AN11023

使用LPC11xx系列微控制器实现电容触摸感应

Rev. 1 – 2011年2月3日 应用笔记

文档信息

信息	内容
关键字	LPC1112、LPC1100、Cortex M0、电容触摸感应
摘要	此应用笔记描述了如何使用NXP半导体的LPC1100微控制器实现简单的电容触摸感应功能。

版本历史记录

版本	日期	描述
1	20110203	初始版本

联络信息

更多信息，请访问：http://www.nxp.com

如欲了解NXP销售办公室地址，请发送电子邮件至：saleaddresses@nxp.com

1. 简介

此应用文档描述了如何使用LPC11xx微控制器的ADC输入实现简单的电容触摸感应的功能。

在本应用笔记中，使用PCF8883评估板（见图1）的覆铜区域作为电容触摸感应区域。四块覆铜区域的每一块都和一个RC网络相连，并连接到微控制器的一个ADC输入通道。见图2。

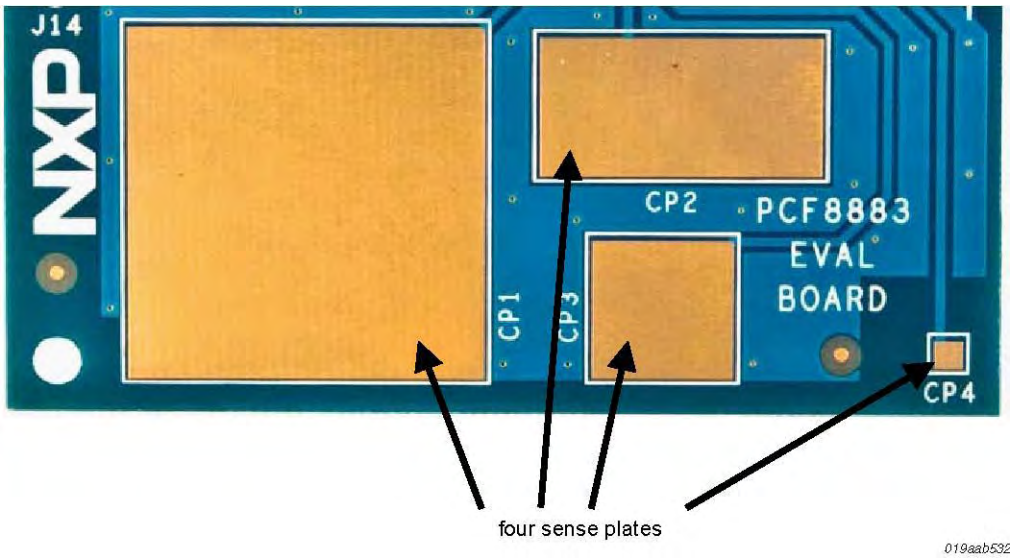

图1　PCF8883评估板上的四个感应区域

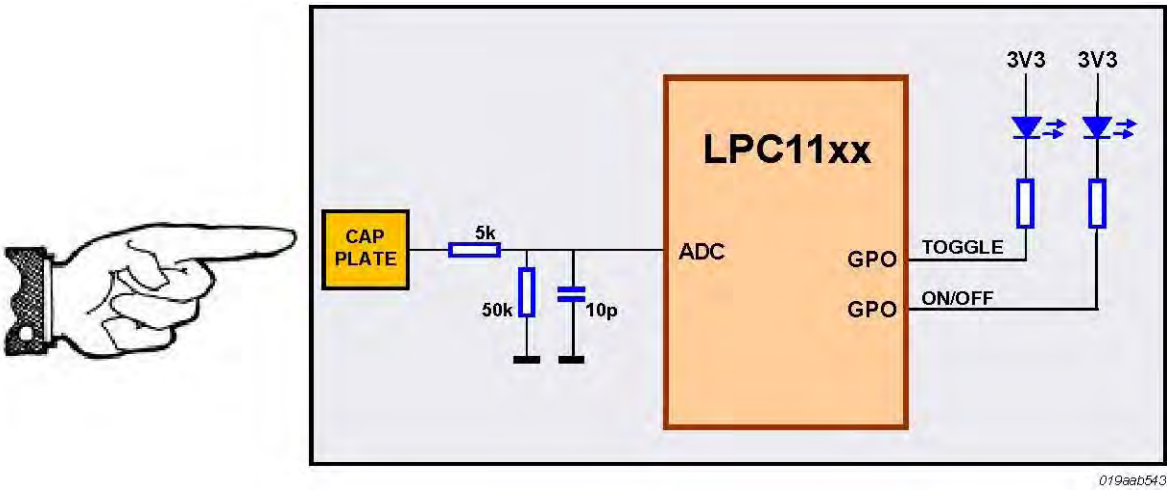

图2　电容触摸感应的简单系统框图

2. 工作模式

为完成电容触摸感应，需要使用一个微控制器引脚，它或被配置成ADC输入功能、或被配置成通用IO口的输出功能。信号读取的过程只需非常少的几个步骤，如图3。

首先，将IO引脚配置成高电平输出。此步骤将对外部10pF的电容和外部电容感应板进行充电。

然后，将IO引脚配置成ADC输入。此步骤将使外部电容和电容感应板通过电阻放电。在本示例中，使用5K-50K的电阻。当有手指接触在感应板上时，会增加电容量，从而使得放电过程变慢。如图3中，放电曲线变缓。

此后，将开始ADC转换过程。对感应板的接触会使ADC读取值上升。在本示例代码中，当感应板未触及时，ADC产生一个平均稳定的值，然后当ADC的值发生变化时，则认为检测到一次触摸。

最后，IO引脚被重新配置到原来的值，呈现高电平输出状态（回到第一步）。

触摸感应的步骤如图3：

1. 将感应线拉至VDD，作为数字信号输出（充电）
2. 将感应线配置成ADC输入（放电），开始ADC转换过程
3. ADC采样点。采样和保持花费一个ADC时钟周期。此后需要10多个时钟周期完成全部10位ADC的转换（设置DONE位，并将读取到的转换结果放置在寄存器LPC_ADC->DR[x]中）
4. 回到第一步。现在，查看读取到的感应板触摸结果，解码，去抖并完成触摸后的相应工作）

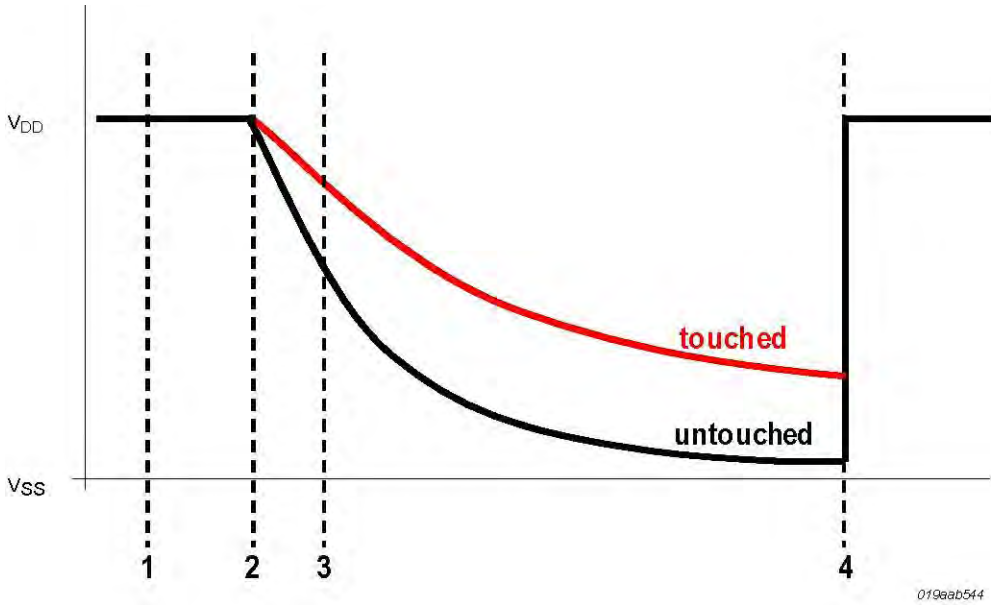

图3 感应线输入波形

3. Demo演示

下面的LPC1100示例代码使用ADC的输入通道1（PIO1_0）作为感应输入线，为了显示感应切换动作，使用2个输出端分别连接到LED灯上。

一个输出端PIO3_2工作在翻转模式（触摸感应或未触摸感应）

另一个输出端PIO3_3工作在瞬时切换模式，类似于一个按键。只要保持在电容感应触摸状态，那么此输出端为高电平，LED灯亮。

软件示例代码是用C语言编写的，并使用KEIL uVision (MDK-ARM V4.14)评估板编译器编译。LPC11xx微控制器配置时使用了KEIL提供的标准CMSIS启动代码（startup_LPC11xx.s和system_LPC11xx.c），并将CCLK设置为IRC，12MHz。

3.1 源代码

```
/*****************************************************************************
* Title   : LPC11xx Capacitive Touch Sensing demo program
* Hardware : MicroCore48 board + PCF8883 evaluation kit
*
* 1. Use Systick timer to generate a 10 msec timer tick (interrupt driven).
* 2. Capacitive sense plate is connected to P1.0 = AD1 input
* 3. Every 10 msec: use ADC to read the capacitive sense input
* 4. P3.3 = ON/OFF LED used to indicate press and release condition
* 5. P3.2 = toggle LED is used to indicate a new press condition
*****************************************************************************/
#include <LPC11xx.h>                          // LPC11xx definitions

static char k_press = 0;
static int  average = 0;

static short ADC_ReadCH1(void)                // read ADC channel AD1
{
    LPC_IOCON->R_PIO1_0 = 2;                  // set sensor line as AD1 input
    LPC_ADC->CR = 2 |                         // SEL = 2, select channel 1 on ADC
              (3 << 8) |                      // ADC_CLK = Fpclk / CLKDIV = 4 MHz
              (1 << 24);                      // start conversion

    while (!(LPC_ADC->DR[1] & 0x80000000));   // wait until end of AD1 conversion

    LPC_IOCON->R_PIO1_0 = 0x81;               // sensor line is output high
    LPC_ADC->CR &= 0xF8FFFFFF;                // stop ADC
    return (LPC_ADC->DR[1] >> 6) & 0x3FF;     // return A/D conversion value
}

void SysTick_Handler(void)                    // SysTick Timer ISR every 10 msec
{
  static char  debounce = 0;
  static char  avgindex = 0;
        char  result   = 0;
        short reading;
   reading = ADC_ReadCH1();                   // read AD1 = Cap sense input

```

```
38     if (reading > average + (average >> 4))      // above (average + 6% of average)?
39     {
40        if (debounce == 4)                        // debounce, 4 triggers for press
41        {
42           k_press = 1;                           // reached max, indicate pressed
43           result  = 1;                           // set result for return value
44        }
45        else
46           debounce ++;                           // still going toward max
47     }
48     else if (reading < average + (average >> 5))// below (average + 3% of average)?
49     {
50        if (debounce == 0)
51        {
52           k_press = 0;                           // reached min, indicate release
53           result  = 0;                           // clear result for return value
54        }
55        else
56           debounce --;                           // going toward min
57     }
58
59     if (result == 0 && debounce == 0)            // recalculate average
60     {
61        if (++avgindex == 8)                      // average index delay
62        {
63           average = (reading + (15 * average)) / 16;
64     avgindex = 0;
65        }
66     }
67  }
68
69  int main (void)
70  {
71    static char  toggle = 0;
72    static char  ledon  = 0;
73          short i;
74
75     SystemInit();
76     LPC_GPIO1->DIR |= (1<<0);                    // P1.0 connected to cap sense plate
77     LPC_GPIO3->DIR |= (1<<2);                    // P3.2 = toggle LED
78     LPC_GPIO3->DIR |= (1<<3);                    // P3.3 = ON/OFF LED
79
80     LPC_SYSCON->PDRUNCFG     &= ~(1<<4);         // disable pd bit to the ADC block
81     LPC_SYSCON->SYSAHBCLKCTRL |=  (1<<13);       // enable AHB clock to the ADC
82
83     for (i=0; i<200; i++)                        // warm up, establish average
84     {
85        average = (32 + ADC_ReadCH1() + (15 * average)) / 16;
86     }
87     SysTick_Config(SystemCoreClock/100);         // generate interrupt each 10 ms
88
89     while (1)
90     {
91        __wfi();                                  // go to sleep
```

```
92
93          if (k_press)                                // key pressed ?
94          {
95             LPC_GPIO3->DATA &= ~(1<<3);              // P3.3 low = LED ON
96             if (!toggle)
97             {
98                toggle = 1;
99                if (!ledon)
100               {
101                  ledon = 1;
102                  LPC_GPIO3->DATA &= ~(1<<2);        // P3.2 low = LED ON
103               }
104               else
105               {
106                  ledon = 0;
107                  LPC_GPIO3->DATA |= (1<<2);         // P3.2 high = LED OFF
108               }
109            }
110         }
111         else                                        // key released
112         {
113            LPC_GPIO3->DATA |= (1<<3);               // P3.3 high = LED OFF
114            if (toggle)
115            {
116               toggle = 0;
117            }
118         }
119      }
120   }
```

4. 参考文献

更多细节，请参考如下出版物：

- 数据手册、用户手册、应用笔记和示例代码：

http://ics.nxp.com/microcontrollers/

- AN10832: “PCF8883 - capacitive proximity switch with auto-calibration”：

http://www.nxp.com/documents/application_note/AN10832.pdf

- UM10370: “User Manual for the PCF8883 Evaluation Kit OM11055”

http://www.nxp.com/documents/user_manual/UM10370.pdf

5. 免责声明

有限保修和责任－本文档中的信息被认为是准确和可靠的。然而，对于信息的准确性和完整性，恩智浦半导体公司不给予任何陈述或担保，明示或暗示，对于此类信息的使用后果不负任何责任。

在任何情况下，恩智浦半导体不会承担任何间接、意外发生、惩罚性、特别或相关性的损害赔偿（包括单不限于利润损失、储蓄损失、业务中断、有关去除或更换任何产品的费用或返工费用），不管这些损害赔偿是基于侵权（包括疏忽）、保修、违约合同或其他法律理论。

对于客户无论任何理由可能招致的任何损害，恩智浦半导体为在这里所提到的产品的汇总和累积责任应限制在恩智浦半导体商业销售的条款及条件里面。

变更的权利－恩智浦半导体有权在任何时间对此文件发布的信息(包括单不限于规格和产品说明）做出任何改动。本文件将取代所有之前所公布的信息。

适用性－恩智浦半导体产品并非为那些用于对生命和安全有重大关系的系统和设备而设计、授权或提供保证，也不用于那些可以合理预见到的因恩智浦半导体的产品的故障会造成人身伤害、甚至死亡、或是严重的财产或环境损害的应用程序中。恩智浦半导体的产品如果应用在此类的设备或应用程序中，恩智浦半导体对所此造成的风险将不承担任何责任，因此这些风险有客户自行承担。

应用－在这里所描述有关产品的任何应用程序仅用于说明的目的。在没有进一步的测试或修改的情况下，恩智浦半导体对该应用程序对指定用途是否合适不作任何表示或保证。

客户应对其使用恩智浦半导体产品的应用以及产品的设计和运行自行负责，恩智浦半导体不负责协助应用程序或客户的产品设计。同时，客户应自行负责决定恩智浦产品是否适合客户应用、计划产品、计划的应用程序以及第三方客户使用。客户应提供适当的设计和运行的保障措施以尽量减少其产品与应用的相关风险。

因客户的应用或产品的弱点或缺陷所产生的，或因使用其第三方客户的产品而产生的任何缺陷、损失、费用支出和问题，恩智浦半导体不承担任何责任。客户应负责为其使用恩智浦半导体芯片的产品或应用以及其第三方客户使用产品或应用做必要的测试，以避免使用不当而造成不必要的损失。恩智浦对在此方面不承担任何责任。

限制值－超过一个或多个限制值（如在IEC60134的绝对值最大额定义）的施压会对设备造成永久的损害。限制值只强调额定功率，这个设备的操作除了应用在此文件中所提到的“推荐工作条件”和“特征”部分之外，恩智浦半导体不担保超过上述要求的操作。恒定或反复超出限制值将永久地和不可逆转地影响设备的质量和可靠性。

商业销售条件－恩智浦半导体产品的销售适用公布于http://www.nxp.com/profile/terms网站上的通用商业销售条款，除非另存一个单独有效的书面协议，在此种情况下，将适用该单独有效的书面协议之条款和条件。关于客户采购恩智浦半导体产品，恩智浦半导体在此明确拒绝适用客户的通用条款和条件。

不构成任何出售要约或许可－本文中任何部分都不可被翻译或解释成可以开放接受或授予、转让或任何暗示许可版权、专利或其它工业或知识产权的销售产品要约。

出口控制－本文件以及其项目描述可能受出口管制条例限制。出口可能需事先获得国家机关许可。

非车规级产品－除非数据手册明确标出此恩智浦半导体产品为车规级，否则该产品不适合于汽车应用。该产品未在汽车产品测试和应用条件下经测试和质量认证。恩智浦半导体对客户将非车规产品运用在汽车设备和应用中不承担任何责任。

当客户使用该产品设计并使用在需要车规级规格和标准的汽车应用时，(1) 客户在该汽车应用、使用和规格中使用恩智浦半导体产品时，不在恩智浦半导体对该产品的保证范围内；(2) 当在汽车应用中使用超出恩智浦半导体规格的产品，客户应该自行承担风险；(3) 因客户超标准和产品规格使用恩智浦半导体产品导致的影响、损坏和失效产品索赔，客户不能要求恩智浦半导体进行赔偿。

6. 目录