目录

1 赛元 SC92_93F 系列 TOUCHKEY MCU 应用指南总体描述

本文档是赛元 SC92_93F 系列 Touchkey MCU 触控的应用指南，主要介绍如何使用赛元提供的触控按键库文件以及触控上位机如何调试参数。赛元触控 MCU 的触控架构分为高灵敏度触控 模式和高可靠触控模式，部分型号内建双模触控(具体参见规格书描述)，可通过选择不同的触控库文件来使用高灵敏度模式或高可靠模式，其特点如下：

- ●高灵敏度模式可适应普通触控按键、隔空按键触控、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/高可靠模式都具有很强的抗干扰能力
- ●最多可实现 31 路触控按键及衍生功能
- ●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
- ●部分型号可以在 MCU STOP 模式下进入低功耗模式工作，12 个触控按键 500mS 唤醒时芯片整体功耗可低至 22uA@3.3V / 25uA@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控功能。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	高可靠模式
特点	●超强抗干扰能力，可通过 10V 动态 CS ●超高灵敏度	●超强抗干扰能力，可通过 10V 动态 CS ●功耗更低
适用的应用	●弹簧触控按键应用 ●隔空触控按键应用 ●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	●需要更低的低功耗电流
如何进入模式	通过项目工程载入赛元所提供的高灵敏度的触控库来选择高灵敏度模式	通过项目工程载入赛元所提供的高可靠的触控库来选择高可靠模式
库体说明	3.3.1 高灵敏库触控软件库移植	3.3.2 高可靠库触控软件库移植
对应的库文件	“SC92/93F8XXX_HighSensitive_Lib_Tn_Vx.x.x.LIB”	“SC92/93F8XXX_HighReliability_Lib_Tn_Vx.x.x . LIB
注意事项	●T1 库应用于弹簧类型的应用 ●T2 库应用于隔空类型的应用，且按键个数至少 3 个以上	●只可应用于弹簧类型的应用
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。	只有以下情况下建议使用高可靠模式： 需要更低的低功耗电流，且高灵敏度模式下电流无法充满

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幅度 ≦3%~4% ，最大不超过 200mv。

2 赛元触控库介绍

2.1 触控库应用类型

赛元 SC92_93F 系列 Touchkey MCU 提供可以供用户调用的库文件，以降低用户触控按键部分的开发难度。大体分为以下几类库体类型：

- ●普通触控按键库
  - ■ 弹簧触控库（简称 T1 库）
  - ■ 隔空触控库（简称 T2 库）
  - ■ 高可靠库
- ●滑轮滑条触控按键库
- ●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）

本文将初步介绍普通触控按键库体使用以及触控调试上位机 SOC TouchKey Tool Menu 软件使用，滑轮滑条库以及接近感应库体将由特殊应用指南详细介绍使用，详细请查看：《赛元 TK 触控特殊应用说明》

用户仅仅需要经过以下几个步骤，便可实现触控按键的功能，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。

2.2 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 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 最合适的触控按键关键参数，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**

用户自行写好除触控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：3 触控开发流程

2.3 赛元触控库文件介绍

赛元触控库包括以下几个文件：

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。

SC92/93F8XXX_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。

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》。**

3 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《赛元触控按键 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。**

3.1 安装开发工具

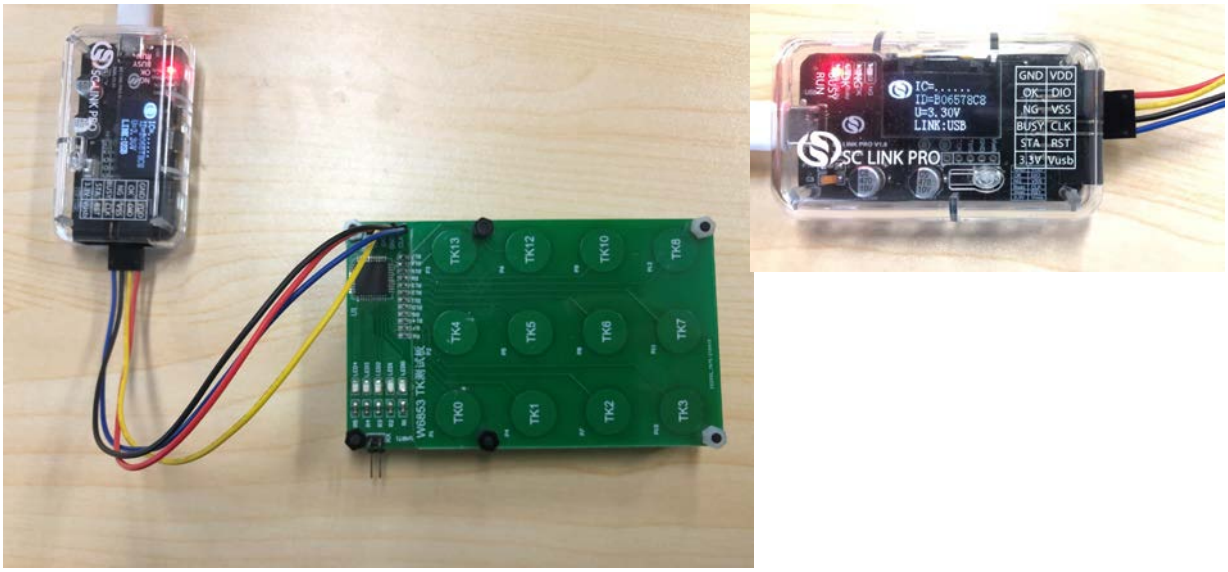
1) Setup SOC Pro51/SOC Programming Tool
安装赛元软件 SOC Pro51 Vx.xx.exe/ SOC Programming Tool (请从赛元网站找最新版本)。

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 DPT52/SC-LINK/SC-LINK PRO 固件,更新 MCU 库
在线烧写器 DPT52/SC-LINK/SC-LINK PRO 的固件和 SOC Pro51/ SOC Programming Tool 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。

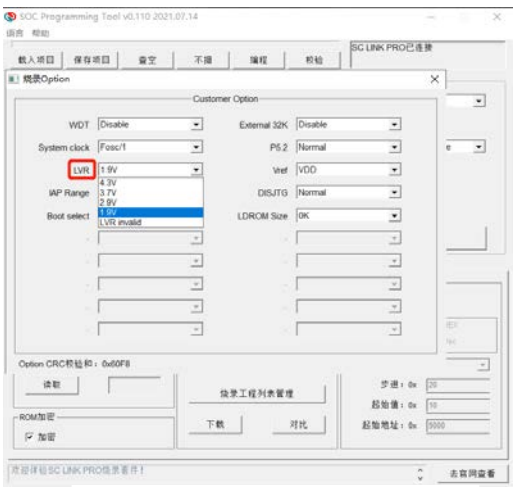
5) 硬件连接顺序：电脑 USB-->DPT52/SC-LINK/SC-LINK PRO(VCC/GND/CLK/DIO)--> 用户 PCB(VCC/GND/tCK/tDIO),并测试连接正常。调试过程需要用到硬件 UART 资源，请 PCB 预留接线。如图为 SCLINK PRO 的接线。



6) 烧录静态调试代码 hex 文件到用户 PCB 上的 SC92_93F8XXX IC 中
打开 SOC Pro51/SOC Programming Tool 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 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**)
见以下 SOC Pro51/SOC Programming Tool 软件所示图：

SOC Pro51 软件



SOC Programming Tool 软件

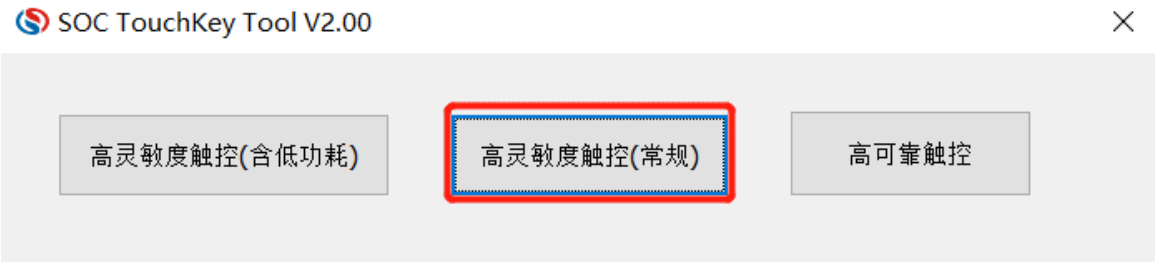
高灵敏调试见以下操作步骤：3.2.1 高灵敏度调试触控参数
高可靠调试见以下操作步骤：3.2.2 高可靠调试触控参数

3.2 调试触控参数

3.2.1 高灵敏度调试触控参数

1) 打开 Touch Key Tool Menu，选择高灵敏度触控

常规：普通触控按键调试，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。



2) 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：

② 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。

隔空距离：**针对隔空按键应用需要设置选择 0~3mm。其他应用无需设置。**
(更远距离请联系赛元工程师协助)

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。

③ 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。

自动校准次数：建议保持默认。该参数决定了初始化基线的速度，次数越多基线越稳定，同时时间也更长。

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。按键时间到达指定次数，则该按键的标志会被清除。

动态更新基线时间：建议保持默认。该参数用于处理按键浮起的更新速度，保持默认不改动 基线更新速度：该参数用于更新基线。

基线复位速度：建议保持默认。该参数决定基线复位的速度。值越大，更新速度越慢。

滤波 K 值：建议保持默认。

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：用于低功耗应用时，禁止开启抗干扰设置。

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。

④ 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。

3) 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。**

4) 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：

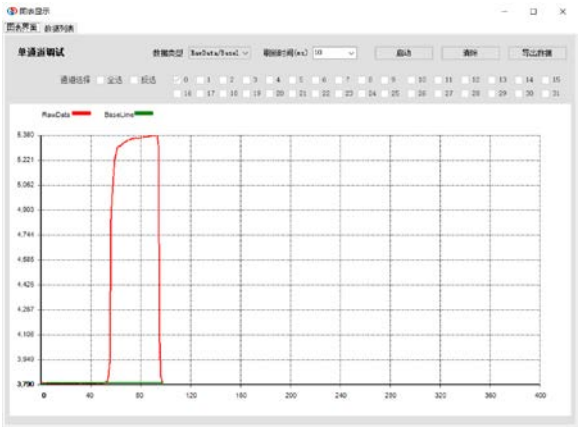
若调试不通过，则显示红色图标。

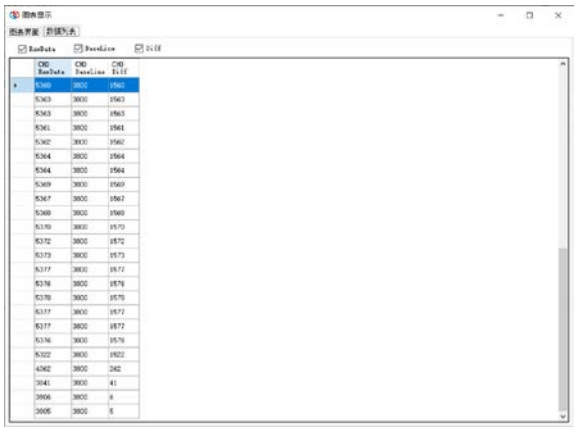
不通过的项目相应会红色字体标出。

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（只针对普通触控按键进行诊断，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

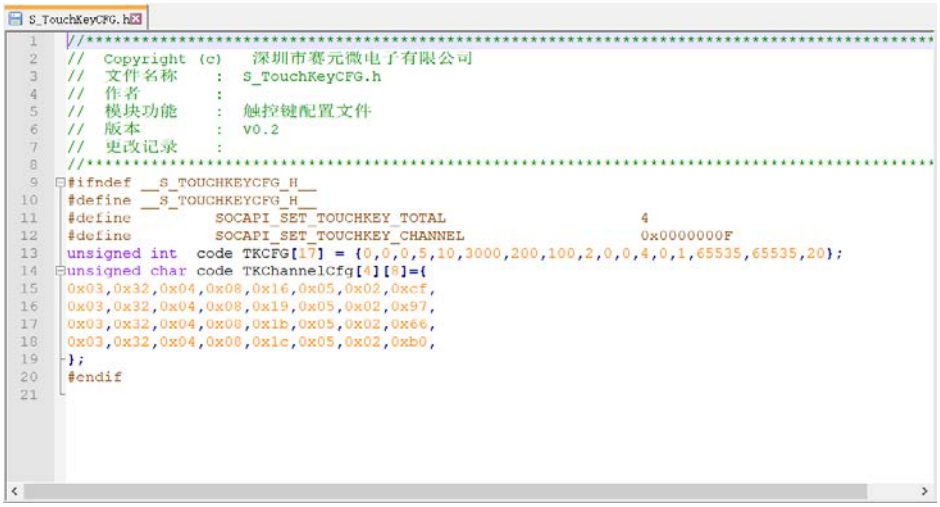
注：软件显示的 TK 通道与 MCU 规格书一致，请根据实际 PCB 的 layout 布局，操作对应的按键，否则得到的结果将会错误！

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**）。

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                   4
#define         SOCAPI_SET_TOUCHKEY_CHANNEL                 0x0000000F
unsigned int  code TKCFG[17] = {0,0,0,5,10,3000,200,100,2,0,0,4,0,1,65535,65535,20};
unsigned char code TKChannelCfg[4][8]={
0x03,0x32,0x04,0x08,0x16,0x05,0x02,0xcf,
0x03,0x32,0x04,0x08,0x19,0x05,0x02,0x97,
0x03,0x32,0x04,0x08,0x1b,0x05,0x02,0x66,
0x03,0x32,0x04,0x08,0x1c,0x05,0x02,0xb0,
};
#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

至此触控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。

5) 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

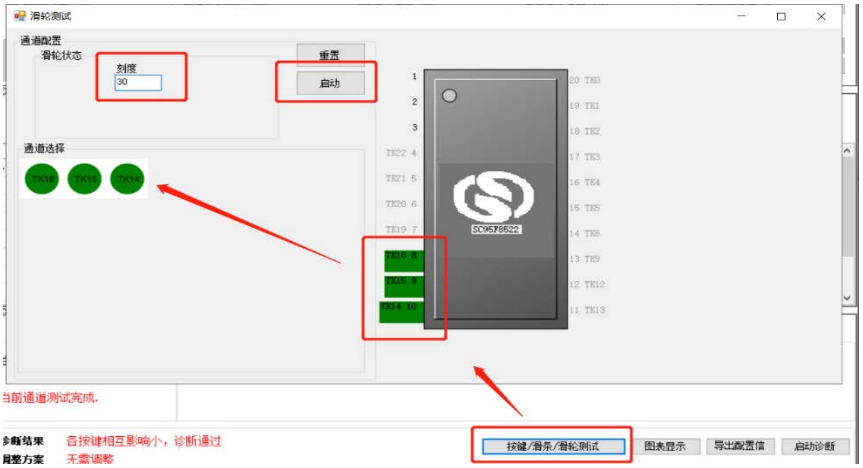
① 设置刻度值：对应滑轮最大刻度

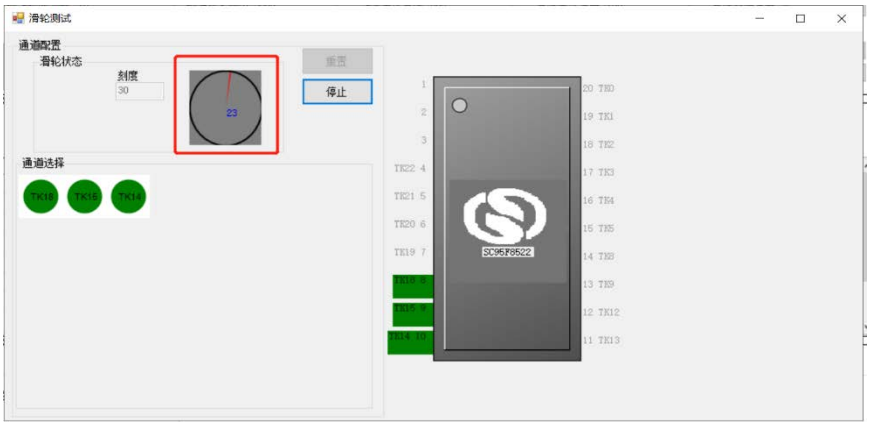
② 设置滑轮通道顺序：注意选择的顺序，需按着硬件上滑轮 TK 通道顺序设置，例如以下图片滑轮按着 TK18->TK15->TK14 的顺序排布

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。**

3.2.2 高可靠调试触控参数

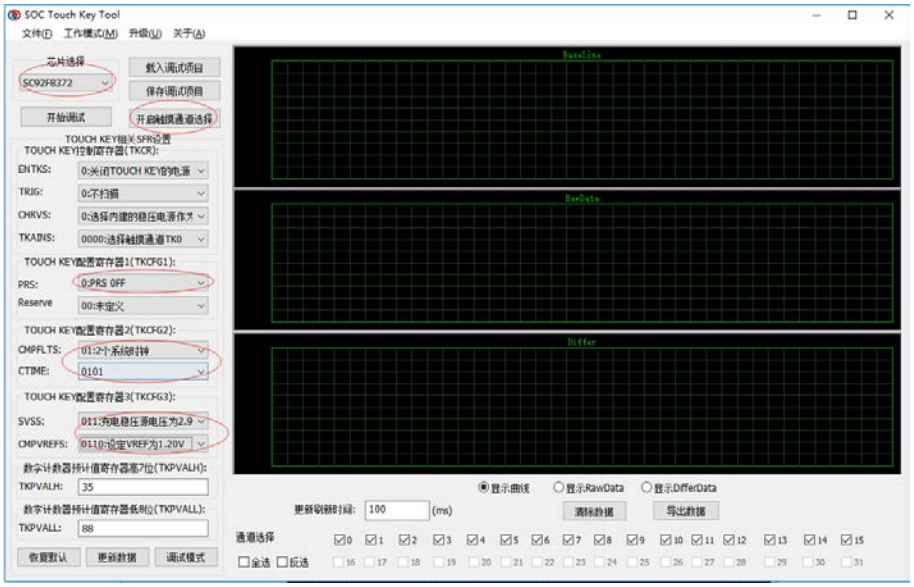
1) 打开 Touch Key Tool Menu，选择高灵敏度触控
2) 调试提取参数

理论数据的采集方式（用电脑的 USB 电源供电），此时数据的 Noise 较实际使用时要小，可供参考使用；另一种是实际数据采集方式（用实际产品电源供电，实际产品电源与电脑电源，两者必须有一端接隔离变压器），此数据接近于实际的数据，特别是信噪比更加准确。

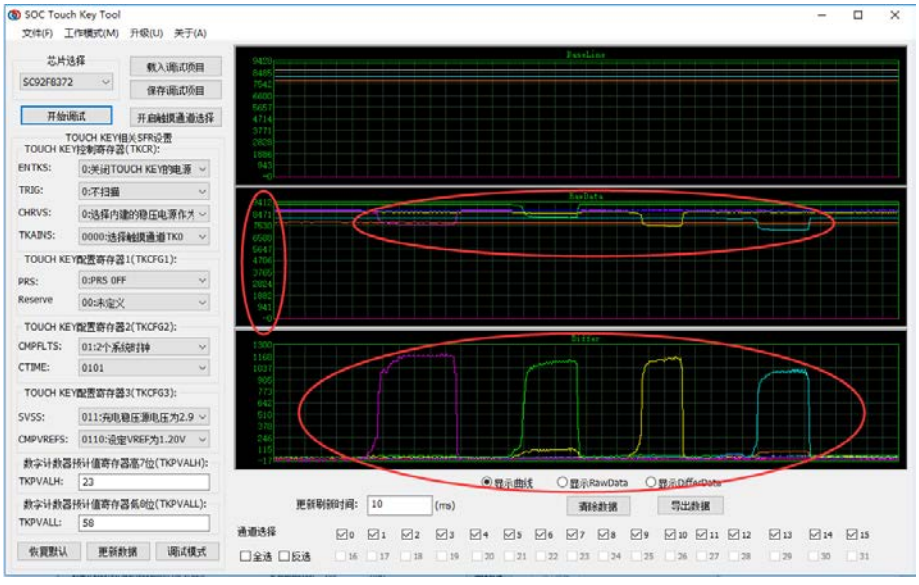
① 配置调试软件 SOC Touch Key Tool 界面中的寄存器：
② 通道选择（将用作 TK 的通道打勾 ）；
③ 配 置 TKCFG1 中的 PRS ； 配 置 TKCFG2 中 的 CTIME 、 CMPFLTS ；
④ 配置 TKCFG3 中的 SVSS、CMPVREFS；
⑤ 点击界面的“保存调试项目”按钮来保存此配置；

赛元建议优先先按缺省的配置来测试：

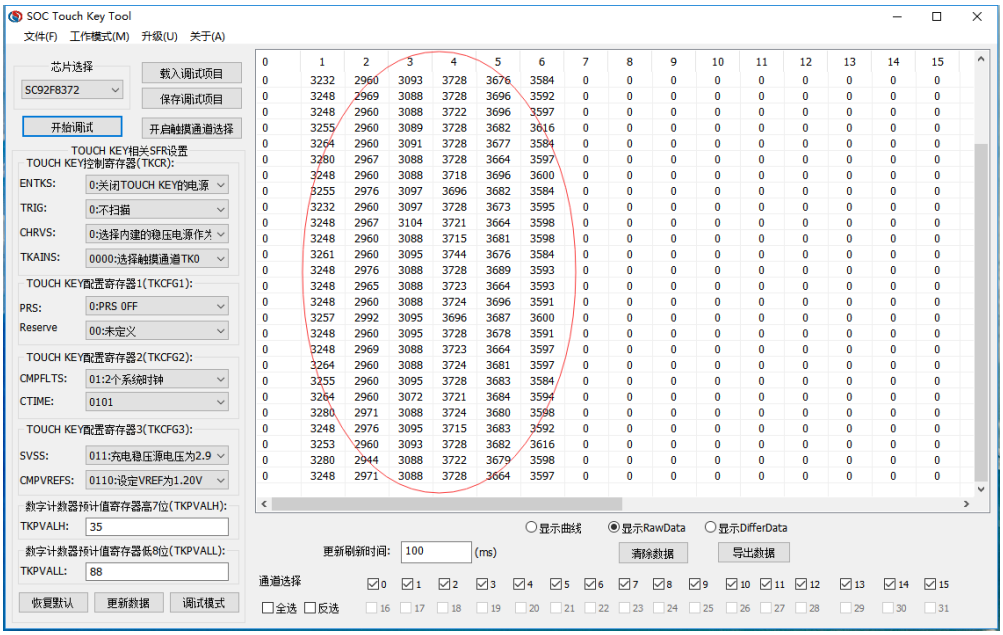
- PRS：设为 0；PRS OFF
- CTIME：0101b=5；建议范围：1~F；(SC92F8X4XB 系列建议设置为 3)
- CMPFLTS：设为 01,两个时钟滤波；建议范围：1~3；
- SVSS：设为 2.90V；建议范围：2V~4.2V
- CMPVREFS：设为 1.20V；建议范围：0.6V~2.1V（SVSS-CMPVREFS 要大于 0.7V）



⑥ 开始调试并读取导出数据；单击“开始调试”按钮，可在图形模式下看到所选中通道出现不同颜色的波形，如果对应通道有按键被按下，对应通道的波形幅度会有相应的变化，如图：



用户可以点击选择“显示 RawData”切换到数据模式，便可看到每个通道对应的原始数据值，此数据就是 IC 的 TKCNT 的真实值。如下图：



图中所圈部分即是选中的 TK1~TK7 的 7 个通道的 TKCNT 值，每一列代表一个通道，没有选中的通道 TKCNT 值为 0。如果出现特殊情况，有可能是数据太大，超出了 TKCNT 的范围，请检查 Cadj 的电容是否 接错，或者减小此电容。

⑦ 配置 TKCFG1、TKCFG2 和 TKCFG3 让所有通道的 RAW DATA 值达到目标值, 不共用项目目标值一般定 3000~9000，对于共用项目，目标值配置在 9000~13000。

建议用户先按照原始配置的建议参数值来看此时的 RAW DATA 是否符合要求，如果不符合要求则按下述方式进行调整（注意，下述调节中如果完成第一步就 OK，请停止修改其它参数；不行再修改下一步；依次类推）：

如果所有通道 RAWDATA 数据过小：

- 可增加 SOCAPI_SET_TKCFG2 中的 CTIME (建议 CTIME： 1~F)；
- 可减小 SOCAPI_SET_TKCFG3 中的 SVSS（建议 SVSS：2.0~4.2V）；
- 可增大 SOCAPI_SET_TKCFG3 中的 CMPVREFS （建 议 CMPVREFS ：0.6~2.1V）；
- SVSS 与 CMPVREFS 之间的关系：SVSS-CMPVREFS>0.7V；
- 可加大电容（建议 473 以下）；

如果所有通道 RAWDATA 数据过大：

- 可减小 SOCAPI_SET_TKCFG2 中的 CTIME (建议 CTIME：1~F)；
- 可增大 SOCAPI_SET_TKCFG3 中的 SVSS（建议 SVSS：2.0~4.2V）；
- 可减小 SOCAPI_SET_TKCFG3 中的 CMPVREFS（建议 CMPVREFS：0.6~2.1V）；
- SVSS 与 CMPVREFS 之间的关系：SVSS-CMPVREFS>0.7V；
- 可减小电容（建议 222 以上）；

如不同触控按键通道的 RAW DATA 值差异太大：

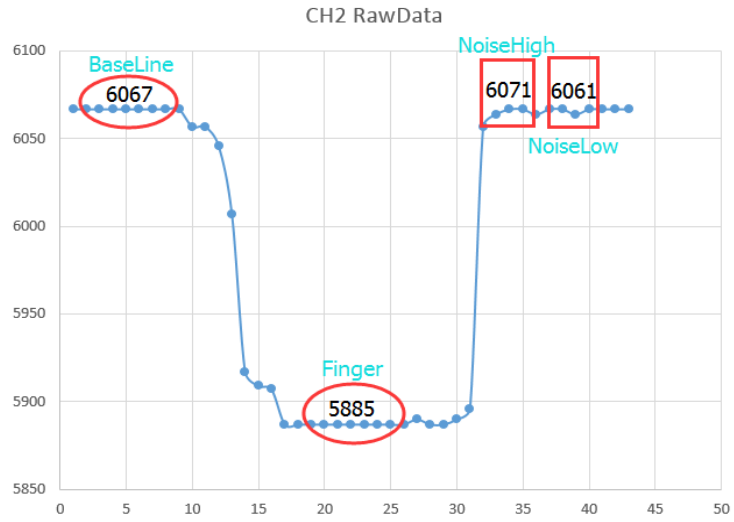
- 可能的原因是该 PCB 走线长度及外围耦合电容有较大差异；要设法排除；
- 再通过上述方式调整参数；
- 若差异太大无法调整，则考虑修改 PCB；

其它对触控按键灵敏度的影响：

- 覆盖物介质厚度：介质越厚 RAWDATA 值越小；介质越厚触控按键灵敏度越低；
- 弹簧直径与覆盖物介质接触面：弹簧直径越大 RAWDATA 值越小；弹簧直径越大触控按键灵敏度越高；

⑧ 配置 TKCFG1 、 TKCFG2 和 TKCFG3 让所有通 道信噪比都符合要求值 （建议 SNR>5）；

下图为将原始数据 RAW DATA 导出成为 EXCEL 表格后（参考下一步导出数据），选择一个通道的数据，在 EXCEL 中做出的图形。



如图所示为一个通道手指按下的波形图，从中我们能取到几个有用的值：

Noise High：没有按下时 RAW DATA 的最大值；

Noise Low：没有按下时 RAW DATA 的最小值；

Baseline：没有按下时的 RAW DATA 的平均值；

Finger：手指按下时 RAW DATA 的平均值

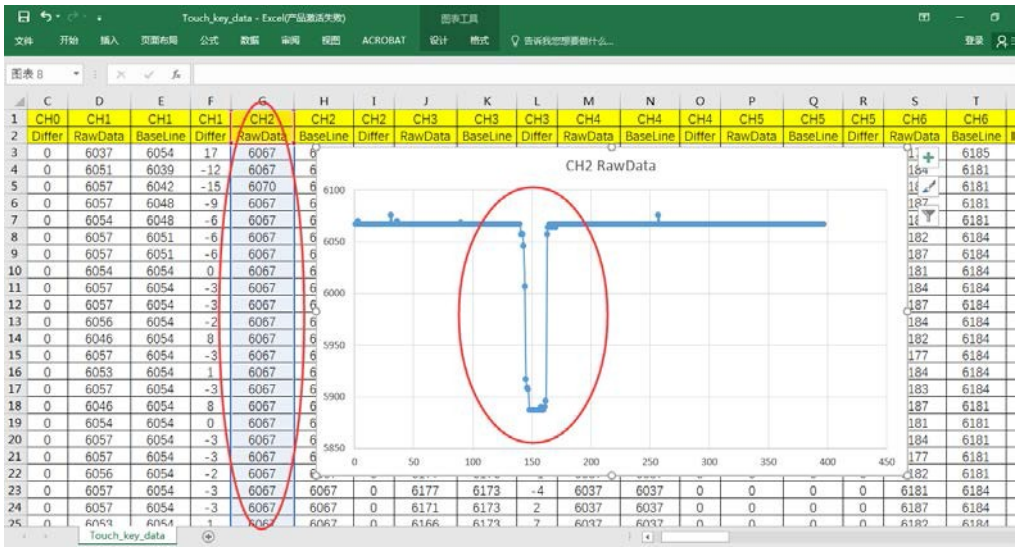
信噪比 SNR 计算方法：

SNR=（Baseline-Finger)÷（NoiseHigh-NoiseLow）

如上图中：SNR=（6067-5885）÷（6071-6061）=18；

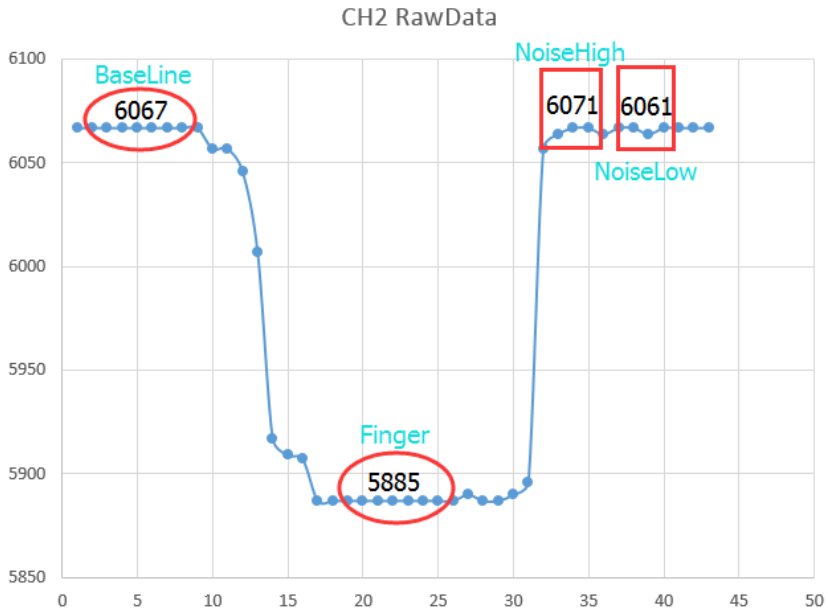
调试方式参考上一步，配置 TKCFG2 和 TKCFG3 的参数，让所有通道的 RAWDATA 值达到目标值，并且信噪比 SNR 符合要求（SNR 大于 5）

⑨ 导出 EXCEL 格式的原始数据 RAW DATA（每一通道的有触控按键值和无触摸按键值）；



用户此时需要记录下参数配置和重要数据：

- **每个通道不同的数据：包括 Baseline、Finger、NoiseHigh、NoiseLow；**
- **所有通道共同的参数为：寄存器 TKCFG2 中 CMPFLTS 和 CTIME 的值,寄存器 TKCFG3 中 SVSS 和 CMPVREFS 的值；**



注意：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。

3.3 实现赛元软件库的功能测试

3.3.1 高灵敏库触控软件库移植

1. 库文件介绍
   1) 弹簧库文件（简称 T1 库，SC92/93F8XXX_HighSensitive_Lib_T1_Vx.x.x.LIB）
   2) 隔空库文件（简称 T2 库，SC92/93F8XXX_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
SC92_93F8XXX_HighSensitive_Lib	库文件，实现触控按键检测算法	
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 用到的资源

库体系列	RAM 占用内存	ROM 占用内存
(T1 库体) SC92_93F8XXX_HighSensitive_Lib 和 S_TouchKeyCFG.C 所占用大小	data 区：49.3 个 byte； Xdata 区：18 个 Byte；与按键的个数无关，均要使用； Xdata 区：每一个按键 15 个 Byte； 如有 3 个按键： Data 区 49.3byte, xdata 区 18+3*15=63 byte	库使用 ROM 大小约 3.6K，且增加或减少若干按键，该大小基本不变，差异不到 200byte
(T2 库体) SC92_93F8XXX_HighSensitive_Lib 和 S_TouchKeyCFG.C 所占用大小	data 区：59.3 个 byte； Xdata 区：10 个 Byte；与按键的个数无关，均要使用； Xdata 区：每一个按键 15 个 Byte； 如有 3 个按键： Data 区 59.3byte, xdata 区 18+3*15=55 byte	库使用 ROM 大小约 3.6K，且增加或减少若干按键，该大小基本不变，差异不到 200byte

注意：每个芯片库体所占内存大小差异很小，具体所占内存大小请以赛元资料为准。

3. 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，自动校准次数相关; 按键每多 N 个，大约时间 24M 主频：54 uS *N 按键 ；16M 主频：48 uS *N 按键**
TouchKeyRestart(void)	**使能一轮触控按键扫描**	**1 用户主程序控制何时启动按键扫描； 2 启动按键扫描后，在一轮触控按键扫描完成之前，不能对触控按键通道进行操作：如操作触控按键通道的 IO，否则触控按键功能将无法实现。**
Unsigned long int TouchKeyScan(void)	**触摸按键算法处理**	**1 用户需要在触控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：(50uS*N 个按键 @32M ， 340 uS*N 个按键 @24M)；**

4. 全局变量 SOCAPI_TouchKeyStatus 的说明

1) 全局变量在 S_TouchKeyCFG.c 头文件中声明

① Unsigned char xdata SOCAPI_TouchKeyStatus;

② SOCAPI_TouchKeyStatus 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_TouchKeyStatus&0x80)时，调用 TouchKeyScan(void) 进行算法数据处理，给出键值；

3) 使能触控按键扫描之前，一定要清掉标志。

清除一轮扫描标志 SOCAPI_TouchKeyStatus &=0x7f;

5. LIB API 接口函数的返回值说明

1) TouchKeyScan(void)函数返回值：

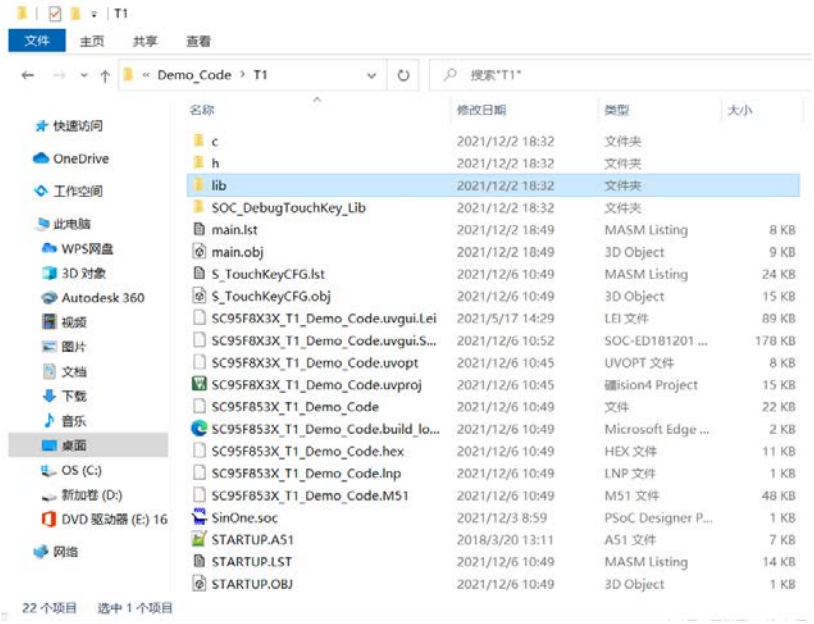
返回值对应 bit 为 1 即该通道有按键，0 为无按键。

若使能双键，且有两键触发，则会有两个 bit 位置起。

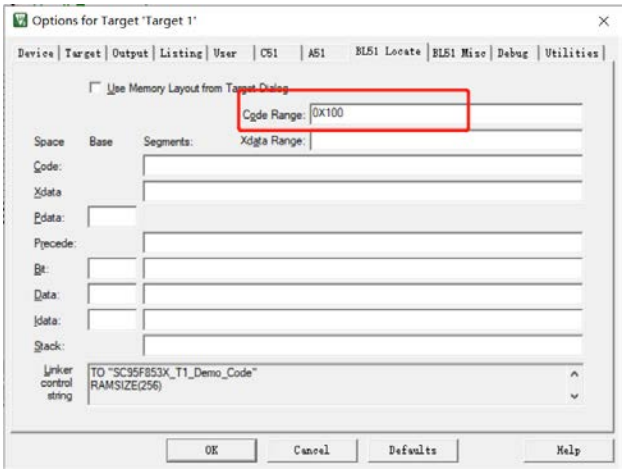
数据位		Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24
含义		TK30	TK29	TK28	TK27	TK26	TK25	TK24
	触控按键状态（1：有效；0：无效）							
数据位	Bit23	Bit22	Bit21	Bit20	Bit19	Bit18	Bit17	Bit16
含义	TK23	TK22	TK21	TK20	TK19	TK18	TK17	TK16
	触控按键状态（1：有效；0：无效）							
数据位	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
含义	TK15	TK14	TK13	TK12	TK11	TK10	TK9	TK8
	触控按键状态（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
	触控按键状态（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。

6. 打开工程项目文件复制“lib”文件夹至工程文件夹内

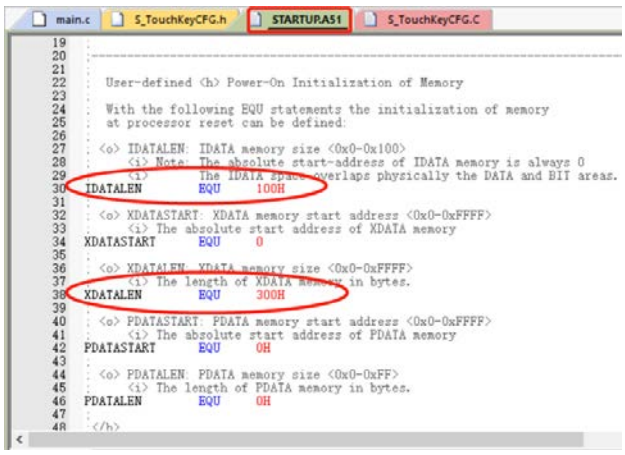


7. 在 Keil 中打开工程项目文件；注意设定（Code range）、设定（XDATALEN EQU xxxH）



设定的目的，参见赛元 MCU 应用注意事项 Vx.xx.PDF 文件。
（部分 92/95F 系列芯片不用设置该项，请仔细阅读）

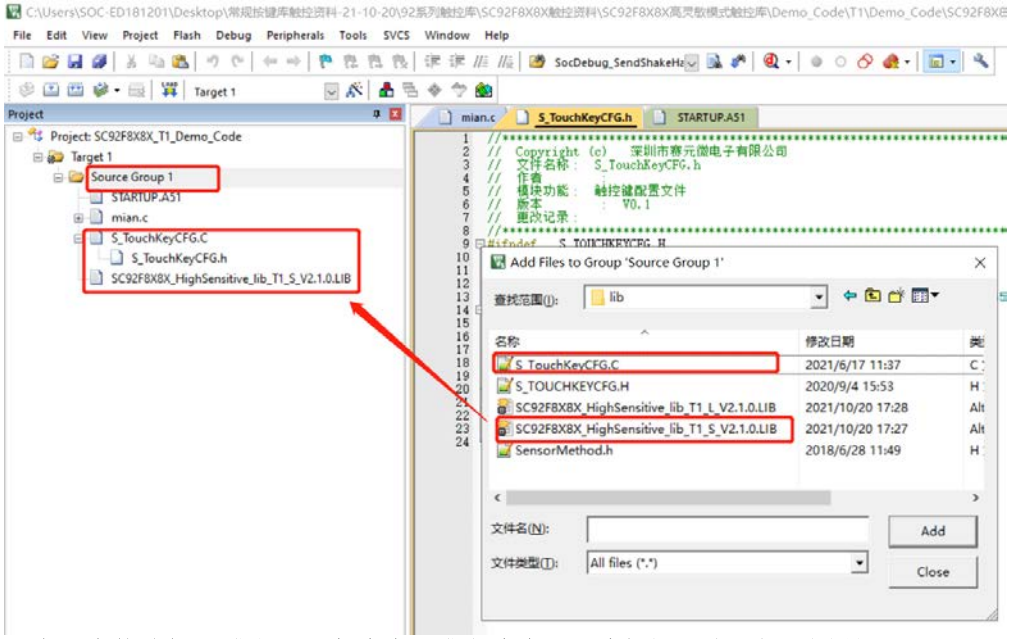
8. 设定 XDATALEN



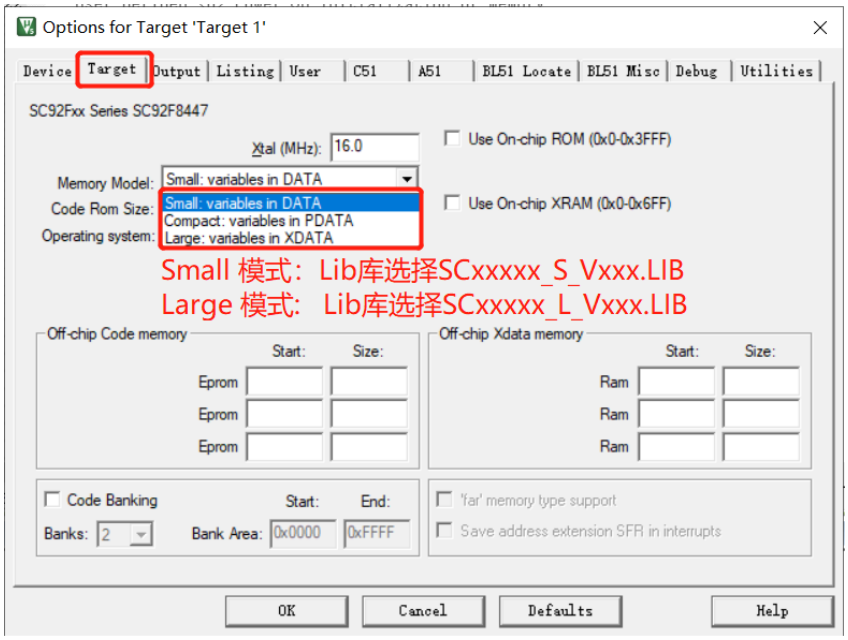
注意：用于 STARTUP.A51 中，清掉外部 XData，具体型号的 XData 大小见规格书。

9. 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）

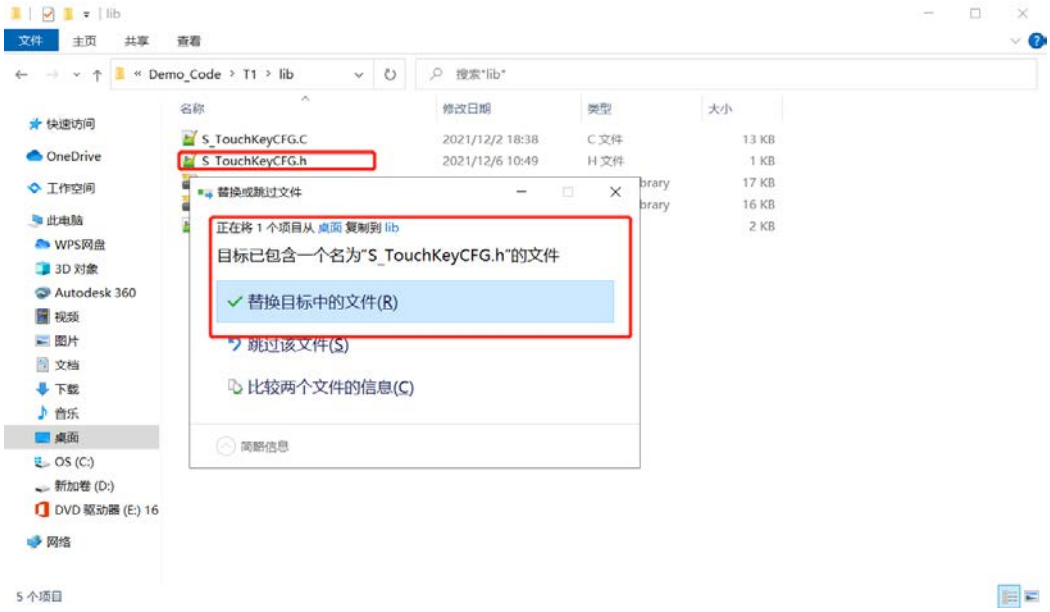


注意：库体选择 L 或者 S（大端编译或小端编译）请仔细区分，如下图所示：



10. 在主程序文件中添加头文件引用

11. 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高。

3.3.2 高可靠库触控软件库移植

1. 库文件介绍

1) 高可靠库文件（SC92_93F8XXX_HighReliability_Lib_T1_Vx.x.x.LIB）

文件	用途	说明
SC92_93F8XXX_HighReliability_Lib_T1_Vx.x.x.lib	**库文件，实现触控按键检测算法**	
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**	**用户不需要修改**
S_TOUCHKEYCFG.H	**头文件，提供 TouchKey 功能的宏定义**	**用户可改变部分宏定义，来改变 TouchKey 寄存器的设置**

2. Lib 用到的资源

1) LIB 库所占资源大小（RAM 和 ROM）

库体系列	RAM 占用内存	ROM 占用内存
SC92_93F8XXX_Reliability _Lib 和 S_TouchKeyCFG.C 所占用大小	**data 区：40.2 个 byte； Xdata 区：23 个 Byte；与按键的个数无关，均要使用； Xdata 区：每一个按键 13 个 Byte；如有 5 个按键： Data 区 40.2byte, xdata 区 23+5*13=88 byte**	**库使用 ROM 大小约 3.2K，且增加或减少若干按键，该大小基本不变，差异不到 200byte**

注意：每个芯片库体所占内存大小差异很小，具体所占内存大小请以赛元资料为准。

2) 中断；仅使用 TK 中断，默认优先级

中断源	中断优先级	中断向量	查询优先级	中断号（C51）	标志清楚方式
TK	低	005BH	12	11	H/W Auto

注意：库仅使用了触摸中断，优先级为低，且中断服务程序内无函数嵌套，执行时间约 6.8us。

3) S_TouchKeyCFG.C 文件内各参数含义

参数	类型	数值	说明
SOCAPI _SET_TKCFG1	无符号 8 位整型常量	建议使用默认值	控制寄存器 TKCFG1 描述
SOCAPI _SET_TKCFG2	无符号 8 位整型常量	建议 CTIME = 0x03 到 0x0f	控制寄存器 TKCFG2 描述
SOCAPI _SET_TKCFG3	无符号 8 位整型常量	建议使用默认值	控制寄存器 TKCFG3 描述
SOCAPI _SET_TouchKey_Total	无符号 32 位整型常量	1~23	设定触控按键的个数
SOCAPI_SET_TouchKey_Channel	无符号 32 位整型常量	用户选定触控按键通道而定	选定触控按键通道； Bit0~Bit23 对应 TK0~TK23； 1 为 TK 通道；0 为 IO； 0000 0000 0000 0101：TK0 和 TK2 是 TK，其它是 IO； 以此类推 SOCAPI _SET_TouchKey_Channel 选中了的通道个数需和 SOCAPI_SET_TouchKey_Total 值相等
SOCAPI _SET_TouchKeyCONFIRM_CNT	无符号 8 位整型常量	建议：5~40，一般选择 10 次	按键的确认时间。 连续 SOCAPI_SET_TouchKey CONFIRM_CNT 轮均扫描到该按键，才认为是有键按下；数值大，会导致按键反应变慢；
SOCAPI _SET_NOISE_Threshold	无符号 8 位整型常量	建议:20-40	设置噪音值
SOCAPI _SET_FINGER_Threshold	无符号 8 位整型常量	根据采集数据设定	设置手指阈值。用 SOC Touch KeyTool 工具采集，用 10mm 直径的铜柱按下后的 diff 值*60%作为手指值，同时保持设置的手指阈值与设置 的噪音值比要大于 5。数组元素的个数与设定触控按键的个数相匹配

3. Lib API 接口函数 S_TouchKeyCFG.c 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：与按键个数，按键扫描时间，自动校准次数相关;** **12M 主频：54*N 按键** **16M 主频：48*N 按键** **32M 主频：45*N 按键**
TouchKeyRestart(void)	**使能一轮触控按键扫描**	**1 用户主程序控制何时启动按键扫描；** 2 启动按键扫描后，在一轮触控按键扫描完成之前，不能对触控按键通道进行操作：如操作触控按键通道的 IO，否则触控按键功能将无法实现。
Unsigned long int TouchKeyScan(void)	**触摸按键算法处理**	**1 用户需要在触控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：算法执行时间会因按键** **选择个数的增减而增减。** **N 个 Key 约(240*N) us@12M。**

4. 全局变量 SOCAPI_ToucKeyStatus 的说明

1) 全局变量在 S_TouchKeyCFG.c 头文件中声明
   Unsigned char xdata SOCAPI_ToucKeyStatus;
   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) 使能触控按键扫描之前，一定要清掉标志。
   清除一轮扫描标志 SOCAPI_ToucKeyStatus &=0x7f;

5. 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。具体如下所示：

数据位		Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24
含义		TK30	TK29	TK28	TK27	TK26	TK25	TK24
	触控按键状态（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
	触控按键状态（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
	触控按键状态（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
	触控按键状态（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。

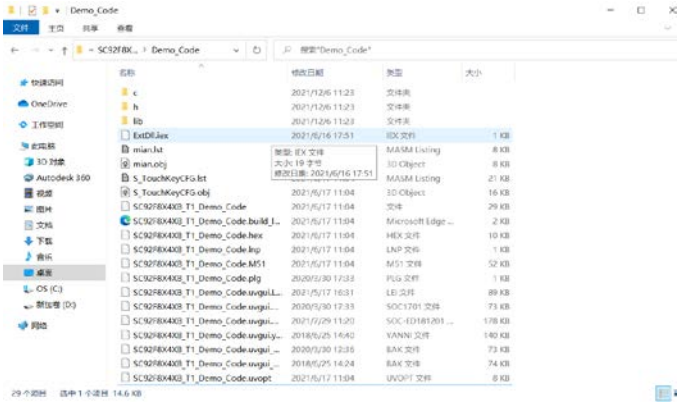
6. 按键有效的最长输出时间

```
#define SOCAPI_SET_KEY_CONTI_TIME 1000    // 按键有效的最长输出时间，设置范围 0~5000，默认 1000，输出时间= 1000*单位每轮扫描时间（如 10ms）=10S
```

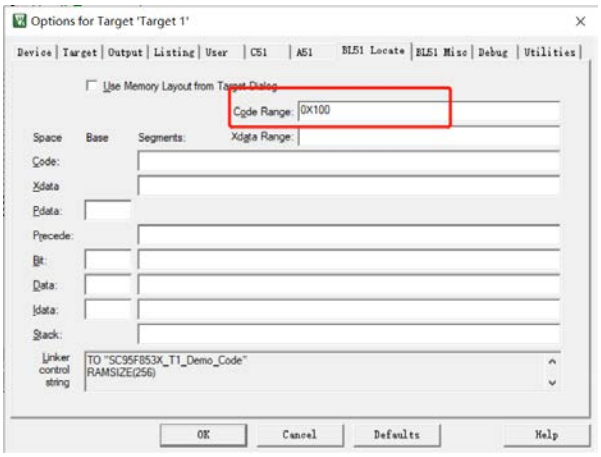
注意事项：

1) TK0~TK31 为触控按键通道；具体细节可参见 SC92_93F8XXX 规格书。
2) 用户在实际应用过程中，可再次消抖，以增强可靠性；如连续 2~5 次读到该键值，才有效。
3) 也可通过读取键值的方式来判断：
   ① 双击：如 1S 内有 2 次按键；
   ② 长按键：如持续按键 2S；

7. 复制“lib”文件夹至工程文件夹内

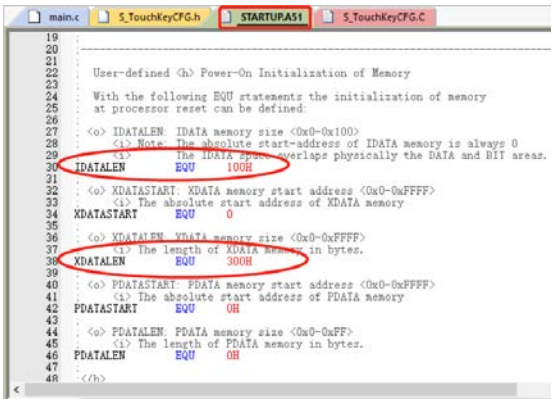


8. 在 Keil 中打开工程项目文件；注意设定（Code range）、设定（XDATALEN EQU xxxH）



设定的目的，参见赛元 MCU 应用注意事项 Vx.xx.PDF 文件。
（部分 92/95F 系列芯片不用设置该项，请仔细阅读）

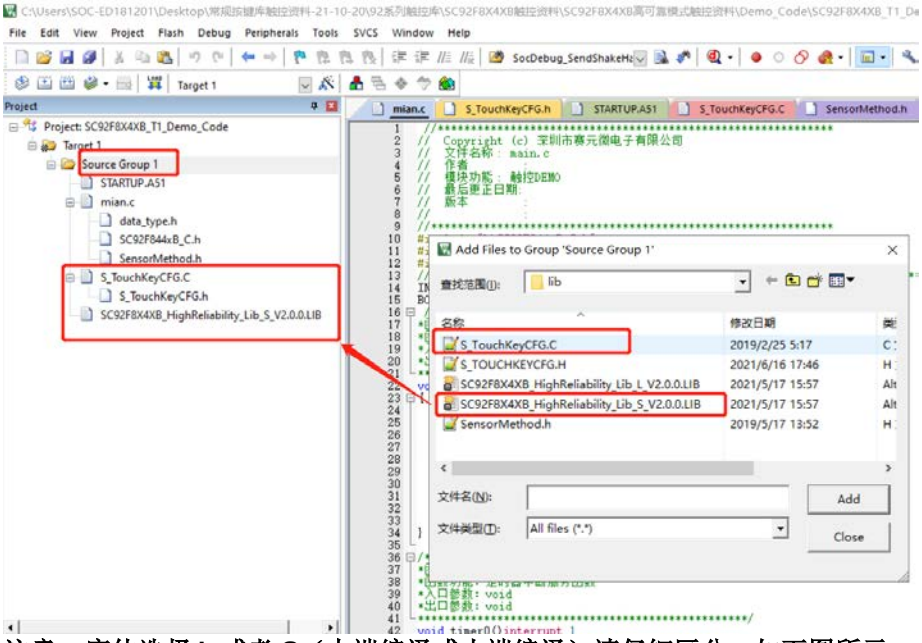
9. 设定 XDATALEN



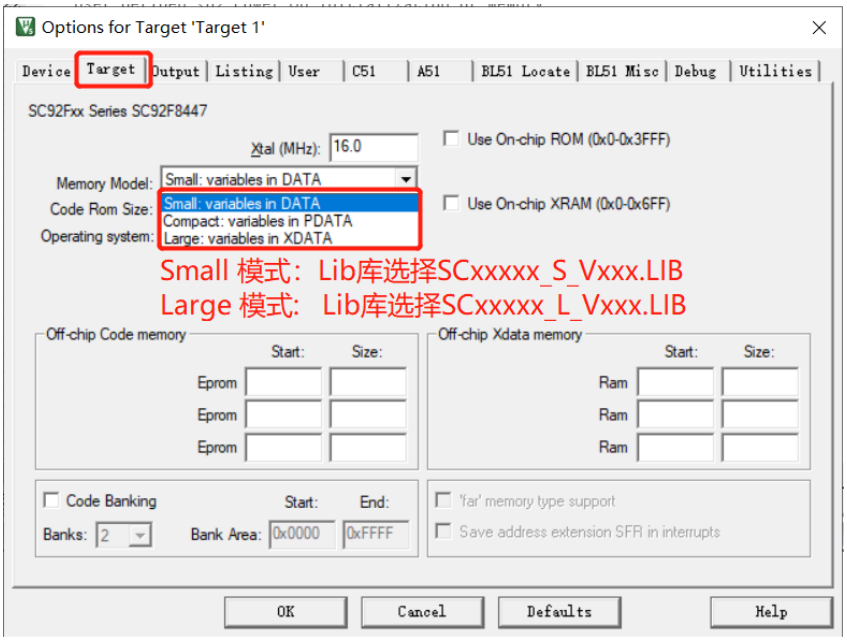
注意：用于 STARTUP.A51 中，清掉外部 XData，具体型号的 XData 大小见规格书。

10. 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。



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：



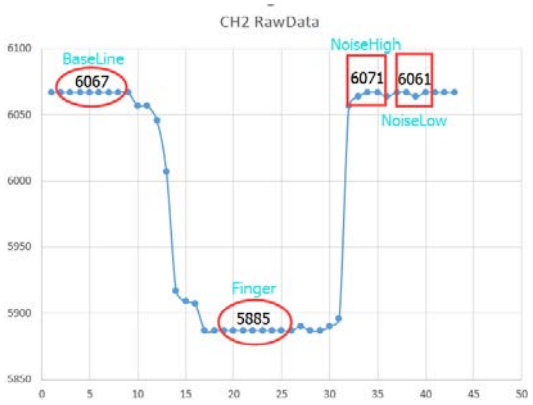
11. 在主程序文件中添加头文件引用

12. 在 S_TouchKeyCFG.h 中修改参数（调试高可靠触控参数步骤中记录的 TKCFG1、TKCFG2 和 TKCFG3 的参数） 以及触控按键的通道和个数；设定触控按键有效的次数

```
//=====================================================================
//TKCFG1:
// bit[4] =    PRS             调频开关
// bit[3:0] =  Not define      未定义
//TKCFG2:
// bit[5:4] =  CMPFLTS         滤波设置
// bit[3:0] =  CTIME           充放电频率
//TKCFG3:
// bit[6:4] =  SVSS            充电稳压源电压
// bit[3:0] =  VREF            参考电压
//---------------------------------------------------------------------
#define         SOCAPI_SET_TKCFG1                   0x00  //默认设置: 0x00//SFR: TKCFG1配置: bit7-bit4(PRS);bit3-bit0(NULL)
#define         SOCAPI_SET_TKCFG2                   0x13  //默认设置: 0x15//SFR: TKCFG2配置:  bit7-bit4(CMPFLTS);bit3-bit0(CTIME)
#define         SOCAPI_SET_TKCFG3                   0x36  //默认设置: 0x36//SFR: TKCFG3配置:  bit7-bit4(SVSS);bit3-bit0(VREF)
//---------------------------------------------------------------------

//触控按键的个数，通道设置，每bit控制一个通道
#define         SOCAPI_SET_TOUCHKEY_TOTAL           7       //用户实际使用的按键通道的数量,如用户使用TK8~TK15共8个键,只填8;
#define         SOCAPI_SET_TOUCHKEY_CHANNEL         0x2A9A0000  //bit15~bit0对应TK15~TK0;  对应为置1则为TK，对应位置0则为IO

//触控按键的程序检测确认次数
#define         SOCAPI_SET_TOUCHKEY_CONFIRM_CNT     15      //确认按键次数(4~30之间，检测次数越大，反应越慢)

//触控按键的噪音值
#define         SOCAPI_SET_NOISE_THRESHOLD          30      //设置噪音阈值范围:16~40

//每一路通道触控按键的手指阈值，范围0~65535，此为有效差值=（baseline-Finger）*0.6，数值越大，灵敏度越低，反之亦然。
//baseline为手指没按下的rawdata值,Finger为手指按下后的rawdata值
//乘以0.6是留点余量，因为每个人的手指接触面积不一样,用户也可根据触摸效果适当的增加或减小。
//用户只需要设置实际使用的通道手指阈值，其余没用到的通道可以随机数;

#define         SOCAPI_KEY0_FINGER_THRESHOLD        1000
#define         SOCAPI_KEY1_FINGER_THRESHOLD        1000
#define         SOCAPI_KEY2_FINGER_THRESHOLD        1000
#define         SOCAPI_KEY3_FINGER_THRESHOLD        1000

#define         SOCAPI_KEY4_FINGER_THRESHOLD        1000
#define         SOCAPI_KEY5_FINGER_THRESHOLD        1000
#define         SOCAPI_KEY6_FINGER_THRESHOLD        1000
#define         SOCAPI_KEY7_FINGER_THRESHOLD        1000
```

13. 根据调试步骤中的 RAW DATA 数据，计算出噪声阈值和手指阈值，在 S_TouchKeyCFG.h 中修改

1) 计算手指阈值和噪声阈值



计算手指阈值：

① 无按键时 Baseline 基线平均为 6067；有按键时 Finger 平均为 5885；

② 数据变化为 Baseline-Finger=6067-5885=182；

③ SOCAPI_KEY3_FINGER_THRESHOLD：Baseline-Finger；故这个按键 SOCAPI_KEY3_FINGER_THRESHOLD： 理论值：182；

④ 考虑到手指接触面问题，建议，理论值*0.6，即有效手指阈值为 182*0.6=109；

计算噪声阈值：

① 无按键时 Baseline 基线平均为 6067；有按键时 Finger 平均为 5885；

② 无按键时峰值 NoiseHigh=6071,NoiseLow=6061

③ 数据变化为 6071-6061=10；SOCAPI_SET_NOISE_THRESHOLD：10；

④ 采集所有触控通道的噪声阈值,取最大值作为所有通道的噪声阈值，一般设置为 20~40 之间。

2) 在 S_TouchKeyCFG.h 中修改噪声阈值和手指阈值

到此，完整的赛元触控高可靠软件库已添加至项目工程内。

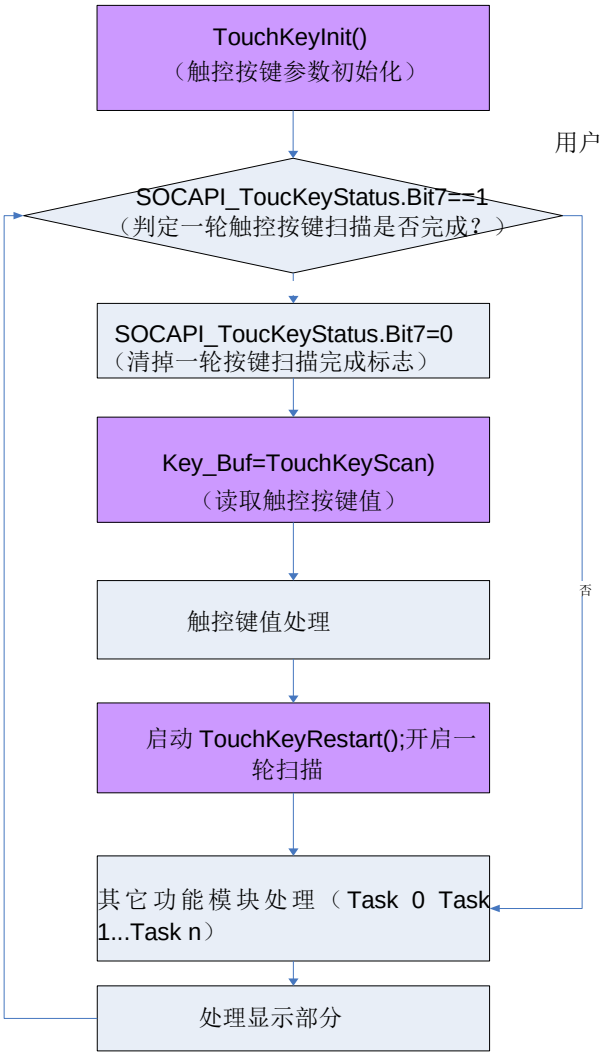
注意：应用程序中需要将 TK 对应的 IO 口设置为强推挽输出高。

3.4 完成用户程序和赛元触控软件库的融合

3.4.1 高灵敏库触控软件和用户程序

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控按键的功能。
   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 库函数只管触控按键功能，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) 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，便可得到触控按键的键值。

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

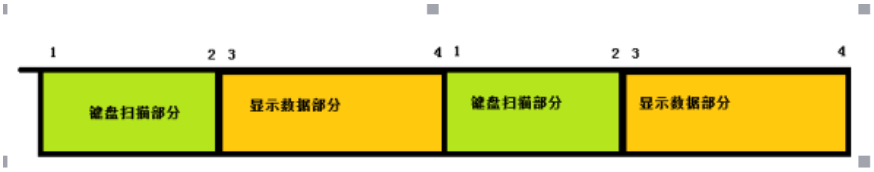
② 主程序调用接口函数“TouchKeyInit()”用于配置触控按键通道的参数，并初始化 Baseline 基线；

③ 若按键个数大于 8 个，主程序通过查看全局变量 SOCAPI_TouchKeyStatus&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。

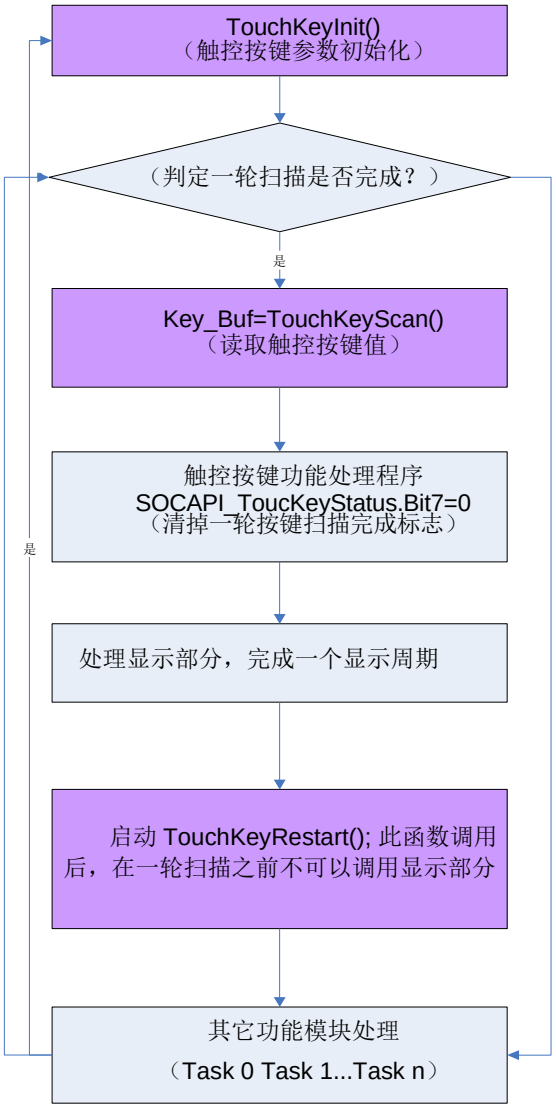
④ 主程序通过查看全局变量 SOCAPI_TouchKeyStatus&0x80 来判定一轮触控按键扫描是否完成；

⑤ 主程序调用接口函数“TouchKeyScan()”用于读取触控按键值；

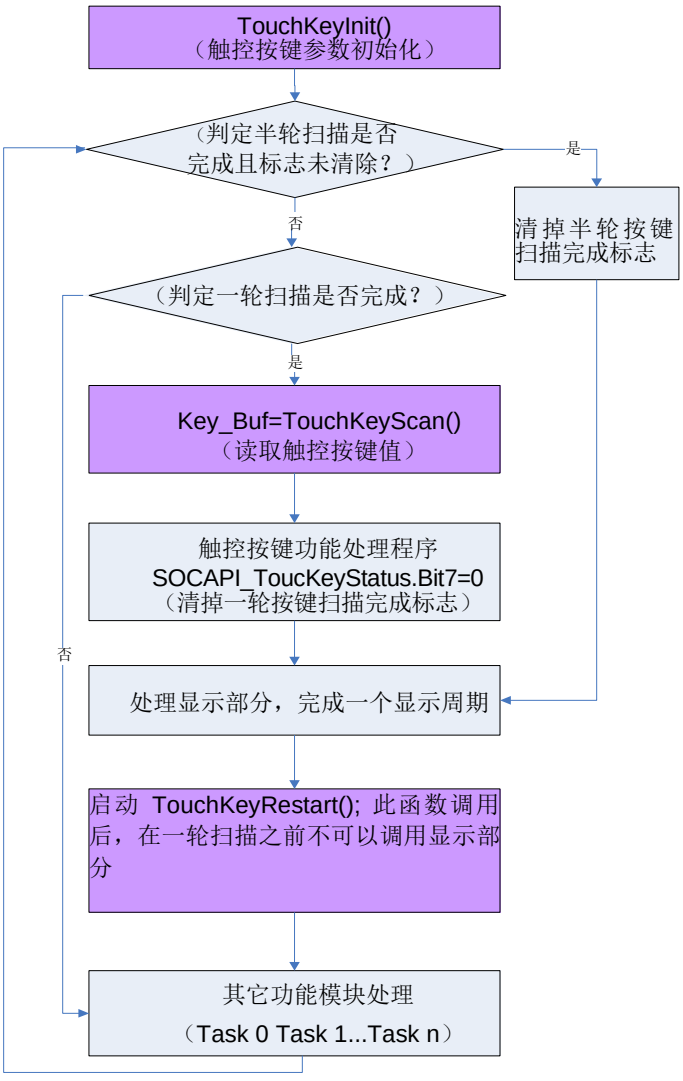
⑥ 特别需要强调一点的是：调用 TouchKeyRestart()开始扫描按键时后，一轮扫描或者半轮扫描的标志 还没有出现时， 一定不要去做显示数据的部分。



（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) 主程序和库文件的时序关系

因为运行触控按键库消耗了部分 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，不能对触控按键通道进行操作；
如触控按键通道为输出 IO；否则触控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 触控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；

软件融合的注意事项:

① 运行时间：

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 3.3 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。

3.4.2 高可靠库触控软件和用户程序

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控按键的功能。

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 库函数只管触控按键功能，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) 库文件的调用流程

用户通过一定的流程调用库文件的接口函数，便可得到触控按键的键值。

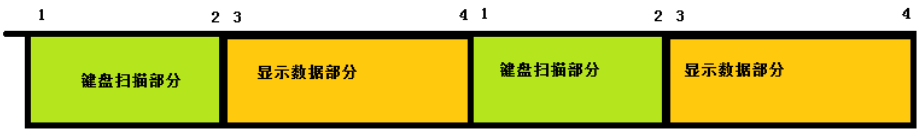
① 将 TK 对应的 IO 设置为强推挽输出高。

② 主程序调用接口函数“TouchKeyInit()”用于配置触控按键通道的参数，并初始化 Baseline 基线；

③ 主程序通过查看全局变量 SOCAPI_TouchKeyStatus&0x80 来判定一轮触控按键扫描是否完成；

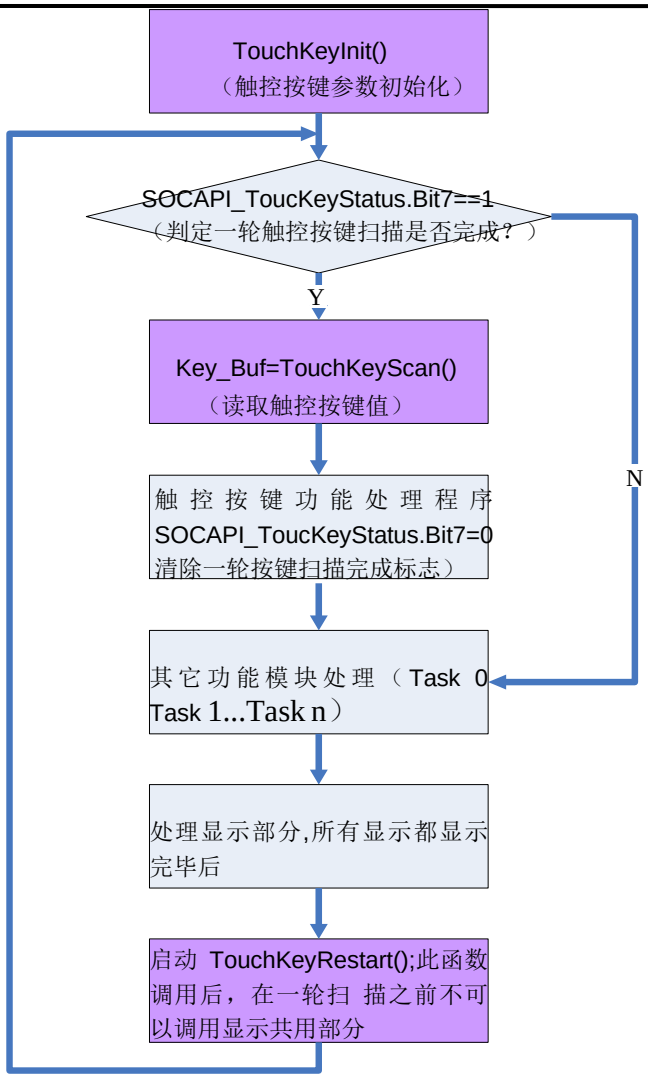
④ 主程序调用接口函数“TouchKeyScan()”用于读取触控按键值；

⑤ 特 别 需要强调一点的是：如果 用户 是 TouchKey 与 LED 共 用项目，那么在 调用 TouchKeyRestart()开始扫描按键时，在 SOCAPI_TouchKeyStatus & 0X80 的标志还没有出现 时，一定不要去做显示数据的部分。



1. 启动TouchKeyRestart() 函数， 启动键盘扫描
2. SOCAPI_TouchKeyStatus 的bit7 置起后， 表示键盘扫描一轮结束
3. 开始启动显示
4. 所有的显示完毕
1. 再次启动TouchKeyRestart(),轮回进行

（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

3) 主程序和库文件的时序关系

因为运行触控按键库消耗了部分 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，不能对触控按键通道进行操作；
如触控按键通道为输出 IO；否则触控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 触控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；

软件融合的注意事项:

① 运行时间：

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 3.3 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。**

3.4.3 注意事项

1) 使用单面 PCB 板，一般用弹簧片来做触控按键。因为其侧面也能同手指形成电场，使用弹簧片比使 用 PCB 上覆铜做触控按键能获得更高的灵敏度。
2) 从感应盘到 IC 管脚的连线长度尽量不绕太远，尽量避免连线之间的耦合电容，也要避免与其他高频 信号线有耦合电容。
3) 灵敏度与感应盘面积成正比，与外壳厚度成反比。根据外壳厚度和尺寸选择合适的触控面积。一般玻璃外壳比塑料具有更高的穿透力。
4) 感应盘与感应盘之间应该尽量留一定的间距，以保证手指头触控时不会覆盖到 2 个感应盘，同时也能 防止感应盘寄生电容过大。
5) 基准电容是赛元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，其容值范围为 472-104，推荐使用 103 电容，材质无特殊要求。
6) TK 对应的 IO 口设置为强推挽输出高。

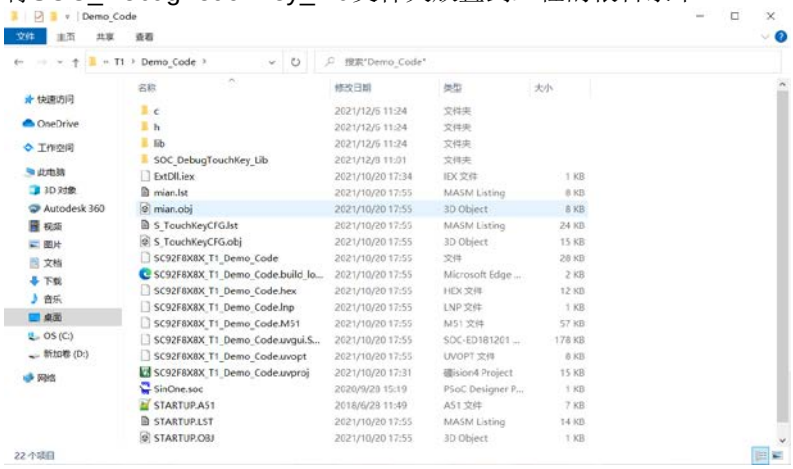
更多 Layout 注意事项请参考指南：《赛元触控按键 MCU PCB 设计要点》。

3.5 附加功能-动态调试功能

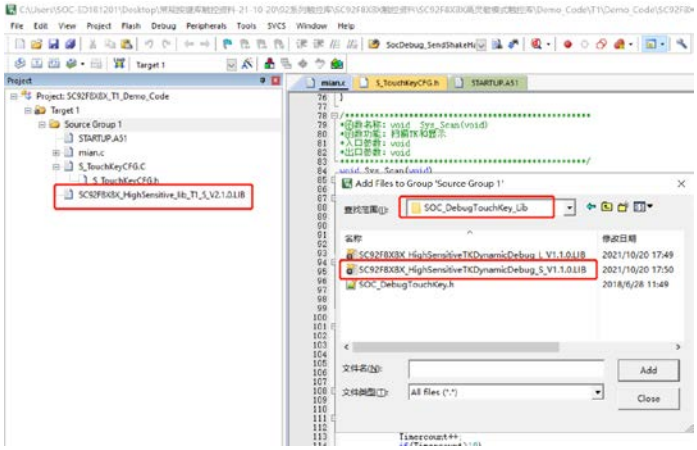
主要功能：利用赛元触控调试上位机软件查看实时的数据情况，帮助用户对系统进行整体的评估，了解系统实际运行的情况，分析异常等。

3.5.1 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) 在用户工程中加入 SC92_93F8XXX_HighSensitiveTKDynamicDebug_S/L_Vx.x.x.LIB
S/L –>指的是编译动态调试库体 lib 是用小端/大端编译，需要和触控库体 S/L 保持一致,如图所示。



3) 在 main.c 中 include 头文件

```
#define   __TOUCHKEY_DEBUG__                    //打开调试数据

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

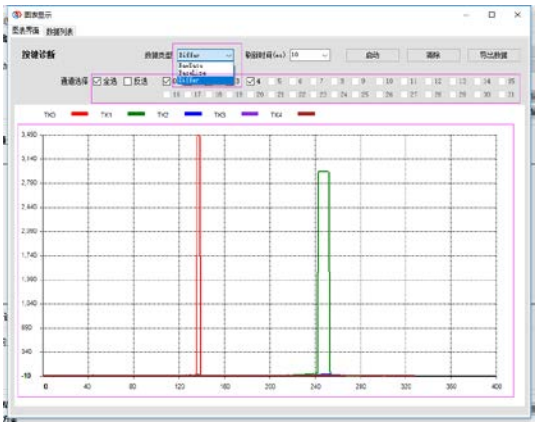
4) 在 main 函数中调用 SOCAPI_DeBugTouchKey_Init 进行初始化。编译成功后烧录到芯片内

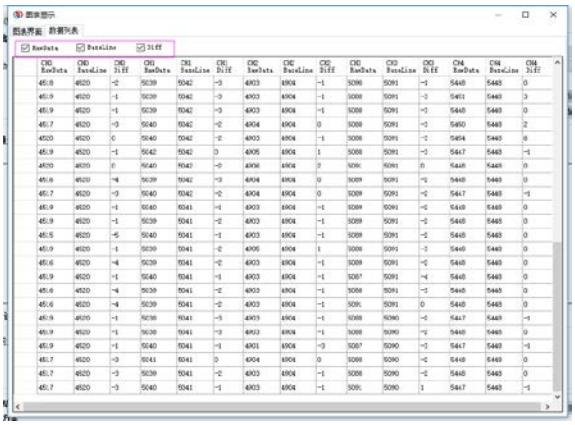
```
216 }
217
218 /*****************************************
219  *函数名称: void main(void)
220  *函数功能: 主函数
221  *入口参数: void
222  *出口参数: void
223  *****************************************/
224 void main(void)
225 {
226     Sys_Init();
227
228     #ifdef    __TOUCHKEY_DEBUG__
229     SOCAPI_DeBugTouchKey_Init();
230     #endif
231
232     //触控按键初始化
233     TouchKeyInit();
234
235     while(1)
236     {
237         WDTCON = 0x10;
238         if(TimerFlag_1ms==1)
239         {
240             TimerFlag_1ms=0;
241             Sys_Scan();
242             BuzzerWork();
243         }
244
245     }
246 }
```

5) 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**）

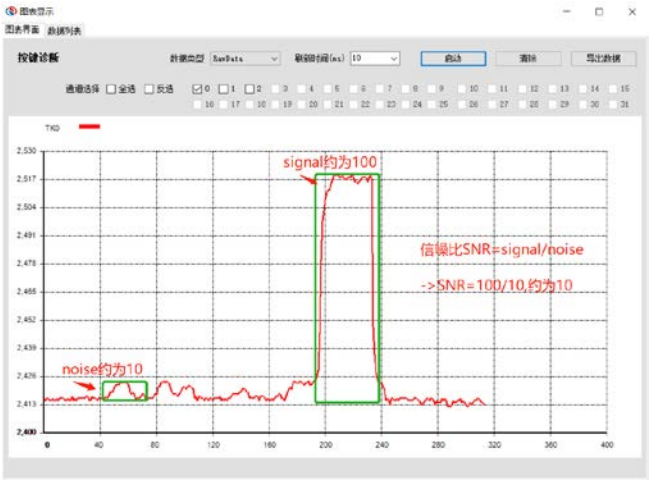
6) 点击确定按钮，然后点击下方“动态调试”按钮

7) 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



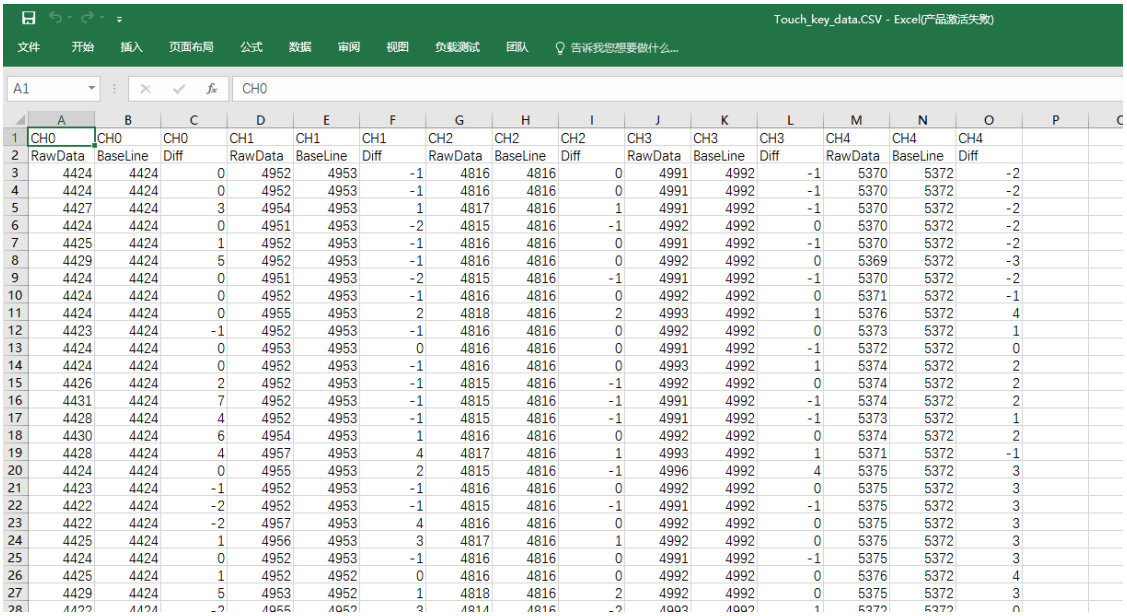


注意：动态调试观测数据时，需要注意信噪比 SNR 是否符合使用条件。
建议是 SNR > 5 可用，推荐是 SNR > 10 效果较好。



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 RawData 的最大值减去最小值的差值
signal 信号幅度为：手指按下时 RawData 的平均值
SNR 计算方式：SNR = signal/noise,如上图计算 SNR=10 左右

8) 点击“导出数据”可以将实时采集数据导出CSV格式文档



Touch_key_data.CSV - Excel(产品激活失败)

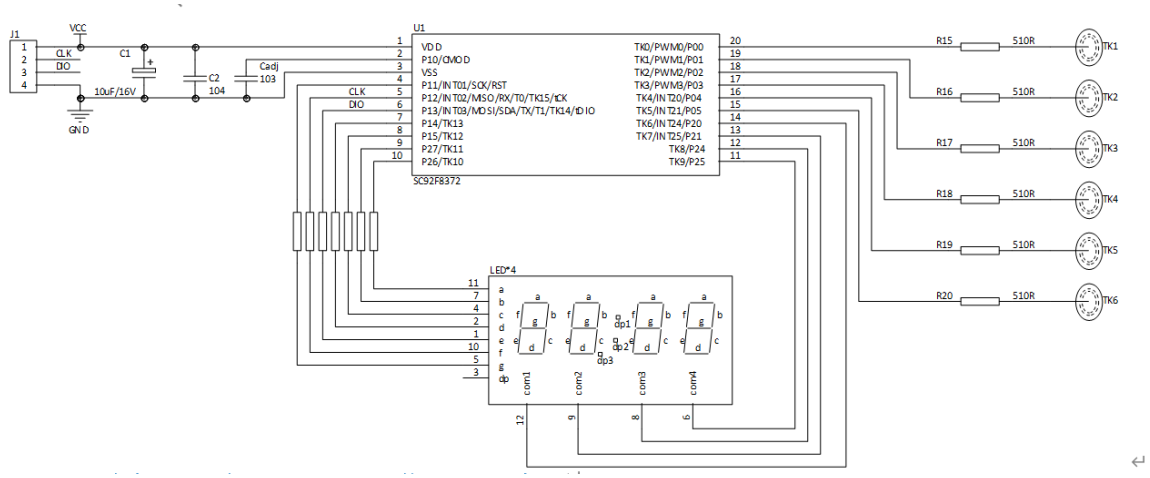
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	CH0	CH0	CH0	CH1	CH1	CH1	CH2	CH2	CH2	CH3	CH3	CH3	CH4	CH4	CH4
2	RawData	BaseLine	Diff	RawData	BaseLine	Diff	RawData	BaseLine	Diff	RawData	BaseLine	Diff	RawData	BaseLine	Diff
3	4424	4424	0	4952	4953	-1	4816	4816	0	4991	4992	-1	5370	5372	-2
4	4424	4424	0	4952	4953	-1	4816	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

9) 动态调试注意事项

- **由于动态调试库使用了烧录口上的 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，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,其中烧录口上为 UART0 的型号 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。**
- **动态调试库占用了 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。**

附录

一、应用参考原理图（以 SC92F8372 为例）



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;
    }
}
```

4 规格更改记录

版本	记录	日期
V1.6	文档格式规范化	2021 年 11 月 29 日

声明

深圳市赛元微电子有限公司（以下简称赛元）保留随时对赛元产品、文档或服务进行变更、更正、增强、修改和改进的权利，恕不另行通知。赛元认为提供的信息是准确可信的。本文档信息于 20XX 年 XX 月开始使用。在实际进行生产设计时，请参阅各产品最新的数据手册等相关资料。