目录

目录 1

1 总体描述 2

2 应用场合 2

2.1 滑轮滑条应用 2

2.2 接近感应应用 2

2.3 低功耗应用 3

3 新定义触控库简介 4

3.1 新定义触控库类型 4

3.2 新定义触控库文件结构和命名 5

3.3 库体移植使用简要步骤 5

4 库体移植说明 6

5 滑轮滑条库使用说明 8

5.1 滑轮滑条参数 8

5.1.1 库体类型选择 8

5.1.2 配置参数 9

5.2 程序调参和调用 11

6 低功耗版本库使用说明 13

6.1 配置参数 13

6.2 可调用接口 13

6.3 程序调用 14

6.4 高灵敏度触控功耗 17

7 更改记录 18

8 声明 18

1 总体描述

本手册主要讲述 RD8 系列 Touchkey MCU 的特殊应用，特殊应用指的是触控按键的衍生应用，其中包括滑轮滑条、接近感应、低功耗等。其中滑轮滑条、接近感应仅适用于高灵敏模式。

为了能让用户快速开发，新定义提供了滑轮滑条库、接近感应库、低功耗库，用户在开发过程中直接调用库接口即可实现相应的功能，本手册中将会介绍库体的使用方法。

2 应用场合

2.1 滑轮滑条应用

滑轮滑条可以通过电容感应获取手指的触碰位置，且感应盘可以直接集成在 PCB 上，进而节省成本和降低组装难度，在整装时也不用破坏一体化结构。常见的应用场景有遥控器、电磁炉等，在电磁炉应用中可用滑轮滑条调节火力，对比传统的加减按键滑动调节更加直观、美观。

2.2 接近感应应用

电容式接近感应可以代替红外(IR)传感器等用于自动开门器、控制面板背光开关、电子锁唤醒、毛巾机、干手机、肥皂液分配机等；电容式接近传感器具有以下优势：

1. 与 IR 接近感应相比它是一个低成本的解决方案。电容式接近感应的传感器可以使用 PCB 上的铜皮构建，而红外接近传感器需要额外的红外传感器且不受环境光影响，电容式接近感应可以应用在不同的光环境；
2. 电容式接近感应可以在覆盖材料上不需要任何开孔用于接近检测。因此电容式接近传降低了加工成本，提高产品的美观度。

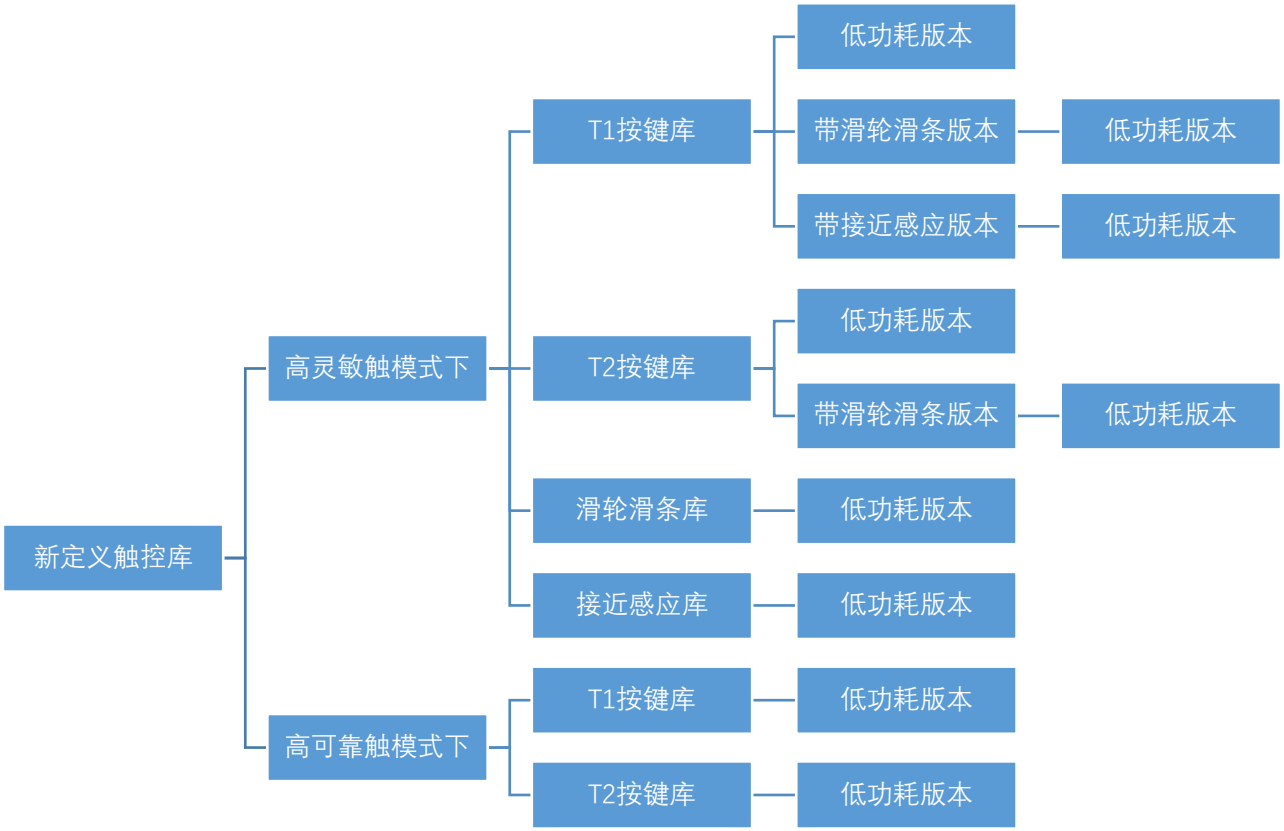
2.3 低功耗应用

在智能门锁等带电池设备中，往往待机时间是考量一个产品好坏的关键，在常规应用中 MCU 功耗都在 mA 级，其严重影响了电池设备的待机时间。新定义为 Touchkey MCU 提供了一个低功耗构架，在这个构架上开发的程序在待机时功耗可以控制在 uA 级，其大大增加了电池设备的待机时间。

3 新定义触控库简介

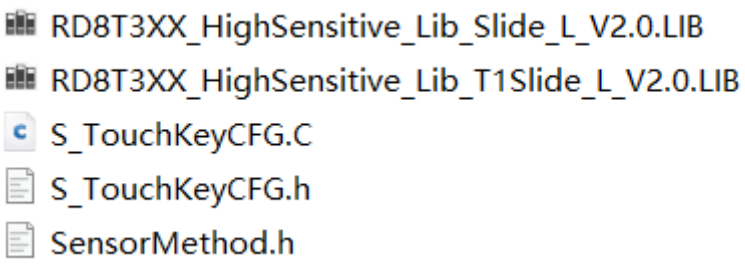
3.1 新定义触控库类型

新定义触控库分为高灵敏、高可靠两大类（目前只有高灵敏），在这两大类之下又衍生有 T1、T2、滑轮滑条、接近感应库，
而各类库还有低功耗版本。库衍生结构如图：



本文只介绍滑轮滑条、接近感应、低功耗等特殊应用。

3.2 新定义触控库文件结构和命名



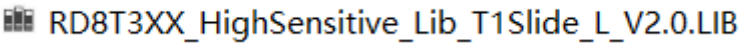
新定义触控库中通常包含：

S_TouchKeyCFG.C：触控配置文件，里面包含各项配置参数；

S_TouchKeyCFG.H：解控配置文件的头文件，由触控调试软件 Touch Key Tool Menu 或者 EasyCodeCube（下载连接: https://www.rdsmcu.com ）调试生成；

SensorMethod.h：包含用户调用接口，是 LIB 的头文件；

LIB 文件:这是核心算法的封装库，库文件命名代表库中的含义，LIB 与 S_TouchKeyCFG.c、SensorMethod.h 文件配套，不能任意替换。



命名格式包含型号、TK 电路工作模式、算法类型、编译模式既 Keil C51 中的 Memory Model(**_L_表示 Memory Model 为 Large，_S_表示 Memory Model 为 Small**)、版本号。

算法类型包括 T1(T1 按键)、T2(T2 按键)、Slide(滑轮滑条)、ProximityInduction(接近感应)、LP(低功耗)。

3.3 库体移植使用简要步骤

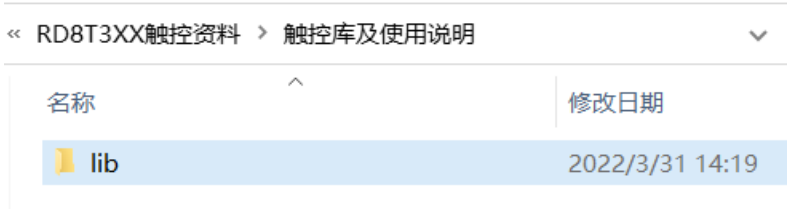
Tk 触控库的使用可以分为以下几个步骤：

1. 触控库移植，具体步骤见 4 库体移植说明。
2. 触控调试(具体见《新定义 RD8 系列 TouchKey MCU 应用指南》)，获取和替换 S_TouchKeyCFG.H 文件。
3. 进行参数更改，参数含义以及修改点见 5 滑轮滑条库使用说明或者 7 低功耗版本库使用说明。
4. 程序调用库函数获取结果。

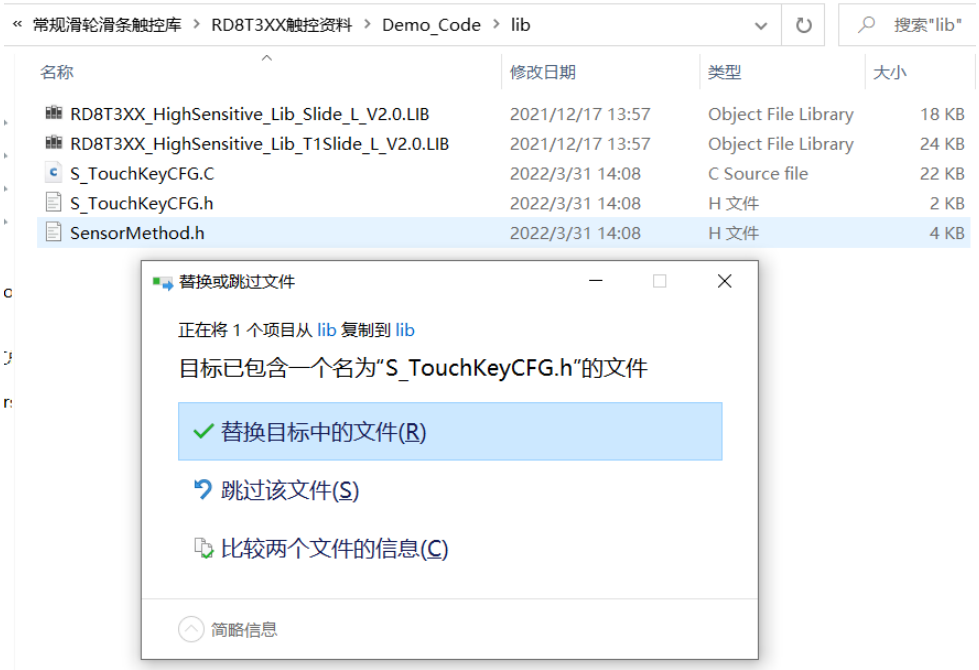
4 库体移植说明

以 RD8T37X 的滑轮滑条库移植为例，可按以下步骤将控库加入用户工程。

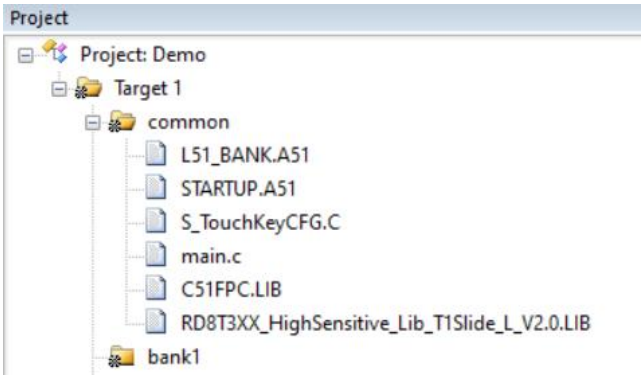
1. 将新定义提供的滑轮滑条触控库资料中的触控库及使用说明文件夹下的 lib 放置到工程的根目录下



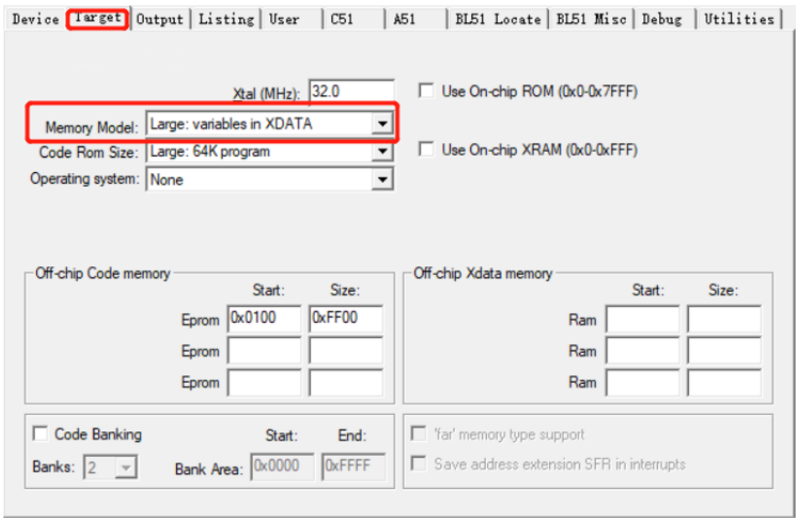
2. 将触控调试软件 Touch Key Tool Menu 或者 EasyCodeCube（下载连接: https://www.rdsmcu.com）调试生成的“S_TOUCHKEYCFG.H”替换到 lib 文件夹中,调试方法见《新定义 RD8 系列 TouchKey MCU 应用指南》。



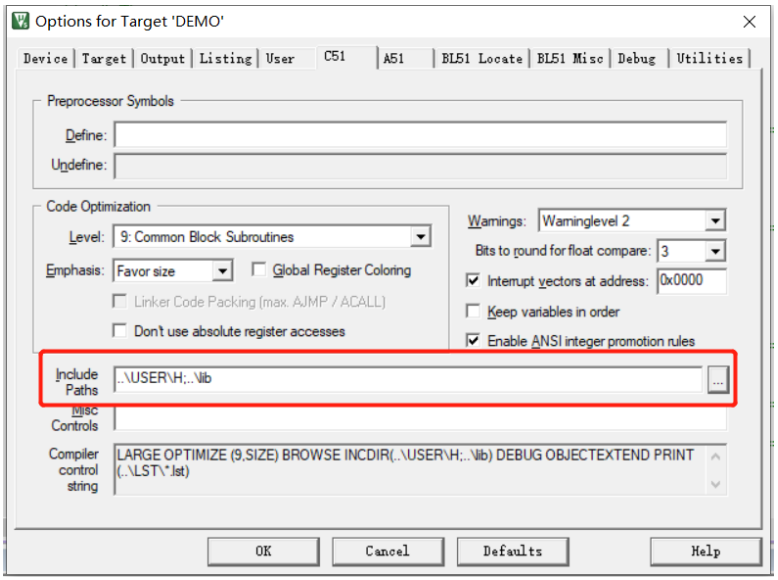
3. 将“S_TouchKeyCFG.C”和“RD8TXXX_HighSensitive_Lib_T1Slide_L_VXXX”添加到用户工程中;



4. 在Options for Target->Target中的Memory Model选择Large:variables in XDATA(如果是带_S_则选择Small:variables in DATA);



5. 在Options for Target->C51的include Path中添加lib文件夹的路径;



6. 最后一步，添加引用头文件 #include " SensorMethod.h" 。请留意你所选择的库是否与你的芯片型号匹配，如RD8T37X芯片应使用“RD8T3XX_HighSensitive_Lib_XXSlide_L_VXXX”的库.

```
1  //*************************************************************
2  //  Copyright (c) 合肥市新定义电子有限公司
3  //  文件名称：main.c
4  //  作者
5  //  模块功能：  触控Demo
6  //  局部函数列表:
7  //  最后更正日期: 2020/10/16
8  //  版本        : V1.0
9  //*************************************************************
10
11 #include "stdio.h"
12 #include "SensorMethod.h"
13
```

5 滑轮滑条库使用说明

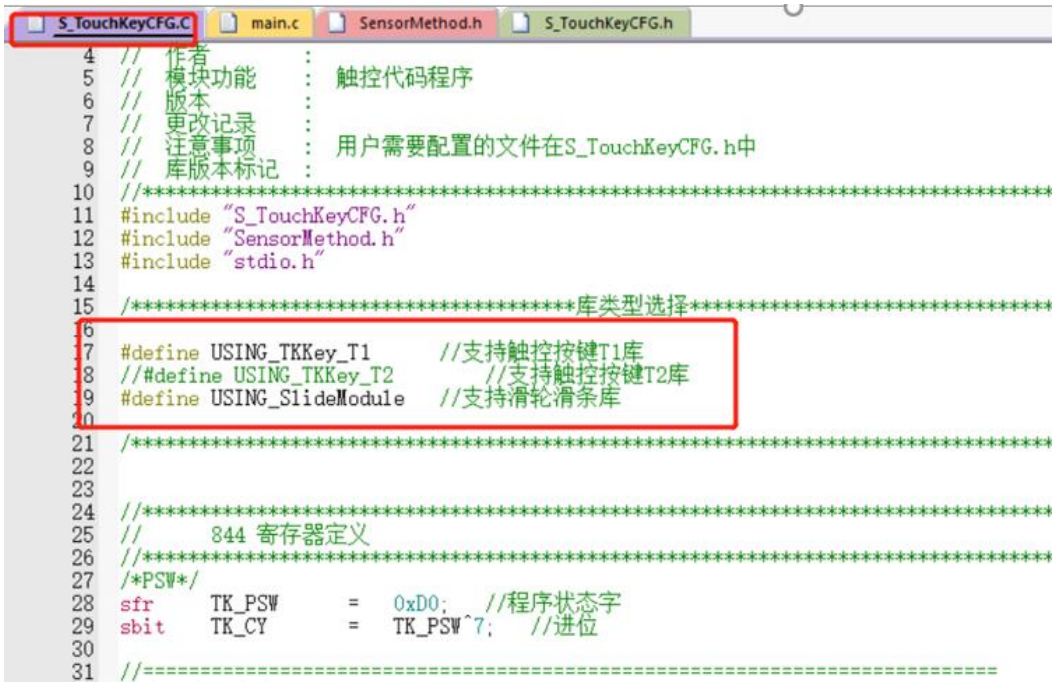
根据库体移植说明操作可以将库体移植到用户工程，本章节主要讲述滑轮滑条库的参数含义以及如何使用。

5.1 滑轮滑条参数

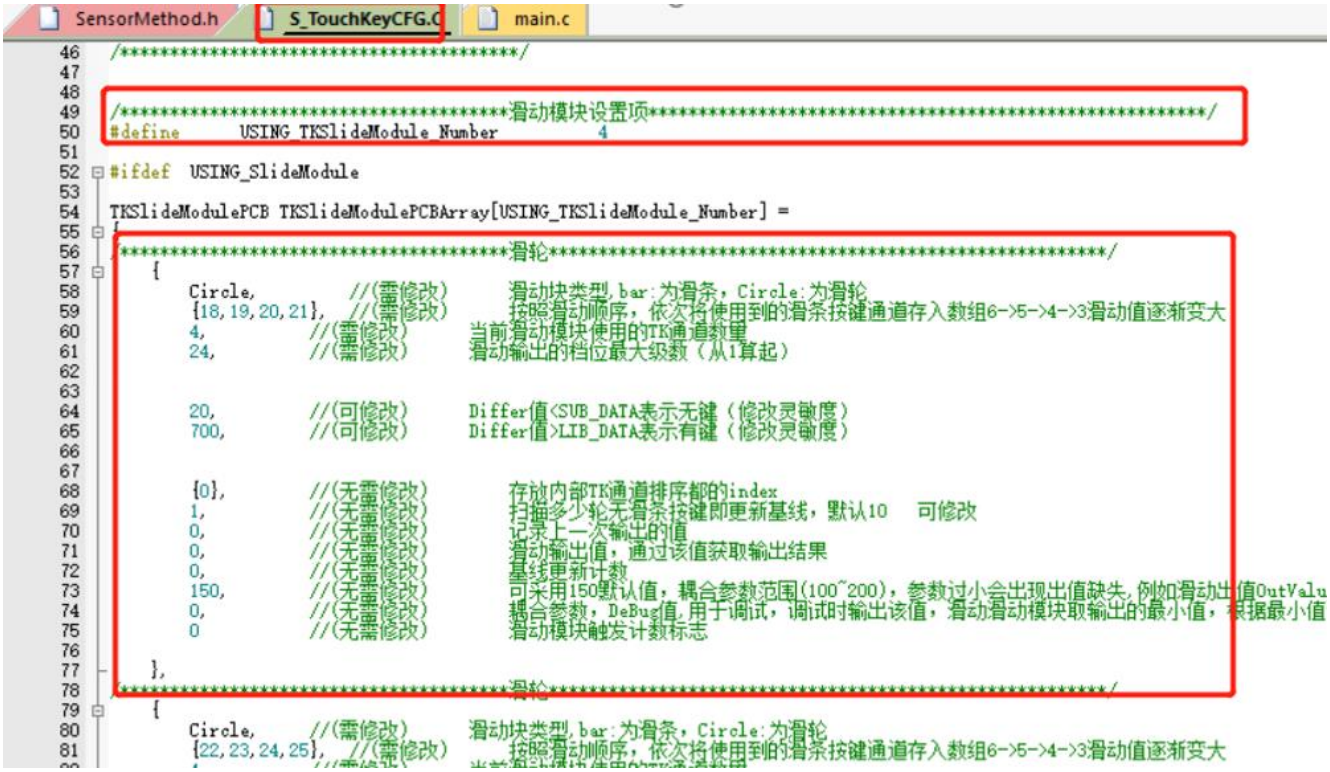
5.1.1 库体类型选择

滑轮滑调配置参数都集中在S_TouchKeyCFG文件中，在使用滑轮滑条库前需要先根据库体类型设置宏定义，具体对照关系如下表。

选用库体	库体选择宏定义
XXXXX_HighSensitive_Lib_Slide_X_VXXXX	#define USING_SlideModule
XXXXX_HighSensitive_Lib_T1Slide_X_VXXXX	#define USING_TKKey_T1 #define USING_SlideModule
XXXXX_HighSensitive_Lib_T2Slide_X_VXXXX	#define USING_TKKey_T2 #define USING_SlideModule



5.1.2 配置参数



USING_TKSlideModule_Number 参数说明：

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。

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。

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;
```

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），该值过小容易误触发，过大容易不响应。建议设置为阈值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 MAXUpdateCount 轮且无滑动则更新基线
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 表示滑动模块未被触摸
UpdateBaseLineNumber	基线更新计数	(无需修改)若无触摸，每扫描一次 UpdateBaseLineNumber 自增 1，触摸后则清 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,该值用于调试。调试时可以输出该值，滑动滑轮滑条取输出的最小值，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)滑轮滑条触发计数标志

5.2 程序调参和调用

1. 先根据移植说明移植好滑轮滑条库，再打开 S_TouchKeyCFG 文件进行库体类型选择，可选项分别为#define USING_SlideModule、#define USING_TKKey_T1、#define USING_TKKey_T2，根据所选的库题进行设置即可，详细见 5.1.1 库体类型选择；
2. 在 S_TouchKeyCFG 文件中设置滑轮滑条参数，设置滑轮滑条数量，配置 TypeFlag、TKChannel[16]、SideLevel、LIBData、CouplingValue，其参数具体含义见 5.1.2 滑轮滑条配置参数
3. 调用接口获取滑轮滑条值，如图所示最先需要调用 TouchKeyInit()函数进行初始化，然后在主循环中调用 Sys_Scan()函数进行扫描且对滑轮滑条值进行处理

```
/*****************************************************
*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
    }
}
```

4. 在 Sys_Scan 中，先判断扫描是否完成，如果扫描完成则清除完成标志位，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（X 是数组元素序号，指定取哪个滑轮滑条的值））。

```
/*****************************************************
*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，是否
     {
        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
    }
}
```

5. 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（X 是数组元素序号，指定取哪个滑轮滑条的值）

```
/*****************************************************
*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)

6 低功耗版本库使用说明

低功耗触控库是基于通用的触控库增加了低功耗的实现方法，按键的调试过程和通用库的方法一致，具体参照《新定义 RD8 系列 TouchKey MCU 应用指南》，在函数调用接口上，低功耗触控库较通用库增加了 7 个函数接口和低功耗下外部中断唤醒处理接口，调用过程和通用库稍有差异。

6.1 配置参数

参数名	参数定义	设置说明
TK_LowPowerMode	TK 低功耗模式使能开关	不可以超过正常扫描的按键个数，按键个数越多，功耗越大
WakeUpKeyNum	低功耗模式唤醒按键数	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	低功模式唤醒按键对应的通道	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	低功耗下确认按键次数	可以设置 10-20

6.2 可调用接口

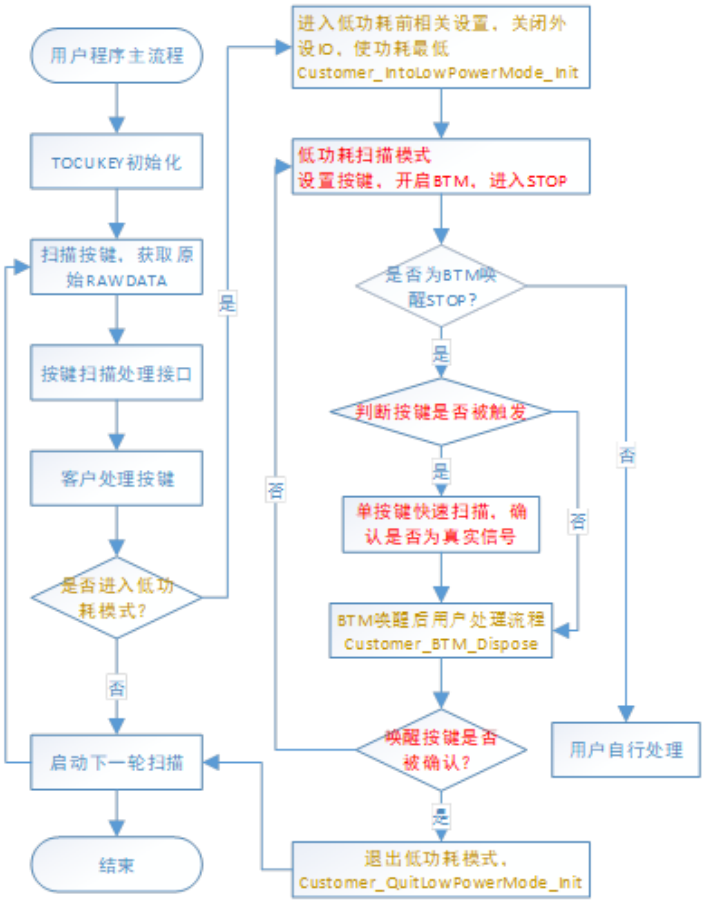
接口名	接口定义	调用说明
GetLowPowerScanFlag	获取低功耗模式标志：0 为正常运行模式，1 为低功耗模式	在主循环内区分两种模式，执行不同的程序分支
TouchKey_IntoLowPowerMode	进入低功耗模式	调用该函数进入低功耗模式。**用户根据实际应用决定调用**，例如，一段时间未有按键按下进入低功耗，调用该函数。但需注意，**进入低功耗要在无按键的情况下进入(即返回的键值为 0)**
Customer_IntoLowPowerMode_Init	用户进入低功耗模式前的初始化	**该函数用户需要自行添加在主程序中，并对其进行实现，无需调用**
LowPower_Touchkey_Scan	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_QuitLowPowerM	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**

ode_Init		**对其进行实现，无需调用**
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

Customer_IntoLowPowerMode_Init、Customer_QuitLowPowerMode_Init、Customer_BTM_Dispose 需要用户添加在文件中并选择性实现。

6.3 程序调用

程序流程图如下：



新定义 TK 触控低功耗是利用 BTM 和 STOP 来实现，在低功耗模板框架中有两个组成部分，分别是正常模式和低功耗模式，该运行那个模式是通过 GetLowPowerScanFlag 标志决定，其中正常模式是用户编写，在正常模式中用户要设置进入低功耗模式条件，满足条件后调用 TouchKey_IntoLowPowerMode 函数即可进入低功耗模式，低功耗模式下的运行逻辑无需用户编写，在低功耗模式中会进入 STOP 以降低功耗，在 STOP 期间通过 BTM 唤醒定期扫描 TK 通道，当检测到有按键触发时就会回到正常工作模式。

1. 打开 S_TouchKeyCFG 文件进行参数配置，具体参数含义见 7.1 配置参数。

```
#define     TK_LowPowerMode

#ifdef  TK_LowPowerMode

#define     BTM_TIMEBASE_15600US    0X00    //低频时钟中断时间为15.6MS
#define     BTM_TIMEBASE_31300US    0X01    //低频时钟中断时间为31.3MS
#define     BTM_TIMEBASE_62500US    0X02    //低频时钟中断时间为62.5MS
#define     BTM_TIMEBASE_125MS      0X03    //低频时钟中断时间为125MS
#define     BTM_TIMEBASE_250MS      0X04    //低频时钟中断时间为250MS
#define     BTM_TIMEBASE_500MS      0X05    //低频时钟中断时间为500MS
#define     BTM_TIMEBASE_1S         0X06    //低频时钟中断时间为1S
#define     BTM_TIMEBASE_2S         0X07    //低频时钟中断时间为2S
#define     BTM_TIMEBASE_4S         0X08    //低频时钟中断时间为4S

#include <intrins.h>

#define     WakeUpKeyNum                12                  //低功耗模式下扫描按键个数
#define     WakeUpKeyChannel            0x00000FFF          //低功耗下扫描按键的对应通道
#define     TK_SeepTimervSetting        BTM_TIMEBASE_500MS  //低功耗下按键之间的扫描间隔
#define     TK_WakeUpConfirmTouchCnt    15                  //低功耗下确认按键次数
```

2. 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
    LED =  LED_OFF;
    Buzz =0;
}
/*************************************************
*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
}
```

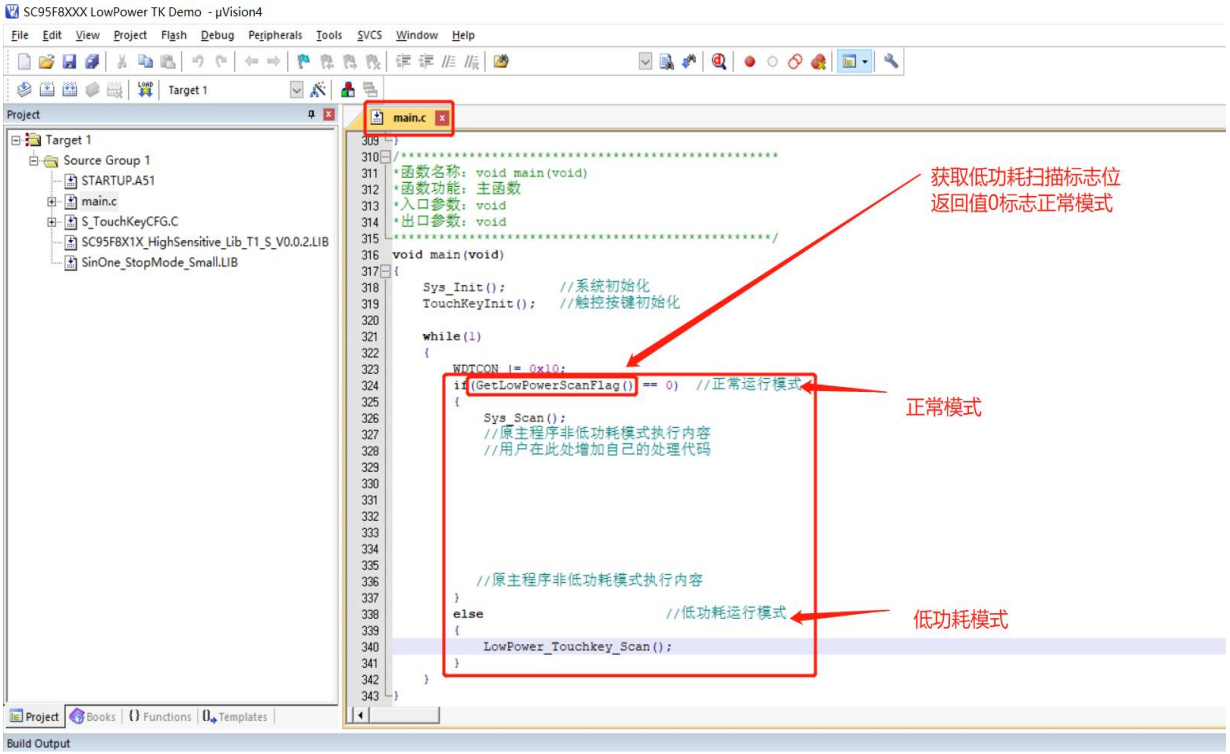
```
/*****************************************************
*函数名称：void Customer_BTM_Dispose(void)
*函数功能：用户BTM唤醒后自己的处理函数
*入口参数：void
*出口参数：void
*备注说明：低功耗实现利用了BTM资源，用户需要在BTM中实现的内容在此函数实现
          该函数已在S_TouchKeyCFG.c文件LowPower_Touchkey_Scan()函数中调用
*****************************************************/
void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

}
```

3. 在 main 函数中添加低功耗函数框架，该框架主要对低功耗模式和正常模式进行分流执行， 其 中 通 过 GetLowPowerScanFlag 判定该执行低功耗模式还是正常模式，低功耗分支 LowPower_Touchkey_Scan() 函数直接调用即可，在正常模式则进行常规逻辑，示例代码中在正常模式中调用 Sys_Scan()扫描按键处理键值。



4. 在 Sys_Scan()函数中主要的工作为按键扫描和处理按键数据，实际逻辑可由用户自行编写，示例代码中一段时间有触发按键则调用 TouchKey_IntoLowPowerMode 进入低功耗模式，在实际应用中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即可，**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。

5. 低功耗模式下 BTM 中断及外部中断处理

6.4 高灵敏度触控功耗

高灵敏度低功耗采用一次 BTM 唤醒扫描所有唤醒 TK 按键的方法实现低功耗模式 TK 按键扫描，设置的 BTM 定时时间越短，唤醒按键个数越多，工作电流越大。如果想要更低的功耗可以将 BTM 设置为 500ms 但是会牺牲响应速度，如果想要更快的响应速度可以将 BTM 设置 250ms 或者 125ms 但是会牺牲一些功耗。

7 更改记录

版本	记录	日期
V1.0	初版	2022 年 4 月

8 声明

合肥市新定义电子有限公司（以下简称新定义）保留随时对新定义产品、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。新定义认为提供的信息是准确可信的。本文档信息于 2022 年 4 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。