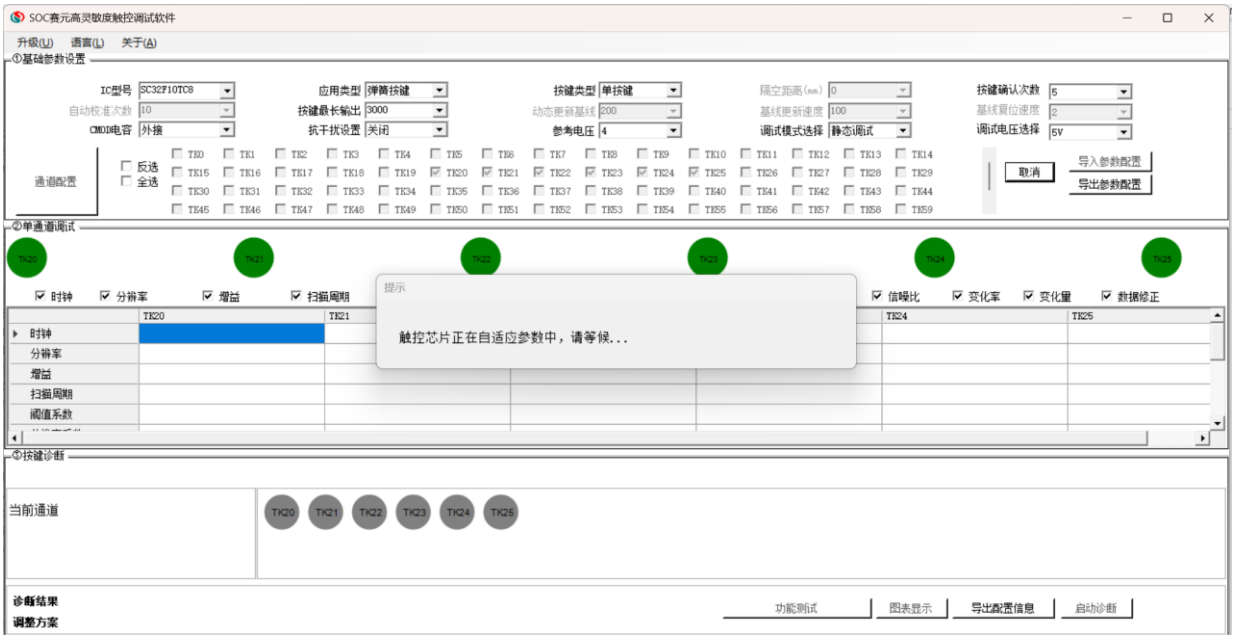
3. 参数配置，进入触控参数调试

选择项目使用的 MCU 型号以及勾选使用的 TK 通道，再点击通道配置，如图所示：

点击通道配置，选择配置并联通道，点击加号添加并联通道组，选择相应并联通道（当前仅支持一组并联通道）。

4. 触摸按键参数自适应

用户点击“确定”按键后会进入按键参数自适应阶段，此时需要等待几十秒到几分钟的时间，具体时间和按键的个数有关，直到弹出的提示窗口关闭，自适应完成。**在此过程，需要用户安装好整机，请勿对面板以及面板周围进行任何操作。**



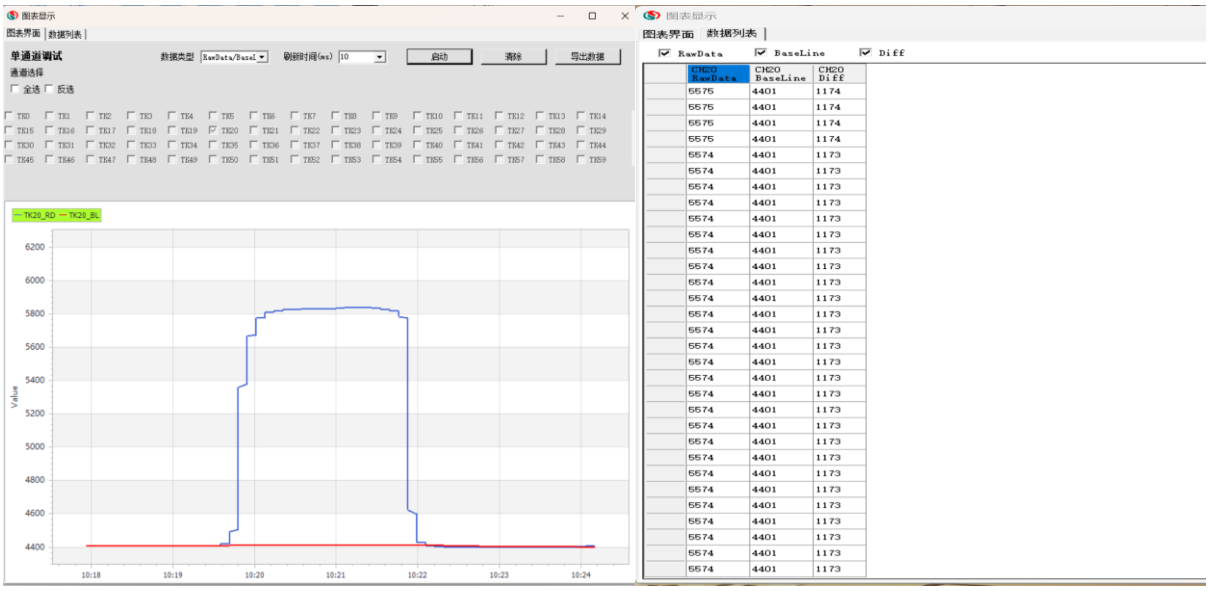
5. 并联通道调试

点击启动调试，按照提示进行调试，等待调试完成，图标显示绿色代表成功，红色代表调试失败，不通过的项目相应会红色字体标出。点击并联通道 Group 0 查看调试参数。（该调试会同时调试单按键，以及并联按键）。

调试通过：

调试不通过：

注：附加观察项（调试非必要流程）
点击“图表显示”按钮，再按“启动”按钮可以实时的观察并联通道的数据变化。



6. S_TOUCHKEYCFG.H 内容解析。

C51 系列：

S_TOUCHKEYCFG.H 内容如下：**红色框内数据和单通道数据一致。**

```
//*****************************************************************************************************
//Copyright (c) 深圳市赛元微电子有限公司
//文件名称: S_TouchKeyCFG.h
//作者:
//模块功能: 触控键配置文件
//版本: V0.2
//更改记录:
//*****************************************************************************************************
#ifndef __S_TOUCHKEYCFG_H__
#define __S_TOUCHKEYCFG_H__
#define         SOCAPI_SET_TOUCHKEY_TOTAL                   13
#define         SOCAPI_SET_TOUCHKEY_CHANNEL                 0x024007FF
#define         SOCAPI_SET_TOUCHKEY_CHANNEL2                0x00000000
unsigned int  code TKCFG[17] = {1,0,2,3,10,3000,200,100,2,0,0,3,0,0,65535,65535,20};
unsigned char code TKChannelCfg[SOCAPI_SET_TOUCHKEY_TOTAL][8]={
0x04,0x19,0x03,0x08,0x0d,0x05,0x00,0xc8,
0x04,0x19,0x03,0x08,0x0f,0x05,0x00,0xc8,
0x04,0x19,0x03,0x08,0x11,0x05,0x00,0xd2,
0x04,0x19,0x03,0x08,0x12,0x05,0x00,0xdc,
0x04,0x19,0x03,0x08,0x0d,0x05,0x00,0xa0,
0x04,0x19,0x03,0x08,0x0f,0x05,0x00,0xa0,
0x04,0x19,0x03,0x08,0x11,0x05,0x00,0xaa,
0x04,0x19,0x03,0x08,0x13,0x05,0x00,0xbe,
0x04,0x19,0x03,0x08,0x11,0x05,0x00,0xa0,
0x04,0x19,0x03,0x08,0x0f,0x05,0x00,0x8c,
0x04,0x19,0x03,0x08,0x0d,0x05,0x00,0x87,
0x04,0x19,0x03,0x08,0x10,0x05,0x00,0xc9,
0x04,0x19,0x03,0x08,0x17,0x05,0x01,0x2c,//
};
#define         SOCAPI_COMBINE_TOUCHKEY_TOTAL                1
unsigned char code TKCombineChannels[1][16]={
0xff,0x07,0x40,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
unsigned char code TKCombineChannelCfg[1][8]={
0x07,0x36,0x03,0x12,0x6e,0x05,0x00,0xB4,

};
#endif
```

数据类型	说明	范围
SOCAPI_COMBINE_TOUCHKEY_TOTA	并联通道个数	1
TKCombineChannels [][]	并联通道使用到的 TK 通道	
TKCombineChannelCfg [][0]		保持默认不改动
TKCombineChannelCfg [][1]		保持默认不改动
TKCombineChannelCfg [][2]		保持默认不改动
TKCombineChannelCfg [][3]	并联通道扫描周期	0x01-0x20
TKCombineChannelCfg [][4]		保持默认不改动
TKCombineChannelCfg [][5]		保持默认不改动
TKCombineChannelCfg [][6]	并联通道阈值高 8 位	0x00-0xff
TKCombineChannelCfg [][7]	并联通道阈值低 8 位	0x01-0xff

ARM 系列：

S_TOUCHKEYCFG.H 内容如下：**红色框内数据和单通道数据一致。**

```
//*****************************************************************************************************
//Copyright (c) 深圳市赛元微电子有限公司
//文件名称:  S_TouchKeyCFG.h
//作者:
//模块功能:  触控键配置文件
//版本:  V0.2
//更改记录:
//*****************************************************************************************************
#ifndef __S_TOUCHKEYCFG_H__
#define __S_TOUCHKEYCFG_H__
#define         SOCAPI_SET_TOUCHKEY_TOTAL                   4
#define         SOCAPI_SET_TOUCHKEY_CHANNEL                 0x000F0000
#define         SOCAPI_SET_TOUCHKEY_CHANNEL2                0x00000000
const unsigned int TKCFG[17] = {0,0,0,5,10,3000,200,100,2,1,0,4,0,0,65535,65535,68};
const unsigned char TKChannelCfg[4][10]={
0x03,0x02,0x28,0x04,0x08,0x03,0x20,0x05,0x02,0xc0,
0x03,0x02,0x28,0x04,0x08,0x03,0x1e,0x05,0x03,0x54,
0x03,0x02,0x28,0x04,0x08,0x03,0x18,0x05,0x02,0x96,
0x03,0x02,0x28,0x04,0x08,0x03,0x1a,0x05,0x02,0xad,
};
#define         SOCAPI_COMBINE_TOUCHKEY_TOTAL               1
const unsigned char TKCombineChannels[1][16]={
0x00,0x00,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
};
const unsigned char TKCombineChannelCfg[1][10]={
0x05,0x02,0x19,0x04,0x08,0x03,0x21,0x05,0x02,0xff,
};
#endif
```

数据类型	说明	范围
SOCAPI_COMBINE_TOUCHKEY_TOTA	并联通道个数	1
TKCombineChannels [][]	并联通道使用到的 TK 通道	
TKCombineChannelCfg [][0]		保持默认不改动
TKCombineChannelCfg [][1]		保持默认不改动
TKCombineChannelCfg [][2]		保持默认不改动
TKCombineChannelCfg [][3]		保持默认不改动
TKCombineChannelCfg [][4]	并联通道扫描周期	0x01-0x20
TKCombineChannelCfg [][5]		保持默认不改动
TKCombineChannelCfg [][6]		保持默认不改动
TKCombineChannelCfg [][7]		保持默认不改动
TKCombineChannelCfg [][8]	并联通道阈值高 8 位	0x00-0xff
TKCombineChannelCfg [][9]	并联通道阈值低 8 位	0x01-0xff

4.3实现赛元软件库的功能测试

4.3.1 C51 高灵敏触控软件库移植

1. 库文件介绍
   1) 弹簧库文件（简称 T1 库，SC9XF8XXX_HighSensitive_Lib_T1_Vx.x.x.LIB）
   2) 隔空库文件（简称 T2 库，SC9XF8XXX_HighSensitive_Lib_T2_Vx.x.x.LIB）

   以下是 T1/T2 库文件简单介绍：（库体含有 4 个文件）

文件	用途	说明
SC9XF8XXX_HighSensitive_Lib	**库文件，实现触摸按键检测算法**	
SensorMethod.h	**头文件，提供接口函数供用户调用**	**声明的函数可供外部调用**
S_TouchKeyCFG.C	**C 文件， 实现触控参数与库交互**	
S_TOUCHKEYCFG.H	**头文件，提供宏供用户修改参数**	

2. Lib API 接口函数的调用说明

函数	用途	说明

TouchKeyInit(void)	触摸按键初始化	**1 用户在上电复位后调用一次； 2 本函数通过 S_TOUCHKEYCFG.H 参数配置用户选定的按键通道、按键参数并初始化 Baseline 基线； 3 执行本函数用时约：200~500mS 与按键个数，按键扫描时间，自动校准次数，触控库类型相关； 95F 系列 T1 按键每多 N 个，大约时间 32M 主频：45 uS *N 按键； 16M 主频：46 uS *N 按键； 95F 系列 T2 按键每多 N 个，大约时间 32M 主频：45 uS *N 按键； 16M 主频：46 uS *N 按键； 92F 系列 T1 按键每多 N 个，大约时间 24M 主频：48 uS *N 按键； 12M 主频：52 uS *N 按键； 92F 系列 T2 按键每多 N 个，大约时间 24M 主频：52 uS *N 按键； 12M 主频：56 uS *N 按键；**
TouchKeyRestart(void)	使能一轮触摸按键扫描	**1 用户主程序控制何时启动按键扫描； 2 启动按键扫描后，在一轮触摸按键扫描完成之前，不能对触摸按键通道进行操作：如操作触摸按键通道的 IO，否则触摸按键功能将无法实现。**
Unsigned long int TouchKeyScan(void)	触摸按键算法处理	**1 用户需要在触摸按键一轮扫描完成后调用； 2 如用户未调用该函数之前，一定不能重新调用 TouchKeyRestart()，否则上一轮数据将被当前数据覆盖； 3 执行该函数用时约为：(95F 系列 T1 按键每多 N 个，大约时间 32M 主频：57 uS *N 按键； 16M 主频：115 uS *N 按键； 95F 系列 T2 按键每多 N 个，大约时间 32M 主频：57 uS *N 按键； 16M 主频：115 uS *N 按键； 92F 系列 T1 按键每多 N 个，大约时间 24M 主频：193 uS *N 按键； 12M 主频： 385 uS *N 按键； 92F 系列 T2 按键每多 N 个，大约时间 24M 主频：179 uS *N 按键； 12M 主频： 365 uS *N 按键；)；**

3. 全局变量 SOCAPI_ToucKeyStatus 的说明

1) 全局变量在 S_TouchKeyCFG.c 头文件中声明

① Unsigned char xdata SOCAPI_ToucKeyStatus;

② SOCAPI_ToucKeyStatus Bit7 为 1 时 表示当前一轮扫描按键完成；

2) 该变量在用户主程序中调用

if(SOCAPI_ToucKeyStatus&0x80)时，调用 TouchKeyScan(void) 进行算法数据处理，给出键值；

3) 使能触摸按键扫描之前，一定要清掉标志。

清除一轮扫描标志 SOCAPI_ToucKeyStatus &=0x7f;

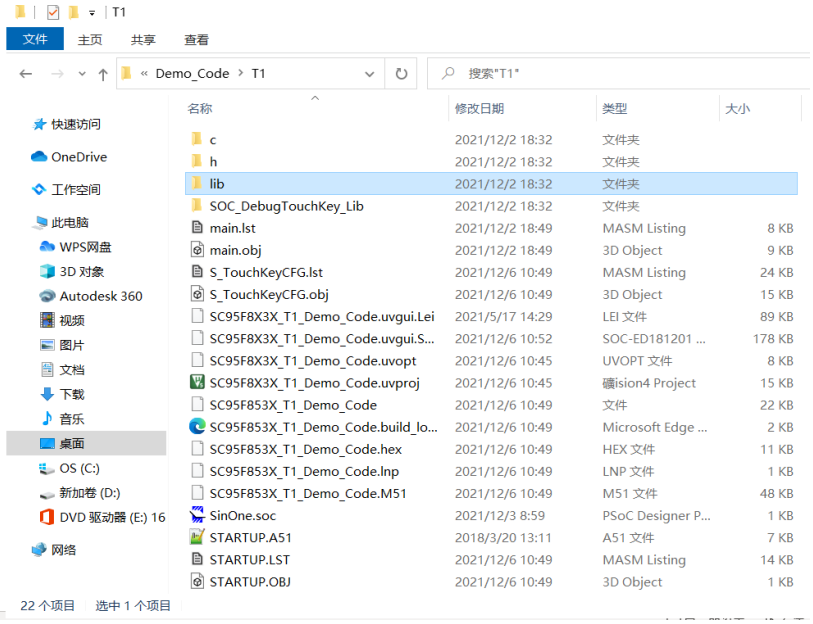
4. LIB API 接口函数的返回值说明

1) TouchKeyScan(void)函数返回值：
返回值对应 bit 为 1 即该通道有按键，0 为无按键。
若使能双键，且有两键触发，则会有两个 bit 位置起。

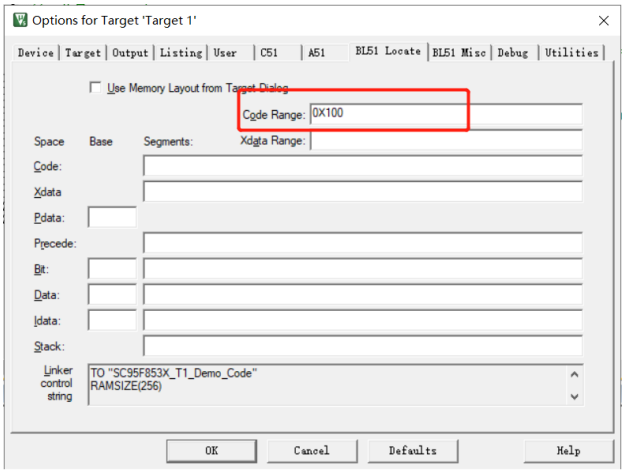
数据位		Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24
含义		TK30	TK29	TK28	TK27	TK26	TK25	TK24
	触摸按键状态（1：有效；0：无效）							
数据位	Bit23	Bit22	Bit21	Bit20	Bit19	Bit18	Bit17	Bit16
含义	TK23	TK22	TK21	TK20	TK19	TK18	TK17	TK16
	触摸按键状态（1：有效；0：无效）							
数据位	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
含义	TK15	TK14	TK13	TK12	TK11	TK10	TK9	TK8
	触摸按键状态（1：有效；0：无效）							
数据位	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
含义	TK7	TK6	TK5	TK4	TK3	TK2	TK1	TK0
	触摸按键状态（1：有效；0：无效）							

注：函数的返回类型是 unsigned long int ；TKn 为触控通道，具体请参照对应规格书。

5. 打开工程项目文件复制“lib”文件夹至工程文件夹内



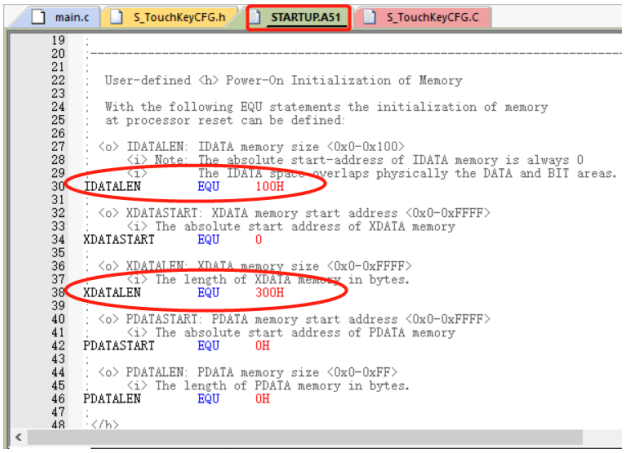
6. 在 Keil 中打开工程项目文件；注意设定（Code range）、设定（XDATALEN EQU xxxH）



设定的目的，参见赛元 MCU 应用注意事项 Vx.xx.PDF 文件。

（部分芯片不用设置该项，请仔细阅读）

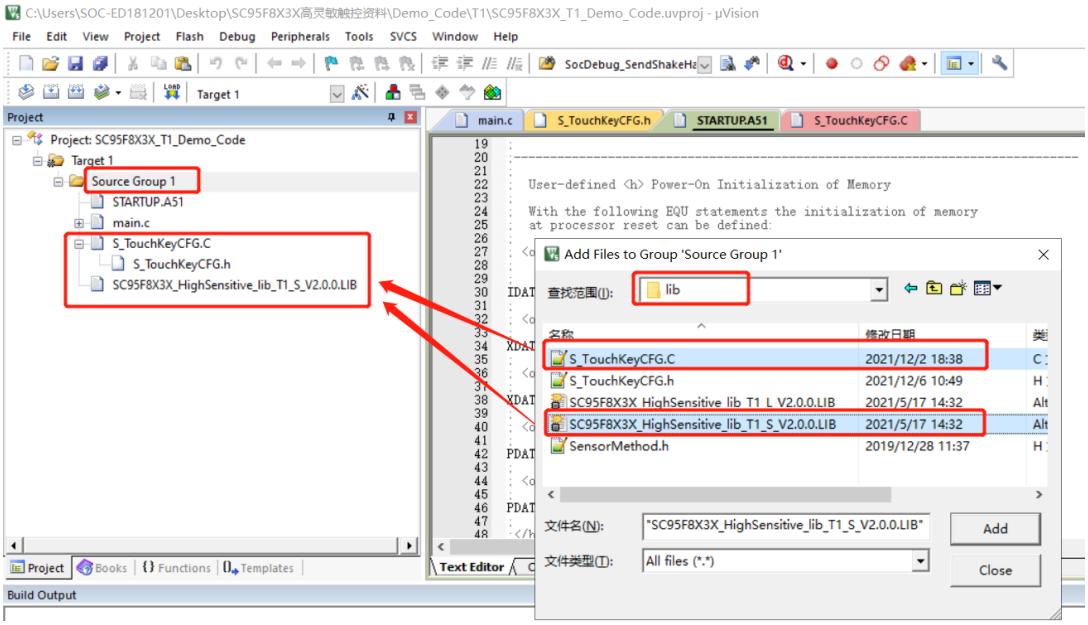
7. 设定 XDATALEN



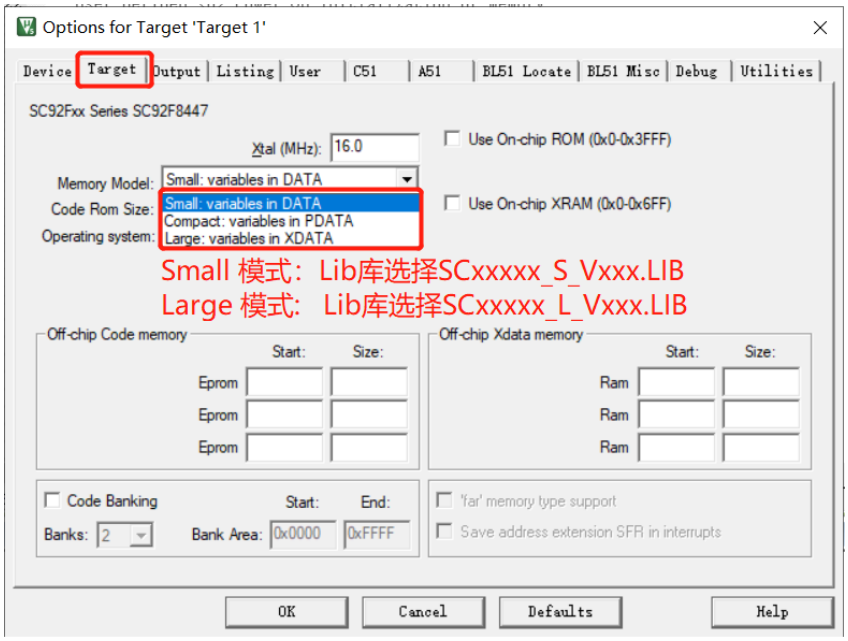
注意：用于 STARTUP.A51 中，清掉外部 XData，具体型号的 XData 大小见规格书。

8. 在项目工程中添加库文件 LIB 以及 S_TouchKeyCFG.C 文件

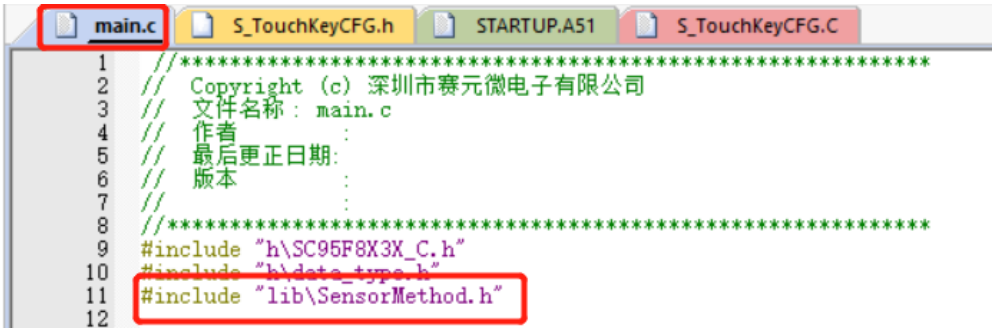
1) 从赛元提供的库体资料 Lib 文件夹内，添加库文件 LIB 以及 S_TouchKeyCFG.C 至工程内。下图以 T1 库体为例：（T2 库操作一致）



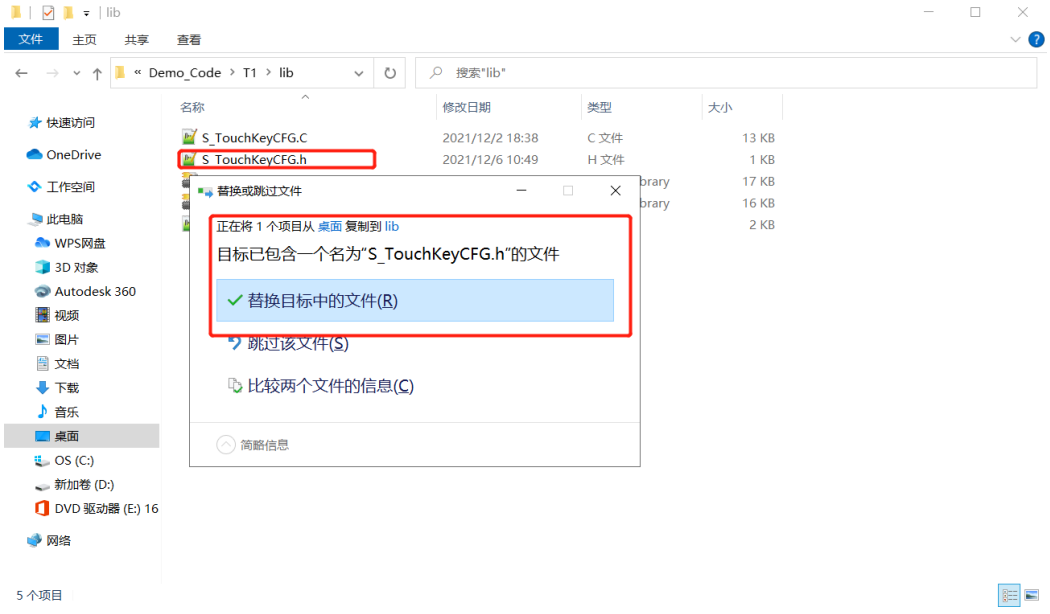
注意：库体选择 L 或者 S（大端编译或小端编译）请仔细区分，如下图所示：



9. 在主程序文件中添加头文件引用



10. 将 TK 触控调试上位机生成的配置文件 S_TOUCHKEYCFG.H 替换到 LIB 文件夹内



到此，完整的赛元高灵敏触控软件库已添加至项目工程内。

注意：应用程序中需要将 TK 对应的 IO 口设置为强推挽输出。

4.3.2 C51 低功耗触控软件库移植

1. 库文件介绍
   1) 弹簧低功耗库文件（SC9XF8XXX_HighSensitive_Lib_T1LowPower_Vx.x.x.LIB）
   2) 隔空低功耗库文件（SC9XF8XXX_HighSensitive_Lib_T2LowPower_Vx.x.x.LIB）

2. Lib API 接口函数的调用说明

接口名	接口定义	调用说明
GetLowPowerScanFlag	获取低功耗模式标志：**0** 为正常运行模式，**1** 为低功耗模式	在主循环内区分两种模式，执行不同的程序分支
TouchKey_IntoLowPowerMode	进入低功耗模式	调用该函数进入低功耗模式。用户根据实际应用决定调用，例如，一段时间未有按键按下进入低功耗，调用该函数。但需注意，进入低功耗要在无按键的情况下进入(即返回的键值为 **0**)
Customer_IntoLowPowerMode_Init	用户进入低功耗模式前的初始化	该函数用户需要自行添加在主程序中，并对其进行实现，无需调用. 如果程序中有使用中断资源，也请在此函数中关闭对应中断使能，以防应用程序逻辑上不严谨导致功能异常。
LowPower_Touchkey_Scan	低功耗 **TK** 扫描处理	在低功耗模式下调用
TouchKey_QuitLowPowerMode	退出低功耗模式	调用该函数系统进入正常模式。当 **TK** 按键被唤醒，自动被调用，退出低功耗。用户也可以根据外部信号退出低功耗
Customer_QuitLowPowerMode_Init	用户退出低功耗模式前的设置	该函数用户需要自行添加在主程序中，并对其进行实现，无需调用。 同时也需在此函数中使能之前所关闭的对应中断资源。
Customer_BTM_Dispose	用户 **BTM** 函数处理	该函数用户需要自行添加在主程序中，并对其进行实现，无需调用。由于 **TK** 低功耗的实现占用了 **BTM** 资源，这里开放了用户接口，用户在此函数内添加 **BTM** 唤醒的执行内容，例如读取外部的 **IO** 状态

3. 参数说明

打开 S_TouchKeyCFG.c 文件，将低功耗功能的宏开关打开（当用户无须低功耗功能可以将其注释，以达到节省代码空间），设置低功耗唤醒的个数，低功耗下要扫描的按键，以及 BTM 定时唤醒的时间

参数名	参数定义	设置说明
WakeUpKeyNum	低功耗模式下扫描的按键个数	不可以超过正常扫描的按键个数，按键个数越多，功耗越大
WakeUpKeyChannel	低功耗下扫描的按键通道	只能设置正常扫描下已开启的按键，将对应的按键 **BIT** 位置起，置起的位数必须与设置的按键个数相同
TK_SeepTimervSetting	低功耗下按键间的扫描	**BTM_TIMEBASE_15600US** **BTM_TIMEBASE_31300US** **BTM_TIMEBASE_62500US** **BTM_TIMEBASE_125MS** **BTM_TIMEBASE_250MS** **BTM_TIMEBASE_500MS**

	周期	**BTM_TIMEBASE_1S** **BTM_TIMEBASE_2S** **BTM_TIMEBASE_4S** **可设置以上的选择，设置的时间越长功耗越低，按键响应速度越慢。用户可根据实际需要进行设置。**
TK_WakeUpConfirmTouchCnt	**低功耗下按键的确认次数**	**可以设置 10-20**

4. 将相应型号的低功耗触控库移植到工程与 C51 高灵敏库触控软件库移植一致，可参考 4.3.1
到此，完整的赛元低功耗触控软件库已添加至项目工程内。
注意：应用程序中需要将 TK 对应的 IO 口设置为强推挽输出。

4.3.3 C51 滑轮滑条触控软件库移植

1. 库文件介绍
   1) 只含滑轮/滑条库文件（SC9XF8XXX_HighSensitive_Lib_Slide_L_Vx.x.x.LIB）
   2) 弹簧+滑轮/滑条库文件（SC9XF8XXX_HighSensitive_Lib_T1Slide_L _Vx.x.x.LIB）
   3) 隔空+滑轮/滑条库文件（SC9XF8XXX_HighSensitive_Lib_T2Slide_L _Vx.x.x.LIB）
   4) 低功耗+只含滑轮/滑条库文件（SC9XF8XXX_HighSensitive_Lib_SlideLP_L_Vx.x.x.LIB）
   5) 弹簧低功耗+滑轮/滑条库文件（SC9XF8XXX_HighSensitive_Lib_T1SlideLP_L _Vx.x.x.LIB）
   6) 隔空低功耗+滑轮/滑条库文件（SC9XF8XXX_HighSensitive_Lib_T2SlideLP_L _Vx.x.x.LIB）

2. 设置滑条/滑轮参数
   1) 打开 S_TouchKeyCFG.c 文件，在滑动模块设置项进行参数设置，设置 USING_TKSlideModule_Number(滑动模块数量，所使用的滑条和滑轮的总数)使用 TK 的个数，滑动挡位，TK 扫描的编号等。

   2) USING_TKSlideModule_Number 参数说明：(**92 型号建议最多使用总数 2 条滑动模块，95 型号最多 4 条，倘若使用过多会导致扫描处理时间过长，带来手感上的不顺畅，需要酌情考虑**)
该参数为所使用的滑动模块的数量即滑轮加滑条的数量，例如要使用一个滑条和一个滑轮那么 USING_TKSlideModule_Number 的值应该为 2。
   3) TKSlideModulePCBArray 参数说明：

该数组用来存放滑动模块的控制参数，每一个滑动模块对应一个 TKSlideModulePCB

4) TKSlideModulePCB 结构与参数说明：

① TKSlideModulePCB 结构：

```
typedef struct
{
 TKSlideModuleType     TypeFlag;
 unsigned        char  TKChannel[16];
 unsigned        char  UsingTKChannelNumber;
 unsigned        int   SideLevel;
 int                   SUBData;
 int                   LIBData;
 unsigned        char  TKOrderChannel[16];
 unsigned        char  MAXUpdateCount;
 unsigned        char  LastOutValue;
 unsigned        int   OutValue;
 unsigned        int   UpdateBaseLineNumber;
 unsigned        int   CouplingValue;
 unsigned        int   DebugCouplingValue;
 unsigned        int   TriggerFlagCount;
}TKSlideModulePCB;
```

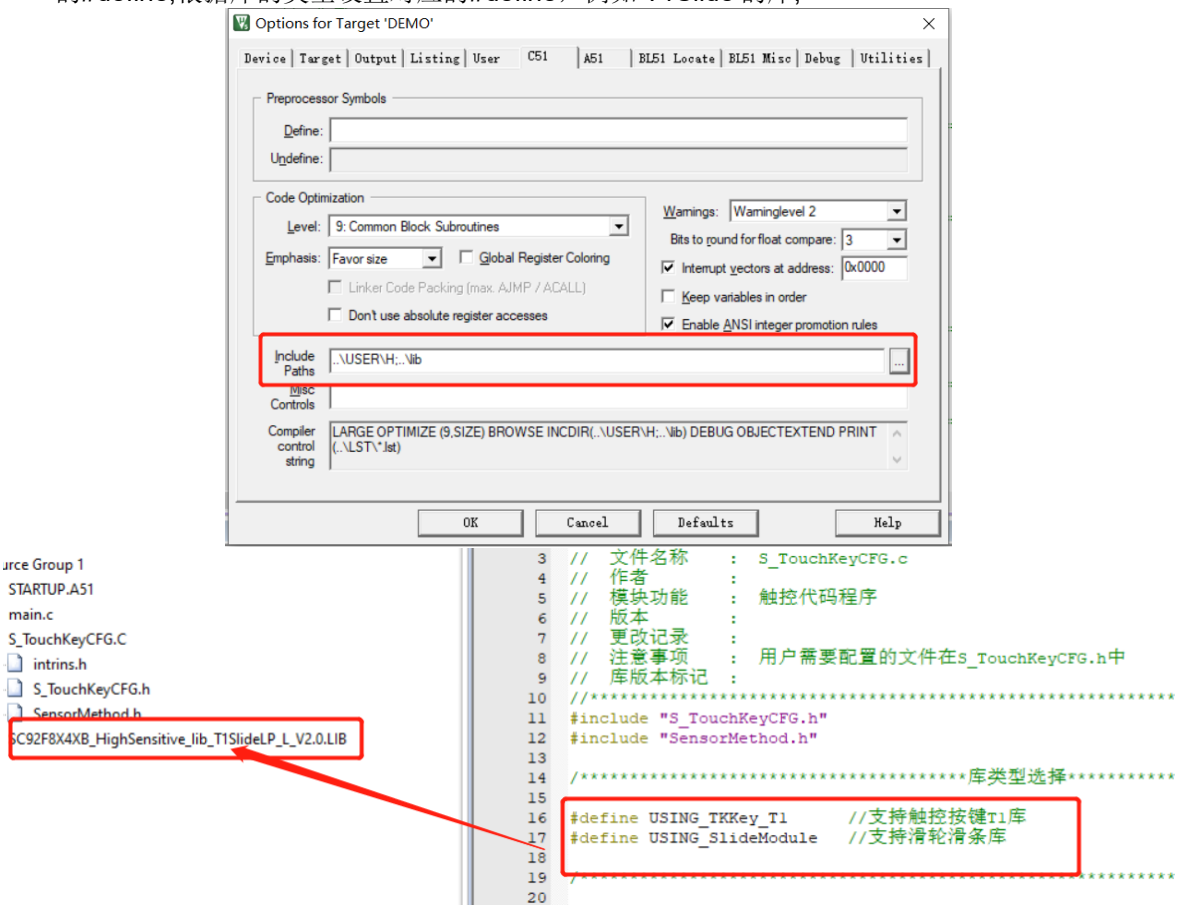
② TKSlideModulePCB 参数意义：

参数名	参数定义	设置说明
TypeFlag	滑动模块类型	**(需修改) 滑动块类型,bar:为滑条，Circle:为滑轮**
TKChannel[16]	滑动模块 **TK** 通道的排布顺序	**(需修改)按照滑动顺序，依次将使用到的滑条按键通道存入数组，依次滑动滑动值逐渐变大**
UsingTKChannelNumber	滑动模块使用的 **TK** 通道数量	**(需修改) 当前滑动模块使用的 TK 通道数量**
SideLevel	滑条档位最大级数	**(需修改)需要设置的档位值**
SUBData	变化值下限	**(可修改)Differ 值< SUBData 表示无键（修改灵敏度），增大该值可以减少干扰。建议默认 20.**
LIBData	变化值上限	**(可修改)Differ 值> LIBData 表示有键（修改灵敏度），该值过小容易触发。建议设置为变化量四分之一到三分之一**
TKOrderChannel[16]	存放内部 **TK** 通道排序的 **index**	**(无需修改)**
MAXUpdateCount	更新基线时机	**(无需修改)扫描多少轮无滑动触发即更新基线**
LastOutValue	上一次输出的位置值	**(无需修改)记录上一次输出的值**
OutValue	输出的位置值	**(无需修改)，输出滑动值，范围 0~SideLevel，0 表示滑动模块为被按下**
UpdateBaseLineNumber	基线更新计数	**(无需修改)若无按下，每扫描一次 UpdateBaseLineNumber 自增一，按下则置 0**
CouplingValue	耦合参数	**((无需修改)，可取 150 作为默认值，若实际使用无异样可不修改)耦合参数范围(100~200)，参数过小会出现出值缺失,例如滑动出值 OutValue 为**

		1->2->3->5->6->7,参数过大会出现出值重叠,例如滑动出值 OutValue 为 1->2->3->4->3->4->5->6->7
DebugCouplingValue	**耦合参数调试输出**	**(无需修改) 耦合参数 DeBug 值,用于调试。调试时输出该值，滑动滑动模块取输出的最小值，根据最小值调节 CouplingValue 耦合参数（可将 DebugCouplingValue 最小值填入 CouplingValue 作为耦合参数）**
TriggerFlagCount	**滑动模块触发计数标志**	**(无需修改)滑动模块触发计数标志**

3. 将相应型号的滑轮滑条触控库移植到工程与 C51 高灵敏库触控软件库移植一致，可参考 4.3.1。
需要在 C51 高灵敏库触控软件库移植的基础加上以下步骤：

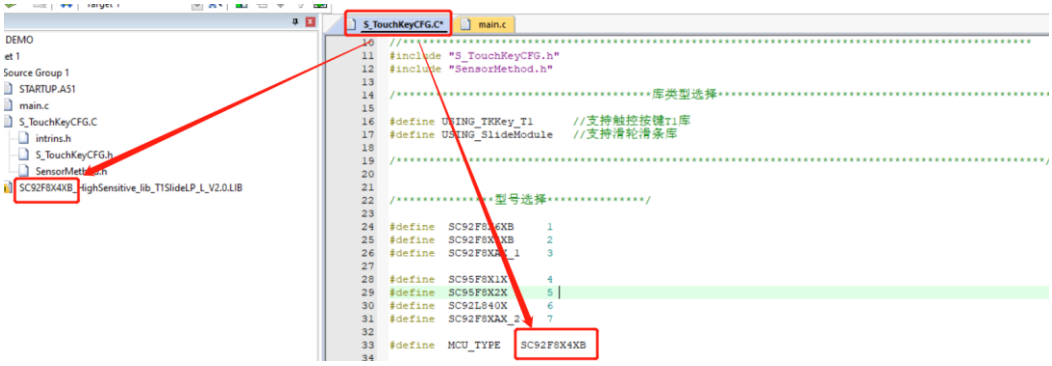
1) 在 Options for Target\C51 的 include Path 中添加 lib 文件夹的路径、设置 S_TouchKeyCFG.c 中的#define,根据库的类型设置对应的#define，例如 T1Slide 的库;



选用库体	库体选择开启宏定义
XXXXX_HighSensitive_Lib_Slide_L_VXXXX	**#define USING_SlideModule**
XXXXX_HighSensitive_Lib_T1Slide_L_VXXXX	**#define USING_TKKey_T1 #define USING_SlideModule**
XXXXX_HighSensitive_Lib_T2Slide_L_VXXXX	**#define USING_TKKey_T2 #define USING_SlideModule**
XXXXX_HighSensitive_Lib_SlideLP_L_VXXXX	**#define USING_SlideModule**
XXXXX_HighSensitive_Lib_T1SlideLP_L_VXXXX	**#define USING_TKKey_T1 #define USING_SlideModule**
XXXXX_HighSensitive_Lib_T2SlideLP_L_VXXXX	**#define USING_TKKey_T2 #define USING_SlideModule**

2) main.c 添加引用头文件 #include " SensorMethod.h" 既可以使用该库。**请留意你所选择的库是否**

与你的芯片型号匹配，如 SC95F8447B 芯片应使用“SC92F8X4XB_HighSensitive_Lib_XXSlideLP_L_V2.0”的库，S_TouchKeyCFG.C 中的型号也一致。



到此，完整的赛元滑轮滑条触控软件库已添加至项目工程内。

注意：应用程序中需要将 TK 对应的 IO 口设置为强推挽输出。

4.3.4 ARM 高灵敏触控软件库移植

1. 库文件介绍

1) 弹簧库文件（简称 T1 库，SC32F1XXX_HighSensitive_Lib_T1_Vx.x.LIB，IAR 平台是.a 后缀）

2) 隔空库文件（简称 T2 库，SC32F1XXX_HighSensitive_Lib_T2_Vx.x.LIB，IAR 平台是.a 后缀）

以下是 T1/T2 库文件简单介绍：（库体含有 4 个文件）

文件	用途	说明
SC32F1XXX_HighSensitive_LIB	库文件，实现触摸按键检测算法	**KEIL 平台是.LIB 后缀 IAR 平台是.a 后缀**
SC32F1XXX_TK_FunctionResource.h	头文件，提供接口函数供用户调用	声明的函数可供外部调用
SC32F1XXX_TK_ParameterAnalysis.C	**C 文件， 实现触控参数与库交互**	
S_TOUCHKEYCFG.H	头文件，提供宏供用户修改参数	

2. Lib API 接口函数的调用说明

函数	用途	说明
TK_Init (void)	触摸按键初始化	**1 用户在上电复位后调用一次； 2 本函数通过 S_TOUCHKEYCFG.H 参数配置用户选定的按键通道、按键参数并初始化 Baseline 基线； 3 执行本函数用时约：51~564ms 与按键个数，按键扫描时间，自动校准次数相关; T1 按键每多 N 个，大约时间 32M 主频：46 uS *N 按键； 16M 主频：49 uS *N 按键 ； T2 按键每多 N 个，大约时间 32M 主频：47 uS *N 按键； 16M 主频：51 uS *N 按键 ；**
TK_Restart (void)	使能一轮触摸按键扫描	**1 用户主程序控制何时启动按键扫描； 2 启动按键扫描后，在一轮触摸按键扫描完成之前，不能对触摸按键通道进行操作：如操作触摸按键通道的 IO，否则触摸按键功能将无法实现。**

Unsigned long int TK_TouchKeyScan (void)	**触摸按键算法处理**	**1 用户需要在触摸按键一轮扫描完成后调用； 2 如用户未调用该函数之前，一定不能重新调用 TK_Restart()，否则上一轮数据将被当前数据覆盖； 3 执行该函数用时约为：(T1 按键每多 N 个，大约时间 32M 主频：49 uS *N 按键； 16M 主频：98 uS *N 按键 ； T2 按键每多 N 个，大约时间 32M 主频：39 uS *N 按键； 16M 主频：77 uS *N 按键 ；**

3. 全局变量 TK_TouchKeyStatus 的说明

1) 全局变量在 S_TouchKeyCFG.c 头文件中声明

① Unsigned char TK_TouchKeyStatus;

② TK_TouchKeyStatusBit7 为 1 时 表示当前一轮扫描按键完成；

2) 该变量在用户主程序中调用

if(TK_TouchKeyStatus &0x80)时，调用 TK_TouchKeyScan(void) 进行算法数据处理，给出键值；

3) 使能触摸按键扫描之前，一定要清掉标志。

清除一轮扫描标志 TK_TouchKeyStatus &=0x7f;

4. LIB API 接口函数的返回值说明

1) TK_TouchKeyScan (void)函数返回值：

返回值对应 bit 为 1 即该通道有按键，0 为无按键。

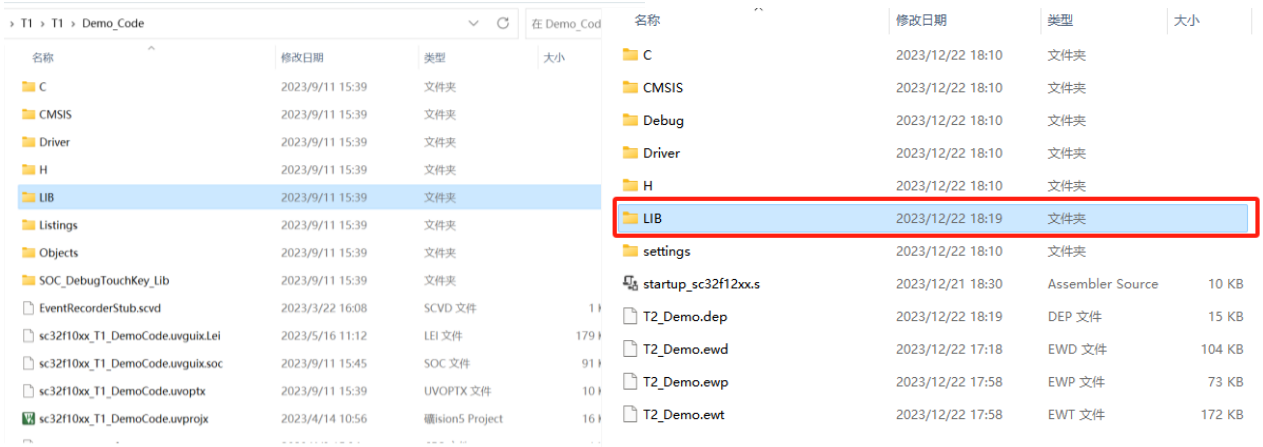
若使能双键，且有两键触发，则会有两个 bit 位置起。

以下以 TK0-31 举例说明

数据位	Bit31	Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24
含义	TK31	TK30	TK29	TK28	TK27	TK26	TK25	TK24
	触摸按键状态（1：有效；0：无效）							
数据位	Bit23	Bit22	Bit21	Bit20	Bit19	Bit18	Bit17	Bit16
含义	TK23	TK22	TK21	TK20	TK19	TK18	TK17	TK16
	触摸按键状态（1：有效；0：无效）							
数据位	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
含义	TK15	TK14	TK13	TK12	TK11	TK10	TK9	TK8
	触摸按键状态（1：有效；0：无效）							
数据位	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
含义	TK7	TK6	TK5	TK4	TK3	TK2	TK1	TK0
	触摸按键状态（1：有效；0：无效）							

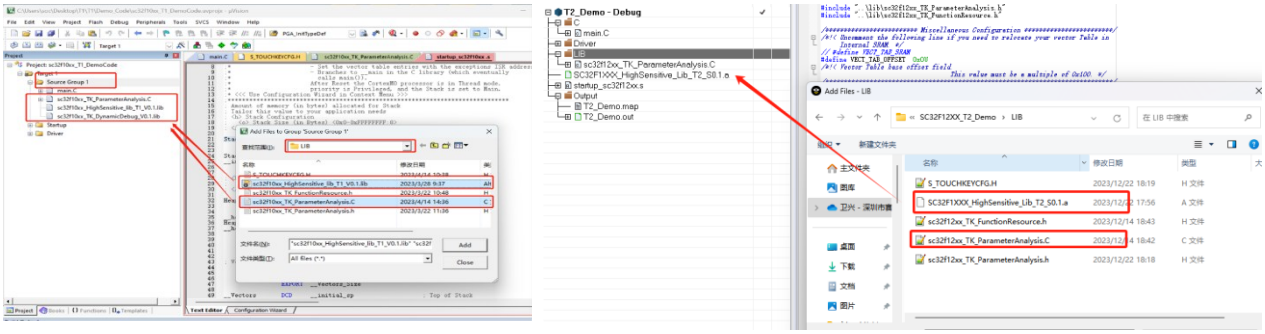
注：函数的返回类型是 unsigned long int ；TKn 为触控通道，具体请参照对应规格书。

5. 打开工程项目文件复制“lib”文件夹至工程文件夹内（左侧是 KEIL 工程,右侧是 IAR 工程）



6. 在项目工程中添加库文件 LIB 以及 sc32f1xxx_TK_ParameterAnalysis.C 文件

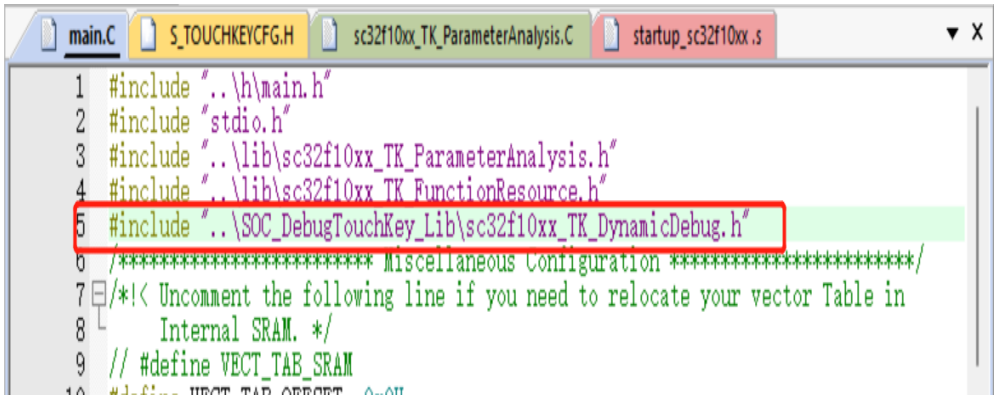
1) 从赛元提供的库体资料 Lib 文件夹内，添加库文件 LIB /.a 以及 sc32f1xxx_TK_ParameterAnalysis.C 至工程内。（左侧是 KEIL 工程,右侧是 IAR 工程）下图为例：（T1/T2 库操作一致）



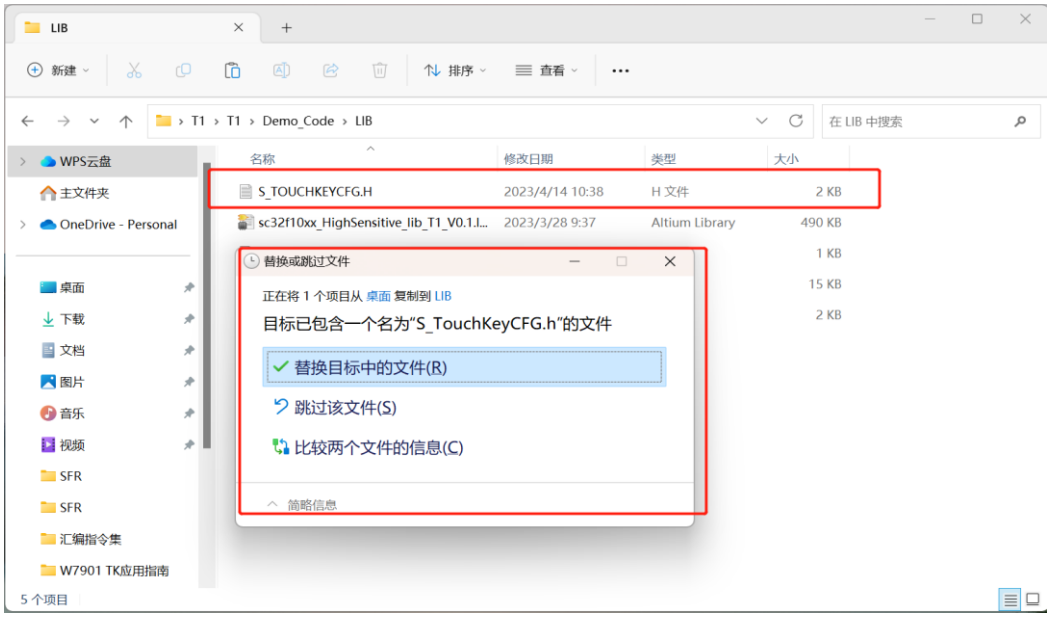
7. 在主程序文件中添加头文件引用

常规添加头文件 sc32f1xxx_TK_ParameterAnalysis.h 和 sc32f1xxx_TK_FunctionResource.h 即可。

红色框内的头文件是用到动态调试库的时候才需调用，请知悉。



8. 将 TK 触控调试上位机生成的配置文件 S_TOUCHKEYCFG.H 替换到 LIB 文件夹内



到此，完整的赛元触控高灵敏软件库已添加至项目工程内。

注意：应用程序中需要将 TK 对应的 IO 口设置为强推挽输出。

4.3.5 ARM 低功耗触控软件库移植

1. 库文件介绍
   1) 弹簧低功耗库文件（简称 T1 库，SC32F1XXX_HighSensitive_Lib_ T1LowPower_Vx.x.LIB，IAR 平台是.a 后缀）
   2) 隔空低功耗库文件（简称 T2 库，SC32F1XXX_HighSensitive_Lib_ T2LowPower_Vx.x.LIB，IAR 平台是.a 后缀）

   以下是 T1/T2 库文件简单介绍：（库体含有 4 个文件）
2. Lib API 接口函数的调用说明

接口名	接口定义	调用说明
GetLowPowerScanFlag	获取低功耗模式标志：**0** 为正常运行模式，**1** 为低功耗模式	在主循环内区分两种模式，执行不同的程序分支
TouchKey_IntoLowPowerMode	进入低功耗模式	调用该函数进入低功耗模式。用户根据实际应用决定调用，例如，一段时间未有按键按下进入低功耗，调用该函数。但需注意，进入低功耗要在无按键的情况下进入(即返回的键值为 **0**)
Customer_IntoLowPowerMode_Init	用户进入低功耗模式前的初始化	该函数用户需要自行添加在主程序中，并对其进行实现，无需调用. 如果程序中有使用中断资源，也请在此函数中关闭对应中断使能，以防应用程序逻辑上不严谨导致功能异常。
LowPower_Touchkey_Scan	低功耗 **TK** 扫描处理	在低功耗模式下调用
TouchKey_QuitLowPowerMode	退出低功耗模式	调用该函数系统进入正常模式。当 **TK** 按键被唤醒，自动被调用，退出低功耗。用户也可以根据外部信号退出低功耗

Customer_QuitLowPowerMode_Init	用户退出低功耗模式前的设置	该函数用户需要自行添加在主程序中，并对其进行实现，无需调用。 同时也需在此函数中使能之前所关闭的对应中断资源。
Customer_BTM_Dispose	用户 BTM 函数处理	该函数用户需要自行添加在主程序中，并对其进行实现，无需调用。由于 TK 低功耗的实现占用了 BTM 资源，这里开放了用户接口，用户在此函数内添加 BTM 唤醒的执行内容，例如读取外部的 IO 状态

3. 参数说明

打开 S_TouchKeyCFG.c 文件，将低功耗功能的宏开关打开（当用户无须低功耗功能可以将其注释，以达到节省代码空间），设置低功耗唤醒的个数，低功耗下要扫描的按键，以及 BTM 定时唤醒的时间

参数名	参数定义	设置说明
WakeUpKeyNum	低功耗模式下扫描的按键个数	不可以超过正常扫描的按键个数，ARM 系列功耗与按键个数无关
WakeUpKeyChannel1	低功耗下扫描的按键通道	只能设置正常扫描下已开启的按键，将对应的按键 BIT 位置起，置起的位数必须与设置的按键个数相同，用于存放 32 位 BIT 以内的按键
WakeUpKeyChannel2	低功耗下扫描的按键通道	只能设置正常扫描下已开启的按键，将对应的按键 BIT 位置起，置起的位数必须与设置的按键个数相同，用于存放 32 位 BIT 以上的按键
TK_SeepTimervSetting	低功耗下按键间的扫描周期	**BTM_TIMEBASE_15600US** **BTM_TIMEBASE_31300US** **BTM_TIMEBASE_62500US** **BTM_TIMEBASE_125MS** **BTM_TIMEBASE_250MS** **BTM_TIMEBASE_500MS** **BTM_TIMEBASE_1S** **BTM_TIMEBASE_2S** **BTM_TIMEBASE_4S** 可设置以上的选择，设置的时间越长功耗越低，按键响应速度越慢。用户可根据实际需要进行设置。
TK_WakeUpConfirmTouchCnt	低功耗下按键的确认次数	可以设置 10-20

4. 将相应型号的低功耗触控库移植到工程与 ARM 高灵敏库触控软件库移植一致，可参考 4.3.4

到此，完整的赛元低功耗触控软件库已添加至项目工程内。

注意：应用程序中需要将 TK 对应的 IO 口设置为强推挽输出。

4.4完成用户程序和赛元触控软件库的融合

4.4.1 高灵敏触控软件库和用户程序

1. 主程序和库文件的整体结构关系
   1) 通过添加库文件至工程项目内，并在用户程序中包含指定的头文件，调用库内的接口函数即可以增加触摸按键的功能。
   2) 库函数仅在主程序调用时才会运行。库文件会占用一些 ROM、 RAM、寄存器、中断等资源，但不占用定时器资源。

3) 库函数只管触摸按键功能，用户必须自己处理其他 的控制部分，如：输入输出、LED 数码管显示、通讯等功能。

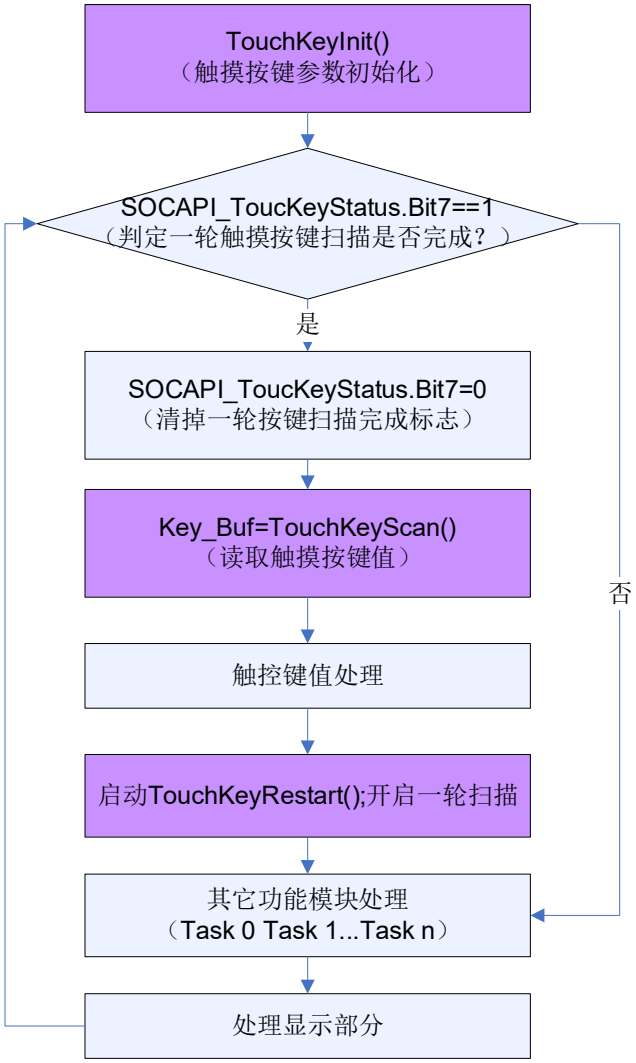
2. C51 库文件的调用流程（**弹簧和隔空库体调用流程有差异，请仔细阅读**）

用户通过一定的流程调用库文件的接口函数，便可得到触摸按键的键值。

弹簧库文件调用流程（简称 T1 库）

1) 将 TK 对应的 IO 设置为强推挽输出。
2) 主程序调用接口函数“TouchKeyInit()”用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 主程序通过查看全局变量 SOCAPI_ToucKeyStatus&0x80 来判定一轮触摸按键扫描是否完成；
4) 主程序调用接口函数“TouchKeyScan()”用于读取触摸按键值；
5) 主程序调用“TouchKeyRestart()”启动下一轮扫描。

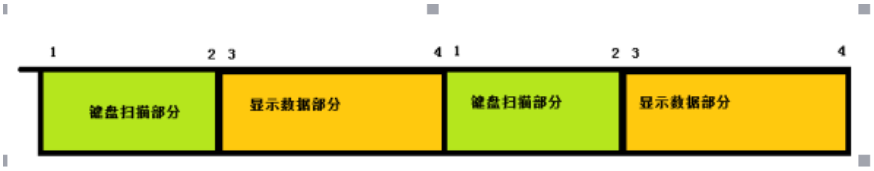
（下图中紫色的部分是库文件，其它部分是用户程序）



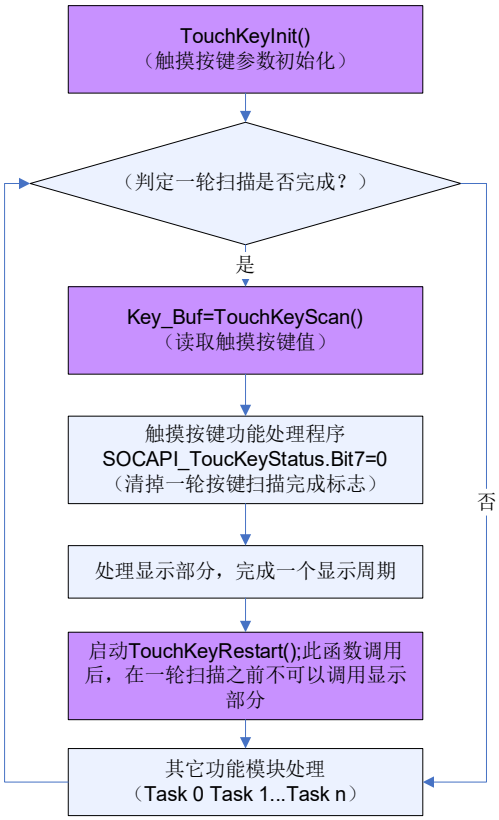
隔空库文件调用流程（简称 T2 库）

1) 将 TK 对应的 IO 设置为强推挽输出
2) 主程序调用接口函数“TouchKeyInit()”用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 若按键个数大于 8 个，主程序通过查看全局变量 SOCAPI_ToucKeyStatus&0x40 来判定半轮触控按 键扫描是否完成；半轮按键扫描完成后跳转完成一个周期的显示后再完成后半轮按键的扫描。
4) 主程序通过查看全局变量 SOCAPI_ToucKeyStatus&0x80 来判定一轮触摸按键扫描是否完成；

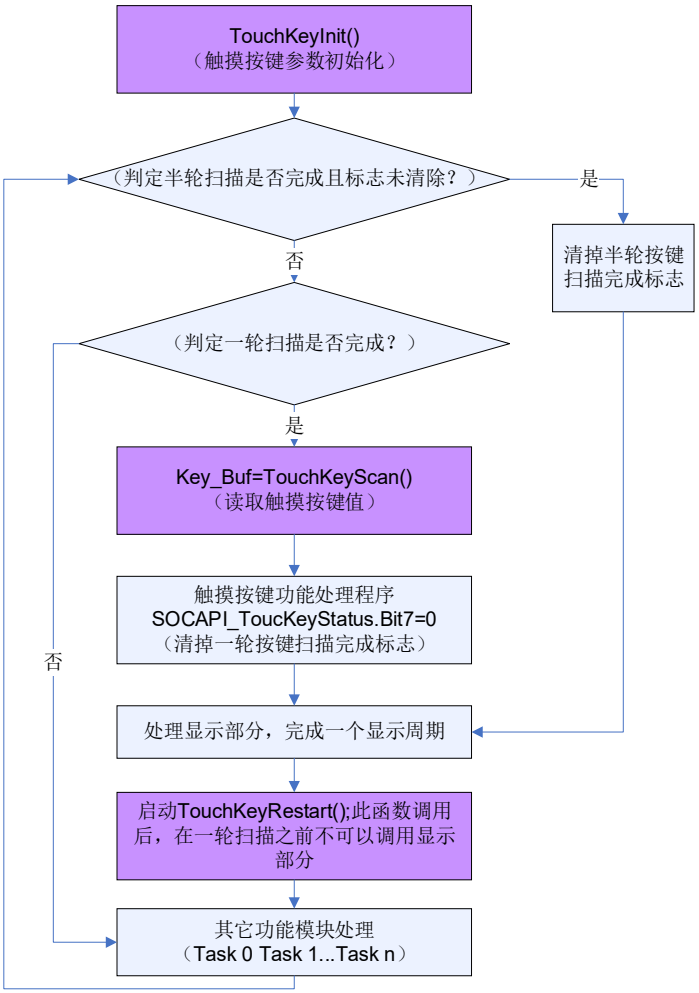
5) 主程序调用接口函数“TouchKeyScan()”用于读取触摸按键值；
6) 特别需要强调一点的是：调用 TouchKeyRestart()开始扫描按键时后，一轮扫描或者半轮扫描的标志 还没有出现时， 一定不要去做显示数据的部分。



（下图中紫色的部分是库文件，其它部分是用户程序）



用户程序调用接口函数控制流程
（按键个数 8 个以下）



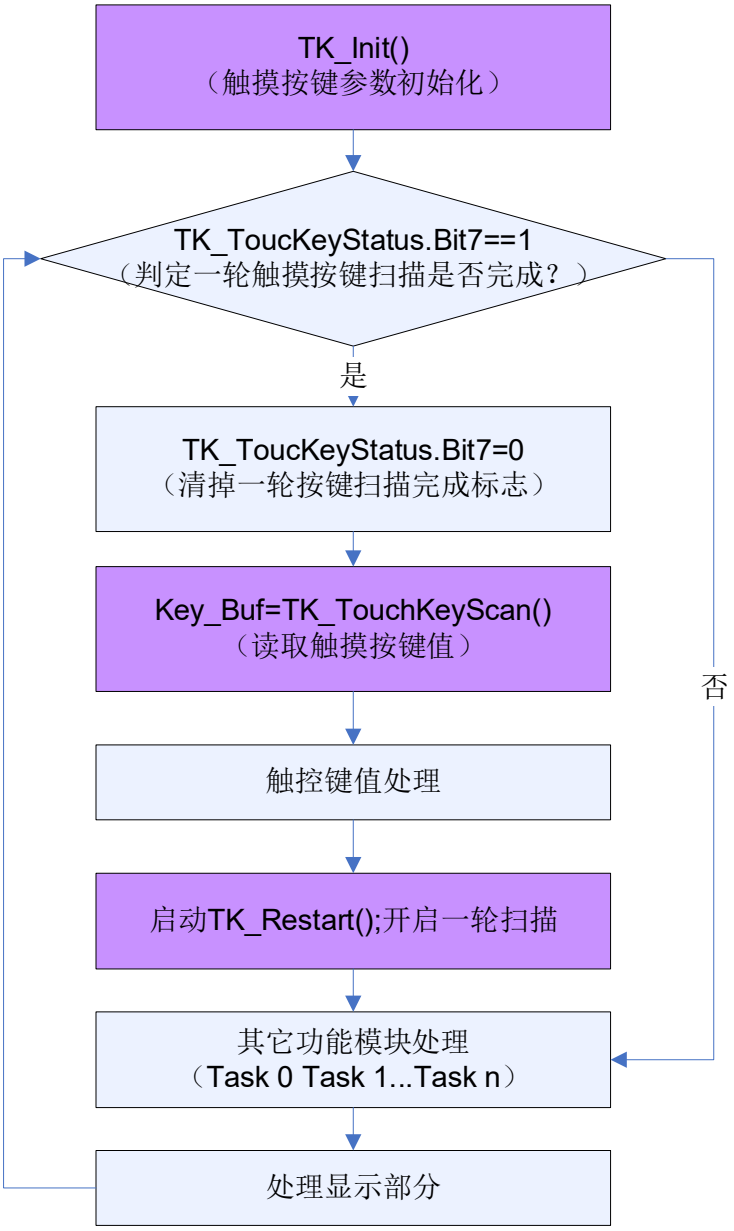
用户程序调用接口函数控制流程
（按键个数大于8个）

3. ARM 库文件的调用流程**（弹簧和隔空库体调用流程有差异，请仔细阅读）**
用户通过一定的流程调用库文件的接口函数，便可得到触摸按键的键值。

弹簧库文件调用流程（简称 T1 库）

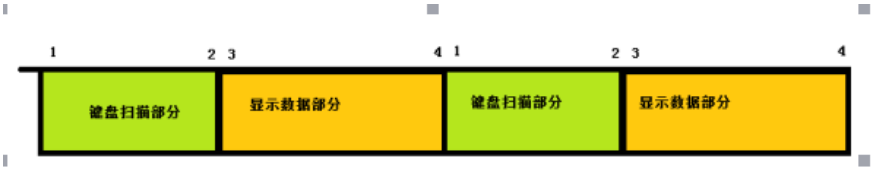
1) 将 TK 对应的 IO 设置为强推挽输出。
2) 主程序调用接口函数“TK_Init()”用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 主程序通过查看全局变量 TK_ToucKeyStatus&0x80 来判定一轮触摸按键扫描是否完成；
4) 主程序调用接口函数“TK_TouchKeyScan()”用于读取触摸按键值；
5) 主程序调用“TK_Restart()”启动下一轮扫描。

（下图中紫色的部分是库文件，其它部分是用户程序）

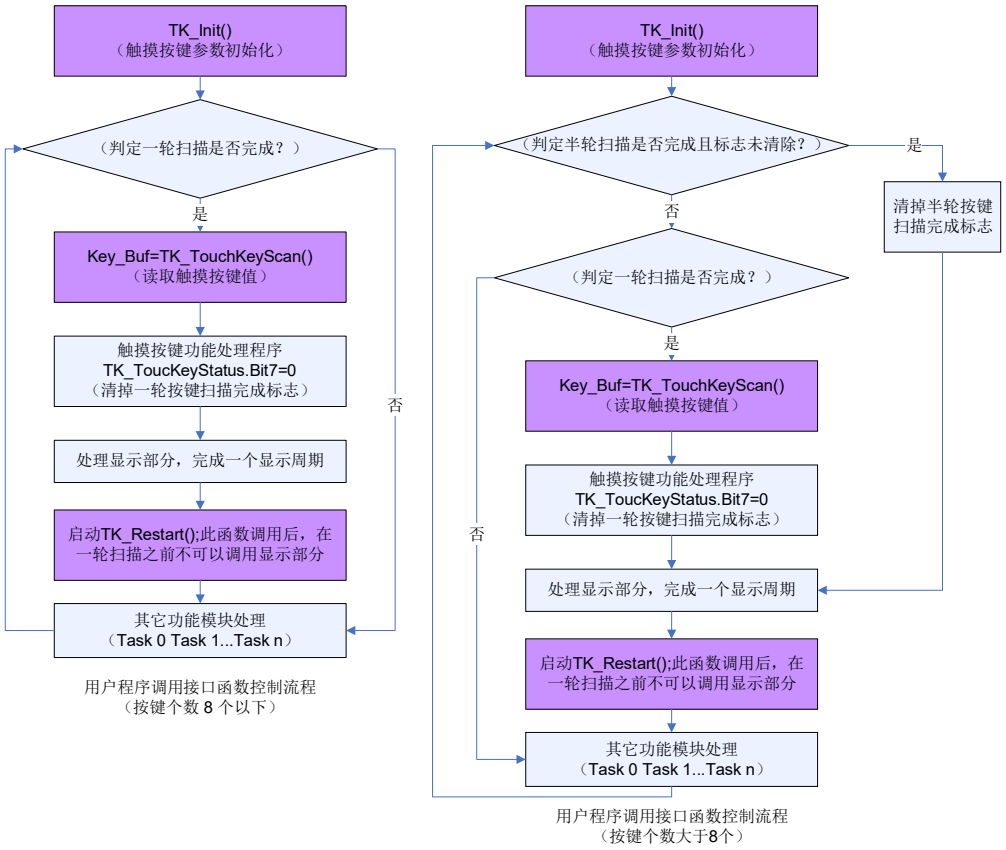


隔空库文件调用流程（简称 T2 库）

1) 将 TK 对应的 IO 设置为强推挽输出
2) 主程序调用接口函数“TK_Init()”用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 若按键个数大于 8 个，主程序通过查看全局变量 TK_ToucKeyStatus&0x40 来判定半轮触控按键扫描是否完成；半轮按键扫描完成后跳转完成一个周期的显示后再完成后半轮按键的扫描。
4) 主程序通过查看全局变量 TK_ToucKeyStatus&0x80 来判定一轮触摸按键扫描是否完成；
5) 主程序调用接口函数“TK_TouchKeyScan()”用于读取触摸按键值；
6) 特别需要强调一点的是：调用 TK_Restart()开始扫描按键时后，一轮扫描或者半轮扫描的标志还没有出现时， 一定不要去做显示数据的部分。



（下图中紫色的部分是库文件，其它部分是用户程序）



4. 主程序和库文件的时序关系

因为运行触摸按键库消耗了部分 IC 资源和时间，为了让用户程序和库程序 能完美融合，主程序需要遵循以下要求：

1) 提供给库运行的资源 ROM、RAM 和时间；
2) 启动按键扫描后，在一轮扫描未完成之前，不能对触摸按键通道进行操作；
   如触摸按键通道为输出 IO；否则触摸按键功能将无法实现；
3) 保证有足够的堆栈深度提供给主程序和库函数；
4) 触摸按键扫描取计数值数据的动作，是在中断内实现的，但数据的算法处理是在主程序中完成的。 用户需要按照一个合理的频度来调用库函数检测按键，以免错过按键动作；

5. **软件融合的注意事项:**

1) 运行时间：

C51 系列：

TouchKeyInit(void)： 算法执行时间会因按键选择个数的增减而增减，200~500ms@12M；

TouchKeyScan(void)：执行该函数用时与不同芯片主频有关联,请参考 4.3 内容表格中数据

ARM 系列：

TK_Init(void)： 算法执行时间会因按键选择个数的增减而增减，51~564ms@32M；

TK_TouchKeyScan(void)：执行该函数用时与不同芯片主频有关联,请参考 4.3 内容表格中数据。

2) 整体 Code 的测试：

用户完成程序调用后，请详细测试相关功能的性能，以防止软件的冲突。如发生异常情况，请在程序流程、调用时序、时间分配、堆栈、ROM/RAM/INT 资源等部分查找原因。

3) 关于整机调试的建议：

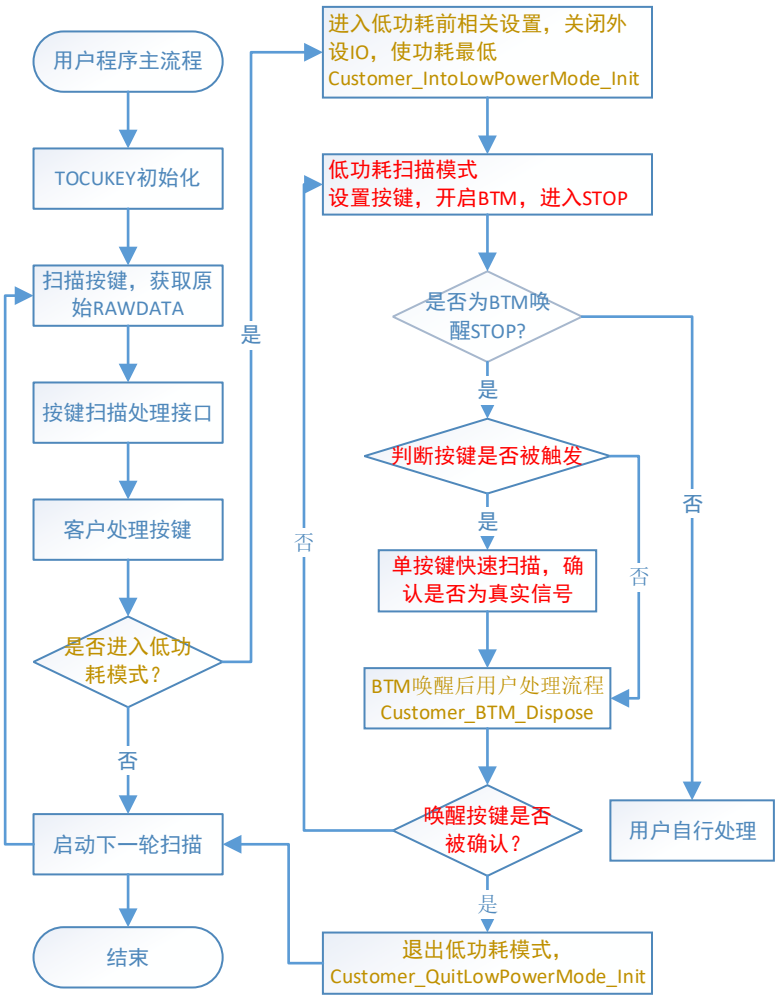
因为元器件的性能差异，建议用户可在一块 PCB 完成调试的情况下，多测试一些 PCB 的效果，以便取到折中效果的参数来去除材料对一致性的影响，更多的建议可以参考 5 触摸按键 MCU PCB 设计要点。

4.4.2 低功耗触控软件库和用户程序

1. C51 库文件的调用流程（**请仔细阅读**）

低功耗触控库文件调用流程（简称 T1 库）

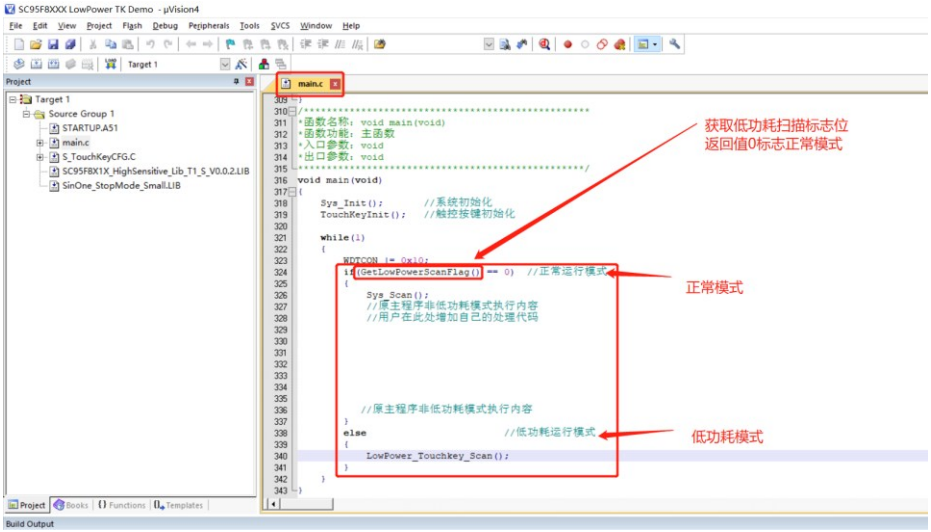
1) 设置唤醒按键的 TK，将 TK 对应的 IO 设置为强推挽输出。
2) 主程序调用触控初始化函数用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 主程序进行扫描按键，获取 RAWDATA，执行客户的处理按键程序，跳转下一轮扫描；
4) 若没有按键按下，主程序调用“Customer_IntoLowPowerMode_Init()”进入低功耗初始化，进入 STOP 模式，关闭其他外设 IO，开启 BTM；
5) 主程序调用 BTM 中断唤醒 STOP 模式，唤醒期间判断按键是否被触发，通过单按键快速扫描确认是否按键被按下；
6) 主程序调用“Customer_QuitLowPowerMode_Init()”退出低功耗模式，开启下一轮按键扫描，进入正常的按键处理程序。



2. 用户程序的调用过程（请仔细阅读）
   1) 在 main 文件中加入下面三个函数，并根据实际应用加入所需的实现代码

```c
/*************************************************
*函数名称：void Customer_IntoLowPowerMode_Init(void)
*函数功能：用户进入低功耗模式前的的初始化
*入口参数：void
*出口参数：void
*备注说明：在进入低功耗前，用户需要关闭外围耗电的电路，使电流最低
*          该函数已在S_TouchKeyCFG.C文件TouchKey_IntoLowPowerMode()函数中调用，用户无需调用
*************************************************/
void Customer_IntoLowPowerMode_Init(void)
{
    /*进入低功耗前设置*/

    //该函数已在TouchKey_IntoLowPowerMode()里调用

    //用户无需调用该函数！！！

    //用户需要自己编写该函数实体！！！

    //关闭耗电的外设，保持最低功耗
    printf("进入低功耗\r\n");
}
/*************************************************
*函数名称：void Customer_QuitLowPowerMode_Init(void)
*函数功能：用户退出低功耗模式后的的初始化
*入口参数：void
*出口参数：void
*备注说明：在退出低功耗前，用户进行的必要操作
*          该函数已在S_TouchKeyCFG.c文件TouchKey_QuitLowPowerMode()函数中调用，用户无需调用
*************************************************/
void Customer_QuitLowPowerMode_Init(void)
{
   /*退出低功耗前设置*/

   //该函数已在低功耗下满足TK唤醒退出低功耗时调用，即在TouchKey_QuitLowPowerMode()中调用

   //用户无需调用该函数！！！

   //用户需要自己编写该函数实体！！！

   //恢复必要的外设
    printf("退出低功耗\r\n");

}
/*************************************************
*函数名称：void Customer_BTM_Dispose(void)
*函数功能：用户BTM唤醒后自己的处理函数
*入口参数：void
*出口参数：void
*备注说明：低功耗实现利用了BTM资源，用户需要在BTM中实现的内容在此函数实现
           该函数已在S_TouchKeyCFG.c文件LowPower_Touchkey_Scan()函数中调用
*************************************************/
void Customer_BTM_Dispose(void)
{
   //该函数已在低功耗模式扫描函数中调用

   //此函数为BTM定时唤醒后，用户的处理函数

   //低功耗模式下，用户需在BTM中实现的功能可在此函数实现

   //比如查询某个IO，电平发生变化等条件成立，（TK唤醒除外）需退出低功耗模式可在此函数中调用TouchKey_QuitLowPowerMode()

   /*
   if(TEST_IO == 0)
   {
     //do someing
     TouchKey_QuitLowPowerMode();
   }
   */

    printf("BTM唤醒\r\n");

}
```

2) 在 main 函数中加入使用下图的调用方法



3) 用户根据实际应用需求在正常运行模式下添加是否进入低功耗的判断，需注意的是，进入低功耗要在无按键的情况下进入(即返回的键值为 0)，满足条件调用函数 TouchKey_IntoLowPowerMode()即可进入低功耗模式。

```
void Sys_Scan(void)
{
    if(SOCAPI_TouchKeyStatus&0x40)       //重要步骤2:  触摸键扫描半轮标志，是否调用TouchKeyScan()一定要根据此标志位置起后
     {
        SOCAPI_TouchKeyStatus &= 0xbf;  //清除标志位
         TouchKeyRestart();             //启动下一轮转换

     }
    if(SOCAPI_TouchKeyStatus&0x80)       //重要步骤2:  触摸键扫描一轮标志，是否调用TouchKeyScan()一定要根据此标志位置起后
     {
        SOCAPI_TouchKeyStatus &= 0x7f;  //清除标志位
        exKeyValueFlag = TouchKeyScan();//按键数据处理函数
        ChangeTouchKeyvalue();          //转换键值
        //这里设置进入低功耗扫描模式条件是：100轮没有检测到按键。也可以是其他，比如按下某个按键后多少S进入低功耗模式等
        if(exKeyValueFlag == 0)
        {
            NOKEYCount++;
            if(NOKEYCount>1000)
            {
                NOKEYCount = 0;
                if TouchKey_IntoLowPowerMode())
                {
                    return; //成功进入低功耗模式
                }
                else
                {
                    TouchKeyRestart(); //启动下一轮转换
                    return;
                }

            }
        }
        else
        {
            NOKEYCount = 0;
        }
        ////////////////////////////////////////////////
        TouchKeyRestart();              //启动下一轮转换
    }
```

可选择在正常运行TK扫描中添加是否满足进入低功耗条件的判断

判断这个函数返回值为1，成功进入低功耗模式

4) 低功耗模式下 BTM 中断及外部中断处理

```
/*****************************************************/
void LowPower_Touchkey_Scan(void)
{
    #ifdef  TK_LowPowerMode
    SleepMode();            //进入STOP模式
    //进行按键处理
    if(BTM_WakeUpFlag)
    {

        TouchKey_LowPower_Dispose();        //检测按键

        if( SingleKeyFastScan_Flag == 1)
        {
            SingleKeyFastScan();            //若有按键信息进入单按键快速多次扫描确定按键是否真实信号。
        }
        // 用户BTM唤醒后的处理函数
        Customer_BTM_Dispose();
    }
    //其他唤醒请在以下分支进行判断并调用退出低功耗函数TouchKey_QuitLowPowerMode()
    //例如：if(外部中断唤醒标志==1)
    //      {
    //          TouchKey_QuitLowPowerMode();
    //          //业务层唤醒后实现逻辑
    //      }
    //
    //
    //
    {
        ;               //非BTM唤醒，用户根据需要自行增添处理程序
    }
    #endif
}
```

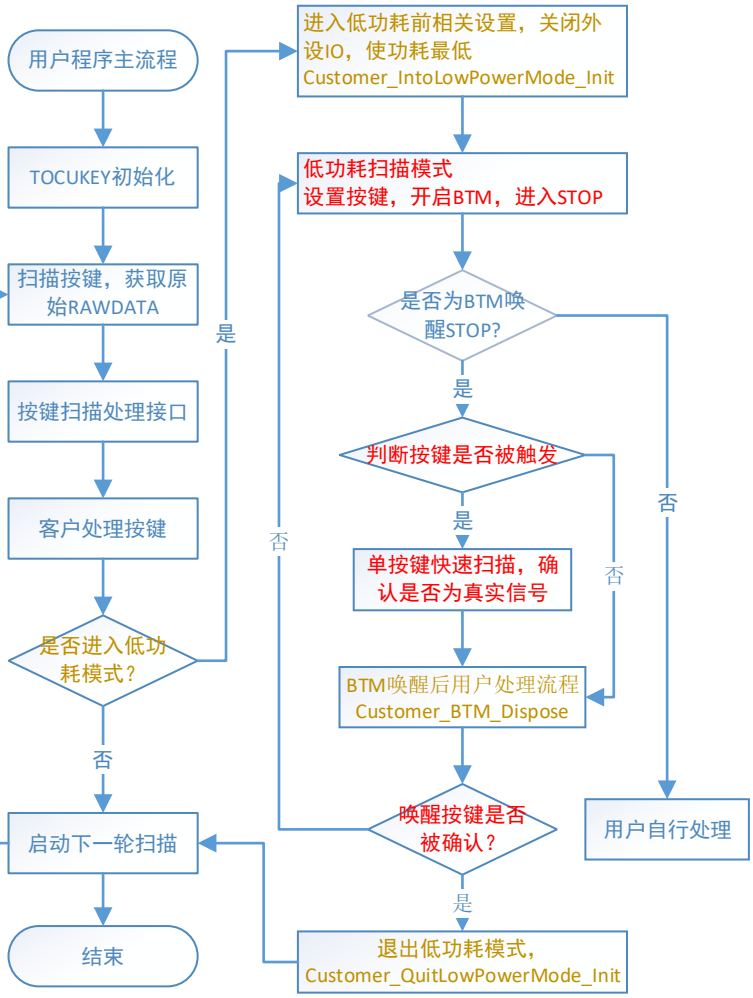
BTM唤醒后进行低功耗TK扫描

低功耗占用BTM资源，若用户需要在BTM中处理其他功能，可在此函数中添加相关处理程序

3. ARM 库文件的调用流程（**请仔细阅读**）

低功耗触控库文件调用流程（简称 T1 库）

1) 设置唤醒按键的 TK，将 TK 对应的 IO 设置为强推挽输出。
2) 主程序调用触控初始化函数用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 主程序进行扫描按键，获取 RAWDATA，执行客户的处理按键程序，跳转下一轮扫描；
4) 若没有按键按下，主程序调用“Customer_IntoLowPowerMode_Init()”进入低功耗初始化，进入 STOP 模式，关闭其他外设 IO，开启 BTM；
5) 主程序调用 BTM 中断唤醒 STOP 模式，唤醒期间判断按键是否被触发，通过单按键快速扫描确认是否按键被按下；
6) 主程序调用“Customer_QuitLowPowerMode_Init()”退出低功耗模式，开启下一轮按键扫描，进入正常的按键处理程序。



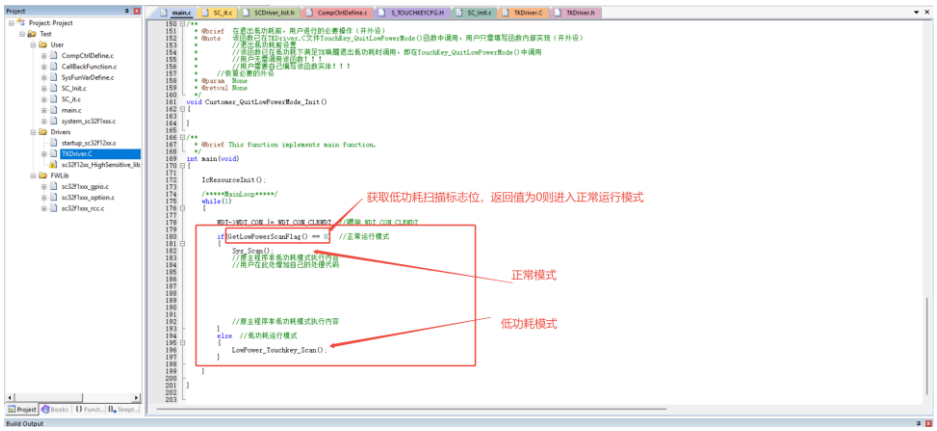
4. 用户程序的调用过程（请仔细阅读）

1) 在 main 文件中加入下面三个函数，并根据实际应用加入所需的实现代码

```
/*****************************************************
*函数名称: void Customer_IntoLowPowerMode_Init(void)
*函数功能: 用户进入低功耗模式前的的初始化
*入口参数: void
*出口参数: void
*备注说明: 在进入低功耗前，用户需要关闭外围耗电的电路，使电流最低
*          该函数已在S_TouchKeyCFG.C文件TouchKey_IntoLowPowerMode()函数中调用，用户无需调用
*****************************************************/
void Customer_IntoLowPowerMode_Init(void)
{
    /*进入低功耗前设置*/

    //该函数已在TouchKey_IntoLowPowerMode()里调用

    //用户无需调用该函数！！！

    //用户需要自己编写该函数实体！！！

    //关闭耗电的外设，保持最低功耗
    printf("进入低功耗\r\n");
}
/*****************************************************
*函数名称: void Customer_QuitLowPowerMode_Init(void)
*函数功能: 用户退出低功耗模式后的的初始化
*入口参数: void
*出口参数: void
*备注说明: 在退出低功耗前，用户进行的必要操作
*          该函数已在S_TouchKeyCFG.c文件TouchKey_QuitLowPowerMode()函数中调用，用户无需调用
*****************************************************/
void Customer_QuitLowPowerMode_Init(void)
{
    /*退出低功耗前设置*/

    //该函数已在低功耗下满足TK唤醒退出低功耗时调用，即在TouchKey_QuitLowPowerMode()中调用

    //用户无需调用该函数！！！

    //用户需要自己编写该函数实体！！！

    //恢复必要的外设
     printf("退出低功耗\r\n");

}
```

```
/*****************************************************
*函数名称：void Customer_BTM_Dispose(void)
*函数功能：用户BTM唤醒后自己的处理函数
*入口参数：void
*出口参数：void
*备注说明：低功耗实现利用了BTM资源，用户需要在BTM中实现的内容在此函数实现
          该函数已在S_TouchKeyCFG.c文件LowPower_Touchkey_Scan()函数中调用
*****************************************************/
void Customer_BTM_Dispose(void)
{
    //该函数已在低功耗模式扫描函数中调用

    //此函数为BTM定时唤醒后，用户的处理函数

    //低功耗模式下，用户需在BTM中实现的功能可在此函数实现

    //比如查询某个IO，电平发生变化等条件成立，(TK唤醒除外)需退出低功耗模式可在此函数中调用TouchKey_QuitLowPowerMode()

    /*
    if(TEST_IO == 0)
    {
      //do someing
      TouchKey_QuitLowPowerMode();
    }
    */

     printf("BTM唤醒\r\n");
}
```

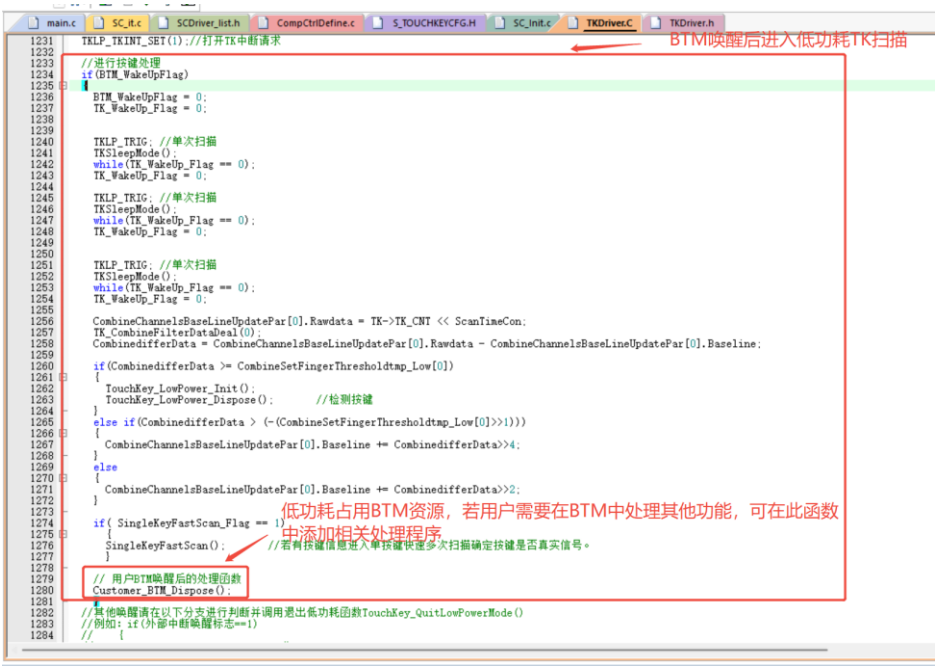
2) 在 main 函数中加入使用下图的调用方法



3) 用户根据实际应用需求在正常运行模式下添加是否进入低功耗的判断，需注意的是，进入低功耗要在无按键的情况下进入(即返回的键值为 0)，满足条件调用函数 TouchKey_IntoLowPowerMode()即可进入低功耗模式。

```
/**
* @brief 扫描TK和显示入口参数
* @note
* @param[in]
* @retval
*/
void Sys_Scan(void)
{
    if(TK_TouchKeyStatus&0x80)        //重要步骤2:  触摸键扫描一轮标志，是否调用TouchKeyScan()一定要根据此标志位置起后
    {
        TK_TouchKeyStatus &= 0x7f;
        //重要步骤3: 清除标志位， 需要外部清除。
        TK_exKeyValueFlag = TK_TouchKeyScan();
        //按键数据处理函数
        ChangeTouchKeyvalue();
        UpdateDisplay();
        if((TK_exKeyValueFlag == 0) && (CombineKeyFlag == 0))
        {
            NOKEYCount++;
            if(NOKEYCount>100)
            {
                NOKEYCount = 0;
                TouchKey_IntoLowPowerMode();
                return;
                //成功进入低功耗模式
            }
        }
        else
        {
            NOKEYCount = 0;
        }
        TK_Restart();
        //启动下一轮转换
    }
}
```


4) 低功耗模式下 BTM 中断及外部中断处理



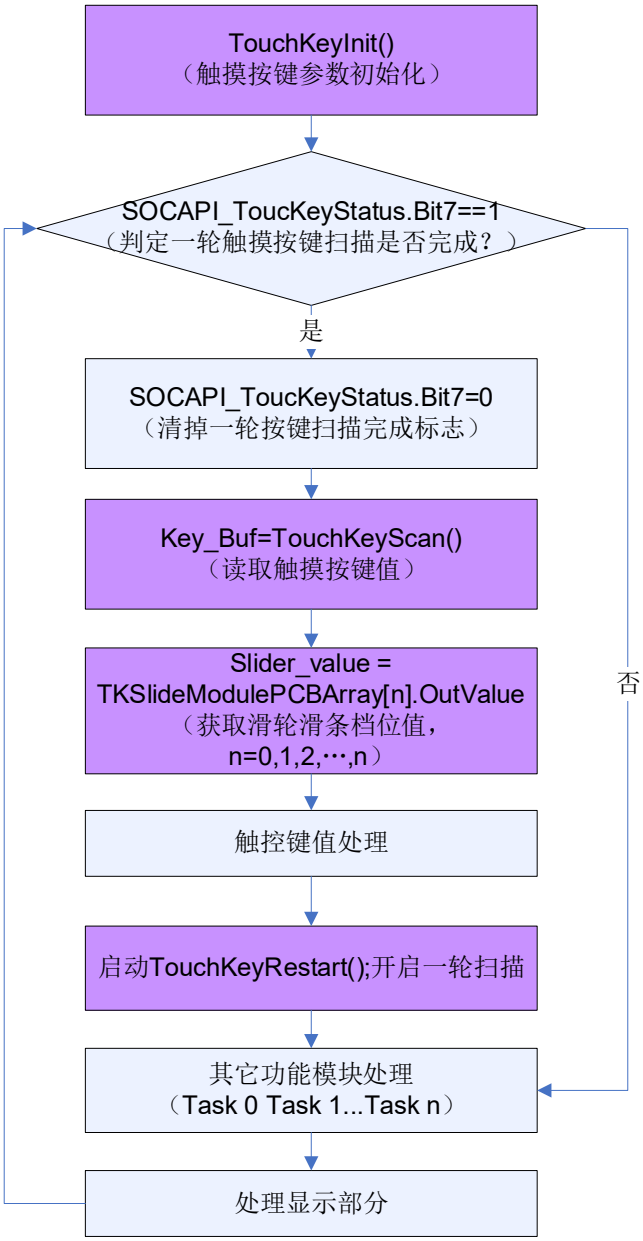
5. **软件融合的注意事项:**
   1) 赛元低功耗库支持需大于 12 个按键唤醒的低功耗应用。
   2) 赛元低功耗库在低功耗模式下仅支持单键。
   3) CMOD 口初始化设置强推挽输出模式，输出 0
   4) 关闭模拟外设和 WDT
   5) 悬空以及未封装出的 IO 设置为强推挽输出模式
   6) 根据实际应用电路设置 IO 口电平
   7) 建议电路匹配 CMOD 电容为 102
   8) ARM 系列：系统时钟源在进入低功耗后由 HIRC 提供系统时钟源，若系统时钟源不是选择 HIRC，则在进入低功耗前需要保存当前时钟源，并在退出低功耗模式后恢复当前时钟源

4.4.3 滑轮滑条触控软件库和用户程序

1. C51 库文件的调用流程（请仔细阅读）
   用户通过一定的流程调用库文件的接口函数，便可得到触摸按键的键值。

   弹簧库文件调用流程（简称 T1 库）
   1) 将 TK 对应的 IO 设置为强推挽输出。
   2) 主程序调用接口函数“TouchKeyInit()”用于配置触摸按键通道的参数，并初始化 Baseline 基线；
   3) 主程序通过查看全局变量 SOCAPI_ToucKeyStatus&0x80 来判定一轮触摸按键扫描是否完成；
   4) 主程序调用接口函数“TouchKeyScan()”用于读取触摸按键值；
   5) 主程序调用“TouchKeyRestart()”启动下一轮扫描。

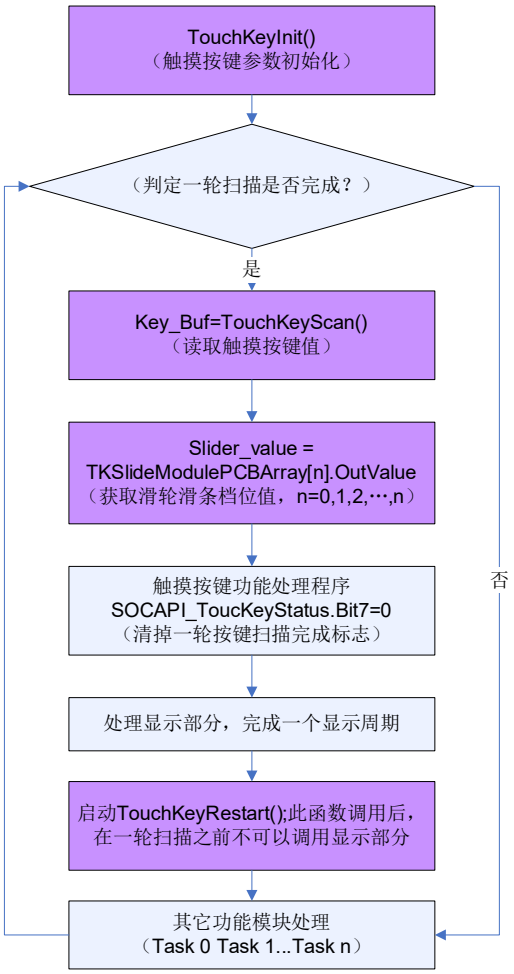
   （下图中紫色的部分是库文件，其它部分是用户程序）



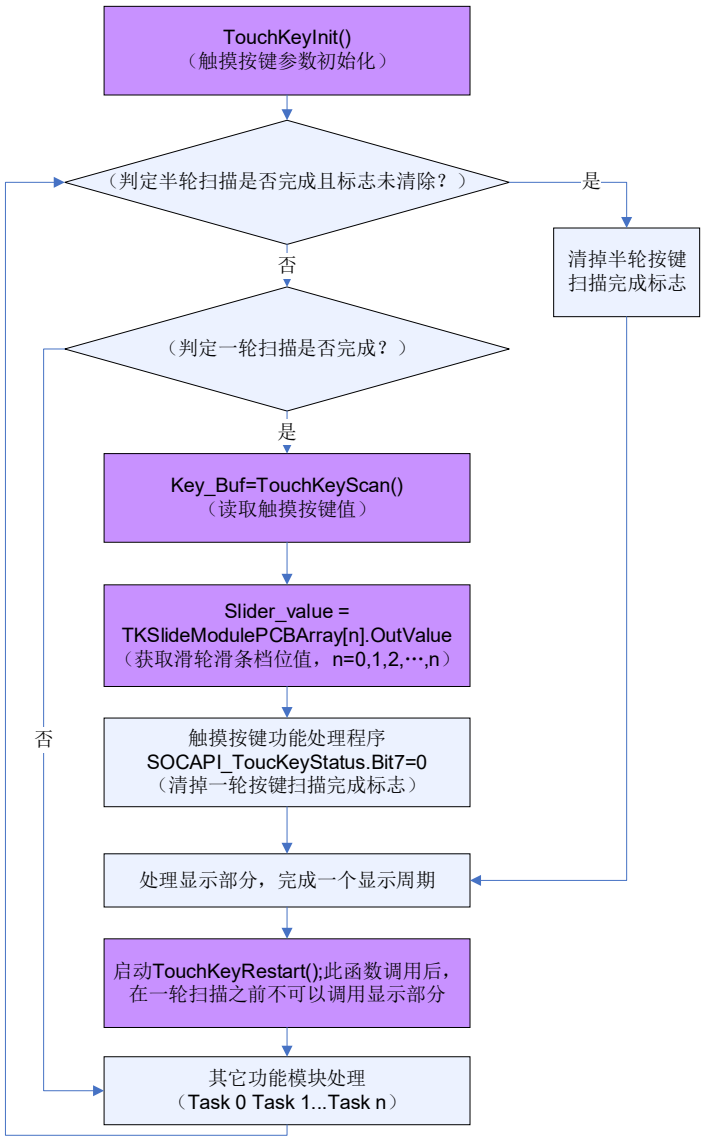
隔空库文件调用流程（简称 T2 库）

1) 将 TK 对应的 IO 设置为强推挽输出
2) 主程序调用接口函数“TouchKeyInit()”用于配置触摸按键通道的参数，并初始化 Baseline 基线；
3) 若按键个数大于 8 个，主程序通过查看全局变量 SOCAPI_ToucKeyStatus&0x40 来判定半轮触控按 键扫描是否完成；半轮按键扫描完成后跳转完成一个周期的显示后再完成后半轮按键的扫描。
4) 主程序通过查看全局变量 SOCAPI_ToucKeyStatus&0x80 来判定一轮触摸按键扫描是否完成；
5) 主程序调用接口函数“TouchKeyScan()”用于读取触摸按键值；
6) 特别需要强调一点的是：调用 TouchKeyRestart()开始扫描按键时后，一轮扫描或者半轮扫描的标志 还没有出现时， 一定不要去做显示数据的部分。

（下图中紫色的部分是库文件，其它部分是用户程序）



用户程序调用接口函数控制流程
（按键个数 8 个以下）



用户程序调用接口函数控制流程
（按键个数大于8个）

2. 用户程序的调用过程（请仔细阅读）

   1) 判断扫描是否完成，清除完成标志位，调用触控扫描函数 TouchKeyScan 获得更新档位值，并对数据做处理，然后启动下一轮转换（TouchKeyScan 函数的返回值为 TK 按键按下标志，滑动模块更新的位置值存放在 TKSlideModulePCBArray[X].OutValue）。

```
/*************************************************
*函数名称：void main(void)
*函数功能：
*入口参数：void
*出口参数：void
*功能说明：
*************************************************/
void main(void)
{
    Sys_Init(); //系统初始化

    TouchKeyInit();   //触控按键初始化

    while(1)
    {
        Sys_Scan(); //扫描
    }
}
```

```
/*************************************************
*函数名称：void  Sys_Scan(void)
*函数功能：扫描TK和显示
*入口参数：void
*出口参数：void
*************************************************/
void Sys_Scan(void)
{
    unsigned long int value;
    if(SOCAPI_TouchKeyStatus&0x80)      //重要步骤2：  触摸键扫描一轮标志，
    {
        SOCAPI_TouchKeyStatus &= 0x7f;  //清除标志位


        value = TouchKeyScan();     //扫描处理，获取触控按键等状态数据
        DataProcessing(value);//扫描TK更新数据并处理

        TouchKeyRestart();              //启动下一轮转换
    }
}
```

2) 在 DataProcessing 函数中对扫描值进行处理如下滑动模块更新的位置值存放在 TKSlideModulePCBArray[X].OutValue：

```
/*************************************************
*函数名称：void DataProcessing(unsigned long int value)
*函数功能：对触控按键、滑轮、滑条扫描值处理
*入口参数：value
*出口参数：void
*************************************************/
void DataProcessing(unsigned long int value)
{
//  if(TKSlideModulePCBArray[0].OutValue != 0)      //判断是否被按下
//  {
//      //输出W2滑轮滑动位置值
//      printf("W2 value:%d \r\n",TKSlideModulePCBArray[0].OutValue);
//  }
//  if(TKSlideModulePCBArray[1].OutValue != 0)      //判断是否被按下
//  {
//      //输出W1滑轮滑动位置值
//      printf("W1 value:%d \r\n",TKSlideModulePCBArray[1].OutValue);
//  }
//  if(TKSlideModulePCBArray[2].OutValue != 0)      //判断是否被按下
//  {
//      //输出S1滑条滑动位置值
//      printf("S1 value:%d \r\n",TKSlideModulePCBArray[2].OutValue);
//  }
//  if(TKSlideModulePCBArray[3].OutValue != 0)      //判断是否被按下
//  {
//      //输出S3滑条滑动位置值
//      printf("S3 value:%d \r\n",TKSlideModulePCBArray[3].OutValue);
//  }
//
//  if(value != 0)
//  {
//      //输出触摸按下按下的值
//      printf("按下 value: %x\r\n", value);
//  }
}
```

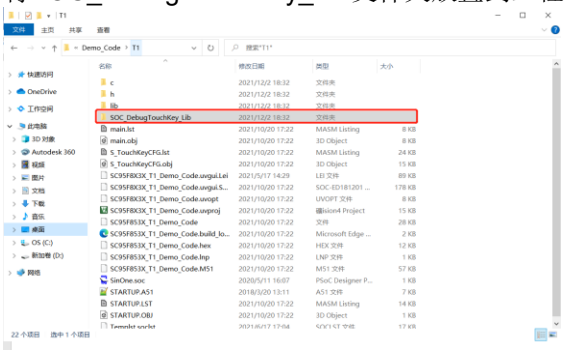
(具体使用可以参考多滑轮滑条库资料中的 Demo_Code)

4.5附加功能-动态调试功能

主要功能：利用赛元触控调试上位机软件查看实时的数据情况，帮助用户对系统进行整体的评估，了解系统实际运行的情况，分析异常等。

4.5.1 C51 高灵敏度动态调试步骤

1. 将SOC_DebugTouchKey_Lib文件夹放置到工程的根目录下



2. 在 main.c 中 include 头文件

```
#include "SOC_DebugTouchKey_Lib\SOC_DebugUart.h"

#ifdef DEBUG_ENABLE
#include "SOC_DebugTouchKey_Lib\SOC_DebugTouchKey.h"
#endif
```

3. 在 SOC_DebugUart.h 中使能动态调试

SOC_DebugUart.h

```
1  #ifndef _SOC_DEBUG_UART_H_
2  #define _SOC_DEBUG_UART_H_
3
4  /**********************动态调试使用说明*****************************/
5  //1.打开动态调试
6  //2.选择对应芯片型号，通过IC_MODEL宏控制
7  //3.选择对应UART口，通过UART_MODEL宏控制，uart初始化内代码实现无需修改
8  //4.选择对应调试模式，动态调试或printf打印
9  //5.烧录OPTION中需注意LVR需要低于芯片VDD供电电压;其他OPTION设置按实际默认即可，无需修
10 /*******************************************************************/
11
12 /*****************************************
13 说明：调试模式是否开启，屏蔽以下宏定义表示不开启调试模式
14 *****************************************/
15 #define DEBUG_ENABLE
```

4. 选择对应的 IC 型号，这里以 SC95F8617B 芯片型号为例，如图所示：

```
16 /*****************************************
17 说明：选择IC型号
18 *****************************************/
19 //sc92_sc93
20 #define SC92F8x1xb      1   //SC92F841xB
21 #define SC92F8x1x       2   //SC92F841x
22 #define SC92F8x4xb      3   //SC92F844xB
23 #define SC92F8x47_6_5   4   //SC95F8547_8546_8545
24 #define SC92F8x43_2_1   5   //SC95F8543_8542_8541
25 #define SC92F8x6xb      6   //SC92F846xB
26 #define SC92F8x7x       7   //SC92F837x
27 #define SC92F8x8x       8   //SC92F848x
28 #define SC92F8x9x       9   //SC92F859x
29 #define SC92F8xAx_1     10  //84A3_84A2_84A1
30 #define SC92F8xAx_2     11  //84A7_84A6_84A5
31 #define SC92F8xHx       12  //SC92F84Hx
32 #define SC92L8x3x       13  //SC92L853x
33 #define SC92R342        14  //SC92R342
34 #define SC92R436        15  //SC92R436
35 #define SC92R439        16  //SC92R439
36 #define SC93F8x3x       17  //SC93F843x
37 //sc95
38 #define SC95F8x1xb      18  //SC95F861xB
39 #define SC95F8x1x       19  //SC95F861x
40 #define SC95F8x2x       20  //SC95F852x
41 #define SC95F8x3x       21  //SC95F873x
42 #define SC95F8x6x       22  //SC95F876x
43 #define SC95F8x7x       23  //SC95F857x
44 #define SC95R503        24  //SC95R503
45 #define SC95R605        25  //SC95R605
46 #define SC95R751        26  //SC95R751
47
48 #define IC_MODEL    SC95F8x1xb    //芯片型号控制宏修改
49
50 /*.....................................
```

5. 提供两种调试模式，动态调试需要配合赛元 Touch Key Tool Menu 上位机使用，printf 打印需要配合串口调试助手使用。

```
/*********************************************
说明：选择调试模式
*********************************************/
#define DynamicDebug     0    //动态调试
#define PrintfDebug      1    //printf打印

#define DebugMode    PrintfDebug //调试模式选择
```

6. 选择 printf 打印时，需要在扫描函数 Sys_Scan()后调用打印函数。

```
/*****************************************************
 *函数名称：void  Sys_Scan(void)
 *函数功能：扫描TK和显示
 *入口参数：void
 *出口参数：void
 *****************************************************/
 void Sys_Scan(void)
{
    if(SOCAPI_TouchKeyStatus&0x80)      //重要步骤2:  触摸键扫描一轮标志，是否调用TouchKeyScan()一定要根据此标志位置1
     {
        SOCAPI_TouchKeyStatus &= 0x7f;  //重要步骤3: 清除标志位， 需要外部清除。
        exKeyValueFlag = TouchKeyScan();//按键数据处理函数

#ifdef DEBUG_ENABLE
#if(DebugMode == PrintfDebug)       //打印模式时，默认波特率38400，入参填写使用到的按键个数
        SOCAPI_Pirnt_TKData(4);
#endif
#endif

        TouchKeyRestart();              //启动下一轮转换
    }
}
```

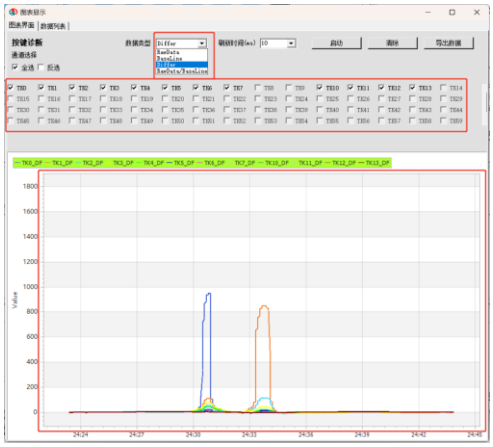
7. 在 main 函数中调用 SOCAPI_DeBugTouchKey_Init 进行初始化。编译成功后烧录到芯片
在使用外部高频晶振作为时钟源时， SOCAPI_DeBugTouchKey_Init()的形参则按照外部高频晶振的频率进行填写。

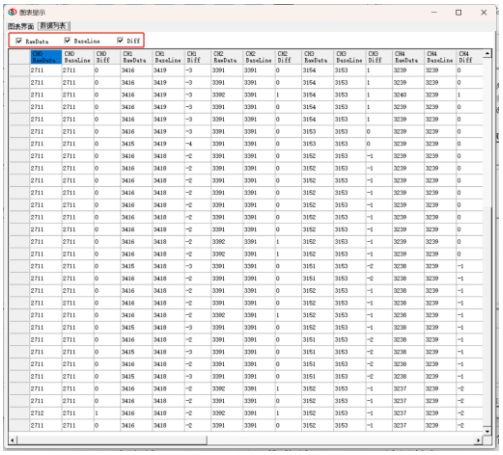
```
}
/*****************************************************
 *函数名称：void main(void)
 *函数功能：主函数
 *入口参数：void
 *出口参数：void
 *****************************************************/
 void main(void)
{
    Sys_Init();

    //触控按键初始化
    TouchKeyInit();

#ifdef DEBUG_ENABLE
    SOCAPI_DeBugTouchKey_Init(32000000);
#endif

    while(1)
    {
        WDTCON  = 0x10;
        if(TimerFlag_1ms==1)
        {
            TimerFlag_1ms=0;
            Timercount++;
            if(Timercount>10)
            {
                Timercount=0;
                Sys_Scan();
            }
        }

    }
}
```

8. 打开 Touch Key Tool Menu,选择高灵敏度触控，芯片型号选择与实际芯片对应的型号，调试模式选择动态调试，勾选 TK 通道（**必须与项目实际使用的通道一致**）

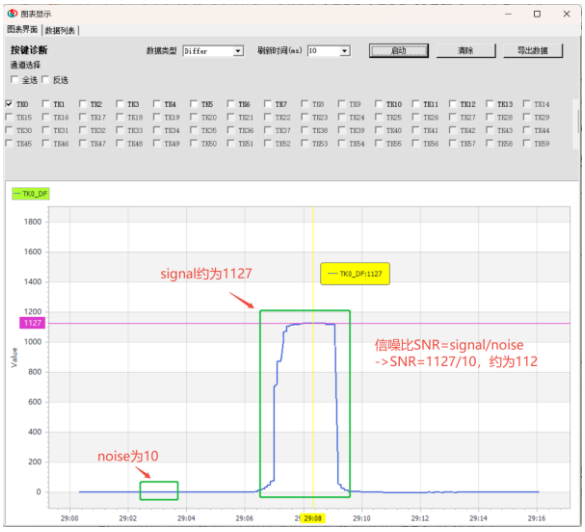
9. 点击确定按钮，然后点击下方“动态调试”按钮

10. 在动态调试界面内，可以通过选择“数据类型”查看想要查看的数据，通过“通道选择”勾选要查看的通道，在图表显示内可以实时看到数据的情况，点击“数据列表”可以查看图表数据



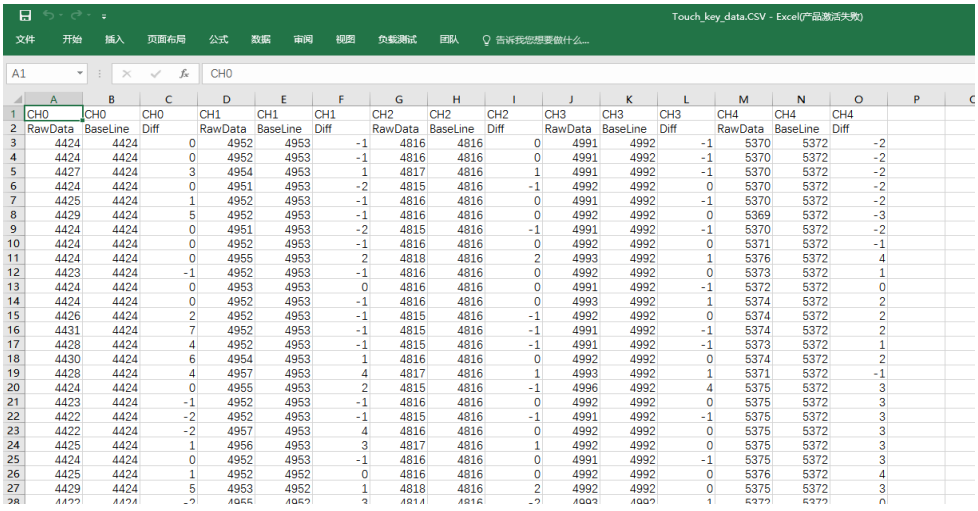


注意：动态调试观测数据时，需要注意信噪比 SNR 是否符合使用条件。
建议是 SNR > 5 可用，推荐是 SNR > 10 效果较好。



SNR 计算方式：动态调试观测中
Noise 噪声幅度为：静置状态下，没有手按下时 Differ 的最大值减去最小值的差值 Differ0
Signal 信号幅度为：手指按下时 Differ 的平均值 Differ1(图示绿色框内所示)
SNR 计算方式：SNR = Signal/Noise=Differ1/Differ0,如上图计算 SNR=112 左右

11. 点击“导出数据”可以将实时采集数据导出 CSV 格式文档

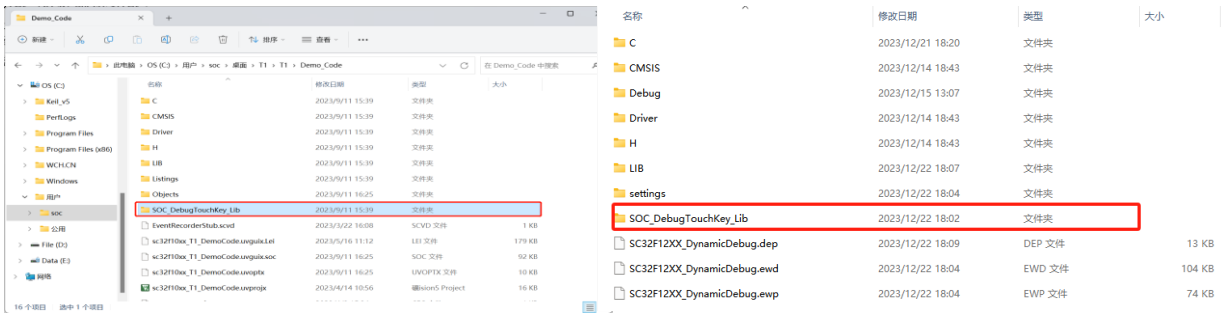


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	CH0	CH0	CH0	CH1	CH1	CH1	CH2	CH2	CH2	CH3	CH3	CH3	CH4	CH4	CH4
2	RawData	BaseLine	Diff	RawData	BaseLine	Diff	RawData	BaseLine	Diff	RawData	BaseLine	Diff	RawData	BaseLine	Diff
3	4424	4424	0	4952	4953	-1	4816	4816	0	4991	4992	-1	5370	5372	-2
4	4424	4424	0	4952	4953	-1	4816	4816	0	4991	4992	-1	5370	5372	-2
5	4427	4424	3	4954	4953	1	4817	4816	1	4991	4992	-1	5370	5372	-2
6	4424	4424	0	4951	4953	-2	4815	4816	-1	4992	4992	0	5370	5372	-2
7	4425	4424	1	4952	4953	-1	4816	4816	0	4991	4992	-1	5370	5372	-2
8	4429	4424	5	4952	4953	-1	4816	4816	0	4992	4992	0	5369	5372	-3
9	4424	4424	0	4951	4953	-2	4815	4816	-1	4991	4992	-1	5370	5372	-2
10	4424	4424	0	4952	4953	-1	4816	4816	0	4992	4992	0	5371	5372	-1
11	4424	4424	0	4955	4953	2	4818	4816	2	4993	4992	1	5376	5372	4
12	4423	4424	-1	4952	4953	-1	4816	4816	0	4992	4992	0	5373	5372	1
13	4424	4424	0	4953	4953	0	4816	4816	0	4991	4992	-1	5372	5372	0
14	4424	4424	0	4952	4953	-1	4816	4816	0	4993	4992	1	5374	5372	2
15	4426	4424	2	4952	4953	-1	4815	4816	-1	4992	4992	0	5374	5372	2
16	4431	4424	7	4952	4953	-1	4815	4816	-1	4991	4992	-1	5374	5372	2
17	4428	4424	4	4952	4953	-1	4815	4816	-1	4991	4992	-1	5373	5372	1
18	4430	4424	6	4954	4953	1	4816	4816	0	4992	4992	0	5374	5372	2
19	4428	4424	4	4957	4953	4	4817	4816	1	4993	4992	1	5371	5372	-1
20	4424	4424	0	4955	4953	2	4815	4816	-1	4996	4992	4	5375	5372	3
21	4423	4424	-1	4952	4953	-1	4816	4816	0	4992	4992	0	5375	5372	3
22	4422	4424	-2	4952	4953	-1	4815	4816	-1	4991	4992	-1	5375	5372	3
23	4422	4424	-2	4957	4953	4	4816	4816	0	4992	4992	0	5375	5372	3
24	4425	4424	1	4956	4953	3	4817	4816	1	4992	4992	0	5375	5372	3
25	4424	4424	0	4952	4953	-1	4816	4816	0	4991	4992	-1	5375	5372	3
26	4425	4424	1	4952	4952	0	4816	4816	0	4992	4992	0	5376	5372	4
27	4429	4424	5	4953	4952	1	4818	4816	2	4992	4992	0	5375	5372	3

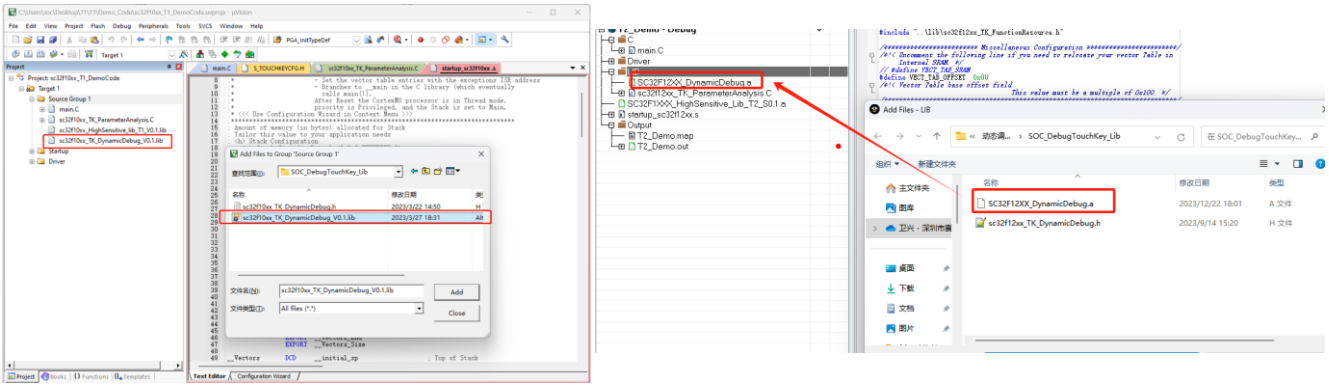
12. 动态调试注意事项
    - **由于动态调试和 printf 打印使用了芯片上的 UART（UART0 或者 SSI）资源，用户程序必须先屏蔽掉 UART（UART0 或者 SSI）部分程序，包括初始化、中断服务函数等，但不能将总中断 EA 关闭，用户程序不能操作和 UART（UART0 或者 SSI）相关的寄存器以及操作对应的管脚，烧录口上为 UART0 的型号使用动态调试和 printf 打印时，还使用了 Timer2 作为波特率发生器,所以 Timer2 也不能使用。**
    - **调试主界面勾选的通道必须与实际工程使用的通道一致。**
    - **动态调试占用了 43byte idata 和 501byte ROM 资源，请预留足够资源，保证动态调试程序运 行正常。**

4.5.2 ARM 高灵敏度动态调试步骤

将SOC_DebugTouchKey_Lib文件夹放置到工程的根目录下（左侧是 KEIL 工程，右侧是 IAR 工程）



1. 在用户工程中加入 SC32F1XXX_TK_DynamicDebug_Vx.x.LIB
   （左侧是 KEIL 工程，右侧是 IAR 工程）



2. 在 main.c 中 include 头文件

```
#ifdef __TOUCHKEY_DEBUG__
#include "..\SOC_DebugTouchKey_Lib\sc32f10xx_TK_DynamicDebug.h"
#endif
```

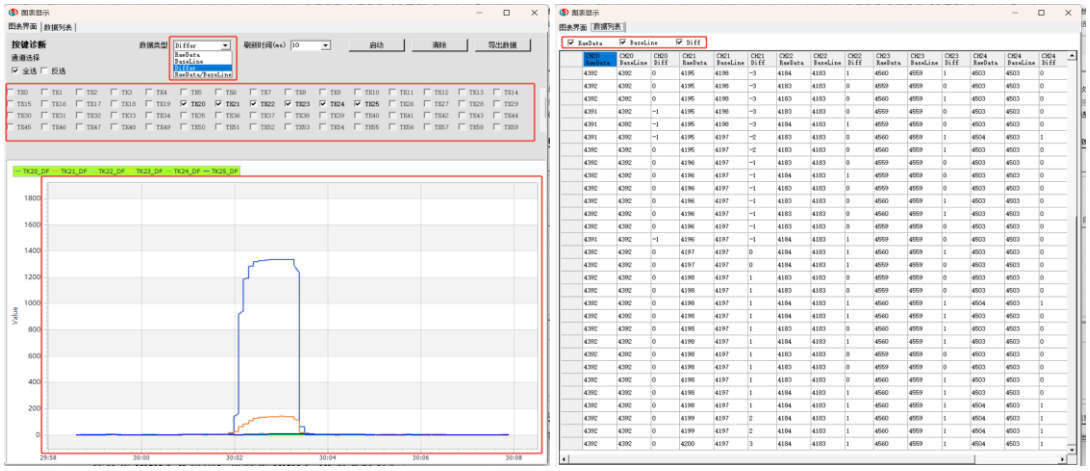
3. 在 main 函数中调用 TK_DynamiDebug_Init 进行初始化。编译成功后烧录到芯片内
   TK_DynamiDebug_Init(n);
   n 为当前使用的芯片烧录口 uart 资源时钟总线的运行时钟频率，如果是 32M，则填写 32000000

```
int main(void)
{
    SystemClock_Config();
    Sys_Init();

    TK_DynamicDebug_Init(32000000);

    //触控按键初始化
    TK_Init();
```

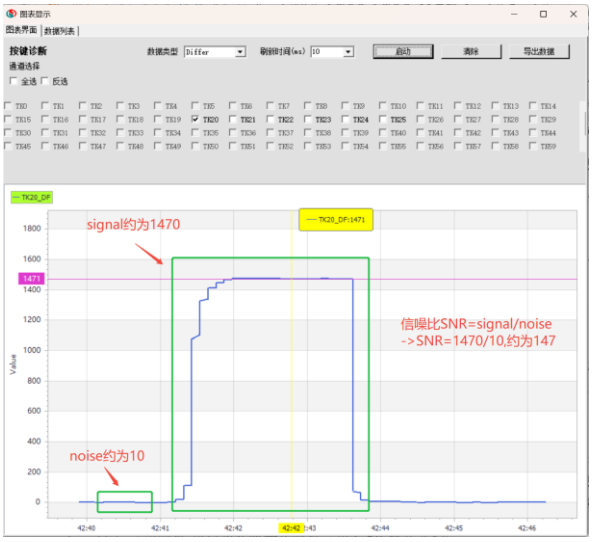
4. 打开 Touch Key Tool Menu,选择高灵敏度触控，芯片型号选择与实际芯片对应的型号，调试模式选择动态调试，勾选 TK 通道（**必须与项目实际使用的通道一致**）

5. 点击确定按钮，然后点击下方“动态调试”按钮

6. 在动态调试界面内，可以通过选择“数据类型”查看想要查看的数据，通过“通道选择”勾选要查看的通道，在图表显示内可以实时看到数据的情况，点击“数据列表”可以查看图表数据

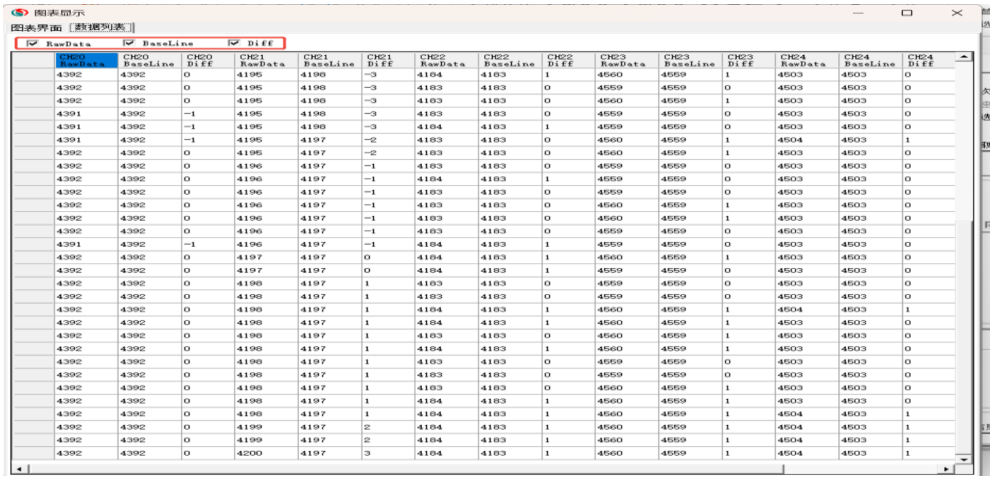


注意：动态调试观测数据时，需要注意信噪比 SNR 是否符合使用条件。
建议是 SNR > 5 可用，推荐是 SNR > 10 效果较好。

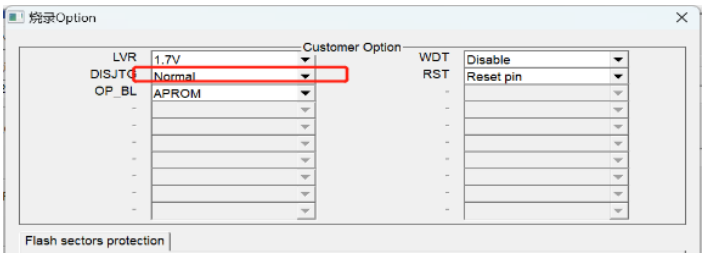


SNR 计算方式：动态调试观测中
Noise 噪声幅度为：静置状态下，没有手按下时 Differ 的最大值减去最小值的差值 Differ0
Signal 信号幅度为：手指按下时 Differ 的平均值 Differ1(图示绿色框内所示)
SNR 计算方式：SNR = Signal/Noise=Differ1/Differ0,如上图计算 SNR=147 左右

7. 点击“导出数据”可以将实时采集数据导出 CSV 格式文档



8. 动态调试注意事项
   - **由于动态调试库使用了烧录口上的 UART 资源，用户程序必须先屏蔽掉 UART 部分程序，包括初始化、中断服务函数等，但不能将总中断 EA 关闭，用户程序不能操作和 UART 相关的寄存器以及操作对应的管脚。**
   - **调试主界面勾选的通道必须与实际工程使用的通道一致。**
   - **使用动态调试库时芯片主频不能小于 2M。**
   - **Option 设置 DISJTG 为 Normal（如下图所示）。**



5 触摸按键 MCU PCB 设计要点

5.1LAYOUT 整体布局要求

5.1.1 触摸按键的形状及材料选择

图 5.1 为自电容方式的手指触摸按键的推荐形状。

材料选择：通常为 PCB 铜箔、金属片、平顶弹簧、导电棉、导电橡胶、ITO 玻璃层等

按键形状：PCB 铜箔推荐圆形，其次方形。**当用弹簧作为感应盘，应保证弹簧压缩量为 3~10mm。**应当避免尖角形状按键（小于 90 度），其尖角部分会集中电场，降低抗干扰能力，不建议使用。

按键尺寸：推荐 10.0x10.0~15.0x15.0mm 之间。

注：如果在电极中配置 LED 或其他器件，请注意开孔尽可能的小，以确保在触摸按键时获取足够的信噪比。

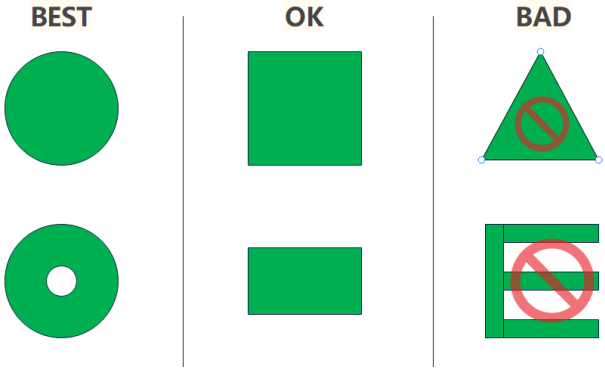


图 5.1 自电容方式的推荐触摸按键设计

5.1.2 触摸按键及其布线

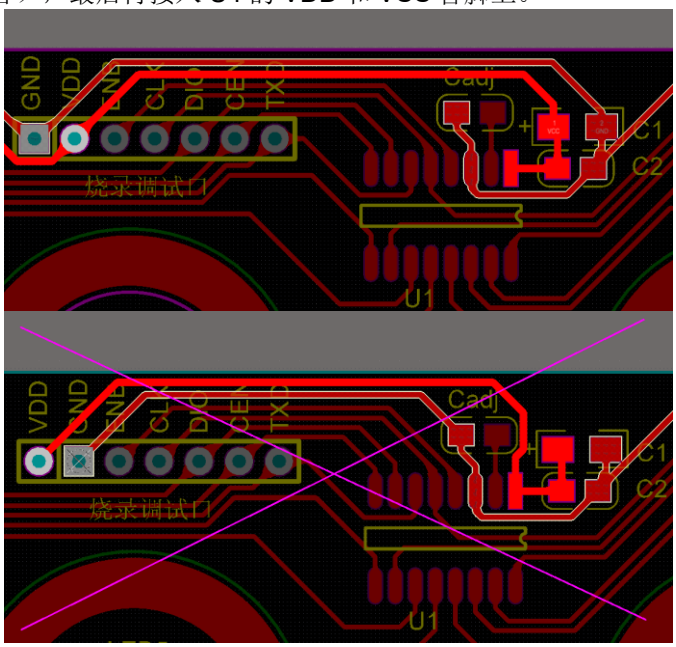
1. 布局时尽量将触摸 MCU 放置在多个触控按键的中心位置，尽量减短触摸芯片引脚和触摸感应按键焊盘之间的走线长度，最大走线长度为 200 mm，最大走线宽度为 10mil。
2. 相邻触摸按键间距应超过 10mm，以避免相邻按键影响。走线禁止布置在任何触摸感应按键下方，除非走线是和该触摸按键相连的。
3. 触摸走线和其他电器器件，金属件的安全间距应超过 10mm。
4. 相邻触摸走线，触摸走线和时钟信号或者通信线之间的安全间距应超过 2mm。若触摸走线必须与之交叉，确保走线应呈直角。
5. 触摸通道的匹配电阻推荐阻值为 510R（最大值应小于 5.6K 欧姆），应尽量靠近触摸芯片对应管脚放置，推荐距离小于 10mm，以降低射频干扰并提供 ESD 保护。
6. 触摸按键若是 PCB 铜箔式感应按键，应敷阻焊油、不露铜。尽量少使用过孔，以减少寄生电容。
7. 基准电容 Cadj 接在 MCU 的 CMOD 与 VSS 管脚之间，作为触控感应电路的充放电电容，是实现触控功能的重要器件，它保障了触控电路的正常工作。Cadj 容值：**推荐 103 电容，但在低功耗应用下，推荐 102 电容。**建议使用温度系数小精度高的电容，以免造成灵敏度不一致或随温度变化而变化。一般插件电容建议 10%精度涤纶电容，如需贴片电容则建议使用 10%或更高精度的 NPO 材质电容或 X7R 材质电容。
   同时 Cadj 电容需要尽量靠近芯片管脚，推荐距离小于 10mm。
8. **当有多颗芯片在同一个 PCB 上，同时不同芯片的 TK 通道 PAD/弹簧或者走线距离过近（≤10mm），优先做法是将距离过近的 TK 通道修改为由同一颗芯片控制。如果不能修改走线，则需要针对相邻芯片其中一颗芯片开启抗干扰设置 1:12bit，另外一颗不开启（在进行触摸上位机静态调试操作时请留意设置）。**
9. **为避免严重的噪声耦合，触控走线区域须远离高频干扰源（如外接晶振、Wi-Fi 模块）。严禁触控扫描频率设置为周边高频信号的整数倍或分频，以免系统因信噪比恶化而产生误触或不稳定。**
   当触摸芯片使用外接高频晶振时，也须规避触控扫描频率与高频晶振信号的倍频冲突，以防止引入强噪声干扰，确保系统稳定性。
   可使用软件版本 V3.0.0 及以上的触控上位机进行调试，选择正确的外部晶振频率后，会自动错开倍频影响，生成参数。

5.1.3 电源电路走线

1. 电源线和地线应先经过电容滤波（电解电容+104 瓷片电容）之后再分别接入 MCU 的 VDD 和 VSS 管脚，也可将电解电容换为钽电容，容值不小于 10uF；
2. 104 电容布局时应紧靠 MCU 的 VDD 及 VSS 引脚放置，距离应小于 10mm；
3. 功率负载部分与 MCU 控制部分的电源分立，功率负载部分应在 104 电容前取电，具体功率负载部分取电操作可**参见“5.6.3 方案设计要点”章节第 4 点**使用显示驱动芯片驱动 LED 的场景。
4. 电源纹波应控制小于 100mv，最大不超过 200mv。

VDD 和 VSS 的布线方式可参考下图：

U1 为触控 IC，两根高亮的走线一个连到 U1 的 VDD，一个连到 U1 的 VSS，这两根线先经过 C1（10uF 钽电容）,再经过 C2（104 贴片电容），最后再接入 U1 的 VDD 和 VSS 管脚上。

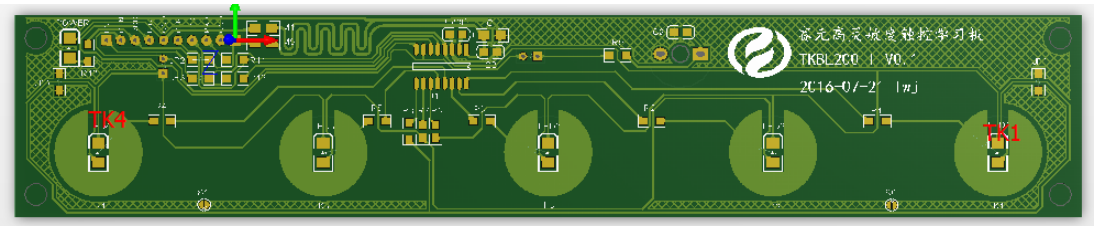


5.1.4 覆盖面板选择

1. **覆盖面板介电常数应介于 2.0-8.0 之间，且不能为导电材料，也不能在覆盖层上使用包含金属颗粒的油漆。**
2. PCB 铜箔式感应按键要求感应盘铜箔与触摸面板表面的垂直距离不大于 3mm，否则触摸灵敏度降低。**距离大于 3mm 的项目请参见“5.6 花架隔空触控感应按键设计”章节。**
3. 覆盖层的厚度选择：非玻璃材料最大厚度应小于 5mm，玻璃材料最大厚度应小于 8mm。
4. 若是弹簧式感应按键，尽量保证弹簧按键到面板的距离一致、弹簧顶端与触摸面板之间尽量不要有缝隙。

5.2敷铜要求

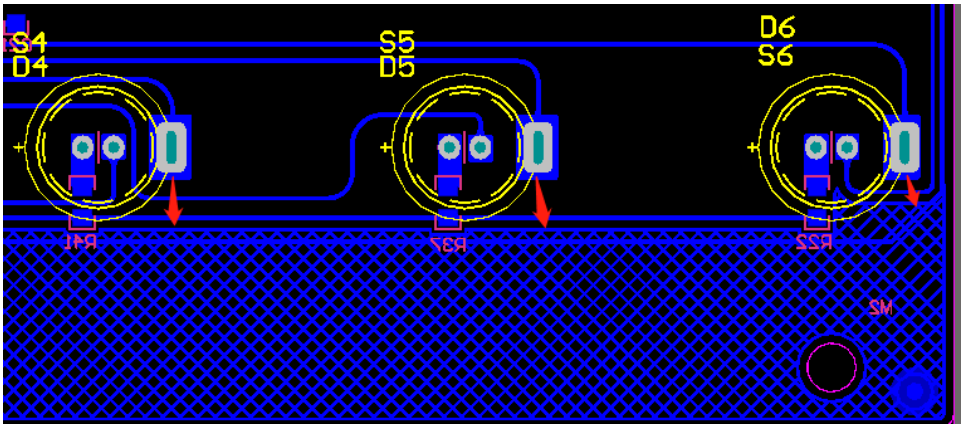
1. GND 敷铜可增强 PCBA 的抗干扰性能，建议使用网格式敷铜，铜皮有效面积需小于 40%；(接近感应应用特殊，Layout 相关请至指定章节 5.5 接近感应感应按键设计进行了解)
2. GND 敷铜的网络应从滤波电容 104 之后的 GND 上引出，这样可以最大限度的屏蔽干扰信号；
3. 感应盘四周其它触控网络最少的 TK 通道，其周围应有 GND 敷铜，且 GND 敷铜与该 TK 通道的感应盘或按键的投影面边沿处的距离不小于 3mm，推荐距离 3mm；花架隔空应用应保持在 5mm 以上。
4. 如下图：最左端 TK4 周围仅有右侧的 1 个 TK 通道，TK4 周围触控网络数量最少，根据第 3 点要求需对 TK4 的感应盘周围进行 GND 敷铜处理。右侧的 TK1 周围只有左侧有一个 TK 通道，同理也需做相同处理。



在感应盘周围的 GND 网络敷铜时要注意：

   1) 避免 GND 网络的铜皮形成闭合环路；
   2) GND 的敷铜仅限于感应盘周围，所有 TK 通道的走线附近不要敷铜，否则会降低触控按键的灵敏度。

5. 除第 3 条情况外，其它触控网络应尽量远离 GND 网络；
6. 如果是双面板，要避免在 TK 通道的走线和感应盘背面进行 GND 网络的敷铜，以免影响触控灵敏度。
7. 如需引出调试接口，SCK 和 SDA 连线到接线端口的引线间距尽可能的保证在两倍线宽以上；不要走平行线，如无法避免，请在两线之间加地线隔离，以确保使用 Touch Key Tool 进行调试时，采集数据正常工作。如采集数据时仍出现无数据上传，可以在 MCU 端 SCK 对地接 101 电容（尽量靠近 IC 管脚）。
8. 每个 TK 通道上串接电阻到 PAD，尽量不在触控走线及 PAD 附近敷铜（减少对地电容），布局及走线可参考下图。触控 PAD 尽量与敷铜间保持一定的间距（>=3mm 以上，花架隔空应用应保持在 5mm 以上）



5.3面板和触摸按键存在间隙的处理

图 5.2 显示了面板和触摸键电极之间存在间隙时的处理方法

1. 当空气间隙加面板的总厚度小于 4mm 时，使用赛元隔空触摸应用能检测到触摸，PCB 设计和常规一致。
2. 当空气间隙超过 3mm，小于 6mm 时，需要按照花架隔空触控的方式进行，详细介绍请参见“花架隔空触控感应按键设计”章节。
3. 当空气间隙过大，超过 6mm 时，已经超过隔空触摸应用范围。此时需要将触摸按键延长到面板上，连接材料常规使用的是弹簧，导电橡胶等导电材料。

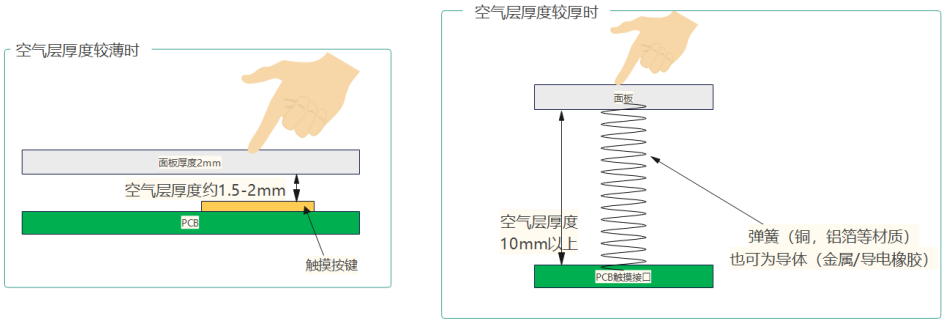


图 5.2 自电容方式的不同空气层厚度应用应对示例

5.4滑轮滑条感应按键设计

图 5.3 显示的是推荐的线性滑条模型 ，而表格显示的是推荐的线性滑条尺寸。

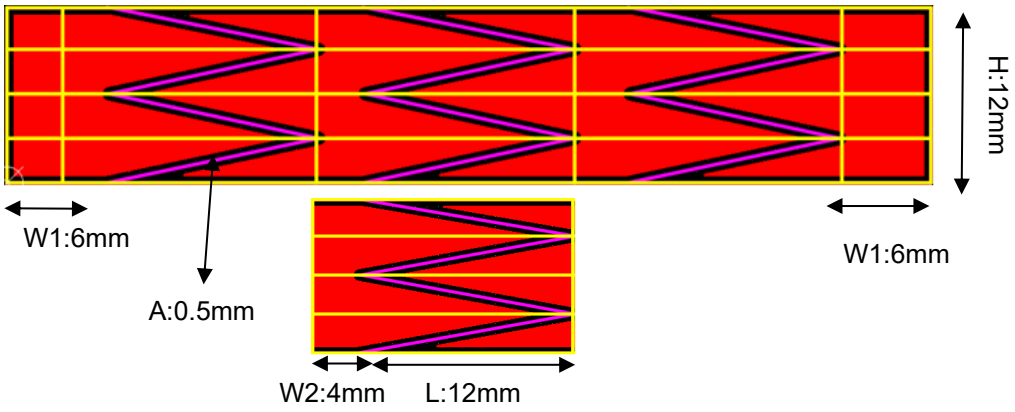


图 5.3 推荐的线性滑条模型

参数	最小值	最大值	建议
滑条两端长度(W1)	5mm	7mm	6mm
单个滑条段中心长度(W2)	3mm	5mm	4mm
不同滑条段的间隙(A)	0.5mm	2mm	0.5mm
滑条段的高度(H)	7mm	15mm	12mm
单个滑条段的长度(L)	9mm	15mm	12mm

推荐的线性滑条尺寸

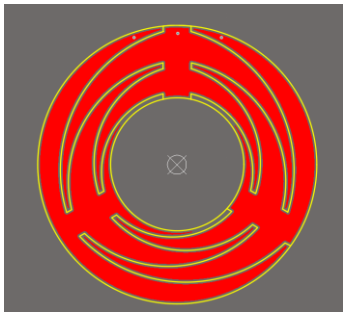
滑轮滑条感应 PAD 在 PCB 板上采用交互锯齿现状排布，为了保证效果，感应盘的宽度应小于一个手指的宽度(7mm~15mm)且感应盘之间间隙应该在 0.5mm 以上。布线时原则上感应盘底部不能有任何器件且禁止在感应盘周围平行走线，尤其是高频信号线。用于滑轮滑条 TK 走线不能穿过其他滑轮滑条感应盘区域。

5.4.1 滑条感应盘 PCB 设计

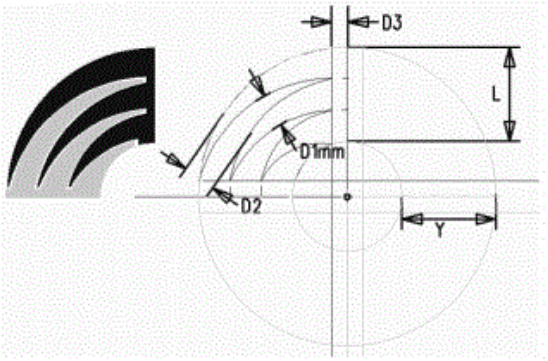
滑条总长度小于 50mm 建议使用 3~4TK 通道来设计。滑条总长度增加 10mm~20mm 建议增加一个 TK 通道，每个锯齿相互咬合，但是前后锯齿之间要留有过渡区域，如图锯齿长度 12mm 而过渡区域取锯齿长度的 3 分之一也就是 4mm，在滑条的最前和最后要留一个较大的过渡区域，一般可取锯齿长度的 2 分之一为 6mm，这两段留过渡区域目的是能更好的滑出最大值和最小值。

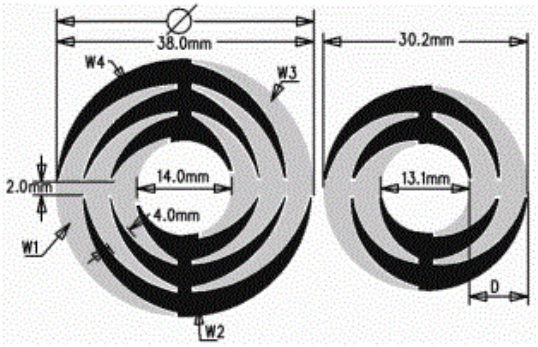
5.4.2 滑轮感应盘 PCB 设计

滑轮感应盘是环状的交互锯齿且每个感应盘形状保持对称， PCB 设计时每个感应盘中心过度区要大于2mm，所用 TK 通道数最小为 3。



设计方法（如图,实例为 4 通道滑轮）:首先根据滑轮的结构尺寸在 Drill Drawing 层画好滑轮的内外圆，然后把环 L 三等分后画出滑轮一个通道二分之一的边框线，其中 D1=D2=4mm，D1、D2 与环的宽度是 Y 的比例关系：D/Y=1/3，D3=1mm。在完成边框线后，将边框线拷贝粘贴旋转 3 份就得到均匀对称的滑轮，最后填充铜皮即可。

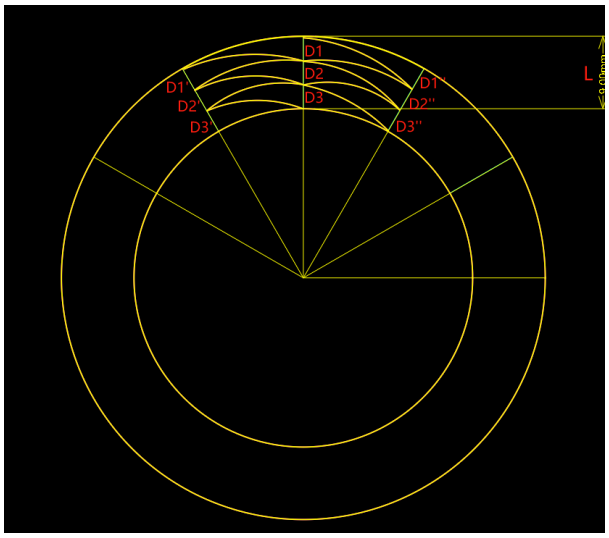


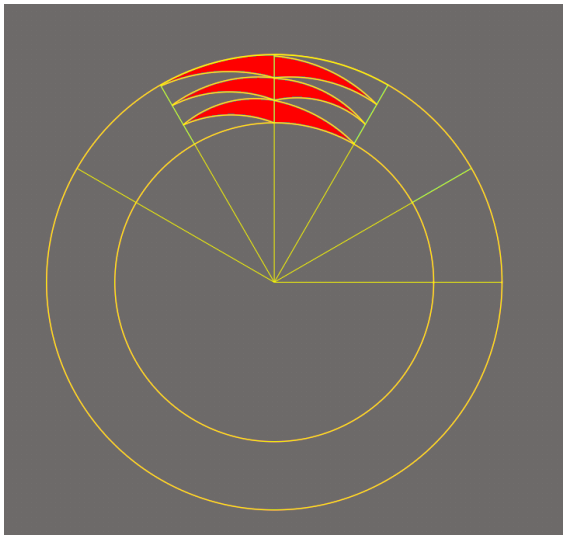


5.4.3 多 TK 通道滑轮感应盘 PCB 设计

设计方法（如图,实例为 12 通道组成滑轮）:首先根据滑轮的结构尺寸在 Drill Drawing 层画好滑轮的内外圆，然后均分 12 通道所占用的角度区域，即 360/12=30 度，因相邻触控通道的 PAD 需要锯齿形咬合组成滑轮，故单个通道所占用的范围为 60 度。如下图(圈内的均分 30 度的细线为滑轮 PAD 做参考辅助线)，再把相邻的 L 长度线三等分后(分别为 D1/D2/D3，D1'/D2'/D3'，D1''/D2''/D3'')，根据下图示意图依次连接 3 等分点绘制单个通道滑轮 PAD 边框线，其中 D1，D2，D3 与 L 的宽度的比例关系：D/L=1/3。在完成边框线后,剩余其他通道依此类推绘制就得到均匀对称的滑轮，最后填充铜皮即可。

左侧是单个滑轮 pad 绘制参考规则示意图 ，右侧是按照规则绘制填充铜皮后，单个滑轮 pad 形状示意图。剩余的 PAD 依次绘制边框线填充铜皮即可。



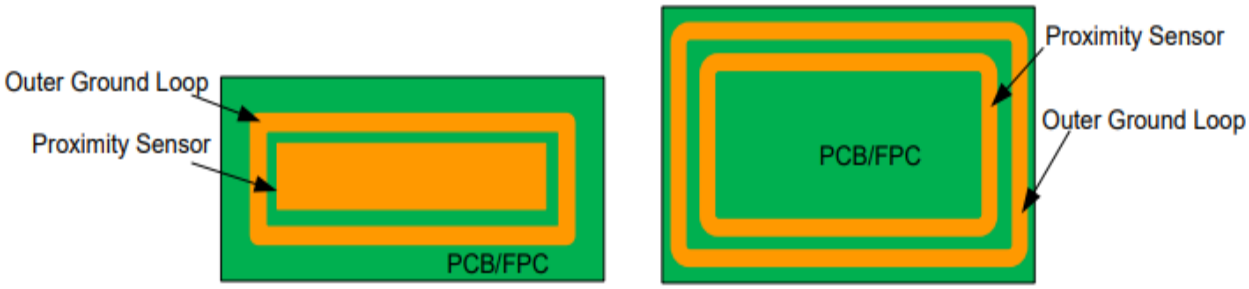


5.5接近感应感应按键设计

5.5.1 实现电容式接近感应的布局设计

实际应用中实现电容式接近感应是一个挑战，因为接近感应距离取决于各种因素，如传感器布局、噪声的存在、以及浮动或接地的导电物体等。

接近感应结构取决于应用和所需的接近范围。在近距离接近感应(最大接近距离为 3CM)的控制应用中，为了得到该接近距离，需要有小面积且实心填充形状的传感器，对于要求更大接近距离的应用，则要使用器件周围的大型传感器回路，以便得到更大的接近距离。



PCB/FPC 上的接近传感器构建

注：图示中 Outer Ground Loop 接地回路，是用以提高对 ESD 事件的抗性，没有 ESD 需求请勿添加。添加 Outer Ground Loop 接地回路后，需要实测权衡接近范围、抗噪性以及 ESD 性能。

常见的接近感应传感器构造：

（1） PCB 或 FPC 上的铜制走线：圆形或者方形等规则环形 PCB 或 FPC 走线可以作为接近传感器使用。

5.5.2 电容式接近感应线圈规格选择

接近感应传感器大小取决于各种因素，比如：所需接近感应距离、噪声源的存在以及悬浮或接地的导电物体。噪声源和悬浮或接地的导电物体会降低信噪比（SNR：信号强度与噪声强度的比值）和接近感应距离。

初步赛元给出的距离与线圈规格大小的关系如下：

假设接近感应线圈是规则的圆形回路（直径 D1）或方形线圈回路（对角线距离 D2）

接近感应约为 D1/D2 的 1.5 倍感应距离。

但由于终端系统的复杂环境，同样的传感器大小也可能会获得不同的接近感应距离。

建议使用与所需接近感应距离相接近的最小回路直径（在圆形回路的情况下）或对角（在方形回路的情况下）开始进行环境实测。如果未能达到需要的接近感应距离，建议逐渐加大传感器回路直径或对角，直到获得所需的接近感应距离为止。

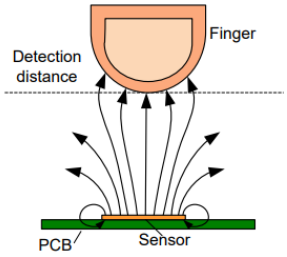
5.5.3 电容式接近感应 Layout 设计要点

类别	详细信息	建议/备注
接近感应线圈	接近感应线圈形状	•在 PCB/FPC 上的空心圆形或矩形（弯曲边缘） 线圈铜皮回路（感应距离需求大于 3cm） • 实心填充的圆形或矩形（感应距离需求小于 3cm）
	接近感应线圈走线宽度	1.5mm~5mm（推荐 2-3mm）
	接近感应回路与接地回路的间隙 **（有 ESD 需求需要注意，没有则忽视）**	1mm~2mm
	接地回路走线宽度 **（有 ESD 需求需要注意，没有则忽视）**	1.5mm
	接近传感器的回路直径或对角长度	根据所需感应距离约为线圈回路直径或对角长度的 1.5 倍自行换算，适当增加线圈大小留有余量
布局和走线	走线宽度	小于 0.25mm(10mil)

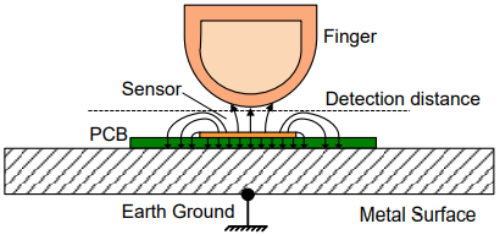
	走线布局	尽量将走线布在接近感应线圈侧面（即推荐线圈内不走线/无其他器件时，感应距离为最佳，如果在线圈内存在走线或器件，将会降低距离，具体影响程度需根据项目环境实测）。如将非 TK 走线穿过 TK 走线，则应确保其垂直相交，或者和 TK 走线间隔控制在 2mm 以上**(任何时钟、数据或者周期信号走线都不能与接近感应线圈的信号走线相邻平行布设。这些信号线应当尽可能与接近感应线圈的信号走线垂直或者布设在 PCB 的其他区域。如果时钟、数据或任何周期信号走线确实需要与传感器的信号走线平行布设，它们应当被布设在不同的层且不能重叠，而其当尽可能地缩短信号走线平行部分长度)**
	接近感应线圈信号走线过孔及连接位置	尽量减少走线长度，过孔应在接近线圈边缘，接近感应信号走线以垂直方式连接接近感应线圈
	串联电阻放置	建议 510 欧姆，为实现噪声抑制，应将串联电阻置于靠近芯片 IC 对应 TK 管脚的位置，推荐距离小于 10mm。
	地/屏蔽覆铜	不建议敷地铜皮，目的为了减小寄生电容 CP，提高灵敏度
覆盖层	覆盖层厚度和材料	覆盖层厚度和材料需要根据 PCB 布局和应用环境做具体评估，需要使用非导体材料（如玻璃、ABS 塑料等等）。

5.5.4 电容式接近感应应用环境注意

接近感应线圈附近禁止放置金属物体或者近地放置，否则会降低灵敏度并降低感应距离。
因为金属或者地会捕捉部分电场，增加寄生电容 CP，从而降低手指所产生的电容，缩短感应距离。



单个传感器的传播电场



单个传感器的传播电场（存在地或者金属）

5.6花架隔空触控感应按键设计

5.6.1 概要

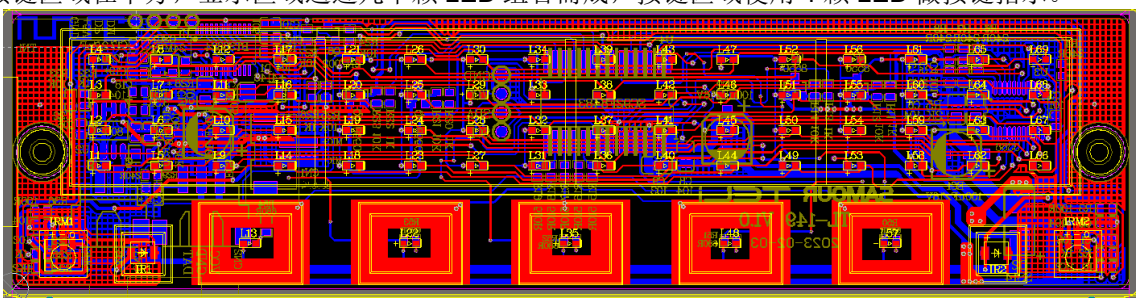
花架隔空触控感应是指在 PCBA 上画置 PAD，通过设计花架模具使模具与 PAD 紧贴，以填充触控感应 PAD 与面板之间的空气间隙。

赛元触控 MCU 内置高灵敏触控电路，可以检测到 0.1pF 级别的电容变化，但是由于空气的介电常数很小，所以为了防止面板与触控感应 PAD 间的空气间隙过大（超过 2mm），需要采用花架贴合触控感应 PAD 的方式来减小空气间隙，从而实现高达 6mm 的隔空距离，为 LED 灯的散光提供了足够的高度。

5.6.2 适用范围

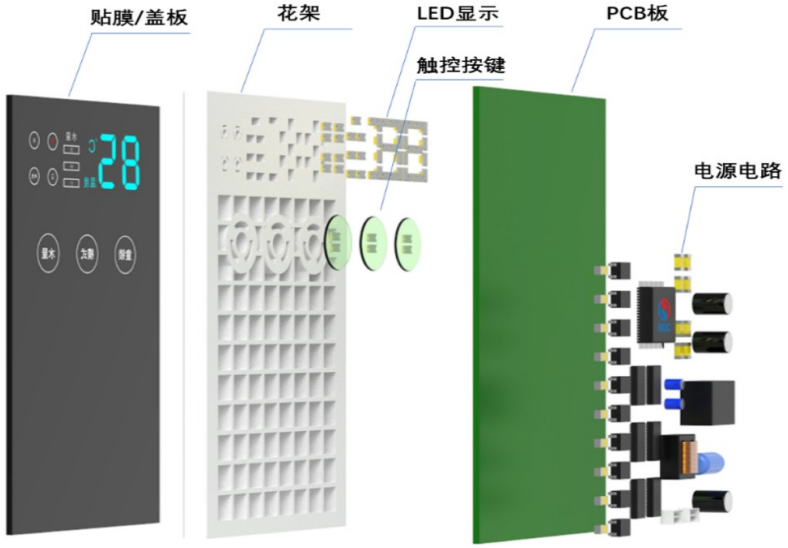
由于花架隔空触控应用同比弹簧触控应用变化量小很多，所以花架隔空触控应用会有一定的限制，需要满足以下基本条件才能使用花架隔空触控应用。

- 触控面板厚度为 6mm 以内玻璃面板，或者 4mm 以内亚克力/PP/ABS 等塑料材质面板；
- 触控感应 PAD 到触控面板下表面的距离不超过 6mm；
- 按键个数不超过 12 个；
- 按键区域的 LED 个数不能超过 2 颗，如果超过 2 颗 LED 则需要适当减少按键个数，确保按键个数不超过 6 个；
- 按键区域与主要显示区域分开，以确保按键区域不会有过多的显示变化，如下图所示，显示区域在上方，按键区域在下方，显示区域通过几十颗 LED 组合而成，按键区域使用 1 颗 LED 做按键指示。



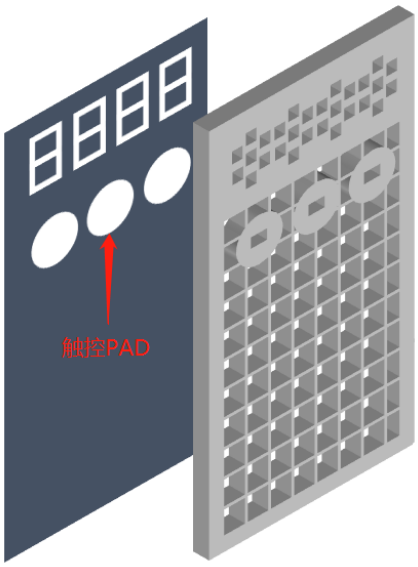
5.6.3 方案设计要点

1. 体积较大的电源类元器件置于PCB的背面，PCB正面放置LED显示和触控电路，可以有效降低PCB板与盖板之间的厚度，如下图所示，膜贴可以定制各种图案，结合不同颜色的 LED 灯珠，可以做到极好的显示效果，且成本低廉；



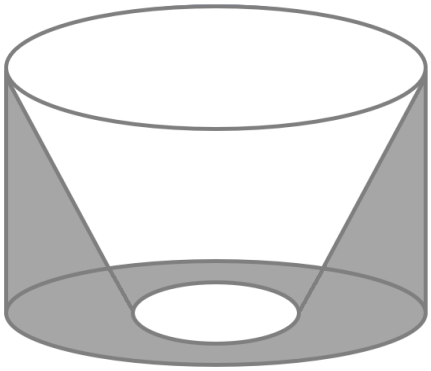
2. 使用成本更低，排列更灵活的 LED 灯组合替代数码显示屏，实现自定义的显示功能；

3. 触控按键处花架的设计需要使花架尽可能多的贴合触控感应 PAD，如下图所示，触控感应 PAD 为圆形，花架对应的按键区域仅留出 LED 显示挖孔，以保证花架最大程度的与触控感应 PAD 接触，减小触控面板与触控感应 PAD 直接的空气填充；



PCBA 花架

如果在实际项目中需要较大的按键显示区域，需要将按键区域对应的花架设计为喇叭状，如下图所示，即与触控感应 PAD 接触的花架下部设计为小开孔，以便于漏出 LED 灯，与触控面板接触的花架上部设计为大开孔，以便于 LED 照亮更大的面板区域，得到较大的显示区域，同时也确保了花架与触控感应 PAD 之间的接触面积；

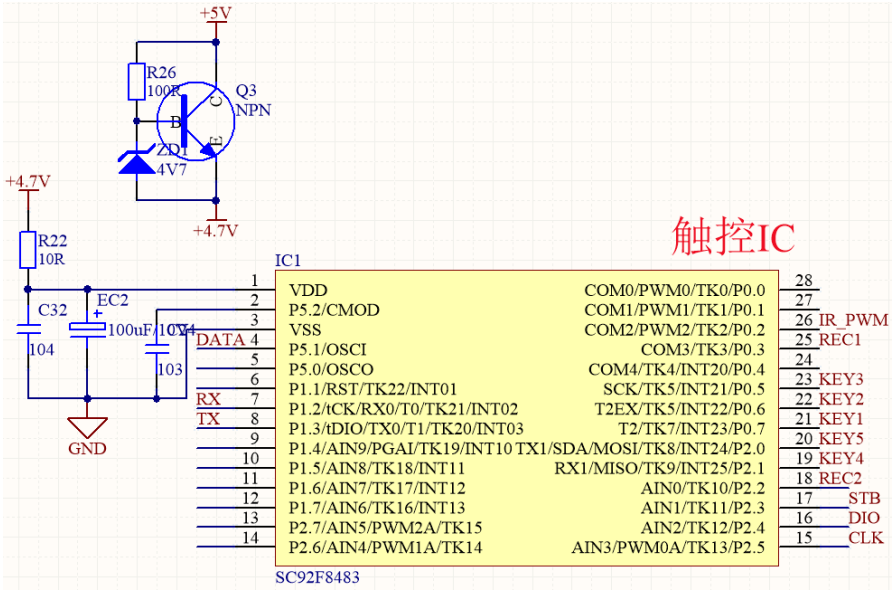


4. 由于隔空应用下触控感应量远低于常规弹簧应用，需要提升触控检测灵敏度以获取较大的变化量，灵敏度的提升会导致触控 MCU 对电源电压比较敏感，所以需要将电流消耗较大的设备电源与触控 MCU 的电源分开，如果使用触控 MCU 的引脚直接驱动电流消耗较大的设备（如，LED 等），会导

致触控 MCU 电源电压波动，提升花架隔空触控的实现难度，针对不同的 LED 驱动情况，推荐使用如下方案实现；

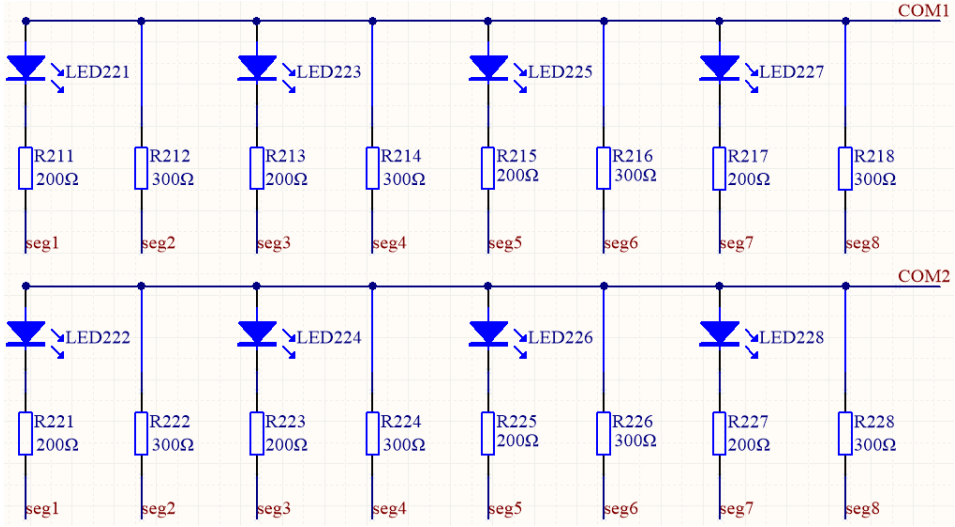
1) 使用显示驱动芯片驱动 LED 的场景

推荐使用恒流源显示驱动 IC（如 AiP3368/3358/3628 等）对 LED 进行驱动（不建议使用恒压源显示驱动芯片），恒流源驱动芯片与触控 MCU 之间通过稳压电路进行稳压，以确保在显示变化时 MCU 端的电压波动不超过 50mV，以避免显示变化对触控数据的影响；如下图触控 MCU 在三极管后级的 4.7V 处取电，并经过 100uF 以上的电解电容再接到触控芯片 VDD 处；其他器件，如显示驱动芯片、LED、红外管、蜂鸣器等器件在 5V 处取电；



2) 触控 MCU 直接驱动 LED 的场景

采用一颗 LED 多加一颗电阻进行电流补偿，使触控 MCU 在驱动 LED 亮灭时 MCU 端的总电流不变，从而保证 MCU 的电压稳定，如下图所示，每颗 LED 旁边都多了一颗电阻连接到 COM 口和 SEG 口，在进行显示时使 seg1 和 seg2，seg3 和 seg4，seg5 和 seg6，seg7 和 seg8 交替导通，比如当要点亮 LED221 时将 seg1 导通，同时将 seg2 关断，反之当要熄灭 LED221 时将 seg1 关断，同时将 seg2 导通，以确保总电流不变，其中电阻值可以根据实际情况选择合适的阻值，最终目的是使 LED 亮灭时触控 MCU 端的总电流保持不变，此方案同样适用于显示驱动芯片驱动 LED 的场景；



3) 对于需要调整显示亮度的场景

建议将需要调整亮度的 LED 平均分成 N 组（N 的取值根据需要的亮度决定），每段时间点亮其中一组，比如要实现 LED 的半亮显示，则将 LED 分成两组，前 10ms 点亮第 1 组同时熄灭第 2 组，后 10ms 点亮第 2 组同时熄灭第 1 组，这样既可以实现亮度调节，也可以保证在进行亮度调节时总电流不变，从而保证触控 MCU 的供电电压稳定；

4) 特定功能切换显示变化较大的场景

在某些场景下需要对 LED 状态进行全亮和全灭的切换，比如开关机时，需要进行全亮显示或者全灭显示，此时会使电源电压产生较大的波动，对于这类特定场景且已知的电压波动可以在电压产生波动后进行触控数据的基线更新，基线更新请调用 void SetNeedUpdateBaseline(void)函数，此函数被封装在触控库中，调用时需要在对应文件内进行外部声明，调用一次就会进行一次全部按键的基线更新，注意一定不能随意调用此函数，需在电压突变并重新稳定在某个状态后进行调用；比如在 LED 由全灭变为全亮后，电源电压会降低，所以需要在 LED 全亮后电源电压已经降低到稳定值后进行调用。

注：以上措施无法尽数，需要在实际项目中灵活组合使用，方案核心是确保功率器件在状态切换时尽可能的保证总电流不变，以确保触控 MCU 端的电压波动较小。

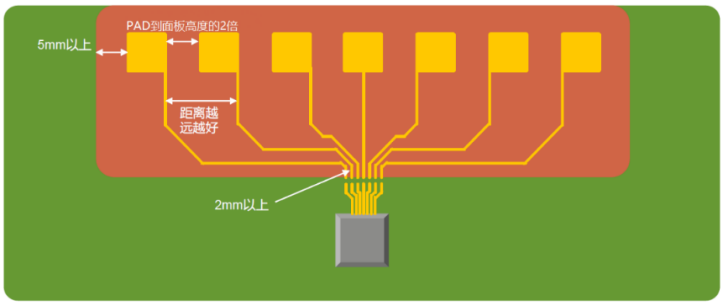
5. 按键尺寸形状和走线

1) 推荐使用圆形或方形按键，最理想尺寸是 12*12mm~18*18mm，注意按键不应有尖角部分，尖角部分具有天线效应；

2) 如果想在按键中放置 LED 或其他元器件，请注意开孔尽可能地小，以确保在触摸按键时获得足够的电容变化量，LED 或其他元器件的走线不能与触控按键走线平行，且走线应尽可能地细，以减小元器件工作对触控的影响；

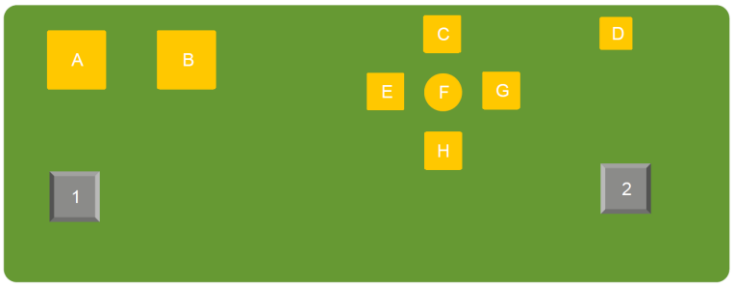
3) 触控按键与 MCU 引脚之间的布线要尽可能的短，触控按键走线之间的间距要尽可能的大，同时触控按键之间也应该保持足够的距离。因为，触控按键与 MCU 引脚之间的布线越长，越容易受到射频噪声的影响，而走线之间的距离太近，也会导致寄生电容的增加；

如下图是一个理想的触控按键布线示例，在示例中，触控周围没有放置其他引线，触控芯片放置在按键中间避免过长或过短的触控走线，此外，建议触控按键之间的距离保持在触控 PAD 至面板距离的 1~2 倍以上（包括花架厚度，面板厚度和空气厚度），以防止按键之间的串扰过大。



4) 触控按键的布线示例

如下图为一个操作面板的布线示例，其中，触控按键被设置在 A~H 的位置，MCU 被设置在“1”或“2”的位置。当 MCU 被分别设置在位置“1”或位置“2”时，触摸键的灵敏度会受到不同的影响。



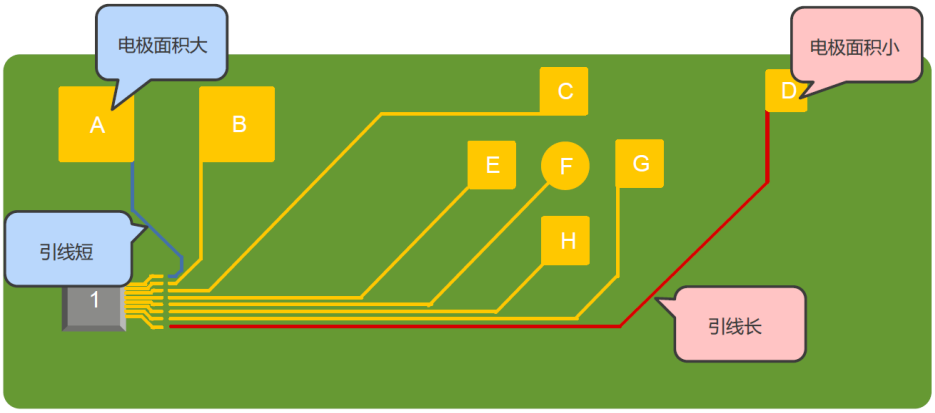
如下图当 MCU 被设置在位置“1”时，触控 PAD A 的两个正面条件（触控 PAD 面积大，触控走线短）重叠，触控 PAD D 的两个负面条件（触控 PAD 面积小，触控走线长）重叠。在这种情况下，触控 PAD A 的灵敏度会升高，触控 PAD D 的灵敏度会降低。

触控走线长，会带来以下 3 个风险：

• 如果触控走线长，引线会成为天线，触控 PAD 易受到射频噪声的影响，从而增加误操作的风险。

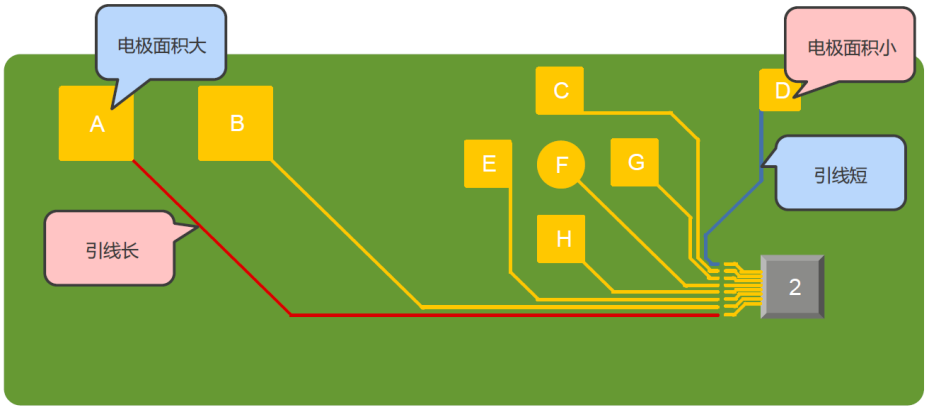
• 如果触控走线长，周围的引线和导体之间会产生更多的寄生电容，从而降低变化量。

• 在使用碳等具有高表面电阻的材料时，触控走线长会增加电阻值，从而降低变化量。



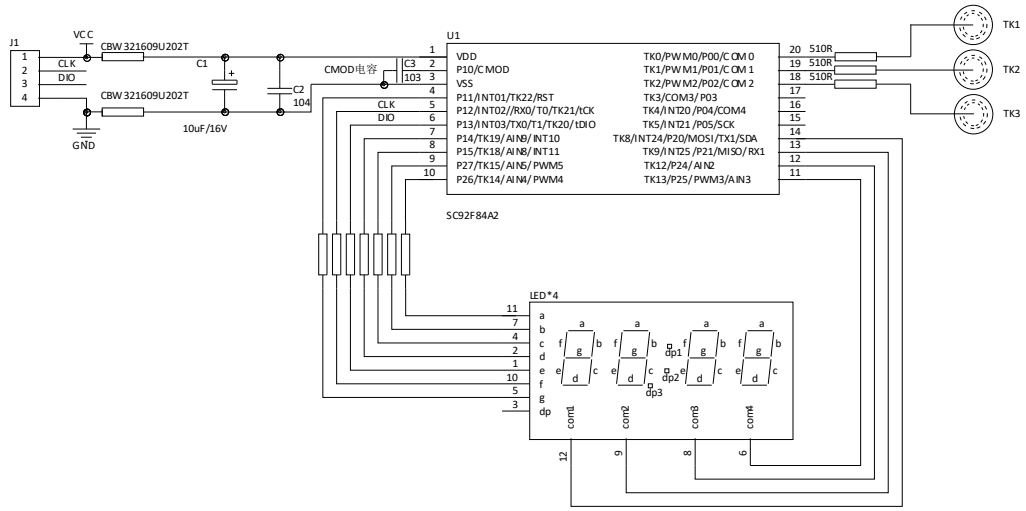
如下图当 MCU 被设置在位置“2”时，触控 PAD A 的正面条件（触控 PAD 面积大）和负面条件（触控走线长）相互抵消，触控 PAD D 的负面条件（触控 PAD 面积小）和正面条件（触控走线短）相互抵消。因此，和 MCU 在位置“1”时相比，触控 PAD A 的灵敏度会降低，而触控 PAD D 则能够抑制灵敏度的恶化。

所以，在设计触控布线时，要综合考虑各种因素，创造一个平衡的布局，从而避免触控 PAD 灵敏度的恶化。



附录

一、C51 应用参考原理图



说明：

1. Cadj 容值：

推荐 103 电容，精度 10%；

低功耗应用推荐 102 电容，精度 10%

2. TK 管脚上匹配电阻取值：

推荐值：510R

二、软件参考示例

1) 对于 TouchKey 与 LED 共用的项目，调用 TouchKeyRestart() 开始扫描按键 时，SOCAPI_TouchKeyStatus & 0X80 的标志还没有出现时，一定不要去做显示数据的事情；
2) 对于 TouchKey 与 LED 不共用的项目，则显示和扫描按键没有必要分开去做。

下面附程序说明

TouchKey 与 LED 共用的项目：

Main.c

```
void main()
{
    unsigned char result =0;
    SegOutState;      // 初始化 IO 口， 显示的 SEG,COM 脚
    ComOutState; ComAllClose; SegAllClose;
    TestIOPortOut;    //P17 作为测试 IO 口

    EA = 1;  //开总中断

    TouchKeyInit();   //重要步骤 1: 扫键的初始化函数
    InitialLcd();     //初始化显示部分

    while(1)
    {
        WDTCON |= 0x10;          //清 watchdog

        //重要步骤 2: 触摸键扫描一轮标志，是否调用 TouchKeyScan()一定要根据此标志位置起后
        if(SOCAPI_TouchKeyStatus & 0X80)
        {
            //重要步骤 3: 清除标//志位， 需要外部清除。
            SOCAPI_TouchKeyStatus &= 0X7F;
            exKeyValue = TouchKeyScan();     //重要步骤 4: 分析按键数据， 并返回结果出来

            //// 如果有按键，则更新显示缓冲区数据
```

```
                    {
                    UpdateLcdBufFunc(); //更新显示数据

                    ////如果没有显示，直接对 IO 口操作用示波器看结果
                    TESTIO=~TESTIO;
                    }

                    //重要步骤 4: bSensorCycleDone 标志位置起后,内部会停止检测按键，此时留出时间片显示数据
                    {
                            DisplayData();  //扫键完成后，立即启动显示
                            OpenPwm();      //启动显示用的 PWM
                    }

            }

        }
}

Display.c
void  DisplayData(void)
{
    ComAllClose; SegAllClose;

    if(isLcdComflag == 0)     //显示 COM0  数据
    {
            SetSegData(gIsLcdDataBuf[0]);
            seg8 = gIsLcdDataBuf[4] & 0x01;
            seg9 = gIsLcdDataBuf[4] & 0x02; COM0 = 0;

            isLcdComflag = 1;
    }
    else if(isLcdComflag ==1)          //显示 COM1 数据
    {
            SetSegData(gIsLcdDataBuf[1]);
            seg8 = gIsLcdDataBuf[5] & 0x01;
            seg9 = gIsLcdDataBuf[5] & 0x02;
            COM1 = 0;

            isLcdComflag = 2;
    }
    else if(isLcdComflag ==2)          //显示 COM2 数据
    {
            SetSegData(gIsLcdDataBuf[2]);
            seg8 = gIsLcdDataBuf[6] & 0x01;
            seg9 = gIsLcdDataBuf[6] & 0x02;
            COM2 = 0;
            isLcdComflag = 3;
    }
    else if(isLcdComflag ==3)          //显示 COM3 数据
    {
            SetSegData(gIsLcdDataBuf[3]);
            seg8 = gIsLcdDataBuf[7] & 0x01;
            seg9 = gIsLcdDataBuf[7] & 0x02;
            COM3 = 0;
            isLcdComflag = 4;
    }
    else
    {
```

```
            if(isLcdComflag == 4)    //COM 都显示完毕，准备启动按键扫描
            {

                ClosePwm();    //关闭显示用到的 PWM。
                isLcdComflag = 0;

                //重要步骤 5: 待所有的显示完毕后，需要重新调用 TouchKeyRestart(); 启动按键扫描，
                否则//不会扫描按键， 同时需要关闭与 TK 共用的一些 IO 口，保持检测按键的一致性。
                ComAllClose;
                SegAllClose;
                TouchKeyRestart();
            }

        }
}
```

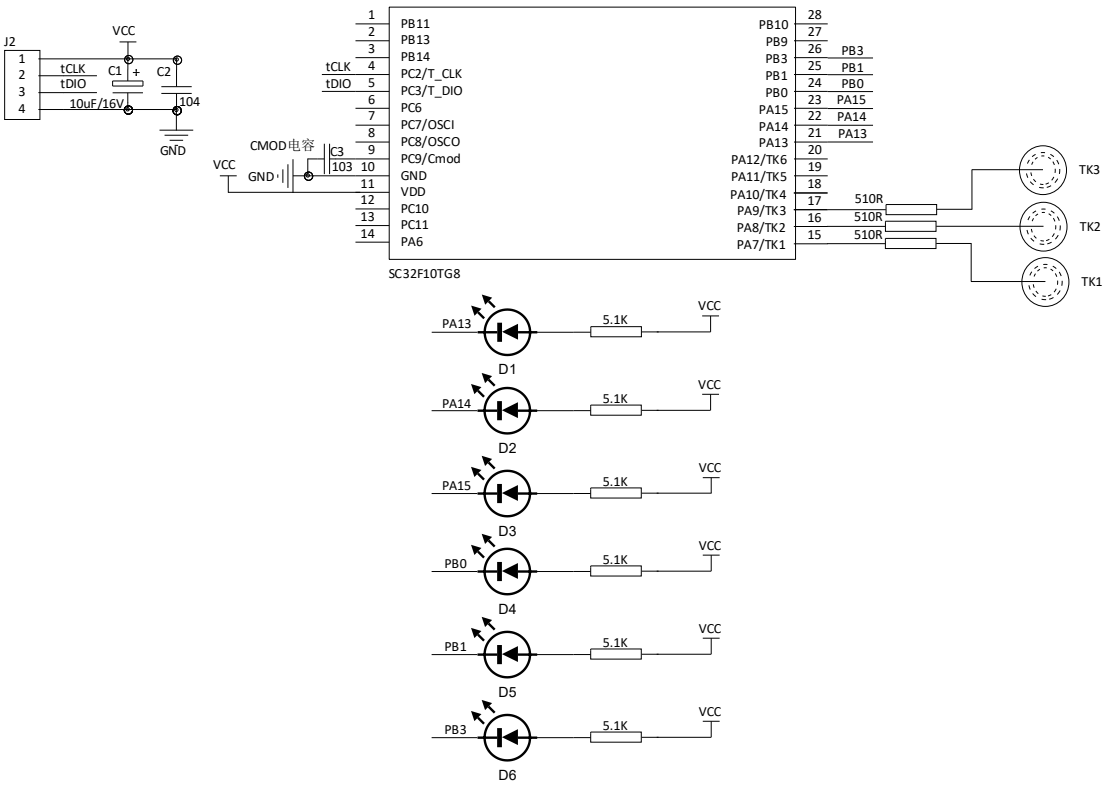
TouchKey 与 LED 不共用的项目

Main.c

```
void main()
{
    unsigned char result =0;
    SegOutState;    // 初始化 IO 口， 显示的 SEG,COM 脚
    ComOutState; ComAllClose; SegAllClose;
    TestIOPortOut;  //P17 作为测试 IO 口

    EA = 1; //开总中断

    TouchKeyInit();  //重要步骤 1: 扫键的初始化函数
    InitialLcd();       //初始化显示部分

    while(1)
    {
            WDTCON |= 0x10;        //清 watchdog if(TimerFlag_1ms==1)
            {
                TimerFlag_1ms=0;
                if(SOCAPI_TouchKeyStatus&0x80)        //重要步骤 2: 触摸键扫描一轮标志，是否调用
                                                      TouchKeyScan()一定要 根据此标志位置起后
                {
                    SOCAPI_TouchKeyStatus &=0x7f;  //重要步骤 3: 清除标志位， 需要外部清除。
                    exKeyValueFlag = TouchKeyScan();
                    ChangeTouchKeyvalue();
                    UpdateLcdBufFunc();  //更新显示数据
                    TouchKeyRestart();       //启动下一轮转换 TimerFlag_1ms=0;

                }
                BuzzerWork();
                //******************蜂鸣器驱动函数**********************
                if(++Timercount>=10)
                {
                    Timercount = 0;
                    DataUpdateCount++;
                }
                UpdateDisplay();          //******************处理显示内容**********************
            }
        }
```

```
}

void  DisplayData(void)
{
    ComAllClose;
    if(isLcdComflag == 0)    //显示 COM0 数据
    {
            LedSetSegData(LcdDisplayBuf[gIsLedDataBuf[0]]); COM0 = 0;
            isLcdComflag = 1;
    }
    else if(isLcdComflag ==1)          //显示 COM1 数据
    {
            LedSetSegData(LcdDisplayBuf[gIsLedDataBuf[5]]); COM1 = 0;
            isLcdComflag = 2;
    }
    else if(isLcdComflag ==2)          //显示 COM2 数据
    {
            LedSetSegData(LcdDisplayBuf[gIsLedDataBuf[1]]); COM2 = 0;
            isLcdComflag = 3;
    }
    else if(isLcdComflag ==3)          //显示 COM3 数据
    {
            LedSetSegData(gIsLedDataBuf[3]); COM3 = 0;
            isLcdComflag = 4;
    }
    else if(isLcdComflag ==4)          //显示 COM4 数据
    {
            LedSetSegData(gIsLedDataBuf[4]); COM4 = 0;
            isLcdComflag = 5;
    }
    else if(isLcdComflag ==5)          //显示 COM5 数据
    {
            LedSetSegData(LcdDisplayBuf[gIsLedDataBuf[2]]); COM5 = 0;
            isLcdComflag = 6;
    }
    else if(isLcdComflag ==6)
    {
            isLcdComflag = 0; ComAllClose;
    }
}
```

三、ARM 应用参考原理图



四、软件参考示例

下面附程序说明

Main.c

```
void main()
{
    Sys_Init();         //初始化 IO 口，强推挽输出
    TK_Init();          //重要步骤 1: 扫键的初始化函数
    while(1)
    {
        WDT->WDT_CON |= WDT_CON_CLRWDT;  //清 watchdog

        //重要步骤 2: 触摸键扫描一轮标志，是否调用 TouchKeyScan()一定要根据此标志位置起后
        if(TK_TouchKeyStatus & 0X80)
        {
            //重要步骤 3: 清除标//志位， 需要外部清除。
            TK_TouchKeyStatus &= 0X7F;
            TK_exKeyValue = TK_TouchKeyScan();      //重要步骤 4: 分析按键数据， 并返回结果出来
            ChangeTouchKeyvalue();                  //转换键值
            UpdateDisplay();

            TK_Restart();                           //启动下一轮转换
        }

    }
}

void ChangeTouchKeyvalue(void)
{
    switch(TK_exKeyValueFlag)
   {
        case 0x00000001:exKeyValue = 1;break;
        case 0x00000002:exKeyValue = 2;break;
```

```
        case 0x00000004:exKeyValue = 3;break;
        case 0x00000008:exKeyValue = 4;break;
        case 0x00000010:exKeyValue = 5;break;
        case 0x00000020:exKeyValue = 6;break;
        case 0x00000040:exKeyValue = 7;break;
        case 0x00000080:exKeyValue = 8;break;
        case 0x00000100:exKeyValue = 9;break;
        case 0x00000400:exKeyValue = 10;break;
        case 0x00001000:exKeyValue = 11;break;
        case 0x00002000:exKeyValue = 12;break;
        default:exKeyValue = 0xff;break;
    }
}


void  UpdateDisplay (void)
{
    if(exKeyValue != 0XFF)        //松手前只出一次键
    {
            gTkIsValid = 1;
            KeyValue=exKeyValue;

            if(KeyValue != 0)
            {
                LED_TurnOver;     //LED 灯翻转
            }
     }
    else
    {
            gTkIsValid = 0;
    }

}
```

布局规则快速检查表

序号	类别		最小值	最大值	推荐/备注
1	按键	形状	N/A	N/A	推荐圆形，其次方形。
		大小	5mm	15mm	12mm
		与 GND 覆铜的间隙	3mm	N/A	如果有覆 GND 铜，请保持推荐间距
2	滑条	滑条两端长度(W1)	5mm	7mm	6mm
		单个滑条段中心长度(W2)	3mm	5mm	4mm
		不同滑条段的间隙(A)	0.5mm	2mm	0.5mm
		滑条段的高度(H)	7mm	15mm	12mm
		单个滑条段的长度(L)	9mm	15mm	12mm
3	覆盖层	材料选择	N/A	N/A	介电常数 2.0-8.0 材料（导电材料除外） 常规触摸按键和外壳的覆盖层尽量不留空气间隙（除隔空感应应用外）
		触摸按键应用的面板厚度	N/A	5mm/8mm	非玻璃材料最大厚度应小于 5mm，玻璃材料最大厚度应小于 8mm。 触摸按键尺寸较大可增加覆盖层厚度
		滑条按键应用的面板厚度	N/A	5mm/8mm	非玻璃材料最大厚度应小于 5mm，玻璃材料最大厚度应小于 8mm。 触摸按键尺寸较大可增加覆盖层厚度
4	触摸走线	长度	N/A	200mm /50mm	对于标准的（FR4）PCB 板，长度为 200 mm 对于柔性 PCB 板，长度为 50 mm
		宽度	7mil	10mil	
		与其他网络走线间距	2mm	N/A	其他走线包括触摸走线和非触摸走线
		与其他器件和金属件	10mm	N/A	
		布线	N/A	N/A	非触摸走线穿过触摸走线时，应确保其垂直相交，走线转角不能小于 90 度。
5	过孔	数量	0	2	若超过过孔数量，应尽可能缩短走线
		大小	N/A	N/A	10mil
6	GND 覆铜	网格填充百分比	N/A	N/A	铺设网格地设置（7 mil 的线宽，45 mil 的空间）
7	触摸按键串接电阻	大小	510R	5.6K	510R
		放置位置	N/A	N/A	电阻放置在离触摸引脚 10 mm 的范围内
8	CMOD 电容	大小	N/A	N/A	103，精度 10%，低功耗应用推荐 102 电容，精度 10%
		放置位置	N/A	N/A	电容放置在离 CMOD 引脚 10 mm 的范围内
9	104 电容/电解电容	放置位置	N/A	N/A	电容放置在离 VDD/GND 引脚 10 mm 的范围内 电源线和地线应先经过电容滤波（电解电容+104 瓷片电容）之后再分别接入 MCU 的 VDD 和 VSS 管脚，也可将电解电容换为钽电容，容值不小于 10uF
10	触控芯片外围高频信号器件	放置位置	N/A	N/A	防倍频干扰：触控走线区远离高频信号源（如晶振、Wi-Fi 等）。严禁扫描频率与周边高频信号成倍频关系，避免噪声超标导致系统异常。

6 规格更改记录

版本	记录	日期
V2.0	1. 引入防倍频干扰注意事项，触控 Layout 远离高频器件，错开扫描频率与外围晶振/Wi-Fi 等信号的倍频点，防止噪声过大	2026 年 5 月
V1.9	1. 格式调整，C51/ARM 触控应用指南统一规整 2. 引入 PCB 设计	2025 年 9 月
V1.8	1. 新增 CMOD 电容设置参数说明	2024 年 5 月
V1.7	1. 新增特殊应用情景下的抗干扰设置说明	2023 年 9 月
V1.6	1. 删除系列区分，统一版本 2. 完善高灵敏度触控描述，去除高可靠模式描述	2022 年 10 月
V1.5	增加打开触控调试上位机调试参数前，需烧录静态调试码的提示	2022 年 7 月
V1.4	文档格式规范化 修改高灵敏度信噪比描述	2021 年 11 月

声明