MICROCHIP

用户指南

QTouch®模块化库外设触摸控制器用户指南

说明

Microchip QTouch®外设触摸控制器（Peripheral Touch Controller，PTC）在作为按钮、滑动条和滚轮使用的传感器上提供用于电容式触摸测量的内置硬件。PTC 支持互电容和自电容测量，无需任何外部元件。它能够提供出色的灵敏度和抗噪性以及自校准功能，并且可最大限度地降低用户调节灵敏度所需的工作量。此外，它还拥有扩展能力，支持电容式触摸表面和手势功能。

PTC 适合自主执行电容式触摸传感器测量。外部电容式触摸传感器通常在 PCB 上形成，传感器电极使用器件 I/O 引脚连接到 PTC 的模拟电荷积分器。PTC 支持构成为不同 X-Y 配置的电容式触摸矩阵的互电容传感器，包括氧化铟锡（Indium Tin Oxide，ITO）传感器网格。在互电容模式下，PTC 的每条 X 线路（驱动线路）和 Y 线路（传感线路）都分别需要一个引脚。在自电容模式下，PTC 的每个自电容传感器只需要一个带有 Y 线驱动器的引脚。

特性

- 实现低功耗、高灵敏度、不受环境影响的可靠电容式触摸按钮
- 支持互电容和自电容传感
- 在自电容模式下多达 32 个按钮
- 在互电容模式下多达 256 个按钮
- 支持集总模式配置
- 每个电极一个引脚——无外部元件
- 负载补偿电荷检测
- 互电容模式的寄生电容补偿
- 可调节增益，提供出色的灵敏度
- 整个温度和 V_{DD} 范围内的漂移均为零
- 无需温度或 V_{DD} 补偿
- 支持硬件噪声滤波和噪声信号去同步，可实现较高的传导抗扰度
- Atmel Start QTouch 配置器支持——向导引导式触摸项目创建

产品支持

如需获得有关 QTouch 电容式触摸传感软件库和相关问题的帮助，请联系您当地的 Microchip 销售代表或访问 https://www.microchip.com/support/。

目录

1. 简介

QTouch®模块化库（QTouch Modular Library，QTML）在模块化架构下提供 QTouch 库的触摸传感功能。通过将库划分为多个功能单元，应用程序开发人员可以仅将提供与目标应用相关的功能的模块包含在内，从而节省器件存储空间和处理时间。

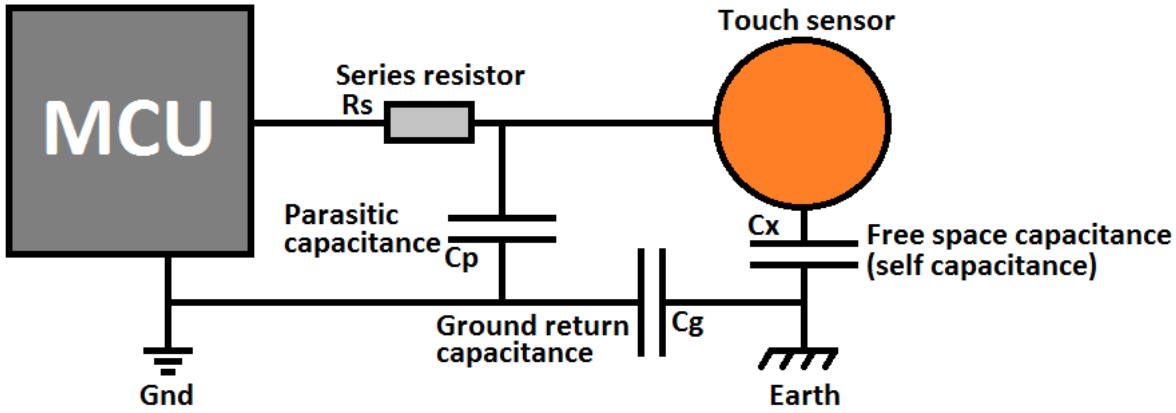
2. 电容式触摸测量

QTouch 模块化库支持一系列 AVR®和 SAM®单片机上自电容和互电容触摸传感器的 PTC 测量。

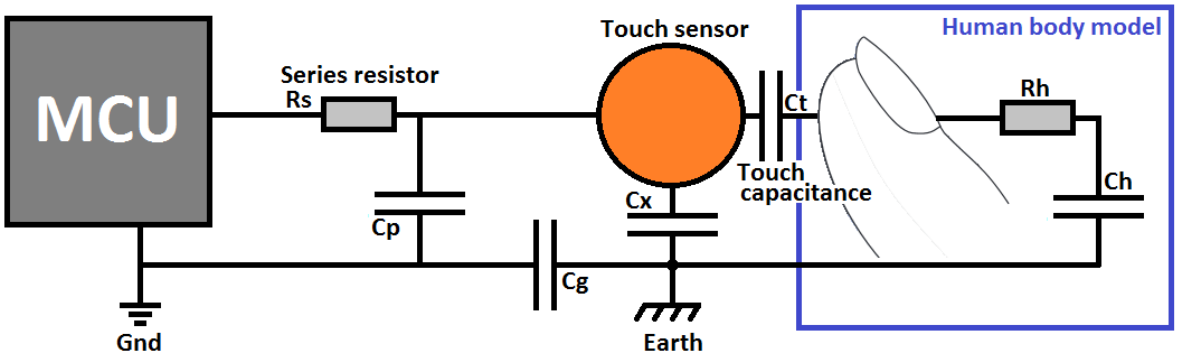
在目前的所有电容式触摸测量方法中，均可实现两种基本功能方法中的一种：自电容或互电容。

2.1 自电容

自电容是指使用单个传感器电极来测量电极与触摸传感器 MCU 电路的 DC 地之间表观电容的电容式测量。



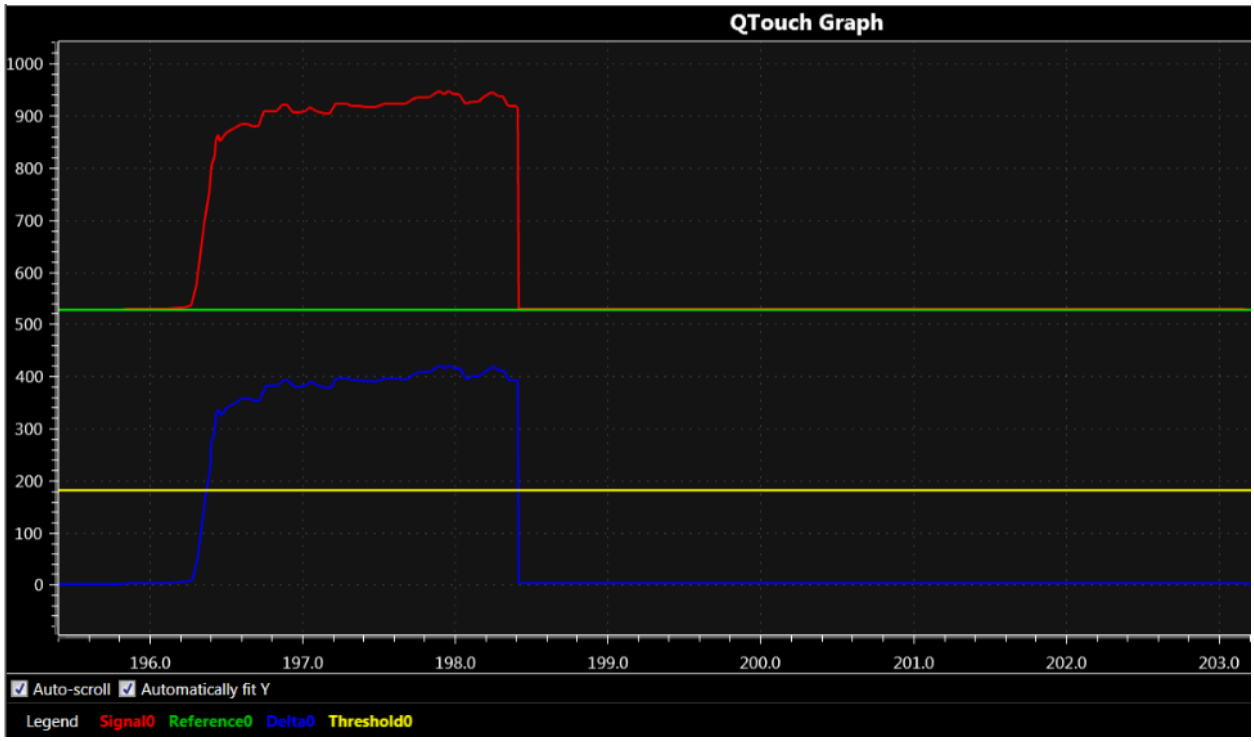
上电或复位时，将记录电容的基准测量值并假设该测量值为*“未触摸”*电容。参考电容是电容 Cp 与一对串联电容 Cg 和 Cx 并联后的组合电容。



当触摸时，如果通过串联 Ct 和 Ch 引入对地的并联路径，电容将增大。增大值将与触摸阈值进行比较，如果超过该阈值，则传感器指示*“触摸中”*。

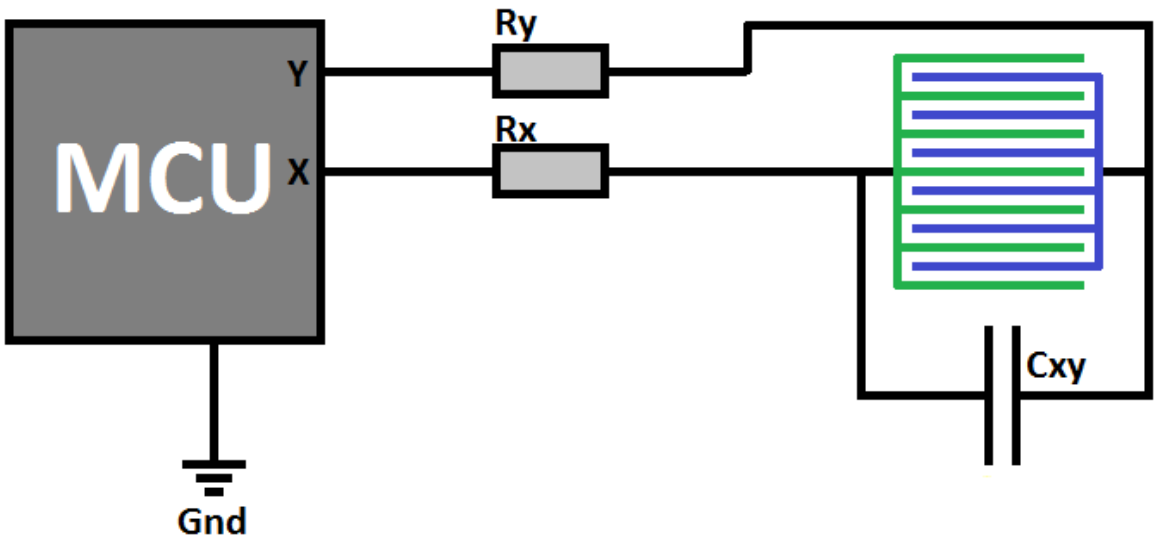
注： 人体电容 Cx 随人和周围环境变化，通常约为 100 pF 至 200 pF。然而，接触点电容 Ct 的值则比较稳定且要小得多，通常为 1 pF 至 5 pF，主要取决于触摸传感器的设计和构造，其次取决于用于激活传感器的手指大小。

由于一对串联电容中起决定作用的是较小的电容（本例中为 Ct），因此，经过精心设计和调节的传感器会呈现极为一致的接触点灵敏度，几乎不依赖于用户。

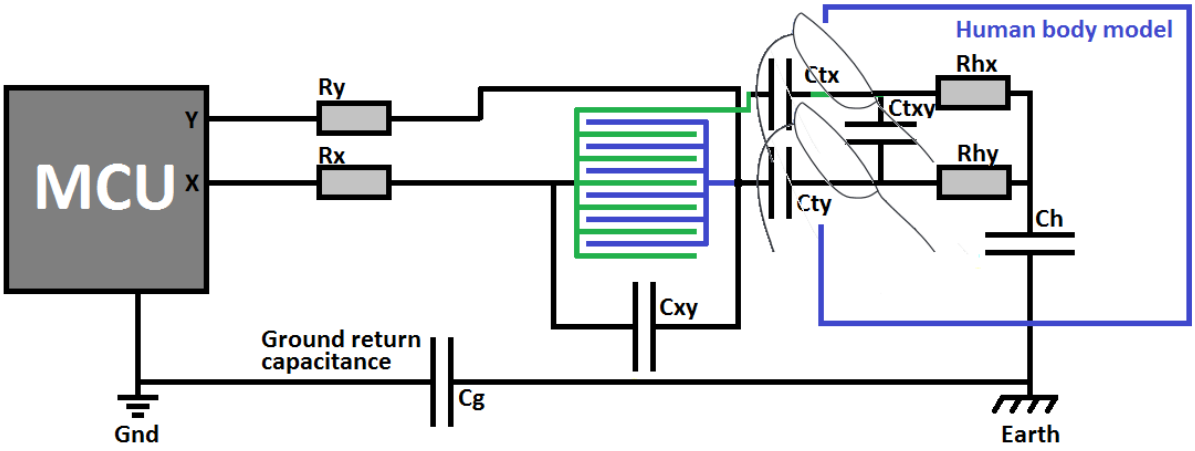


2.2 互电容

互电容是指利用一对传感器电极测量它们之间表观电容的电容测量。通常，一个电极用作驱动器（X），而另一个电极用作接收器（Y）。X 电极将电荷转移到 Y 电极的每个物理位置都是一个传感器节点，这些是对触摸敏感的位置。



与自电容一样，将记录电容的基准测量值并假设该测量值为*“未触摸”*电容。参考电容是 X 电极和 Y 电极之间的表观电容。与自电容不同的是，参考电容不依赖于接地回路。



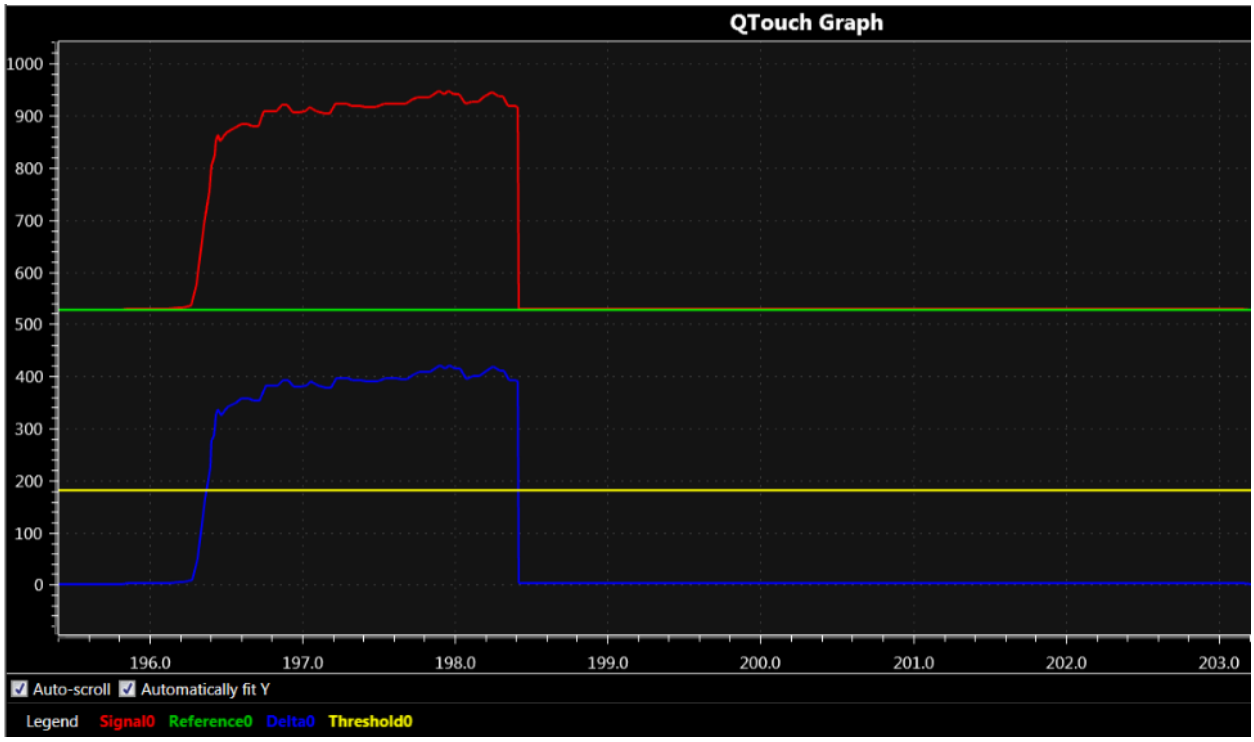
互电容传感器与人体之间的相互作用更加复杂，可将其视为 X 和 Y 电极的两个单独接触点进行建模，其中每个接触点容性耦合到人体，在人体内以阻性方式彼此相连并经由人体电容容性耦合到地。

接触点存在两种竞争效应：

- 在 X 和 Y 电极中引入导电板（手指）会增加 X 与 Y 之间的电容。当在传感器上放置任何导电部件时，会发生这种情况。
- 在 XY 节点处添加另一电容（Ch + Cg）会为 X 电极发射的能量提供另一条路径，从而减少传感器上累积的电荷量。这种效应表现为 XY 电容明显减小。仅当连接到导电部件的材料具有较大的自电容时才会发生这种情况。

当实际接触点出现时，第二种（减小）效应的影响远大于第一种（增加）效应，因此，传感器电容明显减小将指示互电容传感器上出现了接触点。

电容的这种显著变化（增量）将与配置的触摸阈值进行比较，如果超过阈值，则认为传感器处于检测中。


QTouch Graph
1000
900
800
700
600
500
400
300
200
100
0
196.0
197.0
198.0
199.0
200.0
201.0
202.0
203.0
Auto-scroll
Automatically fit Y
Legend
Signal0
Reference0
Delta0
Threshold0

3. 触摸传感器

电容传感器的实现方式多种多样，可以直接通过放置按钮来检测触摸，或者扩展功能来提供距离（接近度）、1D 位置（滑动条或滚轮）、2D 位置（QTouch 表面）或 3D 位置（包含接近度的 QTouch 表面）的相对测量。

在每种情况下，模块化库均通过超过预配置阈值的电容变化来检测接触点。一旦确认发生接触，各种后处理模块将使用计算的触摸增量在相邻传感器之间执行插值并计算触摸位置或相对接近度。

3.1 按钮

电容传感器的最简单实现是按钮，其中传感器由单个节点（一个用于自电容的电极，一对用于互电容的电极）组成，并且被解释为二进制状态：检测中或未检测。

3.2 接近传感器

按钮经过扩展便成为接近传感器。此时会监控单个传感器节点，验证电容变化是否超过预先配置的阈值。与按钮的实现方式相同，超过相应阈值时，即认为传感器处于*“检测中”*。一旦处于检测中，需通过调整初始*“检测”*阈值和第二个*“完全接触”*阈值之间的触摸增量完成接触距离的相对测量。

注： 由于接近传感器依赖于远处物体的容性负载，因此与接触点的“视距”将取决于接触点的形状和大小。

即，10 cm 处一只张开的手“看起来”比 10 cm 处一根伸出的手指更接近，因为前者对电容的影响更大，原因是距离相同的情况下前者的表面积更大。

电容（C）与面积（A）成正比，与距离（d）成反比。

$C \propto \frac{A}{d}$

3.3 集总传感器

集总传感器的实现形式为：将多条传感线路（自电容测量）或多条驱动和传感线路（互电容测量）相组合，以用作一个传感器。在触摸传感器实现中，这为应用程序开发人员提供了更大的灵活性。

- 通过减少测量次数来提高触摸传感器响应速度，从而缩短初始触摸检测所需的时间
- 通过二进制搜索快速获得位置分辨率
- 通过集总在触摸按钮应用中的“All but one”按键改善耐湿性
- 通过任何按键提供触摸唤醒功能（不超过最大电容限值），这将显著降低功耗，因为所有按键只需要一次传感器测量
- 双用途传感器电极——例如，各个按键可以集总在一起以形成接近传感器

集总传感器上触摸检测的实现方式与单节点触摸按钮相同。

3.4 线性传感器

线性传感器利用排成一行的两个或更多个相邻传感器节点的触摸增量来计算沿该行的接触点的位置。设计传感器布局和配置阈值时，应确保接触传感器上的任何位置时都会引起：

1. **至少一个传感器节点上的触摸增量超过阈值。**可将具有最大触摸增量的节点确定为接触点的中心节点，并且利用该节点标识接触点的大致位置。
2. **相邻节点上出现一定触摸增量，用于节点之间的位置插值。**中心节点左侧和右侧节点上的相对增量用于将计算出的触摸位置向具有最大增量的一侧调整。

线性传感器可以构成任何物理形状，无论是否从最后一个传感器折回到第一个传感器均可行。发生折回的传感器配置为*“滚轮”*，不发生折回的传感器则配置为*“滑动条”*。配置为滚轮时，以第一个按键为中心的接触点使用最后一个按键进行*“向左”*插值，反之亦然，而滑动条选项则在末尾实现死区。

3.5 2D 位置传感器

线性传感器在物理上以一行按键的形式实现，利用相同的方法，可以通过按键网格扩展到 2D 位置检测。按键的设计应保证可以在垂直或水平方向上进行插值，并且多个单独接触点可在各自的插值位置单独解析。

3.6 混合和匹配

QTouch 模块化库支持多种前所未有的组合，实现不同的传感器类型和测量技术，在许多情况下，可以在相同固件应用中以多种方式利用相同的传感器电极。

例如，可以在互电容模式下集总或部分集总使用互电容按键传感器的 2D 位置传感器，以提供接近测量，并且可在自电容模式下单独测量 Y 线路来改善耐湿性。

4. PTC

4.1 概述

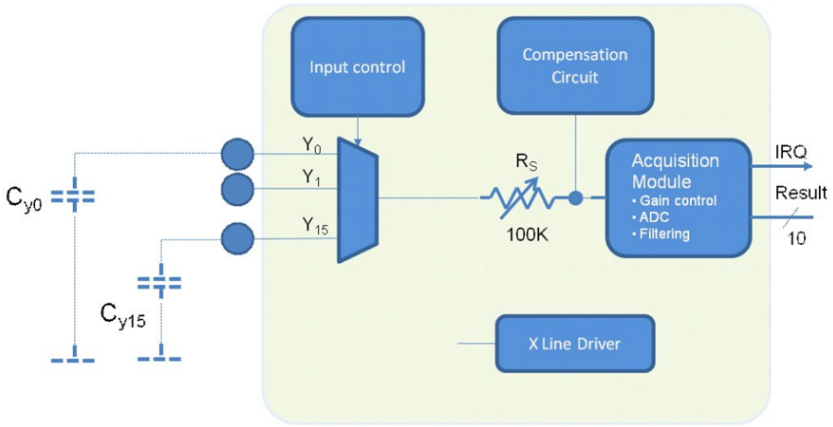
Microchip QTouch®外设触摸控制器（PTC）在作为按钮、滑动条和滚轮使用的传感器上提供用于电容式触摸测量的内置硬件。PTC 支持互电容和自电容测量，无需任何外部元件。它能够提供出色的灵敏度和抗噪性以及自校准功能，并且可最大限度地降低用户调节灵敏度所需的工作量。

PTC 适合自主执行电容式触摸传感器测量。外部电容式触摸传感器通常在 PCB 上形成，传感器电极使用器件 I/O 引脚连接到 PTC 的模拟电荷积分器。PTC 支持构成为不同 X-Y 配置的电容式触摸矩阵的互电容传感器，包括氧化铟锡（ITO）传感器网格。

4.2 自电容

在自电容模式下，PTC 的每个自电容传感器只需要一个带有 Y 线驱动器的引脚。

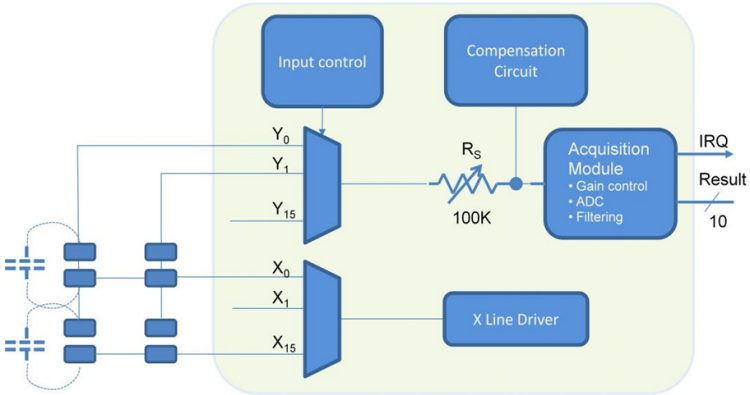
图 4-1. 自电容 PTC 测量



4.3 互电容

在互电容模式下，PTC 的每条 X 线路（驱动线路）和 Y 线路（传感线路）都分别需要一个引脚。

图 4-2. 互电容 PTC 测量



5. QTouch 模块化库

5.1 简介

QTouch 模块化库在重新设计的模块化架构下提供 QTouch 库的触摸传感功能。通过将库划分为多个功能单元，应用程序开发人员可以仅将提供与目标应用相关的功能的模块包含在内，从而节省器件存储空间和处理时间。

5.2 QTouch 库模块

根据功能，可将 QTouch 库模块分为三类，如下所示。

Acquisition Module	Signal Conditioning module	Post processing module
Acquisition auto tune module Acquisition run-time module	Frequency Hop module Frequency Hop Auto tune module	Touch Key module Scroller module
• Touch measurement • Channel sequencing • CC calibration • Auto/manual calibration of prescalar/series resistor/charge share delay	• Hop frequency • Median filter • Noise measurement • Change frequency based on noise	• Buttons post processing • Drifting, Detect integration • AKS groups • Slider/Wheel position • Touch active/inactive status • Hysteresis, IIR filtering

5.3 模块命名约定

下面给出了 QTouch 库模块遵循的命名约定。

```
qtm _ <module_name_identifier> _ <device_architecture> _ <module_ID> .
<file extension>
```

qtm / libqtm	缩写表示 QTouch 模块。为了便于识别，所有 QTouch 模块都以“qtm_”开始。对于 GCC 模块，将“lib”添加到模块名称前面，合在一起即为“libqtm”。

module_name_identifier	acq——具有自动调节功能的采集模块 acq_runtime——不带自动调节代码的采集模块 freq_hop——跳频模块 freq_hop_auto_tune——具有自动调节功能的跳频模块
device_architecture	cm0p——适用于所有 Cortex M0+后处理模块 cm4——适用于所有 Cortex M4F 后处理模块 samd1x——仅 samd10/d11 采集模块 t81x——AVR tiny817 器件系列的所有模块 t161x——AVR tiny1617 器件系列的所有模块 t321x——AVR tiny3217 器件系列的所有模块 m328pb——AVR mega328pb 器件的所有模块 m324pb——AVR mega324pb 器件的所有模块 saml21——仅 saml21 采集模块 saml22——仅 saml22 采集模块 samc21——仅 samc21 采集模块 samc20——仅 samc20 采集模块 samd21——仅 samd21 采集模块 samda1——仅 samda1 采集模块 samha1——仅 samha1 采集模块 samd20——仅 samd20 采集模块 saml10——仅 saml10 采集模块 saml11——仅 saml11 采集模块 cm23——适用于所有 Cortex M23 后处理模块
module_id	每个模块的惟一 16 位标识符
file_extension	.a——AVR®和 Arm®器件的 GCC 模块，Arm 器件的 IAR 模块 .r90——所有 AVR 模块的 IAR 模块

请参见以下示例：

表 5-1. AVR® mega328pb 器件的采集模块

GCC 模块：	libqtm_acq_m328pb_0x0001.a
IAR 模块：	qtm_acq_m328pb_0x0001.r90

SAMD2x、SAMDA1、SAMHA1、SAMD1x、SAML2x 和 SAMC2x 器件的触摸按键处理模块

GCC 模块：	libqtm_touch_key_cm0p_0x0002.a
IAR 模块：	qtm_touch_key_cm0p_0x0002.a

SAML1x 器件的触摸按键处理模块

GCC 模块：	libqtm_touch_key_cm23_0x0002.a
IAR 模块：	qtm_touch_key_cm23_0x0002.a

5.4 QTouch 库应用程序接口

除库模块外，构建完整的触摸应用程序还需要各种组件，如下所示。

1. 模块 API 文件
2. Touch.c 和 Touch.h 文件
3. Common_components_api.h
4. Touch_api_ptc.h
5. 模块重新检测标志
6. 绑定层模块

5.4.1 模块 API 文件

每个模块的 API 在其关联的头文件中定义。这可最大程度减少模块之间的依赖关系并在应用程序级别实现这些依赖关系，从而轻松地将应用程序代码从一个器件移植到另一个器件；此时，只需要调整硬件相关的模块配置即可实现移植。采集自动调节和采集手动调节模块具有相同的 API 文件。所有其他模块都有自己的 API 文件，这些 API 文件需要链接到用户应用程序。

5.4.2 Touch.c 和 Touch.h 文件

每个模块的用户选项均在应用程序代码（通常为 touch.h 和 touch.c）中配置，并通过指针引用与库模块共用。类似地，数组在应用程序中创建，用于存储模块的运行时数据，并通过指针提供给模块。

在触摸传感器的测量间隙，可通过应用程序代码实时修改配置。所有运行时数据均可用于应用程序代码。

5.4.3 Common_components_api.h

该应用程序需要所有模块共用的结构和定义。共用的定义、宏和数据结构均置于 ***“qtm_common_components_api.h”*** 文件中。

5.4.4 Touch_api_ptc.h

此文件包含内容中包含的所有模块 API 文件，因此在必要时将此文件包含在应用程序源文件中便已足够。

5.4.5 模块重新测量标志

每个模块的模块配置和功能都是惟一的，但任何模块都可能需要特定传感器的重复测量。为了实现这一点，可使用信号调理模块暂时更改采集配置，例如禁止那些不需要重新测量的传感器。

通过在信号调理组数据结构的第一个位置实现共用的 *“状态”* 字节，可以向应用程序指示这种情况。bit 7 为 1 表示应用程序应重新开始传感器组的测量，无需等待测量周期超时。

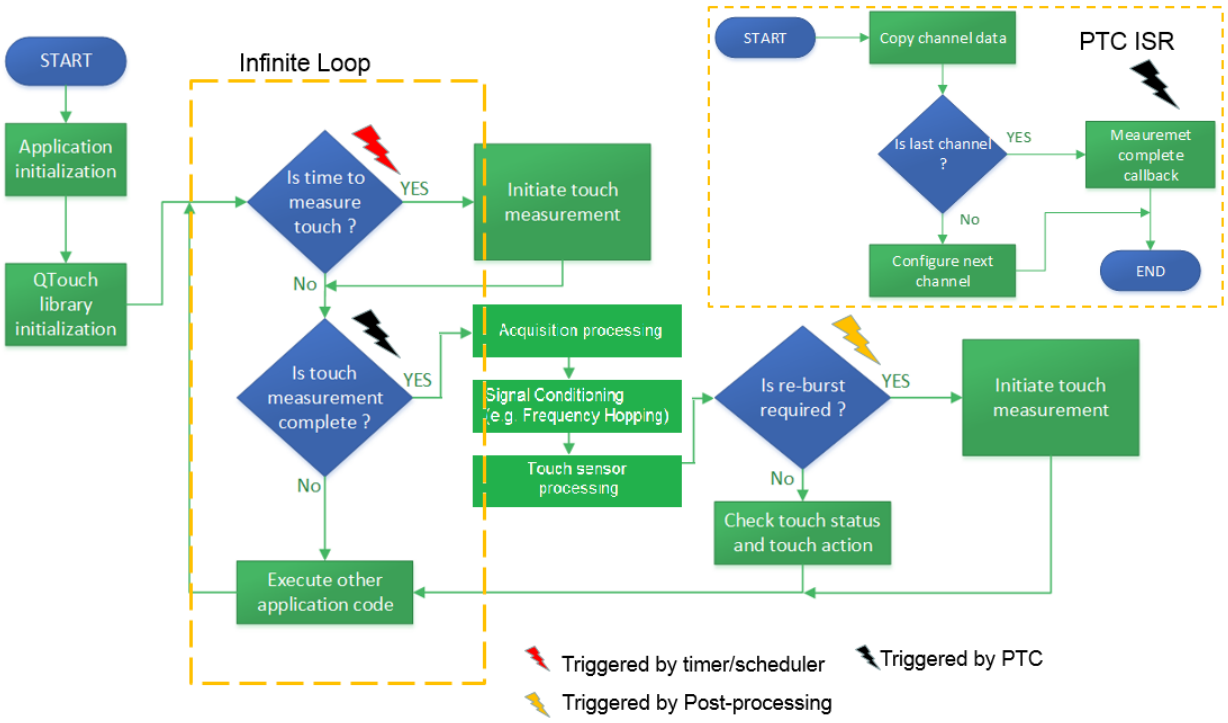
图 5-1. `uint8_t qtm_xxx_status`

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Re-burst	Module specific status flags						

5.4.6 绑定层模块

绑定层模块为用户应用程序提供简单的 QTouch 模块接口。绑定层会使用最少的 API 函数以适当的顺序绑定所有已配置的模块。它负责模块的初始化、同步调用过程并处理错误状态。

5.5 应用程序流程



5.6 MISRA 合规性

QTouch 库模块源代码符合 MISRA 2004 的 *“所需”* 规则集，但存在以下例外情况：

表 5-2. AVR® MCU 采集模块和例外情况：

Mega32xpb、Tiny81x、Tiny161x 和 Tiny321x 器件的采集模块		
MISRA 规则	**定义**	**备注**
1.1	所有代码均应符合 ISO 9899:1990 编程语言——C 语言，并应根据 ISO/IEC 9899/COR1:1995、ISO/IEC 9899/AMD1:1995 和 ISO/IEC 9899/COR2:1996 修订和更正	编译器配置为允许扩展

.....（续）		
Mega32xpb、Tiny81x、Tiny161x 和 Tiny321x 器件的采集模块		
MISRA 规则	**定义**	**备注**
8.5	头文件中不应有对象或函数的定义	内联函数用于头文件中
17.4	数组索引应该是惟一允许的指针算术形式	模块数据结构的指针作为参数传送，单个对象数据通过将数据结构作为数组索引迭代来获取。

表 5-3. AVR®后处理模块和例外情况：

Touch_key、绑定层、跳频自动调节、跳频、滚动条，2D 触摸表面和手势		
MISRA 规则	**定义**	**备注**
17.4	数组索引应该是惟一允许的指针算术形式	模块数据结构的指针作为参数传送，单个对象数据通过将数据结构作为数组索引迭代来获取。

表 5-4. Arm®采集模块和后处理模块：

模块		
MISRA 规则	**定义**	**备注**
1.1	所有代码均应符合 ISO 9899:1990 编程语言——C 语言，并应根据 ISO/IEC 9899/COR1:1995、ISO/IEC 9899/AMD1:1995 和 ISO/IEC 9899/COR2:1996 修订和更正	编译器配置为允许扩展
17.4	数组索引应该是惟一允许的指针算术形式	模块数据结构的指针作为参数传送，单个对象数据通过将数据结构作为数组索引迭代来获取。

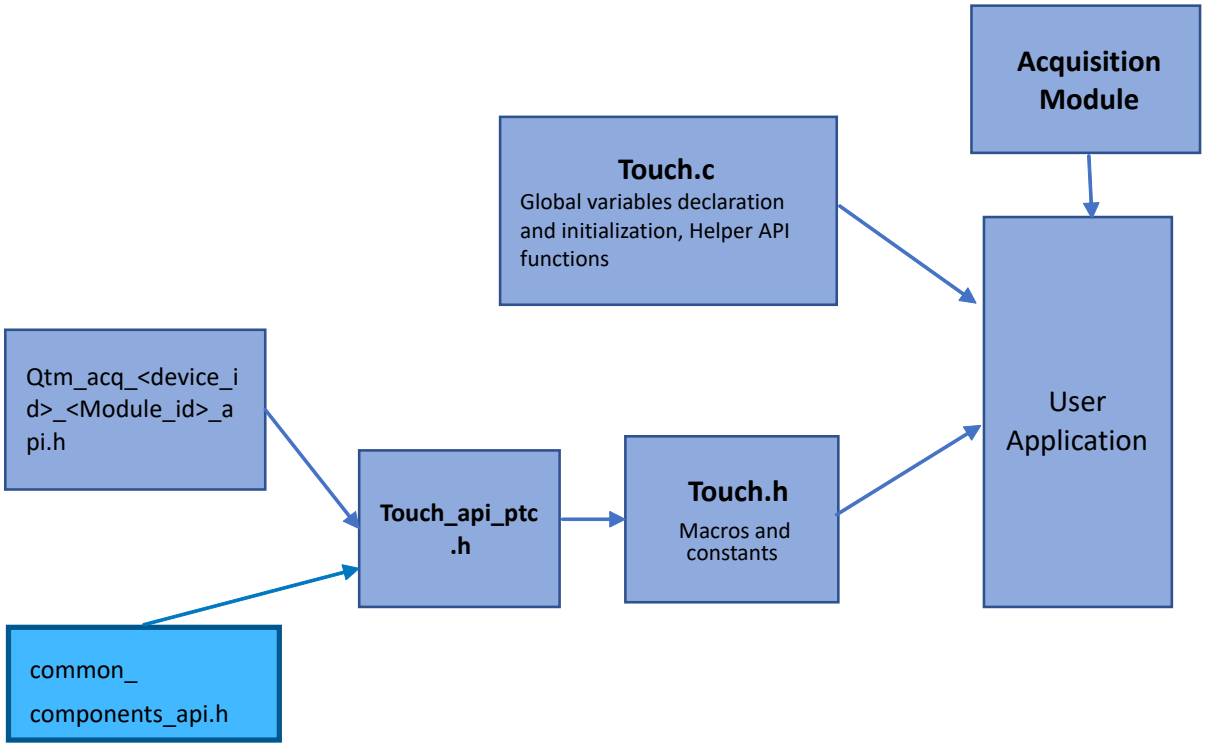
6. 采集模块

6.1 概述

触摸传感器应用的最低要求是采集模块，采集模块实现的是用于电容式触摸或接近传感器配置和测量的所有硬件相关操作。

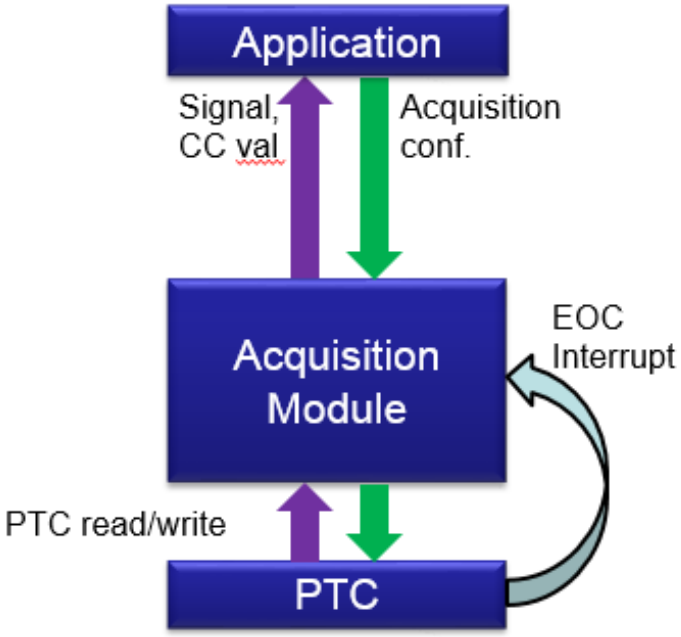
6.2 接口

数据结构的定义和 API 声明包含在 API 文件“qtm_acq_<device_id>_<module_id>_api.h”中。数据结构涵盖所有配置和输出数据变量。该文件应包含在共用 api“touch_ptc_api.h”文件中。



6.3 功能说明

采集模块是目标特定的，每个模块具有一个硬件配置结构，具体取决于所应用的触摸传感技术和方法。



此采集模块中实现的功能

- 传感器节点的硬件校准
  - 校准预分频器/电阻/电荷共用延时，以补偿传感器电极的时间常数
  - 校准内部补偿电路以匹配传感器负载
- 具有正常排序的自电容和互电容传感器触摸测量
- 使用事件系统的低功耗自动扫描模式（Atmel Start 配置器目前不支持）

6.4 配置

6.4.1 数据结构

采集模块实现了进行传感器电容相对测量所需的所有功能。这是单个器件特别构建的惟一模块，它必须访问和控制用于实现触摸传感器的引脚。

由于器件具有不同的硬件功能，因此每个器件上有不同的配置选项。为了最有效地使用系统资源（ROM 和 RAM），需要不同的传感器配置结构。

但是，如果在结构中使用相同的变量名称，则由该变量控制的功能是相同的。任何相关函数均应利用对变量的引用，而不是依赖于对结构和指针算术的引用。

采集组配置

指向“`&ptc_qtlib_acq_gen1.freq_option_select`”的引用将始终指向正确的存储位置，与器件无关。但是，如果将一个器件中的代码重新用于另一个器件，则基于指针算数形式的任何实现均需要重构。

参数	大小	范围/选项	使用
num_sensor_nodes	16 位	0 至 65535	组中配置的传感器节点数。

.....（续）

参数	大小	范围/选项	使用
acq_sensor_type	8 位	NODE_SELFCAP NODE_MUTUAL	定义应用于这一组节点的测量方法。
calib_option_select	1 字节	**Bit 7:4** **校准类型：** CAL_AUTO_TUNE_NONE CAL_AUTO_TUNE_RSEL CAL_AUTO_TUNE_PRSC CAL_AUTO_TUNE_CSD*	校准类型选择应自动调节哪个参数来实现最佳电荷转移。
		Bit 3:0 **校准类型：** CAL_CHRG_2TAU CAL_CHRG_3TAU CAL_CHRG_4TAU CAL_CHRG_5TAU	校准目标对允许的电荷转移损耗应用了一个限值，其中较高的目标设置可确保转移更大比例的满充电荷。
freq_option_select	1 字节	FREQ_SEL_0 至 FREQ_SEL_15 或 FREQ_SEL_SPREAD	FREQ_SEL_0 至 FREQ_SEL_15 在过采样期间在测量之间插入延迟周期，其中 0 是最短延时，15 是最长延时。 在过采样设置期间，FREQ_SEL_SPREAD 使这段延时以锯齿方式从 0 变化到 15
PTC_interrupt_priority**	1 字节	1 至 3	PTC 的中断优先级。
注： *——并非适用于所有器件 **——仅适用于 Arm Cortex 器件			

节点配置

类似地，节点配置结构取决于使用的器件。

- X 线路的数量
- Y 线路的数量
- 功能可用性

参数	大小	范围/选项	使用
node_xmask	1/2/4 字节	（位域）	设置与 X 线路编号对应的位置处的位。 **示例：** 仅 X0 = 0b00000001 = 0x01 X0 和 X2 = 0b00000101 = 0x05 1 字节用于最多 8 条“X”线路的器件 2 字节和 4 字节分别用于最多 16 条“X”线路和最多 32 条“X”线路的器件

.....（续）

参数	大小	范围/选项	使用
node_ymask	1/2/4 字节	（位域）	设置与 Y 线路编号对应的位置上的位。 **示例：** 仅 Y5 = 0b00100000 = 0x20 Y1、Y2 和 Y7 = 0b10000110 = 0x86 1 字节用于最多 8 条“Y”线路的器件 2 字节和 4 字节分别用于最多 16 条“X”线路和最多 32 条“Y”线路的器件
node_csd*	1 字节	0 至 255	确保传感器节点电容充电的延迟周期数。 （仅适用于 Tiny 和 Mega AVR®器件，以及 SAM E54、SAMCx、SAML22 系列 Arm®器件）
node_rsel_prsc	1 字节	**Bit 7:4 = RSEL** RSEL_VAL_0 RSEL_VAL_3* RSEL_VAL_6* RSEL_VAL_20 RSEL_VAL_50 RSEL_VAL_75* RSEL_VAL_100 RSEL_VAL_200*	内部 Y 线路串联电阻选择 （SAM E54 系列和 SAML22 系列 Arm 器件可使用 75k 和 200k 电阻）
		Bit 3:0 = PRSC PRSC_DIV_SEL_1 PRSC_DIV_SEL_2 PRSC_DIV_SEL_4 PRSC_DIV_SEL_8 PRSC_DIV_SEL_16* PRSC_DIV_SEL_32* PRSC_DIV_SEL_64* PRSC_DIV_SEL_128*	**时钟预分频比** 采集时钟由 AVR 器件的 CPU 时钟换算而来。 （TinyAVR、SAM E5x 系列和 SAM D51 可使用预分频值 16、32、64 和 128）

.....（续）

参数	大小	范围/选项	使用
node_gain	1 字节	**Bit 7:4 = 模拟增益** GAIN_1 GAIN_2 GAIN_4 GAIN_8 GAIN_16	**模拟增益设置** 调整积分电容以控制积分器增益。
		Bit 3:0 = 数字增益 GAIN_1 GAIN_2 GAIN_4 GAIN_8 GAIN_16	**数字增益设置** 将累加的和换算为数字增益。
node_oversampling	1 字节	FILTER_LEVEL_1 FILTER_LEVEL_2 FILTER_LEVEL_4 FILTER_LEVEL_8 FILTER_LEVEL_16 FILTER_LEVEL_32 FILTER_LEVEL_64 FILTER_LEVEL_128* FILTER_LEVEL_256* FILTER_LEVEL_512* FILTER_LEVEL_1024*	每次测量累加的采样数。 **注：**必须将过采样配置为大于或等于数字增益才能正确操作。 （> 64 的较高滤波等级值仅适用于 SAM E54 系列）
注： *——并非适用于所有器件			

6.4.2 状态和输出数据

虽然不同的目标硬件要求传感器节点的配置结构因器件而异，但是所有采集模块均符合标准传感器节点数据结构。已处理模块的输出数据在运行时存储在此数据结构中。

输出/状态信息可以由其他后处理模块或应用程序使用。

参数	大小	范围/选项	使用
node_acq_status	1 字节	Bit 7	指示节点校准错误 NODE_CAL_ERROR
		Bit 6	上升时间校准完成
		Bit 5	-
		Bit <4:2>（3 位） **节点校准状态** NODE_MEASURE NODE_CC_CAL NODE_PRSC_CAL NODE_RSEL_CAL NODE_CSD_CAL	指示校准是否正在进行及其当前阶段。
		校准请求	写入 1 可触发此节点上的校准序列。 （执行操作后由模块复位为 0）
		使能	写入 1 可使能此节点进行测量
node_acq_signals	2 字节	此传感器节点的最新测量。	16 位无符号值 根据 node_oversampling 和 node_gain_digital 设置进行累加和换算。
node_comp_caps	2 字节	硬件校准数据	指示此节点补偿电路的调节。

表 6-1. node_acq_status

位	7	6	5	4	3	2	1	0
	节点校准错误	上升时间校准完成	-	节点状态			校准请求	使能

NODE_MEASURE	0
NODE_CC_CAL	1
NODE_PRSC_CAL	2
NODE_RSEL_CAL	3
NODE_CSD_CAL*	4
注： *——CSD 校准不适用于 SAMD10/D11、SAMD2x 和 SAML21 器件。	

采集库状态

表 6-2. touch_lib_state_t

TOUCH_STATE_NULL	0
TOUCH_STATE_INIT	1
TOUCH_STATE_READY	2
TOUCH_STATE_CALIBRATE	3
TOUCH_STATE_BUSY	4

返回参数

表 6-3. `touch_ret_t` 共用返回类型，供所有 QTML 模块使用

TOUCH_SUCCESS	0
TOUCH_ACQ_INCOMPLETE	1
TOUCH_INVALID_INPUT_PARAM	2
TOUCH_INVALID_LIB_STATE	3
TOUCH_INVALID_POINTER	11
TOUCH_LIB_NODE_CAL_ERROR	14
注： 其他值保留供将来使用。	

7. 跳频模块

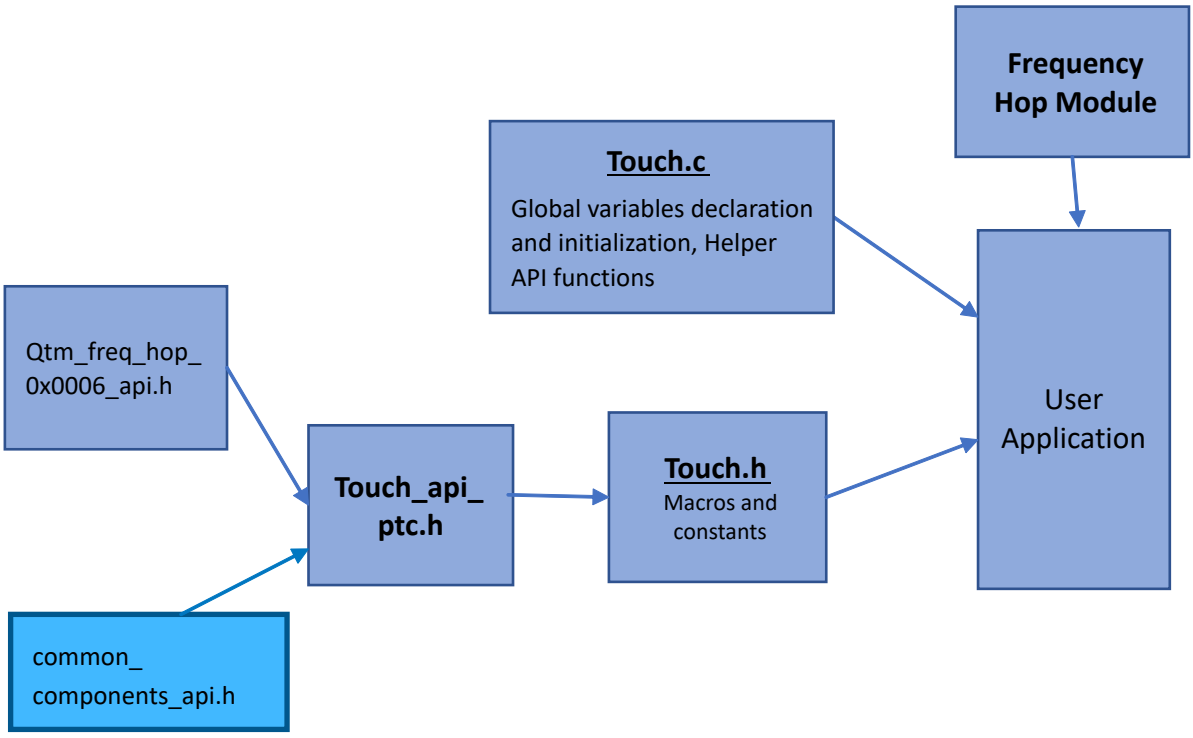
7.1 概述

跳频模块提供了一种通过改变传感器触发频率在传感器测量期间滤除噪声的方法。跳频模块的模块 ID 为 0x0006，模块名称的格式如下所示。

GCC 编译器：	`libqtm_freq_hop_xxxxx_0x0006.a`
IAR 编译器（AVR MCU）：	`qtm_freq_hop_xxxxx_0x0006.r90`
IAR 编译器（Arm MCU）：	`qtm_freq_hop_xxxxx_0x0006.a`
注： “xxxxx”——基于构建模块的器件架构的字符串。	

7.2 接口

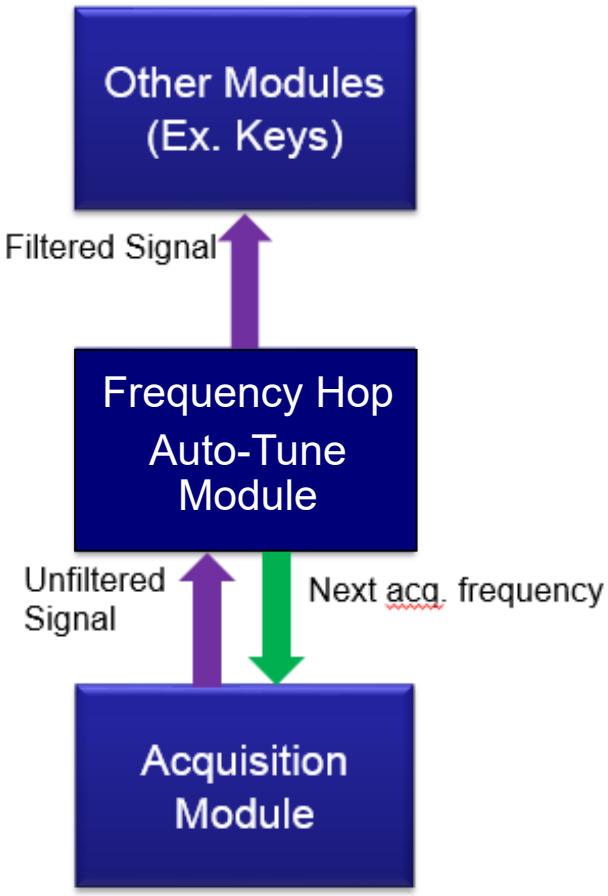
数据结构的定义和 API 声明包含在 API 文件“`qtm_freq_hop_0x0006_api.h`”中。数据结构涵盖所有配置和输出数据变量。该文件应包含在共用 api“`touch_ptc_api.h`”文件中。



配置的默认值应在 `touch.c` 和 `touch.h` 文件中定义。必须在 touch.c 文件中初始化数据结构的全局变量，并且必须在应用程序文件中使用结构的引用。

7.3 功能说明

跳频模块连接在采集模块和其余后处理模块之间，如下所示。



跳频模块应用可配置的循环跳频算法，能够在每个测量周期使用不同的采样频率。该模块使用预定义的频率初始化，这些频率在连续测量周期期间按循环顺序设置。

从采集模块测量的原始信号值随后通过**“中值滤波器”**传送。最后，滤波后的值将存储回存储器，以供后处理模块进一步处理。

频率数越多，就可以通过处理更多采样来实现有效滤波。但是，这也会增加中值滤波器使用的缓冲区大小，并且需要更多的测量周期来报告滤波后的值。因此，应根据可用的 RAM 存储器配置频率数量。

7.4 配置

7.4.1 数据结构

参数	大小	范围/选项	使用
num_sensors	1 字节	0-255	为中值滤波器缓冲数据的传感器数
num_freqs	1 字节	3-7	中值滤波器的循环/深度的频率数

.....（续）

参数	大小	范围/选项	使用
*freq_option_select	2/4 字节	N/A	指向采集库频率选择参数的指针
*median_filter_freq	2/4 字节	N/A	指向所选频率数组的指针

7.4.2 状态和输出数据

参数	大小	范围/选项	使用
module_status	1 字节	N/A	模块状态——N/A。
current_freq	1 字节	0-15	当前频率步长
*filter_buffer	2/4 字节	N/A	指向测量信号的滤波缓冲区数组的指针
*qtm_acq_node_data	2/4 字节	N/A	指向采集组的节点数据结构的指针

表 7-1. 支持频率列表

PTC 时钟 = 4 MHz		
PTC 频率延迟周期		频率（kHz）
0	FREQ_SEL_0	66.67
1	FREQ_SEL_1	62.5
2	FREQ_SEL_2	58.82
3	FREQ_SEL_3	55.56
4	FREQ_SEL_4	52.63
5	FREQ_SEL_5	50
6	FREQ_SEL_6	47.62
7	FREQ_SEL_7	45.45
8	FREQ_SEL_8	43.48
9	FREQ_SEL_9	41.67
10	FREQ_SEL_10	40
11	FREQ_SEL_11	38.46
12	FREQ_SEL_12	37.04
13	FREQ_SEL_13	35.71
14	FREQ_SEL_14	34.48
15	FREQ_SEL_15	33.33
16	FREQ_SEL_SPREAD	可变频率

8. 跳频自动调节模块

8.1 概述

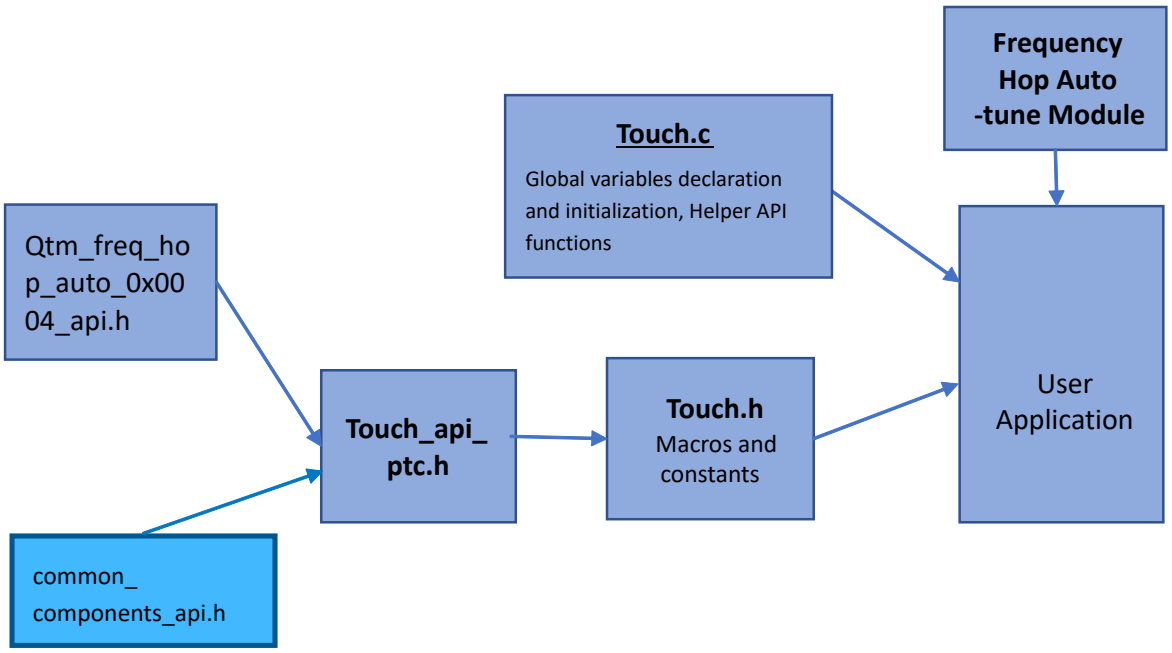
跳频自动调节模块是跳频模块的超集，额外提供噪声监视功能，并可根据测得的噪声系数调节频率。

跳频自动调节模块的模块 ID 为 0x0004，模块名称的格式如下所示。

GCC 编译器：	`libqtm_freq_hop_auto_xxxxx_0x0004.a`
IAR 编译器（AVR MCU）：	`qtm_freq_hop_auto_xxxxx_0x0004.r90`
IAR 编译器（Arm MCU）：	*`qtm_freq_hop_auto_xxxxx_0x0004.a`*
注： “xxxxx”——基于构建模块的器件架构的字符串。	

8.2 接口

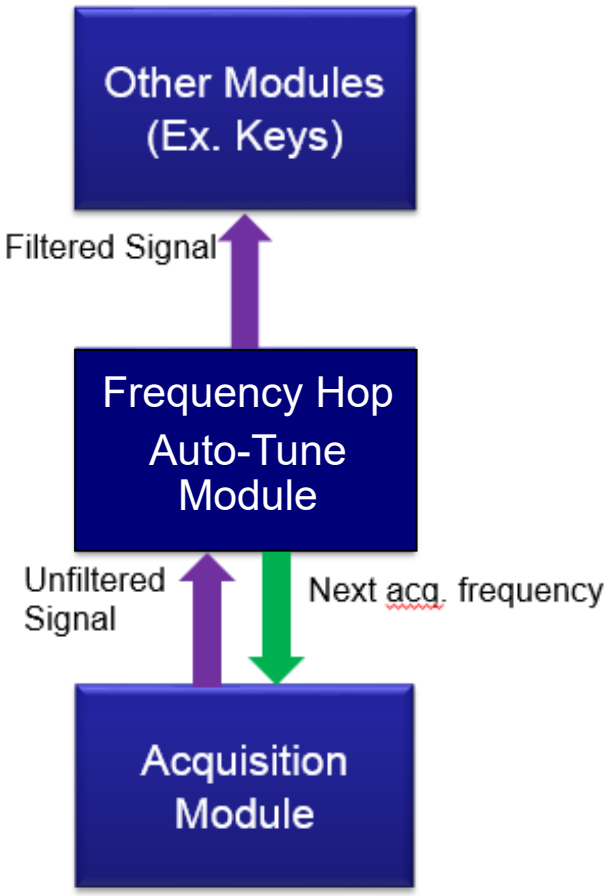
数据结构的定义和 API 声明包含在 API 文件“`qtm_freq_hop_auto_0x0004_api.h`”中。数据结构涵盖所有配置和输出数据变量。该文件应包含在共用 api `touch_ptc_api.h` 文件中。



配置的默认值应在 touch.c 和 touch.h 文件中定义。必须在 touch.c 文件中初始化数据结构的全局变量，并且必须在应用程序文件上使用结构的引用。

8.3 功能说明

跳频自动调节模块连接在采集模块和其余后处理模块之间，如下所示。



跳频自动调节模块应用可配置的循环跳频算法，能够在每个测量周期使用不同的采样频率。许多预先配置的频率在连续的测量周期期间依次实现。

在循环中包括“n”个频率的情况下，将“n”点中值滤波器应用于输出数据。

要执行自动调节，需针对每个选定的频率记录在各传感器节点上测得的信号。当一个频率的变化高于其他频率时，该频率将从测量序列中移除并替换为另一个频率。

8.4 配置

8.4.1 数据结构

参数	大小	范围/选项	使用
num_sensors	1 字节	0-255	为中值滤波器缓冲数据的传感器数
num_freqs	1 字节	3-7	中值滤波器的循环/深度的频率数
*freq_option_select	指针（2/4 字节）	指针	指向采集库频率选择参数的指针
*median_filter_freq	指针（2/4 字节）	指针	指向所选频率数组的指针

.....（续）

参数	大小	范围/选项	使用
enable_freq_autotune	1 字节	0 或 1	禁止（0）或使能（1）跳频的自动重新调节
max_variance_limit	1 字节	1-255	触发跳频恢复所需的信号变化
Autotune_count_in	1 字节	1-255	触发跳频重新调节所需的 max_variance_limit 的出现次数

8.4.2 状态和输出数据

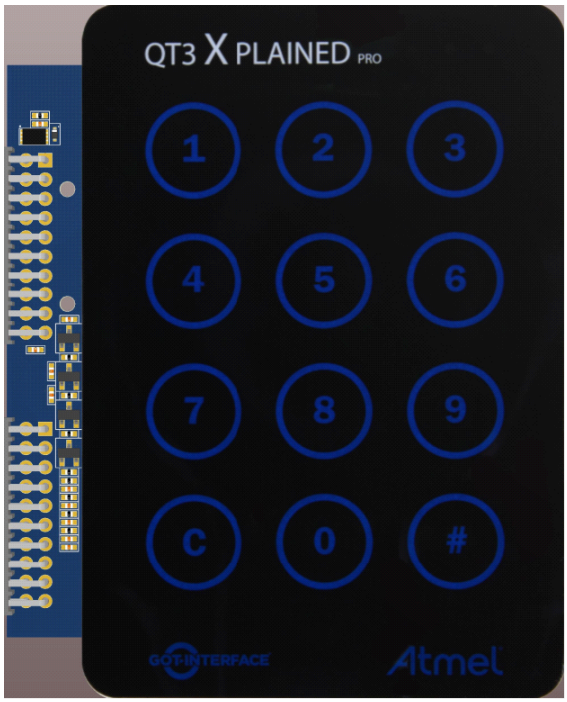
参数	大小	范围/选项	使用
module_status	1 字节	N/A	模块状态——N/A。
current_freq	1 字节	0-15	当前频率步长
*filter_buffer	指针（2/4 字节）	指针	指向测量信号的滤波缓冲区数组的指针
*qtm_acq_node_data	指针（2/4 字节）	指针	指向采集组的节点数据结构的指针
*freq_tune_count_ins	指针（2/4 字节）	指针	指向触发频率变化的计数器数组

9. 触摸按键模块

9.1 概述

触摸按键模块实现了可处理按键传感器（也称为一维触摸传感器）的功能。该模块接收采集模块的原始输出，随后对输出进行处理并提供按键传感器的触摸状态。执行的处理包括用于实现应用触摸传感器的信号后处理、环境漂移、触摸检测、触摸状态机和时序管理。提供的触摸传感器参考设计旨在帮助用户评估和设计其定制传感器板。下面给出了触摸传感器板视图和 QT3 XPlained Pro 传感器板的传感器设计。

QT3 Sensor Board Overlay



QT3 Sensor Board Design

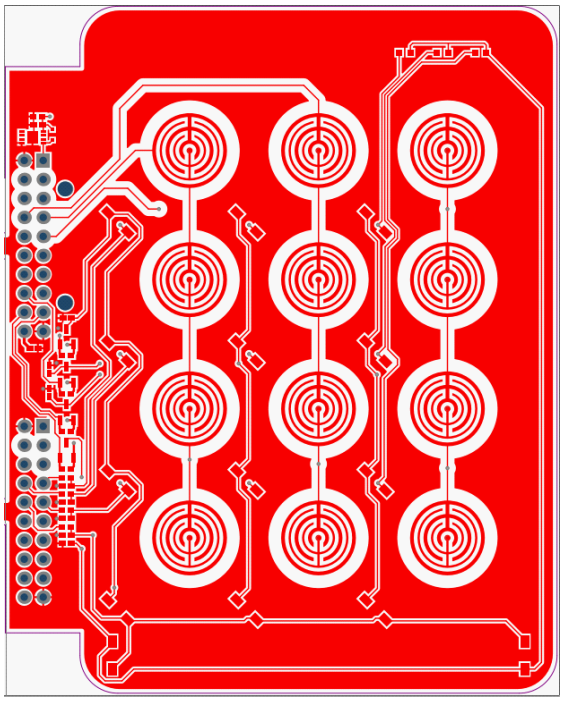
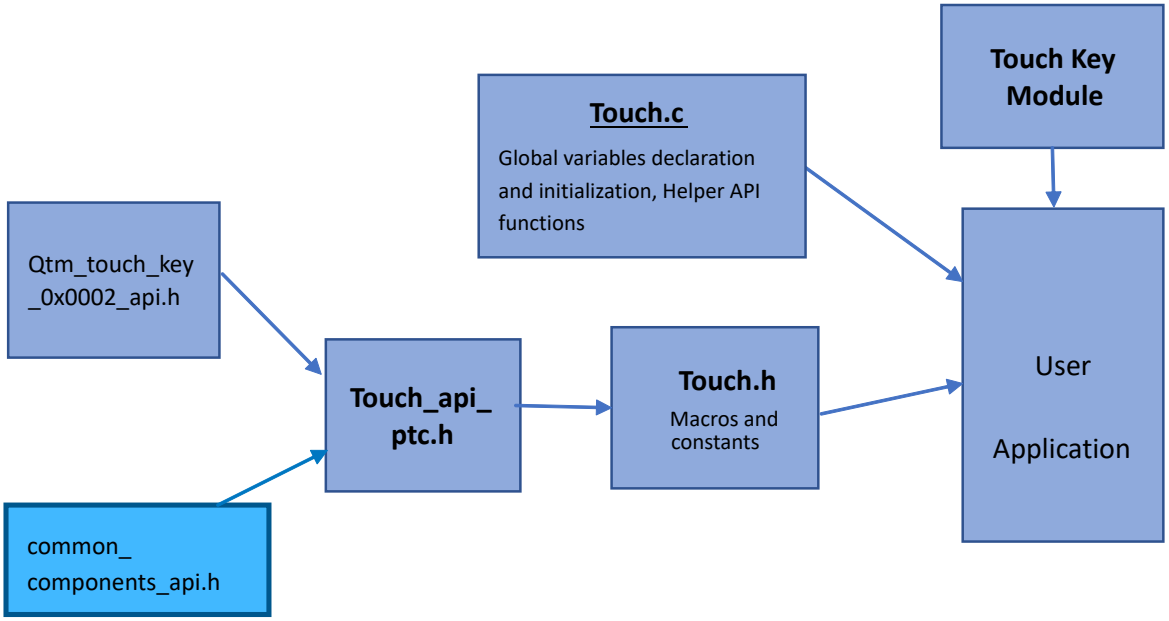


表 9-1. 模块格式

GCC 编译器：	`libqtm_touch_key_xxxxx_0x0002.a`
IAR 编译器（AVR MCU）：	`qtm_touch_key_xxxxx_0x0002.r90`
IAR 编译器（Arm MCU）：	`qtm_touch_key_xxxxx_0x0002.a`

9.2 接口

数据结构的定义和 API 声明包含在 API 文件“`qtm_touch_key_0x0002_api.h`”中。数据结构涵盖所有配置和输出数据变量。该文件应包含在共用 api `touch_ptc_api.h` 文件中。

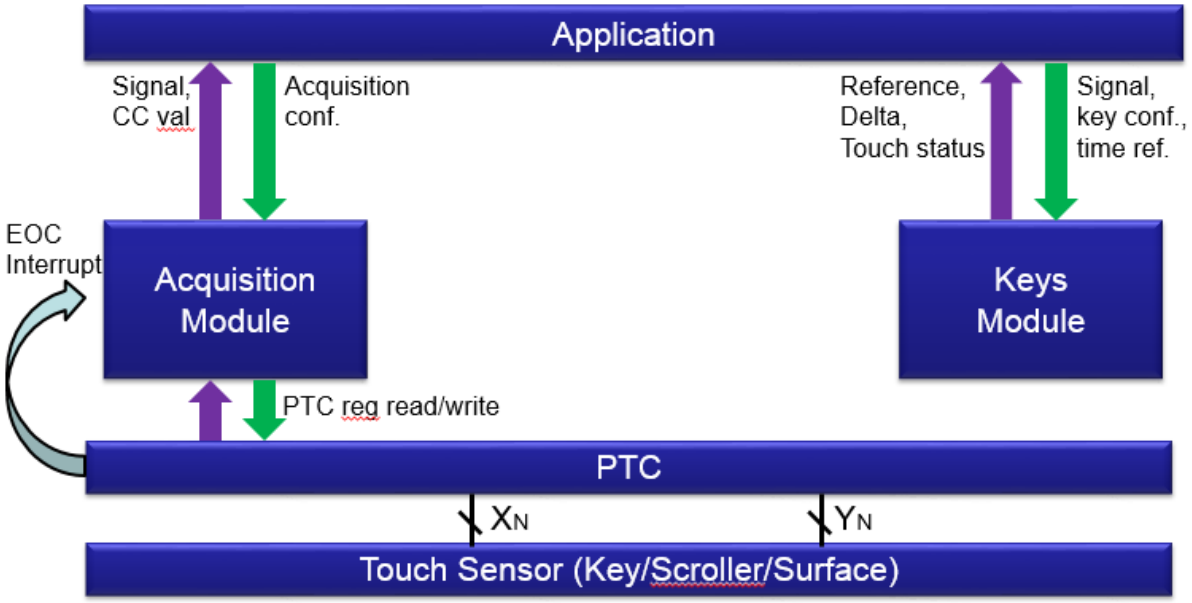


9.3 功能说明

触摸按键模块负责检测接触点，其中更高级模块执行位置插值、手势识别和接触跟踪等。

触摸按键模块中实现的功能：

- 用于检测靠近触摸和远离触摸的时序管理
- 软件校准
  - 参考信号
  - 参考漂移
- 触摸检测状态机



9.4 配置

9.4.1 数据结构

表 9-2. 组配置

参数	大小	范围/选项	使用
num_key_sensors	2 字节	1-65535	组中的传感器按键数
sensor_touch_di	1 字节	0-255	确认触摸检测的重复测量次数
sensor_max_on_time	1 字节	0（禁止），1-255	自动“重新校准”前传感器处于“检测中”的定时器周期数
sensor_anti_touch_di	1 字节	0（禁止），1-255	确认所需防触摸重新校准的重复测量次数
sensor_anti_touch_recal_thr	1 字节	0-5	减小触摸阈值以设置防触摸阈值。 0 = 100%触摸阈值 1 = 50% 2 = 25% 3 = 12.5% 4 = 6.25% 5 = 最大重新校准
sensor_touch_drift_rate	1 字节	0（禁止），1-255	靠近触摸漂移之间要倒计时的定时器周期数
sensor_anti_touch_drift_rate	1 字节	0（禁止），1-255	远离触摸漂移之间要倒计时的定时器周期数

.....（续）

参数	大小	范围/选项	使用
sensor_drift_hold_time	1 字节	0（禁止）， 1-255	触摸事件后停止漂移所需的定时器周期数
sensor_reburst_mode	1 字节	0 = 无 1 = 未解析（快速重新突发） 2 = 全部	无——从未设置重新检测，根据应用程序调度进行测量。 未解析——已设置重新检测，所有传感器均已暂停，但与目标传感器采用同一 AKS。 全部——已设置重新检测，传感器均未暂停。

表 9-3. 单个传感器配置

参数	大小	范围/选项	使用
channel_threshold	1 字节	0-255	指示接触点的最小信号增量
channel_hysteresis	1 字节	0（50%）-4（3.125%）	当滤除已移除的接触点时，降低触摸阈值可去抖动
channel_aks_group	1 字节	0-255	对控制同时触摸检测的按键传感器进行分组。

9.4.2 状态和输出数据

表 9-4. 组数据

参数	大小	范围/选项	使用
qtm_keys_status	1 字节	Bit 7：需要重新检测 Bit 6-1：保留 Bit 0：触摸检测	指示触摸按键组的当前状态
acq_group_timestamp	2 字节	0-65535	处理的最后一个漂移周期的时间戳
dht_count_in	1 字节	0-“sensor_drift_hold_time”	触摸事件后漂移保持释放的倒计时
tch_drift_count_in	1 字节	0-“sensor_touch_drift_rate”	下一个靠近触摸漂移周期的倒计时
antitch_drift_count_in	1 字节	0-“sensor_anti_touch_drift_rate”	下一个远离触摸漂移周期的倒计时

单个按键传感器数据

其他后处理模块（如滚动条）需要单个按键传感器数据。因此，此数据结构定义置于 common_compoenents_api.h 文件。

参数	大小	范围/选项	使用
sensor_state	1 字节	位字段	触摸按键传感器状态
sensor_state_counter	1 字节	0-255	确认触摸检测的重复测量次数
*node_data_struct_ptr	2/4 字节	指针	指向节点数据结构数组的指针
Channel_reference	2 字节	0-65535	参考测量，触摸检测的基线

sensor_state	
QTM_KEY_STATE_DISABLE	0x00

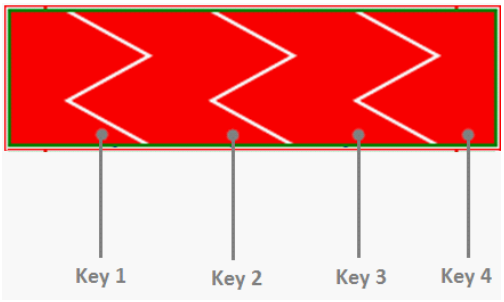
.....（续）	
sensor_state	
QTM_KEY_STATE_INIT	0x01
QTM_KEY_STATE_CAL	0x02
QTM_KEY_STATE_NO_DET	0x03
QTM_KEY_STATE_FILT_IN	0x04
QTM_KEY_STATE_DETECT	0x85
QTM_KEY_STATE_FILT_OUT	0x86
QTM_KEY_STATE_ANTI_TCH	0x07
QTM_KEY_STATE_SUSPEND	0x08
QTM_KEY_STATE_CAL_ERR	0x09
注： 在触摸按键传感器处于“检测中”的每个状态下，Bit 7（0x80u）置 1	

10. 滚动条模块

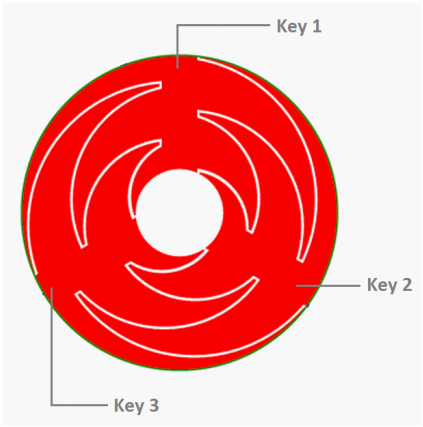
10.1 概述

滚动条模块处理构造为线性滑动条或圆形滚轮的触摸传感器组，如下图所示。滑动条/滚轮传感器也称为一维表面传感器，用于跟踪在其上滚动的触摸运动，并向用户应用程序报告状态和位置。滑动条/滚轮的大小取决于形成线性/圆形表面的触摸按键传感器的基础数量。

Slider Sensor



Wheel Sensor

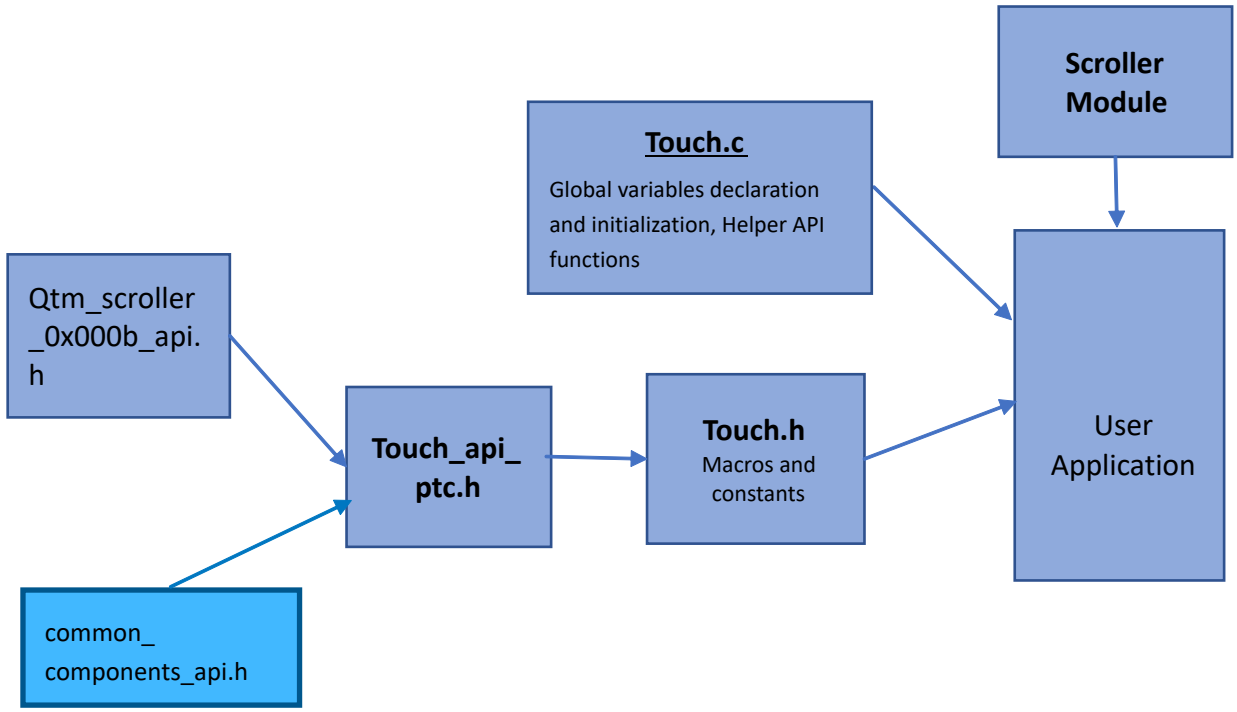


滑动条/滚轮可以通过使用自电容和互电容传感器形成。上图所示为基于自电容技术的 4 通道滑动条和 3 通道滚轮传感器。当在传感器表面上滚动触摸时，为了在报告的触摸位置上获得良好的线性度，触摸按键应采用互相交叉的方式布置，如上图所示。

GCC 编译器：	`libqtm_scroller_xxxxx_0x000B.a`
IAR 编译器（AVR MCU）：	`qtm_scroller_xxxxx_0x000B.r90`
IAR 编译器（ARM MCU）：	`qtm_scroller_xxxxx_0x000B.a`

10.2 接口

数据结构的定义和 API 声明包含在 API 文件“`qtm_scroller_0x000b_api.h`”中。数据结构涵盖所有配置和输出数据变量。该文件应包含在共用 api `touch_ptc_api.h` 文件中。



10.3 功能说明

滚动条模块的处理取决于触摸按键模块输出。处理完按键并在数据结构中更新状态后，滑动条模块会检查这些状态。可基于按键状态，根据采集模块变量可用的当前信号值计算滑动条/滚轮位置。

下面给出了可能的用例以及每种用例的操作顺序。

用例 1：滑动条/滚轮传感器上形成的接触点

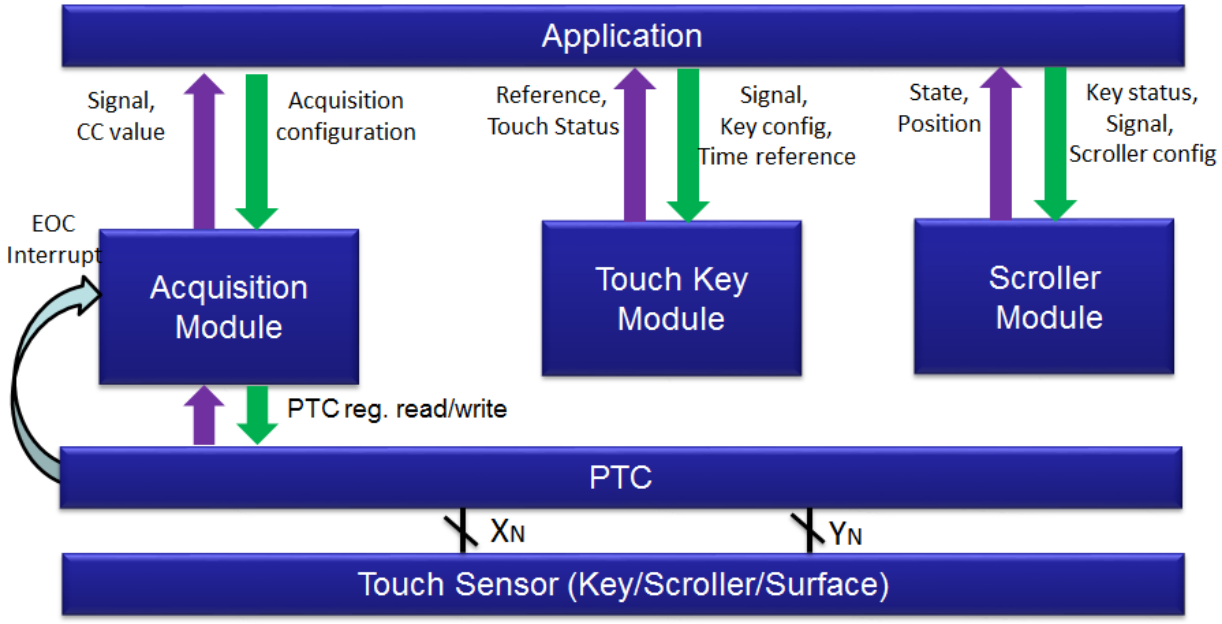
1. 模块检查滚动条中所有按键的状态以进行接触点检测。
2. 如果任何按键处于“检测中”状态，则使用三个相邻按键的信号值计算触摸位置。
3. 同时计算原始位置和滤波位置。
4. 滚动条状态变为“TOUCH_ACTIVE”并且滚动条重新测量标志置 1。
5. “POSITION_CHANGE”标志现在置 1。如果触摸静止且触摸位置没有变化，则在下一个测量周期将该标志清零。

用例 2：在滑动条/滚轮表面上滚动的接触点

1. 模块检查所有按键以确定是否存在接触点
2. 如果没有按键处于“检测中”，则模块会搜索触摸增量超过最小接触阈值的一对相邻按键
3. 如果发现此类接触，则计算新的位置**或者**
4. 如果未发现此类接触，则滚动条将返回“未检测”状态

用例 3：从滑动条/滚轮传感器上移除接触点

1. 模块检查滚动条中所有按键的状态以进行接触点检测。
2. 如果没有按键处于“检测中”，则模块会搜索触摸增量超过最小接触阈值的一对相邻按键。
3. 如果发现此类接触，则计算新的位置**或者**
4. 如果未发现此类接触，则滚动条将返回“未检测”状态。即，将标志“TOUCH_ACTIVE”清零。



10.4 配置

10.4.1 数据结构

表 10-1. 组配置

参数	大小	范围/选项	使用
*qtm_touch_key_data	指针（2/4 字节）	qtm_touch_key_data_t	指向基础触摸按键组的触摸按键数据的指针
num_scrollers	1 字节	1-255	此按键组中实现的滚动条数

表 10-2. 单个传感器配置

参数	大小	范围/选项	使用
type	1 字节	0 = 线性滑动条 1 = 滚轮	滚动条类型
start_key	2 字节	0-65535	形成滚动条第一个组成按键的按键编号
number_of_keys	1 字节	2-255	形成滚动条的组成按键数。构成滑动条所需的最小按键数为 2，构成滚轮所需的最小按键数为 3。
resol_deadband	1 字节	Bit 7:4 = 分辨率（2 至 12 位）	为滚动条报告的满量程位置分辨率
		Bits 3:0 = 死区占 0% 至 15%（每侧）	边缘校正死区的大小（用满量程范围的百分比表示）
position_hysteresis	1 字节	0-255	接触或方向变化后报告的最小行程距离
contact_min_threshold	2 字节	0-65535	持久接触跟踪的最小接触尺寸测量值。接触尺寸是形成接触点的两个相邻按键的触摸增量总和

10.4.2 状态和输出数据

表 10-3. 组数据

参数	大小	范围/选项	使用
scroller_group_status	1 字节	位字段 Bit 7：需要重新测量 Bit 0：触摸检测	需要重新测量 = 1 指示组中的某个滚动条需要传感器重新测量。 触摸检测 = 1 指示组中的某个滚动条处于“触摸检测”状态

单个按键传感器数据

参数	大小	范围/选项	使用
scroller_status	1 字节	位字段 Bit 7：需要重新测量 Bit 1：接触点发生移动 Bit 0：触摸检测	需要重新测量 = 1 指示组中的某个滚动条需要传感器重新测量。 报告的接触点位置发生变化 触摸检测 = 1 指示组中的某个滚动条处于“触摸检测”状态
right_hyst	1 字节	滞后限制	指示接触点何时向“右”（即，增大触摸位置的方向）移动
left_hyst	1 字节	滞后限制	指示接触点何时向“左”（即，减小触摸位置的方向）移动
raw_position	2 字节	0-4095	运动滤波之前接触点的计算位置
position	2 字节	0-4095	运动滤波之后接触点的计算位置
contact_size	2 字节	0-65535	形成接触点的两个相邻按键的触摸增量总和

11. 2D 表面（单指触摸）CS 模块

11.1 概述

此模块提供 2D 触摸表面的功能，支持单触点，适用于触摸屏/触摸板应用。

- 接触点 X 和 Y 位置
- 最多 255 x 255 个传感器
  - 受每行/每列最大补偿电容的限制
- 自电容
- 经优化的交叉滑动条测量
  - （MxN）传感器阵列以（M+N）次采集测量
- 持久接触跟踪
- 位置滤波
  - IIR
  - 中值
  - 滞后
  - 死区

表面 CS 模块适合与以网格形式排列在表面区域上的传感器按键配合使用。

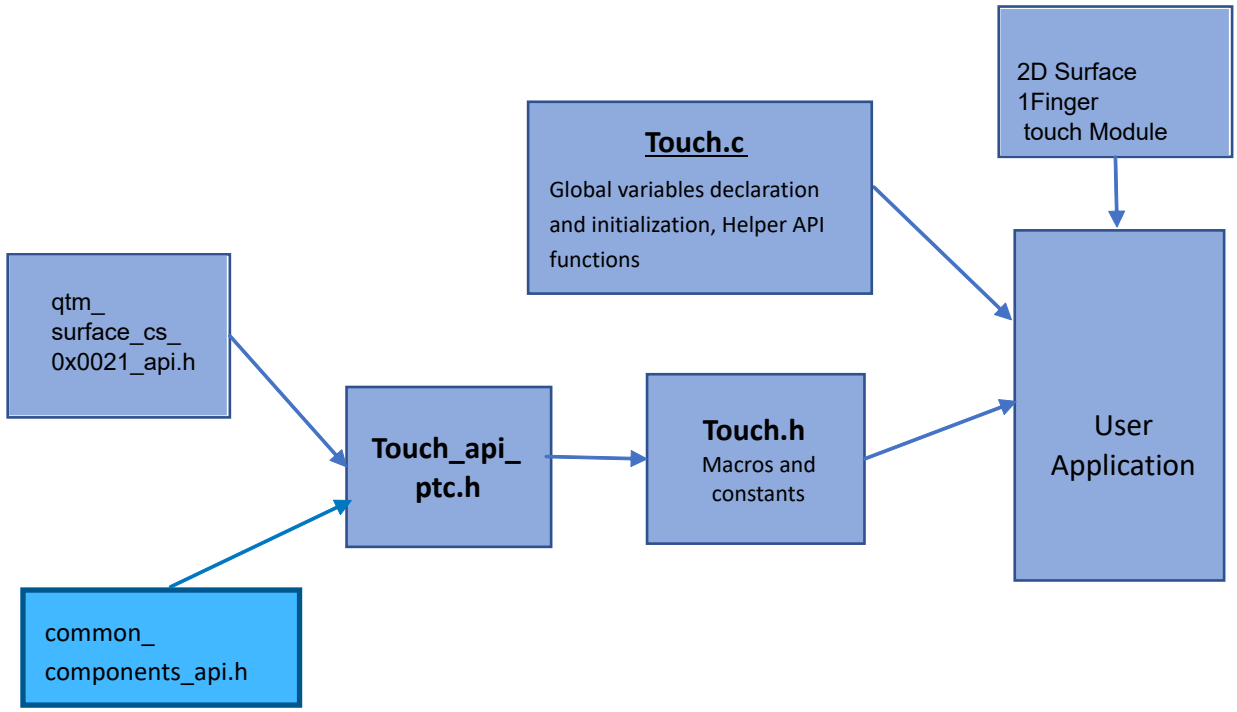
表面 CS 模块配置为与 QTML 触摸按键模块（0x0002）或兼容的触摸检测模块连接。指针需要指向采用公共 API 文件中定义的标准格式的触摸按键数据的位置。

表 11-1. 模块文件

GCC 编译器	例如 Atmel Studio 7	`libqtm_surface_cs_xxxxx_0x0021.a`
IAR 编译器	AVR 器件：	`qtm_surface_cs_xxxxx_0x0021.r90`
	Arm 器件：	`qtm_surface_cs_xxxxx_0x0021.a`
注： *“xxxxx”指目标处理器或架构。*		

11.2 接口

所有用户选项均在应用程序代码（`touch.h`/`touch.c`）中配置，并通过指针引用与库模块共用。



qtm_surface_cs_0x0021_api.h	此头文件包含与模块相关的所有 API 实现。它必须与已编译模块一起包含在应用程序项目中。
qtm_common_components_api.h	此头文件包含所有 QTML 模块可访问的 API 声明，例如节点、按键数据结构和 touch_ret_t 返回代码。

11.3 功能说明

初始化
在使用库之前，必须使用配置加载数据结构。“surface_cs”模块通过调用“qtm_init_surface_cs()”API 来初始化。

支持模块
传感器节点（“acquisition”模块）和按键（“touch_key”模块）必须按照“surface_cs”模块的要求正确配置。

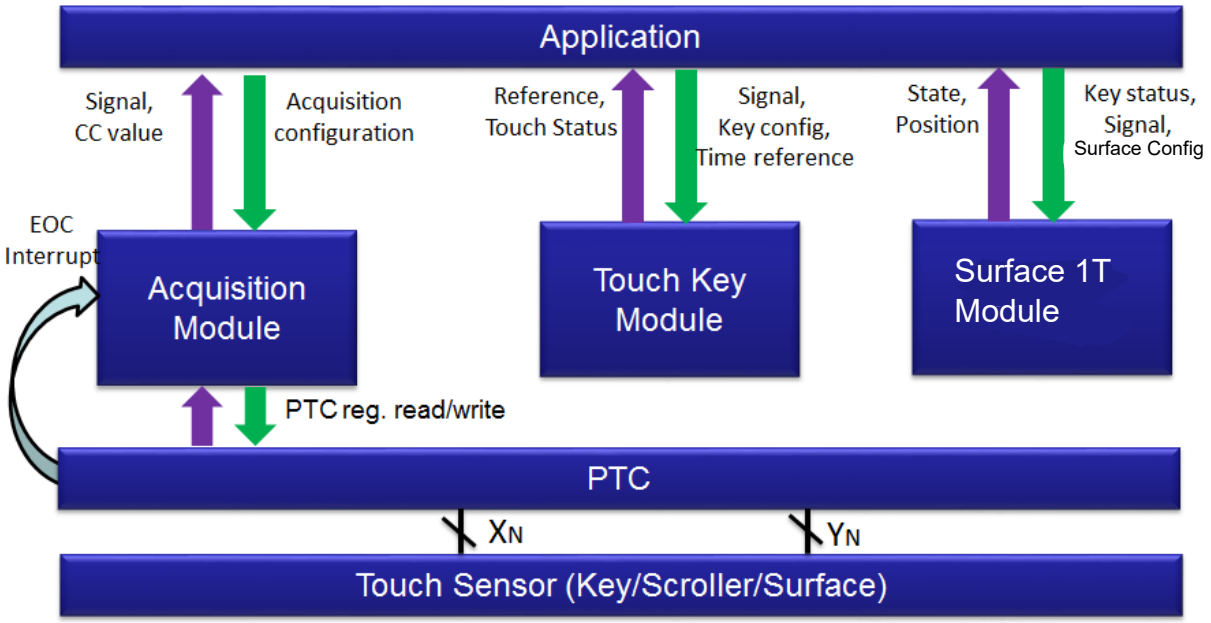
传感器节点配置
表面 CS 要求传感器的配置和分组能够实现一个水平滑动条和一个垂直滑动条。如果 X 引脚水平排列，则垂直滑动条由（全部 X）/一个 Y 组成。类似地，水平滑动条由一个 X/（全部 Y）组成。

运行时操作
表面 CS 用作顶层模块，为应用程序提供接触点信息。它利用触摸按键库模块实现传感器校准、信号漂移、触摸检测和定时功能。触摸按键模块本身使用目标特定的采集模块与电容测量硬件连接。

在 QTML 应用程序中，必须正确调用模块处理顺序：

1. 对所有传感器节点执行测量采集——qtm_ptc_start_measurement_seq();
2. 对所有传感器节点执行采集处理——qtm_acquisition_process();

3. 对所有按键进行触摸按键处理——`qtm_key_sensors_process();`
4. 表面 CS 处理——`qtm_surface_cs_process();`



11.4 操作

在采集和触摸按键处理后调用 API 函数“`qtm_surface_cs_process()`”。

进行接触：

与表面传感器接触时

1. 模块检查表面的所有按键以进行接触点检测。
2. 如果发现按键处于“检测中”，则计算位置。
3. 计算接触尺寸并与最小接触阈值进行比较。
4. 表面变为“检测”状态。

跟踪/释放接触：

1. 模块检查所有按键以进行接触点检测。
2. 如果没有按键处于“检测中”，则模块会搜索触摸增量超过最小接触阈值的一对相邻按键。
3. 如果发现此类接触，则计算新的位置**或者**
4. 如果未发现此类接触，则表面将返回“未检测”状态。

11.5 配置

表面 CS 模块必须配置有操作参数以及指向基础触摸按键的按键数据集的指针。

11.5.1 数据结构

`qtm_surface_cs_control_t`

- 用于表面配置的顶层容器

- 包含指向数据和配置结构的指针

结构	内容
qtm_surface_cs_control_t	qtm_surface_contact_data_t *qtm_surface_contact_data;
	qtm_surface_cs_config_t *qtm_surface_cs_config;

qtm_surface_contact_data_t

- 触摸表面的运行时数据

参数	大小	范围/选项	使用
qt_surface_status	1 字节	位字段 Bit 7：需要重新检测	需要重新检测 = 1 指示需要进一步测量才能解析/更新接触状态。
		Bit 6：—	—
		Bit 5：POS_V_DEC	垂直位置减小
		Bit 4：POS_V_INC	垂直位置增大
		Bit 3：POS_H_DEC	水平位置减小
		Bit 2：POS_H_INC	水平位置增大
		Bit 1：POS_CHANGE	报告的位置发生变化
		Bit 0：触摸检测	触摸检测 = 1 指示表面上存在接触点
h_position_abs	2 字节	0 至 4095	水平视位
h_position	2 字节	0 至 4095	运动滤波后的水平位置
v_position_abs	2 字节	0 至 4095	垂直视位
v_position	2 字节	0 至 4095	运动滤波后的垂直位置
contact_size	2 字节	—	接触位置的触摸增量总和

qtm_surface_cs_config_t

- 触摸表面的配置参数

参数	大小	范围/选项	使用
start_key_h	2 字节	0 至 65534	水平轴的起始按键
number_of_keys_h	1 字节	0 至 255	形成水平轴的按键数
start_key_v	2 字节	0 至 65534	垂直轴的起始按键
number_of_keys_v	1 字节	0 至 255	形成垂直轴的按键数
resol_deadband	1 字节	Bit 7:4 = 分辨率（2 至 12 位）	为轴报告的满量程位置分辨率
position_hysteresis	1 字节	0 至 255	接触或方向变化后报告的最小行程距离。适用于水平和垂直。
position_filter	1 字节	Bit7:5：—	—
		Bit 4：中值滤波器	中值滤波器使能
		Bit3：—	—
		Bit 2：—	—
		Bit 1:0：IIR 配置	IIR 配置 0 = 无 1 = 25% 2 = 50% 3 = 75%

.....（续）

参数	大小	范围/选项	使用
contact_min_threshold	2 字节	0 至 65535	持久接触跟踪的最小接触尺寸测量值。接触尺寸是形成接触点的相邻按键的触摸增量总和。
*qtm_touch_key_data	指针 2/4 字节	qtm_touch_key_data_t	指向基础触摸按键组的触摸按键数据的指针。

12. 2D 表面（双指触摸）CS/2T 模块

12.1 概述

此模块提供 2D 触摸表面的功能，支持双触点，适用于触摸屏/触摸板应用。

- 接触点 X 和 Y 位置
- 最多 255 x 255 个传感器
  - 受每行/每列最大补偿电容的限制
- 互电容和自电容
- 经优化的交叉滑动条测量
  - （MxN）传感器阵列以（M+N）次采集测量
- 持久接触跟踪
- 接触路径外推
- 位置滤波
  - IIR
  - 中值
  - 滞后
  - 死区

表面 CS/2T 模块适合与以网格形式排列在表面区域上的传感器按键配合使用。

表面 CS/2T 模块配置为与 QTML 触摸按键模块（0x0002）或兼容的触摸检测模块连接。指针需要指向采用公共 API 文件中定义的标准格式的触摸按键数据的位置。

重要： 该 2D 表面（双指触摸）CS/2T 模块输出的两个位置仅可用于手势检测。因此，不建议在不支持手势的情况下使用此模块。

要使用两个接触点实现可靠的性能，建议每个维度至少使用 8 个传感器。使用的传感器较少会严重限制独立接触点的隔离空间。

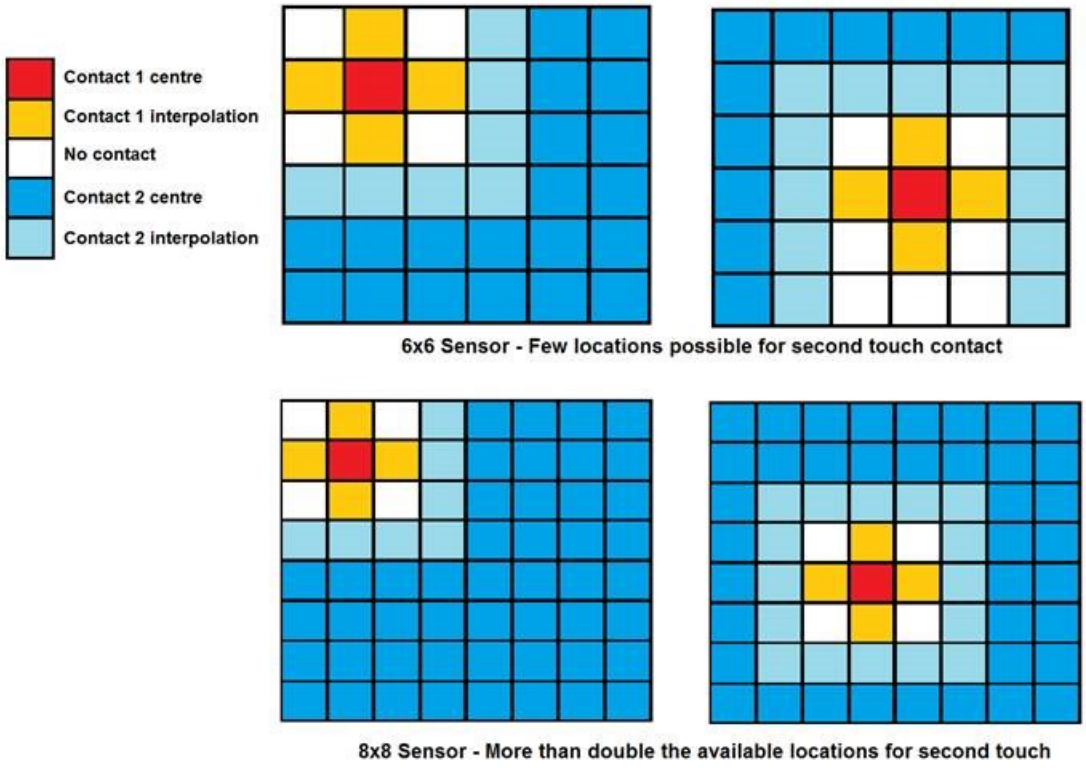
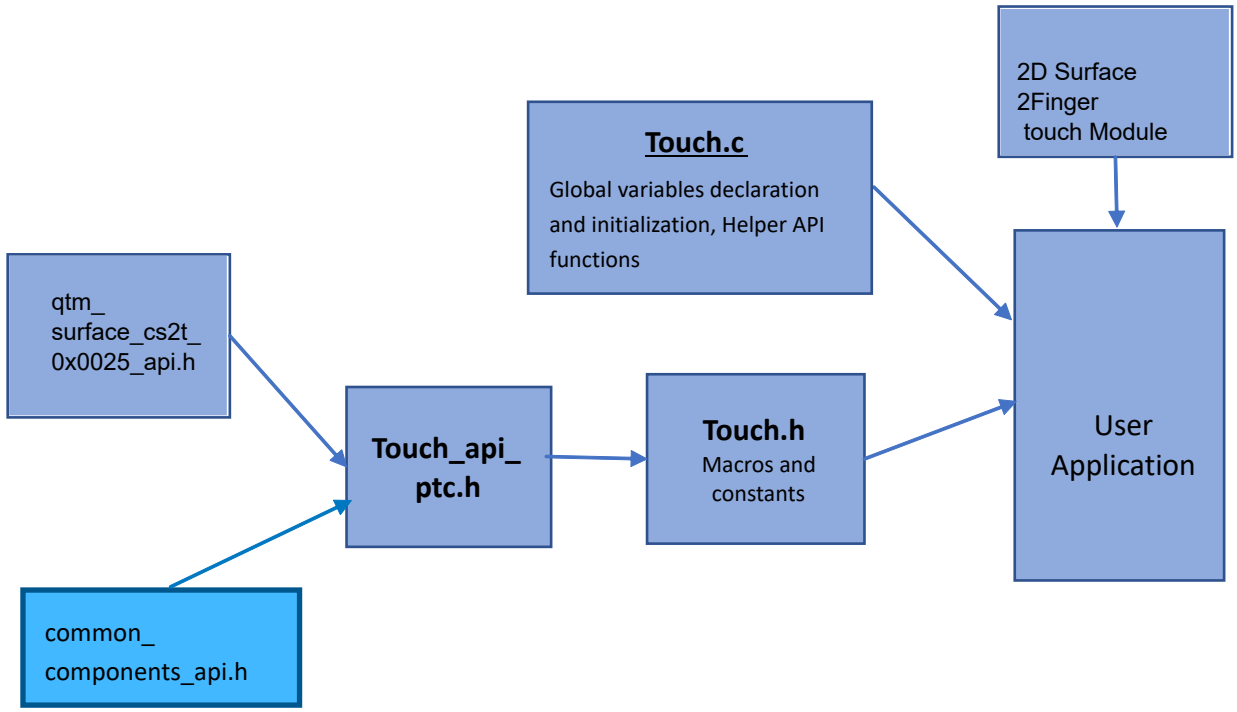


表 12-1. 模块文件

GCC 编译器	例如 Atmel Studio 7	libqtm_surface_cs2t_xxxxx_0x0025.a
IAR 编译器	AVR 器件：	qtm_surface_cs2t_xxxxx_0x0025.r90
	Arm 器件：	qtm_surface_cs2t_xxxxx_0x0025.a
注： *“xxxxx”指目标处理器或架构。*		

12.2 接口

所有用户选项均在应用程序代码（touch.h/touch.c）中配置，并通过指针引用与库模块共用。



qtm_surface_cs2t_0x0025_api.h	此头文件包含与模块相关的所有 API 实现。它必须与已编译模块一起包含在应用程序项目中。
qtm_common_components_api.h	此头文件包含所有 QTML 模块可访问的 API 声明，例如节点、按键数据结构和 touch_ret_t 返回代码。

12.3 功能说明

初始化
在使用库之前，必须使用配置加载数据结构。“surface_cs2t”模块通过调用“qtm_init_surface_cs2t()”API 来初始化。

支持模块
传感器节点（“acquisition”模块）和按键（“touch_key”模块）必须按照“surface_cs”模块的要求正确配置。

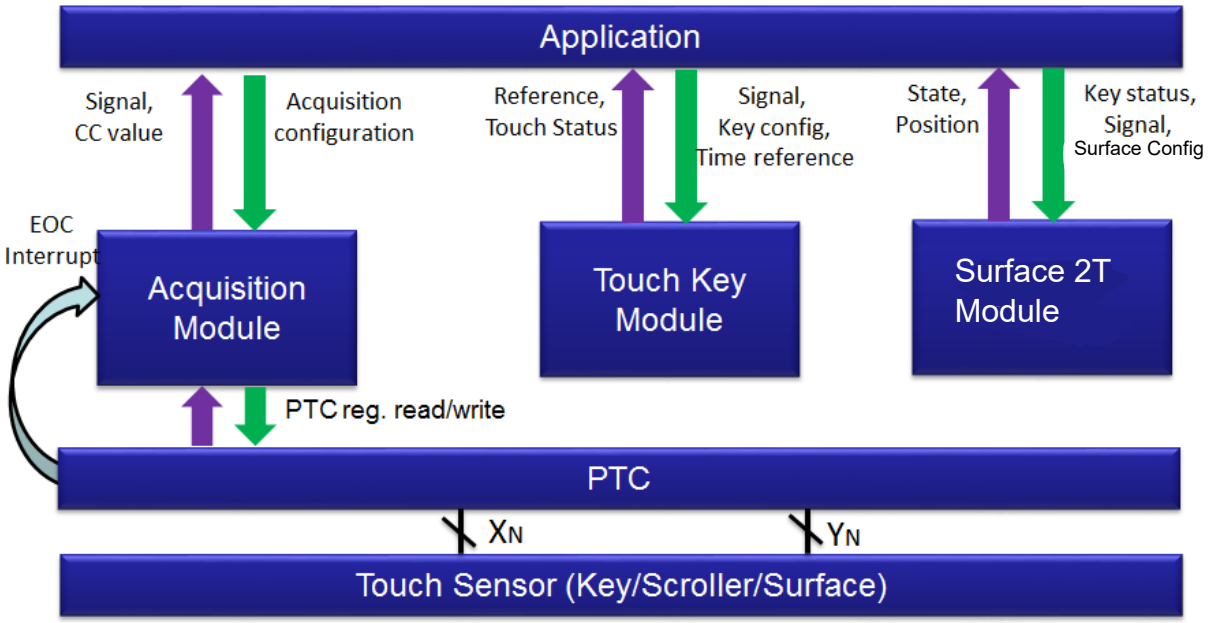
传感器节点配置
表面 CS/2T 要求传感器的配置和分组能够实现一个水平滑动条和一个垂直滑动条。如果 X 引脚水平排列，则垂直滑动条由（全部 X）/一个 Y 组成。类似地，水平滑动条由一个 X/（全部 Y）组成。

运行时操作
表面 CS/2T 用作顶层模块，为应用程序提供接触点信息。它利用触摸按键库模块实现传感器校准、信号漂移、触摸检测和定时功能。触摸按键模块本身使用目标特定的采集模块与电容测量硬件连接。

在 QTML 应用程序中，必须正确调用模块处理顺序：

1. 对所有传感器节点执行测量采集——qtm_ptc_start_measurement_seq();
2. 对所有传感器节点执行采集处理——qtm_acquisition_process();

3. 对所有按键进行触摸按键处理——`qtm_key_sensors_process();`
4. 表面 CS 处理——`qtm_surface_cs2t_process();`



12.4 操作

在采集和触摸按键处理后调用 API 函数“`qtm_surface_cs2t_process()`”。

进行接触：

与表面传感器接触时

1. 模块检查表面的所有按键以进行接触点检测。
2. 如果发现按键处于“检测中”，则计算位置。
3. 计算接触尺寸并与最小接触阈值进行比较
4. 表面变为“检测”状态。

跟踪/释放接触：

1. 模块检查所有按键以进行接触点检测。
2. 如果没有按键处于“检测中”，则模块会搜索触摸增量超过最小接触阈值的一对相邻按键。
3. 如果发现此类接触，则计算新的位置**或者**
4. 如果未发现此类接触，则表面将返回“未检测”状态。

12.5 配置

表面 CS/2T 模块必须配置有操作参数以及指向基础触摸按键的按键数据集的指针。

12.5.1 数据结构

`qtm_surface_cs2t_control_t`

- 用于表面配置的顶层容器

- 包含指向数据和配置结构的指针

结构	内容
qtm_surface_cs2t_control_t	qtm_surface_cs2t_data_t *qtm_surface_cs2t_data;
	qtm_surface_contact_data_t *qtm_surface_contact_data;
	qtm_surface_cs_config_t *qtm_surface_cs_config;

qtm_surface_cs2t_data_t

- 表面 CS/2T 模块的运行时数据

参数	大小	范围/选项	使用
qt_surface_cs2t_status	1 字节	位字段 Bit 7：需要重新检测	需要重新检测 = 1 指示需要进一步测量才能解析/更新接触状态。
		Bit 6：—	—
		Bit 5：POS_MERGED_V	存在两个接触点，垂直位置太靠近而不能分离。
		Bit 4：POS_MERGED_H	存在两个接触点，水平位置太靠近而不能分离。
		Bit 3 —	—
		Bit 2：—	—
		Bit 1：—	—
		Bit 0：触摸检测	触摸检测 = 1 指示表面上存在接触点。

qtm_surface_contact_data_t

- 各个接触点（阵列）的运行时数据

参数	大小	范围/选项	使用
qt_contact_status	1 字节	位字段 Bit 7：—	—
		Bit 6：—	—
		Bit 5：POS_V_DEC	垂直位置减小
		Bit 4：POS_V_INC	垂直位置增大
		Bit 3：POS_H_DEC	水平位置减小
		Bit 2：POS_H_INC	水平位置增大
		Bit 1：POS_CHANGE	报告的位置发生变化
		Bit 0：触摸检测	触摸检测 = 1 指示表面上存在接触点。
h_position_abs	2 字节	0 至 4095	未滤波的水平位置
h_position	2 字节	0 至 4095	滤波后的水平位置
v_position_abs	2 字节	0 至 4095	未滤波的垂直位置
v_position	2 字节	0 至 4095	滤波后的垂直位置
contact_size	2 字节	—	接触位置的触摸增量总和

qtm_surface_cs_config_t

- 触摸表面的配置参数

参数	大小	范围/选项	使用
start_key_h	2 字节	0 至 65534	水平轴的起始按键

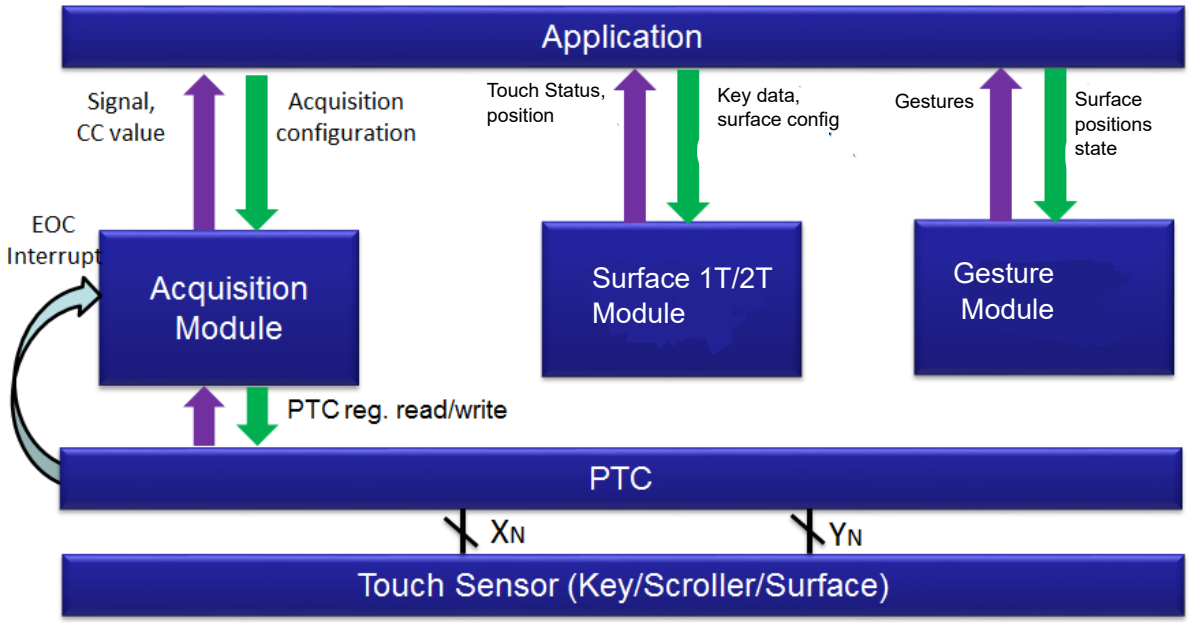
.....（续）

参数	大小	范围/选项	使用
number_of_keys_h	1 字节	0 至 255	形成水平轴的按键数
start_key_v	2 字节	0 至 65534	垂直轴的起始按键
number_of_keys_v	1 字节	0 至 255	形成垂直轴的按键数
resol_deadband	1 字节	Bit 7:4 = 分辨率（2 至 12 位）	为每个轴报告的满量程位置分辨率
position_hysteresis	1 字节	0 至 255	接触或方向变化后报告的最小行程距离。适用于水平和垂直。
position_filter	1 字节	Bit 7:5：—	—
		Bit 4：中值滤波器	中值滤波器使能
		Bit 3：—	—
		Bit 2：—	—
		Bit 1:0：IIR 配置	IIR 配置 0 = 无 1 = 25% 2 = 50% 3 = 75%
contact_min_threshold	2 字节	0 至 65535	持久接触跟踪的最小接触尺寸测量值。接触尺寸是形成接触点的相邻按键的触摸增量总和。
*qtm_touch_key_data	指针 2/4 字节	qtm_touch_key_data_t	指向基础触摸按键组的触摸按键数据的指针。

13. 手势模块

13.1 概述

手势模块基于从表面模块接收的触摸位置识别手势并报告检测到的手势。此模块处理水平和垂直触摸位置，并报告是否识别出任何手势。



模块配置为与 QTouch 表面模块连接。需要指向表面模块接触数据的指针来识别手势。

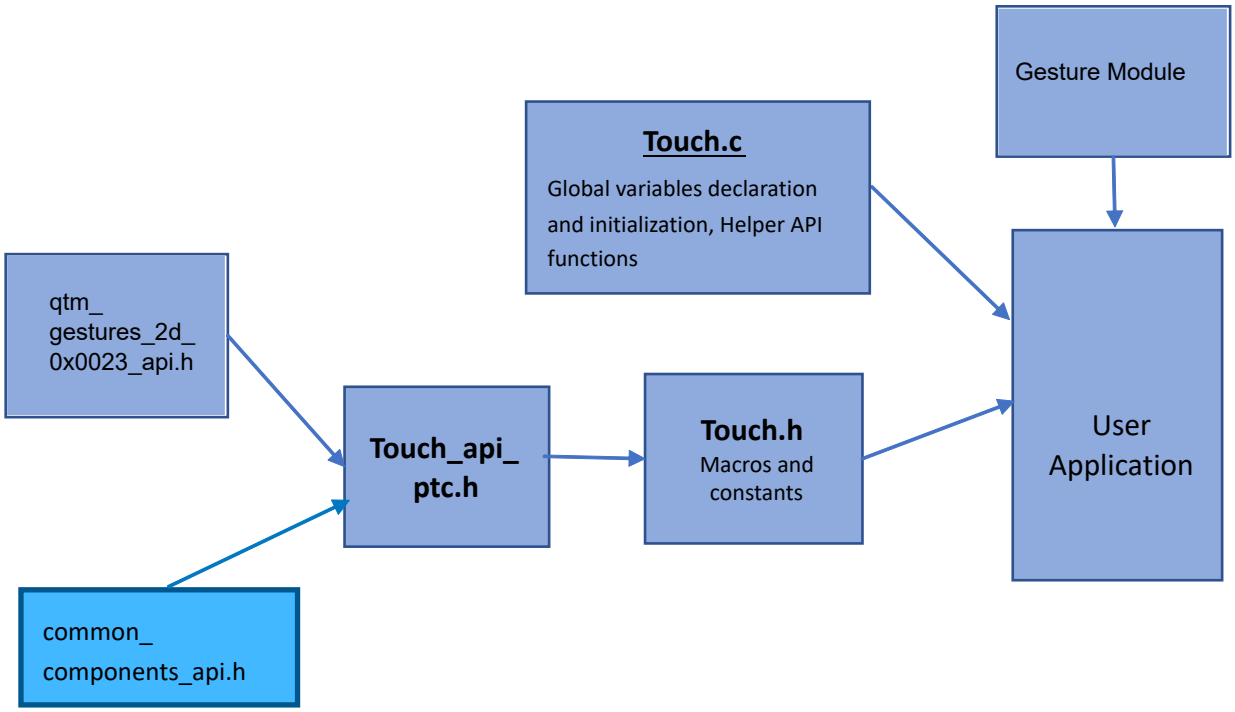
表 13-1. 模块文件

GCC 编译器	SAM 器件：	libqtm_surface_gestures_cm0p_0x0023.a
	AVR 器件：	libqtm_surface_gestures_t1614_0x0023.a libqtm_surface_gestures_t1616_0x0023.a libqtm_surface_gestures_t1617_0x0023.a libqtm_surface_gestures_t3214_0x0023.a libqtm_surface_gestures_t3216_0x0023.a libqtm_surface_gestures_t3217_0x0023.a
IAR 编译器	SAM 器件：	qtm_surface_gestures_cm0p_0x0023.r90
	AVR 器件：	qtm_surface_gestures_t1614_0x0023.r90 qtm_surface_gestures_t1616_0x0023.r90 qtm_surface_gestures_t1617_0x0023.r90 qtm_surface_gestures_t3214_0x0023.r90 qtm_surface_gestures_t3216_0x0023.r90 *qtm_surface_gestures_t3217_0x0023.r90*

13.2 模块的接口

由于此模块依赖于表面模块，因此需要指向表面接触数据的指针来读取表面触摸位置和状态，从而处理手势。

所有用户选项均在应用程序代码（touch.h/touch.c）中配置，并通过指针引用与库模块共用。



qtm_gestures_2d_0x0023_api.h	此头文件包含与手势模块相关的所有 API 实现。它应与已编译模块一起包含在应用程序项目中。
qtm_common_components_api.h	此头文件包含所有 QTML 模块可访问的 API 声明，例如节点、按键数据结构和 touch_ret_t 返回代码。

13.3 配置

13.3.1 数据结构

qtm_gestures_2d_control_t

qtm_gestures_2d_control_t 数据接口是一个容器结构，用于控制该模块的输入和输出。

字段	单位	范围/选项	参数
qtm_gestures_2d_data	qtm_gestures_2d_data _t*	指针	指向手势数据结构的指针
qtm_gestures_2d_config	qtm_gestures_2d_config _t*	指针	指向手势配置结构的指针

qtm_gestures_2d_config _t

qtm_gestures_2d_config_t 数据结构是传送给模块的配置结构。

字段	单位	范围/选项	参数
horiz_position0	uint16_t	指针	指向水平接触点 0 位置的指针

.....（续）

字段	单位	范围/选项	参数
vertical_position0	uint16_t	指针	指向垂直接触点 0 位置的指针
surface_status0	uint8_t	指针	指向接触点 0 的状态的指针
horiz_position1	uint16_t	指针	指向水平接触点 1 位置的指针
vertical_position1	uint16_t	指针	指向垂直接触点 1 位置的指针
surface_status1	uint8_t	指针	指向接触点 1 的状态的指针
surface_resolution	uint8_t	0 至 255	此参数定义表面的分辨率
tapReleaseTimeout	uint8_t	0 至 255	此参数限制手指最初按下和抬起之间允许的时长。超过此值将导致固件不将手势视为单击手势。 **注：** 此值应小于 tapHoldTimeout 和 swipeTimeout
tapHoldTimeout	uint8_t	0 至 255	如果手指在 TAP_AREA 设定的范围内停留而不移开，一旦手势定时器超过此值，固件将报告“单击并按住”手势。 **注：** 此值应大于 tapReleaseTimeout 和 swipeTimeout
swipeTimeout	uint8_t	0 至 255	此值限制滑动手势（从手指最初按下，到沿特定方向移动，再到超过距离阈值后抬起的过程）所允许的时长。 **注：** 此值应大于 tapReleaseTimeout 且小于 tapHoldTimeout
xSwipeDistanceThreshold	uint8_t	0 至 255	此参数控制检测左右滑动手势时沿 X 轴方向行进的距离。
ySwipeDistanceThreshold	uint8_t	0 至 255	此参数控制检测上下滑动手势时沿 Y 轴方向行进的距离。
edgeSwipeDistanceThreshold	uint8_t	0 至 255	此参数控制边缘滑动手势的行进距离。
tapDistanceThreshold	uint8_t	0 至 255	此参数将手指限制在其必须停留的区域，以便在移开手指时认为执行的是“单击”手势；在一段时间内未移开手指时则认为执行的是“单击并按住”手势。
seqTapDistanceThreshold	uint8_t	0 至 255	此参数限制当前触摸的初始按下位置与上一次触摸的抬起位置的允许距离。它适用于多次单击（双击和三击等）。 如果第一次单击之后的单击不超过此阈值，则单击计数器将递增。 如果后续单击手势超过此阈值，则先前的触摸将作为单击发送，当前触摸将复位单击计数器。
edgeBoundary	uint8_t	0 至 255	此外，也可以修改固件以沿着触摸传感器的边界定义边缘区域。定义后，会将在边缘区域中开始的滑动手势报告为边缘滑动手势，而不是正常的滑动手势。
wheelPostscaler	int8_t	-128 至 127	此参数调整在 GUI 中更新滚轮手势的速率。
wheelStartQuadrantCount	int8_t	-128 至 127	滚轮手势的运动可以分解为多个 90°弧。在开始报告滚轮手势信息之前，固件需观察是否有一定数量的弧形以圆形图案的形式出现。此参数确定必须首先检测的弧形数。 此参数的值越小，启动滚轮手势的速度越快，但也会使固件容易过早报告滚轮手势信息。
wheelReverseQuadrantCount	int8_t	-128 至 127	此参数的功能与 wheelStartQuadrantCount 类似，但它会在改变滚轮方向而不是启动新滚轮时使用。这用于防止快速切换方向。

.....（续）

字段	单位	范围/选项	参数
pinchZoomThreshold	uint8_t	0 至 255	此参数限制要检测缩放手势的两个手指之间的允许距离。超过此参数值后，如果接触点之间的距离缩小，则手势将报告为“缩小”。超过此参数值后，如果接触点之间的距离增大，则手势将报告为“放大”。

qtm_gestures_2d_data_t

qtm_gestures_2d_data_t 数据结构在模块内部用于识别手势以及存储状态和信息。

字段	单位	范围/选项	参数
gestures_status	uint8_t	0 或 1	此参数指示手势是否已被解码。
gestures_which_gesture	uint8_t	0 至 255	此参数包含当前已解码的手势。
gestures_info	uint8_t	0 至 255	此参数包含附加的手势信息。

14. 绑定层模块

14.1 概述

绑定层是绑定 QTouch 库模块并自动初始化和处理模块的通用框架。绑定层配置有 QTouch 模块的数据指针和函数指针，这些指针用于以适当的顺序执行模块 API 函数。绑定模块还在完成每个阶段时向用户应用程序提供回调。

绑定包括采集模块、信号调理模块和后处理模块。使用统一的应用程序接口控制所有模块降低了处理多个模块、模块状态和错误以及回调函数的复杂性。用户应用程序代码也可构建为库模块，并可使用绑定层实现自动化，前提是用户模块符合 QTouch 模块化库架构。

图 14-1. 绑定层框架框图

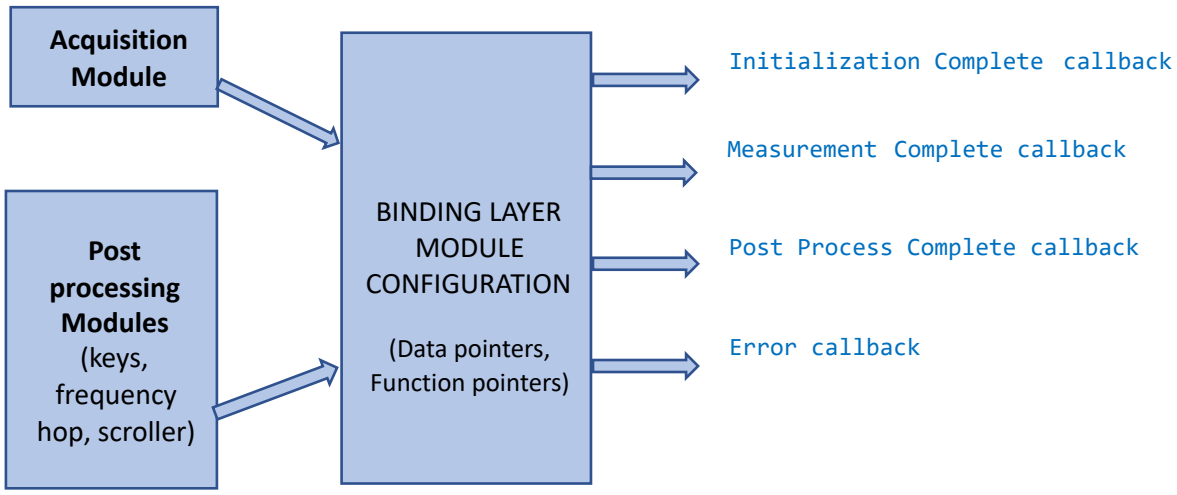
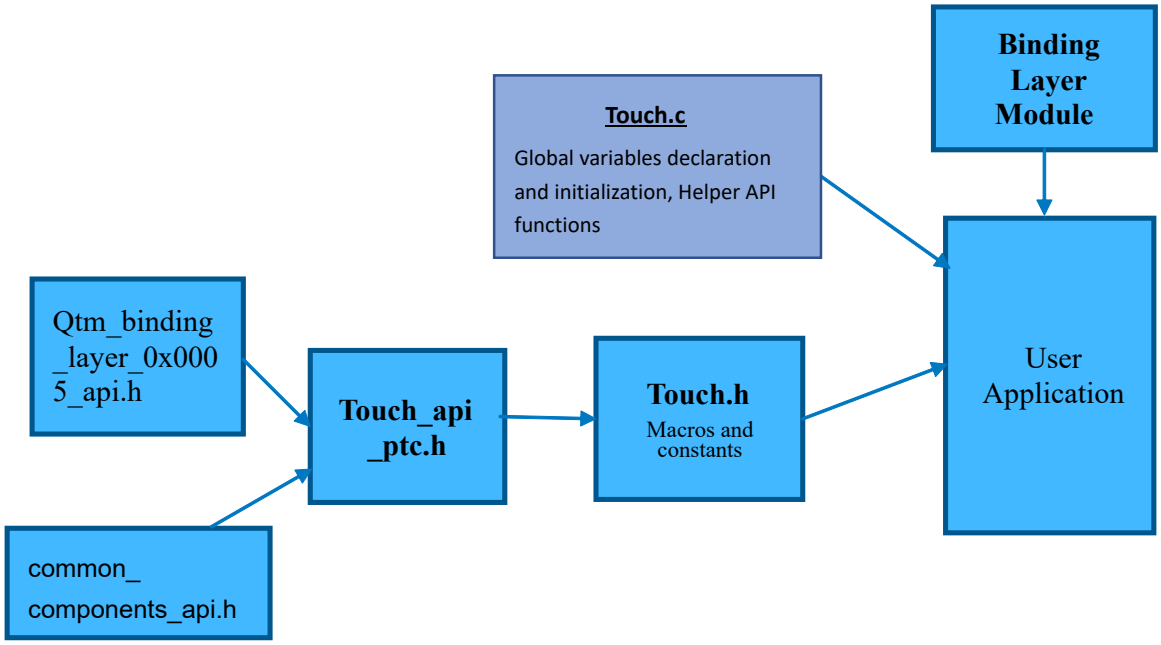


表 14-1. 模块格式

GCC 编译器：	`libqtm_binding_layer_xxxxx_0x0005.a`
IAR 编译器（AVR MCU）：	`qtm_binding_layer_xxxxx_0x0005.r90`
IAR 编译器（ARM MCU）：	`qtm_binding_layer_xxxxx_0x0002.a`

14.2 接口

数据结构的定义和 API 声明包含在 API 文件“`qtm_binding_layer_0x0005_api.h`”中。数据结构涵盖所有配置和输出数据变量。该文件应包含在共用 api `touch_ptc_api.h` 文件中。

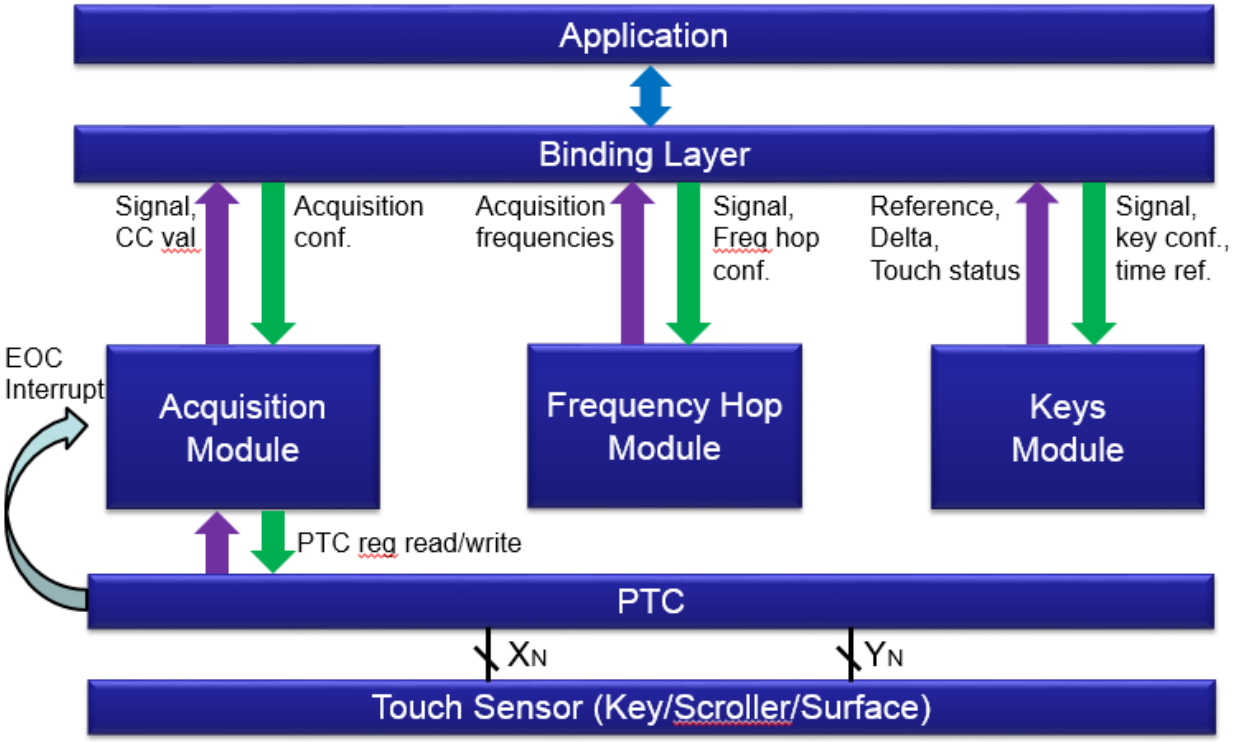


14.3 功能说明

绑定层自动执行每个模块的以下过程。

1. 模块初始化
2. 捕捉成功/错误并通过回调报告
3. 模块后处理
4. 捕捉成功/错误并通过回调报告
5. 捕捉“模块重新检测”标志并根据“重新检测”状态重新触发采集

图 14-2. 基于绑定层的 QTouch®应用程序



绑定层模块的错误处理支持：
各个模块错误会在绑定层内进行验证并完成编码，然后作为单个错误代码传送给应用程序。

错误代码在 touch.c 文件中解码并显示在 Data Visualizer 软件上。错误代码格式如下。

```
Acquisition Module Error codes: 0x8<error code>
    0x81 - Qtm init
    0x82 - start acq
    0x83 - cal sensors
    0x84 - cal hardware

Post processing Modules error codes: 0x4<process_id>
    0x40, 0x41, 0x42, ...
process_id is the sequence of process IDs listed in #define LIB_MODULES_PROC_LIST
macro.Process IDs start from zero and maximum is 15

        Examples:
        0x40 -> error in post processing module 1
        0x42 -> error in post processing module 3

Decoded Module_error_codes:
    Acquisition module error = 1
    post processing module1 error = 2
    post processing module2 error = 3
    ... and so on
```

14.4 配置

14.4.1 数据结构

下面给出了保存绑定层整个配置的容器结构。

```
typedef struct qtm_control_tag
{
    uint8_t binding_layer_flags;

    module_init_t *library_modules_init;
    module_proc_t *library_modules_proc;
    module_acq_t  *library_modules_acq;

    module_arg_t *library_module_init_data_model;
    module_arg_t *library_module_proc_data_model;
    module_arg_t *library_modules_acq_dm;

    qtm_acq_pp_t *qtm_acq_pp;

    /*****************************/
    /* Callbacks for Binding layer */
    /*****************************/
    qtm_library_init_complete_t qtm_init_complete_callback;
    qtm_error_callback_t        qtm_error_callback;
    qtm_measure_complete_t      qtm_measure_complete_callback;
    qtm_pre_process_callback_t  qtm_pre_process_callback;
    qtm_post_process_callback_t qtm_post_process_callback;

} qtm_control_t;
```

参数	说明
*library_modules_init	指向包含模块初始化函数指针列表的数组的指针。
*library_modules_proc	指向包含模块后处理函数指针列表的数组的指针。
*library_modules_acq	指向包含采集模块函数指针列表的数组的指针。
*library_module_init_data_model	指向包含采集模块数据指针的数组的指针。
*library_module_proc_data_model	指向包含后处理模块数据指针的数组的指针。
*library_modules_acq_dm	指向包含采集组指针的数组的指针。
qtm_init_complete_callback	执行所有模块初始化后绑定层模块提供的回调。
qtm_error_callback	仅当绑定层在模块进程期间发生任何错误时才会触发的回调函数。
qtm_measure_complete_callback	测量完成后、后处理开始前由绑定层模块触发的回调。
qtm_pre_process_callback	采集过程后、后处理开始前触发的回调。此参数使用户能够实现定制滤波模块。
qtm_post_process_callback	模块的所有后处理完成后由绑定层模块触发的回调。

14.4.2 状态和输出数据

参数	说明
binding_layer_flags	在绑定层回调函数内设置三个状态标志，以进一步处理应用程序。 当前版本支持三个绑定层标志，如下所示。
	time_to_measure_touch： 当请求任何模块重新检测时，该标志在定时器 ISR 处理程序中置 1。该标志用于在满足上述任一条件的情况下触发测量。
	node_pp_request： 该标志在测量完成回调中设置以指示是否需要后处理。该标志在 touch_process 函数中处理。
	reburst_request： 该标志在后处理完成回调中设置，并将根据各个模块的重新检测标志进行设置。该标志在 touch_process 函数中处理。

15. 使用 Atmel START 构建应用程序

Atmel START 帮助用户选择和配置 Microchip MCU 的软件组件。可以使用 Atmel START 创建 QTouch 项目。用户可以添加传感器并配置以图形方式表示的 QTouch 参数。创建的项目支持 GCC 和 IAR 编译器。

有关如何使用 Amel START 平台创建 QTouch 项目的更多信息，请访问以下链接：http://microchipdeveloper.com/touch:introduction-to-qtouch-project-creation.

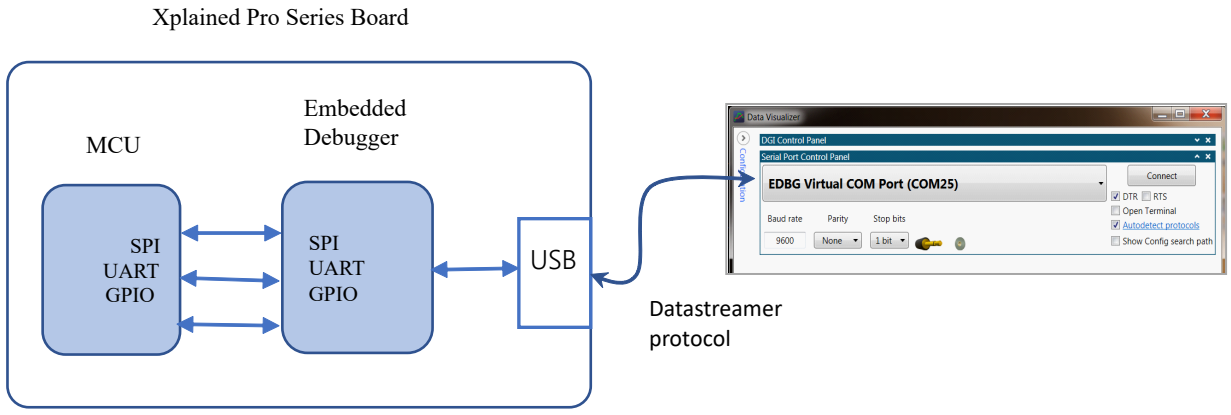
有关使用 START 平台为 SAML1x 器件创建触摸项目的信息，请访问以下链接：http://microchipdeveloper.com/touch:generate-saml1x-touch-project.

16. 将 Data Visualizer 与 QTouch®应用程序配合使用

16.1 概述

Data Visualizer（DV）是用于处理和可视化来自目标硬件的运行时数据的程序。Data Visualizer 可以从嵌入式调试器数据网关接口（Data Gateway Interface，DGI）和串行端口（COM 端口）等各种来源接收数据。

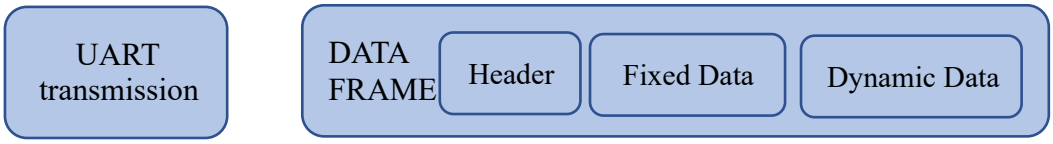
Data Visualizer 与目标硬件的典型连接模型如下所示。



16.2 Datastreamer 模块

Datastreamer 模块嵌入了简单的单向数据传输协议和传输到 Data Visualizer 软件的数据帧。当前版本的 datastreamer 仅支持 UART 端口通信。

图 16-1. Datastreamer 模块框图



UART 传输

UART 传输功能取决于器件，Atmel START 会自动选择正确的驱动程序并将其包含在用户板/工具包示例项目中。所有器件都使用简单的异步模式（非中断驱动）驱动程序。

数据帧：

数据帧包含报头、固定模块数据和动态模块数据字节。

报头详细信息：

报头包含 19 个字节，需要作为数据包的一部分进行传输。不需要在每个数据包上均发送报头，而是每发送 15 个数据包发送一次报头。下面列出了数据包报头的详细信息。

```
// uint8_t data[] =
// {
```

```
//      0x5F,
//      0xB4, 0x00, 0x86, 0x4A,
//      0x51, 0x54, 0x38, 0x31, 0x37, 0x54, 0x4F, 0x55, 0x43, 0x48, 0x55, 0xAA,
//      0xF9,
//      0xA0
// };
```

字节	说明
字节 0	开始令牌。包含固定值“0x5F”
字节 1 至 14	校验和类型。对应于 LRC8（数据包的 XOR 总和，不包括开始和结束令牌）
字节 5 至 16	GUID，目标硬件的标识符
字节 17	数据包报头的校验和
字节 18	结束令牌。包含固定值“0xA0”

固定模块数据：

1. 所有已配置按钮传感器的基本按钮传感器数据
2. 错误状态数据。

动态模块数据：

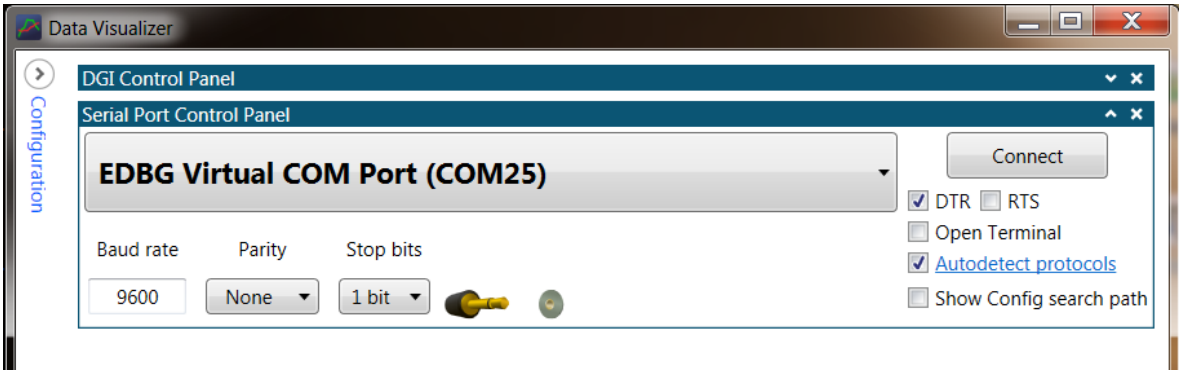
1. 在 Atmel START QTouch 配置器中使能自动调节时，包含采集自动调节参数。
2. 根据在 Atmel START 上完成的配置，包含跳频自动调节数据。
3. 在 Atmel START 上配置滑动条/滚轮传感器时，会传输滚动条模块参数。

16.3 使用 Data Visualizer 进行调试

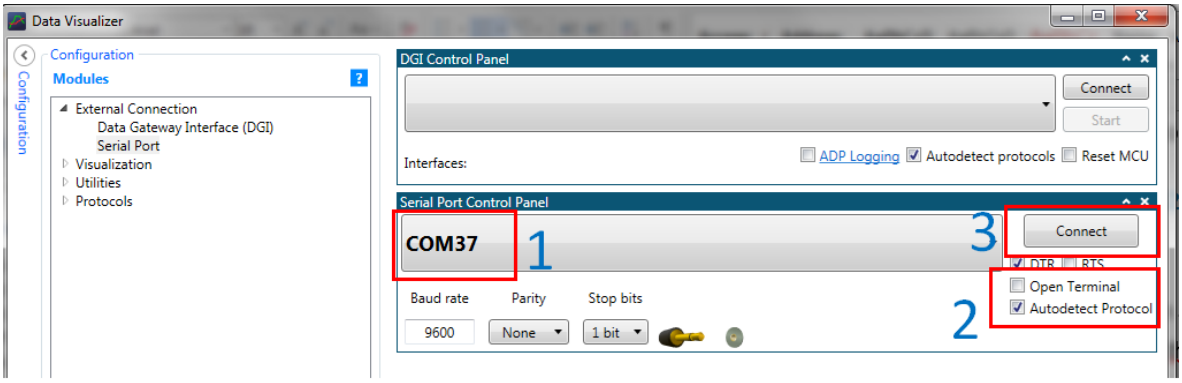
Data Visualizer 支持许多用于可视化终端、标签和图形等数据的小部件。连续数据及其类型使用三个扩展名分别为 `*.db`、`*.ds` 和`*.sc` 的脚本文件解析并显示在相应元素上。这些脚本文件由 Atmel START 平台根据项目配置自动生成，只需要在 Data Visualizer 软件上配置脚本的路径。Data Visualizer 软件既可作为独立可安装版本，也可作为 Atmel Studio IDE 的扩展，可从以下链接下载：https://gallery.atmel.com/Products/Details/0b2891f4-167a-49fc-b3f0-b882c7a11f98。

下面给出了调试所使用的步骤。

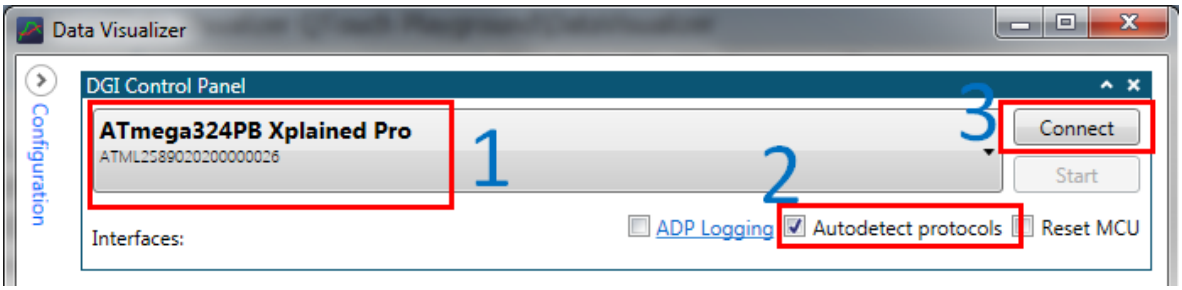
1. 在所需位置为 Data Visualizer 创建一个配置文件夹“dv_config”。
2. 将仪表板配置（.db、.ds 和.sc）文件从“..\thirdparty\qtouch\datastreamer”项目文件夹复制到“dv_config”文件夹。
   注： 这些文件不是源文件，不会自动提取到项目文件夹中。要提取这些文件，请将 `selfcap_3ch.atzip` 文件重命名为 `selfcap_3ch.zip`。然后提取内容。
3. 打开 **Data Visualizer**。
   注： 如果使用 SPI 或 I^2C 接口发送 QTouch 调试数据，则转到步骤 6。如果使用 COM 端口发送 QTouch 调试数据，请继续。
4. 双击 Serial Port Control Panel（串行端口控制面板），然后单击“**Connect**”（连接）按钮与目标建立连接。关闭 DGI Control Panel（DGI 控制面板）选项卡。



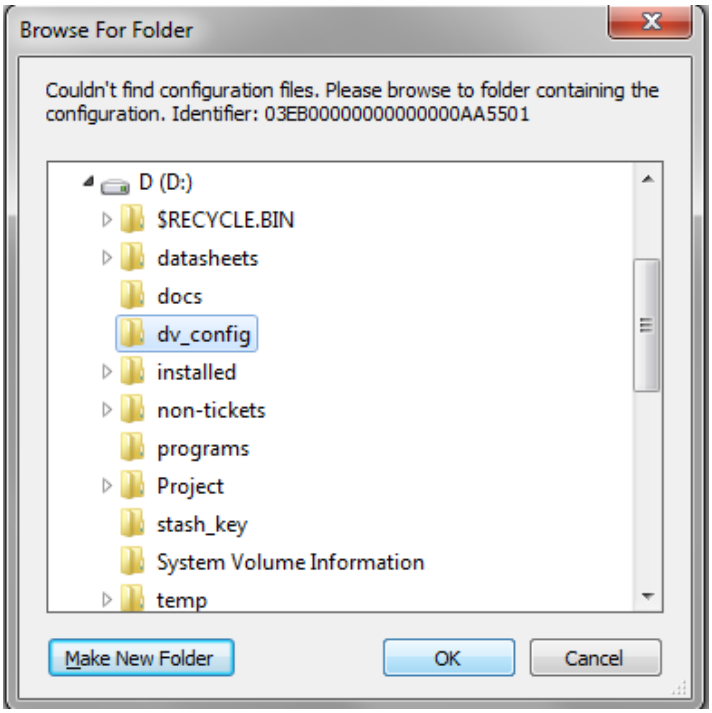
5. 另一种建立连接的方法是通过左侧的 **Configuration**（配置）选项。展开 Configuration 选项，在 **External Connection**（外部连接）选项下，双击 **Serial Port**（串行端口）选项，然后单击 Serial Port Control Panel 上的“**Connect**”（连接）按钮。将配置选项还原为最小化状态



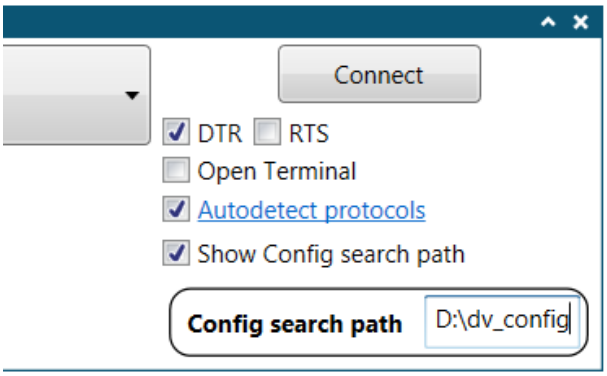
6. 选择所需的工具包，选择 **Autodetect protocol**（自动检测协议）复选框并单击 **Connect**。



7. Data Visualizer 第一次会提示用户选择包含配置信息的文件夹，如下所示。浏览并选择“**dv_config**”文件夹，单击 **OK**（确定）。



也可在 config_path 选项卡中指定配置文件夹“dv_config”的路径，如下所示。单击复选框选项“**show config search path**”（显示配置搜索路径）使能配置路径选项卡。

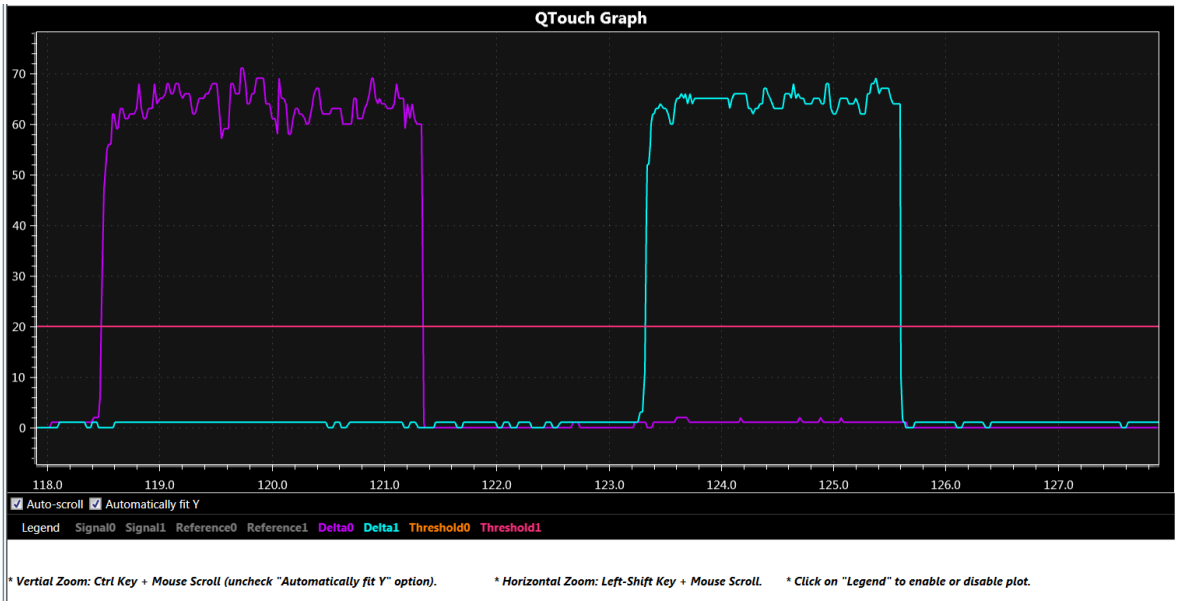


注： 所选文件夹将由 Data Visualizer 保存。Data Visualizer 不会提示用户为后续连接选择文件夹。如果更改了传感器配置，则需要将 Atmel START 项目中的新仪表板配置文件复制到此文件夹。由于配置文件名相同，因此旧文件应替换为新文件。不应修改文件名。

8. 仪表板视图显示三个数据部分。第 1 部分转换所有已配置按钮的状态信息以及增量和阈值。

Button Data				
Channel ID	**Sensor Type**	**State**	**Delta**	**Threshold**
0	**Button 0**	1	65	20
1	**Button 1**	0	2	20

9. 第 2 部分显示使用已配置通道的信号、参考值和增量值绘制的图表视图，如下所示。通过单击图表底部的图例可以使能/禁止这些图表。



10. 第 3 部分显示由抗噪声模块的数据、详细的传感器信息（包括补偿电容值和错误状态数据）构成的表。

Sensor Data

Channel ID	Sensor Type	Signal	Reference	Delta	Compensation
0	Button 0	509	508	1	11495
1	Button 1	515	513	2	7287

Compensation: Represents PTC compensation circuit value which is equivalent to sensor capacitance

"Compenstation" value can be used to check whether sensor is saturated. Refer to User Guide

Frequency Hop Data

CurrentFrequency	1	HopFrequency0	0	HopFrequency1	1	HopFrequency2	2

Displayed frequencies are auto-tuned by QTouch Library based on noise levels

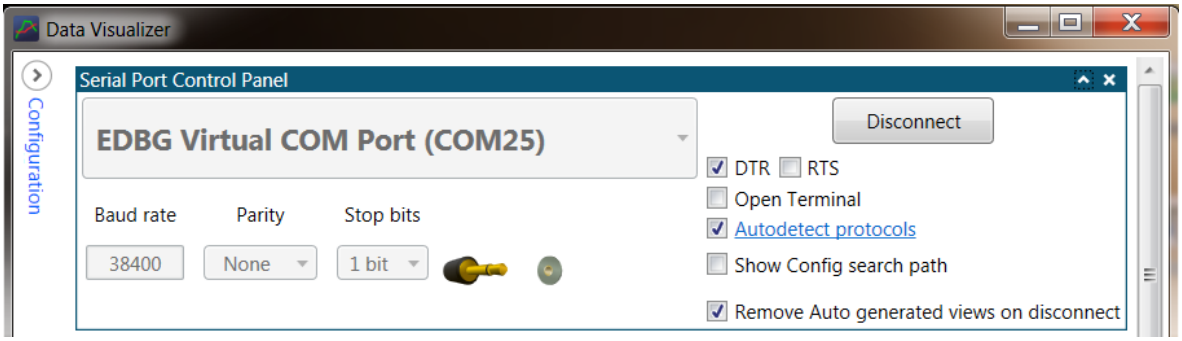
Debug Data

FrameCounter	41
QTouchLibError	0

Counter for datastreamer packets. Missing count indicate packet drop

Indicates library error state. Zero: no error. Refer "Error Code" section in User Guide

11. 要断开硬件连接，请双击 **Serial Port Control Panel** 选项卡将其打开，然后单击“**Disconnect**”（断开连接）按钮，如下所示：



16.4 使用 2D 触摸表面实用程序进行调试

要调试表面和手势项目，需使用称为“2D 触摸表面实用程序”的工具。

有关此工具和表面手势项目的更多详细信息，请访问以下链接：

- http://microchipdeveloper.com/touch:generate-qtouch-surface-gesture-project
- http://microchipdeveloper.com/touch:guide-for-surface-sensor-design-using-modular-library
- http://microchipdeveloper.com/touch:guide-to-connect-to-touch-surface-utility

17. 调节步骤

17.1 调节噪声性能

在任何触摸传感应用中，系统设计人员都必须考虑目标环境中的电气干扰如何影响传感器的性能。

调节不充分的触摸传感器可能会在辐射或传导噪声的测试中出现故障，故障可能在设备的周围环境或电源域中发生，也可能在正常操作期间由设备本身产生。

在许多应用中，必须满足相应的质量标准，这些标准中明确定义了 EMC 性能标准。但是，不能将满足标准视为系统永远不会出现 EMC 问题的证据，因为标准中仅包括最常见的噪声类型和来源。

抗噪声是以增加触摸响应时间和功耗为代价的。系统设计人员必须对触摸传感器进行适当调节，以确保最低功耗。QTouch 模块化库具有许多用户可配置功能，这些功能支持调节，从而在触摸响应时间、抗噪性和功耗之间取得最佳平衡。

噪声源

影响触摸传感器性能的噪声源可能在各种应用中发生

- 电机
- 压电蜂鸣器
- PWM 控制辐射
- 日光灯
- 无线电传输
- 感应炉灶面
- 电源/充电器
- 市电电源

适用的 EMC 标准

- 传导抗扰度 EN61000-4-6

抗噪措施

噪声的影响高度依赖于感应或注入到传感器的噪声信号的幅值，以及该噪声信号的频率曲线。

通常，这种噪声可归类为：

- 宽带噪声

或

- 窄带噪声

宽带抗噪措施

宽带噪声是指频率分量与电容测量频率不呈谐波相关的干扰信号。假设采集信号连同噪声信号的最大和最小电压处于测量系统的输入范围内，并且获取了足够多的采样，则可以通过设置过采样上限值来对宽带噪声干扰执行平均处理。

如果噪声幅值过大，则传感器电路元件会经历测量饱和。在这种情况下，采集信号连同噪声信号将超出测量电路的输入范围，从而导致测量削波。

通常，在经过解析的测量中不会观察到削波，因为削波仅在一部分测量样本上发生，但是削波采样的存在会阻止采样点的有效平均处理。

在这种情况下，即使采用较大的过采样率，采样的平均处理也不会产生无噪声测量。由于信号削波的不对称性，解析的信号将偏离正确的水平。

17.1.1 第 1 步：避免削波

这需要实现硬件低通滤波器，以便减少采集信号中的噪声比例。传感器电容与 Y（传感）线路上的串联电阻相结合，此电阻可以在单片机的内部或外部。

注： 内部串联电阻仅适用于带 PTC 的互电容模式。

外部串联电阻应安装在器件的 Y 线路与传感器之间最靠近器件引脚的位置。

17.1.2 第 2 步：电荷转移测试

通过添加串联电阻来形成低通滤波器的影响是：传感器充电的时间常数增大。在每次测量采样期间，必须确保传感器电容完全充电和放电。

可以观察到充电不足会导致触摸增量减小或补偿电路校准减少。

但是，这个问题在触摸传感器操作中可能不明显；即使在存在低水平噪声的情况下，该应用也可能表现良好，但与不包括电阻器的配置相比，在添加电阻的噪声测试期间性能要差很多。

电荷转移校准

QTouch®模块化库提供自动调整时序参数的功能，以确保完整的电荷转移。

校准可以配置为根据目标器件和测量技术调节三个参数中的一个。

CAL_AUTO_TUNE_RSEL 时钟预分频器和 CSD 保持所配置的设置不变，而内部串联电阻调整到最大值，以便每个传感器节点充分充电。

- 仅适用于 PTC 互电容采集。

CAL_AUTO_TUNE_PRSC 串联电阻和 CSD 保持所配置的设置不变，而预分频器调整到最小值，以便每个传感器节点充分充电。

- 递增会使采集时间加倍，递减会使采集时间减半

CAL_AUTO_TUNE_CSD 预分频器和电阻保持所配置的设置不变，而电荷共用延时调整到最小值，以便每个传感器节点充分充电。

- 递增 CSD 会将一个周期添加到采集序列的电荷转移阶段

17.1.3 第 3 步：调整过采样

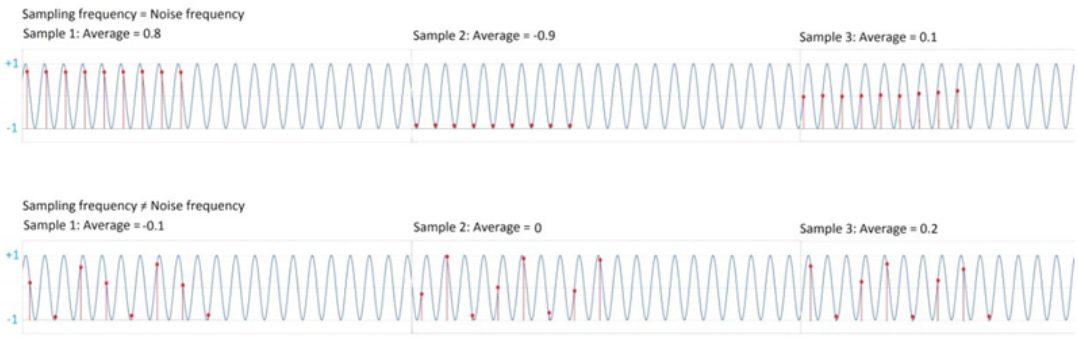
通过硬件滤波避免削波并确保完整的电荷转移后，下一步是找到过采样的最佳设置。

这是抗噪性与响应时间和功耗之间的一种权衡。采样越多，数据质量越高，但采集时间也越长。

窄带抗噪措施

如果噪声包含与电容测量频率相关的频率分量，则过采样不会对噪声效应进行平均处理。以相同采样频率进行的任何一批测量采样都将导致测量偏移。每次测量产生的实际偏移取决于噪声分量的相对相位以及采样频率。

下图说明了这种影响，其中噪声用正弦波表示。



17.1.4 第 4 步：选择频率模式

在噪声处于（或接近）与采样频率谐波相关的频率时，噪声问题会变得严重，如上文所述。在这种情况下，必须调整过采样频率以避免噪声。

这一点在需要进行频率扫描测试的应用中尤其重要，例如 EN61000-4-6。

采集模块（PTC）

可用频率（4 MHz PTC 时钟）	
频率选择	频率（kHz）
FREQ_SEL_0	66.67
FREQ_SEL_1	62.5
FREQ_SEL_2	58.82
FREQ_SEL_3	55.56
FREQ_SEL_4	52.63
FREQ_SEL_5	50
FREQ_SEL_6	47.62
FREQ_SEL_7	45.45
FREQ_SEL_8	43.48
FREQ_SEL_9	41.67
FREQ_SEL_10	40
FREQ_SEL_11	38.46
FREQ_SEL_12	37.04
FREQ_SEL_13	35.71
FREQ_SEL_14	34.48
FREQ_SEL_15	33.33
FREQ_SEL_SPREAD	可变频率

采集模块提供两种频率选择策略：

1. 选择单个采集频率，并且仅在该频率下进行过采样。FREQ_SEL_0 提供最快的测量速度，FREQ_SEL_15 提供最慢的测量速度。如果不需要高性能 EMC 标准，但应用设备产生干扰特定采集频率的噪声，则设计人员只需更改频率。
2. 在过采样期间使用可变频率。FREQ_SEL_SPREAD 在采集过采样期间改变频率。在过采样期间的连续采样中，延时以锯齿方式从 0 变化到 15，以应用更宽的采样频率范围。与单频采集相比，频率扩展选项降低了在特定“最差情况”频率下对噪声的敏感度，但增加了最差情况频率周围的噪声频率范围，这将表现为谐波干扰——尽管噪声影响的严重程度有所降低。在许多应用中，FREQ_SEL_SPREAD 足以达到所需的抗噪性。

跳频模块
模块 ID：0x0006

跳频模块利用三个或更多基频和中值滤波器来避免使用受谐波干扰影响的测量值。选择的频率应当能够最大程度减少问题频带内的一系列交叉谐波。

选定的每个频率都用于连续测量周期期间的采集过采样。

Example

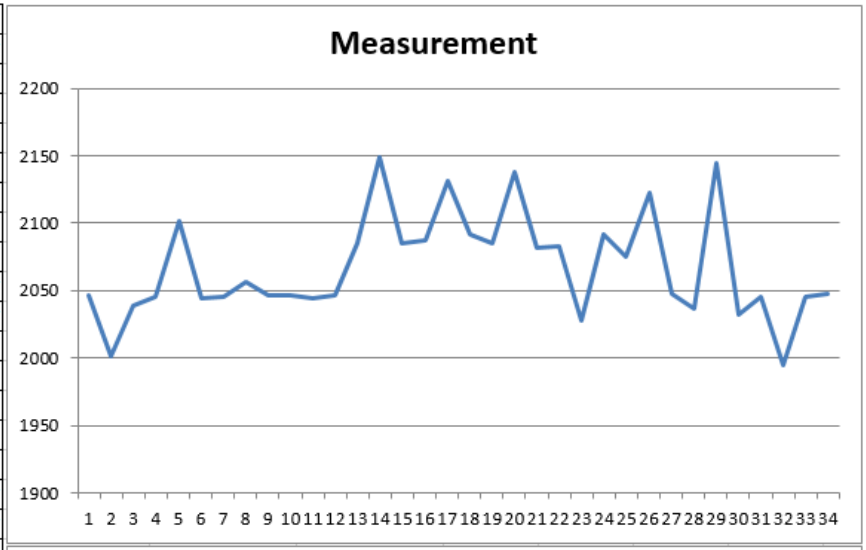
- **周期 1**：所有传感器以频率 0 测量

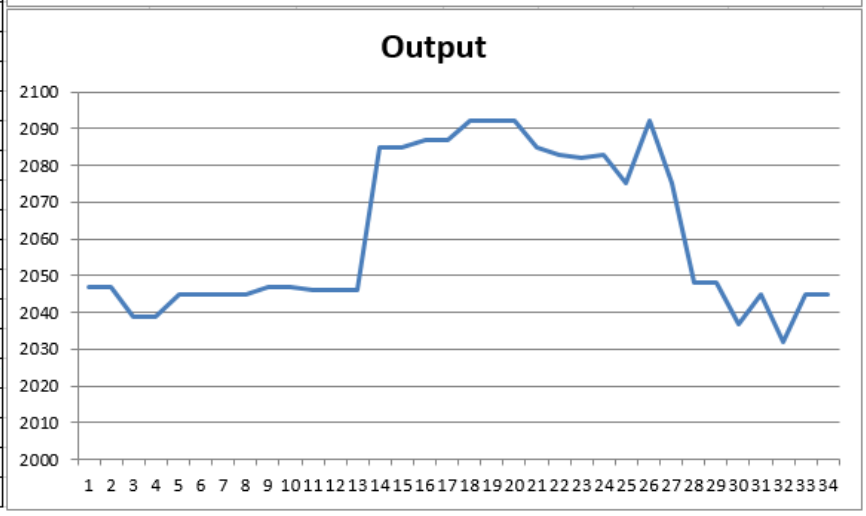
- **周期 2：**所有传感器以频率 1 测量
- **周期 3：**所有传感器以频率 2 测量
- **周期 4：**所有传感器以频率 0 测量
- **周期 5：**所有传感器以频率 1 测量

如果频率 0 与噪声频率相关，则使用 F0 获得的测量值将呈现出较大变化。使用中值滤波器时，将拒绝无关的测量值。

图 17-1. 在频率 1 下获得的测量值受噪声影响

Cycle	Frequency	Measurement	Output
0	F0	2047	2047
1	F1	2001	2047
2	F2	2039	2039
3	F0	2045	2039
4	F1	2102	2045
5	F2	2044	2045
6	F0	2045	2045
7	F1	2056	2045
8	F2	2047	2047
9	F0	2046	2047
10	F1	2044	2046
11	F2	2046	2046
12	F0	2085	2046
13	F1	2149	2085
14	F2	2085	2085
15	F0	2087	2087
16	F1	2131	2087
17	F2	2092	2092
18	F0	2085	2092
19	F1	2138	2092
20	F2	2082	2085
21	F0	2083	2083
22	F1	2028	2082
23	F2	2092	2083
24	F0	2075	2075
25	F1	2122	2092
26	F2	2048	2075
27	F0	2037	2048
28	F1	2145	2048
29	F2	2032	2037
30	F0	2045	2045
31	F1	1995	2032
32	F2	2045	2045
33	F0	2048	2045





常见谐波

无论选择哪个频率，都可能存在更高频率的噪声，这些频率是选定的多个频率的谐波。

继续研究频谱可知，有一些频率是所有可用频率的谐波，但这些超集谐波的频率较高，因此被低通滤波器阻断。

在某些应用中，暴露于噪声频率的可能性可能是一个未知的变化量。

例如，使用 USB 充电器的设备可能无法始终插入随附的充电器中。廉价的替换充电器更容易产生高水平的共模噪声，并且频率可变——通常与采集频率处于同一频带。

根据传导抗扰度 EN61000-4-6 测试的应用会出现类似现象。测试设备以 1%为步长扫描 150 kHz 至 80 MHz 范围内的注入噪声。这提供了一个绝佳的机会来命中干扰频率，而干扰频率是跳频的常见谐波。

在这两种情况下，静态选择频率无法确保通过中值滤波器避免谐波。

支持自动调节的跳频
模块 ID：0x0004

支持自动调节的跳频通过中值滤波器功能提供循环跳频，经过扩展后，可以量化在各个频率下测量的信号变化。

模块配置了稳定限值，当在特定过采样频率下测量的信号呈现出超出此限值的重复变化时，模块会将此频率切换为其他频率，以便搜寻性能更好的选项。

17.2 调节滑动条/滚轮传感器

例如，配置了两个按钮和一个三通道自电容滑动条。令形成滑动条传感器的按钮 B0、B1 和按键传感器为 Slider0[0]、Slider0[1]和 Slider0[2]。

按钮、滑动条/滚轮数据按如下所示的方式显示在 Data Visualizer 控制面板视图中。

Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Button 0	0	0	20
1	Button 1	0	0	20
2	Slider 0[0]	0	1	70
3	Slider 0[1]	0	1	70
4	Slider 0[2]	0	0	70

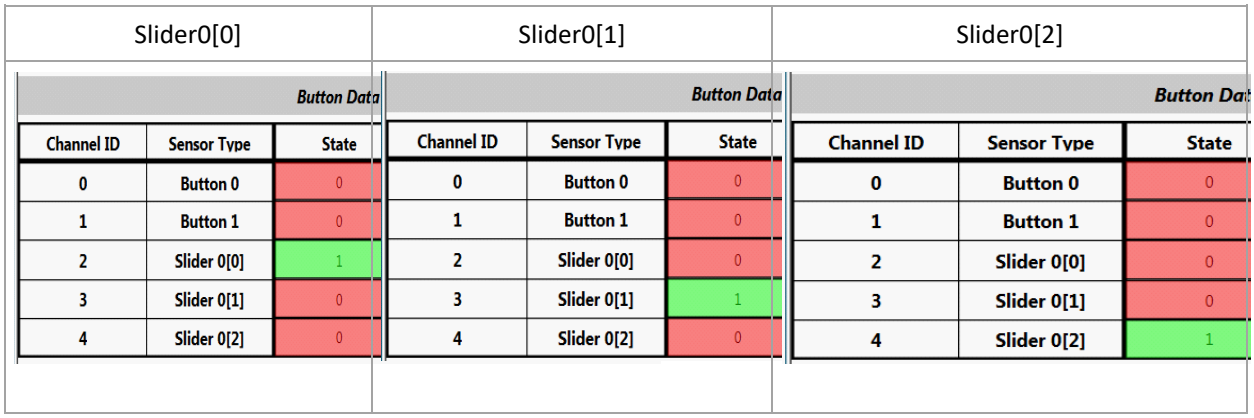
Slider & Wheel Data

Sensor	State	SW Delta	SW Threshold	Position
Slider 0	0	0	60	0

以下步骤描述了调节滑动条/滚轮传感器的过程。

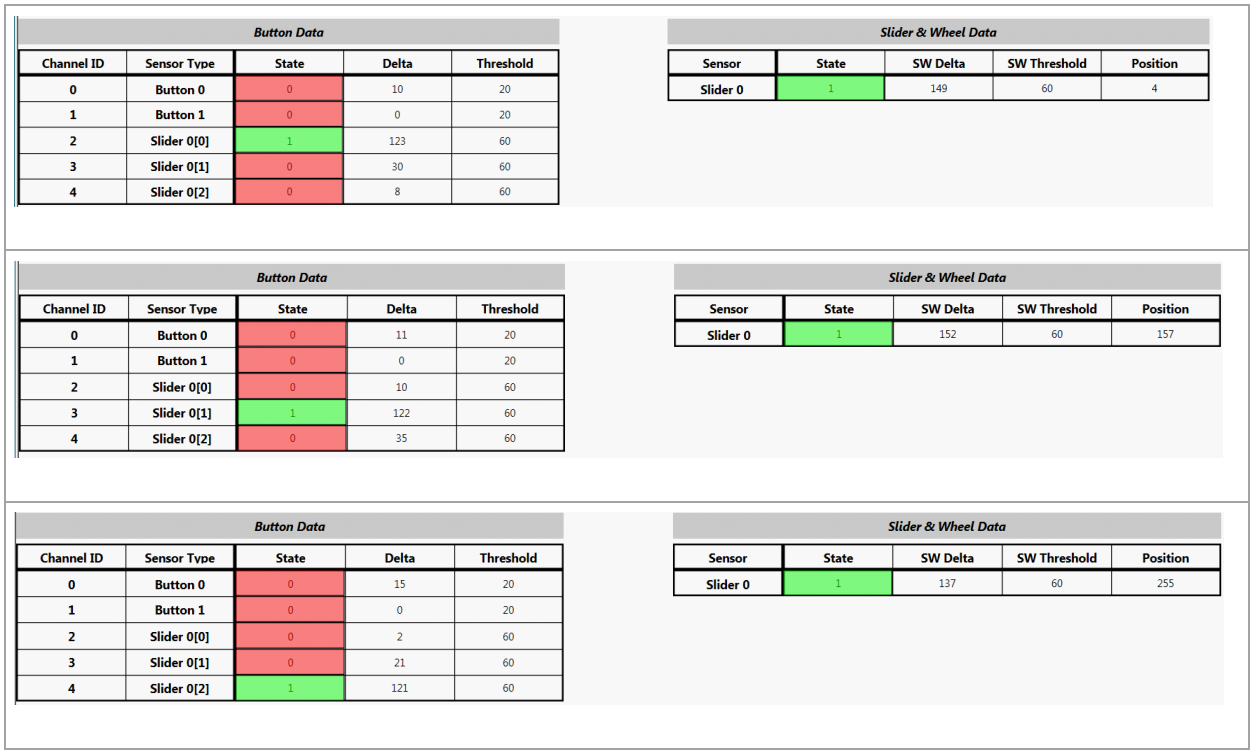
第 1 步：
触摸每个按键传感器 Slider0[0]、Slider0[1]和 Slider0[2]并注意增量值。例如，在 slider0[0]上观察到的增量如下所示。

将单个按键传感器阈值设置为观察到的增量值的一半。在这种情况下，按键阈值应设置为 60。



Slider0[0]			Slider0[1]			Slider0[2]		
Button Data			Button Data			Button Data		
Channel ID	Sensor Type	State	Channel ID	Sensor Type	State	Channel ID	Sensor Type	State
0	Button 0	0	0	Button 0	0	0	Button 0	0
1	Button 1	0	1	Button 1	0	1	Button 1	0
2	Slider 0[0]	1	2	Slider 0[0]	0	2	Slider 0[0]	0
3	Slider 0[1]	0	3	Slider 0[1]	1	3	Slider 0[1]	0
4	Slider 0[2]	0	4	Slider 0[2]	0	4	Slider 0[2]	1

第 2 步：
将第 1 步中计算得到的按键阈值设置为 SW 阈值，并验证当在各个传感器上进行触摸时，滑动条传感器是否按如下所示变为检测状态。



Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Button 0	0	10	20
1	Button 1	0	0	20
2	Slider 0[0]	1	123	60
3	Slider 0[1]	0	30	60
4	Slider 0[2]	0	8	60

Slider & Wheel Data

Sensor	State	SW Delta	SW Threshold	Position
Slider 0	1	149	60	4

Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Button 0	0	11	20
1	Button 1	0	0	20
2	Slider 0[0]	0	10	60
3	Slider 0[1]	1	122	60
4	Slider 0[2]	0	35	60

Slider & Wheel Data

Sensor	State	SW Delta	SW Threshold	Position
Slider 0	1	152	60	157

Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Button 0	0	15	20
1	Button 1	0	0	20
2	Slider 0[0]	0	2	60
3	Slider 0[1]	0	21	60
4	Slider 0[2]	1	121	60

Slider & Wheel Data

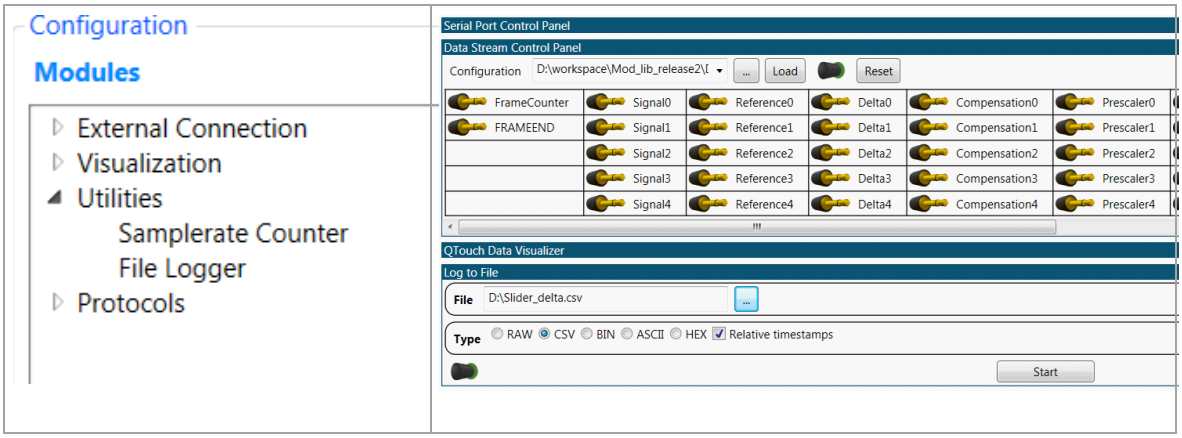
Sensor	State	SW Delta	SW Threshold	Position
Slider 0	1	137	60	255

第 3 步：
在滑动条传感器上的开始和结束拐角之间来回滚动，并使用文件记录器实用程序将 SW 增量记录到 CSV 文件上。

要使用文件记录器，请按照以下说明操作：

1. 通过选中调试控制面板底部的编辑框切换到编辑模式
2. 通过 *Configurations -> Utilities -> File Logger*（配置 -> 实用程序 -> 文件记录器），打开 **File Logger**（文件记录器）组件

3. 双击标题栏最小化 **QTouch Data Visualizer** 视图窗口
4. 在 **File Logger** 视图中，单击文件浏览器并选择文件名以记录数据。
5. 单击 **SWDelta0** 连接器，将其拖动并插入到文件记录器插座，如下图所示。
6. 现在单击 **Start/Stop**（启动/停止）按钮来记录数据。
7. 记录完成后，删除 SWDelta0 连接，取消选中编辑模式复选框并关闭文件记录器。



第 4 步：
打开文件日志并从采样中识别出最小的 SWDelta0 值。下面列出了从 Slider_delta.csv 文件中收集的一些 SWDelta0 采样。

103,102,101,106,98,92,86,82,78,76,78,86,0,118,113,109,105,102,100,106,102,93,86,80,76,73,72,73,79,88,90,90,89,89,89,90,94,87,81,76,72,71,69,72,81,89,94,98,100,102

将 SW 阈值设置为小于识别的最小值“69”。在这种情况下，SW 阈值设置为比观察值小约 15 个计数，即 54。重复步骤 3 和步骤 4 进行数次迭代，并根据日志调节 SW 阈值。

调节后的按钮和滑动条/滚轮数据如下所示。

Button Data

Channel ID	Sensor Type	State	Delta	Threshold
0	Button 0	0	3	20
1	Button 1	0	0	20
2	Slider 0[0]	0	2	60
3	Slider 0[1]	0	49	60
4	Slider 0[2]	0	34	60

Slider & Wheel Data

Sensor	State	SW Delta	SW Threshold	Position
Slider 0	1	80	54	192

18. 已知问题

序号	问题描述	分类	解决方案
1	当 PTC 用于自电容模式时，内部串联电阻在降低噪声方面无效。	PTC	建议使用外部串联电阻
2	SAMD2x/SAMDA1/SAMHA1 器件中的 Y0 和 Y2 引脚具有更高的寄生电容。SAMC21/C20/CA1 中的 Y0、Y26 和 Y27 引脚	PTC	1. 如果有备用 Y 线路，请避免使用这些引脚。 2. 如果需要将此引脚用于触摸，则需要提供更长的充电时间，这可能影响响应时间。

19. 附录 A——版本历史

文档版本	日期	备注
A	2016 年 11 月	文档初始版本。
B	2017 年 2 月	增加了 SAMD10/D11 库信息
C	2017 年 6 月	**修订部分：** 触摸库介绍、Data Visualizer 和 Atmel START 配置器 **新增内容：** 采集模块、触摸按键模块、跳频模块、跳频自动调节模块、滑动条/滚轮模块、绑定层模块、MISRA 报告、每个模块的 API 引用、工具包示例项目、模块命名约定和 API 文件接口
A	2017 年 12 月	Microchip DS40001986 修订版 A 取代了 Atmel 42805A。 **修订部分：** 使用最新的 QTouch 配置器 GUI 更新了 KIT 示例/用户板项目创建部分中的屏幕截图 包含了触摸按键传感器的图片，在介绍部分包含了多张图 **新增内容：** 新增“已知问题”部分
B	2018 年 5 月	**新增内容：** 增加了 2D 触摸表面和手势模块支持
C	2018 年 6 月	**新增内容：** 用新器件更新了“模块命名约定”和“附录 B”部分。

20. 附录 B——采集模块 API 引用

```
------------------------------------------------------------------------------
touch_ret_t qtm_acquisition_process(void)
------------------------------------------------------------------------------
Purpose: Signal capture and processing
Input  : (Measured signals, config)
Output : touch_ret_t
Notes  : Called by application after 'touch_measure_complete_callback'

------------------------------------------------------------------------------
touch_ret_t qtm_ptc_init_acquisition_module(qtm_acquisition_control_t* qtm_acq_control_ptr);
------------------------------------------------------------------------------
Purpose: Initialize the PTC & Assign pins
Input  : pointer to acquisition set
Output : touch_ret_t: TOUCH_SUCCESS or INVALID_PARAM
Notes  : ptc_init_acquisition module must be called ONLY once with a pointer to each config
set

------------------------------------------------------------------------------
touch_ret_t qtm_ptc_qtlib_assign_signal_memory(uint16_t* qtm_signal_raw_data_ptr);
------------------------------------------------------------------------------
Purpose: Assign raw signals pointer to array defined in application code
Input  : pointer to raw data array
Output : touch_ret_t: TOUCH_SUCCESS
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_enable_sensor_node(qtm_acquisition_control_t* qtm_acq_control_ptr, uint16_t
qtm_which_node_number);
------------------------------------------------------------------------------
Purpose:  Enables a sensor node for measurement
Input  :  Node configurations pointer, node (channel) number
Output : touch_ret_t:
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_calibrate_sensor_node(ptc_seq_acq_settings* qtm_acq_control_l_ptr, uint16_t
which_node_number)
------------------------------------------------------------------------------
Purpose:  Marks a sensor node for calibration
Input  :  Node configurations pointer, node (channel) number
Output : touch_ret_t:
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_ptc_start_measurement_seq(qtm_acquisition_control_t* qtm_acq_control_pointer,
void (*measure_complete_callback) (void));
------------------------------------------------------------------------------
Purpose:  Loads touch configurations for first channel and start,
Input  :  Node configurations pointer, measure complete callback pointer
Output : touch_ret_t:
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_autoscan_sensor_node(qtm_auto_scan_config_t* qtm_auto_scan_config_ptr, void
(*auto_scan_callback)(void));
------------------------------------------------------------------------------
Purpose: Configures the PTC for sleep mode measurement of a single node, with window
comparator wake
Input  : Acquisition set, channel number, threshold, scan trigger
Output : touch_ret_t
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_autoscan_node_cancel(void)
------------------------------------------------------------------------------
Purpose: Cancel auto-scan config
Input  : None
Output : touch_ret_t
Notes  : none

------------------------------------------------------------------------------
```

```
void qtm_ptc_de_init(void)
-------------------------------------------------------------------------------
Purpose: Clear PTC Pin registers, set TOUCH_STATE_NULL
Input  : none
Output : none
Notes  : This API function is used to RESET the PTC during runtime without power cycle the
hardware.The application may include this function as part of other soft reset functions to
restart the application at runtime.

-------------------------------------------------------------------------------
uint16_t qtm_<device_family>_acq_module_get_id(void)
Applicable <device_family> =
m328pb,m324pb,t81x,t161x,samd1x,samd20,samd21,samda1,same51,same53,same54,samd51,tiny321x,samc
20,samc21,saml21,saml22,samha1,saml10,saml11
-------------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none

-------------------------------------------------------------------------------
uint8_t qtm_<device_family>_acq_module_get_version(void);
Applicable <device_family> =
m328pb,m324pb,t81x,t161x,samd1x,samd20,samd21,samda1,same51,same53,same54,samd51,tiny321x,samc
20,samc21,saml21,saml22,samha1,saml10,saml11
-------------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none

-------------------------------------------------------------------------------
void qtm_ptc_clear_interrupt(void) -> ARM Cortex SAMD10,SAMD11,SAME51/E53/E54/D51
-------------------------------------------------------------------------------
Purpose :  Clears the eoc/wcomp interrupt bits
Input   :  none
Output  :  none
Notes   :  none

-------------------------------------------------------------------------------
void qtm_<device_family>_ptc_handler_eoc(void)
Applicable <device_family> =
m328pb,m324pb,t81x,t161x,samd1x,samd20,samd21,samda1,same51,same53,same54,samd51,tiny321x,samc
20,samc21,saml21,saml22,samha1,saml10,saml11
-------------------------------------------------------------------------------
Purpose :  Captures  the  measurement,  starts  the  next  or  End  Of  Sequence  handler
Input   :  none
Output  :  none
Notes   :  none

-------------------------------------------------------------------------------
void qtm_<device_family>_ptc_handler_wcomp(void)
Applicable <device_family> =
m328pb,m324pb,t81x,t161x,samd1x,samd20,samd21,samda1,same51,same53,same54,samd51,tiny321x,samc
20,samc21,saml21,saml22,samha1,saml10,saml11
-------------------------------------------------------------------------------
Purpose :  Captures  the  measurement,  calls the callback
Input   :  none
Output  :  none
Notes   :  none
```

21. 附录 C——跳频模块 API 引用

```
-------------------------------------------------------------------------------
touch_ret_t qtm_freq_hop(qtm_freq_hop_control_t *qtm_freq_hop_control);
-------------------------------------------------------------------------------
Purpose: Runs freq hop process
Input  : Pointer to container structure
Output : touch_ret_t
Notes  : none

-------------------------------------------------------------------------------
uint16_t qtm_get_freq_hop_module_id(void)
-------------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none

-------------------------------------------------------------------------------
uint8_t qtm_get_freq_hop_module_ver(void)
-------------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none

-------------------------------------------------------------------------------
```

22. 附录 D——跳频自动调节模块 API 引用

```
------------------------------------------------------------------------------
touch_ret_t qtm_freq_hop_autotune(qtm_freq_hop_autotune_control_t
*qtm_freq_hop_autotune_control);
------------------------------------------------------------------------------
Purpose: Runs freq hop auto tune process
Input  : Pointer to container structure
Output : touch_ret_t
Notes  : none

------------------------------------------------------------------------------
uint16_t qtm_get_freq_auto_module_id(void);
------------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none

------------------------------------------------------------------------------
uint8_t qtm_get_freq_auto_module_ver(void);
------------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none

------------------------------------------------------------------------------
```

23. 附录 E——触摸按键模块 API 引用

```
------------------------------------------------------------------------------
touch_ret_t qtm_init_sensor_key(qtm_touch_key_control_t* qtm_lib_key_group_ptr, uint8_t
which_sensor_key, qtm_acq_node_data_t* acq_lib_node_ptr)
------------------------------------------------------------------------------
Purpose: Initialize a touch key sensor
Input  : Pointer to key group control data, key number, pointers to sensor node status and
signal
Output : TOUCH_SUCCESS
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_key_sensors_process(qtm_touch_key_control_t* qtm_lib_key_group_ptr)
------------------------------------------------------------------------------
Purpose: Sensor key post-processing (touch detect state machine)
Input  : Pointer to key group control data
Output : TOUCH_SUCCESS
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_key_suspend(uint16_t which_sensor_key, qtm_touch_key_control_t*
qtm_lib_key_group_ptr)
------------------------------------------------------------------------------
Purpose: Suspends acquisition measurements for the key
Input  : Key number, Pointer to key group control data
Output : TOUCH_SUCCESS
Notes  : Used to suspend the key temporarily, like to save the power by avoiding the
continuous scan on all the sensors.A single key can be defined to act as wake up sensor and
other key sensors can be suspended using this API.This API function works in association with
resume API function

------------------------------------------------------------------------------
touch_ret_t qtm_key_resume(uint16_t which_sensor_key, qtm_touch_key_control_t*
qtm_lib_key_group_ptr)
------------------------------------------------------------------------------
Purpose: Resumes acquisition measurements for the key
Input  : Key number, Pointer to key group control data
Output : TOUCH_SUCCESS
Notes  : Can be used along with suspend API function to avoid scanning of sensors
temporarily.For instance, some of the keys may be suspended from scanning during the idle
time and resumes based on touch on a defined key

------------------------------------------------------------------------------
void update_qtlib_timer(uint16_t time_elapsed_since_update)
------------------------------------------------------------------------------
Purpose: Updates local variable with time period
Input  : Number of ms since last update
Output : none
Notes  : none

------------------------------------------------------------------------------
uint16_t qtm_get_touch_keys_module_id(void)
------------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none

------------------------------------------------------------------------------
uint8_t qtm_get_touch_keys_module_ver(void)
------------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none

------------------------------------------------------------------------------
```

24. 附录 F——滚动条模块 API 引用

```
------------------------------------------------------------------------------
touch_ret_t qtm_init_scroller_module(qtm_scroller_control_t *qtm_scroller_control)
------------------------------------------------------------------------------
Purpose: Initialize a scroller
Input  : Pointer to scroller group control data
Output : Touch return status value
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_scroller_process(qtm_scroller_control_t *qtm_scroller_control)
------------------------------------------------------------------------------
Purpose: Scroller position calculation and filtering
Input  : Pointer to scroller group control data
Output : Touch return status value
Notes  : none

------------------------------------------------------------------------------
uint16_t qtm_get_scroller_module_id(void)
------------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none

------------------------------------------------------------------------------
uint8_t qtm_get_scroller_module_ver(void)
------------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none
------------------------------------------------------------------------------
```

25. 附录 G——2D 表面（单指触摸）CS 模块

```
/*============================================================================
touch_ret_t qtm_init_surface_cs(qtm_surface_cs_control_t *qtm_surface_cs_control);
----------------------------------------------------------------------------
Purpose: Initialize a scroller
Input  : Pointer to scroller group control data
Output : TOUCH_SUCCESS
Notes  : none
============================================================================*/
touch_ret_t qtm_init_surface_cs(qtm_surface_cs_control_t *qtm_surface_cs_control);

/*============================================================================
touch_ret_t qtm_surface_cs_process(qtm_surface_cs_control_t *qtm_surface_cs_control);
----------------------------------------------------------------------------
Purpose: Scroller position calculation and filtering
Input  : Pointer to scroller group control data
Output : TOUCH_SUCCESS
Notes  : none
============================================================================*/
touch_ret_t qtm_surface_cs_process(qtm_surface_cs_control_t *qtm_surface_cs_control);

/*============================================================================
uint16_t qtm_get_scroller_module_id(void)
----------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none
============================================================================*/
uint16_t qtm_get_surface_cs_module_id(void);

/*============================================================================
uint8_t qtm_get_scroller_module_ver(void)
----------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none
============================================================================*/
uint8_t qtm_get_surface_cs_module_ver(void);
```

26. 附录 H——2D 表面（双指触摸）CS/2T 模块

```
/*============================================================================
touch_ret_t qtm_init_surface_cs2t(qtm_surface_cs_control_t *qtm_surface_cs_control);
----------------------------------------------------------------------------
Purpose: Initialize a scroller
Input  : Pointer to scroller group control data
Output : TOUCH_SUCCESS
Notes  : none
============================================================================*/
touch_ret_t qtm_init_surface_cs2t(qtm_surface_cs2t_control_t *qtm_surface_cs2t_control);

/*============================================================================
touch_ret_t qtm_surface_cs_process(qtm_surface_cs_control_t *qtm_surface_cs_control);
----------------------------------------------------------------------------
Purpose: Scroller position calculation and filtering
Input  : Pointer to scroller group control data
Output : TOUCH_SUCCESS
Notes  : none
============================================================================*/
touch_ret_t qtm_surface_cs2t_process(qtm_surface_cs2t_control_t *qtm_surface_cs2t_control);

/*============================================================================
uint16_t qtm_get_surface_cs2t_module_id(void)
----------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none
============================================================================*/
uint16_t qtm_get_surface_cs2t_module_id(void);

/*============================================================================
uint8_t qtm_get_surface_cs2t_module_ver(void)
----------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none
============================================================================*/
uint8_t qtm_get_surface_cs2t_module_ver(void);
```

27. 附录 I——手势模块

```
------------------------------------------------------------------------------
void qtm_gestures_2d_clearGesture(void);

/*============================================================================
touch_ret_t qtm_init_gestures_2d(void);
----------------------------------------------------------------------------
Purpose: Initialize gesture tracking variables
Input : -
Output : TOUCH_SUCCESS
Notes : none
============================================================================*/
touch_ret_t qtm_init_gestures_2d(void);

/*============================================================================
touch_ret_t qtm_gestures_2d_process(qtm_gestures_2d_control_t *qtm_gestures_2d_control);
----------------------------------------------------------------------------
Purpose: Gesture engine processes updated touch info
Input : Gesture control struct pointer
Output : ?TOUCH_SUCCESS?
Notes : none
============================================================================*/
touch_ret_t qtm_gestures_2d_process(qtm_gestures_2d_control_t *qtm_gestures_2d_control);

/*============================================================================
void qtm_update_gesture_2d_timer(uint16_t time_elapsed_since_update);
----------------------------------------------------------------------------
Purpose: Updates local variable with time period
Input  : Number of ms since last update
Output : none
Notes  : none
============================================================================*/
void qtm_update_gesture_2d_timer(uint16_t time_elapsed_since_update);

/*============================================================================
uint16_t qtm_get_gesture_2d_module_id(void);
----------------------------------------------------------------------------
Purpose: Returns the module ID
Input : none
Output : Module ID
Notes : none
============================================================================*/
uint16_t qtm_get_gesture_2d_module_id(void);

/*============================================================================
uint8_t qtm_get_gesture_2d_module_ver(void);
----------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes : none
============================================================================*/
uint8_t qtm_get_gesture_2d_module_ver(void);
------------------------------------------------------------------------------
```

28. 附录 J——绑定层模块 API 引用

```
------------------------------------------------------------------------------
void qtm_binding_layer_init(qtm_control_t *qtm_control);
------------------------------------------------------------------------------
Purpose: This function internally executes the individual module initialization functions
using the pointers.Based on the initialization output, init_complete_callback or the
error_callback function is triggered.
Input  : Pointer to binding layer container structure
Output : none
Notes  : none

------------------------------------------------------------------------------
void qtm_lib_start_acquisition(qtm_control_t *qtm_control);
------------------------------------------------------------------------------
Purpose: This function internally executes the “qtm_ptc_start_measurement_seq” function to
start the measurement of sensors.The functions of multiple acquisition groups are executed
sequentially.
Input  : Pointer to binding layer container structure
Output : none
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_lib_acq_process(void)
------------------------------------------------------------------------------
Purpose: Executes the acquisition post process functions.The acquisition post process of
multiple groups is executed sequentially according to the configuration.
Input  : none
Output : Touch return status value
Notes  : none

------------------------------------------------------------------------------
touch_ret_t qtm_lib_post_process(void)
------------------------------------------------------------------------------
Purpose: Executes the individual module post processes.The sequence of post processes
executed is based on the configuration of qtm_config_t
Input  : none
Output : Touch return status value
Notes  : none

------------------------------------------------------------------------------
qtm_control_t* qmt_get_binding_layer_ptr(void)
------------------------------------------------------------------------------
Purpose: Returns the pointer to the binding layer container structure
Input  : none
Output : pointer to the binding layer container
Notes  : none

------------------------------------------------------------------------------
uint16_t qtm_get_lib_state(void)
------------------------------------------------------------------------------
Purpose: Returns the binding layer state
Input  : none
Output : Module state
Notes  : none

------------------------------------------------------------------------------
uint16_t qtm_get_binding_layer_module_id(void)
------------------------------------------------------------------------------
Purpose: Returns the module ID
Input  : none
Output : Module ID
Notes  : none

------------------------------------------------------------------------------
uint8_t qtm_get_binding_layer_module_ver(void)
------------------------------------------------------------------------------
Purpose: Returns the module Firmware version
Input  : none
Output : Module ID - Upper nibble major / Lower nibble minor
Notes  : none
```

29. 附录 K——器件支持

有关最新器件支持列表的信息，请访问：http://microchipdeveloper.com/touch:release-notes。

Microchip 网站

Microchip 网站 http://www.microchip.com/为客户提供在线支持。客户可通过该网站方便地获取文件和信息。只要使用常用的互联网浏览器即可访问，网站提供以下信息：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题（FAQ）、技术支持请求、在线讨论组以及 Microchip 顾问计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

变更通知客户服务

Microchip 的变更通知客户服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请登录 Microchip 网站 http://www.microchip.com/。在“支持”（Support）下，点击“变更通知客户”（Customer Change Notification）服务后按照注册说明完成注册。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（FAE）
- 技术支持

客户应联系其代理商、代表或应用工程师（FAE）寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过以下网站获得技术支持： http://www.microchip.com/support

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

商标

ISBN：978-1-5224-5294-2

质量管理体系

有关 Microchip 质量管理体系的更多信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：1-480-792-7200 传真：1-480-792-7277 技术支持： http://www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，乔治亚州 电话：1-678-957-9614 传真：1-678-957-1455 **奥斯汀，德克萨斯州** 电话：1-512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：1-774-760-0087 传真：1-774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：1-630-285-0071 传真：1-630-285-0075 **达拉斯** 艾迪生，德克萨斯州 电话：1-972-818-7423 传真：1-972-818-2924 **底特律** 诺维，密歇根州 电话：1-248-848-4000 **休斯敦，德克萨斯州** 电话：1-281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：1-317-773-8323 传真：1-317-773-5453 电话：1-317-536-2380 **洛杉矶** 米申维耶霍，加利福尼亚州 电话：1-949-462-9523 传真：1-949-462-9608 电话：1-951-273-7800 **罗利，北卡罗来纳州** 电话：1-919-844-7510 **纽约，纽约州** 电话：1-631-435-6000 **圣何塞，加利福尼亚州** 电话：1-408-735-9110 电话：1-408-436-4270 **加拿大 - 多伦多** 电话：1-905-695-1980 传真：1-905-695-2078	**中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 珠海** 电话：86-756-3210040 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **台湾地区 - 新竹** 电话：886-3-577-8366	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880- 3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4450-2828 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-67-3636 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 赖阿南纳** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-7289-7561 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820