AN11621

LPC82x Touch Solution Library User Guide

Rev. 1.0 — 22 December 2014 User Guide

Document information

Info	Content
Keywords	LPC82x Touch Solution, User Guide, Library, Capacitive Touch, Touchpad, Sensor, Electrode, Drive/Sensing lines, Dielectric, Overlay panel, Sensitivity, Touch, False Touch.
Abstract	This document describes how to use NXP's LPC82x Touch Solution firmware library. It contains a functional description of the library software, its application programming interface and a description of all user adjustable configuration parameters.

Revision history

Rev	Date	Description
1.0	20141222	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

The LPC82x Touch Solution is based on a royalty free software library for capacitive touch sensing developed specifically for NXP's LPC82x family of Microcontrollers. It uses the microcontroller's GPIO functionality and an analog comparator input to implement a touch sensing circuit. See Fig 1. The GPIOs are controlled to emulate a switched-capacitor integrator circuit that senses relative change in the capacitance of a touch sensor (touch sensing element) in case of touch by a human finger. The external circuitry consists of only one charge storage capacitor (Cs) and a sensor layout interfaced with MCU. The sensor layout can be a standard capacitive touch input element composed of inter-digitated electrodes of copper, ITO or any other conductive material housed in a dielectric (glass, plastic, etc.) that meets the design specifications of an intended application. The touch solution library can be used to implement any user interface that involves simple conventional touch sensors (buttons/slider/wheel) or complex sensors (two dimensional touch) inputs.

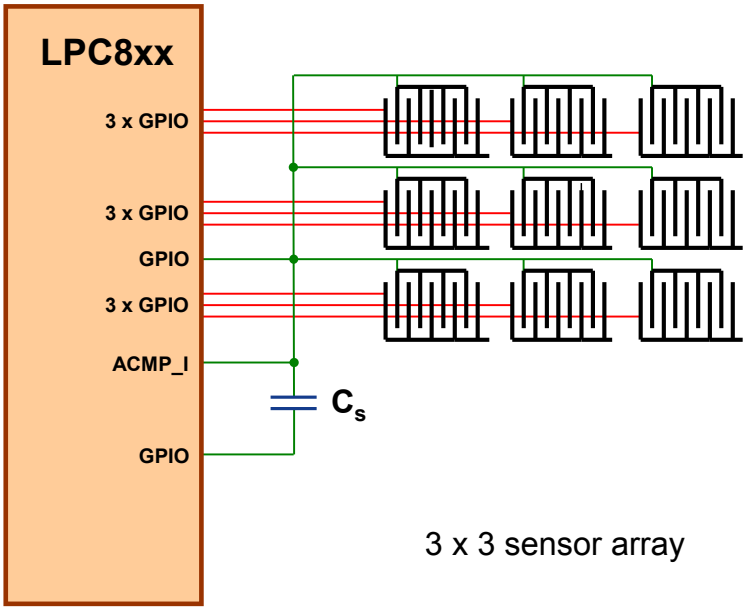

Fig 1. External Touch Circuitry

The LPC82x touch solution library also conditions the touch data (one or two dimensional) by an integrated processing algorithm and then provides to the application.

User input elements (sensor configurations) and touch configuration parameters may be altered using data structures. To notify the touch events to other interfaced components or a host, the default communication interfaces (GPIO/I2C/SPI/UART/USB) of the microcontroller can be used. For example, to indicate the user touch input, the user application software could output the touch position data to a (LCD) display unit or to a host PC running a GUI utility.

2. Functional description

2.1 Overview

Fig 2 shows the complete system architecture of NXP's Touch Solution. The functional block representation is on the left and to the right is the circuit schematic representation. The touch sensors act as inputs to a sensing circuit (SW based Touch Solution IP) realized within the NXP's LPC82x microcontroller.

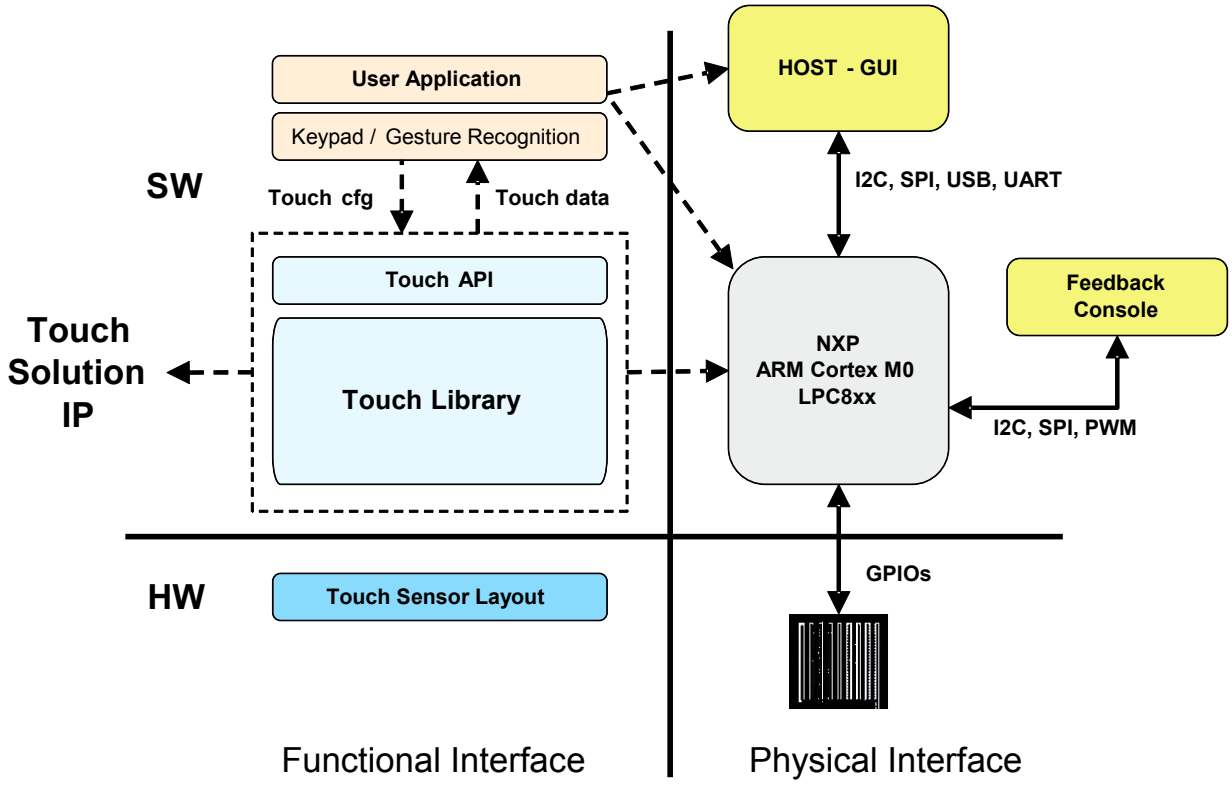

Fig 2. LPC82x Touch Solution Functional Block Diagram

As seen from the figure, the Touch Solution IP (Touch Library) drives the individual sensor elements, scans the complete sensing circuit layout, processes the touch signal data for robust/noise-free touch detection, and finally provides this data to the user application. Users can define their own application layer on top of the touch solution library to achieve his end objectives. The touch library gives complete flexibility to the user application to select between one or two dimensional sensing mode as well as configuring the touch sensing parameters, even during run time.

The touch solution library handles the microcontroller's GPIO pins and analog comparator configuration, the sensor calibration, automatic gain control and scanning of the individual sensor elements for touch event detection.

In addition, the Touch Library is also responsible for noise filtering and signal conditioning. It also defines single sensor touch / no touch status and it calculates the touch position coordinates in two-dimensional mode.

The touch position calculation algorithm adapts automatically to a sensor layout of 3x3 matrix and the associated user settings. The position calculation is robust against ghost touches or sensor element non-uniformities.

The Application Interface layer handles all data and command exchange between the user application and the touch solution library. It takes care of tasks like initialization, service calls, and configuration and parameter settings.

2.1.1 Touch sensor

In capacitive touch sensing, the ultimate goal is to detect a human touch using relative change in the capacitance of capacitive touch sensor.

NXP's touch solution is based on mutual capacitance touch sensor whose fringing fields are influenced by a nearby conductor or human touch. Inter-digitated electrodes are well known examples for mutual capacitance sensors. A small section of such inter-digitated electrodes is shown in Fig-3 (A). It consists of two electrodes (X & Y) placed under an insulating or dielectric overlay (for example, glass, plexi-glass, or polycarbonate) that are held at different potentials. The electric field due to applied potential difference causes storage of electric charges between the electrodes and hence resulting in mutual capacitance between X & Y. The electrode geometry ensures that the direct electric field lines between the electrodes are much lesser than the fringing field lines. These fringing field lines arc through the dielectric overlay material and reach the other electrode. A human touch or any electrically conducting object nearby distracts some of these fringing filed lines as shown in Fig 3 (B). Therefore, some amount of the stored charge is lost as this conducting object provides parallel path to ground.

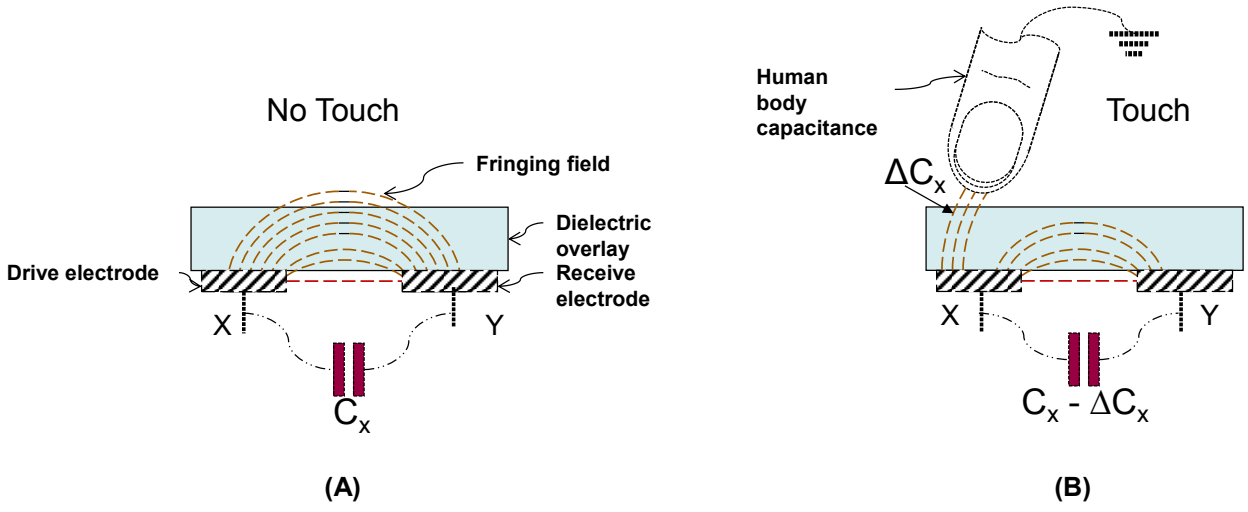

Fig 3. Section of Touch Sensor Layout

The electric flux coupled into Y electrode is much lesser in case of touch which results in reduced mutual capacitance. This change in mutual capacitance is sensed by the touch circuit to detect a touch event.

2.1.2 Touch Sensing Mechanism

NXP's touch sensing works on the principle of Switched Capacitor Integration circuit as shown in Fig-4. It consists of two capacitors (Cx-transfer and Cs-integration capacitor) controlled by two switches (S1 and S2) switched in non-overlapping fashion. When S1 is closed Cx charges to Vcc. Then S1 is opened and S2 is closed. This results in transfer of charge stored in Cx to Cs until both are at same potential. It is termed as one charge cycle where charge that is first stored in Cx is shared with Cs by alternate switching of S1 and S2. The value of Cs is chosen to be very large compared to Cx. Therefore, multiple charge cycles results in integration of charge stored on Cs (consequently increasing the voltage of Cs) and so, the name switched capacitor integrator circuit. After Cs is sufficiently charged to a measurable voltage, it is discharged using S3. One complete charge cycle of Cs forms an integration cycle (it is composed of multiple Cx charge cycles).

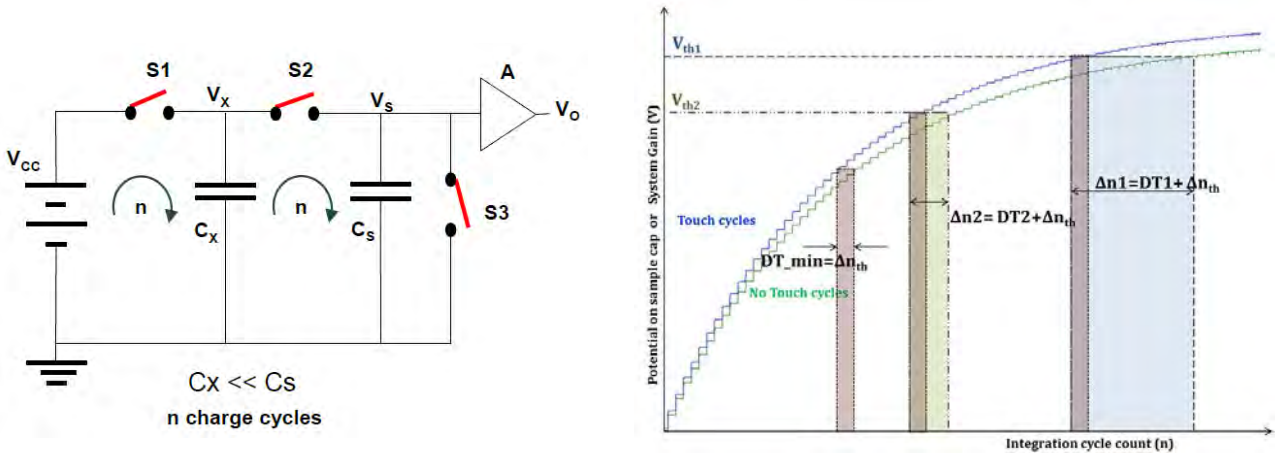

Fig 4. Switched Capacitor Integrator

A touch sensor circuit can be realized using switched capacitor integrator circuit by replacing the transfer capacitor Cx by a touch sensor that changes its mutual capacitance on touch event. When the sensor is touched with finger, some amount of charge stored on it is lost. This results in less amount of charge being transferred to Cs in every charge cycle. Therefore, in case of a touch event, it would require more charge cycles to reach the required measurement voltage level on Cs. In other words, the touch circuit requires more charge cycles in case of touch, to reach the same voltage level as the no-touch case. By calculating the difference in charge cycles with and without touch event, it is possible to detect touch event.

2.1.3 Touch Sensing Circuit

NXP's Touch Solution algorithm has fast detection speed and can support high sample rate that allows the implementation of additional filters on top to deal with harsh conditions arising due to the presence of moisture/water, dust, grease/fat, temperature variations, or stray electric field.

The equivalent touch sensing circuit is shown in Fig-5. The Cx is represented by a touch sensor layout and Cs as discrete passive integration capacitor. The switching is realized using GPIO pins of the microcontroller.

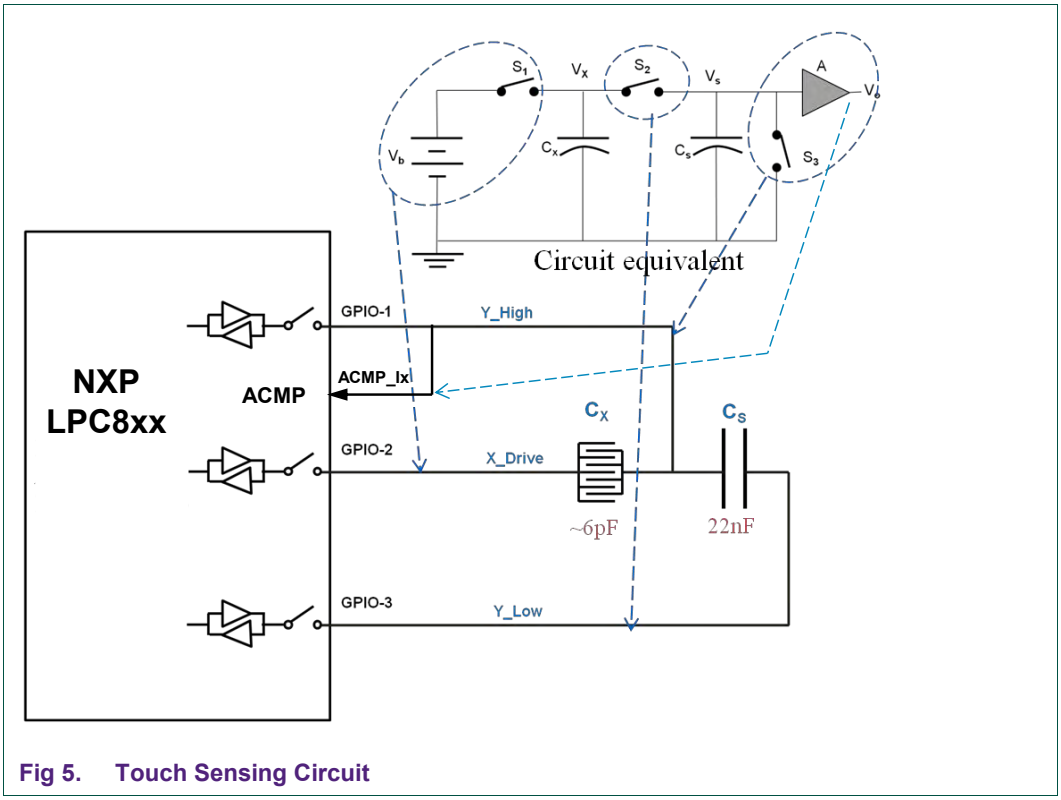

Fig 5. Touch Sensing Circuit

The sensor capacitor Cx forms a part of switched capacitor integrator. The CMOS tri-state drivers connected to the nodes X, Y-H and Y-L are controlled by trains of pulses, resulting in the emulation of integrator switches. The applied pulse sequence causes a repeated charge transport through the sensor capacitor Cx into sample capacitor Cs. With advancing number of integration cycles an integration voltage Vcs develops across the sample capacitor Cs.

In summary, the voltage at X-line drives the charge and charges the external sampling capacitor Cs through the touch sensor capacitor Cx. The touch sensor capacitor (Cx=~6pF) is much smaller than that of the external sampling capacitor (Cs=22nF).

In more detail, the sensor capacitor Cx is charged in every integration cycle to a voltage equivalent to the difference between supply voltage (Vdd) and the voltage across the sampling capacitor (Vcs). The same charge is also transferred into the series connected sampling capacitor (Cs), resulting in an increase of the voltage across sampling capacitor (Vcs). This integration process is repeated until the voltage Vcs reaches a fixed threshold voltage, see Fig-4. The number of integration cycles required to reach that threshold voltage is a measure for the capacitance of sensor capacitor Cx. The integration cycle count performed with no-touch is the reference and forms the basis for touch detection.

With a finger touch, the effective sensor capacitance (Cx-ΔCx) is reduced as the finger distracts some charge. Hence, it takes more integration cycles until the voltage across Cs reaches the threshold voltage. And that's the indication of a touch. Once the touch has been detected, further filtering and processing techniques are deployed to ensure the validity of touch. The sensitivity of the touch sensor is determined by the threshold voltage (Vth) as well as the capacitance of the sample capacitor Cs.

2.2 Touch Scanning Modes

The Touch Solution Library scans each sensor element for a touch event. At start-up all sensor elements are auto-calibrated for optimum gain, and the reference charge cycle count for each of them is calculated. After that each sensor element is scanned in a continuous loop for touch event detection.

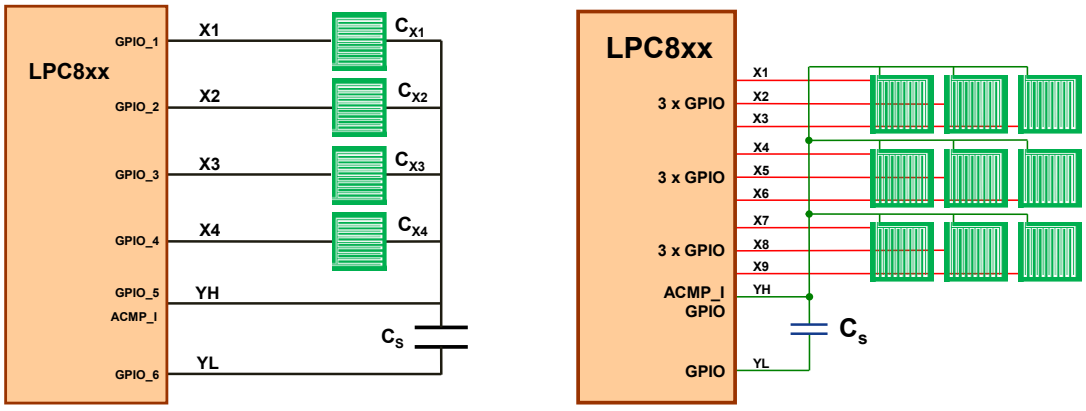

Fig 6. Touch Scanning Modes

Fig-6 represents two types of scanning modes. The one on the left is simple, one dimensional mode, while the right one is a more complex and consists of 9 sensor elements placed in 3 x 3 matrix format to enable recognition of 2D gestures. All sensor elements are connected to a common sampling capacitor Cs and scanned with a special technique for a touch event/position.

For a 2D layout with 3x3 matrix, sensor scanning can be done either in horizontal or vertical fashion. The choice of scan direction can also influence the noise and sensitivity of a touch board. The field patterns from the touch sensors may be affected by the external or stray field differently. Thus the scan direction could offer higher sensitivity and noise immunity if correctly chosen (configurable in current Touch Solution Library). Note that this choice of scan direction is not applicable for a smaller matrix than 3x3 or in one dimensional mode.

2.2.1 One Dimensional mode

In this mode each sensor is simply handled individually, hence no relation to physical arrangement of the sensors. It's an ideal mode for Keys/Buttons implementation.

Dual sensor touches are supported in this mode. The touch solution library deals with detection and evaluation of touch events. It simply calculates the difference in charge cycles for touch and no touch case and compares it with the detection threshold[1] before signaling an event to the application layer. In one dimensional mode, the sensor status change events are simple calls to an application defined "call back function". Data passed when calling this (user application) function is:

- **Event**: 1 byte indicating either a START – DATA – END event
- **Data**: 4 bytes of sensor status (touch / no touch) info

1. Minimum difference in charge cycles that is required to detect a valid touch event. Larger the difference between touch and no touch charge cycle count, better the design.

2.2.2 Two Dimensional mode

This mode can be used by an application to implement complex sensors or gestures, because the touch solution library handles interpolation of touch / no touch count differences into a touch position of a touch pad. In this mode, only single-touch (one finger) is supported. It transforms the sensor layout into a touch area as per user configuration.

The touch position evaluation is shown in Fig 7. It uses the difference in count between touch and no-touch charge cycle count of each sensor element which are weighted to transform touch values into touch position with predefined range. The touch position is reported in 8-bit resolution (0-255 positions).

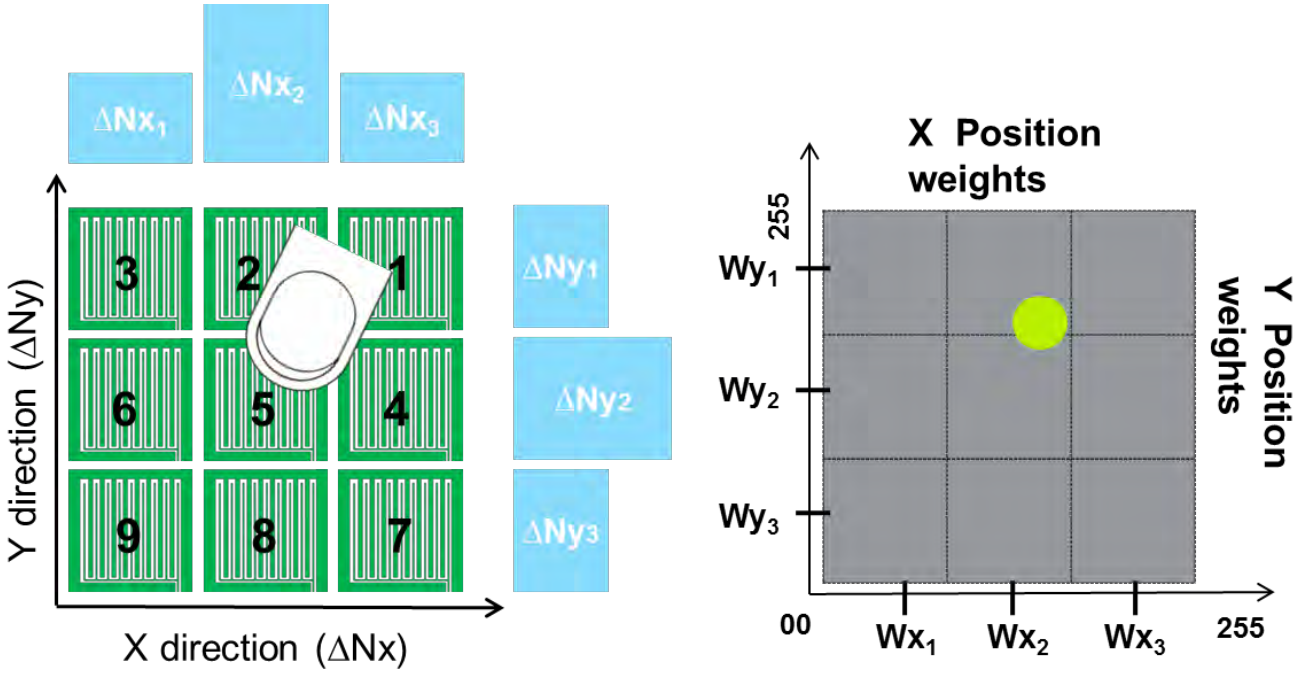

Fig 7. Touch Position Calculation

In two dimensional mode touch position events are again simple calls to a “call back function” (same as in one dimensional mode). Data passed when calling this (user application) function is:

- **Event type**: 1 byte indicating either a START – POS – END event
- **Position Index**: 1 word (16 bits) giving an index number (increments between a START and END event.
- **X - coordinate**: 1 byte giving the touch position X coordinate (0 - 255)
- **Y - coordinate**: 1 byte giving the touch position Y coordinate (0 - 255)

2.3 Touch Application Interface

This layer handles all data and command exchange between the user application and the touch solution library. It takes care of tasks like initialization, service calls, and configuration and parameter settings.

2.4 Microcontroller Resources

The LPC82x Touch Solution library acquisition method requires only one GPIO pin (X) per sensor plus two additional GPIO pins (YH and YL) and it uses the analog comparator (one of its inputs) in the process. These GPIO pins must be able to switch between High Impedance, Input and Output (High/Low), preferably with low pad/stray capacitance. They should not have any additional filters enabled / implemented. Furthermore, the Touch Solution library uses the MRT (Multi-Rate Timer) channels CH0 and CH1. These resources are dedicated for touch and cannot be used by the user application. For current library, memory footprint of flash is just over 3 kb and SRAM usage is around 80 bytes.

Compiled with Keil V5.10:

```
-------------------------------------------------------------------
Code        RW Data      ZI Data      Library Name
3150        76           208          LPC8xx_Touch_lib_v0100.lib
```

Compiled with LPCXpresso V7.2:

```
-------------------------------------------------------------------
text        data         bss          Library Name
3343        32           67           LPC8xx_Touch_lib_v0100.a
```

2.5 Performance indicators and Constraints

Number of sensors

The LPC8xx Touch Solution library supports a maximum number of row sensors of 3 (MAX_ROW_CNT) and also a maximum number of Touch sensors in a column of 3 (MAX_COL_CNT). This means that the maximum number of cap sensors that can be handled by the Touch Solution library is 9.

Timing

A complete cycle of scanning 9 sensor elements for Touch detection, eventual signal condition and position calculation takes about 3.6 milliseconds[2] (approximately 400us per sensor). This results in around 278 sample points per second with a spatial resolution of over 80 dpi.

Interrupts

The Touch Solution library itself is not using any interrupts. However, during the most time critical part of the charge cycle time measurement (inside Touch core loop) interrupts are shortly disabled (generally some tenths of system clock cycles). This can influence the real time behavior of the users end application.

2. Measured at LPC824 Microcontroller running at 24 MHz clock.

3. Touch Solution API

This section describes the LPC8xx Touch Solution library Application Programming Interface (API). Using the API, configuration of the Touch hardware (sensors and the associated GPIO port pins and analog comparator input channel) as well as the Touch parameters (like Touch mode, scan mode, filter depths, etc) can be defined. Once touch sensing has been initialized by the user, the host application gets triggered on changing touch events.

3.1 Public header file

The LPC8xx_Touch_Solution.h header file is the only public header file which needs to be included in a user's application project. It contains all type and data definitions and all function prototypes required by the host application to make use of the LPC8xx Touch Solution library.

3.2 Enumeration types

This section lists the enumerations used in the LPC8xx Touch Solution library.

3.2.1 TOUCH_MODES_T

This enumeration is used to define the Touch Solution operating modes.

Table 1. TOUCH_MODES_T

Item	Description
ONE_DIMENSIONAL	The Touch Processor handles each sensor individually.
TWO_DIMENSIONAL	The Touch Processing layer now handles interpolation of touch / no touch count differences into a touch position X / Y coordinate.

Note: In TWO_DIMENSIONAL mode, currently the sensor matrix is fixed as 3 x 3.

3.2.2 SCAN_MODES_T

This enumeration is used to select the Touch Solution scan mode.

Table 2. SCAN_MODES_T

Item	Description
HORIZONTAL	Selects the scanning mode of sensor elements to be in horizontal direction.
VERTICAL	Selects the scanning mode of sensor elements to be in vertical direction.
DIRECT	All sensors are scanned individually.

Note: The Horizontal and Vertical scan modes are not applicable for a smaller matrix than 3 x 3 or in one dimensional mode.

3.2.3 EVENTS_T

This enumeration defines the event types generated by the touch library and passed with the application call back function.

Table 3. EVENTS_T

Item	Description
EV_START	Touch detected. This indicates a touch START event (could be the beginning of a gesture).
EV_DATA	Touch data event. In one dimensional mode status of all sensors is provided. In two dimensional mode touch position index and X/Y coordinates are provided.
EV_END	Touch END event, no more touch detected (like key release, a finger lift off or end of a gesture).

3.3 Data structures

This section describes and explains the used data structures that hold touch system, touch configuration and touch parameter data.

3.3.1 TOUCH_CFG_T

The Touch configuration **const** data structure is declared once at startup and used by the application to output the (hardware) sensor circuit configuration to the Touch Solution library.

Table 4. TOUCH_CFG_T

Field	Type	Description
ROW_CNT	uint8_t	Number of rows in sensor matrix. Minimum 1, maximum is 3.
COL_CNT	uint8_t	Number of columns in sensor matrix. Minimum 1, maximum is 3. The total number of attached sensors are: ROW_CNT x COL_CNT
X [ROW] [COL] MAX_ROW_CNT = 3 MAX_COL_CNT = 3 MAX_SEN_CNT = 9	[uint8_t] [uint8_t]	Two dimensional array that gives a definition of the used sensor configuration. Row and column array members indicate which GPIO pin numbers are used to connect to which sensor. Note that if just a few individual sensors are used (like in one-dimensional mode) a row or column count of 1 can be defined. Note that the max number of rows and columns is 3, means that the total maximum number of possible sensors are 9.
YL	uint8_t	GPIO pin used as Y-low line. Able to switch between High Impedance, Input & Output (High/Low), preferably with low pad/stray capacitance. For example, this field is filled with 23, if port PIO0_23 is used.
ACMP_Ix	uint8_t	Analog comparator input number used. For example, this field is filled with 2, if ACMP_I2 input is used.
YH	uint8_t	GPIO pin used as Y-high line. Able to switch between High Impedance, Input & Output (High/Low), preferably with low pad/stray capacitance. For example, this field is filled with 14, if port PIO0_14 is used.

This data structure needs to be declared and defined by the host application. After that a pointer to it is passed when calling the Touch Solution library main initialization function.

```
extern void  Touch_Init(const TOUCH_CFG_T *cfg);
```

3.3.2 TOUCH_SYS_T

The Touch system data structure is defined by the library and exported to application, so that it can define the fixed touch system settings and provide the touch event “call back” function pointer to the library.

The library in its turn is using the “service_timer” field to indicate after how many milliseconds the Touch Task needs to be serviced again.

Table 5. TOUCH_SYS_T

Field	Type	Description
service_timer	uint8_t	This is the time in milliseconds returned by the Touch Task, after which it needs to be serviced again. The host application needs to keep track of this time elapsing before it calls the touch task again. See Public Functions section.
version	uint16_t	Touch library version number.
system_grounded	uint8_t	Indicates if target is a grounded or contactless system.
scan_mode	uint8_t	Defines the scanning mode used by the Touch Solution library (either direct, vertical or horizontal).
use_square	uint8_t	If ON (1) use non-linear amplification.
use_equalization	uint8_t	If ON (1) use reference based equalization.
ghost_touch_suppression	uint8_t	If ON (1) enables ghost touch suppression.
use_nearest_neighbor	uint8_t	If ON (1) enables noise suppression of sensors far away.
* cb_func (E, B[4])	void *(uint8_t , uint8_t)	Pointer to touch event call back function. Parameters passed by the touch lib when calling are: E Event type (START – DATA – END) B[4] 4 bytes of either sensor status or position data

The LPC8xx Touch Solution library exports a variable of this data type so that the host application can modify the required system settings before starting the Touch Task.

```
extern TOUCH_SYS_T sys;
```

3.3.3 TOUCH_PAR_T

The Touch parameters data structure is used by the host application to read and write the global touch parameters which determines the overall operation of Touch Solution library algorithm. The possibility of dynamically changing these parameters gives a great advantage and flexibility to the end user.

Table 6. TOUCH_PAR_T

Field	Type	Description
touch_mode	uint8_t	Touch operating mode (one - or two dimensional). One dimensional Mode: Keys/Buttons (touch or no touch) Two dimensional Mode: Touchpad/Gesture (X-Y position)
agc_mode	uint8_t	Automatic Gain Control feature for adjusting the sensitivity as desired. It enables auto-setting independent gain for each sensor such that they all have uniform sensitivity level as a group. The Touch Solution library internally varies gain of individual sensors to get a uniform behavior for the complete matrix. The gain values are set such that there is a considerable difference

Field	Type	Description
		between touch and no-touch state of sensors. ON (1): AGC feature enabled. It's recommended for one dimensional (Keys/Buttons) mode, when reasonable sensitivity couldn't be achieved only by fixing the overall system_gain value. OFF (0): AGC feature disabled, the common system_gain value loaded automatically for all the sensors. It's recommended to disable it for two dimensional mode.
agc_min	uint16_t	Only when Automatic Gain Control feature is enabled, the AGC minimum value is applicable. AGC MIN is the minimum gain value setting for any sensor to get reasonable sensitivity between touched and not-touched state. It's range is limited from 500 to 7000 based on manual calibration experiments.
agc_max	uint16_t	Only when Automatic Gain Control feature is enabled, the AGC maximum value is applicable. AGC Max is the maximum gain value setting for any sensor to get reasonable sensitivity between touched and not-touched state. Please note that by increasing the gain, the sensitivity can be increased, but at the cost of speed. So developers have to strike a balance between speed and sensitivity based on the application requirement. It's range is limited from 1000 to 7500 based on manual calibration experiments.
system_gain (only used if AGC is off)	uint8_t	The System gain is the common overall gain value for all sensors. It's applicable only when the AGC mode is disabled. It's range is limited from 0 to 31 based on manual calibration experiments.
dt_mode	uint8_t	Dynamic Detection Threshold feature for changing the threshold value dynamically as needed sometimes while dealing with the harsh conditions arising due to the presence of moisture/water, dust, grease/fat, temperature variations, stray electric field, etc. ON (1): Dynamic Detection Threshold feature enabled. The detection threshold value will be dynamically changed from DT_MIN to DT_MAX to achieve the reasonable sensitivity at all times in harsh or varying conditions. The Touch Solution library will automatically vary the detection threshold based on the sensitivity level at that point in time. OFF (0): Dynamic Detection Threshold feature disabled. The DT_MIN value will act as the Detection Threshold setting.
dt_min	uint16_t	DT MIN is the minimum value of Detection Threshold range. The detection threshold cannot have a large range of values, otherwise the performance will have an adverse effect. Hence it's range is limited from 0 to 1000.
dt_max	uint16_t	Maximum value of Detection Threshold range. Please note that setting the DT min or max to lower values increases the sensitivity of the system. It's range is limited from 1 to 1400.
fd_raw	uint8_t	fd_raw: is the IIR filter tap size for instantaneous signal (number of cycles necessary to charge the storage cap to sys gain level). Filtering reduces the jitter in the calculated postion and hence these cycles are averaged before position calculation to get stable values. Based on comparison with reference, they can be classified as touch cycles in case of touch event or update to reference cycles in case of no-touch event. In order to avoid transient noises, these cycles are also averaged. An IIR filter is realized in SW and filter depth refers to the number of taps. Typical value of 2 denotes the tap size of 2 to the power 2.
fd_ref	uint8_t	fd_ref: is the IIR filter tap size for reference signal, number of cycles

Field	Type	Description
		necessary to charge the storage cap to system gain level under untouched conditions. In order to avoid transient noises, these cycles are as well averaged using IIR filter realized in SW and filter depth refers to the number of taps. Typical value of 5 denotes the tap size of 2 to the power 5.
fd_pos	uint8_t	fd_pos: is the Moving Average filter tap size for touch position data. Touch position value calculated is not always stable due to non-uniform sensitivities of individual sensors or because of noise or even interference. This results in a small jitter in the calculated position value. To reduce this jitter, a Moving average filter is realized in the SW. The filter depth refers to the number of taps in the moving average filter. Maximum value of 3 denotes the tap size of 2 to the power 3.

The LPC82x Touch Solution library exports a variable of this data type so that the host application can modify and refine the operation and configuration parameters for the library:

```
extern TOUCH_PAR_T par;
```

3.4 Public Functions

This section describes the public functions available in the LPC8xx Touch Solution library and their usage. Only 3 functions are available which makes the application interface very clear and uncomplicated.

3.4.1 Touch_Init

This function is used to initialize the touch sensing for all connected sensors and performs calibration (measure the reference touch charge cycles). All hardware configuration, touch system settings and touch parameter configuration should be done before calling this function.

```
extern void  Touch_Init(const TOUCH_CFG_T *cfg);
```

Argument	Type	Description
cfg	const TOUCH_CFG_T *	Pointer to the hardware configuration of the sensor matrix (defines which port pins are used for which sensor and YH / YL lines)

3.4.2 Touch_Task

This function executes a scan of all capacitive sensors. The measured charge cycles for all sensors are then processed to check for user touches, to calculate touch positions and for touch releases. If any of these events did occur an application function (call back function) is called. On return this function has programmed a new task service time value in “timer_service” variable (see Table 55) of the TOUCH_SYS_T structure. This time value indicates after how many milliseconds the Touch_Task function needs to be serviced (called by the host application) again.

```
extern void Touch_Task(void);
```

Note: The host application should not call the Touch_Task before “timer_service” value elapses.

3.4.3 Touch_Update

This function needs to be called any time the user makes a modification to the system settings or the touch parameter values. This function simply notifies the touch library that a re-configuration and re-calibration of all sensors is needed.

```
extern void Touch_Update(void);
```

3.5 Startup sequence

The following sequence of actions is required to add the Touch Solution functionality to a user end application.

1. Make sure the touch library is included in your project and also include the library header file inside your source code module.

```
#include "LPC8xx_Touch_Solution.h"
```

2. Initialize and define the sensor configuration structure (TOUCH_CFG_T):

```
const TOUCH_CFG_T cfg = {3, 3,                      // 3 rows and 3 columns
                         {{PIN_X0, PIN_X3, PIN_X6},
                          {PIN_X1, PIN_X4, PIN_X7},
                          {PIN_X2, PIN_X5, PIN_X8}},
                          PIN_YL, ACMP_I3, PIN_YH };
```

3. Inside your application code define a function that will be used as a call back by the touch library and that handles the touch events. Example:

```
void TouchEventHandler(uint8_t event, uint8_t buf[4])
{
    switch (event)
    {
        case EV_START : LCD_LED(1, 1);          // LCD LED1 on
                        break;
        case EV_DATA  : LCD_Send(buf, 4);       // send position data to LCD
                        break;
        case EV_END   : LCD_LED(1, 0);          // LCD LED1 off
                        break;
    }
}
```

4. Add a pointer to this call back function to the system structure (TOUCH_SYS_T):

```
    sys.cb_func = TouchEventHandler;
```

5. Make sure all configuration touch solution parameters are initialized in structure (TOUCH_PAR_T). Their default values can be found in the LPC8xx_Touch_Solution header file. Example:

```
par.touch_mode      = TWO_DIMENSIONAL;
par.agc_mode        = USE_AGC
par.agc_min         = AGC_MIN_CYCLES;
```

```
par.agc_max          = AGC_MAX_CYCLES;
par.system_gain      = SYSTEM_GAIN;
par.dt_mode          = USE_DDT;
par.dt_min           = DDT_MIN_LIMIT;
par.dt_max           = DDT_MAX_LIMIT;
par.system_grounded = SYSTEM_GROUNDED;
par.fd_pos           = POS_FILTER;
par.fd_ref           = REF_FILTER;
par.fd_raw           = RAW_FILTER;
```

6. Call the touch library initialization function (passing the sensor configuration).

```
Touch_Init(&cfg);
```

7. Now all initialization is done. Finally the user application only needs to initializes a 1 millisecond system tick timer and use that to periodically call the Touch_Task function to read the status of capacitive touch sensors.

   Note: Please refer to "AN11620: LPC82x Touch Solution Application Note" for more details on example projects using Touch Solution library.

4. Legal information

4.1 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.

4.2 Disclaimers

Limited warranty and liability — 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.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

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 life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an 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.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

4.3 Trademarks

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

5. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 December 2014
Document identifier: AN11621