NEC

Application Note

Capacitive Touch

Using 78K0/Kx2-L 8-bit MCU

Document No. U19788EE1V0AN00
Date published May 2009

Printed in Germany

Legal Notes

- **The information in this document is current as of May, 2008. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**

- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

- The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime

systems, safety equipment and medical equipment (not specifically designed for life support).
"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives anddistributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and otherlegal issues may also vary from country to country.

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668, Japan
Tel: 044 4355111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554,
U.S.A.
Tel: 408 5886000
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211 65030
http://www.eu.necel.com/

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908 691133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01 30675800

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 6387200

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02 667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven,
The Netherlands
Tel: 040 2654010

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27
ZhiChunLu Haidian District,
Beijing 100083, P.R.China
Tel: 010 82351155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2511-2512, Bank of China
Tower,
200 Yincheng Road Central,
Pudong New Area,
Shanghai 200120, P.R. China
Tel: 021 58885400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886 9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R.O.C.
Tel: 02 27192377

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253 8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku, Seoul,
135-080, Korea Tel: 02-558-3737
http://www.kr.necel.com/

Table of Contents

Chapter 1 Introduction

This application note explains the solution used by NEC to implement projected capacitive touch sensing with the new 8-bit MCU family 78K0/Kx2-L using the onboard op-amp. More than one switch can be implemented by using a few low cost components.

There are many technologies used today to achieve touch sensing including resistive, infra red, surface acoustic wave, surface capacitive and projected capacitive. All these technologies have advantages and disadvantages, the decision to use one technology depends on the application requirements. The projected capacitive technology, discussed in this document, offers more advantages as the sensing element is solid with no moving parts and therefore provides a robust solution that can be used with different switches in all environments.

Capacitive touch sensing is becoming increasingly popular in the Human Machine Interface applications, where soft membrane and hard solid switches are being replaced by a more intuitive solution. Applications of the touch sensing are found every where in our lives including domestic appliances, consumer electronics, access control and in general it offers replacement for all switches and panels.

Chapter 2 Theory behind capacitive touch

2.1 Principle of capacitive touch

The principle of capacitive sensing is based on the simple fact that there is a mutual capacitance that builds up between two adjacent conductors separated by an insulator, the same way as the parallel plate capacitor, following the well known formula:

Equation 1

$$C = \frac{\varepsilon A}{d}$$

Where:
ε: Dielectric constant of the insulator between the electrodes.
A: Surface area of the electrodes.
d: Distance between the electrodes.

In free space every object has capacitance to ground and mutual capacitance to other conductive objects. Because of the conductivity of the human body, capacitance builds up whenever it approaches another conductive object given that there is an insulator layer between the two including air.

2.2 System Construction

To build a capacitive touch system the sensing element can be constructed from any conductive material behind an insulator. If the application requires transparency like the case of a sensor used on the top of a TFT display, the solution can use either optically transparent ITO (Indium Tin Oxide) or embedded micro fine wires. If the application requires no transparency then the solution can use the PCB tracks or conductive printed ink behind the panel. Therefore the sensing element can have any shape and size which contributes to the flexibility of the design.

For the general system performance care must be taken in the choice of the parameters influencing the capacitance cited by **equation 1**.

2.2.1 Surface

The surface area of the sensing element is directly proportional to the capacitance and therefore the bigger the surface area, the better the sensitivity achieved, This is true to a certain limit, however when the surface is much bigger than the surface area of the touching finger any excess surface has no effect.

2.2.2 Insulator

The dielectric constant of the insulating material is directly proportional to the capacitance and therefore using higher dielectric constant improves the sensitivity of the system. Glass is preferred but other insulator can be used with the expense of reduced sensitivity.

2.2.3 Thickness

The thickness of the insulator is inversely proportional to the capacitance therefore thicker overlays will reduce the capacitance.

2.2.4 Ground

The initial capacitance before the introduction of the human body effect depends on the switch's surrounding ground and this should not be left floating. A good ground is essential for providing a reliable system.

Chapter 3 NEC Electronics solution

Now that we know what capacitive touch system is and factors affecting its performance we can look at how this capacitance is measured. The basic technique used in this solution relies on measuring the charging time constant of the capacitor through a predefined resistor. The time constant increases when a finger approaches the sensing element. The new NEC 8-bit MCU family 78K0/Kx2-L are equipped with an onboard op-amp and this is used to create an oscillator with frequency depending on the capacitance of the sensing element.

3.1 Astable multivibrator

An astable multivibrator as shown in *Figure 3-1* is an oscillator whose low and high states are both unstable. The output of the oscillator toggles between the two states continuously, generating a train of pulses. This circuit is therefore also known as a pulse generator circuit and is widely used in electronics.

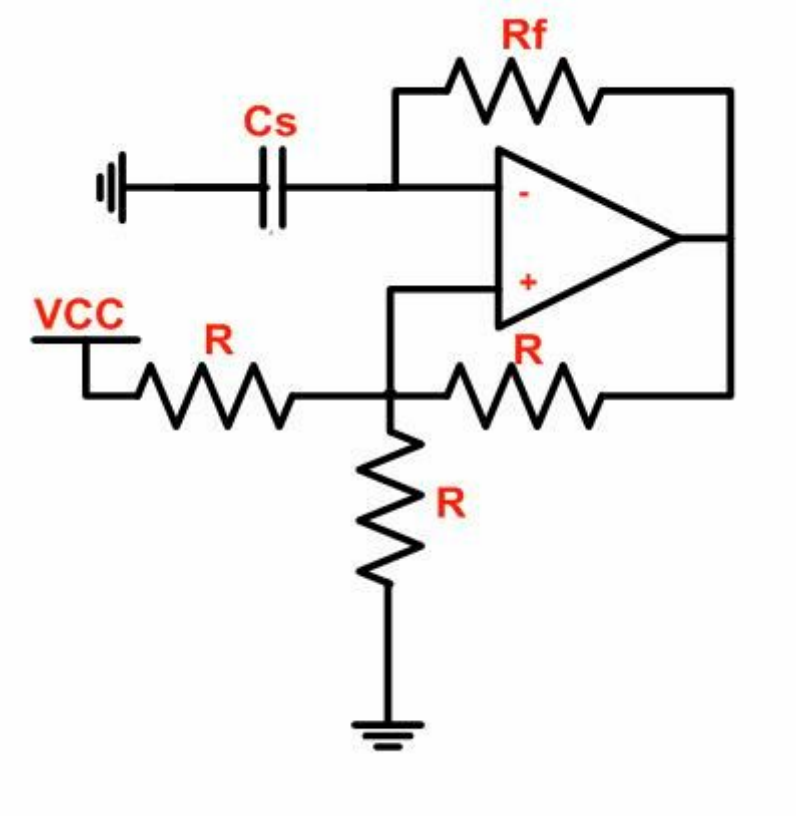


Figure 3-1 Astable multivibrator

The circuit uses both positive and negative feedback to achieve oscillation. When the op-amp output voltage is high the voltage at the positive input is equal to **2/3VCC**. The capacitor charges through **Rf** until the voltage is above **2/3VCC** which causes the output of the op-amp to swing to the other rail and consequently the positive input is held at **1/3VCC**. The capacitor discharges until it is below **1/3VCC** which causes the output of the op-amp to change and the whole cycle starts again. The operation is illustrated by the obtained signals in *Figure 3-3* and the frequency of oscillation is governed by the equation:

Equation 2

$$f = \frac{1}{2ln(2) * Rf * Cs}$$

CS: Switch capacitance

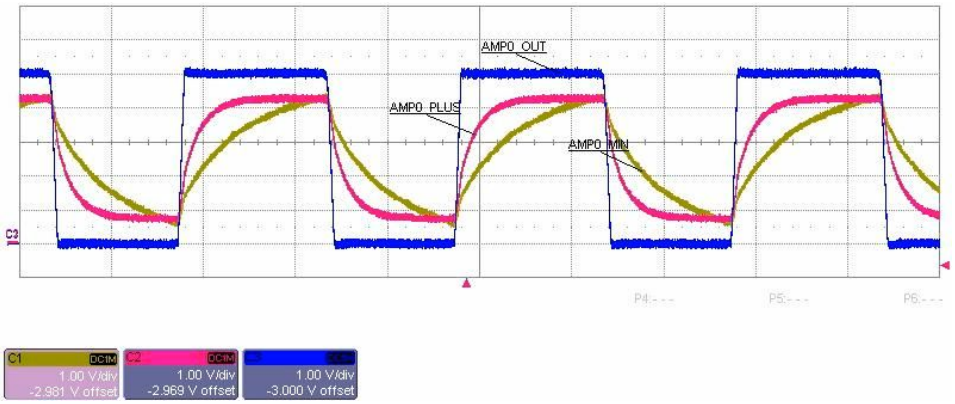


Figure 3-3 Obtained signals

AMP0 MIN: OP-AMP negative input
AMP0 PLUS: OP-AMP positive input
AMP0 OUT : OP-AMP output

3.2 Tuning the oscillator

When the switch is touched, the switch capacitance **Cs** will increase and as can be seen from Equation 2, the oscillator's frequency will decrease. The human body capacitance introduced in the system is usually just a few pF therefore the initial capacitance of the sensing element should be kept very low by tacking care when designing the tracks of the sensing elements.

To adjust the oscillation frequency the feedback resistor **Rf** can use a variable resistor in the development stage to achieve the required oscillation and once this value is established it can be used for the final product.

In order to achieve a low system response time it is necessary to have a high oscillation frequency, however this frequency is limited by the slew rate of the op-amp and should be kept lower than 100 KHz. If the frequency is set up to be very high the oscillator output voltage is attenuated and becomes more sinusoidal than a square wave.

As for the other resistors it is better to use higher values to achieve minimal power consumption.

3.3 Measuring the sensor capacitance

To detect whether a sensor is touched or not we need to measure the oscillation frequency. There are many ways to achieve this using a MCU. The oscillating frequency deviation is very small, therefore NEC Electronics' solution measures the accumulative effect of a touch on a number of periods. 78K0/Kx2-L has an internal 8 MHz high speed oscillator and it uses two sets of timers to keep track of the frequency. The 8-bit timer is configured as an external event counter and will count the number of periods while the internal 16-bit timer is configured as free running timer and provides the time measured in terms of CPU clocks as it is illustrated on *Figure 3-4*. When the sensor is touched, the oscillator's frequency is reduced and therefore the free running timer counts more ticks for the same amount of periods. The total number of CPU clocks is not really relevant as we are more interested in the relative change than the absolute value.

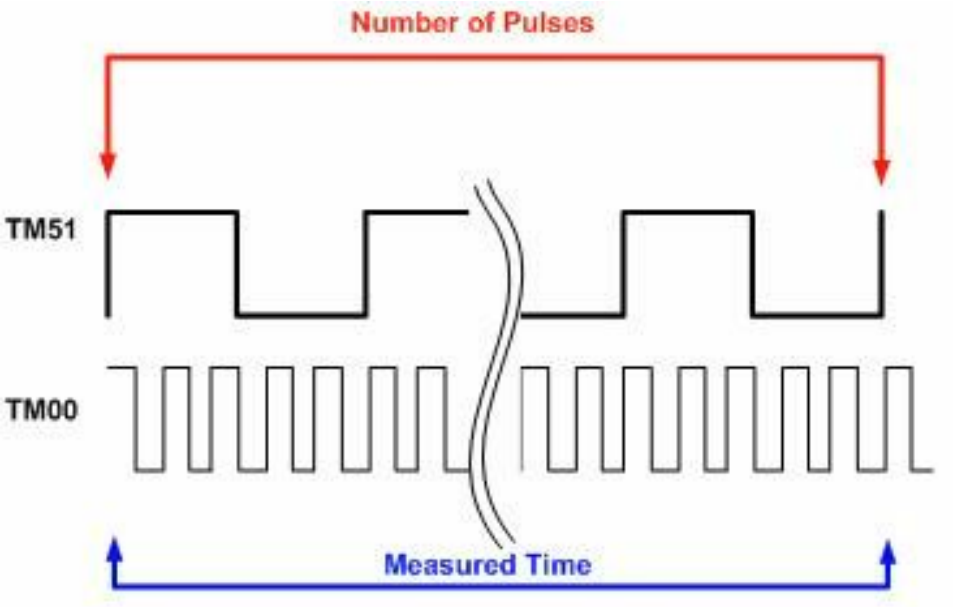


Figure 3-4 Measuring oscillator frequency

For illustration purposes the measured values for a sensor plotted as it can be seen in *Figure 3-5*. From this figure it can be seen that the touch affected the timer counts and the difference is obvious to when no touch is applied. Therefore the software needs to keep track of the non touched measurement and compare it to the new measured value. If the change is higher than the defined threshold a touch is registered.

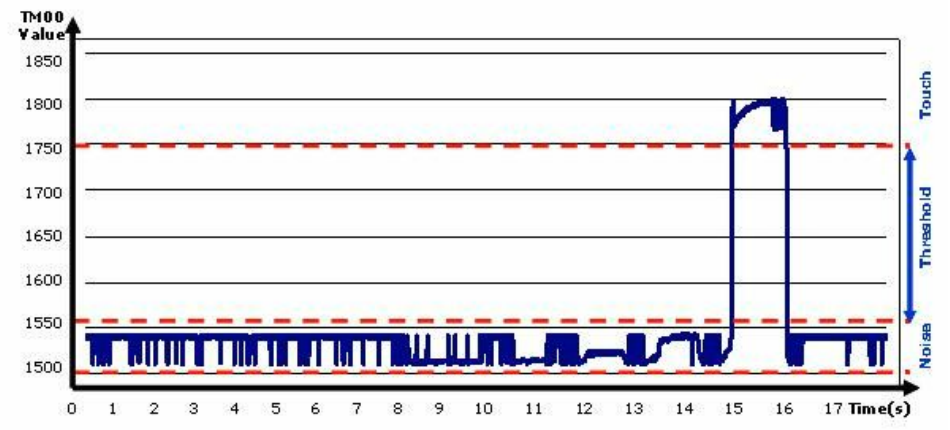


Figure 3-5 Touch effect on measured values

3.4 Sensitivity

By sensitivity we mean the differential change in the TM00 counts when a touch is applied. The aim is to achieve a bigger value compared to the change caused by the noise. The overall sensitivity of the system is affected by many factors like the sensor surface area, overlay thickness and material. These factors are all imposed on the designer and most of the time these are defined by the end application. From a design point of view the only factors left to change the sensitivity are the oscillator frequency and the TM00 main frequency. The higher the frequency of oscillation the more sensitive the system will be but this is again limited by the op-amp characteristics. The TM00 clock frequency should be set to the highest frequency possible to achieve maximum TM00 count value for the oscillation frequency set by the OPAMP oscillator. This is limited to maximum 10 MHz for the 78K0/Kx2-L MCU family. To be able to change the sensitivity the designer can change the number of the periods counted by TM51 by changing the value in the two software definitions shown below:
(See appendix for the full software listing)

#define TOUCH_SENSITIVITY 128 /* Number of pulses to be counted */

CR51 = TOUCH_SENSITIVITY; /* Compare value for TM51 */

When more periods are counted the effect of a touch is accumulated and therefore an adjustable sensitivity is achieved by software. However this comes with a draw back in terms of the response time so the more sensitive we try to make our system the longer time it will take to scan the sensors. The results obtained showed that 1mS per sensor was enough to achieve a good sensitivity through 4 mm of glass.

Chapter 4 Hardware implementation

4.1 Single sensor configuration

The following *Figure 4-1* shows typical connections when only one capacitive touch sensor '**CS**' is required. The resistor network forms an oscillator with the sensing element and the oscillating frequency is measured by means of capture and compare timer in conjunction with a free running timer. The snapshot of the code bellow shows how to set up the timers and the op-amp to initialize the oscillator and the timers.

```
#define START_TMR51() TCE51 = 1  /* Starts TM51 as external event counter*/
#define START_TMR16() TMC00 = 0x04  /* Starts Free running timer TMR00*/

/*  Timer 51 to count exetrnal intervals */
    PM3_bit.no0 = 1                /* Port P30 Mode */
    TCL51 = 0x01                   /* Rising Edge */
    CR51 = TOUCH_SENSITIVITY       /* Compare value for TM51 */

/* Timer00 TM00 Free-running timer mode */
     START_TMR16();

/*********************  Amplifier setup  ***********************/
    ADPC0 = 0x38;                 /* Set AMP0 Pins to analog*/
    PM2 = 0x07;                   /* Set P20 - P22 as I/P */
    AMP0M = 0x80;                 /* Enables operational amplifier 0
                                     (single AMP mode only) operation*/
    START_TMR51();
```

The single sensor configuration uses 4xI/Os, 1xOP-AMP, 1x8-Bit Timer and 1x16-Bit Timer of the MCU resources. Other devices of the same family have 2 onboard op-amps and therefore can implement 2 capacitive touch sensors using the same configuration.

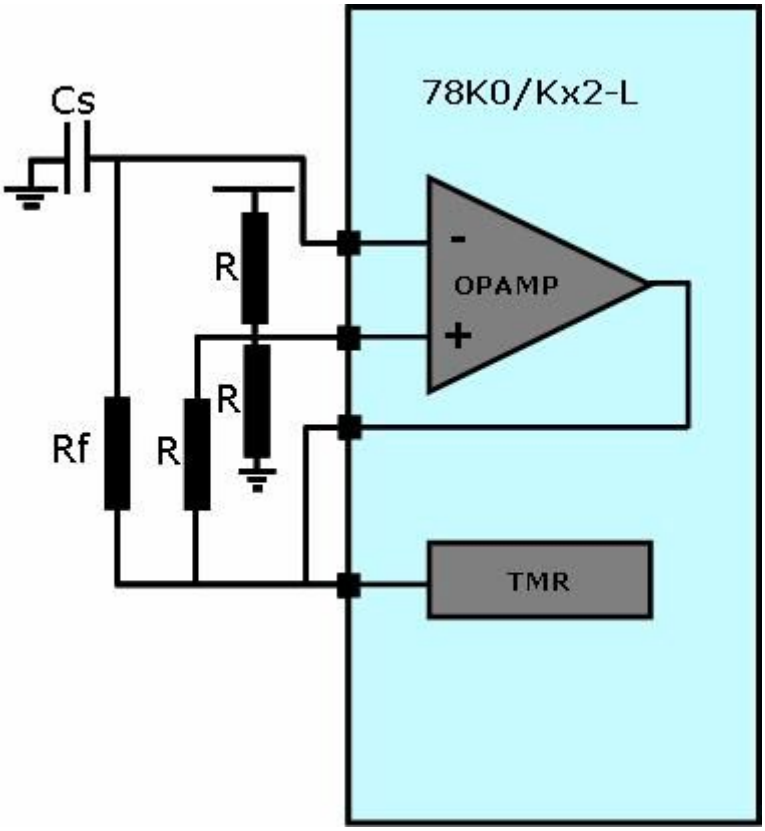


Figure 4-1 Single sensor connections

4.2 Multiple sensor configuration

When multiple sensors are required external multiplexers can be used to implement the additional sensors. The configuration is illustrated in *Figure 4-2*. The oscillator is set up with the same operation as for the single sensor configuration except this time the sensors are scanned one at a time. For illustration the set up used here was for 8 sensors therefore the MCU uses additional 3xI/Os to select the channel to be scanned. The multiplexer's enable signal can be tied to ground if only one is used but needs to be controlled by an I/O in case more than 1 multiplexer is used to disable the non used one. Depending on how many sensors are required bigger devices of the Kx2-L family can be used. The MCU uses port pins P23 to P25 to control the multiplexer channel and the following software is used to set up the I/O's to initialize the multiplexer port and select the sensor to be scanned.

```
#define TOTAL_SENSORS  8  /* Total sensor number used in the application*/

/*  Port Initialisation to control Multiplexer P23-24-25*/
        PM2   = 0x07;           /* Set P23 - P25 to O/P */

/************************************************************/
/* FUNCTION:  Select_Mux( UINT8 channel)                    */
/* PURPOSE:   Selects the channel to be scanned             */
/* PARAMETER: channel number                                */
/* RETURNS:   None                                          */
/************************************************************/
void Select_Mux( UINT8 channel)
{
         P2 = (channel & 0x07) << 1;          /* Only 8 sensors */
}
```

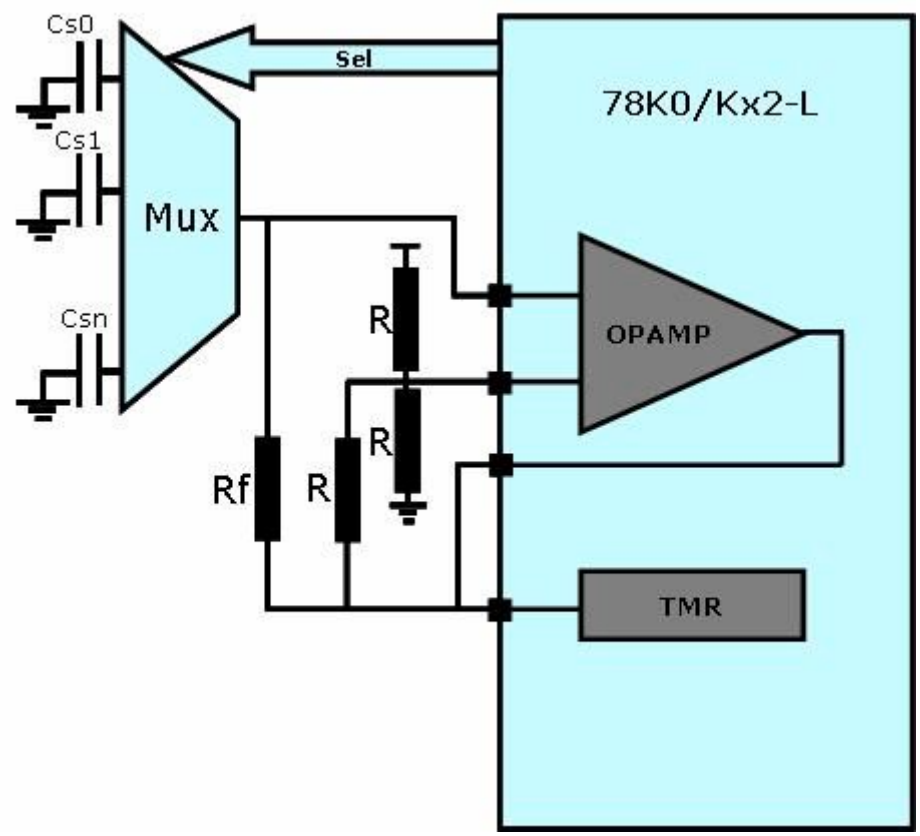


Figure 4-2 Multiple sensor connections

Chapter 5 Software implementation

5.1 Software overview

In General the software configures all the hardware to set up the ports to the required states, set up the timers, starts the oscillator and enters a while loop. The software then waits for the sensors to be scanned as it can be seen from the flowchart in *Figure 5-1*. During this time other tasks can be executed. Once the scanning is finished and the measured values are available, a touch detection process can take place depending on the predefined touch threshold to determine whether a touch was applied or not.

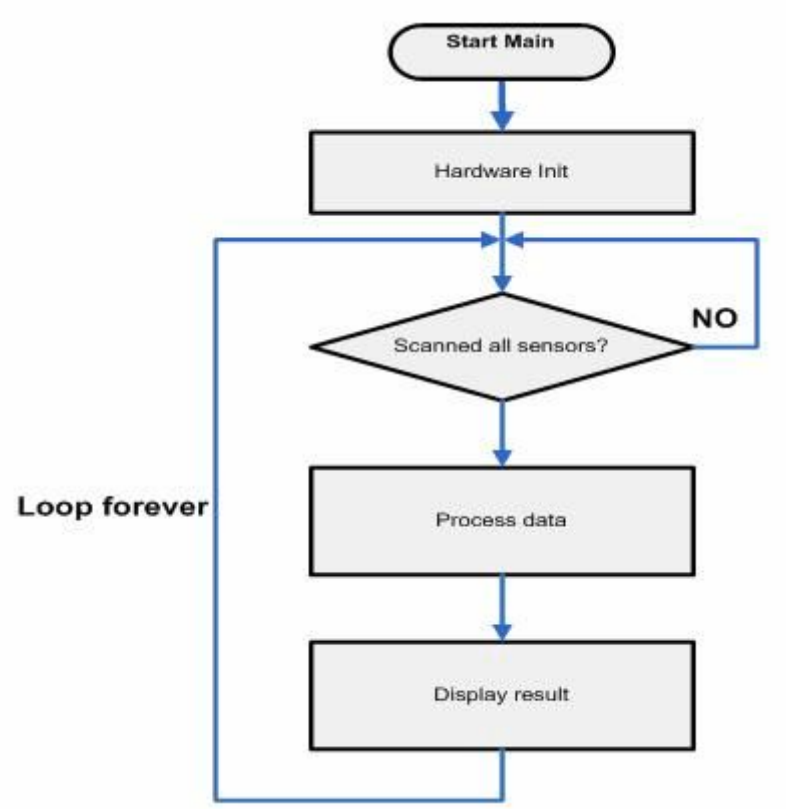


Figure 5-1 Flowchart "Main program overview"

5.2 Measuring the oscillator frequency

The measurement of the oscillation frequency is performed in the timer capture/ compare interrupt. Every time this interrupt is triggered the free running timer is read and the software points to the next sensor to be scanned. If all the sensors are scanned, the software signals the end of the scanning so that the main program can proceed with processing the measured values and decide if there was a genuine touch applied. The flowchart showing the operation is shown in

Figure 5-2. To reduce the noise effect only the 6 MSB bits of the measured value are taken into account.

```
/* Ignore least 2 significant bits to reduce noise */
     ChannelScoreBuff = (NewVal - OldVal) >> 2;
```

The buffering method is used here so that the main program can copy and then process the measured values while the buffer can be updated within the interrupt service routine.

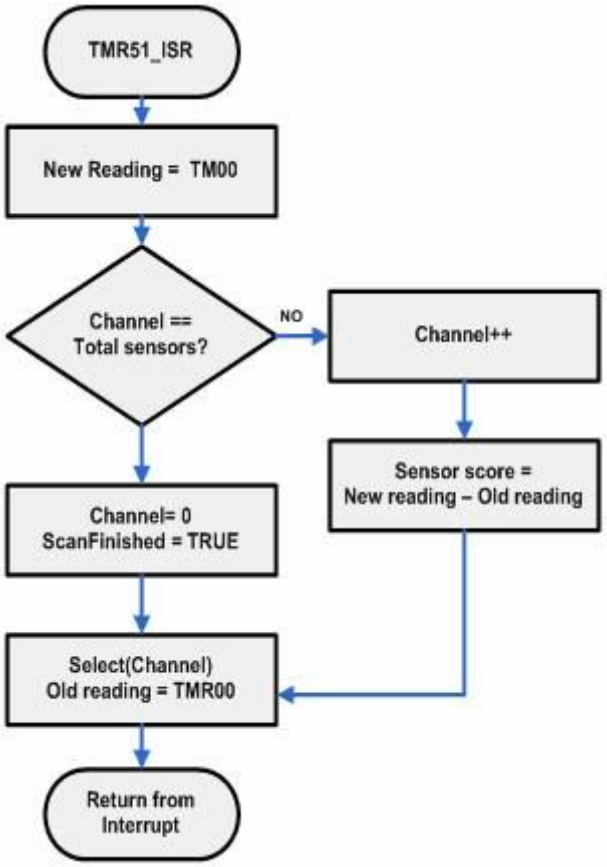


Figure 5-2 Flowchart "Interrupt service routine"

5.3 Processing measured values

Determining whether a touch has been applied or not based only on the threshold is not a trivial task. Capacitance value is not stable and its value changes with the environment especially with temperature and humidity. This makes keeping track of the untouched value a difficult task. The software provided implements a simple but powerful algorithm to track small changes in the untouched value. The software uses five variables for every sensor to store the current measured value, the previous value, the sensor's reference value and a variable to count how many cycles the measured value had a stable value.

```
typedef struct ChannelRec__
{
  UINT16 ChannelScore;
  UINT16 ChannelScoreBuff;
  UINT16 ChannelReference;
  UINT16 ChannelPreviousScore;
UINT16 ChannelValueTracker;

}ChannelRec, *ChannelPtr;
```

The flowchart in *Figure 5-3* shows how the measured value of every sensor is processed at the end of a complete scan. The reference value is taken in the very first scans and evaluated every cycle to adjust it to the external environment changes. The measured values are subjected to noise and fluctuates all the time therefore the software allows for a noise tolerance which can be adjusted depending on the environment where the solution is to be implemented.

```
#define NOISE_TOLERANCE    6       /* Tolerance value allowed for measured value */
#define TOUCH_THRESHOLD    250     /* Value change needed to consider sensor touched*/
#define TOUCH_MAXVAL       750     /* Maximum change allowed in sensor measured value */
#define TRACKING_VALUE     400     /* Number of cycles needed to adopt new reference value*/
```

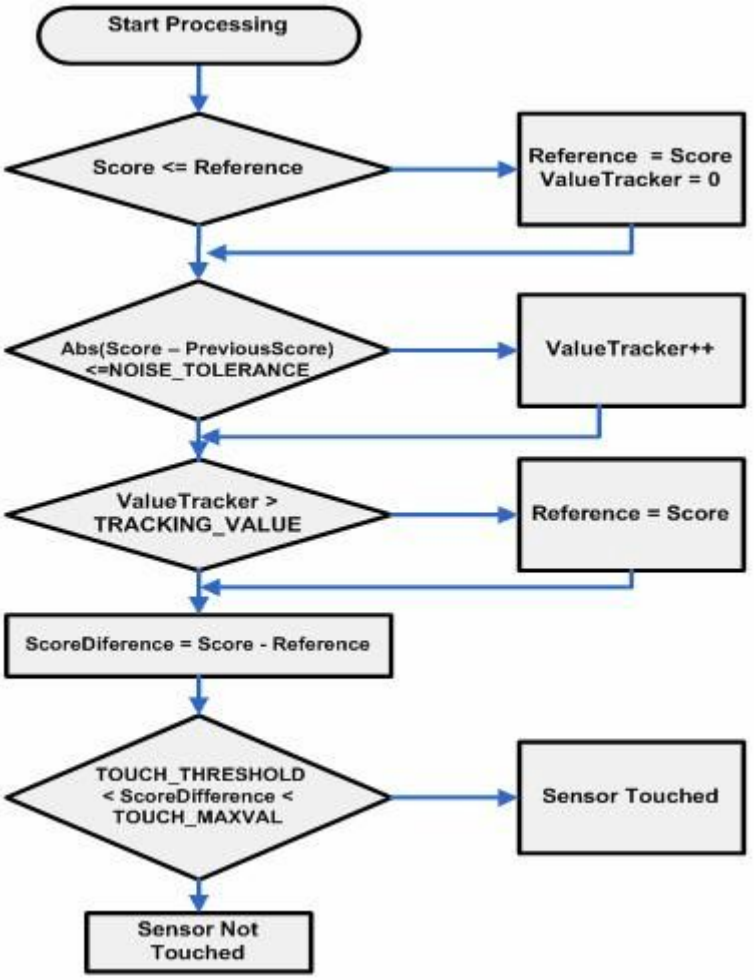


Figure 5-3 Flowchart "Processing measured values"

5.4 Improving reliability

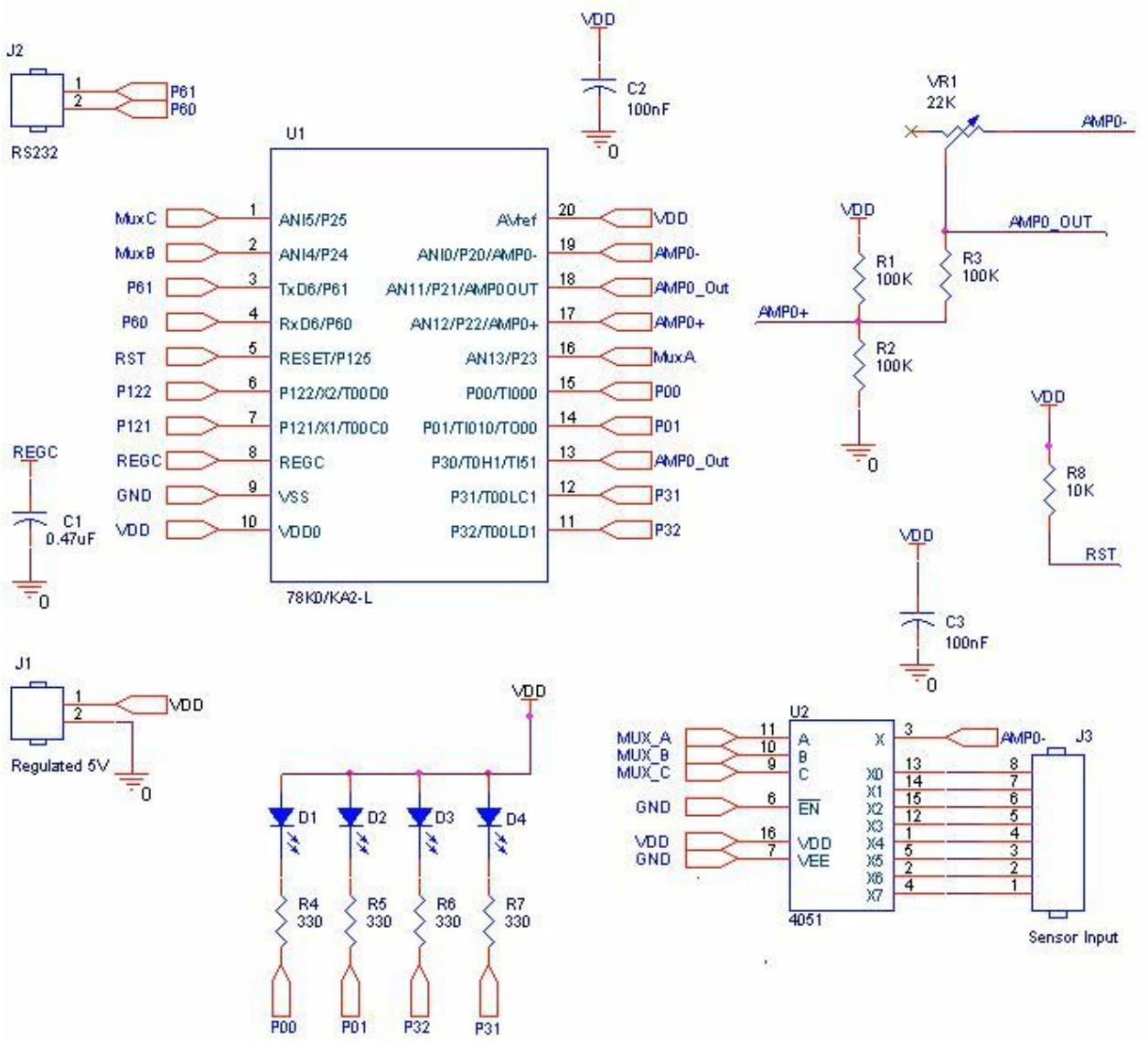
To improve reliability of the touch decision, the user can use de-bounce and timeout technique. The software will not register a touch until the touch exceeded the threshold for a minimum amount of time. The software defines this minimum time as number of cycles that the touch was registered for. In the case when the sensor has been touched for long time the software times out the touch and resets the reference value.

```
#define TOUCH_TIMEOUT       2500        /* Number of cycles needed before touch time out*/
#define TOUCH_BOUNCE           3        /* Number of cycles needed to debounce sensor*/
```

5.5 Conclusion

This application note shows how to use NEC Electronics MCUs with onboard OPAMP to implement capacitive touch sensing for one or more sensors. The software implements all the necessary functionalities to implement a self calibrated and reliable touch system.

Chapter 6 Appendix A: Circuit diagram



Chapter 7 Appendix B: Software program

```
/*****************************************************************************/
/* PROJECT = 78K0/Kx2-L Capacitive touch                                     */
/* MODULE = main.c                                                           */
/* DEVICE = 78K0/KA2-L (uPD78F0567)                                          */
/* VERSION = 1.0                                                             */
/* DATE = 14.03.2009                                                         */
/* LAST CHANGE = -                                                           */
/*****************************************************************************/
/* Description: Implementation of Capacitive Touch                           */
/*****************************************************************************/
/* By: NEC Electronics (Europe) GmbH                                         */
/*****************************************************************************/

/*****************************************************************************/
/* Warranty Disclaimer                                                       */
/*                                                                           */
/* Because the Product(s) is licensed free of charge, there is no warranty   */
/* of any kind whatsoever and expressly disclaimed and excluded by NEC,      */
/* either expressed or implied, including but not limited to those for       */
/* non-infringement of intellectual property, merchantability and/or         */
/* fitness for the particular purpose.                                       */
/* NEC shall not have any obligation to maintain, service or provide bug     */
/* fixes for the supplied Product(s) and/or the Application.                 */
/*                                                                           */
/* Each User is solely responsible for determining the appropriateness of   */
/* using the Product(s) and assumes all risks associated with its exercise   */
/* of rights under this Agreement, including, but not limited to the risks   */
/* and costs of program errors, compliance with applicable laws, damage to   */
/* or loss of data, programs or equipment, and unavailability or             */
/* interruption of operations.                                               */
/*                                                                           */
/* Limitation of Liability                                                   */
/*                                                                           */
/* In no event shall NEC be liable to the User for any incidental,           */
/* consequential, indirect, or punitive damage (including but not limited   */
/* to lost profits) regardless of whether such liability is based on breach */
/* of contract, tort, strict liability, breach of warranties, failure of     */
/* essential purpose or otherwise and even if advised of the possibility of */
/* such damages. NEC shall not be liable for any services or products        */
/* provided by third party vendors, developers or consultants identified or */
/* referred to the User by NEC in connection with the Product(s) and/or the */
/* Application.                                                              */
/*                                                                           */
/*****************************************************************************/
/* Environment:                                                              */
/*              Device:          µPD78F0567                                  */
/*              Target Hardware: ---                                         */
/*              IDE:             IAR Systems                                 */
/*                               Embedded Workbench for 78K V4.xx            */
/*                                                                           */
/*****************************************************************************/


/*****************************************************************************/
/*  Include files                                                            */
/*****************************************************************************/
#include     "io78f0567_20.h"
#include    "intrinsics.h"
#include             "typed.h"
#include              "stdlib.h"

/*****************************************************************************/
/* Option Bytes                                                              */
/*****************************************************************************/
#pragma location = "OPTBYTE"
__root const unsigned char opbytes[5]={0x6E,0x00,0x00,0x1E,0x02} ;
```

```
/*****************************************************************************/
/* Security ID CODE: OCD on-chip debugging (TK-78K0R + QB-MINI2 )             */
/*****************************************************************************/
#pragma location = "SECUID"
__root const unsigned char secuid[10]={0xff,0xff,0xff,0xff,0xff, 0xff,0xff,0xff,0xff,0xff};

/*****************************************************************************/
/* Program defines                                                           */
/*****************************************************************************/
/* Define display LED */
#define LED_One     P6_bit.no0
#define LED_Two     P6_bit.no1
#define LED_Three   P3_bit.no1
#define LED_Four    P3_bit.no2
/* Define Cpacitive touch constant*/
#define TOTAL_SENSORS       8          /* Total sensor number used in this application*/
#define NOISE_TOLERANCE     6          /* Tolerance value allowed for measured value */
#define TOUCH_THRESHOLD     130        /* Value change needed to consider sensor touched*/
#define TOUCH_MAXVAL        500        /* Maximum change allowed in sensor measured value */
#define TRACKING_VALUE      400        /* Number of cycles needed to adopt new reference value*/
#define TOUCH_TIMEOUT       2500       /* Number of cycles needed before touch time out*/
#define TOUCH_BOUNCE        3          /* Number of cycles needed to debounce sensor*/
#define INVALID_SENSOR      0xFF       /* Invalid sensor number*/
/* Define Sensor states*/
#define ON_STATE    1
#define OFF_STATE   0
/* Macros*/
#define START_TMR51() TCE51 = 1        /* Starts TM51 as external event counter*/
#define STOP_TMR51()  TCE51 = 0        /* Stops TMR51*/
#define START_TMR16() TMC00 = 0x04     /* Starts Free running timer TMR00*/
#define STOP_TMR16()  TMC00 = 0x00     /* Stops Free running timer TMR00*/


/*****************************************************************************/
/* Capacitive Touch Variables                                                */
/*****************************************************************************/
UINT16 OldVal;
UINT8  FreeRun = 5;
UINT8  ScanFinished = FALSE;

typedef struct ChannelRec__
{
  UINT16 ChannelScore;
  UINT16 ChannelScoreBuff;
  UINT16 ChannelReference;
  UINT16 ChannelPreviousScore;
  UINT16 ChannelValueTracker;
}ChannelRec, *ChannelPtr;

typedef struct SenChannelRec__
{
  UINT8 CurrentChannel;
  ChannelPtr Channel[TOTAL_SENSORS];

}SenChannelRec, *SenChannelPtr;

typedef struct SensorState__
{
  UINT16 ChannelStateTracker;
  UINT8  SensorTouchedIndex;
  UINT8  PreviousSensorTouched;
}SenStateRec, *SenStatePtr;


SenChannelPtr  SenChannel;
SenStatePtr SenState;

/*****************************************************************************/
/* Functions Prototypes                                                      */
/*****************************************************************************/
__interrupt void TMR51_ISR (void);
void SystemInit(void);
void Select_Mux( UINT8 channel);
```

```
void Touch_Process(void);
void Display_Result(UINT8 index);
void SensorResetScores(void);
UINT8 GetTouchedChannels(void);
void TrackSensorState(UINT8 ChannelsTouched );



/*****************************************************************************/
/* FUNCTION:  __low_level_init                                               */
/* PURPOSE:    Hardware initialision                                         */
/* PARAMETER: None                                                           */
/* RETURNS:  1                                                               */
/*****************************************************************************/
int __low_level_init(void)
{
      __disable_interrupt();

    /* device 768Bytes RAM and 16K ROM */
    IMS = 0x04;
    /* System initialization */
    SystemInit();

    __enable_interrupt();

    return(1);
}


/*****************************************************************************/
/* FUNCTION:  Select_Mux( UINT8 channel)                                     */
/* PURPOSE:   Selects the channel to be scanned                              */
/* PARAMETER: channel number                                                 */
/* RETURNS:   None                                                           */
/*****************************************************************************/
void Select_Mux( UINT8 channel)
{
  /* Only 8 sensors */
  P2 = (channel & 0x07)<<1;
}
/*****************************************************************************/
/* FUNCTION:   main                                                          */
/* PURPOSE:    main program                                                  */
/* PARAMETER:  None                                                          */
/* RETURNS:    None                                                          */
/*****************************************************************************/
void main(void)
{
  UINT8 i;

    while (1)
    {
        /* Wait until all sensors are scanned*/
        while(!ScanFinished);
        for(i=0; i<TOTAL_SENSORS;i++)
        {
          /* Load measured value from the buffer*/
         SenChannel->Channel[i]->ChannelScore = SenChannel->Channel[i]->ChannelScoreBuff;
        }
        /* Need to wait for next full scan to finish*/
        ScanFinished  = FALSE;
        /* Process the measured values*/
        Touch_Process();

   }
}

/*****************************************************************************/
/* FUNCTION:  Touch_Process                                                  */
/* PURPOSE:   process scores and decide if touch was applied                 */
/* PARAMETER: None                                                           */
/* RETURNS:   None                                                           */
/*****************************************************************************/
```

```
void Touch_Process()
{
  UINT8 i;
  UINT8 ChannelsTouched = 0;

  /* The first few cycles we just measure values but don't do any processing */
    if(FreeRun > 0)
    {
        FreeRun--;
        for(i=0; i<TOTAL_SENSORS; i++)
        {
            SensorResetScores();
        }
    }
    else
    {
        ChannelsTouched = GetTouchedChannels();
    }
    TrackSensorState(ChannelsTouched );
}

/*************************************************************************/
/* FUNCTION:   SensorResetScores(void)                                   */
/* PURPOSE:    resets the scores and references                          */
/* PARAMETER:  -                                                         */
/* RETURNS:    None                                                      */
/*************************************************************************/
void SensorResetScores(void)
{
  UINT8 i;
      for(i=0; i<TOTAL_SENSORS; i++)
      {
          SenChannel->Channel[i]->ChannelReference = SenChannel->Channel[i]->ChannelScore;
          SenChannel->Channel[i]->ChannelPreviousScore = SenChannel->Channel[i]->ChannelScore;
      }
}

/*************************************************************************/
/* FUNCTION:   GetTouchedChannels(void)                                  */
/* PURPOSE:    tracks the scores and returns nbr of touched channels     */
/* PARAMETER:  -                                                         */
/* RETURNS:    Number of channel's score above threshold                 */
/*************************************************************************/
UINT8 GetTouchedChannels(void)
{
  UINT8 i;
  UINT16 ScoreDifference;
  UINT8 ChannelsTouched = 0;
/* Keep track of reference and previous score values of each sensor*/
    for (i=0; i<TOTAL_SENSORS ; i++)
    {

        /* Sensor reference value tracking */
        if(SenChannel->Channel[i]->ChannelScore <= (SenChannel->Channel[i]->ChannelReference +
           NOISE_TOLERANCE))
        {
          SenChannel->Channel[i]->ChannelReference = SenChannel->Channel[i]->ChannelScore;
           SenChannel->Channel[i]->ChannelValueTracker = 0;
        }
        /* Sensor score value not changed from previous measurement*/
      if(abs(SenChannel->Channel[i]->ChannelScore - SenChannel->Channel[i]->ChannelPreviousScore)
          <=  NOISE_TOLERANCE)
        {
           SenChannel->Channel[i]->ChannelValueTracker++;
        }
        /* Score value was satble for at least the TRACKING_VALUE cycles */
         if(SenChannel->Channel[i]->ChannelValueTracker > TRACKING_VALUE)
        {
            SenChannel->Channel[i]->ChannelReference = SenChannel->Channel[i]->ChannelScore;
            SenChannel->Channel[i]->ChannelValueTracker = 0;
        }
        ScoreDifference =
        SenChannel->Channel[i]->ChannelScore - SenChannel->Channel[i]->ChannelReference;
```

```
        /* How many sensors score value exceeded the threshold? */
        if((ScoreDifference > TOUCH_THRESHOLD) && (ScoreDifference < TOUCH_MAXVAL))
        {
            SenState->SensorTouchedIndex = i;
            ChannelsTouched++;
        }
    }
    return ChannelsTouched;
}
/*************************************************************************/
/* FUNCTION:   TrackSensorState(UINT8 ChannelsTouched )                  */
/* PURPOSE:    Tracks sensor state ON/OFF states                         */
/* PARAMETER:  the SenChannelPtr                                         */
/* RETURNS:    None                                                      */
/*************************************************************************/
void TrackSensorState(UINT8 ChannelsTouched )
{
  UINT8 i;
  /* Sensor has been touched */
  if(ChannelsTouched == 1)
  {
      /* Same sensor is touched again! keep tracking*/
      if(SenState->PreviousSensorTouched == SenState->SensorTouchedIndex)
      {
           SenState->ChannelStateTracker++;
      }
      /* Same sensor is been touched for maximum time allowed -> Time out*/
      if(SenState->ChannelStateTracker > TOUCH_TIMEOUT)
      {
            SenState->ChannelStateTracker = 0;
            SenChannel->Channel[SenState->SensorTouchedIndex]->ChannelReference =
            SenChannel->Channel[SenState->SensorTouchedIndex]->ChannelScore;
       }
      /* Touch state needs to be at least for x cysles to remove bouncing*/
       if(SenState->ChannelStateTracker > TOUCH_BOUNCE)
       {
          /* Confirmed touch -> display result*/
          Display_Result(SenState->SensorTouchedIndex);
       }
      /* Register which sensor is touched*/
       SenState->PreviousSensorTouched = SenState->SensorTouchedIndex;

  }
  /* Not considered as touch if more than one sensor has high score */
  else
  {
      for(i=0; i<TOTAL_SENSORS; i++)
      {
           /* Register previous value for next scan*/
       SenChannel->Channel[i]->ChannelPreviousScore =
           SenChannel->Channel[i]->ChannelScore;
      }
      Display_Result(OFF_STATE);
      SenState->PreviousSensorTouched = INVALID_SENSOR;
      SenState->ChannelStateTracker = 0;
  }

}

/*************************************************************************/
/* FUNCTION:   Display_Result(UINT8 index)                               */
/* PURPOSE:    Switch the LEDs with binary code of the switch            */
/* PARAMETER:  LED number                                                */
/* RETURNS:    None                                                      */
/*************************************************************************/
void Display_Result(UINT8 index)
{
  LED_One   = 1;
  LED_Two   = 1;
  LED_Three = 1;
  LED_Four  = 1;

    switch(index)
```

```
    {
         case 0:
            LED_One   = 0;   LED_Two   = 1;    LED_Three = 1;  LED_Four  = 1;
            break;
         case 1:
            LED_One   = 1;   LED_Two   = 0;    LED_Three = 1;  LED_Four  = 1;
            break;
         case 2:
            LED_One   = 0;   LED_Two   = 0;   LED_Three = 1;   LED_Four  = 1;
            break;
         case 3:
            LED_One   = 1;   LED_Two   = 1;   LED_Three = 0;   LED_Four  = 1;
            break;
         case 4:
            LED_One   = 0;  LED_Two   = 1;    LED_Three = 0;   LED_Four  = 1;
            break;
         case 5:
            LED_One   = 1;  LED_Two   = 0;    LED_Three = 0;   LED_Four  = 1;
            break;
         case 6:
            LED_One   = 0;  LED_Two   = 0;    LED_Three = 0;   LED_Four  = 1;
            break;
         case 7:
            LED_One   = 1;  LED_Two   = 1;    LED_Three = 1;   LED_Four  = 0;
            break;
         default:
           break;
    }

}
/*****************************************************************************/
/* FUNCTION:   SystemInit                                                    */
/* PURPOSE:    Initialise hardware                                           */
/* PARAMETER:  None                                                          */
/* RETURNS:    None                                                          */
/*****************************************************************************/

void SystemInit(void)
{

    RSTMASK = 0x20;
    /*****************************************************************/
    /********************** Clock Setup  *****************************/
    /* Set fxh */
    OSCCTL = 0x00; /* X1, X2 as I/O port */
    MSTOP = 1;
    /* Set frh */
    RSTOP = 0;
    /* Set fprs */
    MCM = 0x00;/* fxp = frh,fprs = frh */
    /* Set fcpu */
    PCC = 0x00;
    /* Set frl */

    /*****************************************************************/
    /*****************************  Timers Setup   *******************/
    /*  Timer 51 to count exetrnal intervals */
    /* Port Mode */
    PM3_bit.no0 = 1;
    /* Rising Edge */
    TCL51 = 0x01;
    /* Compare value  256 pulses to count*/
    CR51 = 0xFF;

    /* Timer00 TM00 Free-running timer mode */
     START_TMR16();

    /*****************************************************************/
    /************************  Port6 Setup   *************************/

     ASICL6 = 0;
     PM6 = 0;
    /*****************************************************************/
```

```
    /*  Port Init conditions Setup */
    /* PM30 I/P PM31 & PM32 O/P */
    PM3 = 0xF9;
    P3 = 0;
    /*P0 all O/P */
    PM0 = 0x00;
    P0 = 0;
 /* Switch Off all LEDs*/
    LED_One   = 1;
    LED_Two   = 1;
    LED_Three   = 1;
    LED_Four   = 1;
 /*****************************************************************/
    /*  Capacitive touch Init conditions Setup */
   /*****************************************************************/
    /*********************  Amplifier setup  ***********************/
    /* Set Amp0 Pins to analog0x3F*/
    ADPC0 = 0x38;
    /* Set AMpo to 0; */
    PM2   = 0x07;
    /* Enables operational amplifier 0 (single AMP mode only) operation*/
    AMP0M = 0x80;
    START_TMR51();
    /* Enable interrupt TM51 */
    TMMK51 = 0;
    SenChannel->CurrentChannel = 0x00;
    SenState->SensorTouchedIndex = INVALID_SENSOR;
    SenState->PreviousSensorTouched =INVALID_SENSOR;
    SenState->ChannelStateTracker = 0;
    Select_Mux(0);
}


/*************************************************************************/
/* FUNCTION:  __interrupt void TMR51_ISR (void)                          */
/* PURPOSE:    TMR51 interrupt                                           */
/* PARAMETER:  None                                                      */
/* RETURNS:    None                                                      */
/*************************************************************************/

#pragma vector=INTTM51_vect
 __interrupt void TMR51_ISR (void)
{
 UINT16 NewVal;
    /* Save TMR00 value*/
    NewVal = TM00;
    /* Last Sensor been scanned*/
    if(SenChannel->CurrentChannel > TOTAL_SENSORS- 2)
    {
        /* We can now proceed with processing*/
        ScanFinished = TRUE;
        /* Point to the first sensor*/
        SenChannel->CurrentChannel = 0x00;
    }
    /* Not all sensors scanned yet*/
    else
    {
        /* Ignore least 2 significant bits to reduce noise */
       SenChannel->Channel[SenChannel->CurrentChannel]->ChannelScoreBuff = (NewVal - OldVal) >> 2;
        /* channel points to next sensor*/
        SenChannel->CurrentChannel++;
        /* Not all sensors scanned yet*/
        ScanFinished = FALSE;
    }
    /* Select next sensor to be scanned*/
    Select_Mux(SenChannel->CurrentChannel);
    /* Read TMR00 just before leaving Interrupt*/
    OldVal = TM00;

}
```