应用笔记

Application Note

文档编号：**AN1090**

APM32 TSC 触摸模块

版本：**V1.1**

引言

本应用笔记向用户介绍 APM32 TSC 触摸模块，包括硬件设计原理及软件设计方法，帮助用户实现按键，线性，旋转等基本功能。关于芯片和 TSC 寄存器等信息请参考官方网站的用户手册和数据手册。

本手册以 APM32F0xx 系列中的 APM32F051R8 型号进行举例说明。

目录

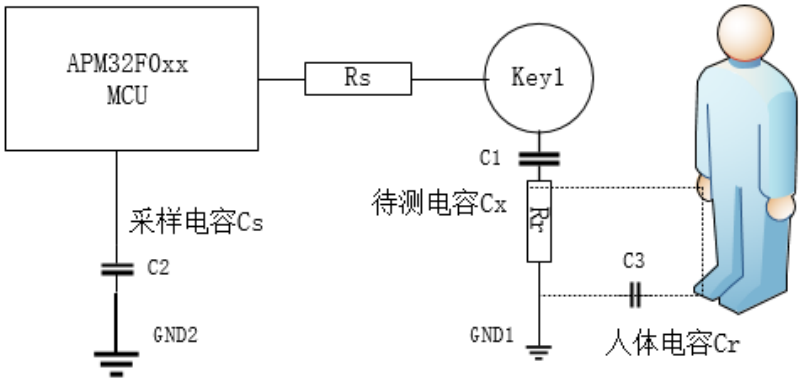
1. 简介

1.1. 术语

术语	描述
TSC	Touch sensing controller peripheral 触摸传感控制外设
Cs	Sampling capacitance 采样电容
Cx	Sensor capacitance 传感器电容
Channel	通道，基本采集端口，对应硬件 GPIO。
Block	块，同时采集的多个通道
Object	Any touch sensor (such as touchkey, linear or rotary) 触摸传感器（按键，线性，旋转）
LinRot	Linear or rotary touch sensor 线性或旋转传感器（多通道）
TouchKey	按键传感器（单通道）
Meas	Current signal measured on a channel 在通道上测量的信号值
Delat	测量值和参考值的差值
ECS	Environment change system 环境优化机制
DTO	Detection time-out 检测超时
DXS	Detection exclusion system 检测排除机制
Filter	Noise filters 噪声滤波机制

1.2. 原理

电荷转移采集原理：



当人手触摸时，环路引入Cr，使得按键对地电容增大。

图 1

1. 利用电容储存电荷的特性。
2. 电极上的待测电容 Cx 向采样电容 Cs 充电。
3. 电荷转移过程中将模拟开关做在硬件 GPIO 里。
4. 重复电荷转移的过程，直到采样电容 Cs 上的电压达到 GPIO 的门限值。
5. 达到阈值所需电荷转移次数直接表示待测电极电容的大小。
6. 电极被触摸时，传感器对地电容增大，电极中储存电荷增加（Cx），所以采样电容（Cs）电压达到 GPIO 门限值所需电荷转移次数减少，测量值变小。
7. 当测量值低于阈值时，TSC 进行触摸检测。

1.3. 功能特性

1. 基于有经验的表面电荷转移原理。
2. 每 3 个待测电容感应通道搭配 1 个采样电容，以减少系统开销。
3. 根据不同型号系列，内嵌 6~8 个模拟 I/O 组，最多支持 3*6~3*8 个通道。
4. 每个 I/O 组对应一个计数器来记录当前测量值。
5. 每个 I/O 组里的采样电容引脚和通道均可配置。
6. 软件设置最大转移次数，避免出现 DTO 状态。
7. 设置专用测量完成标志和最大次数错误标志，均可触发中断。
8. 该 TSC 支持按键，线性和旋转等感应方式。
9. 通过 DXS（检测排除）ECS（环境优化）Filter（噪声滤波）等控制来提高灵敏度和抗干扰能力。
10. 支持简单的 API 程序，供用户进行配置和查看运行状态。

模拟 I/O 组：

组编号	每组电容传感器通道数			
	APM32F051Rx	APM32F051Cx	APM32F051KxT	APM32F051KxU
G1	3	3	3	3
G2	3	3	3	3
G3	3	2	1	2
G4	3	3	3	3
G5	3	3	3	3
G6	3	3	0	0
电容传感器通道总数	18	17	13	14

2. 硬件设计

2.1. 测量电路

举例按键触摸时测量电路设计，线性和旋转同理，只是电极位置不同，如图:

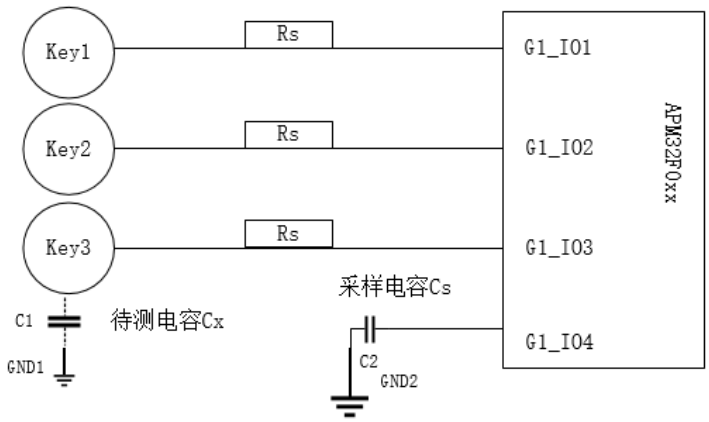


图 2

注：靠近 MCU 引脚处增加 470Ω – 10k Ω 电阻，用于 ESD 保护与抗噪滤波。每个电极上串联一个 10K Rs 用来提高 ESD 健壮性。

2.2. 介电常数

面板是手指与电极间电容介质的主要部分。其介电常数 (εr) 可用于区分面板材料。此表 2-3 表示材料内部的电场传播能力。介电常数越大，传播能力越强。

材料	εr
空气	1.00053
玻璃	4 ~ 11
蓝宝石玻璃	8 ~ 12
云母	4 ~ 8
尼龙	2.5 ~ 3.6
有机玻璃	3.3 ~ 4.5
聚乙烯	2.3
聚苯乙烯	2.45 ~ 2.6
聚酯 (PET)	3.7
FR4（玻璃纤维 + 环氧树脂）	4.2~4.5
PMMA（聚甲基丙烯酸甲酯）	2.9 ~ 4.1

2.3. 灵敏度

$$T_V = d/\varepsilon_r \quad \text{公式 1}$$

对于面板材料和厚度，其中 d 为电介质的厚度（面板厚度）。Tv 是材料的电场传导等效真空厚度。该值越小，电场就越容易穿透。对于 Tv 值相同的面板，按键的灵敏度相同。

$$Ct = \varepsilon_r\varepsilon_0(A/d) \quad \text{公式 2}$$

对于触摸传感器，Ct 为触摸电容，A 为电极相对于导电体的面积，d 为面板厚度，εr 介电常数，ε0 是真空介电常数。由上公式可见，触摸电容值与触摸相对面积大小成正比。增加电极尺寸能使 Ct 增大，但是当电极超过手指的触摸尺寸（即电极相对于导电体面积不变）只会增加寄生电容而不会增大手指触摸电容，从而导致相对灵敏度下降。

$$C=Cx+1/【(1/Ct)+(1/Cf)】 \quad \text{公式 3}$$

C 为触摸总电容，Cx 为电极的寄生电容，Ct 为触摸电容，Cf 为大地于系统地间的反馈电容。当触摸总电容增加时，电极的充放电时间增大，充放电次数增多，相对灵敏度降低。

改善灵敏度的方式：

（1）使用适宜大小的电极，电极底部尽量不铺铜。

（2）减小面板厚度。

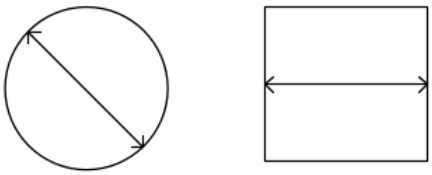
（3）选择较大值 εr 的电介质。

（4）GND 层不要离传感器太近。

（5）避免在传感器附近使用金属涂料。

2.4. 触摸按键

每个按键占用 1 个触摸通道，可以设计独立按键或者矩阵按键。



6mm（最小值）

图 3

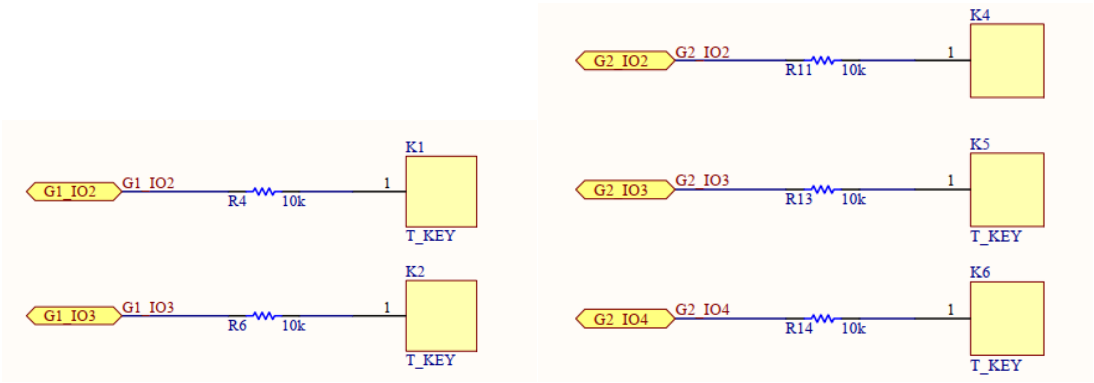


图 4（原理图示意）

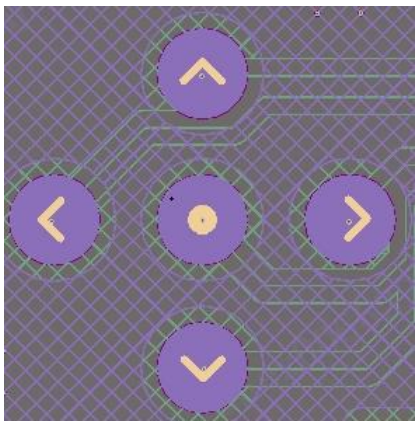
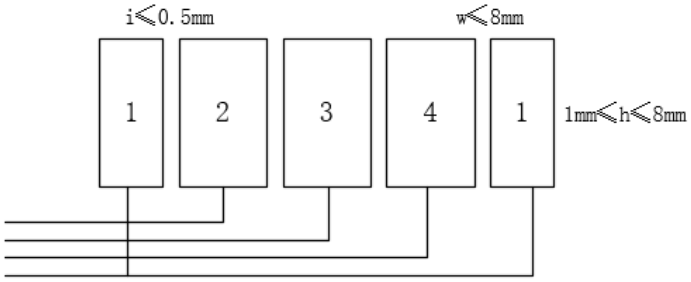


图 5（PCB 示意）

2.5. 线性触摸

对于线性触摸感应采用半通道电极方式，至少占用 3 个触摸通道，软件所支持的滑条的分辨率范围为 0 到 256。实际占用的 GPIO 数量与所选分辨率，得根据实际需求进行调整。



注：i 为两个传感器之间的间距，h 为传感器电极的高度，w 为传感器宽度。

图 6

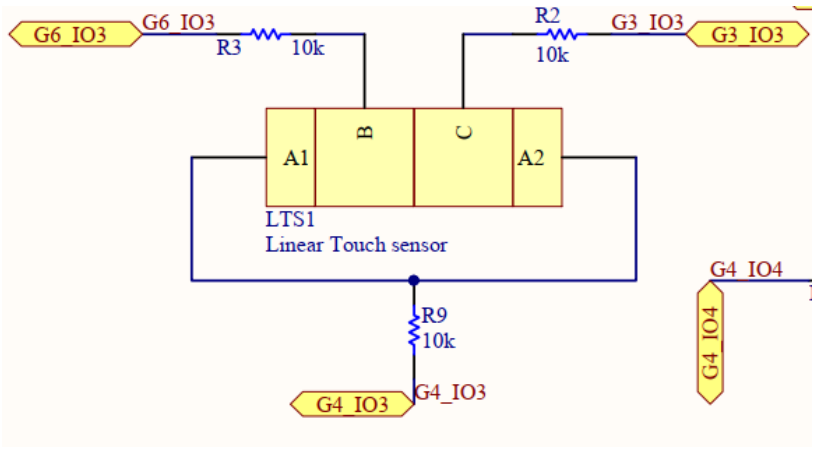


图 7（原理图示意）

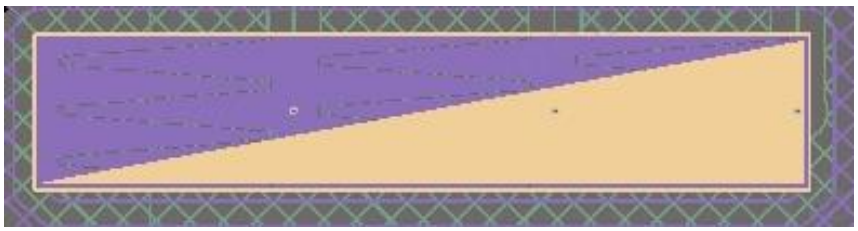
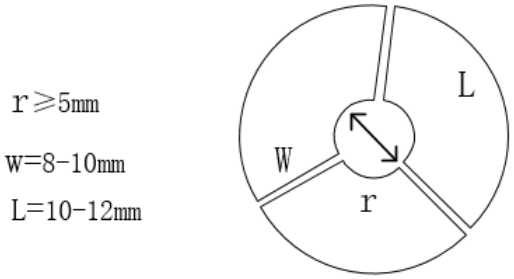


图 8（PCB 示意）

2.6. 旋转触摸

对于旋转触摸感应，采用三电极旋转，也适用于五电极和八电极旋转，但至少占用 3 个 触摸通道， 软件所支持的滑条的分辨率范围为 0 到 256。 实际占用的 GPIO 数量与所选分辨率，得根据实际需求进行调整。



注：r 为中心直径，W 为传感器电极宽度，L 为传感器电极外周长的长度。

图 9

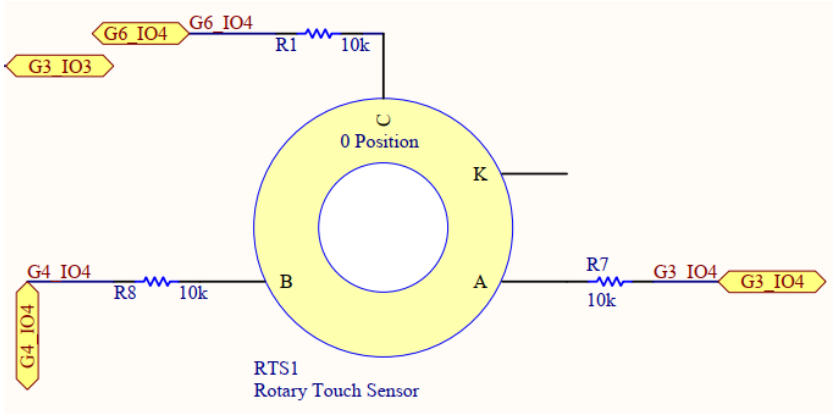


图 10（原理图示意）

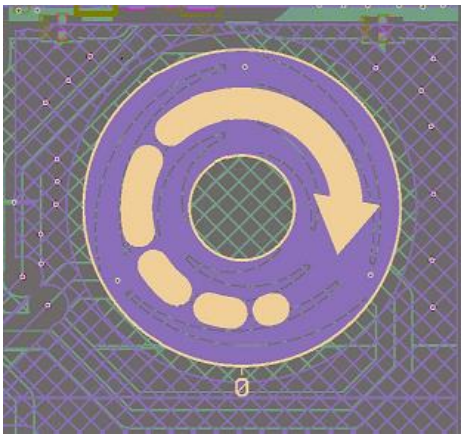


图 11（PCB 示意）

2.7. 开发板

图 12

2.8. 原理图

2.8.1. 电源电路

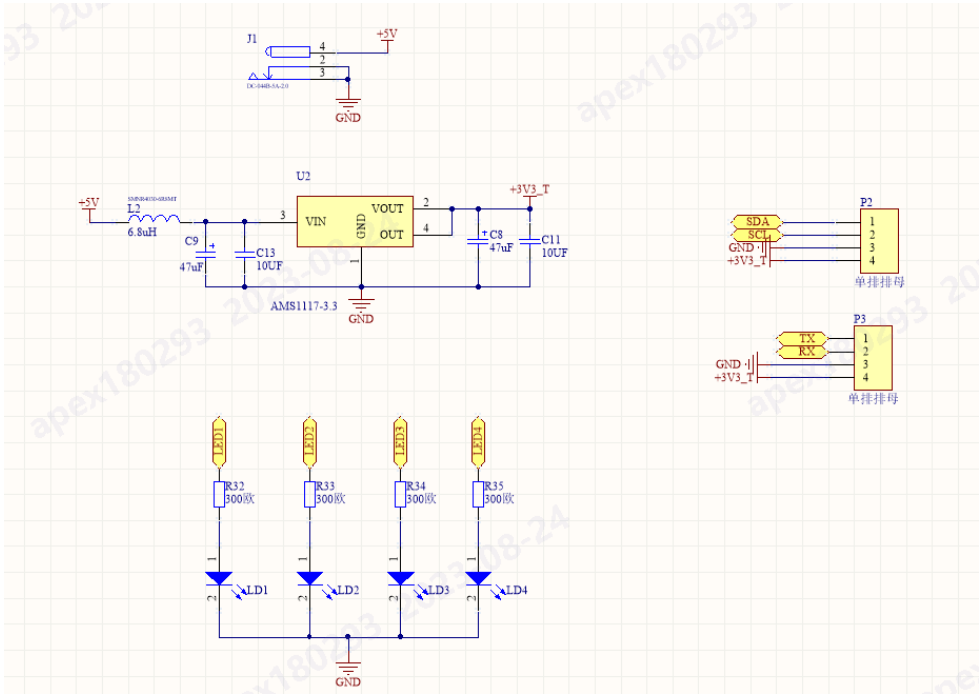


图 13

2.8.2. MCU

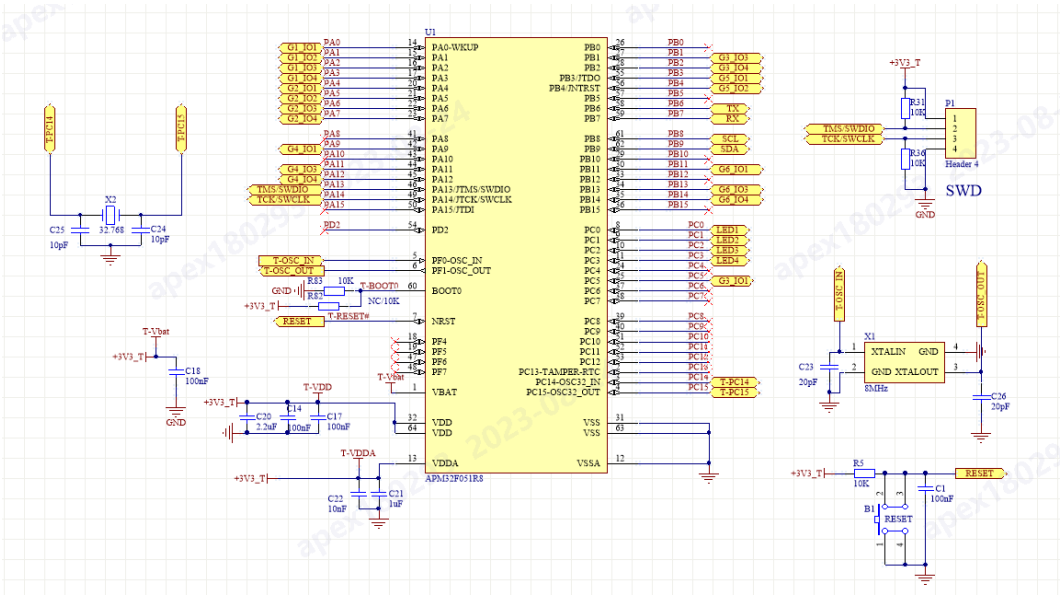


图 14

2.8.3. 触摸电路

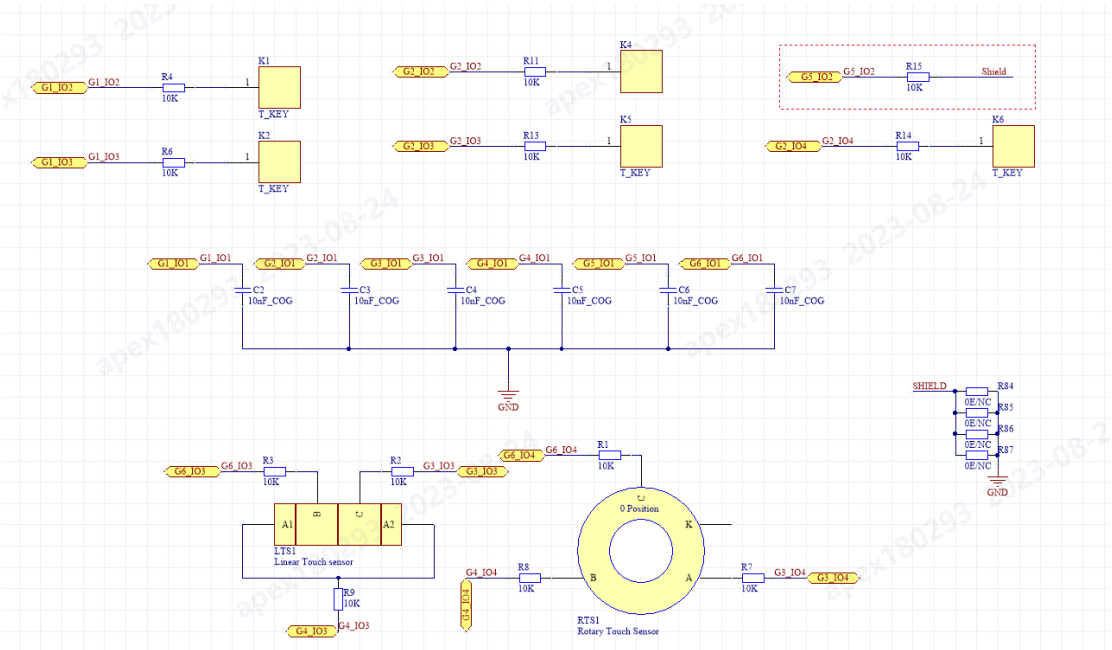


图 15

2.8.4. 设计建议

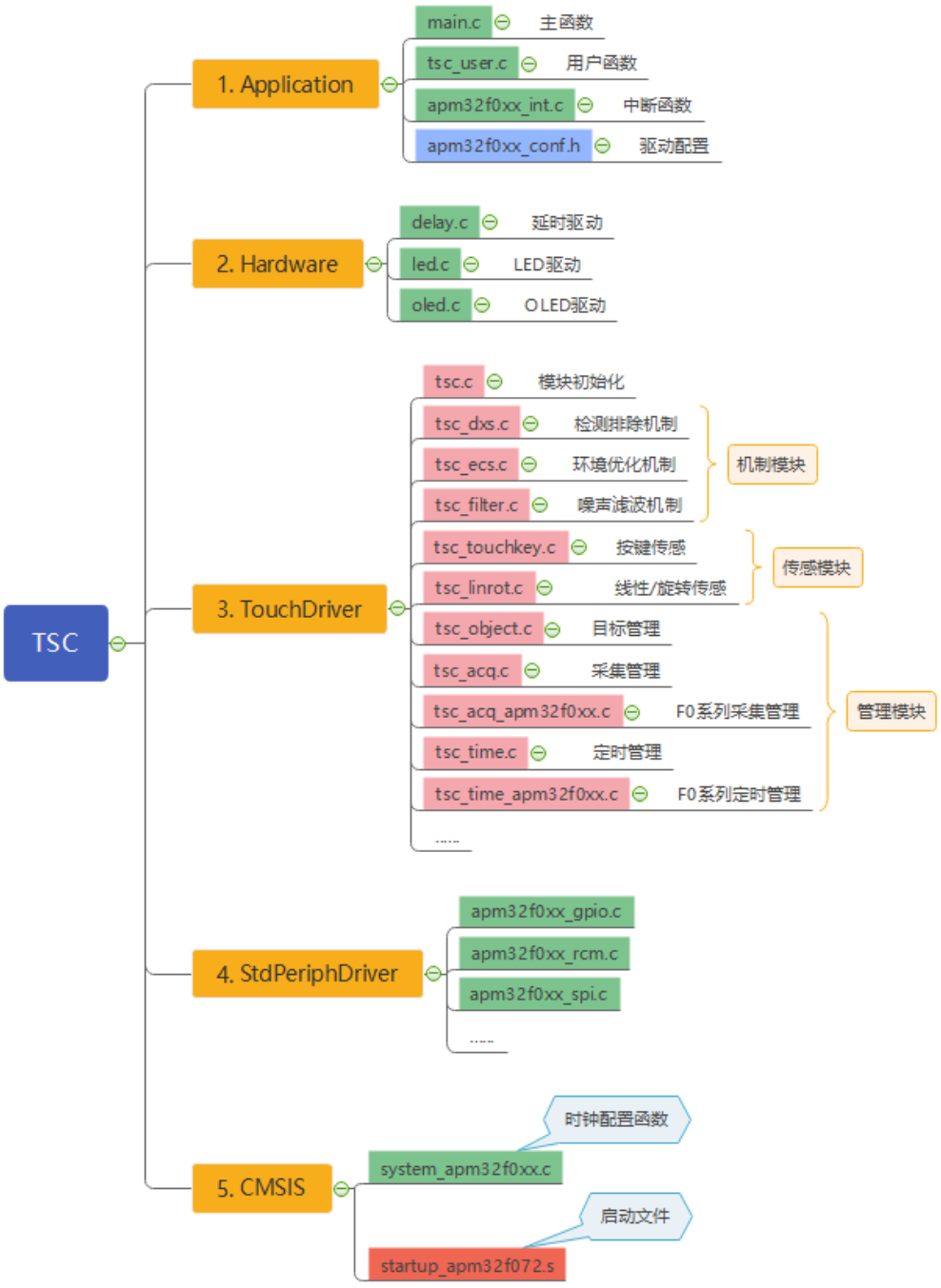
（1）使用低纹波的 LDO 给系统供电，LDO 的滤波电容建议使用大电容＋小电容，具体取值根本实际应用而定，从而提高系统的电源质量。
（2）MCU 建议独立供电，如与系统其他模块共用一组电源时，建议使用磁珠隔开。
（3）可通过调整 CS 电容以及 RS 电阻的大小来调整采样时间以及灵敏度，从而提高系统的传导抗扰能力。
（4）可以使用主动屏蔽功能，减少传感器和地面的之间的寄生电容，但是需要消耗一组模拟通道。适用于 IO 足够的情况。

2.9. PCB 布线布局

（1）触摸 PAD 应远离干扰源，例如电源输入口，高频信号，大电流回路等。
（2）IC 尽量放置在多个触摸 PAD 的中心位置，以减少各个 PAD 之间走线距离差异。
（3）触摸 PAD 底层避免布置其他元器件以及其他信号走线。
（4）触摸 PAD 底层尽量不铺铜，避免 PAD 与 GND 产生的寄生电容影响采集，同时 PAD 的同层与 GND 间距应保持 40mil 以上。
（5）触摸走线尽量短而细（建议 7-15mil），目的是减少走线带来的寄生参数影响，同时走线尽量减少过孔。
（6）触摸走线之间尽量保持一定距离，尽可能保持两倍线宽。
（7）触摸走线尽量远离其他信号线，特别是高频信号以及大电流。在没有办法避免的情况下，让两者垂直走线，如需同层走线应保持足够大的间距（30mil 以上）并且加 GND 屏蔽。

3. 目录架构

3.1. 工程目录



3.2. 文件目录

a) 源文件

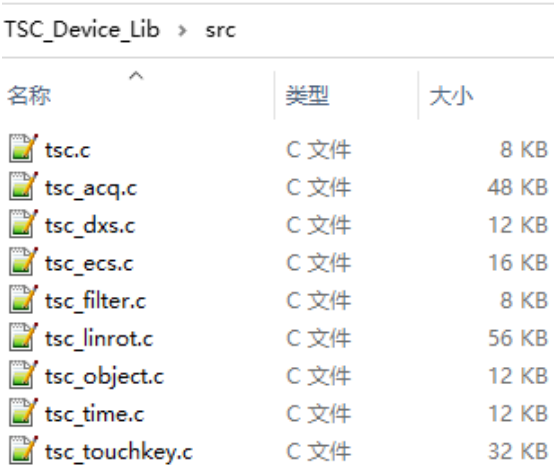


图 16

b) 头文件

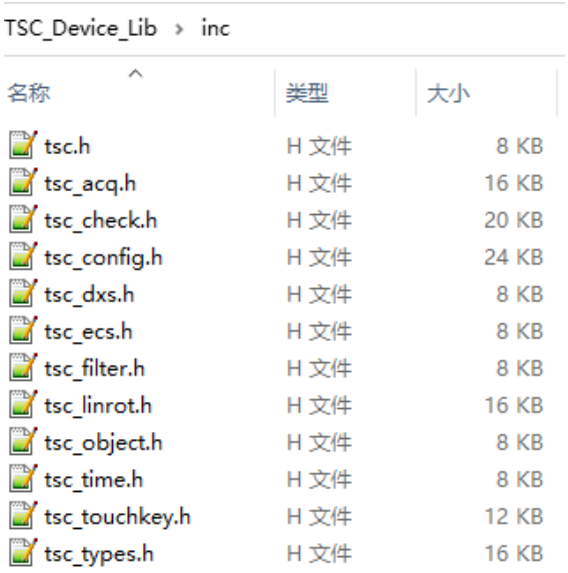


图 17

3.3. 层级结构

下图 3-3 说明了整个触摸控制的层级区分：

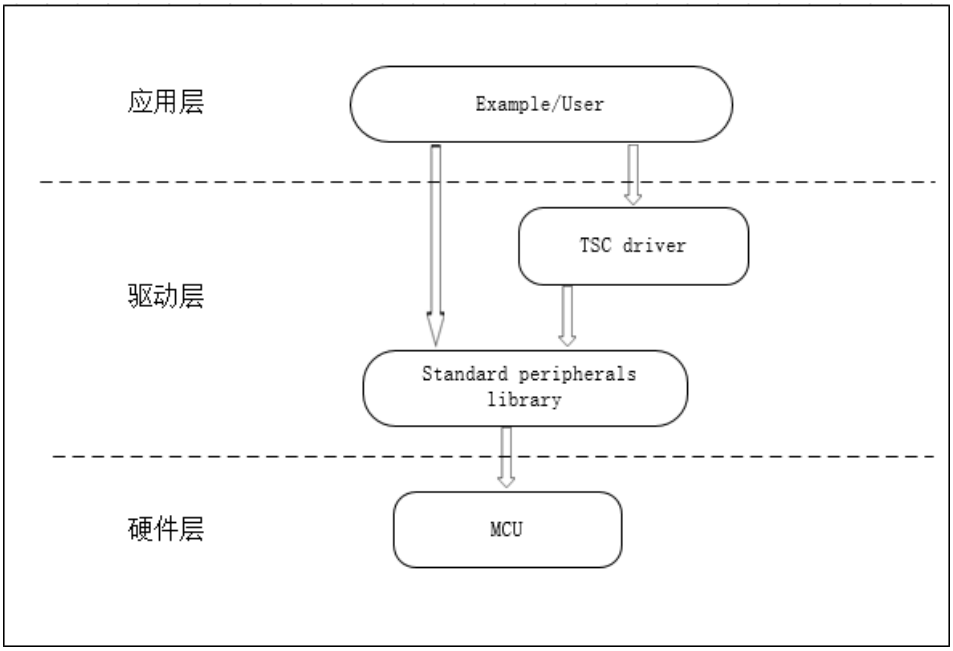


图 18

应用层：..\Example

驱动层：..\Library\APM32F0xx_StdPeriphDriver

..\Library\TSC_Device_Lib

3.4. 驱动层

根据图 3-4 将详细说明驱动层触摸模块的调用关系：

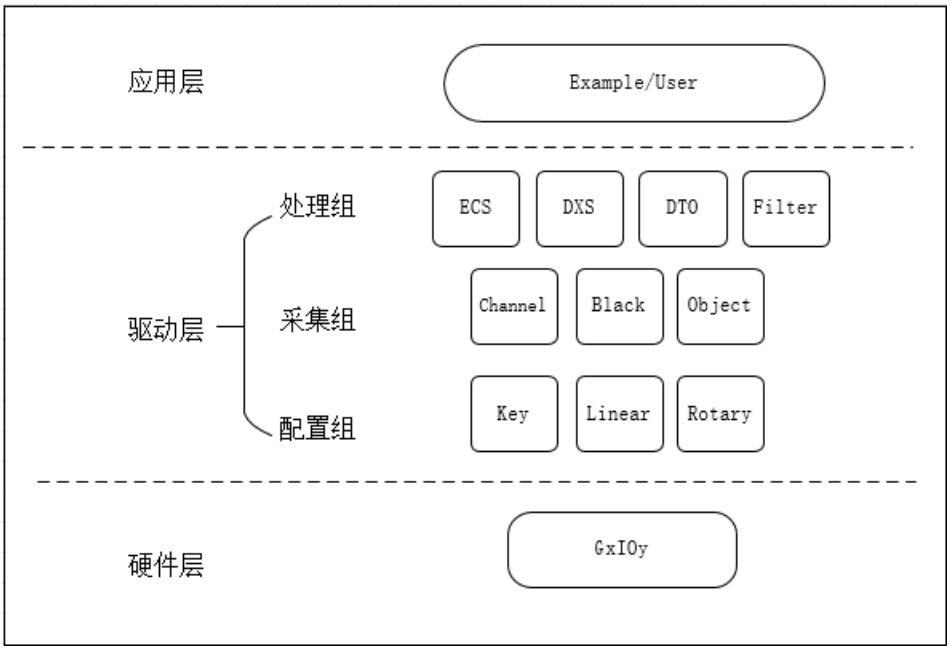


图 19

处理组：tsc_ecs.c \ tsc_dxs.c \ tsc_time.c \ tsc_filter.c \ …

采集组：tsc.c \ tsc_acq.c \ tsc_object.c \ …

配置组：tsc_touchkey.c \ tsc_linrot.c \ …

4. 软件设计

4.1. 通道

Channel 用来存储一些基本信息的参数。

4.1.1. 参数

#define　TOUCH_TOTAL_CHANNELS

4.1.2. 枚举结构

TSC_Channel_Src_T：通道源结构，包括 GROUPx 值，GxIOy 值，IOGXCR 寄存器值。
TSC_Channel_Dest_T：通道目的结构。
TSC_Channel_Data_T：通道数据结构，包括标志位，参考值，差值，采样值。

4.1.3. 使用方式

```
/* Source and Configuration (ROM) */
CONST TSC_Channel_Src_T MyChannels_Src[TOUCH_TOTAL_CHANNELS] =
{
    /* Block 0 */
    { CHANNEL_0_SRC, CHANNEL_0_IO_MSK, CHANNEL_0_GRP_MSK },
    { CHANNEL_2_SRC, CHANNEL_2_IO_MSK, CHANNEL_2_GRP_MSK },

    /* Block 1 */
    { CHANNEL_1_SRC, CHANNEL_1_IO_MSK, CHANNEL_1_GRP_MSK },
    { CHANNEL_3_SRC, CHANNEL_3_IO_MSK, CHANNEL_3_GRP_MSK },
    { CHANNEL_5_SRC, CHANNEL_5_IO_MSK, CHANNEL_5_GRP_MSK },
    { CHANNEL_6_SRC, CHANNEL_6_IO_MSK, CHANNEL_6_GRP_MSK },
    { CHANNEL_7_SRC, CHANNEL_7_IO_MSK, CHANNEL_7_GRP_MSK },

    /* Block 2 */
    { CHANNEL_9_SRC, CHANNEL_9_IO_MSK, CHANNEL_9_GRP_MSK },
    { CHANNEL_10_SRC, CHANNEL_10_IO_MSK, CHANNEL_10_GRP_MSK },
    { CHANNEL_8_SRC, CHANNEL_8_IO_MSK, CHANNEL_8_GRP_MSK },
    { CHANNEL_4_SRC, CHANNEL_4_IO_MSK, CHANNEL_4_GRP_MSK },
};
```

```
/* Destination (ROM) */
CONST TSC_Channel_Dest_T MyChannels_Dest[TOUCH_TOTAL_CHANNELS] =
{
    /* Block 0 */
    { CHANNEL_0_DEST },
    { CHANNEL_2_DEST },

    /* Block 1 */
    { CHANNEL_1_DEST },
    { CHANNEL_3_DEST },
    { CHANNEL_5_DEST },
    { CHANNEL_6_DEST },
    { CHANNEL_7_DEST },

    /* Block 2 */
    { CHANNEL_8_DEST },
    { CHANNEL_9_DEST },
    { CHANNEL_10_DEST },
    { CHANNEL_4_DEST },
};

/* Data (RAM) */
TSC_Channel_Data_T MyChannels_Data[TOUCH_TOTAL_CHANNELS];
```

4.2. 组

1 个组(Block)需要同时采集多个通道，可以设置多个组。

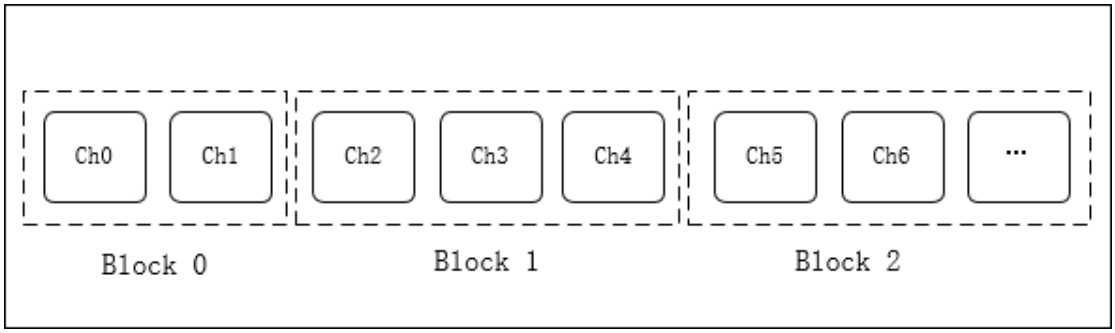


图 20

4.2.1. 参数

#define　TOUCH_TOTAL_BLOCKS

4.2.2. 枚举结构

TSC_Block_T：　包括各通道配置。

4.2.3. 函数描述

函数名	用途
TSC_Acq_ReadBlockResult	读取所有通道测量值，并计算出差值。
TSC_Acq_CalibrateBlock	校准块计算值。
TSC_Acq_ConfigBlock	配置块初始化。
TSC_Acq_StartPerConfigBlock	开始块采样。
TSC_Acq_WaitBlockEOA	等待块采样完成。

4.2.4. 使用方式

```
/* List (ROM) */
CONST TSC_Block_T MyBlocks[TOUCH_TOTAL_BLOCKS] =
{
    {&MyChannels_Src[0], &MyChannels_Dest[0], MyChannels_Data, BLOCK_0_NUMCHANNELS, BLOCK_0_MSK_CHANNELS, BLOCK_0_MSK_GROUPS},
    {&MyChannels_Src[2], &MyChannels_Dest[2], MyChannels_Data, BLOCK_1_NUMCHANNELS, BLOCK_1_MSK_CHANNELS, BLOCK_1_MSK_GROUPS},
    {&MyChannels_Src[7], &MyChannels_Dest[7], MyChannels_Data, BLOCK_2_NUMCHANNELS, BLOCK_2_MSK_CHANNELS, BLOCK_2_MSK_GROUPS}
};
```

4.3. 目标

目标(Object)等价于传感器，包括驱动文件中支持的按键传感，线性传感和旋转传感。

4.3.1. 参数

#define　TOUCH_TOTAL_OBJECTS

4.3.2. 枚举结构

TSC_Object_T：　表示一个目标。
TSC_ObjectGroup_T：表示一个组多个目标。

4.3.3. 函数描述

函数名	用途
TSC_Obj_ConfigGroup	配置一个组的多个目标。
TSC_Obj_ProcessGroup	处理一个组的多个目标。
TSC_Obj_ConfigGlobalObj	配置全局目标。

4.3.4. 使用方式

```
/* List (ROM) */
CONST TSC_Object_T MyObjects[TOUCH_TOTAL_OBJECTS] =
{
    { TSC_OBJ_TOUCHKEY, (TSC_TouchKey_T*)& MyTouchKeys[0] },
    { TSC_OBJ_TOUCHKEY, (TSC_TouchKey_T*)& MyTouchKeys[1] },
    { TSC_OBJ_TOUCHKEY, (TSC_TouchKey_T*)& MyTouchKeys[2] },
    { TSC_OBJ_TOUCHKEY, (TSC_TouchKey_T*)& MyTouchKeys[3] },
    { TSC_OBJ_TOUCHKEY, (TSC_TouchKey_T*)& MyTouchKeys[4] },

    { TSC_OBJ_LINEAR, (TSC_LinRot_T*)& MyLinRots[0] },
    { TSC_OBJ_LINEAR, (TSC_LinRot_T*)& MyLinRots[1] },
};
/* Group (RAM) */
TSC_ObjectGroup_T MyObjGroup =
{
    &MyObjects[0],          /*!< First object           */
    TOUCH_TOTAL_OBJECTS,    /*!< Number of objects      */
    0x00,                   /*!< State mask reset value */
    TSC_STATE_NOT_CHANGED   /*!< Current state          */
};
```

4.4. 按键传感

Touch Key 仅由一个通道组成，类似按键，包括两个状态：释放和检测。

4.4.1. 参数

#define　TOUCH_TOTAL_TOUCHKEYS

4.4.2. 枚举结构

TSC_TouchKeyB_T：表示基本触摸按键。
TSC_TouchKey_T：表示扩展触摸按键，增加方法和状态机。

4.4.3. 函数描述

函数名	用途
TSC_TouchKey_Config	配置触摸按键。
TSC_TouchKey_Process	处理按键状态机。

4.4.4. 使用方式

```
/** Methods for "extended" type (ROM) */
CONST TSC_TouchKeyMethods_T MyKeys_Methods =
{
    TSC_TouchKey_Config,
    TSC_TouchKey_Process
};
```

```
/* TouchKeys list (ROM) */
CONST TSC_TouchKey_T MyTouchKeys[TOUCH_TOTAL_KEYS] =
{
    { &MyKeys_Data[0], &MyKeys_Param[0], &MyChannels_Data[CHANNEL_0_DEST], MyKeys_StateMachine, &MyKeys_Methods },
    { &MyKeys_Data[1], &MyKeys_Param[1], &MyChannels_Data[CHANNEL_1_DEST], MyKeys_StateMachine, &MyKeys_Methods },
    { &MyKeys_Data[2], &MyKeys_Param[2], &MyChannels_Data[CHANNEL_2_DEST], MyKeys_StateMachine, &MyKeys_Methods },
    { &MyKeys_Data[3], &MyKeys_Param[3], &MyChannels_Data[CHANNEL_3_DEST], MyKeys_StateMachine, &MyKeys_Methods },
    { &MyKeys_Data[4], &MyKeys_Param[4], &MyChannels_Data[CHANNEL_4_DEST], MyKeys_StateMachine, &MyKeys_Methods },
};
```

4.5. 线性/旋转传感

线性/旋转传感器(LinRot)，是由可变数量的单通道组成。而线性与旋转的区别在于如何组织电极位置。

通道数量（n = 1、3、4、5、6），
差值系数表（MyLinRot0_DeltaCoeff），
位置偏移表（TSC_POSOFF_xCH_LIN/ROT_M1/M2/H/D），
旋转扇区计算（TSC_SCTCOMP_ xCH_LIN/ROT_M1/M2/H/D），
线性位置校正（TSC_POSCORR_ xCH_LIN/ROT_M1/M2/H/D）。

4.5.1. 差值系数表

用于调整线性传感器的每个通道，其个数不受限制，可以共享同一个系数表。
系数值为 16 位。MSB 为整数部分，LSB 为小数部分，例如：

系数值为 1.50 时：
0x01 to the MSB
0x0D to the LSB (0.50 x 256 = 12.8 -> rounded to 13 = 0x0D)
系数值为 0.80 时:
0x00 to the MSB
0xCD to the LSB (0.80 x 256 = 204.8 -> rounded to 205 = 0xCD)

应用系数值为 1：

```
CONST uint16_t MyLinRot0_DeltaCoeff[3] = {0x0100, 0x0100, 0x0100};
CONST uint16_t MyLinRot1_DeltaCoeff[3] = {0x0100, 0x0100, 0x0100};
```

4.5.2. 位置偏移表

在“tsc_conf.h”文件中通过相关宏进行选择。

```
/* Select which Linear and Rotary sensors you use in your application.
 *  - 0 = Not Used
 *  - 1 = Used
 *
 *  LIN = Linear sensor
 *  ROT = Rotary sensor
 *  M1 = Mono electrodes design with 0/255 position at extremities of the sensor
 *  M2 = Mono electrodes design
 *  H = Half-ended electrodes design
 *  D = Dual electrodes design
 */
#define TOUCH_USE_3CH_LIN_M1  (1)
#define TOUCH_USE_3CH_LIN_M2  (1)
#define TOUCH_USE_3CH_LIN_H   (1)
#define TOUCH_USE_3CH_ROT_M   (1)

#define TOUCH_USE_4CH_LIN_M1  (1)
#define TOUCH_USE_4CH_LIN_M2  (1)
#define TOUCH_USE_4CH_LIN_H   (1)
#define TOUCH_USE_4CH_ROT_M   (1)

#define TOUCH_USE_5CH_LIN_M1  (1)
#define TOUCH_USE_5CH_LIN_M2  (1)
#define TOUCH_USE_5CH_LIN_H   (1)
#define TOUCH_USE_5CH_ROT_M   (1)
#define TOUCH_USE_5CH_ROT_D   (1)

#define TOUCH_USE_6CH_LIN_M1  (1)
#define TOUCH_USE_6CH_LIN_M2  (1)
#define TOUCH_USE_6CH_LIN_H   (1)
#define TOUCH_USE_6CH_ROT_M   (1)
```

4.5.3. 电极位置

可以根据设计方式分为单通道电极设计，双通道电极设计，半通道电极设计。

a) 单通道电极

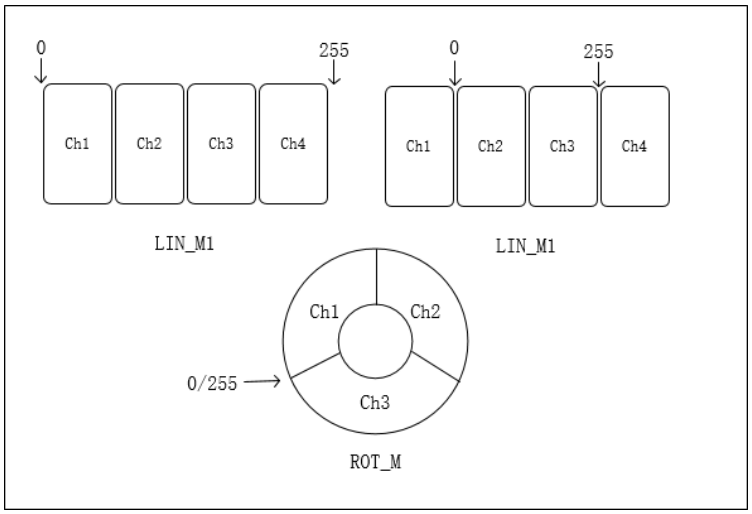


图 21

采用电极数量等于通道数量，适用于线性和旋转传感器。

包括（CH1，CH2，CH3）

（CH1，CH2，CH3，CH4）

（CH1，CH2，CH3，CH4，CH5）

LIN_M1：线性传感器设计值 0 在头部 255 在尾部。

LIN_M2：线性传感器设计值 0 在第一二电极之间，255 在后两个电极之间。

ROT_M：旋转传感器设计值 0 在头部 255 在尾部。

b) 双通道电极

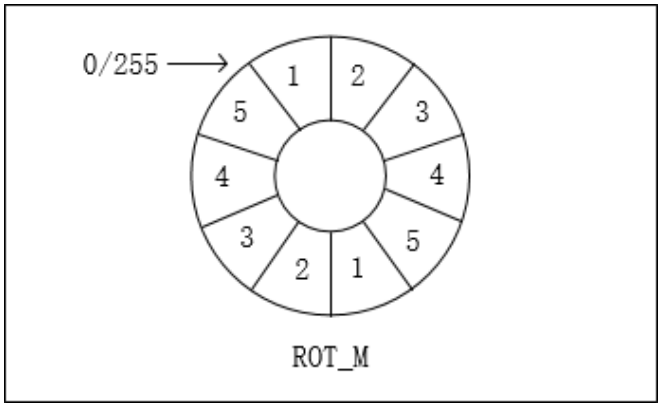


图 22

采用电极复制交叉组合，增加触摸面积，适用于旋转传感器。（只支持 5 通道）
包括（CH1，CH2，CH3，CH4，CH5，CH3，CH1，CH4，CH2，CH5）
（CH1，CH2，CH3，CH4，CH5，CH1，CH3，CH5，CH2，CH4）
（CH1，CH2，CH3，CH4，CH5，CH2，CH4，CH1，CH3，CH5）
ROT_D：旋转传感器设计值 0 在头部 255 在尾部。

c) 半通道电极

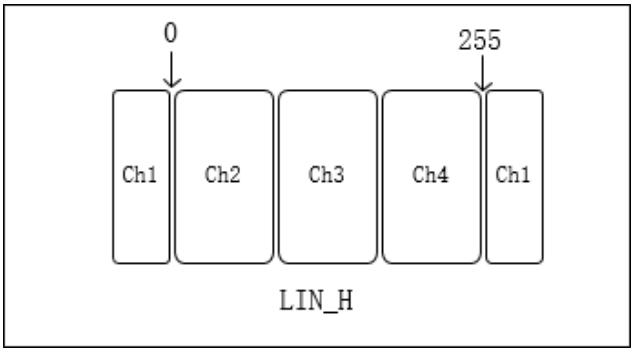


图 23

将第一个电极分为两部分，一部分在最前端，一部分在最后端，只适用于线性传感器。优于单通道。
包括（ch1，CH2，CH3，ch1）
（ch1，CH2，CH3，CH4，ch1）
（ch1，CH2，CH3，CH4，CH5，ch1）
LIN_H：线性传感器设计值在 0 在第一二电极之间，255 在后两个电极之间。

4.5.4. 参数

#define TOUCH_TOTAL_LINROTS

4.5.5. 枚举结构

TSC_LinRotB_T： 表示基本线性旋转触摸。
TSC_LinRot_T：表示扩展线性旋转触摸，增加方法和状态机。

4.5.6. 函数描述

函数名	用途
TSC_Linrot_Config	配置线性旋转触摸。
TSC_Linrot_Process	处理线性旋转状态机。
TSC_Linrot_CalcPos	计算线性旋转的位置。

4.5.7. 使用方式

```
/* Methods for "extended" type (ROM) */
CONST TSC_LinRotMethods_T MyLinRots_Methods =
{
    TSC_Linrot_Config,
    TSC_Linrot_Process,
    TSC_Linrot_CalcPos
};

/* LinRots list (ROM) */
CONST TSC_LinRot_T MyLinRots[TOUCH_TOTAL_LINROTS] =
{
    /* LinRot sensor 0 = LTS */
    &MyLinRots_Data[0],
    &MyLinRots_Param[0],
    &MyChannels_Data[CHANNEL_5_DEST],
    3, /*!< Number of channels */
    MyLinRot0_DeltaCoeff,
    (TSC_tPosition_T*)TSC_POSOFF_3CH_LIN_H,
    TSC_SCTCOMP_3CH_LIN_H,
    TSC_POSCORR_3CH_LIN_H,
    MyLinRots_StateMachine,
    &MyLinRots_Methods,

    /* LinRot sensor 1 = RTS */
    &MyLinRots_Data[1],
    &MyLinRots_Param[1],
    &MyChannels_Data[CHANNEL_8_DEST],
    3,  /*!< Number of channels */
    MyLinRot1_DeltaCoeff,
    (TSC_tPosition_T*)TSC_POSOFF_3CH_LIN_H,
    TSC_SCTCOMP_3CH_LIN_H,
    TSC_POSCORR_3CH_LIN_H,
    MyLinRots_StateMachine,
    &MyLinRots_Methods
};
```

4.6. 超时检测机制

Detection Time Out (DTO)
该机制为所有传感器的检测状态提供一个最大持续时长，用来避免液体或其他障碍误触。
当超过该时间后，传感器会自动进行重新校准，即使液体和障碍依旧存在。

4.6.1. 参数

#define　TOUCH_DTO

4.6.2. 函数描述

函数名	用途
TSC_TouchKey_ReadTimeForDTO	读取按键超时值。
TSC_Linrot_ReadTimeForDTO	读取线性/旋转超时值。

4.6.3. 使用方式

DTO 在驱动中自动执行，无需调用该模块函数，只需通过宏定义参数进行开关选择。

4.7. 定时管理

Time 通过定义全局变量，进行定时递增，通过比较，来判断是否达到延时效果。
同时可以针对 ECS，DTO 等机制作为时间基准，也可以在应用层使用。

4.7.1. 参数

#define　TOUCH_TICK_FREQ

4.7.2. 函数描述

函数名	用途
TSC_Time_Config	初始化开始计时
TSC_Time_Delay_ms	延时 ms
TSC_Time_Delay_sec	延时 sec
TSC_Time_ProcessInterrupt	定时中断函数

4.7.3. 使用方式

```
/* Process objects, DxS and ECS, Check if all blocks have been acquired */
if (idx_block > TOUCH_TOTAL_BLOCKS - 1)
{
    idx_block = 0;
    config_done = 0;

    TSC_Obj_ProcessGroup(&MyObjGroup);
    TSC_Dxs_FirstObj(&MyObjGroup);

    /* ECS every 100ms */
    if (TSC_Time_Delay_ms(100, &Global_ECS_last_tick) == TSC_STATUS_OK)
    {
        if (TSC_Ecs_Process(&MyObjGroup) == TSC_STATUS_OK)
        {
            Global_ProcessSensor = 0;
        }
        else
        {
            Global_ProcessSensor = 1;
        }
    }
    status = TSC_STATUS_OK;
}
```

4.8. 检测排除机制

Detection Exclusion System (DXS)

该机制主要用于防止多个传感器之间的误触发，当各个传感器之间距离太近，或者灵敏度太高，都将引起不必要的误触发，所以通过设计传感器组中第一个传感器具有最高优先级，进入 DETECT 状态，其他传感器将被“阻塞”以 TOUCH 状态进入。所谓第一个传感器，取决于传感器在 DXS 组中的位置和 DXS 组的处理顺序，详见下图：

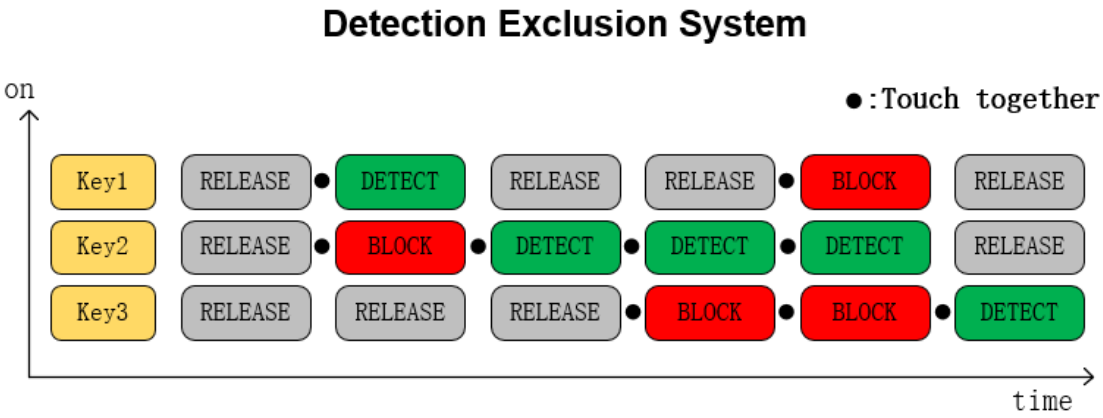


图 24

4.8.1. 参数

#define TOUCH_USE_DXS

4.8.2. 函数描述

函数名	用途
TSC_Dxs_FirstObj	探测首个目标

4.8.3. 使用方式

DXS 可以通过用户自定义执行，也可以在中断程序中执行，但在执行前必须判断传感器状态机是否处理完成，即执行 TSC_Obj_ProcessGroup 此函数。

```
/* Process objects, DxS and ECS, Check if all blocks have been acquired */
if (idx_block > TOUCH_TOTAL_BLOCKS - 1)
{
    idx_block = 0;
    config_done = 0;

    TSC_Obj_ProcessGroup(&MyObjGroup);
    TSC_Dxs_FirstObj(&MyObjGroup);

    /* ECS every 100ms */
    if (TSC_Time_Delay_ms(100, &Global_ECS_last_tick) == TSC_STATUS_OK)
    {
        if (TSC_Ecs_Process(&MyObjGroup) == TSC_STATUS_OK)
        {
            Global_ProcessSensor = 0;
        }
        else
        {
            Global_ProcessSensor = 1;
        }
    }
    status = TSC_STATUS_OK;
}
```

4.9. 环境优化机制

Environment Change System（ECS）

环境包括电源电压，温度和空气湿度等外界因素，任何一种因素的改变都将影响测量信号，为了改善环境的影响，采用 ECS 处理。ECS 基于一阶低通滤波原理：

$$Y(n) = KX(n) + (1-K)Y(n-1)$$

Y = 滤波输出值（参考值）。

X = 本次采样值（最后一次测量值）。

K = 滤波系数。

其中 K 值越大响应越快，可以通过参数进行配置响应速度。

当处于超时检测或长时间触摸时，ECS 将被禁用，此时 Yn=Y(n-1)。

4.9.1. 参数

#define TSLPRM_ECS_K_SLOW

#define TSLPRM_ECS_K_FAST

#define TSLPRM_ECS_DELAY

4.9.2. 函数描述

函数名	用途
TSC_Ecs_Process	ECS 处理函数（用户）
TSC_Ecs_CalculateK	计算 K 系数。
TSC_Ecs_ProcessK	处理 K 系数。

4.9.3. 使用方式

ECS 可以通过用户自定义时间间隔执行，也可以在中断程序中执行，但在执行前必须判断传感器状态机是否处理完成。

```
/* ECS every 100ms */
if (TSC_Time_Delay_ms(100, &Global_ECS_last_tick) == TSC_STATUS_OK)
{
    if (TSC_Ecs_Process(&MyObjGroup) == TSC_STATUS_OK)
    {
        Global_ProcessSensor = 0;
    }
    else
    {
        Global_ProcessSensor = 1;
    }
}
status = TSC_STATUS_OK;
```

5. TouchPanel-APM32F051 测试举例

5.1 参数

TouchPanel-APM32F051 软件使用 APM32F0xx_SDK_v1.7 中的 TSC_KeyLinearRotate 例程，基于 TSC_Device_lib。测试采用的相关参数如下表：

表 1

Parameter	Value	Parameter	Value
HCLK	48MHz	TOUCH_TSC_CTPHSEL	1
TOUCH_KEY_DETECT_IN_TH	200	TOUCH_TSC_CTPLSEL	1
TOUCH_KEY_DETECT_OUT_TH	150	TOUCH_TSC_USE_SSEN	1
TOUCH_LINROT_DETECT_IN_TH	180	TSLPRM_TSC_USE_SS	1
TOUCH_LINROT_DETECT_OUT_TH	90	TOUCH_USE_DXS	1
TOUCH_DEBOUNCE_DETECT	30	KEY_DETECT_IN_TH/OUT_TH	-=10
TOUCH_TICK_FREQ	125	TOUCH_TICK_DICHARGE_ALL	2000

表 2

Parameter	Value	Parameter	Value
HCLK	48MHz	TOUCH_TSC_CTPHSEL	1
TOUCH_KEY_DETECT_IN_TH	180	TOUCH_TSC_CTPLSEL	1
TOUCH_KEY_DETECT_OUT_TH	90	TOUCH_TSC_USE_SSEN	1
TOUCH_LINROT_DETECT_IN_TH	180	TSLPRM_TSC_USE_SS	1
TOUCH_LINROT_DETECT_OUT_TH	90	TOUCH_USE_DXS	1
TOUCH_DEBOUNCE_DETECT	2	KEY_DETECT_IN_TH/OUT_TH	-=10
TOUCH_TICK_FREQ	1000	TOUCH_TICK_DICHARGE_ALL	1000

5.2 传导骚扰抗扰度(CS)测试结果

CS 测试执行标准为 EN61000-4-6，测试条件如下：

- 频率范围：150kHz-80MHz
- Frequency Steps：1 %
- Dwell Time ：0.5 s

当采用 5.1 表 1 的参数测试时，TouchKey 可通过 10 Vrms class A，Linear&Rotate 可通过 6 Vrms class A。

当采用 5.1 表 2 的参数测试时，TouchKey 可通过 6 Vrms class A，Linear&Rotate 可通过 3 Vrms class A。

在测试不通过的条件下，主要表现为在部分频点或频段出现采样值不准确导致误触的情况。

6. 版本历史

表格 1 文件修订历史

日期	修订	变 化
2023.02.28	V1.0	初始版本
2023.08.24	V1.1	增加硬件设计建议以及软件参数配置举例，提高系统抗扰能力

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“TM”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，除非在极海销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及／或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

对于用户后续在针对极海产品进行设计、使用的过程中所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册相关信息而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失）。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。