防水電容式感測技術

Victor Kremin與Ruslan Bachunskiy

摘要

電容式感測系統能應用在各種暴露於雨天或潮濕的環境中，如汽車方面的應用、戶外設備、自動櫃員機（ATMs）、公用系統、手機或PDA等可攜式裝置、以及廚房及衛浴應用。本篇應用說明將討論使用電容感測技術的系統如何在潮濕、下雨或有水的環境下仍能維持運作。即使感測範圍的表面完全被水覆蓋，電容式感測技術也能防止觸碰偵測誤判動作發生。

簡介

電容式感測技術能有效應用於可能暴露在潮濕、下雨、或有水的環境之裝置中。PSoC® CapSense讓您不需使用高成本的機械式開關，並能同時提昇裝置的可靠性與降低整體系統成本。

防水電容式感測技術的應用範圍相當廣。車內應用包括車門開啟裝置、密碼輸入系統、警報感測器、以及電容式近距停車感測器等。防水電容式感測器亦可廣泛應用在居家生活中，包括廚房設備、浴室燈光開關裝置、自動式水龍頭、洗碗機或洗衣機等。此外，防水電容式觸控螢幕也可應用於自動櫃員機（ATMs）、PDA、手機、可攜式GPS導航系統、以及其他室外用的裝置。

本實驗主要是利用鍵盤上三個簡易式感測按鍵之模擬應用，針對防水電容式感測技術進行測試。您可以進一步修改擴充實驗規模，以符合您特定應用之需求，例如增加按鍵數目，使用不同尺寸與形狀的按鍵等。本實驗所用到的技術特性如表1所示。

表1. 鍵盤特性

特性	數值
按鍵數	3
感測器尺寸	15 x 15 mm
主機通訊介面	I^2C（除錯用）
絕緣覆蓋層厚度	1-5mm（玻璃或塑膠）
電源供應器之電壓	5 ± 0.25V

實驗安排

首先用一個簡單的試驗來測試水對於電容式感測介面的影響。測試裝置採用了CYC821x34 PSoC系列元件以及CSD（CapSense Sigma Delta）使用者模組感測方法。PSoC元件置於感測器電路板上。為了顯示感測器狀態，

我們在另一小塊電路板上額外安裝了幾個LED燈，並連接至PSoC電路板上。此顯示用的電路板在測試時可以安置在不同的地點，以協助感測器狀態的觀察。

為了監測感測器、原始計次數值、以及其他即時測試所得資料，我們採用了I^2C-USB橋接器。該橋接器的GUI PC工具可同時監測多項數據軌跡並記錄成檔，也可以根據所收集的資料計算數值規模與偏移係數。圖1所示為實驗環境。

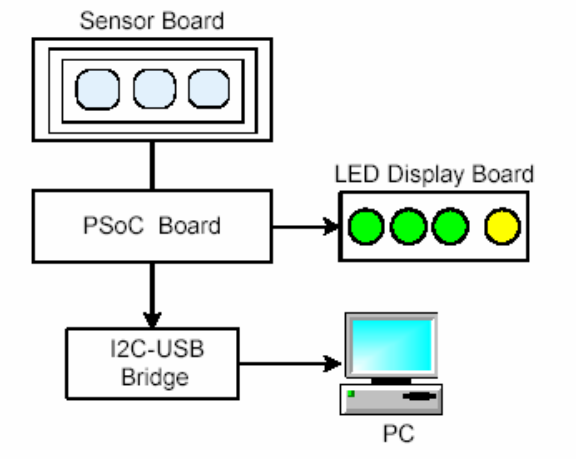


圖1. 實驗環境

感測器建置於一片獨立的電路板上，並用軟質的電線與PSoC CapSense電路板相連接。由於感測器電路板與CapSense PCB分開，因此感測器電路板能很輕易地運用在不同的狀況中，例如在感測器電路板上加上水流。此感測器電路板以2-mm厚的單面FR4銀箔材質製作而成。圖2所示為感測器電路板配置示意圖。

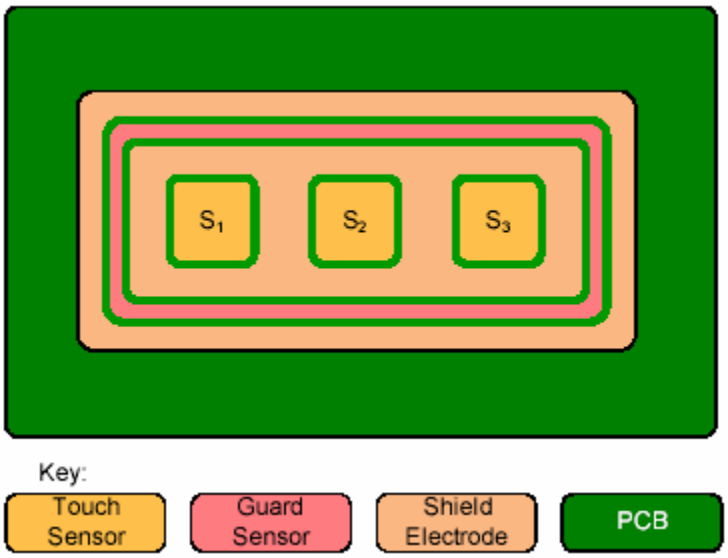


圖2. 感測器電路板配置示意圖

感測器電路板包含三個觸碰感測器（touch sensor）、一個圍繞成圓圈的防護感測器（guard sensor）、以及兩個平行放置的遮蔽電極（shield electrodes）。

三個觸碰感測器被內圈的遮蔽電極所包圍。內圈的遮蔽電極是用來避免按鍵感測器受水滴的影響而產生觸碰誤判偵測情形。內外圈遮蔽電極之間則還有一圈防護感測器，可用來偵測水流。外圈遮蔽電極則是避免防護感測器受水滴的影響而產生觸碰偵測誤判情形。使用遮蔽電極也可降低sigma delta modulator的輸入電流，以減少寄生電容的影響。

將水從絕緣覆蓋層的那一側倒在感測器電路板，為了防止感測電極直接碰到水，上面都會漆上一層絕緣的透明漆。連接的電線也都經過防水絕緣處理。感測器電路板的金屬區域尺寸為95*45 mm。每個感測器為15 mm見方，至於防護感測器的寬度則為2.5 mm。

測試設定線路圖

圖3所示為測試設定的線路圖。附帶限流電阻的LED燈則裝設於獨立的顯示用電路板上。標準的CY3212或CY3213電路板皆能用於本實驗中。其中CY3213板子支援CSD方法。CY3212要支援CSD方法則需裝設調諧器迴授電阻（modulator feedback resistor）。

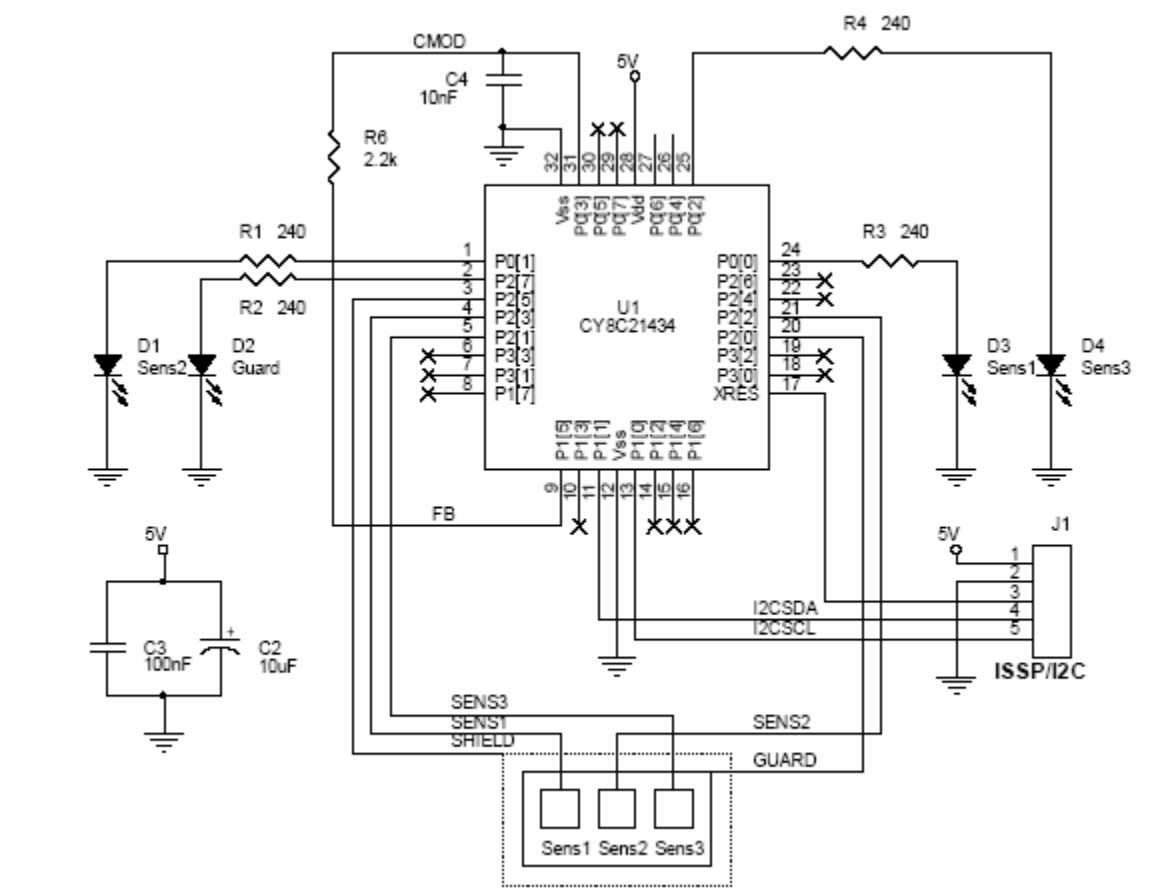


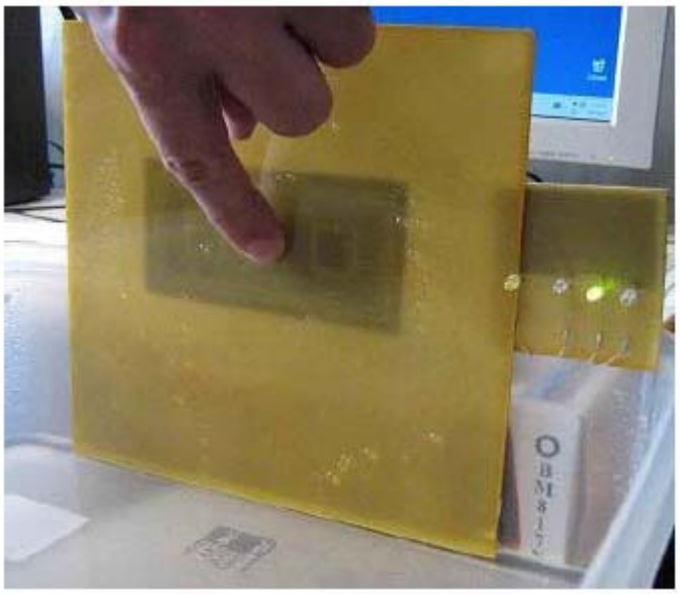
圖3. 測試設定線路圖

水的影響測試

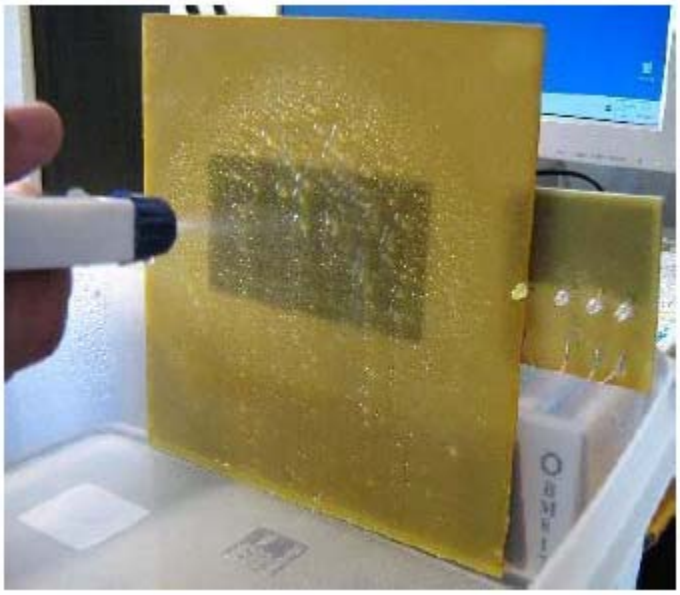
下列所有測試主要是針對水對於原始感測器計次值（raw sensor counts）的影響。感測器電路板在所有試驗中都直立擺放。為了模擬最惡劣的情形，測試所用的自來水中還加入了食用鹽，以增加水的導電性。

下列幾項為完成的測試：(圖a至圖d為各項鍵盤防水測試照片)

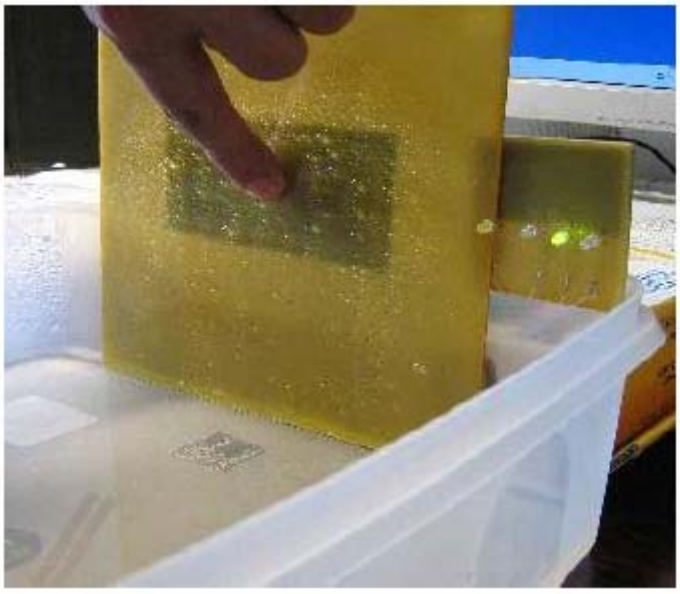
- 用手指觸碰乾的感測器。
- 當遮蔽電極接地時，用噴霧器噴灑小水滴在鍵盤上。
- 當絕緣板上佈滿水滴與水滴流動時，用手指觸碰感測器。
- 當遮蔽電極啟用時，再用噴霧器噴灑小水滴。
- 從大杯子中將水流持續倒在電路板上。



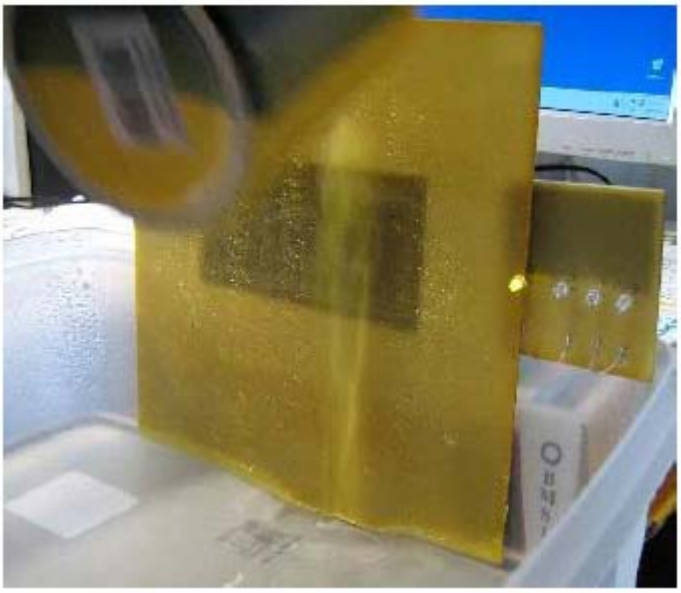
圖a. 為乾燥鍵盤測試。以手指觸碰感測器的中央時，中央的LED燈也隨之亮起。



圖b. 為鍵盤噴灑水滴後的情形。由照片可以發現並無任何LED燈亮起。



圖c. 為手指觸碰佈滿水滴鍵盤中央的感測器時，中央的LED燈也隨之亮起，且無任何誤判偵測發生。



圖d. 為水流流過鍵盤時，旁邊的防護感測器LED燈亮起。當防護感測器觸發時，其他感測器就無法被觸發。

為了讓這些收集到的資料易於觀察分析，從不同感測器所得的原始計次資料都利用I²C-USB bridge GUI PC工具上的偏移（Offset）功能進行位準調整。本範例中的設定如圖4所示。而測試結果如圖5所示。

N	Acti..	Variable Name	Type	Si...	Sc..	Offset	Color
1	☑	RawCount1	int	☐	1	-300	Black
2	☑	RawCount2	int	☐	1	-400	Blue
3	☑	RawCount3	int	☐	1	0	BlueViolet
4	☑	Guard	int	☐	1	0	Magenta

圖4. I²C-USB GUI變數設定範例

此外，在水滴測試中也檢視了不同調諧器參數設定所造成的影響。這些結果之後會再進一步討論。

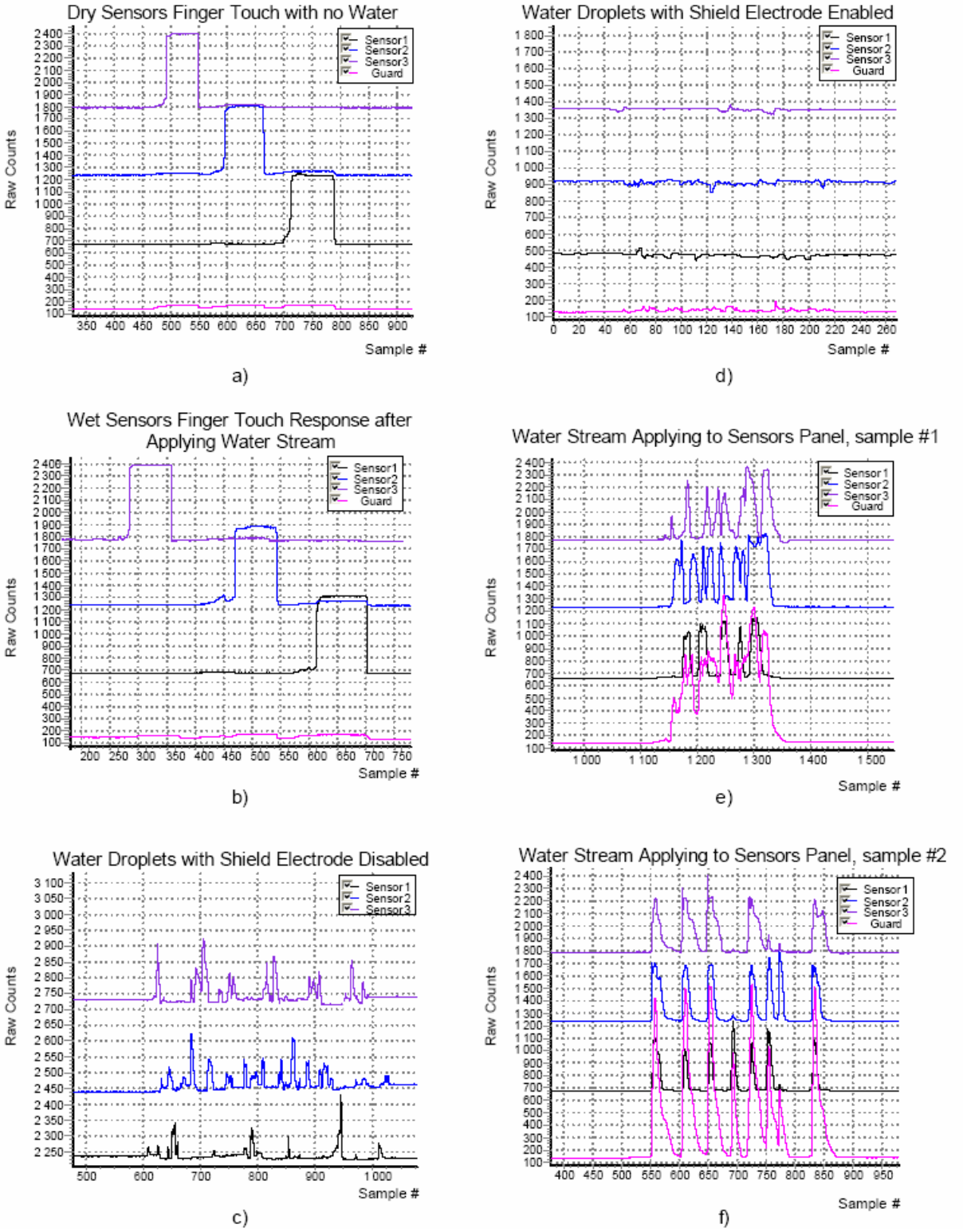


圖5. 針對水的影響所作的測試結果

第一個測試檢查在沒有水的狀況下觸碰鍵盤的運作情形，手指依序輪流觸碰感測器（見圖5a）。在圖中可以發現指觸能輕易地被偵測出來。

第二個測試檢查當水滴存在時、遮蔽電極啟動、且手指依序觸碰感測器時，感測器的功能狀況（見圖5b）。在圖中可以發現訊號上有小突波出現，這是因為手指實際上是先碰到水滴，再碰到板子。這些小突波與指觸訊號比較起來幾乎是微不足道的，且不會造成任何觸碰偵測問題。由此可知，指觸訊號依然可以很輕易地被偵測到。

第三項測試檢查水滴噴灑在感測板上，且遮蔽電極關閉時，感測器的功能狀況（見圖5c）。圖中顯示噴灑水滴所造成的訊號突波與指觸的訊號變化相近。指觸原始計次數大約比基準線多400次；而噴灑水滴所產生的訊號計次數則是高於基準線200次，大約是指觸效果的一半，這在某些情形下會造成誤判指觸的偵測結果。

第四項測試則是檢查水滴噴灑在感測板上，且遮蔽電極啟動時，感測器的功能狀態（見圖5d）。從圖中可以發現噴灑水滴造成的突波還不到指觸訊號變化的六分之一，兩者很容易就能作出區隔。因此，遮蔽電極能有效地防止鍵盤控制器產生觸碰誤判偵測情形。

下一個測試是檢查當直接到水在電路板，且遮蔽電極啟動時，感測器的功能態（如圖5e與5f）。由圖中可以發現水流在所有感測器上所造成的訊號突波效果與指觸相當，因此會造成指觸誤判的偵測結果。然而因為防護感測器的感應面積比較大，因此在防護感測器上所造成的訊號甚至比觸碰感測器的訊號來的高。防護感測器是用來判斷電路板上是否有水流，一旦偵測到水流時，就立即中斷決策邏輯（decision logic）的工作。

圖6所示為CapSense在潑水測試時的配置示意圖。當手指碰觸乾燥表面時，手指在感測電極上所增加的電容值為C_x（如圖6a）。當水滴落在感測器與遮蔽電極之間時（如圖6b），兩電極之間的電容耦合就會增加（經由C_{wd}與C_{ws}的電容質）。

若感測表面有連續水流時（如圖6c），水流本身會帶來相當大的電容值C_{st}，且其可能是遮蔽電極與水之間電容值（C_{wd}）的好幾倍。因此，遮蔽電極在此情況下完全起不了作用，也造成感測器的原始計次值與指觸相當，甚至更高。遇到這種情形，防護感測器就能派上用場，當它偵測到水流存在時，可先中斷其他感測器的觸發。

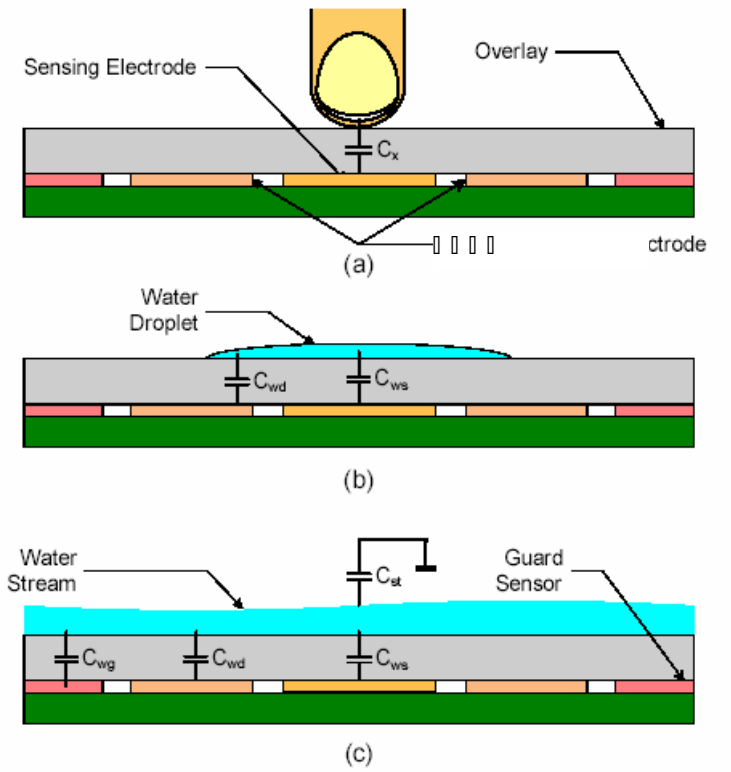


圖6. 乾燥表面之感測 (a)，小水滴附著於表面 (b)，及水流流過表面 (c)。

此外，我們亦在不同sigma
delta參考電壓設定下，進行數個額外的測試，以決定這些設定對測試有何影響。圖7為其中兩項測試結果。其中原始計次值變動較小的區段是當感測器處於乾燥的狀態，之後則是有水滴噴灑在絕緣層表面的結果。

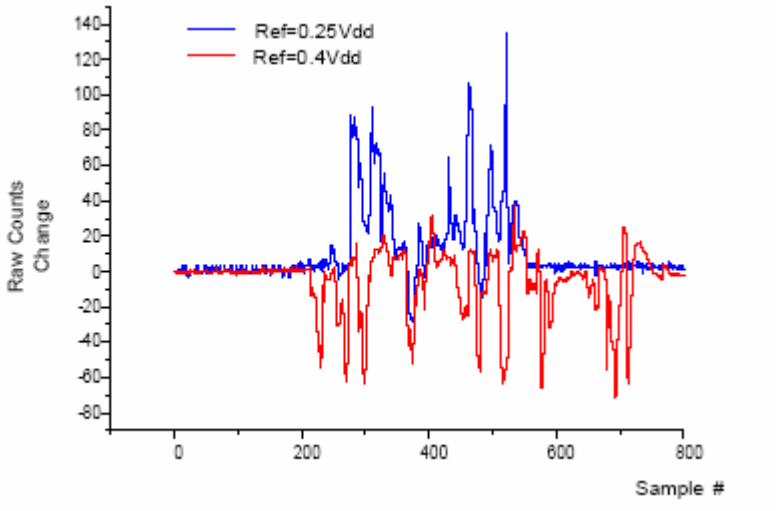


圖7. 當水滴在不同參考電壓設定下，所產生的原始計次值變動情形

由圖7可以發現，在較低的參考電壓設定時，多數的突波都為正向，也就是和指觸所造成的變化同方向。當參考電壓升高時，突波幾乎都低於「無水」狀態時的位準，因為當水滴落在遮蔽與感測電極之間時，較高參數的調諧器電流減少的程度也較大，因而造成突波方向與指觸訊號變化方向相反的情形。高階API常式可善加利用此一特性，當只有幾小段負向突波時，並不用重設基準線的位準。目前1.0版本的CSD使用者模組並不支援這項功能，但有可能會新增到未來的版本中。

當水滴噴灑在絕緣層上時，水在絕緣層表面所形成的薄膜會增加感測電極的電容值。這就是為什麼在較低的調諧器參考電壓時，原始計次值的變化較偏正向，而較高的參考電壓則會偏負向變化。這是由於遮蔽電極的增加效果。

此處的目標就是要藉由選擇最佳的參考電壓，以獲得指觸訊號與水滴突波訊號之間最大的差異性，特別是當水滴會造成負向突波的時候，因為此時CSD API會將基準線重設成最小值。圖8所示為不同參考電壓下，水滴的「訊雜比」（signal-to-noise ratio；SNR）結果。

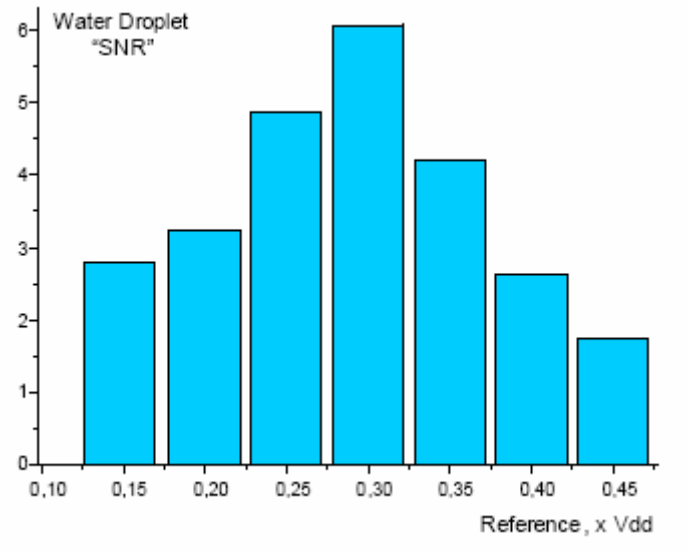


圖8. 在不同參考電壓設定時，指觸反應與水滴雜訊之間的關係

在此圖中您可以發現，當參考電壓為0.30 Vdd時呈現出最大的訊雜比，而這項結果在CSD Um參數中會用*Ref Value* = 1代表。不同的電極與覆蓋層配置方式所得到的電壓值也會不同，因此您的配置可採用不同的電壓值。在另一版本的感測板上，所得到的最佳結果為*Ref Value* = 2。

我們所使用的簡易模組與實驗結果清楚顯示出，要可靠地進行觸碰偵測，並消除水滴與水流所造成的偵測誤判，必須採用結合遮蔽電極與防護感測器的設計。遮蔽電極可在實體層降低水滴的影響；而防護感測器則可在邏輯層中重設決策邏輯的運作。下一章節中將會討論一套高階資料的處理機制。

示範韌體

本韌體利用CSD User Module，其中已經加入後端處理的部分，以處理防護感測器訊號。更新的基準線演算法維持不變。此處選用的CSD User Module參數如圖9所示。而要注意的是，當中的Sensors Autoreset參數設定為啟用（enabled），讓基準線在更新時無須考慮感測器的狀態。

User Module Parameters	Value
FingerThreshold	150
NoiseThreshold	10
BaselineUpdateThreshold	150
Sensors Autoreset	Enabled
Hysteresis	10
ESDDebounce	Disabled
Scanning Speed	Slow
Resolution	12
Modulator Capacitor Pin	P0[3]
Feedback Resistor Pin	P1[5]
Reference	ASE11
Ref Value	1
ShieldElectrodeOut	Row_0_Output_1

圖9. CSD UM參數

韌體掃瞄觸碰感測器與防護感測器時，會分別依照各自的*FingerThreshold*與*Reference*這兩個不同的參數值進行掃瞄，因為這兩種感測器的建置區域與用途都不一樣。防護感測器位於觸碰感測器的四周，所佔的電路板面積也較大。在設計上，防護感測器應比觸碰感測器具備較佳的靈敏度與更低的*FingerThreshold*值。

此處也採用了一套特別的演算法，以提供可靠的指觸偵測，並消除指觸偵測誤判的情形。這套演算法為每個觸碰感測器與防護感測器提供一個建置在韌體內的特殊可重置式計數器（resettable counter）。這些計數器皆具備debounce功能。觸碰感測器上的計數器會設定最小的時間間隔，在這期間必須觸碰感測器，讓決策邏輯偵測實際觸碰。觸碰計數器讓您可以消除因漏水而造成的短時間訊號突波，並且防止觸碰偵測誤判情形發生。

當水流流過感測器電路板後，防護感測器的計數器讓您可消除殘餘水滴所造成的觸碰偵測誤判情形。當電路板上有水流時，防護感測器會偵測到此一狀況，並且停止觸碰感測器的處理邏輯。額外防護感測器的「靜止」（dead）時間可避免感測器的計數器過早解除鎖定。當水流通過後，防護計數器仍會抑制觸碰感測處理一小段時間。如此可消除電路板上殘餘水滴所造成的觸碰偵測誤判。圖10為此套演算簡化後的等效線路圖，讓您可以更容易瞭解運作的方式。

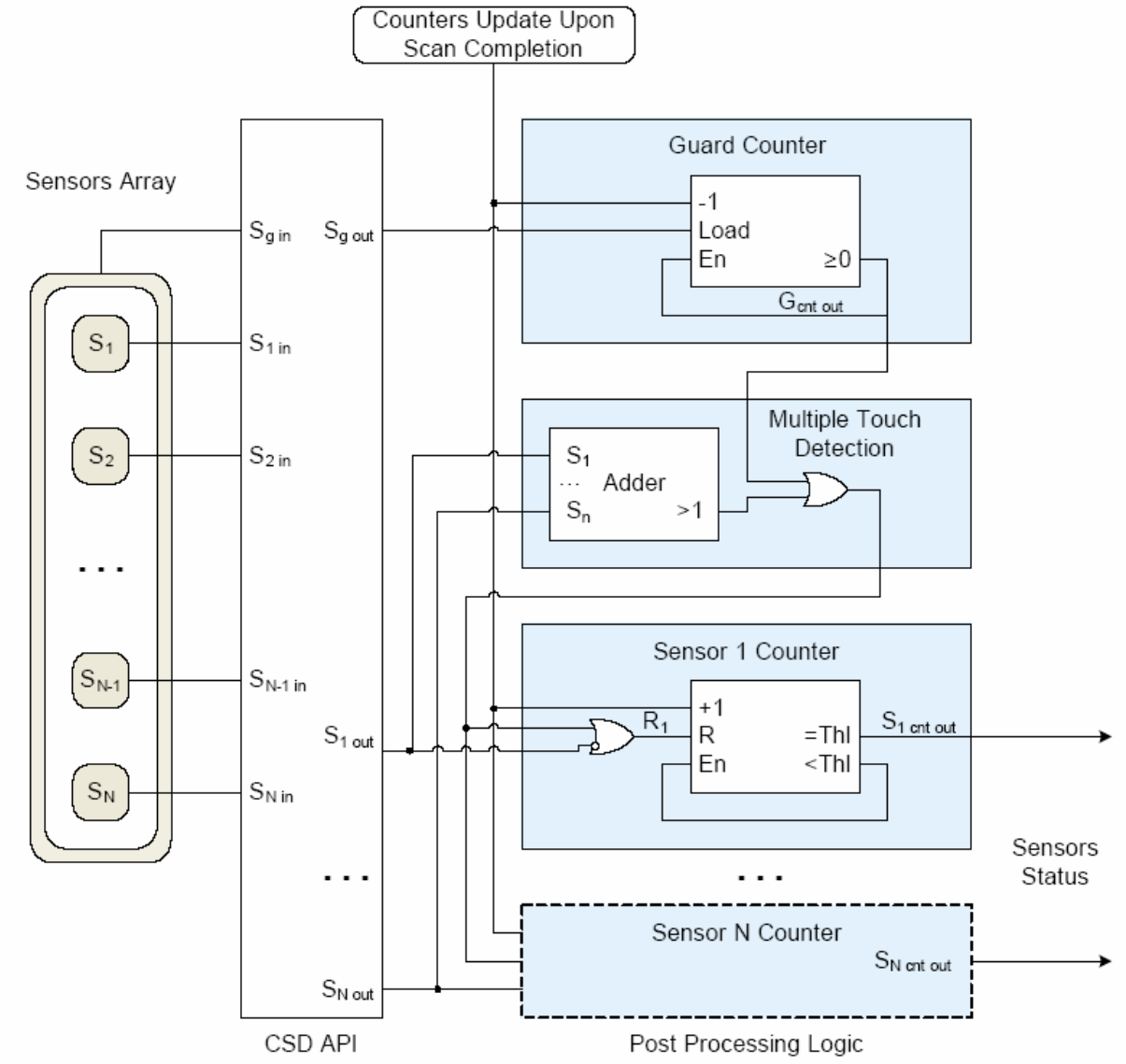
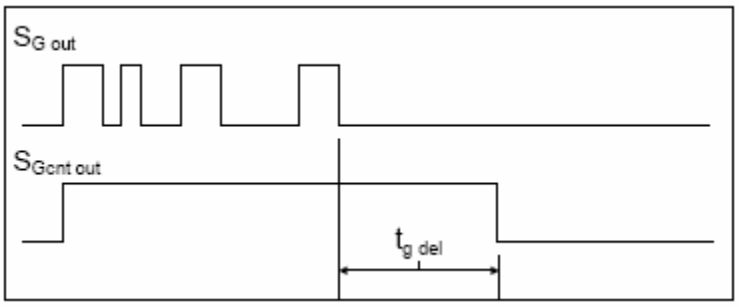


圖10. 後端處理邏輯的等效簡化線路圖

每次掃瞄周期完成後，計數器狀態就會進行更新。當防護感測器被觸發時，防護計數器就會載入其計數臨界值。防護計數器是以倒數的方式計數，當計數值倒數至零時，感測器的計數器就會再度啟動。當感測器的計數值到達某個臨界值時便會停止增加計數，而維持最大的計數值。到達最大計數值後，訊號感測器就會開始動作。圖11所示為計數器的時序圖。

防護計數器運作時序圖



感測器的計數器運作時序圖

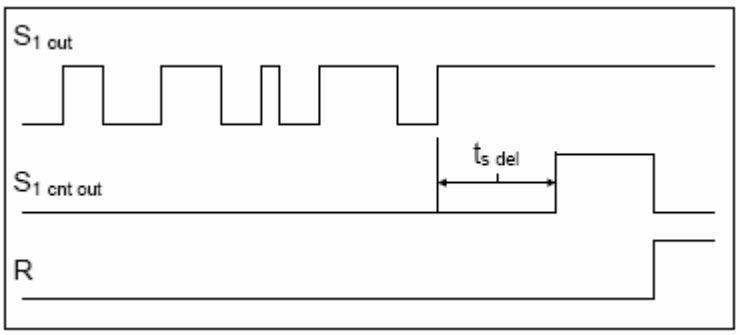


圖11. 防護與感測器的計數器之時序圖

$S_{g\ out}$ — 防護感測器偵測訊號。

$S_{Gcntout}$ — 防護計數器輸出訊號：「1」表示偵測到水滴；「0」為偵測到無水滴。

t_{g_del} — 最後一次偵測到水滴到啟動觸碰偵測決策邏輯間，最短的時間間隔。

S_{1out} — 第一個感測器觸碰偵測訊號。

$S_{1\ cnt\ out}$ — 第一個感測器的計數器輸出訊號。

$t_{s\ del}$ — 感測器必須被持續觸碰的最短時間間隔，這是為了讓決策邏輯得以判斷

觸碰動作的發生。

若是同時觸碰一個以上的感測器時，在API中還有另外一項機制可以追蹤多個感測器觸碰並且重置所有感測器的計數器，這樣就可以避免水流流過或將整個手掌放在感測區域上時可能發生的觸碰偵測誤判情形。此外，當您將數個感測板平行並列放置時，可能會發生較大的水滴同時覆蓋住多個感測器的情況。若您觸碰其中一個感測器，就也有可能同時觸發其他所有的感測器。此時這套多重觸碰偵測機制就可發揮作用，防止觸碰偵測誤判情形的發生。後端處理邏輯已完全列於表2中。

表2. 後端處理程式碼

```
//-------------------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------------------

#include <m8c.h>        // part-specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

#define SENSOR_TOUCH_COUNT_TH 15
#define GUARD_TOUCH_COUNT_TH 10
#define GUARD_SENS_NUM 3

WORD iI2CBuf[CSD_TotalSensorCount];
BYTE touch_cnt_array[CSD_TotalSensorCount];
BYTE touch_cnt = 0;
BYTE guard_state = 0;
BYTE i;

extern BYTE EzI2C_bRAM_RWcntr;

void main()
{


    M8C_EnableGInt;

    CSD_Start();
    CSD_SetDefaultFingerThresholds();
    CSD_baBtnFThreshold[GUARD_SENS_NUM] = 100;

    CSD_SetRefValue(0);
   CSD_InitializeBaselines();

    for (i = 0; i < CSD_TotalSensorCount; i++)
      touch_cnt_array[i] = 0;

    EzI2C_SetRamBuffer(2*CSD_TotalSensorCount, 0, (BYTE *)iI2CBuf );
    EzI2C_Start();

    while (1) {

       CSD_SetRefValue(1);
       CSD_ScanSensor(GUARD_SENS_NUM);

       CSD_SetRefValue(1);
       CSD_ScanSensor(0);
       CSD_ScanSensor(1);
       CSD_ScanSensor(2);

      CSD_UpdateAllBaselines();

      M8C_DisableGInt;
      if ((0 == EzI2C_bRAM_RWcntr) || (EzI2C_bRAM_RWcntr > (2*CSD_TotalSensorCount-1)))
        for (i = 0; i < CSD_TotalSensorCount; i++) iI2CBuf[i] = CSD_waSnsResult[i];
      M8C_EnableGInt;


      touch_cnt = 0;
      for (i = 0; i < CSD_TotalSensorCount; i++)
         if (GUARD_SENS_NUM != i)
         {
```

```
            if (0 != CSD_bIsSensorActive(i))
            {
               touch_cnt++;
               if (touch_cnt_array[i] < SENSOR_TOUCH_COUNT_TH) touch_cnt_array[i]++;
            }
            else
              touch_cnt_array[i] = 0;
          }

        if(0 != CSD_bIsSensorActive(GUARD_SENS_NUM))
           touch_cnt_array[GUARD_SENS_NUM] = GUARD_TOUCH_COUNT_TH;
        else
        {
          if (0 != touch_cnt_array[GUARD_SENS_NUM])
           touch_cnt_array[GUARD_SENS_NUM]--;
        }

        guard_state = (touch_cnt_array[GUARD_SENS_NUM] > 0);

        if ((0 != guard_state) || (touch_cnt > 1))
          for (i = 0; i < CSD_TotalSensorCount; i++)
            if (GUARD_SENS_NUM != i) touch_cnt_array[i] = 0;

         (0 != guard_state)?(PRT2DR |= 0x80):(PRT2DR &= ~0x80);
        (SENSOR_TOUCH_COUNT_TH == touch_cnt_array[0])?(PRT0DR |= 0x01):(PRT0DR &= ~0x01);
         (SENSOR_TOUCH_COUNT_TH == touch_cnt_array[1])?(PRT0DR |= 0x02):(PRT0DR &= ~0x02);
         (SENSOR_TOUCH_COUNT_TH == touch_cnt_array[2])?(PRT0DR |= 0x04):(PRT0DR &= ~0x04);

    }
}
```

設計建議

本篇應用說明亦可做為其他防水CapSense設計方案的基礎。根據我們的實驗法則，在此提出幾項可能對您有用的建議：

- 將感測器垂直或傾斜擺設，如此一來水滴自然就會從感測器電路板子流走，也不會有大型水滴出現。
- 利用防水性、不吸水的材質作為面板的絕緣層，如此可以減少裝置面板上的水痕與水膜的殘留。這些特性對於一些如海水等高度導電性的水特別有用。
- 防護感測器適用於有連續水流的環境，對於僅需應付下雨的裝置而言，就不是那麼必要。
- 按鍵之間的遮蔽電極至少要有10 mm的寬度，這樣才能更有效地抑制水滴造成的影響。

結論

本篇應用說明提出了一套防水電容式感測技術，可成功運用於潮濕、水膜附著、水滴、甚至連續水流的環境中。在有水滴的狀況下，裝置依然能夠持續運作，並且可消除水流通過感測區域時，可能造成的誤判觸發情形。

後端處理演算法也可用於其他任何需要可靠觸碰偵測之應用中，例如家電用品，因為在電器運作時，系統絕不能產生誤判觸動的情形。而軟體計數器機制在傳統CapSense應用中也可有效地避免同時觸碰多個感測器的情形，例如將手臂或手掌放在感測區上。

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone: 408-943-2600
Fax: 408-943-4730
http://www.cypress.comT