AN10675

Interfacing 4-wire and 5-wire resistive touchscreens to the LPC247x

Rev. 02 — 13 November 2008 Application note

Document information

Info	Content
Keywords	ARM, LPC247x, touchscreen
Abstract	This application note describes how to interface 4-wire and 5-wire touchscreens to the LPC247x series ARM MCUs. Reference schematics and source code are included.

founded by Philips

Revision history

Rev	Date	Description
02	20081113	Changed LPC2300 to LPC247x where applicable.
01	20080208	Initial version.

Contact information

For more information, please visit: **http://www.nxp.com**

For sales office addresses, please send an email to: **salesaddresses@nxp.com**

1. Introduction

1.1 About the LPC247x

The 16/32-bit LPC247x family is based on an ARM7TDMI-S core operating at up to 72 MHz together with a wide range of peripherals including multiple serial interfaces, programmable I/O port structures, 10-bit ADC and external bus options.

2. Four-wire touchscreen basics

A four-wire resistive touchscreen is a sensor consisting of two transparent resistive plates, ideally of uniform resistivity, normally separated by insulating spacers. The metalized contacts of the “x” layer run along the y-direction and thus the resistance is measured between the two x-direction ends. Similarly, the “y” layer has metalized contacts that run in the x-direction so that the resistance is measured along the y-axis (see Figure 1).

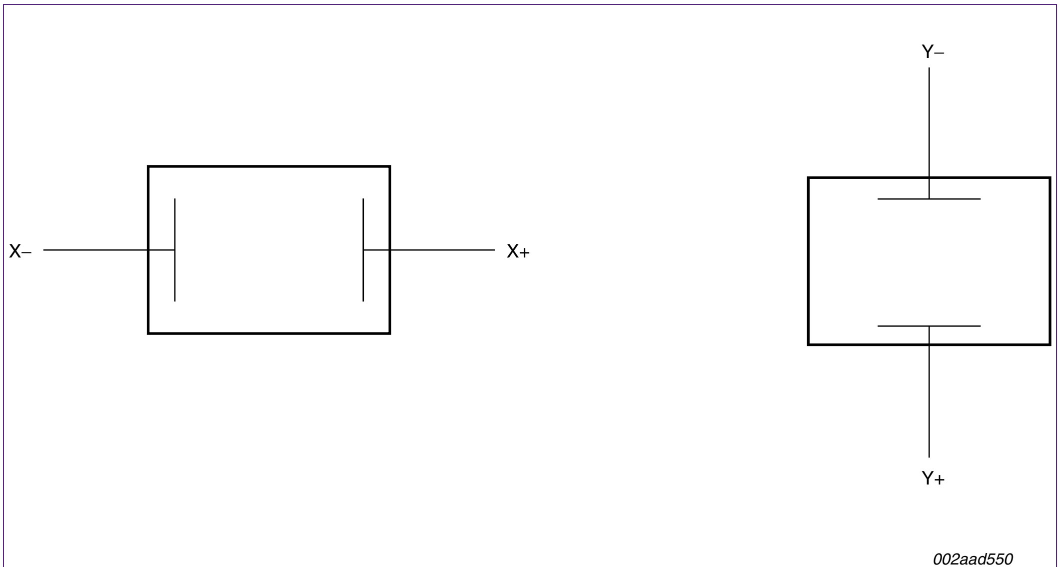

Fig 1. Structure of 4-wire resistive touchscreen

When touched with sufficient pressure, the top plate deforms making contact with the bottom plate. At the point of contact, the bottom layer effectively divides the top layer into two resistors in series, in a manner similar to the way the wiper on a potentiometer divides the potentiometer into two series resistors. Similarly, the bottom layer is effectively divided into two resistors at the point of contact with the top layer. Each plate is analogous to the two ends of a potentiometer where the other plate serves as the wiper. (see Figure 2). By proper biasing, each plate can function as a voltage divider where the output (wiper) voltage represents the rectangular coordinate of the point of contact.

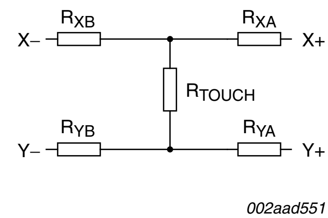

Fig 2. Resistive dividers formed by a touching a screen (4-wire)

Biasing the x-axis allows us to use the y-axis to measure the tap on the x-axis. In a similar manner, biasing the y-axis allows us to use the x-axis to measure the tap on the y-axis. (see Figure 3). Biasing both axis can be used to have the hardware detect when the screen has been touched and generate an interrupt (see Figure 4).

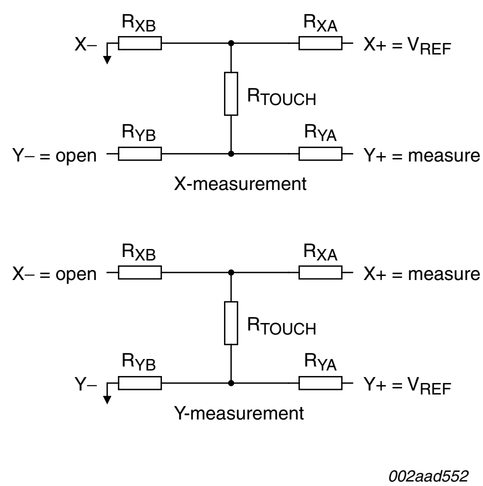

Fig 3. Touchscreen biasing for reading x and y directions (4-wire)

In addition to reading the x and y positions, its also possible to detect that a touch has occurred and use this condition to interrupt the CPU. In detecting a touch condition, the X+ signal from the screen is connected to a port pin programmed as an input with a high resistance pullup (see Figure 4). The Y- pin is connected to another port pin programmed as an output driving a logic zero. The remaining touch-screen pins, X - and Y+, are connected to port pins programmed to be inputs without pullups, effectively making these two pins opens.

When the screen is touched a voltage divider will exist between the internal pullup of the port pin and the resistance of the touch-screen (R_{XA} and R_{YB} in Figure 4). The resistance of the touchscreen is significantly less than the pullup connected to X+. When a touch occurs the voltage seen at the X+ pin will be close to zero. This will cause an interrupt.

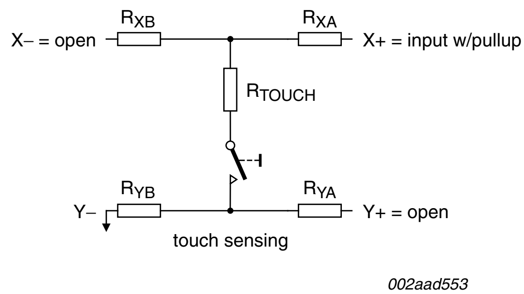

Fig 4. Touchscreen biasing for detecting a touch (4-wire)

Some NXP devices have a touchscreen controller included in the silicon. Such an interface offers advantages over the software approach. It automatically detects a touch, delays the settling times, measures both the x and y positions, and provides an interrupt when the specified number of A/D measurements have been completed. This reduces CPU overhead to a minimum and decreases software development time. The hardware touch screen controller increases silicon area which does increase chip cost.

3. Interfacing to the four-wire touchscreen

3.1 Biasing requirements

The resistance of each axis of a touchscreen is typically less than 1K. The datasheet for one particular display module, for example, lists the minimum x-direction resistance as 300 ohms and the maximum as 900 ohms. Similarly, the y-direction resistance is specified as a minimum of 200 ohms and a maximum of 650 ohms. It is prudent then to consider the effects of a microcontroller port pin's output resistance when interfacing to a touchscreen.

In the case of the LPC247x family, x-position measurements are made by driving the X+ signal with a logic one output of a port pin and the X- pin with the logic zero output of a different port pin. An A/D input, connected to the Y+ signal, is used to measure the voltage between the point of contact and V_{SS}. The Y- signal needs to be "open". This is accomplished by putting its respective port pin into input mode with no internal pullup or pulldown. Y- position measurements are made in a similar manner.

The biasing and measurement requirements for each of the four wires of the touchscreen are summarized in Table 1.

Table 1. Touchscreen interface requirements

	touchscreen signal			
function	**X+**	**Y+**	**X-**	**Y-**
hardware touch detection	digital input with pullup	open	open	V_{SS}
read x-position	voltage source	voltage measurement	Vss	open
read y-position	voltage measurement	voltage source	open	Vss

Both the X- and Y- pins have a requirement to be either open or connected to Vss, as shown in Table 1. A classic open drain output structure meets this requirement.

As shown in Table 1, both the X+ and Y+ pins have a requirement to be either a voltage source or a voltage measurement point. A voltage source can be achieved by having a pin function as an outport port with a sufficient current sourcing capability. Voltage measurement can be accomplished with an ADC. Thus the X+ and Y+ signals need to be connected to pins of the MCU that have both the current sourcing logic one and ADC capability. Additionally, the X+ signal needs to be connected to a pin that has a digital input with a moderately high pullup resistor if hardware touch detection is required.

3.2 I/O pin assignments

Based on these requirements, I/O pins were assigned to the touchscreen signals as shown in Table 2.

Table 2. Touchscreen pin assignments and modes

	touchscreen signal			
function	**X+ (P0.24/AD0.1)**	**Y+ (P0.25/AD0.2)**	**X- (P0.8)**	**Y- (P0.9)**
hardware touch detection	digital input with pullup	digital input with no pullup	input with no pullup	output logic zero
read x-position	output logic one	AD0.2	output logic zero	input with no pullup
read y-position	AD0.1	output logic one	input with no pullup	output logic zero

On the LPC247x family the port pin pullup and pulldown devices have a specification of 100 ohms maximum. The pullup device will impose an upper limit on the A/D readings while the pulldown device will impose a lower limit. In many applications this might not be significant. A 320 x 240 dot display module will need to have the 10-bit A/D readings scaled from 1024 counts down to 320 (or 240) counts so the loss of some A/D range is likely not an issue. Consider a display module with 240 dots in the vertical direction with a screen resistance of 200 ohms, minimum, in the y-direction. Each pin used for the y-direction measurement, Y+ and Y-, contributes 100 ohms in series with the y-direction of the touchscreen. Only half of the supply voltage appears across the touchscreen. With a 10-bit A/D, 512 counts will still be available to represent the 240 pixels in the y-direction.

In addition, when the touchscreen is mounted to a display module the four corner pixels of the display likely do not align with the endpoints of the touchscreen plates. Some calibration is needed to correlate the touchscreen measurements to the pixel position on the display.

The software used for this application note is shown in Section 8.1 “Source code for 4-wire touchscreen”

4. 4-wire software

The software for the 4-wire touchscreen consists of six primary functions: main, touch_detect, detected,read_ch_x, read_ch_y, and timer_delay.

As noted earlier, touchscreens require the deformation of the top plate in order to make contact with the bottom plate. Like a mechanical switch, they require debouncing. They also tend to be very susceptible to receiving noise. Oversampling and averaging of the position measurements seems in order. The overall rate after debounce and sampling needs to be sufficiently fast enough that the system's response to the user does not appear to be slow.

An on-board dot-matrix LCD module is used to display the x-position and y-position A/D readings when a touch condition has been detected. P0.11 and P0.10 are used to drive a two-color discrete LED to indicate that a touch condition exists.

4.1 Function: main

The main function performs the initialization of the dot-matrix LCD display, conditions the touch-screen pins to produce a Port 0 interrupt when a touch condition occurs, installs the interrupt vector for the interrupt handler, and enables the Port 0 interrupt.

4.2 Function: touch_detect

The touch detect function conditions the pins to sense a touch condition. After conditioning the pins this function waits for a prescribed settling time. In detecting a touch condition, the X+ signal from the screen is connected to a port pin programmed as an input with a high resistance pullup (See figure D). The Y- pin is connected to another port pin programmed as an output driving a logic zero. The remaining touchscreen pins, X - and Y+, are connected to port pins programmed to be inputs without pullups, effectively making these two pins opens.

When the screen is touched a voltage divider will exist between the internal pullup of the port pin and the resistance of the touch-screen (R_{XA} and R_{YB} in Fig D). The resistance of the touchscreen is significantly less than the pullup connected to X+. When a touch occurs the voltage seen at the X+ pin will be close to zero. This will cause an Port 0 interrupt.

In addition to conditioning the pins so that an interrupt can occur, this routine also returns a true condition if touch condition occurs and a false if there is no touch.

4.3 Function: detected

This function is the interrupt service routine for the touch detection. The function first waits for a prescribed debounce time and then verifies that a touch conditions exists, calls functions to read the x and y positions of the touch. The function then averages the readings stored in the x_values and y-values buffers. If a touch condition stills exists, this function displays the position information on the dot-matrix LCD. This function will continue to read the x and y positions as long as a touch condition occurs. When the screen is no longer being touched, the interrupt status will be cleared and the function exited.

4.4 Function: read_ch_x

The read_ch_x function conditions the pins to read the x-position using the ADC (See Fig C). The X+ signal is connected to a port pin programmed as a logic high output. The X- signal is connected to a port pin programmed as a logic low output. The point of contact during a touch condition forms a voltage divider between the X+ and X- signals. The Y- signal is connected to a port pin programmed as an input without a pullup, effectively making these pin an open. The Y+ signal is connected an ADC input, allowing the x-position voltage to be read.

After conditioning the pins this function waits for a prescribed settling time. The function then reads the ADC for a prescribed number of samples storing the samples in the x_values buffer.

4.5 Function: read_ch_y

The read_ch_y function conditions the pins to read the y-position using the ADC (See Fig C). The Y+ signal is connected to a port pin programmed as a logic high output. The Y- signal is connected to a port pin programmed as a logic low output. The point of contact during a touch condition forms a voltage divider between the Y+ and Y- signals. The X- signal is connected to a port pin programmed as an input without a pullup, effectively making these pin an open. The X+ signal is connected an ADC input, allowing the y-position voltage to be read.

After conditioning the pins this function waits for a prescribed settling time. The function then reads the ADC for a prescribed number of samples storing the samples in the y_values buffer.

4.6 Function: timer_delay

The timer_delay function users Timer 0 to delay a specified number of clock cycles. It uses the match register and match interrupt flag to achieve this.

5. Five-wire touchscreen basics

Like a four-wire touchscreen, a five-wire resistive touchscreen also consists of two transparent resistive plates separated by insulating spacers. The top plate contains a metalized contact and serves as the voltage sensing node. The four corners of the bottom plate are used to produce voltage gradients in the x and y directions. (see Figure 5). A specific bias configuration is used to obtain the x-direction measurement and a different bias configuration is used to obtain the y-direction measurement.

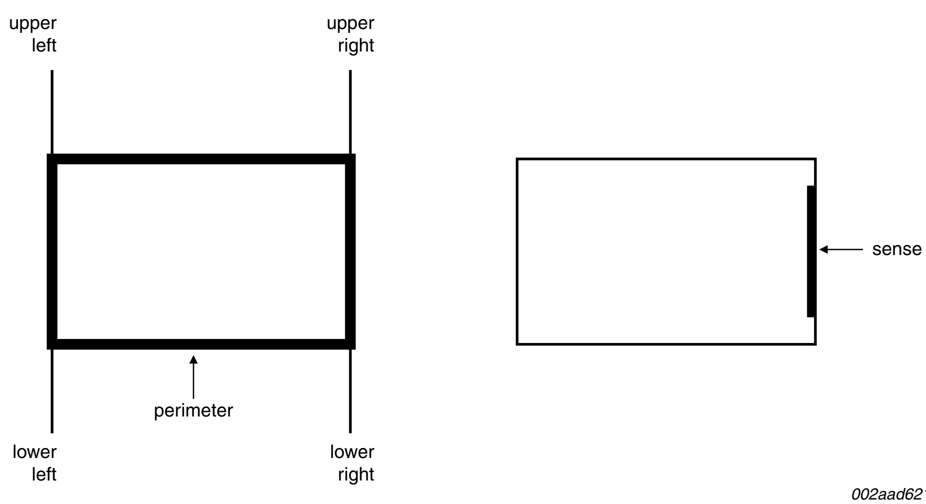

Fig 5. Structure of 5-wire resistive touchscreen

When touched with sufficient pressure, the top plate deforms making contact with the bottom plate. The model of the 5-wire touchscreen resistance can be complex. Circuitry added along the perimeters of the touchscreen by its manufacturer allows the user to treat the touchscreen as a voltage divider in each direction at the point of contact, provided the correct biasing is used.

Biasing the upper left and right corners to Vss and biasing the lower corners to Vdd allows us to measure the y-coordinate. Biasing the left side of the screen to Vss and biasing the right side of the screen to Vdd allows us to measure the x-coordinate. Biasing all four corners to Vss both can be used to detect when the screen has been touched and generate an interrupt (see Figure 6).

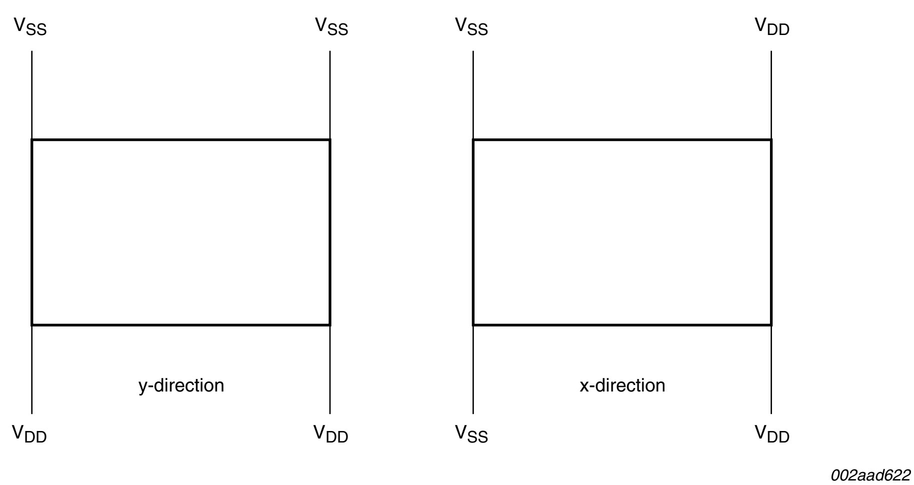

Fig 6. Touchscreen biasing for reading x and y directions (5-wire)

In detecting a touch condition, the sense signal from the screen is connected to a port pin programmed as an input with a high resistance pullup (see Figure 7). All corners of the touchscreen are driven to a logic zero.

When the screen is touched a voltage divider will exist between the internal pullup of the port pin and the resistance of the touch-screen. The resistance of the touch-screen is significantly less than the pullup connected to the sense signal. When a touch occurs the voltage seen at the sense signal pin will be close to zero. This will cause an interrupt.

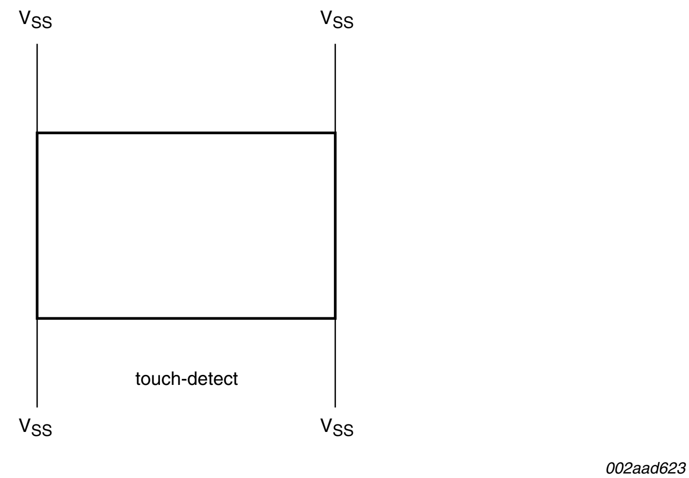

Fig 7. Touchscreen biasing for detecting a touch (5-wire)

6. Interfacing to the five-wire touchscreen

6.1 Biasing requirements

In the case of the LPC247x family, the corners of the screen are driven with a logic one output of a port pin when biased to VDD and with the logic zero output of the port pin when biased to VSS. An A/D input, connected to the sense signal, is used to measure the voltage of the x and y coordinates.

The biasing and measurement requirements for each of the four wires of the touchscreen are summarized in Table 3.

Table 3. Touchscreen interface requirements

	touchscreen signal				
function	**upper left**	**lower left**	**upper right**	**lower right**	**sense**
hardware touch detection	Vss	Vss	Vss	Vss	logic zero interrupt
read x-position	Vss	Vss	V_{DD}	V_{DD}	voltage measurement
read y-position	Vss	V_{DD}	Vss	V_{DD}	voltage measurement

The resistance between corners of a 5-wire touchscreen is about 100 ohms. It is prudent then to consider the effects of a microcontroller port pin's output resistance when interfacing to a touchscreen. Note that in all cases, the upper left corner of the screen is biased at Vss. This pin should not be hardwired to Vss. Its important that this pin be driven from an MCU output pin in order to balance the effects of the microcontroller port pin's output resistance on the other touchscreen pins.

As shown in Table 3, the corner pins have a requirement to be either a voltage source or sink current. A voltage source can be achieved by having a pin function as an outport port with a sufficient current sourcing capability. A standard I/O pin meets this requirement.

Voltage measurement can be accomplished with an ADC. Additionally, the sense signal needs to be connected to a pin that has a digital input with a moderately high pullup resistor if hardware touch detection is required.

6.2 I/O pin assignments

Based on these requirements, I/O pins were assigned to the touchscreen signals as shown in Table 4.

Table 4. Touchscreen pin assignments and modes

	touchscreen signal				
function	**sense (P0.24/AD0.1)**	**upper left (P0.26)**	**lower left (P0.9)**	**upper right (P0.8)**	**lower right (P0.25/AD0.2)**
hardware touch detection	digital input with pullup	output logic zero	output logic zero	output logic zero	output logic zero
read x-position	AD0.1	output logic zero	output logic zero	output logic one	output logic one
read y-position	AD0.1	output logic zero	output logic one	output logic zero	output logic one

On the LPC247x family the port pin pullup and pulldown devices have a specification of 100 ohms maximum. The pullup device will impose an upper limit on the A/D readings while the pulldown device will impose a lower limit. In many applications this might not be significant.

In addition, when the touchscreen is mounted to a display module the four corner pixels of the display likely do not align with the endpoints of the touchscreen plates. Some calibration is needed to correlate the touchscreen measurements to the pixel position on the display.

7. 5-wire software

The software for the 5-wire touchscreen is very similar to that used in the 4-wire software. The differences are in the functions that configure the ports for reading each direction, and a single ADC input is used for all measurements.

The software used for this application note is shown in Section 8.2 “Source code for 5-wire touchscreen”

8. Appendix

8.1 Source code for 4-wire touchscreen

```
/*****************************************************************************/
/*                              header files                                 */
/*****************************************************************************/

#include "LPC23xx.H"                /* LPC23xx definitions                   */
#include "LCD.h"                    /* Graphic LCD function prototypes       */

/*****************************************************************************/
/*                            pin definitions                                */
/*****************************************************************************/

#define X_plus  0x01000000              // X+ on P0.24
#define X_plus_mask0x00030000           // X+ pin select mask (ADC0.1)
#define X_plus_no_pull0x00020000        // X+ no pullup value
#define ADC_on_X 0x00010000             // X+ pin select (1) ADC

#define X_minus 0x00000100              // X- on P0.8
#define X_minus_mask0x00030000          // X- pin select mask
#define X_minus_no_pull0x00020000       // X- no pullup value

#define Y_plus  0x02000000              // Y+ on P0.25
#define Y_plus_mask0x000C0000           // Y+ pin select mask (ADC0.2)
#define Y_plus_no_pull0x00080000        // Y+ no pullup value
#define ADC_on_Y0x00040000              // Y+ pin select (1) ADC

#define Y_minus 0x00000200              // Y- on P0.9
#define Y_minus_mask0x000C0000          // Y- pin select mask
#define Y_minus_no_pull0x00080000       // Y- no pullup value


/*****************************************************************************/
/*                        timer count definitions       */
/*****************************************************************************/

#definedebounce 1000                    // debounce delay
#definesettling 100                     // settling time delay


/*****************************************************************************/
/*                        function prototypes                                */
/*****************************************************************************/

extern unsigned long install_irq( unsigned long IntNumber, void *HandlerAddr, unsigned
     long Priority );

void config_pins_x (void);
```

```
void config_pins_y (void);
void config_pins_touch (void);
void detected(void)__irq;
void display_lcd(short x_value, short y_value);
char hex_to_ascii(char ch);
void led_green (void);
void led_red (void);
void read_ch_x (void);
void read_ch_y (void);
void timer_delay (unsigned int count);
unsigned int touch_detect (void);


/******************************************************************************/
/*                                  globals                                   */
/******************************************************************************/

#definenum_samples 16                    // number of A/D samples per axis

unsigned int x_values[num_samples];      // array to store x_samples
unsigned int y_values[num_samples];      // array to store y_samples


/******************************************************************************/
/*                                  start of main code      */
/******************************************************************************/

int main (void)
{
     int i, j;

     PCONP |= (1 << 12);                     // Enable power to AD block
     lcd_init();                             // init the LCD display
     lcd_clear();                            // clear the LCD display
     for (i = 0; i < 20000000; i++);         // Wait for initial display
     IODIR0 |= 0x00000C00;                   // config touch LED pins as outputs
     led_green();                            // make the LED green

     PINMODE4 &= ~(0xFFFF);                  // P2[7:0] pullups
     PINSEL4 &= ~(0xFFFF);                   // P2[7:0] are GPIO
     FIO2DIR0 = 0xFF;                        // P2[7:0] are outputs
     FIO2MASK0 = 0x00;                       // P2[7:0] enabled for fast I/O

     touch_detect();                         // setup for touch detection
     install_irq(17, (void*)detected, 1);    // setup interrupt vector
     IO0_INT_EN_F = X_plus;                  // enable falling edge X-plus interrupt

     j = 0;
     while (1)                              // Loop forever until interrupt
          {
               for (i = 0; i < 200000; i++);     // delay
```

```
                FIO2PIN0 = (j & 0xFF);          // output the count to LEDs
                j++;                            // increment the count
            }
}

void detected(void)__irq
{
        short x_value, y_value, i;

        timer_delay (debounce);                 // debounce the touch
        while ((touch_detect()))                // loop as long as screen is touched
            {
                led_red();
                read_ch_x();                    // read and collect the x values
                read_ch_y();                    // read and collect the y values

                x_value = 0;                    // initial value
                for (i=0; i < num_samples; i++)
                    {
                        x_value += x_values[i];     // add up the conversion results
                    }
                x_value = x_value /num_samples;     // get average

                y_value = 0;                        // initial value
                for (i=0; i < num_samples; i++)
                    {
                        y_value += y_values[i];     // add up conversion results
                    }
                y_value = y_value /num_samples;     // get average

                if (touch_detect())display_lcd(x_value, y_value); // display values if
                                                                  // still have a touch
            }
            IO0_INT_CLR = X_plus;                   // clear X-plus interrupt
            led_green();
            VICVectAddr = 0;                        // Acknowledge Interrupt
}

void read_ch_x (void)
{
            unsigned int i;

            config_pins_x();                    // configure pins for read x-dir
            timer_delay (settling);             // settling time for switching
            AD0CR = 0x00200304;                 // Power up, PCLK/4, sel AD0.2

            for (i=0; i < num_samples; i++)
                {
                    AD0CR     |= 0x01000000;        // Start A/D conversion
                    while (AD0DR2 & 0x80000000);    // wait conversion completed
                    x_values[i] = ((AD0DR2 >> 6) & 0x3FF);  // store result
```

```
                }
}

 void read_ch_y (void)

{
          unsigned int i;

          config_pins_y ();                    // configure pins for read y-dir
          timer_delay (settling);              // settling time for switching
          AD0CR       = 0x00200302;            // Power up, PCLK/4, sel AD0.1

          for (i=0; i < num_samples; i++)
                {
                     AD0CR     |= 0x01000000;              // Start A/D conversion
                     while (AD0DR1 & 0x80000000);          // wait until completed
                     y_values[i] = ((AD0DR1 >> 6) & 0x3FF); // store result
                }
}

  unsigned int touch_detect (void)
{
          config_pins_touch ();           // configure pins for touch detection
          timer_delay (settling);         // settling time for switching
          return((IOPIN0 & X_plus)^X_plus); // return true if touch is detected
}

void config_pins_x (void)
{
          PINSEL0    &=   ~(X_minus_mask);          // X- is digital I/O
          PINMODE0&= ~(X_minus_mask);
          PINMODE0|= X_minus_no_pull;               // no pullup on X-
          IODIR0     |=   X_minus;                  // X- is an output
          IOCLR0     |=   X_minus;                  // make X- low

          PINSEL0    &=   ~(Y_minus_mask);          // Y- is digital I/O
          PINMODE0&= ~(Y_minus_mask);
          PINMODE0|= Y_minus_no_pull;               // no pullup on Y-
          IODIR0     &=   ~(Y_minus);               // Y- is an input

          PINSEL1    &=   ~(X_plus_mask);           // X+ is digital I/O
          PINMODE1&= ~(X_minus_mask);
          PINMODE1|= X_plus_no_pull;                // no pullup on X+
          IODIR0     |=   X_plus;                   // X+ is an output
          IOSET0     |=   X_plus;                   // make X+ high

          PINSEL1    &=   ~(Y_plus_mask);
          PINSEL1    |=   ADC_on_Y;                 // Y+ is an ADC pin
}

 void config_pins_y (void)
```

```
{
        PINSEL0     &=    ~(X_minus_mask);          // X- is digital I/O
        PINMODE0&= ~(X_minus_mask);
        PINMODE0|= X_minus_no_pull;                 // no pullup on X-
        IODIR0      &=    ~(X_minus);               // X- is an input

        PINSEL0     &=    ~(Y_minus_mask);          // Y- is digital I/O
        PINMODE0&= ~(Y_minus_mask);
        PINMODE0|= Y_minus_no_pull;                 // no pullup on Y-
        IODIR0      |=    Y_minus;                  // Y- is an output
        IOCLR0      |=    Y_minus;                  // make Y- low

        PINSEL1     &=    ~(X_plus_mask);
        PINSEL1     |=    ADC_on_X;                 // X+ is an ADC pin

        PINSEL1     &=    ~(Y_plus_mask);           // Y+ is digital I/O
        PINMODE1&= ~(Y_plus_mask);                  // clear the two bits &
        PINMODE1|= Y_plus_no_pull;                  // no pullup on Y+
        IODIR0      |=    Y_plus;                   // Y+ is an output
        IOSET0      |=    Y_plus;                   // make Y+ high
}

 void config_pins_touch (void)

{
        PINSEL0     &=    ~(X_minus_mask);          // X- is digital I/O
        PINMODE0&= ~(X_minus_mask);                 // clear the two bits &
        PINMODE0|= X_minus_no_pull;                 // no pullup on X-
        IODIR0      &=    ~(X_minus);               // X- is an input

        PINSEL0     &=    ~(Y_minus_mask);          // Y- is digital I/O
        PINMODE0&= ~(Y_minus_mask);
        PINMODE0|= Y_minus_no_pull;                 // no pullup on Y-
        IODIR0      |=    Y_minus;                  // Y- is an output
        IOCLR0      |=    Y_minus;                  // make Y- low

        PINSEL1     &=    ~(Y_plus_mask);           // Y+ is digital I/O
        PINMODE1&= ~(Y_plus_mask);                  // clear the two bits &
        PINMODE1|= Y_plus_no_pull;                  // no pullup on Y+
        IODIR0      &=    ~(Y_plus);                // Y+ is an input

        PINSEL1     &=    ~(X_plus_mask);           // X+ is digital I/O
        PINMODE1&= ~(X_plus_mask);                  // pullup on X
        IODIR0      &=    ~(X_plus);                // X+ is an input
}

void display_lcd(short x_value, short y_value)

{
   unsigned char ch;
```

```
   set_cursor (0, 0);
   lcd_print ("x-value = ");

   ch = ((x_value >> 8) & 0x03);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = ((x_value >> 4) & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = (x_value & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);

   set_cursor (0, 1);
   lcd_print ("y-value = ");

   ch = ((y_value >> 8) & 0x03);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = ((y_value >> 4) & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = (y_value & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);

 }

void led_green (void)

{
           IOSET0     |= 0x00000800;              // P0.11 = high
           IOCLR0     |= 0x00000400;              // P0.10 = low
}

void led_red (void)

{
           IOSET0     |= 0x00000400;              // P0.10 = high
           IOCLR0     |= 0x00000800;              // P0.11 = low
}

 char hex_to_ascii(char ch)

{
    if ( ch < 10) ch += 0x30;
     else ch += (0x41 - 0x0A);
     return (ch);
}
```

```
void timer_delay (unsigned int count)

{
     T0TCR = 0x00000002;                 // disable and reset the timer
     T0CTCR = 0;                         // timer mode
     T0MR0 = count;                      // desired count
     T0PR = 0;                           //
     T0PC = 0;                           // prescaler
     T0MCR = 7;                          // reset timer , stop, and set flag on
     match
     while (T0IR & 1);                   // wait for match flag
     T0IR |= 1;                          // clear the IR bit
}
```

8.2 Source code for 5-wire touchscreen

```
// driver for 5-wire touch screeen


/*****************************************************************************/
/*                              header files                                 */
/*****************************************************************************/
#include "LPC23xx.H"                   /* LPC23xx definitions                */
#include "LCD.h"                       /* Graphic LCD function prototypes    */
/*****************************************************************************/
/*                                 pin definitions                           */
/*****************************************************************************/

#define probe                  0x01000000     // probe/ADC on P0.24  (X+)
#define probe_mask             0x00030000     // probe select mask (ADC0.1)
#define probe_no_pull          0x00020000     // probe no pullup value
#define ADC_on_probe           0x00010000     // probe pin select (1) ADC

#define upper_right            0x00000100     // upper_right on P0.8 (X-)
#define upper_right_mask       0x00030000     // upper_right pin select mask
#define upper_right_no_pull    0x00020000     // upper_right no pullup value

#define upper_left             0x04000000     // upper_left on P0.26
#define upper_left_mask        0x00300000     // upper_left pin select mask

#define lower_right            0x02000000     // lower_right on P0.25 (Y+)
#define lower_right_mask       0x000C0000     // lower_right select mask(ADC0.2)
#define lower_right_no_pull    0x00080000     // lower_right no pullup value

#define lower_left             0x00000200     // lower_left on P0.9 (Y-)
#define lower_left_mask        0x000C0000     // lower_left pin select mask
#define lower_left_no_pull     0x00080000     // lower_left no pullup value
```

```
/*****************************************************************************/
/*                          timer count definitions      */
/*****************************************************************************/

#definedebounce             1000           // debounce delay
#definesettling             100            // settling time delay


/*****************************************************************************/
/*                              function prototypes     */
/*****************************************************************************/

extern unsigned long install_irq( unsigned long IntNumber, void *HandlerAddr, unsigned
     long Priority );

void config_pins_x (void);
void config_pins_y (void);
void config_pins_touch (void);
void detected(void)__irq;
void display_lcd(short x_value, short y_value);
char hex_to_ascii(char ch);
void led_green (void);
void led_red (void);
void read_ch_x (void);
void read_ch_y (void);
void timer_delay (unsigned int count);
unsigned int touch_detect (void);


/*****************************************************************************/
/*                                        globals      */
/*****************************************************************************/

#definenum_samples16                   // number of A/D samples per axis

unsigned int x_values[num_samples];        // array to store x_samples
unsigned int y_values[num_samples];        // array to store y_samples


/*****************************************************************************/
/*                              start of main code      */
/*****************************************************************************/

int main (void)
{
     int i, j;

     PCONP |= (1 << 12);                        // Enable power to AD block
     lcd_init();                                // init the LCD display
     lcd_clear();                               // clear the LCD display
```

```
        for (i = 0; i < 20000000; i++);           // Wait for initial display
        IODIR0 |= 0x00000C00;                     // config touch LED pins as outputs
        led_green();                              // make the LED green

        PINMODE4 &= ~(0xFFFF);                    // P2[7:0] pullups
        PINSEL4 &= ~(0xFFFF);                     // P2[7:0] are GPIO
        FIO2DIR0 = 0xFF;                          // P2[7:0] are outputs
        FIO2MASK0 = 0x00;                         // P2[7:0] enabled for fast I/O

        touch_detect();                           // setup for touch detection
        install_irq(17, (void*)detected, 1);      // setup interrupt vector
        IO0_INT_EN_F = probe;                     // enable falling edge probe interrupt

        j = 0;
        while (1)                                 // Loop forever until interrupt
              {
                    for (i = 0; i < 200000; i++); // delay
                    FIO2PIN0 = (j & 0xFF);        // output the count to LEDs
                    j++;                          // increment the count
              }
}

void detected(void)__irq
{
        short x_value, y_value, i;

        timer_delay (debounce);                   // debounce the touch
        while ((touch_detect()))                  // loop as long as screen is touched
              {
                    led_red();
                    read_ch_x();                  // read and collect the x values
                    read_ch_y();                  // read and collect the y values

                    x_value = 0;                  // initial value
                    for (i=0; i < num_samples; i++)
                          {
                                x_value += x_values[i]; // add up the conversion results
                          }
                    x_value = x_value /num_samples; // get average

                    y_value = 0;                  // initial value
                    for (i=0; i < num_samples; i++)
                          {
                                y_value += y_values[i]; // add up the conversion results
                          }
                    y_value = y_value /num_samples; // get average

                    if (touch_detect())display_lcd(x_value, y_value); // display values if
        still have a touch
              }
```

```
          IO0_INT_CLR = probe;                // clear falling edge interrupt on
     probe
          led_green();
          VICVectAddr = 0;                    // Acknowledge Interrupt
}

void read_ch_x (void)
{
          unsigned int i;

          config_pins_x();                    // configure pins for reading x
     direction
          timer_delay (settling);             // settling time for switching
          AD0CR       = 0x00200302;           // Power up, PCLK/4, sel AD0.1

          for (i=0; i < num_samples; i++)
               {
                    AD0CR    |= 0x01000000;         // Start A/D conversion
                    while (AD0DR1 & 0x80000000);    // wait until conversion is
     completed
                    x_values[i] = ((AD0DR1 >> 6) & 0x3FF);// store result
               }
}

 void read_ch_y (void)

{
          unsigned int i;

          config_pins_y ();                   // configure pins for reading x
     direction
          timer_delay (settling);             // settling time for switching
          AD0CR       = 0x00200302;           // Power up, PCLK/4, sel AD0.1

          for (i=0; i < num_samples; i++)
               {
                    AD0CR    |= 0x01000000;         // Start A/D conversion
                    while (AD0DR1 & 0x80000000);  // wait til conversion completed
                    y_values[i] = ((AD0DR1 >> 6) & 0x3FF);// store conversion result
               }
}

  unsigned int touch_detect (void)
{
          config_pins_touch ();          // configure pins for touch detection
          timer_delay (settling);        // settling time for switching
          return((IOPIN0 & probe)^probe);  // return true if touch is detected
}


void config_pins_x (void)
```

```
{
        PINSEL0   &=  ~(upper_right_mask); // upper_right is digital I/O
        PINMODE0&= ~(upper_right_mask);
        PINMODE0|= upper_right_no_pull;     // no pullup on upper_right
        IODIR0    |=  upper_right;          // upper_right is an output
        IOSET0    |=  upper_right;          // make upper_right high

        PINSEL0   &=  ~(lower_left_mask);   // lower_left is digital I/O
        PINMODE0&= ~(lower_left_mask);
        PINMODE0|= lower_left_no_pull;      // no pullup on lower_left
        IODIR0    |=  lower_left;           // lower_left is an output
        IOCLR0    |=  lower_left;           // make lower_left low

        PINSEL1   &=  ~(lower_right_mask); // lower_right is digital I/O
        PINMODE1&= ~(lower_right_mask);     // clear the two bits &
        PINMODE1|= lower_right_no_pull;     // no pullup on lower_right
        IODIR0    |=  lower_right;          // lower_right is an output
        IOSET0    |=  lower_right;          // make lower_right high

        PINSEL1   &=  ~(upper_left_mask);   // upper_left is digital I/O
        PINMODE1&= ~(upper_left_mask);      // clear the two bits &
        IODIR0    |=  upper_left;           // upper_left is an output
        IOCLR0    |=  upper_left;           // make upper_left low

        PINSEL1   &=  ~(probe_mask);        //
        PINSEL1   |=  ADC_on_probe;         // X+ is an ADC pin
}

void config_pins_y (void)
{
        PINSEL0   &=  ~(upper_right_mask);  // upper_right is digital I/O
        PINMODE0&= ~(upper_right_mask);
        PINMODE0|= upper_right_no_pull;     // no pullup on upper_right
        IODIR0    |=  upper_right;          // upper_right is an output
        IOCLR0    |=  upper_right;          // make upper_right low

        PINSEL0   &=  ~(lower_left_mask);   // lower_left is digital I/O
        PINMODE0&= ~(lower_left_mask);
        PINMODE0|= lower_left_no_pull;      // no pullup on lower_left
        IODIR0    |=  lower_left;           // lower_left is an output
        IOSET0    |=  lower_left;           // make lower_left high

        PINSEL1   &=  ~(lower_right_mask);  // lower_right is digital I/O
        PINMODE1&= ~(lower_right_mask);     // clear the two bits &
        PINMODE1|= lower_right_no_pull;     // no pullup on lower_right
        IODIR0    |=  lower_right;          // lower_right is an output
        IOSET0    |=  lower_right;          // make lower_right high

        PINSEL1   &=  ~(upper_left_mask);   // upper_left is digital I/O
        PINMODE1&= ~(upper_left_mask);      // clear the two bits &
```

```
        IODIR0    |=   upper_left;          // upper_left is an output
        IOCLR0    |=   upper_left;          // make upper_left low

        PINSEL1   &=   ~(probe_mask);       //
        PINSEL1   |=   ADC_on_probe;        // X+ is an ADC pin
}


 void config_pins_touch (void)

{
        PINSEL0   &=   ~(upper_right_mask); // upper_right is digital I/O
        PINMODE0&= ~(upper_right_mask);
        PINMODE0|= upper_right_no_pull;     // no pullup on upper_right
        IODIR0    |=   upper_right;         // upper_right is an output
        IOCLR0    |=   upper_right;         // make upper_right low

        PINSEL0   &=   ~(lower_left_mask);  // lower_left is digital I/O
        PINMODE0&= ~(lower_left_mask);
        PINMODE0|= lower_left_no_pull;      // no pullup on lower_left
        IODIR0    |=   lower_left;          // lower_left is an output
        IOCLR0    |=   lower_left;          // make lower_left low

        PINSEL1   &=   ~(lower_right_mask); // lower_right is digital I/O
        PINMODE1&= ~(lower_right_mask);     // clear the two bits &
        PINMODE1|= lower_right_no_pull;     // no pullup on lower_right
        IODIR0    |=   lower_right;         // lower_right is an output
        IOCLR0    |=   lower_right;         // make lower_right low

        PINSEL1   &=   ~(upper_left_mask);  // upper_left is digital I/O
        PINMODE1&= ~(upper_left_mask);      // clear the two bits &
        IODIR0    |=   upper_left;          // upper_left is an output
        IOCLR0    |=   upper_left;          // make upper_left low

        PINSEL1   &=   ~(probe_mask);       // probe is digital I/O
        PINMODE1&= ~(probe_mask);           // pullup on probe
        IODIR0    &=   ~(probe);            // probe is an input
}



void display_lcd(short x_value, short y_value)

{
   unsigned char ch;

   set_cursor (0, 0);
   lcd_print ("x-value = ");

   ch = ((x_value >> 8) & 0x03);
   ch = hex_to_ascii(ch);
```

```
   lcd_putchar (ch);
   ch = ((x_value >> 4) & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = (x_value & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);

   set_cursor (0, 1);
   lcd_print ("y-value = ");

   ch = ((y_value >> 8) & 0x03);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = ((y_value >> 4) & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);
   ch = (y_value & 0x0F);
   ch = hex_to_ascii(ch);
   lcd_putchar (ch);

 }

void led_green (void)

{
           IOSET0     |= 0x00000800;              // P0.11 = high
           IOCLR0     |= 0x00000400;              // P0.10 = low
}

void led_red (void)

{
           IOSET0     |= 0x00000400;              // P0.10 = high
           IOCLR0     |= 0x00000800;              // P0.11 = low
}

 char hex_to_ascii(char ch)

{
    if ( ch < 10) ch += 0x30;
     else ch += (0x41 - 0x0A);
     return (ch);
}


void timer_delay (unsigned int count)

{
     T0TCR = 0x00000002;                          // disable and reset the timer
     T0CTCR = 0;                                  // timer mode
```

```
    T0MR0 = count;                          // desired count
    T0PR = 0;                               //
    T0PC = 0;                               // prescaler
    T0MCR = 7;                              // reset timer , stop, and set flag
    on match
    while (T0IR & 1);                       // wait for match flag
    T0IR |= 1;                                          // clear the IR bit
}
```

9. Legal information

9.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

9.2 Disclaimers

General — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in medical, military, aircraft, space or life support equipment, nor in applications where failure or malfunction of a NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors accepts no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

9.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

10. Contents

Date of release: 13 November 2008
Document identifier: AN10675_2