UM10897

Gesture Recognition Library User Guide

Rev. 1.0 — 09 June 2015 User Guide

Document information

Info	Content
Keywords	LPC82x Touch Solution, Gesture Recognition, Touch Analysis, User Guide, Library, Capacitive Touch, Touchpad, Sensor, Drive / Sensing lines, Touch.
Abstract	This document describes how to use NXP's Touch Solution Gesture Recognition firmware library. It contains a functional description of the library software, its application programming interface and description of the GUI and training set creation.

Revision history

Rev	Date	Description
1.0	20150609	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 Gesture Recognition library of NXP can be used along with the LPC82x Touch Solution library to enable a unique gesture/pattern recognition feature. With this, it is able to recognize a number of predefined (trained) drawing patterns (gestures or hand writing) created with the touch solution library code. Fig 1 shows the overall gesture recognition application concept.

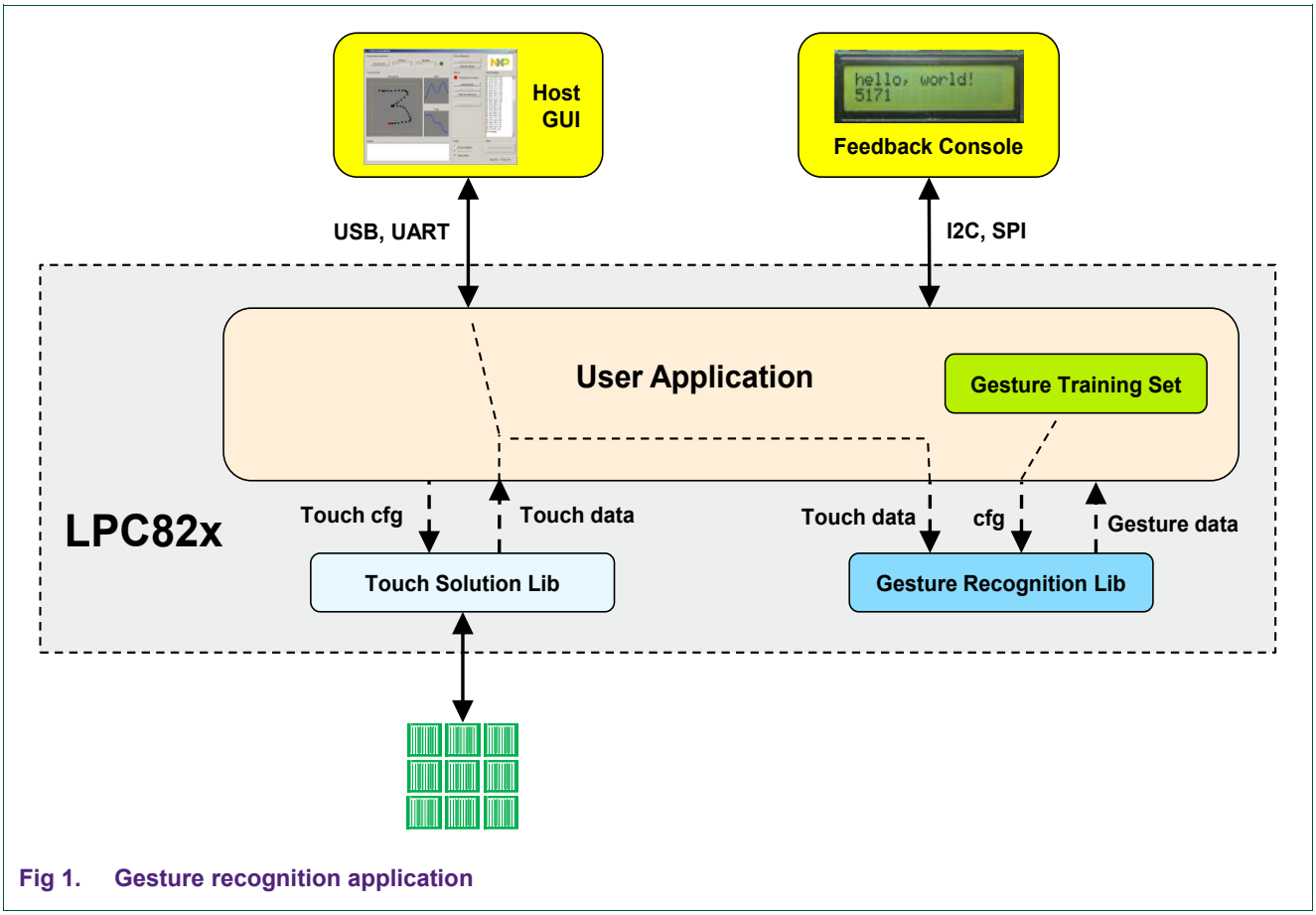

Fig 1. Gesture recognition application

To use the gesture recognition feature, the Touch Solution firmware must be configured for two dimensional mode that transforms the capacitive sensor layout into a touch area. See Fig 2. The differences in charge cycle count between touch and no-touch of all sensor elements are weighted and translated into a two dimensional touch position. The touch positions (X, Y coordinates) are reported in 8-bit resolution (0-255 positions) and subsequently passed to the gesture recognition library.

For more information regarding the LPC82x Touch Solution library see:

LPC82x Touch Solution | www.LPCware.com

The Gesture Recognition library receives its input data as a series of touched positions on the touch pad. Next, the input data is compared with the pre-loaded reference training data for a ‘gesture’ match.

The gesture recognition algorithm is designed to be robust enough to detect handmade gestures and still be computationally light (that is, less CPU, memory and fast enough to detect a drawn gesture before the next is entered). Moreover, it can be ported to any application and microcontroller family with no or minor modifications.

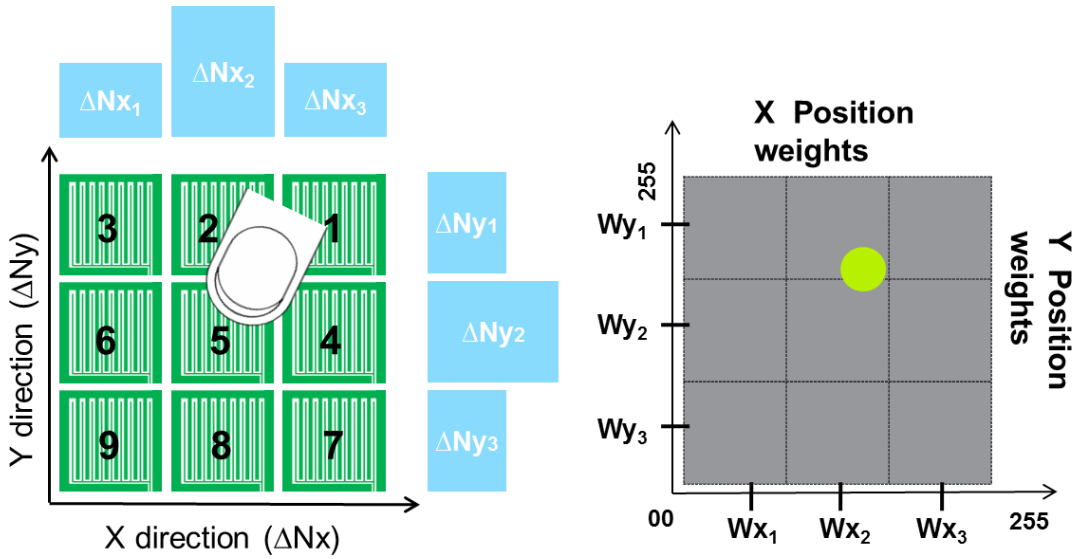

Fig 2. Touch position calculation

Gesture recognition consists of two main steps:

- Gesture training.
- Gesture capturing and recognition.

2. Gesture training

The first step in implementing the Gesture recognition functionality is **training**. Drawn patterns for every gesture are recorded and stored into a “training set”. A training set can be created with the help of a GUI (PC windows application) based NXP gesture creation and recognition application. See Fig 3.

2.1 Create training set

A new training set can be created by clicking the “Create new training set” button. This will open a new application window, see Fig 4. Currently, the maximum number of gestures a training set can contain is 10 (numbered 0 – 9) while the maximum number of trainings per gesture can be up to 99.

A drawing pattern for a gesture (or a character) can be arbitrary and does not need to match the gesture number in size or shape. For each gesture, the series of touch positions are recorded multiple times (user configurable) and the reference pattern considering deviations (within predetermined tolerance range) of each iteration is calculated. Multiple iterations while training a specific gesture helps in taking into consideration all user variations or dynamics (speed, size). Thus for every gesture, a reference map is created. All trained gestures together form a total training set.

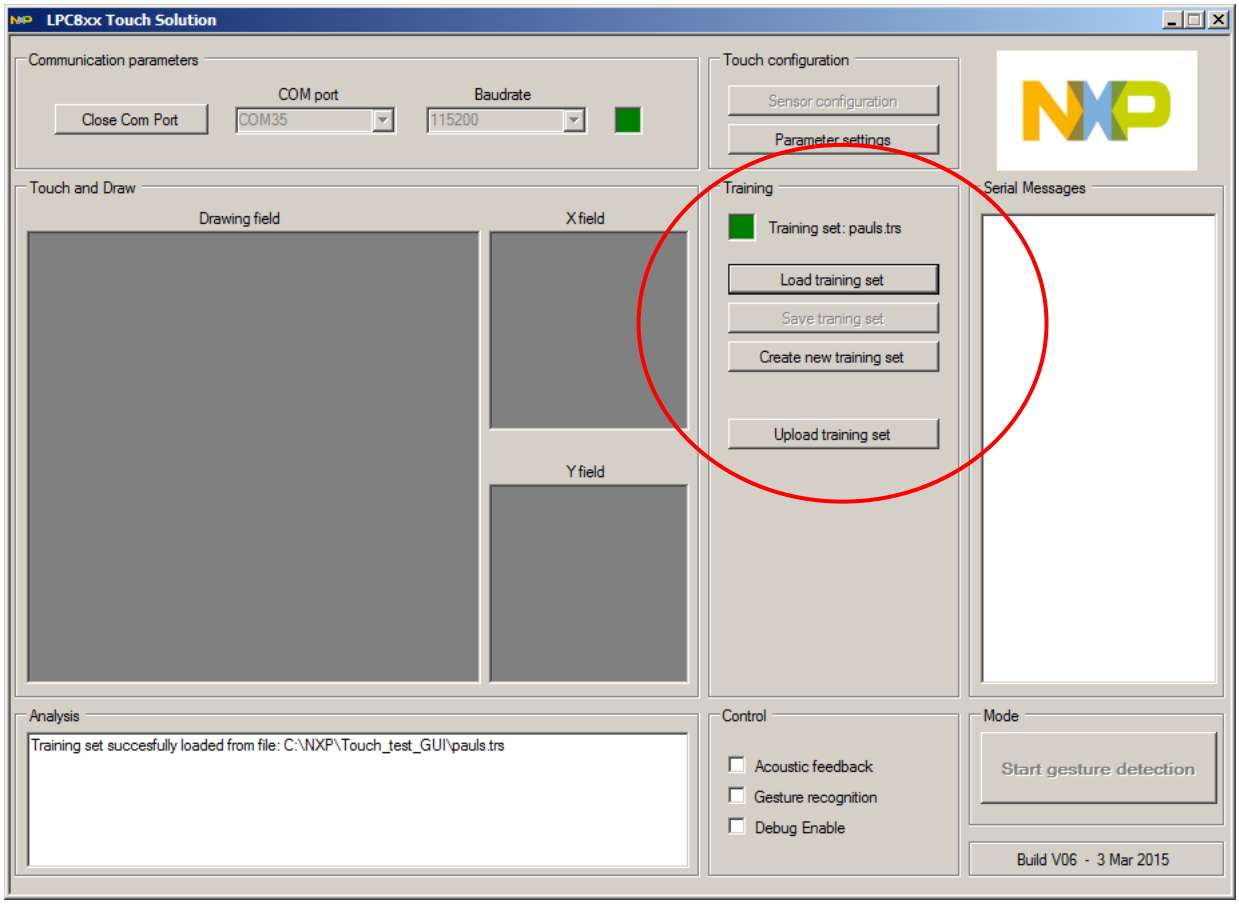

Fig 3. NXP touch solution GUI

The training set that was created can be saved to a file and uploaded to the target microcontroller (using UART communication) for storage in SRAM, flash, or EEPROM (if available). The gesture solution application example from AN11666, stores it at the highest available pages of the internal Flash memory. After the training set is uploaded or stored, it can be used by the gesture recognition library of the target. For more information about the use of the tool and creation of a training set, see AN11666, Gesture Recognition Application Note in:

LPC82x Touch Solution | www.LPCware.com

2.2 Training set storage

During the training, every gesture is drawn repeatedly. The GUI application calculates the maximum and minimum value (of X and Y coordinates) of every touch position coordinate (that is. coordinate range for every sample). During recognition phase, the difference in the actual position coordinates of input gesture and reference range of interested pattern is squared and summed to compute the error value.

For creation of the minimum and maximum values and to eliminate outlier effects, the mean and variance are considered. All iterations of a gesture during training process are first resampled and filtered. For each position coordinate, the mean and standard deviation value (separately for X and Y coordinate) are calculated. The mean determines the most probable locus of the user gesture and the standard deviation would indicate the average variation from the probable locus. Thus, the minimum value is the difference between the mean and the standard deviation value and the maximum value is the sum of the mean and the standard deviation value. This reduces the overall area between the minimum and the maximum. Regions with large variations have a broader range while consistent regions have a small range.

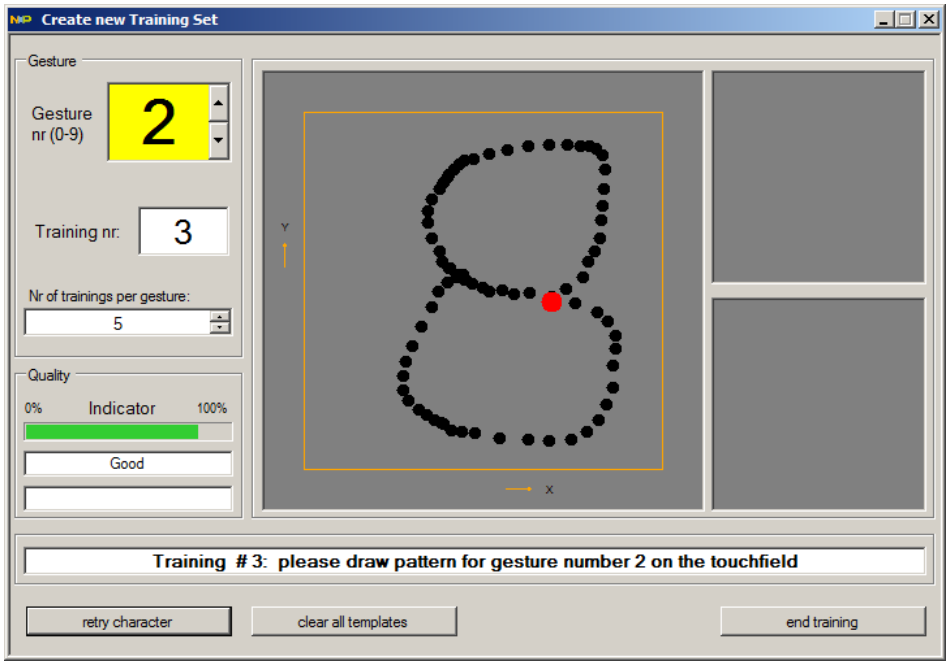

Fig 4. Gesture training application

2.3 Training set upload

The Touch Solution GUI allows the direct uploading of the training set to the target microcontroller (using USB - UART communication) for storage in SRAM, flash, or EEPROM (if available). Note: the host application of the user takes care of this task. An implementation example can be found in AN11666, Gesture Recognition Application Note.

To upload a training set, use the ‘Upload training set’ button in the ‘Training’ panel. The status during the upload is shown in the analysis box. Currently the 10 gestures are uploaded as 40 packets of 64 bytes (+ packet number and 16 bit CRC).

2.4 Uniqueness and quality indication

While gestures are trained, a quality indicator is displayed. The best compromise between FAR (False Acceptance Rate) and FRR (False Rejection rate) is a relation between the gesture area and full area of ~10%. A lower value sets the limits very close together and it is hard to repeat the exact gesture, resulting in a higher FRR. If the error limits are too wide, nearly every drawing will be recognized as a valid gesture.

Therefore, the indicator marks the quality as 100% if the summed area between the high and low limit divided by (256x256) is 10%. Anything more or less, the quality decreases, until the indicator shows 0 quality at 0%, respectively 20% area.

After the process of creating the training set is completed, a “uniqueness” test is performed. This test checks if two gestures are not too similar (that is, ‘1’ & ‘I’). The test uses a Monte Carlo algorithm (=random points inside a gesture limit, which are compared to all other gestures).

3. Gesture capturing and recognition

After the training set is uploaded to the target microcontroller, the user can start the gesture recognition process. When the user draws the gesture on the touch pad it is compared with the training set for a match.

3.1 Process

The mean square error between points of reference and the gesture that is entered is computed for every gesture. The reference gesture number that has the minimum error value (a detection threshold or limit that decides the minimum probability for match) is returned as a match. The gesture recognition process is executed in four phases.

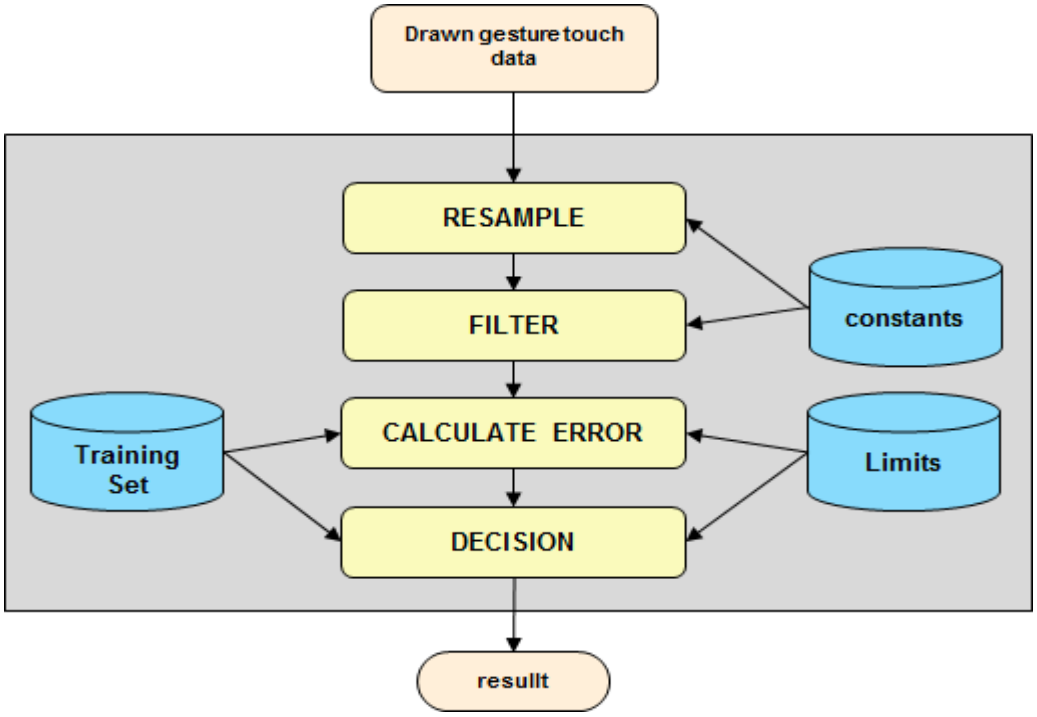

Fig 5. Gesture recognition process

3.1.1 Resampling

Every time a gesture is drawn, it is composed of a different number of position samples (depending on size, complexity, and drawing speed). To compensate for this non-uniformity, the input gesture is first resampled to a fixed size of 64 samples using a linear interpolation formula.

3.1.2 Filtering

The jitter caused by uneven sensor performance, interference noise, or unstable supply voltage is mostly filtered out at the lower touch-sensing layers. To remove more of the jitter, a median filter, based on noise types and levels, is applied to the input gesture position coordinates.

3.1.3 Error calculation

In this step, the “Manhattan” distance (sum of its X and Y coordinate values) parameter for every touch position coordinate is calculated. Next, the error or the difference between reference touch position and current touch position in terms of its Manhattan distance is computed. If the X, Y coordinate of the current sample point is outside the limit, the error will be increased by square of the distance from the current sample point to the limit.

This is done for all 64 sample points for X and Y direction. Finally, an algorithm is used to define and compare all offset errors between the reference/trained and current gesture with respect to its X and Y coordinates and the time shift.

3.1.4 Decision

Using the error value between reference and input gesture, the error probability is computed and compared against a detection threshold of the reference gesture. The detection threshold is computed from the training set during initialization of the gesture recognition library. It determines the maximum permissible error value between input gestures and reference/trained gestures.

Finally, if the error probability is within detection threshold limits (fixed in software) of the reference gesture, then the input gesture is identified as a valid match.

3.2 Result

A match or no-match result received from the Gesture library can be further processed by the application software. See Fig 6 for an example (derived from AN11666, Gesture Recognition Application Note) where gesture reference number 4 was recognized.

Fig 6. Gesture recognition by LPC824 target

3.3 Recognition by the GUI

It is also possible to use the GUI application to do gesture recognition based on its loaded training set. See Fig 7.

If both target and GUI have a training set in memory they can perform gesture recognition at the same time.

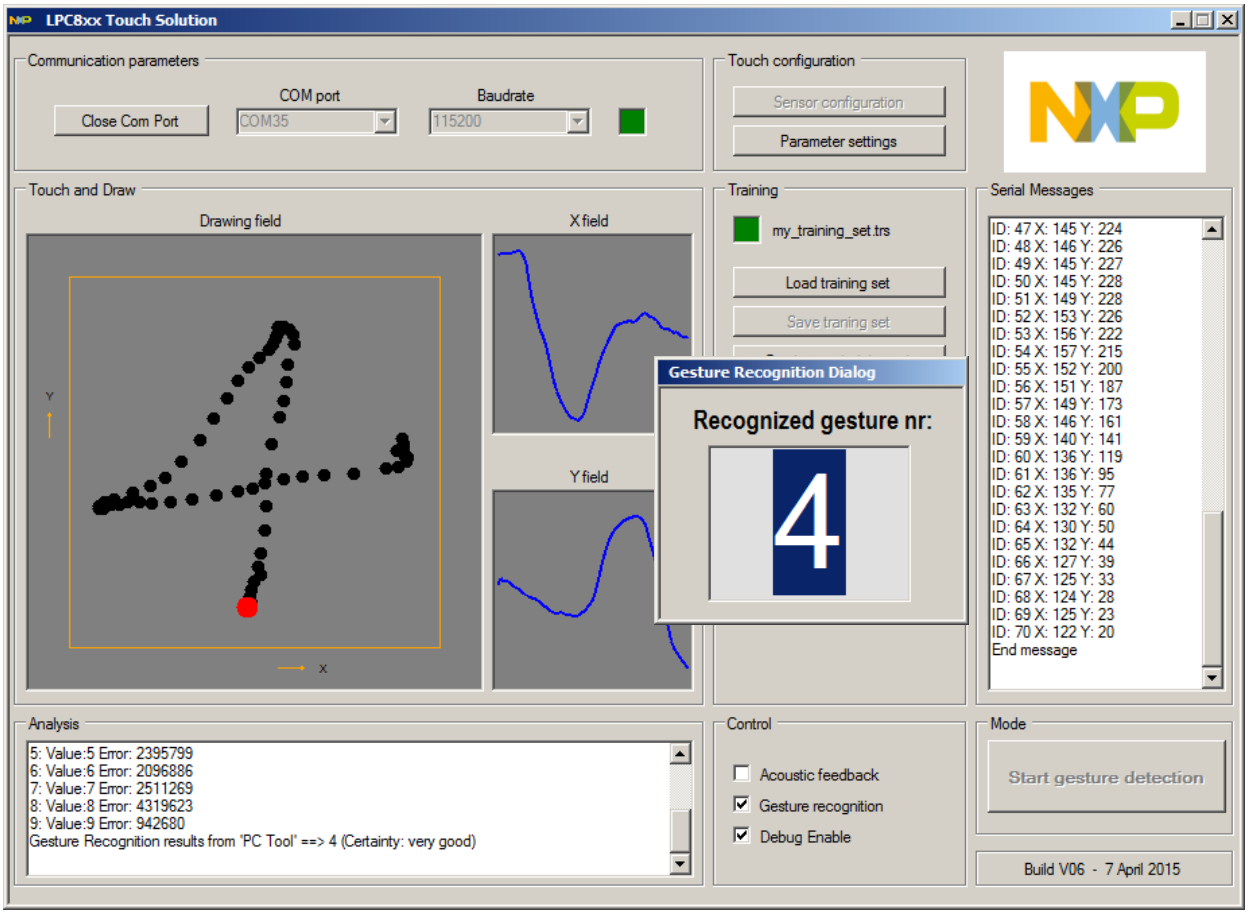

Fig 7. Gesture recognition by GUI

4. Gesture Recognition API

This section describes the Gesture Recognition library Application Programming Interface (API). During initialization the user application provides the required training set information to the gesture library. The user application can then pass all touch event data to the library. This data is analyzed and checked for a match with all gestures inside the training set.

The complete API consists of two data structures, a general header file and two callable public functions.

4.1 Public header file

The Gesture_Recognition.h header file is the only public header file that needs to be included in an application project of the user. It contains all type and data definitions and also function prototypes required by the host application to make use of the Gesture Recognition library.

4.2 Data structures

This section describes and explains the used data structure(s) that hold the gesture recognition library configuration data.

4.2.1 TOUCH_DAT_T

This data structure is used by the host application to pass (redirect) the received touch event data to the gesture library.

Table 1. TOUCH_DAT_T

Field	Type	Description
index	uint16_t	Position index number.
x	uint8_t	X coordinate.
y	uint8_t	Y coordinate.

4.2.2 GESTURE_T

The Gesture data structure is defined by the library and exported to the host application, so that it can provide a pointer to the training set memory location and the number of gestures inside that training set (maximum of 10).

Table 2. GESTURE_T

Field	Type	Description
version	uint16_t	Gesture library version number.
nr_of_gestures	uint8_t	Indicates the number of gestures inside the training set. Maximum number is 10.
trs	uint8_t *	Pointer to the pre-loaded training set (can be in flash or SRAM).

The Gesture Recognition library exports a variable of this data type so that the host application can modify the required fields before using it.

```
extern GESTURE_T gr;
```

4.3 Public functions

This section describes the public functions available in the Gesture Recognition library and their usage. Only two callable functions are available that makes the application interface easy to use.

One function is called once to initialize and the other function does the actual touch gesture recognition task.

4.3.1 Gesture_Init

This function is used to initialize the gesture recognition library.

```
void Gesture_Init(GESTURE_T *gr);
```

Argument	Type	Description
gr	GESTURE_T *	Pointer to the gesture library configuration data structure.

4.3.2 Gesture_Task

This function is called every time the touch solution library generates a touch event. The host application passes the touch data by calling this function.

It runs the recognition algorithm to identify the same pattern match with reference to the recognized gesture.

On return this function notifies the host application whether or not it found a match between the latest touch input data and the pre-loaded training set gestures.

```
uint8_t Gesture_Task(TOUCH_DAT_T *td);
```

Argument	Type	Description
td	TOUCH_DAT_T *	Pointer to the touch event data structure.

Return	Value	Description
uint8_t	0 – 9	Match: number of the recognized gesture.
	0xFF	No-match: no gesture pattern recognized.

4.4 Start-up/Usage sequence

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

1. Make sure the touch solution library is included and running in your project.

   Note: See "LPC82x Touch Solution" for more details.

2. Make sure a valid **Training Set** is loaded into flash or SRAM (see section 2).

3. Initialize the gesture configuration parameters (GESTURE_T) and pass them to the library. Example:

```
gr.nr_of_gestures = 6;          // 6 gestures in loaded training set
gr.trs = (uint8_t *)TRS_ADDR;   // pointer to training set location
Gesture_Init();
```

4. Add Gesture Task call to the touch event call back function to pass the touch data, example:

```
void TouchEventHandler(uint8_t event, uint8_t buf[4])
{
  TOUCH_DAT_T td;               // define touch event data structure
  uint8_t     i = 0xFF;

   td.index = (buf[0] << 8) + buf[1];
   td.x     = buf[2];
   td.y     = buf[3];
   i = Gesture_Task(&td);       // pass touch data to Gesture Analysis module
                                // function returns a recognized gesture nr
                                // or 0xFF if no gesture recognized
}
```

5. Handle a recognized gesture (this is application dependent). Example (following previous step):

```
if (i != 0xFF)
{
    LCD_Send(i);                // display recognized gesture number
}
```

Note: See "AN11666: Gesture Recognition Application Note" for more details on example projects using the Gesture Recognition library.

5. Microcontroller Resources

5.1 Memory footprint

The current Gesture Recognition library memory footprint of flash is just over 1.3 kB and SRAM usage is around 600 bytes.

Compiled with Keil V5.10:

```
-------------------------------------------------------------------
Code        RW Data      ZI Data      Library Name
1352        12           592          Gesture_lib_v0100.lib
```

Compiled with LPCXpresso V7.7:

```
-------------------------------------------------------------------
text        data         bss          Library Name
1384         8           596           libGesture_lib_v0100.a
```

5.2 Training set size

The application firmware downloads and stores the training set into either flash, RAM, or EEPROM (if available). Every recorded Gesture inside the training set takes 256 bytes, that is, 64 samples x 4 bytes (Xmin, Xmax, Ymin and Ymax).

For example, a training set that contains 8 gestures requires 8 x 256 bytes = 2048 bytes of memory.

5.3 Performance indicators and constraints

Currently, the maximum number of gestures inside a training set is limited to 10.

Assuming the training set contains 10 gestures, running the complete gesture recognition algorithm to find a match or no-match takes an average of 1.000.000 CPU clock cycles. Running at 24 MHz takes approximately 42 milliseconds.

6. Legal information

6.1 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.

6.2 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.

6.3 Trademarks

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

7. 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: 09 June 2015
Document identifier: UM10897

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

NXP:

OM13081 OM13081UL