

带接近传感器软件的 触摸感测型MCU的应用

作者: Eduardo Muriel Hernandez
Ulises Corales
RTAC Americas

1 简介

本应用指南描述怎样使用具有适当的设计布局和简单软件模块的触摸感测型MCU。该免费软件兼容所有S08、RS08和V1版本的飞思卡尔系列微控制器。此软件的灵活性及其使用最小化硬件的特性，为触摸感测的应用奠定了坚实的基础。本文档将详细介绍它的硬件安装和软件移植所需要考虑的因素。

这些应用程序可以联同附加评估工具包(KITPROXIMITYEVM)来构成完整的评估系统。

目录

1	简介	1
2	综合描述	2
2.1	系统描述	2
2.2	接近感测应用描述	3
3	接近感测模块	8
3.1	综合描述	8
3.2	接近感测的初始化函数	8
3.3	电极的采样函数	8
3.4	电极的状态更新函数	10
4	硬件设备	14
4.1	RS08、S08和ColdFire平台的移植	14
4.2	配置电极和蜂鸣器的位置	15
4.3	定时器模块	16
5	图形用户介面	17
5.1	安装	17
5.2	电容面板条	18
5.3	阈值检测	18
6	结论	19
6.1	接近感测软件的可重用性	19
6.2	飞思卡尔接近感测解决方案	19

2 综合描述

2.1 系统描述

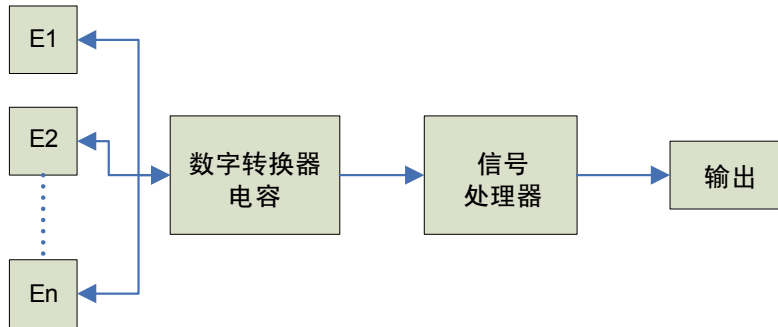


图1 模块框图

该系统的主要组成为：

- 电极阵列 — 这些是用户接口，给系统提供输入。
- 数字转换器电容 — 提供与每个电极的输入电容相称的数字量
- 信号处理器 — 这里是处理度量并判定电极是否被接触的场所。
- 输出 — 有一个蜂鸣器，用户可以通过它获取系统进程的反馈信息。

2.1.1 电极

电极阵列是印刷电路板上安放导电材料的物质区域，它实际上是个电容面板。为了使用这些电容器，每个电极都与上拉电阻器连接从而构成了各自的简单RC电路。

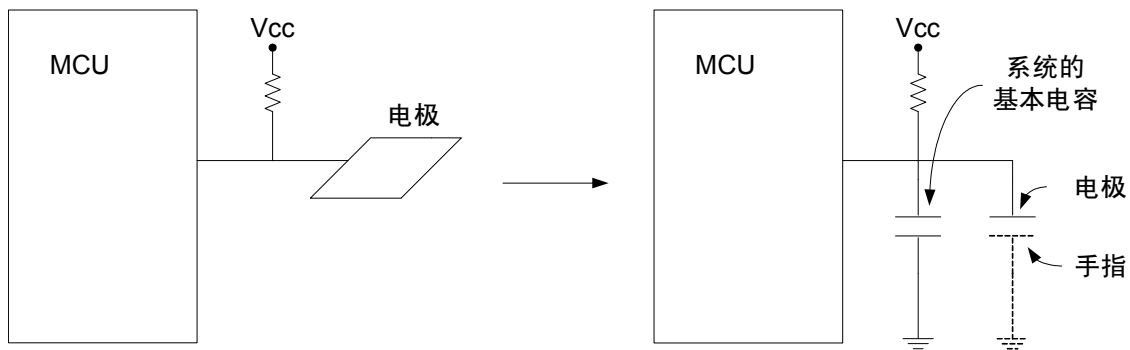


图2 电极等效电路

2.1.2 数字转换器的电容

RC电路的功能是把电容转变为时间。该转变决定了电路的时间常量(等式1)。

等式1

$$\tau = RC$$

由于R是常量，电容变化将线性的改变使电容器达到参考电压所花费的时间。该时间可以由定时器来测量，但这种方法只在参考电压已经达到时执行。所有微控制器的输入引脚都集成了用以判定其输入逻辑电平的比较器，因此，引脚的阈值可被用来作为参考电压。

该数字转换器的电容用来提供当前输入引脚的值。当电极被接触时，电容的数值将会增加。这个不需要考虑进行准确的电容测量，因为这些值会被综合处理和比较，用以确定有无物体在靠近电极。

2.1.3 信号处理部件

数字转换器的电容所提供的采样值，必须经过信号处理过程以得到正确的解析。这些样本是不准确的测量值，但通过这些值，电容的变化可被很容易的检测到。这正是触摸检测所需要的。在这个过程中，可能会用到各种算法以便进行数值测量和改善对目标的检测。例如，利用滤波器用以减少杂波的影响，利用去抖(debouncing)机制来避免虚假的检测。

2.1.4 系统输出

当数字转换器电容输出值被处理且接触被检测到时，系统必须能够对检测给出相应的反馈信号。在描述过的应用实例中，蜂鸣器就能提供反馈，当触摸被检测到发生时，该反馈发出固定频率的声音。任何其他的输出也都可以被实现。

2.2 接近感测应用描述

图3 显示了在前面章节所描述的系统组件如何在评估软件上实现。

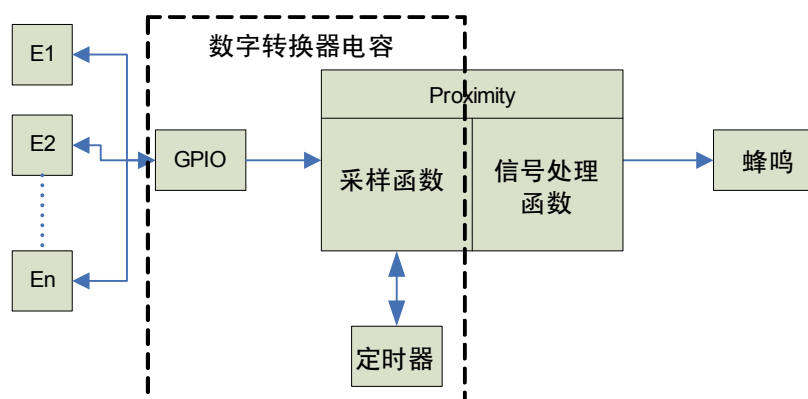


图3 应用模块框图

每个模块都有特定的功能：

- 通用输入输出(GPIO)模块 — 管理与微控制器I/O通路相关的所有事务。此模块提供与电极和系统输出的接口，如蜂鸣器。
- 定时器模块 — 管理硬件定时器模块在应用程序中使用的所有功能。例如，定时器时钟和分频配置以及计数开始、停止和复位等。定时器的ISR也在此模块中设定。
- 接近感测模块 — 它是应用软件的核心，是GPIO模块和定时器模块获得电容测量值的接口。此模块处理电容值，并评估电极是否被接触。
- 蜂鸣器模块 — 当触摸被检测到时，提供控制蜂鸣器产生声音的功能。

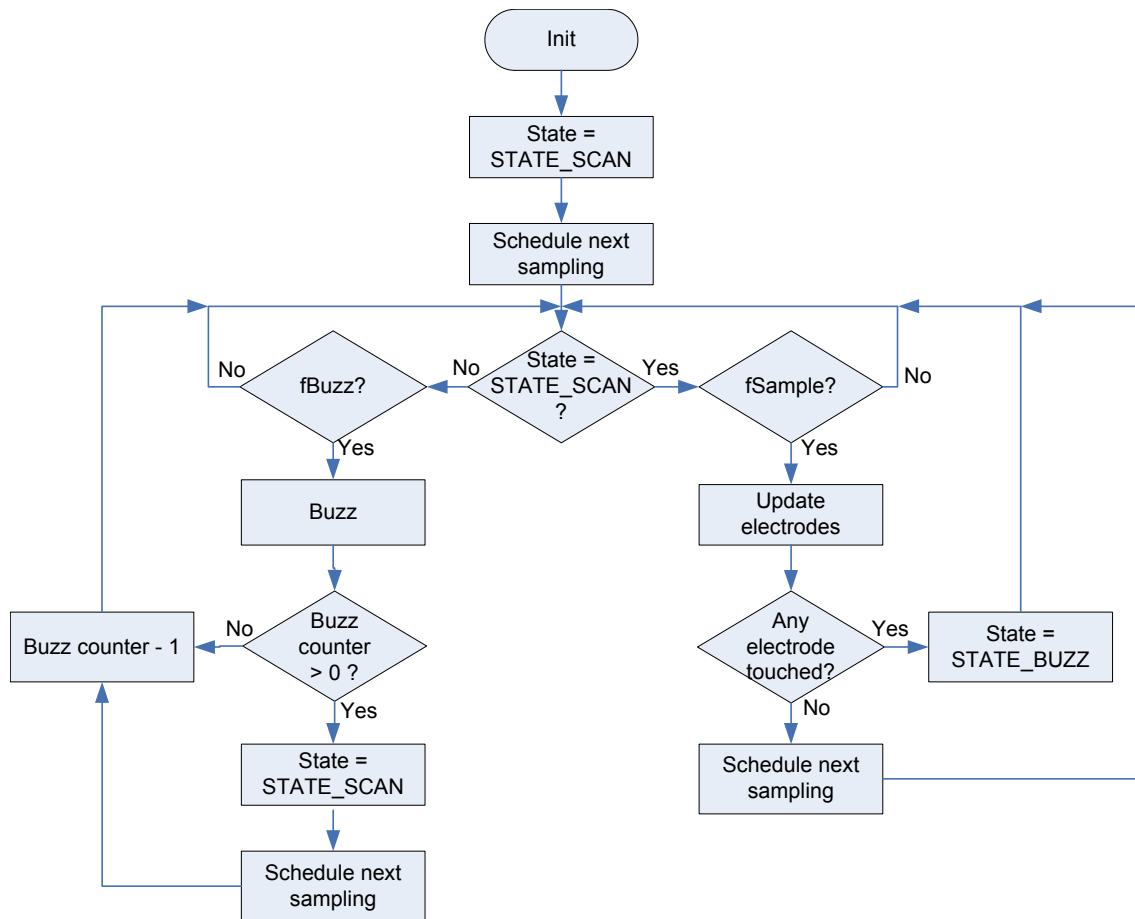


图4 流程图

该接近感测应用软件首先初始化所有相关的模块，然后安排进行第一个采样任务并进入扫描状态，这样一个或多个电极都会被周期性的取样。在采样周期过去后，负责采集和处理的函数将被调用，并更新所选电极的状态(接触/非接触)。然后应用程序会去评估是否有电极已被触动，如果有，则改变蜂鸣器的状态。如果没有电极被接触，则下次采样将会如期进行。在蜂鸣状态下，声音通过蜂鸣器发出，持续10ms。在当前周期完成后，下次采样将被安排，而应用程序又回归到扫描状态。

2.2.1 接近感测软件结构

对于这种应用，图5中所显示的软件体系结构模型业已实现。

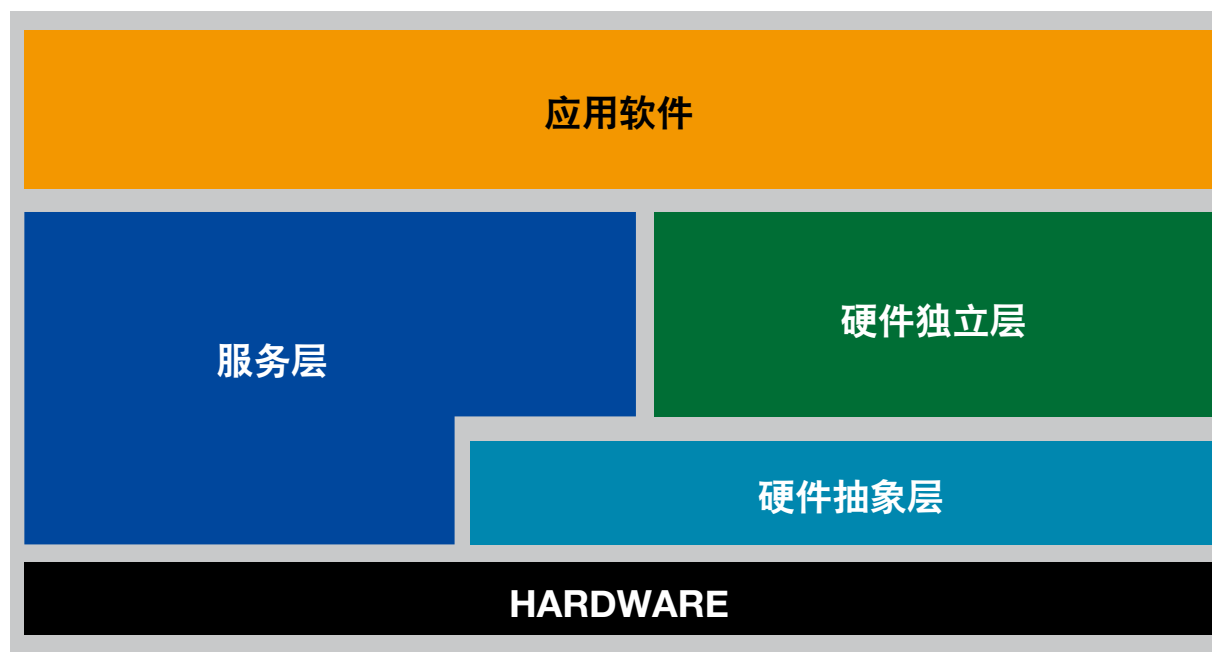


图5 软件体系结构模型

该模型由以下不同层次构成：

- 硬件层 (HL) — 硬件层定义从上层到硬件资源的接口，如外设、配置寄存器或任何其他依赖硬件的资源。没有功能性的特点被界定到这一层。该层在CodeWarrior所提供的头文件中实现，例如MC9S08QE128.h。
- 硬件抽象层 (HAL) — 硬件抽象层可被定义为直接访问硬件资源的软件组件的集合。外设驱动程序是在这一层上实现的。其中的例子如，管理外设配置的软件模块以及外设事件的处理。硬件访问是通过HL接口来实现的。
- 硬件独立层 (HIL) — 硬件独立层组是，所有的不管理微控制器的资源，但可以利用他们来执行某些具体的任务或功能的软件模块。下面这些是属于这个层组：
 - 不需要特定硬件的算法。例如，过滤器、数学函数、排序和查找算法。
 - 需要使用硬件资源以及通过HAL实现的硬件接口的外部资源驱动。例如，通过I²C或SPI通信的DAC模块。
 - 不属于当前微控制器，但软件利用现有的硬件资源构造出的外设。举例来说，利用GPIO通路和定时器模拟实现的串行端口，就是这种类型。

因为HIL软件模块不包含任何具体的硬件信息，所以倘若需要，它们可以很容易的移植到其他平台上。有的时候移植需要一定的HAL接口。

- 服务层 (SL) — 服务层是指，提供其他层需要的基本服务模块的软件组件的集合。这包括时间管理、任务调度、内存管理、系统管理和电源管理等。例如看门狗、系统时钟配置和低功耗模式。
- 应用层 — 所有特定应用软件包含在应用层。所有其他层实现比较抽象的功能，而应用层则整合它们，以创造具体的功能应用。例如，需要管理用户接口的软件组件，被包含在HIL层和HAL层，但是，任务的执行是要根据应用层输入的条件来确定。

依据这个软件体系结构模型，系统的模块必须安置在它们各自的层次上。显然，触摸感测仅仅需要用到的MCU硬件资源是：定时器和GPIO模块。它们直接与硬件相结合，属于HAL层。接近感测和蜂鸣器模块使用HAL的组件。它们给MCU的外部资源提供支持，因此属于HIL层。MCU时钟初始化是SL的组成部分。由此产生的应用的典型模型如图6所示。

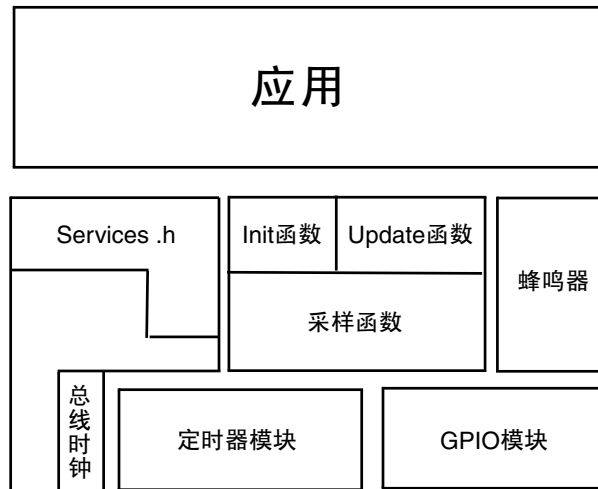


图6 接近感测软件体系结构

上述每个模块都由一个代码文件和一个头文件组成。头文件有两部分，一个命名为公有的，而另一个则命名为私有的。这种区分是为了清晰，因为在该文件里所定义的所有事物其实都是公用的。然而，这些限制必须得到用户的遵循，以改善代码的可携性和可重用性。每个模块的接口定义，包含在各自文件的公共部分，并在文件中

File	Code	Data	
Sources	732	18	
Application	149	0	
main.c	149	0	
Hardware Abstraction Layer	36	1	
Timer.c	9	1	
GPIO.c	27	0	
Hardware Independent Layer	487	17	
Proximity.c	439	17	
Buzzer.c	48	0	
Services	60	0	
Services.c	60	0	
Includes	0	0	
Application	0	0	
main.h	0	0	
Hardware Abstraction Layer	0	0	
derivative.h	0	0	
Timer.h	0	0	
GPIO.h	0	0	
Hardware Independent Layer	0	0	
Proximity.h	0	0	
Buzzer.h	0	0	
Services	0	0	
SL_HAL_Int.h	0	0	
Common.h	0	0	
Services.h	0	0	
MC9S08QE128.h	0	0	
Libs	12633	2225	
Project Settings	132	6	
22 files		13497	2249

图7 接近感测应用的文件结构

稍后解释。

2.2.2 应用层

SL提供基本的系统服务，它包括时钟和定时器初始化。有三个文件属于这个层：

- SL_HAL_Int.h — 此文件是SL和HAL之间的接口。总线的时钟频率就在此被定义，它是定时器设定和MCU时钟模块配置(ICG或IC被支持的微控制器)所需要的。
- Services.h — 所有时钟模块配置参数在此被定义为私有的。它们由SL来管理，没有其他任何模块可以利用这些值。基本调度功能被定义为公有的，它是通过配置定时器来产生周期性的中断。有个标记也被设置用

以表示此事件。这个标记可以被应用程序使用，或被HIL模块用于任务调度。MCU_Init函数原型也被声明在此，它必须由应用程序调用。

- Services.c — 微控制器初始化和时钟模块的初始化功能设置在这个文件中。应用程序必须调用MCU_Init函数来初始化必要的模块，以便系统运作。此函数用Clocks_Init和TIMER_CONFIGURE函数初始化时钟及定时器模块。

3 接近感测模块

3.1 综合描述

接近感测模块是本应用的核心。此模块管理每个电极的采样进程。它驱动数据的测量，并判定是否电极应该被报告为触及或未触及。它使用了GPIO和定时器模块。GPIO用于连接电极，定时器模块则用于配合测量电容。接近感测模块属于HIL，它给本应用提供了一个用于报告电极状态的公共变量和一个用于执行和存储每个电极先前采样值平均数的数组。

表1 给出了三个执行上述任务的函数。

表1 接近感测的函数表

函数名	输入参数	返回值
接近感测初始化	无	状态代码
电极的采样	端口指针，位掩码，缓冲区	状态代码
电极的状态更新	电极标识符	状态代码

3.2 接近感测的初始化函数

此函数通过对每个电极采样并存储为当前平均值，来设置平均值的初始状态。这使得设置时间减少为0，避免了在该过程中可能发生的虚假检测。如果样本值无效，平均值将被设置为0xFF，以避免虚假检测，而稳定的平均值将作为有效样本值。它没有接收数据，并返回用于报告任何可能发生的故障的状态代码。

3.3 电极的采样函数

该函数是这个模块所私有的，它执行在选择的电极引脚上测量电容的实际操作。这是通过利用GPIO和定时器模块来实现的。输入参数是个与期望电极相关联端口的指针、该电极所对应的特定引脚的掩码，指向用于存放测量值的变量。和函数返回状态代码，以显示测量是否是有效的。

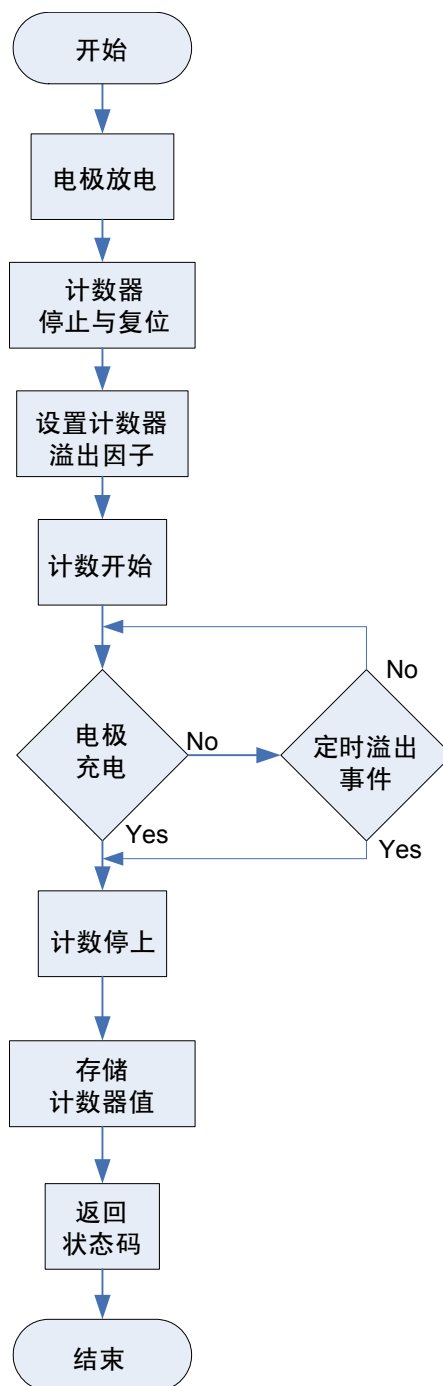


图8 电极采样函数流程图

如图8，首先，被选择的电极放电，置其初始状态为0V。该操作的执行是通过将某个电极的引脚设为输出，并将它的值设为逻辑0，以引起前次充电的电容连至地。接下来，中止测量计数器以防它继续计数，然后重置并使它重新开始。随后，电机的引脚被设置为输入。这就允许了电容重新充电，且它的值将始终被维持。该操作使测量进程开始，同时计数器开始计数，且电容器开始充电，尽管在计数器开始计数和开始充电之间存在延时，但其影响对测量值却并没有影响。当电容器的电压达到一定值，引脚值将被检测为逻辑1。在此之后，计数器停止计数，它的值也将被存储到指定的存储区。

如果因为任何原因电容器永远无法达到阈值，定时溢出事件可以使引脚停止轮流检测。该事件是由中断处理程序计数器通过软件标志发出的，当测量计数器开始时，定时溢出值被设置。中断标志位与引脚值一起被轮流检测，以确保CPU不会被死锁在具体的一个测量中。如果计数值溢出，则函数返回溢出状态，并表示读数是无效的。如果测量有效，函数返回一个成功状态码。

注意

在任何应用程序中使用接近感测模块时，谨记测量进行的时间里CPU是被死锁的。

而这个时间是由系统的底层电容决定的。最糟的情况莫过于设定定时溢出值(此应用程序为75us)。同样，如果另一个中断在引脚轮流检测时发生，取样也可能是无效的，因此用户必须执行一种方法来判别它。

3.4 电极的状态更新函数

该函数是应用层的端口。它必须被应用程序调用以更新电极状态。它执行的操作是对取样的平均值进行计算，并将新的取样与前一取样进行比较，来决定哪个电极已被接触。为了使目标电极获得新的取样，电极采样函数在更新函数中将被调用。该函数唯一的输入参数是被更新的电极(指针)，输入参数也可以是常数SAMPLE_ALL，这使得电极能被挨个的采样。

在图9中，首先输入参数被检验以避免系统错误。如果无效，出范围的状态码将被返回，之后函数退出。当参数是电极标志(E1,E2...En)时，取样函数将被调用以获得那个电极的当前电容值。在处理取样之前，状态码由取样函数返回，并被检查以确保读取有效。若无效，电极在被报告未被触及的同时，相应的状态码将被更新函数返回。如果取样有效，当前取样与之前取样平均值的差值将与阈值进行比较，以决定该变化对于考虑电极的被触及是否有足够的意义。随后，在gesElectrodeStatus变量中相应的位将被置位或清零，来反映比较的结果。最后，平均值被计算以应对下次的采样，同时返回成功状态码。如果输入参数是SAMPLE_ALL中的常数，前面描述的整个进程将对每个电极执行同一个函数调用。

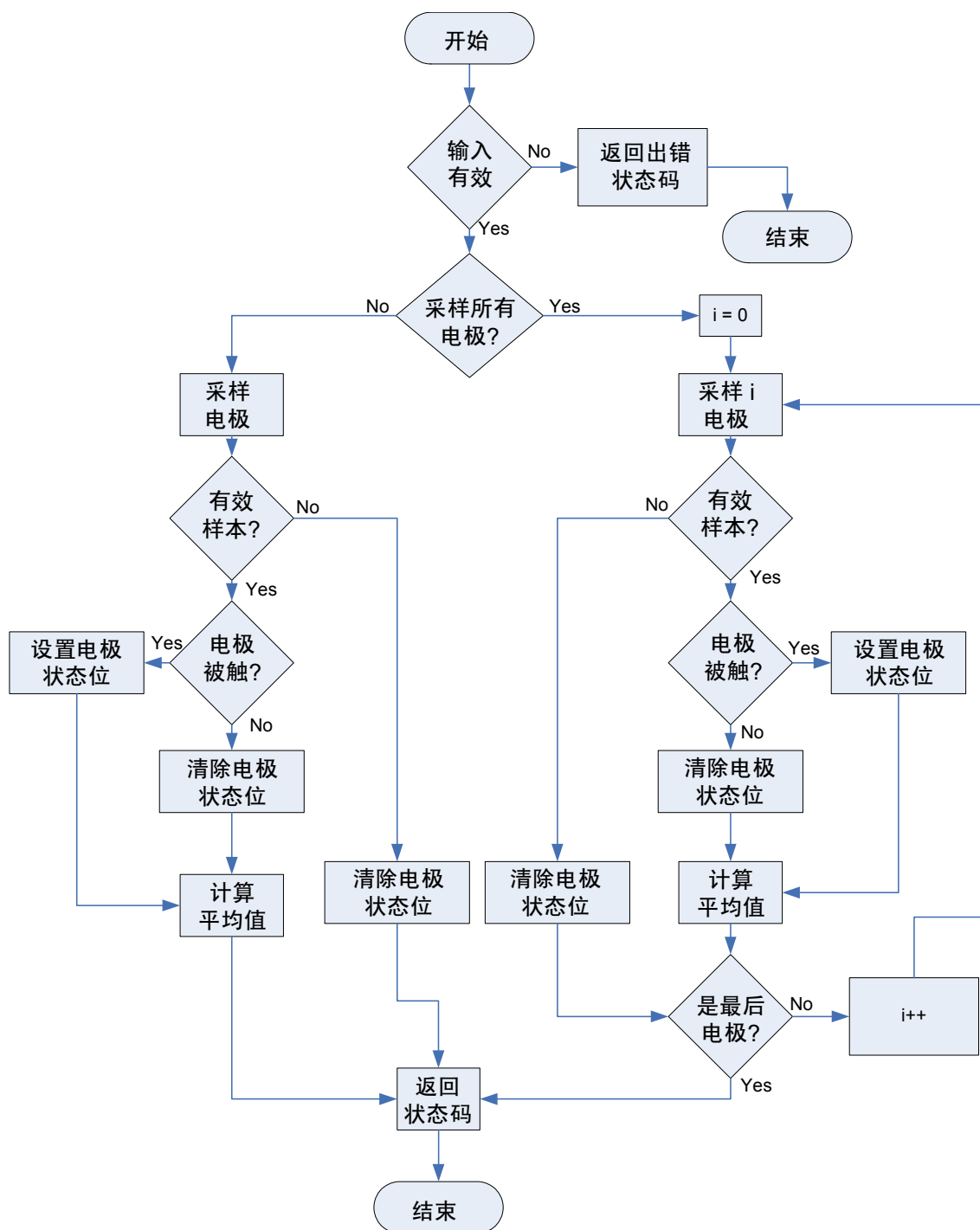


图9 状态更新函数流程图

3.4.1 平均值

均值滤波可作为低通滤波器，减小噪声的影响。在此模块中有一个函数就是均值滤波的实现(如等式2)。尽管它的响应很慢，但这种方法不需要存储采样值的缓冲区，同时也避免了每次都去整合所有采样值。因为数据是与整型值一起处理，相对可观的截断误差就出现了。如果系统的灵敏性很好，这种错误并不代表有什么问题。如果精度要求更高，平均值的计算将使用定点小数值。考虑到系统底层的电容器，这个操作可能以8位精度执行，也可能被扩展为16位。

等式2

$$Avg = Avg + \frac{(Sample - Avg)}{N}$$

N是与均值滤波取样数相等的常数。求8个取样的平均值时，N = 8。

考虑到方程3中定义的真正的平均值，带入方程2中，这样就获得了方程4。

等式3

$$Avg = \frac{\sum_{i=0}^{N-1} Sample[i]}{N}$$

等式4

$$Avg = \frac{\sum_{i=0}^{N-1} Sample[i]}{N} + \frac{(Sample[N] - \frac{\sum_{i=0}^{N-1} Sample[i]}{N})}{N} = \frac{(N-1)\sum_{i=0}^{N-1} Sample[i]}{N^2} + \frac{Sample[N]}{N}$$

对于定常态(所有前面取样值是相同的值)信号来说，方程4可以被描述为：

等式5

$$Avg = \frac{N(N-1)Sample}{N^2} + \frac{Sample[N]}{N} = \frac{(N-1)Sample + Sample[N]}{N}$$

用和式来表达等式5得：

等式6

$$Avg = \frac{\sum_{i=1}^{N-1} Sample[i] + \sum_{i=N-1}^N Sample[i]}{N} = \frac{\sum_{i=1}^N Sample[i]}{N}$$

注意：等式6的结果几乎和最初等式3的平均数相等，但是在等式6中总和的极限已经被改变。通过观察其运行

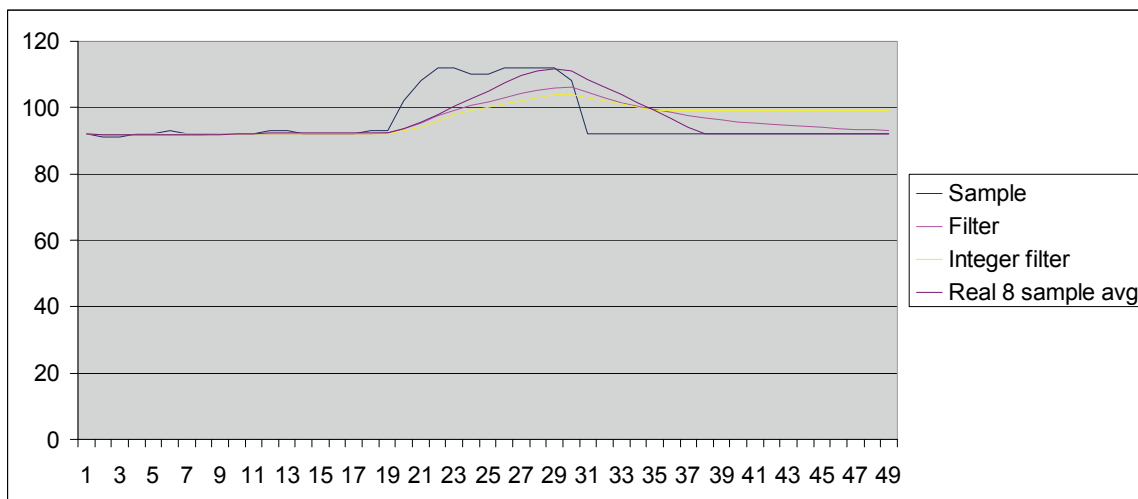


图10 均值情况

结果，等式2定义的操作(从平均值减去采样值)确实是奏效了。因为所有采样值相等，所以这个采样值被认为是最老的。新的采样值是对它先前各自分量平均的取整。

如图10所示，当偏移量很小状态稳定的时候它的效果很好。当信号改变时，它比实际均值滤波需要更长的时间来得到新的值。

3.4.2 阈值比较

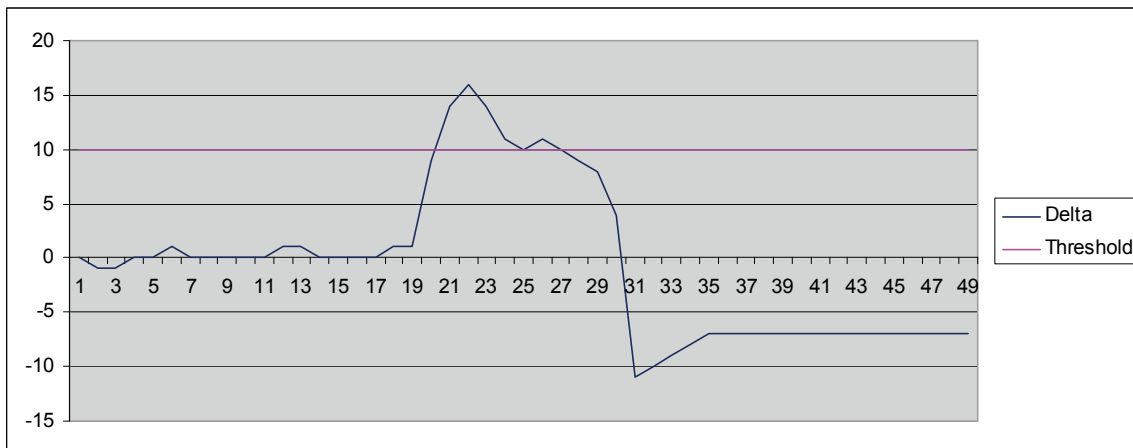


图11 阈值比较

平均值作为参考来与每个新的采样值比较，从而决定电极是否被接触。这些，是通过采样值减去平均值所得的差，和先前确定的阈值进行比较，来实现的。如果差值超过阈值，被报告的电极状态是接触。如果它低于阈值，电极的状态将被设置为未接触。

图11 显示了图10中采样值与平均值的差。当这个差值高于阈值(本例为10)时，报告的电极状态是接触。

3.4.3 电极状态寄存器

接近感测模块提供gesElectrodeStatus变量来显示每个电极的状态。变量中的每位都代表一个电极，与LSB的相应的是第一电极(E1)。该变量是一个ELEC_STAT_REG，且默认值是无符号8位数字。如果超过8个电极需要使用，ELEC_STAT_REG必须要重定义为16位。这个类型的定义连同其他类型一起在头文件实现。

3.4.4 接近感测应用的头文件

在接近感测模块中，“proximity.h”文件包括了公有和私有成分的定义。公有部分的组成包括ELEC_STAT_REG定义、公共函数的原型、公共变量的定义、电极符号和参数N_ELECTRODES。N_ELECTRODES定义模块所支持的电极个数。如果电极的构造改变，这个参数必须重定义以适合实际的构造。私有部分包含电极的宏定义和计数处理部分，定时器溢出参数被定义成TIME_LIMIT。这个参数使用在计数单元中。

4 硬件设置

当为特定系统的应用构建软件时，首先要明确目标平台。这决定了可用的硬件和内核的功能。目前此应用都支持RS08、S08和V1平台。

4.1 RS08、S08和ColdFire V1平台的移植

贯彻软件体系结构的主要目的是，实现代码的可移植性。这使得完整的应用软件很容易移植到不同的平台，各个单独的模块也可以重复使用在不同的应用程序中。为实现这种考虑，所有RS08、S08和V1系列微控制器被切分为两类不同的项目，一类针对RS08，另一类则用于S08和V1。

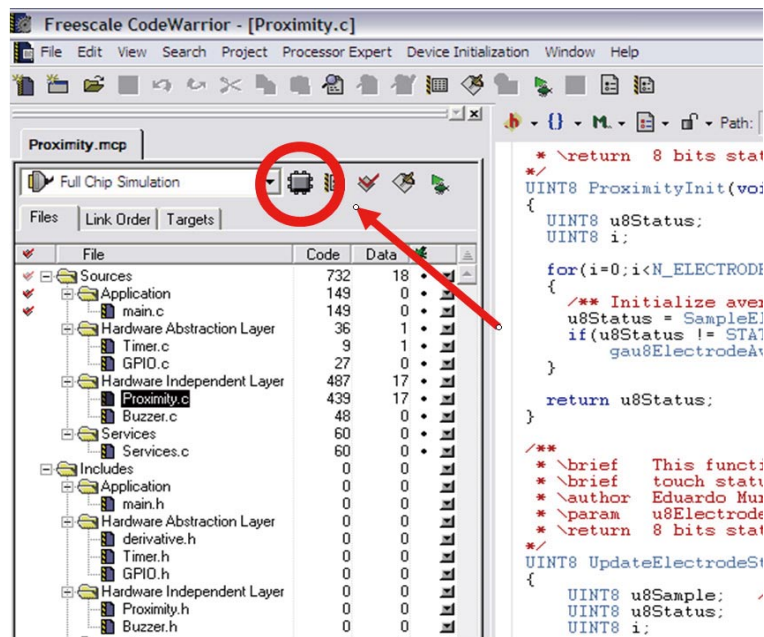


图12 切换MCU链接关系处

在工程被正确的打开后，具体的MCU必须被指定。RS08工程的默认芯片被设置为RS08KA2，而S08或V1默认被设定为S08QE128。CodeWarrior提供了一个功能来改变目标的MCU，此功能被调用后，会切换MCU的链接关系，它位于工程菜单，或在“Make and Debug”按钮区。如图12所示。

当选择此按钮时，有个显示了可用MCU列表的窗口会出现。目标MCU及其适当的与调试器连接的方法必须被选定。

注意

当使用RS08工程时，没有其他S08或V1版本的MCU可供选择，反之亦然。这两种内核在体系上的差异是导致产生不同的工程的原因。

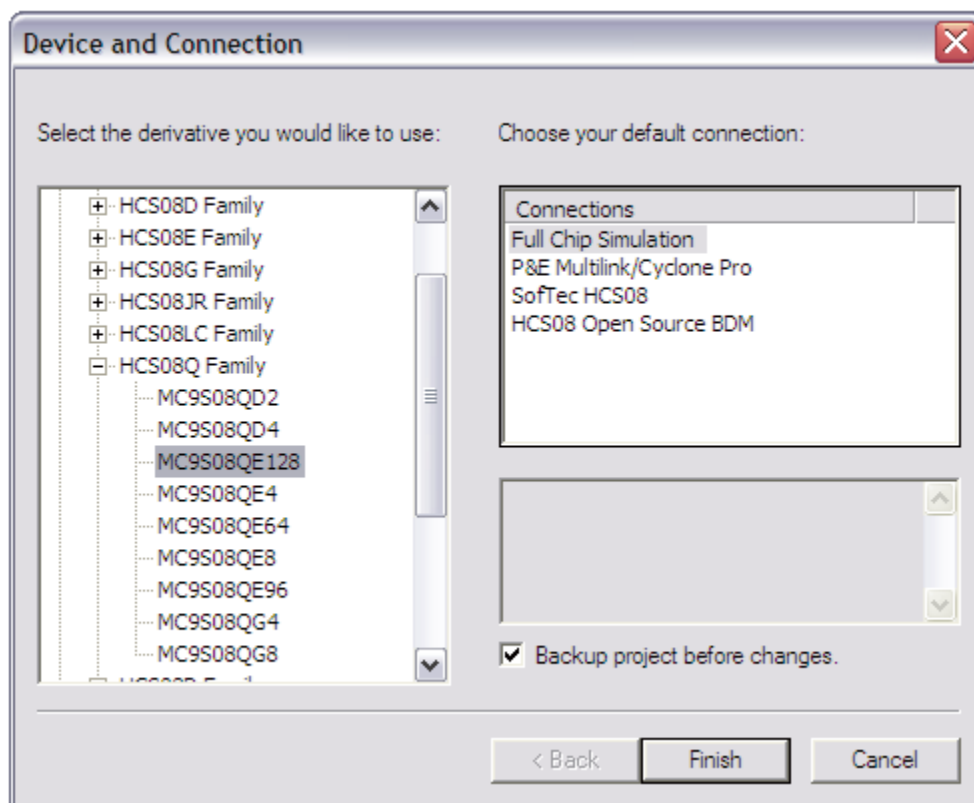


图13 MCU和链接选择窗口

4.2 配置电极和蜂鸣器的位置

后续定义的硬件集合，是那些用于电极和输出信号的GPIO引脚。在系统的硬件定义的基础上，电极及其引脚数目可通过该工程来配置。当使用带附加评估工具包模块的MCU演示板的时候，可从用户指南上发现电极的定位。这个位置是在工程的GPIO模块内被定义的。

4.2.1 GPIO 模块

为了提高灵活性，所有用于电极的MCU引脚被安排成一个引脚阵列。这便产生了一个为某个特定的功能而专责管理共同信号的虚拟端口。该阵列没有大小的限制，任何GPIO引脚都可被列入，并允许被配置成任何电极和输出。

注意

只有输入/输出引脚可用于接近感测。单独的输入或输出功能的引脚不可以连接到电极，因此也不能作为电极引脚阵列的一部分。

这些函数被包含在隶属于软件体系HAL层的GPIO模块中。每一个模块由一头文件和一个代码文件组成。对于本模块，每个文件的内容如下：

- 头文件——包含上层接口所需的宏，以及虚拟端口的类型定义。
- 代码文件——包含实际的虚拟端口定义。这里，电极和蜂鸣器的定位必须修改。

虚拟端口类型是一个结构，其包含一个指向MCU GPIO端口的指针，以及为选定引脚接口的位掩码。有了此信息可进行任何操作，来指定具体引脚，并且所有端口引脚指针都可归并成阵列以方便访问。这种结构是灵活的，因为通过索引变量可以动态的访问引脚。

4.2.2 蜂鸣器模块

为这个应用，创建了基本的蜂鸣器驱动模块。它使用一个GPIO引脚(该引脚是在GPIO模块定义的虚拟端口的一部分)发出10ms长的有固定频率的信号。不同频率的索引以及相应接口的宏和功能，都被定义在头文件中。

接口有以下组成：频率索引、用于初始化蜂鸣器输出引脚BUZZER_INIT()宏、用于切换蜂鸣器引脚电平的BUZZER_SOUND()宏，以及用于计算定时控制需要的值以产生预期频率的声音的BuzzerConfigure函数。此函数接受频率索引和一个存储10ms计数器的8位变量的指针。

BUZZER_OUTPUT的值存在于头文件的私有部分，并且用于选择蜂鸣器的虚拟端口的要素。在这个部分里，用于产生声音的定时值也被定义了。

4.3 定时器模块

定时器模块为整个应用程序提供定时功能。S08和V1工程支持TPM模块，RS08工程使用MTIM。此模块使用服务层来取得MCU总线时钟，并为预期的定时器时钟频率设定适当的时钟配置值。

头文件的公共部分包括了位结构类型的定义，该结构创建了，向上层报告中断事件的标志和HAL组件所需的所有宏接口。即使RS08和S08/V1使用的硬件模块不同，但为了兼容，也给出相同的接口。当不同的硬件被使用时，使用这些硬件的模块不需修改。

硬件模块配置所有必需的定义都可在私有部分找到。在某些微控制器中，可能有多个TPM模块可用，用户可选择其中的一个使用。这里，该模块基地址就是TPMxSC的地址，文件中其他的模块寄存器被定义为这个基地址的偏移。这样，用户就可以简单的通过改变TPMxSC，来实现对整个TPM硬件模块的控制了。定时器时钟频率参数必须在TIMER_CLK中设定，适当分频会自动的被设定用来实现就近的时钟频率。对于该应用程序，总线时钟是唯一的时钟源。

5 图形用户界面

接近感测的GUI软件可以使用户更好的了解配置和调试信息。应用程序用户可以看到电容测量的实时数据，且能用图形式和更直观的界面，来为触摸检测配置所需的阈值。因为是通过BDM接口与微控制器通信，所以这是一种非侵入式(non-intrusive)的嵌入式应用接口。这避免了MCU进行通讯时额外的开销。



图14 应用程序前端

5.1 GUI安装

此应用程序通过BDM接口直接从RAM读取数据值，而不干预CPU的处理。GUI必须要有用于存储数据的内存地址空间的支持。这是通过选择主窗口的配置选项，手工输入数组第一个元素的地址，以及选择系统中活跃电极的数目来实现的。

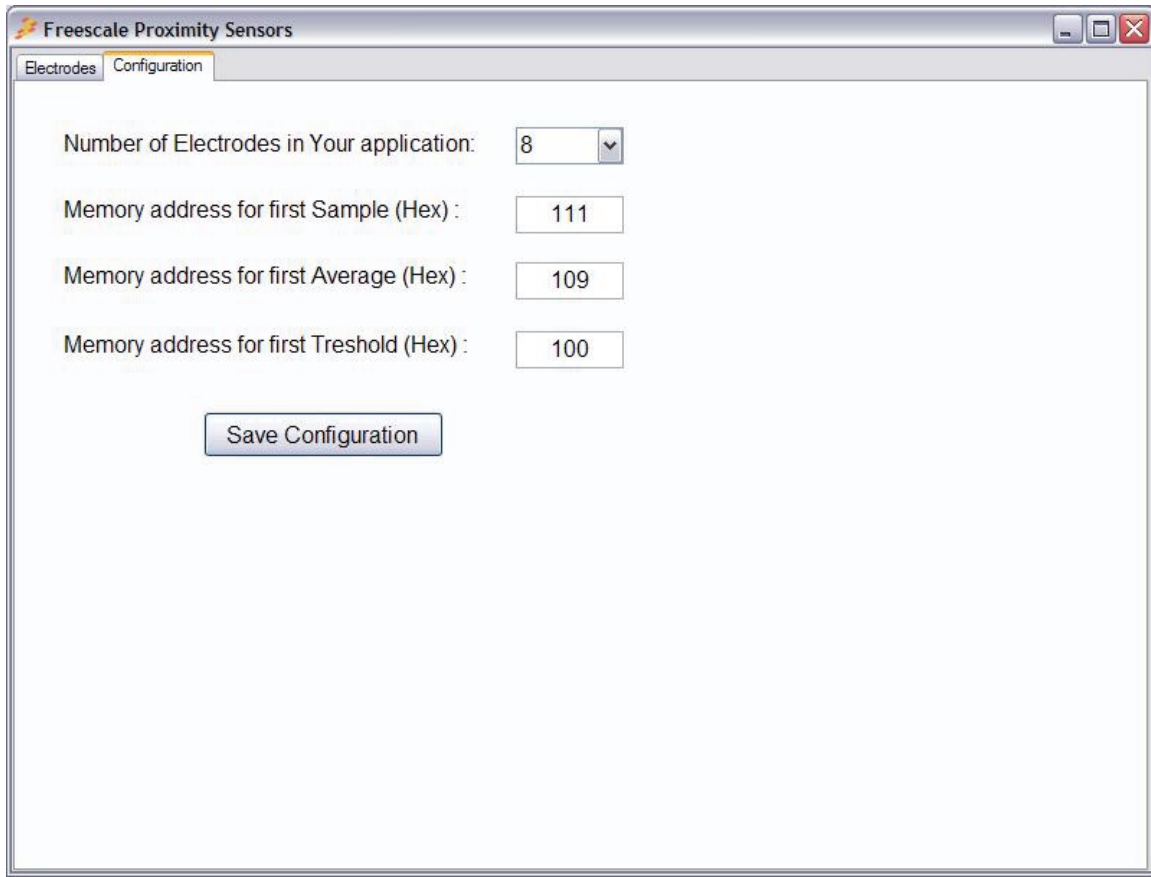


图15 数据配置选项

5.2 电容面板条

图14显示的面板给出了目前每个电极的电容。橘黄色的面板条代表平均值，白色的面板条代表当前的采样值。采样的数值显示在邻近面板条的文本框内。用户可即时地观察系统的行为，决定系统的基准电容，并观察当有物体靠近电极时电容的变化情况。

5.3 阈值检测

触摸检测的阈值可在每个电极面板条处进行配置。当有人接触到电极时，用户可以观察到电容的变化，并决定如何设置它的阈值。每次当电容值高于标记阈值时，邻近面板条的按钮将变成绿色。

注意

为了能正确进行阈值检测，GUI开启时必须保证没有任何电极被接触。

6 结论

6.1 接近感测软件的可重用性

因为没有对硬件的直接依赖性，由proximity.c和proximity.h文件组成的接近感测模块，在不同平台和不同应用之间的移植是极为方便的。如果用户要在不同的应用中整合触摸感测功能，则该接近感测模块可以被集成到工程中。下层接口函数必须提供对这个模块的支持。这包括计时器处理和该模块需要使用的GPIO函数。对于定时器模块，接口由表2中的元素组成。

表2 定时器模块接口

宏	描述
TIMER_START()	启动用于测量的定时器
TIMER_STOP()	停止计数器
TIMER_RESET()	设定定时计数器为0
TIMER_GET_COUNT()	返回8位计数器的值
TIMER_CONFIGURE()	调用一次用于初始化外部设备
TIMER_SET_MOD(x)	定义计数器值x用于引发定时事件

除了这些宏，有一个用于记录超时事件的位结构包含的标志位是需要的，它被声明为frTimer_flags.Bits.Timeout。在没有TPM可用的单片机中使用接近感测模块，必须构建以上所提到的所有定时器模块的功能函数。对接近感测模块来说，硬件的改变是透明的，因此没有修改的必要。

对于GPIO模块，表3给出了接口的元素。所有的宏都接受一个端口指针和具体引脚的掩码。

表3 GPIO模块显示

宏	描述
PIN_OUTPUT(x,y)	配置端口*x的引脚y作为输出
PIN_INTPUT(x,y)	配置端口*x的引脚y作为输入
PIN_SET(x,y)	置位端口*x的引脚y
PIN_CLEAR(x,y)	清除端口*x的引脚y
PIN_TOGGLE(x,y)	切换端口*x的引脚y

建议将端口指针和引脚掩码储存到VIRTUAL_PORT类型中，此类型是在应用程序所提供的GPIO.h文件中声明的。即使不同的GPIO模块被使用，也必须给接近感测模块提供同样的宏和结构体，如创建在GPIO.c文件中的vpPortx。

6.2 飞思卡尔接近感测解决方案

本文档所阐释的接近感测软件解决方案是很有价值的，但它本身并怎么不适合实际应用。虽然用户可以在接近感测模块的基础上增加其他算法，但其数据处理水平和数据运算能力在商业产品的综合应用中还不够可靠。飞思卡尔提供了，各种各样包含外部的数字和模拟传感器的，高度集成的接近感测解决方案。这些都集成了高效算法和有关接近感测的强劲设计。若想了解更多的信息，请访问网址www.freescale.com/proximity。

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2008. All rights reserved.

Document Number: AN3579
Rev. 0
11/2007

