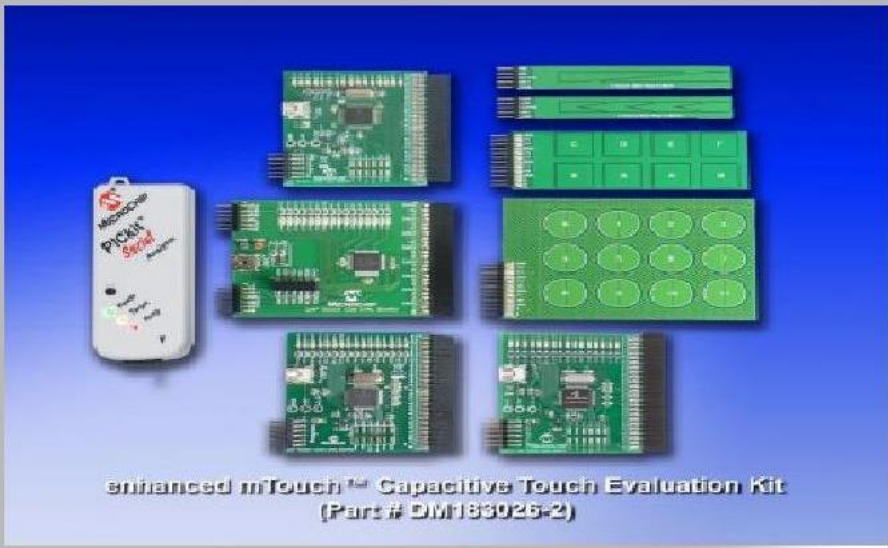


mTouch Cap Library Help

Table of Contents

1 Introduction

Introduction

The Capacitive mTouch™ Software Library provides the API's to develop capacitive touch applications using the Charge Time Measurement Unit (CTMU) and Capacitive Voltage Divider (CVD) technique on PIC18F, PIC24F, PIC24H and dsPIC33 Microcontrollers (MCUs).

The software stack is developed using ‘C’ language and can be compiled by Microchip's C18 , XC8, PICC18, XC16 and C30 compilers for PIC18, PIC24F, PIC24H and dsPIC33 Microcontrollers.

Users of the mTouch™ Software Library can select the PIC microcontroller used for the application and configure the CTMU or CVD Demos as required for the application. The API's helps the user to integrate the mTouch Capacitive Library with the end application. This library is also designed to operate with other libraries developed by Microchip.

The CTMU has a constant current source that can be used for relative capacitance measurement, absolute capacitance measurement and accurate time measurement. This library will use the relative capacitance measurement for capacitive touch sensing application. Refer to the CTMU Family Reference Manual (DS39724) for more details of CTMU.

The CVD technique resides in successive charging and discharging cycles of ADC sample and holds capacitor and the external capacity of the sensor, while measuring the voltage left on the sample and hold capacitor after each cycle. This library contains the implementation of the CVD technique. Refer to the Capacitive Touch Using Only ADC (CVD) – AN1298 for more details.

The Capacitive mTouch™ Software library is also implemented for PIC16F and PIC18F CVD Framework.

The Help file for PIC16F and PIC18F CVD Framework is available in the following location:

....\Microchip\Help\mTouch CVD Framework Documentation.

Hardware Setup :

The PIC18F, PIC24F and PIC24H Enhanced Capacitive Touch Evaluation kit (DM183026-2) is used for demonstrating the Capacitive mTouch™ Software Library functionality.

2 Software License Agreement

MICROCHIP IS WILLING TO LICENSE THE ACCOMPANYING SOFTWARE AND DOCUMENTATION TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, your heirs, successors and assigns ("Licensee") and Microchip Technology Incorporated, a Delaware corporation, with a principal place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology (Barbados) II Incorporated (collectively, "Microchip") for the accompanying Microchip software including, but not limited to, Graphics Library Software, IrDA Stack Software, MCHPFSUSB Stack Software, Memory Disk Drive File System Software, mTouch(TM) Capacitive Library Software, Smart Card Library Software, TCP/IP Stack Software, MiWi(TM) DE Software, Security Package Software, and/or any PC programs and any updates thereto (collectively, the "Software"), and accompanying documentation, including images and any other graphic resources provided by Microchip ("Documentation").

1. Definitions. As used in this Agreement, the following capitalized terms will have the meanings defined below:

a. "Microchip Products" means Microchip microcontrollers and Microchip digital signal controllers.

b. "Licensee Products" means Licensee products that use or incorporate Microchip Products.

c. "Object Code" means the Software computer programming code that is in binary form (including related documentation, if any), and error corrections, improvements, modifications, and updates.

d. "Source Code" means the Software computer programming code that may be printed out or displayed in human readable form (including related programmer comments and documentation, if any), and error corrections, improvements, modifications, and updates.

e. "Third Party" means Licensee’s agents, representatives, consultants, clients, customers, or contract manufacturers.

f. "Third Party Products" means Third Party products that use or incorporate Microchip Products.

2. Software License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to:

a. use the Software in connection with Licensee Products and/or Third Party Products;

b. if Source Code is provided, modify the Software; provided that Licensee clearly notifies Third Parties regarding the source of such modifications;

c. distribute the Software to Third Parties for use in Third Party Products, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept") and this Agreement accompanies such distribution;

d. sublicense to a Third Party to use the Software, so long as such Third Party agrees to be bound by this Agreement (in writing or by "click to accept");

e. with respect to the TCP/IP Stack Software, Licensee may port the ENC28J60.c, ENC28J60.h, ENCX24J600.c, and ENCX24J600.h driver source files to a non-Microchip Product used in conjunction with a Microchip ethernet controller;

f. with respect to the MiWi (TM) DE Software, Licensee may only exercise its rights when the Software is embedded on a Microchip Product and used with a Microchip radio frequency transceiver or UBEC UZ2400 radio frequency transceiver which are integrated into Licensee Products or Third Party Products.

For purposes of clarity, Licensee may NOT embed the Software on a non-Microchip Product, except as described in this

Section.

3. Documentation License Grant. Microchip grants strictly to Licensee a non-exclusive, non-transferable, worldwide license to use the Documentation in support of Licensee's authorized use of the Software

4. Third Party Requirements. Licensee acknowledges that it is Licensee’s responsibility to comply with any third party license terms or requirements applicable to the use of such third party software, specifications, systems, or tools. This includes, by way of example but not as a limitation, any standards setting organizations requirements and, particularly with respect to the Security Package Software, local encryption laws and requirements. Microchip is not responsible and will not be held responsible in any manner for Licensee’s failure to comply with such applicable terms or requirements.

5. Open Source Components. Notwithstanding the license grant in Section 1 above, Licensee further acknowledges that certain components of the Software may be covered by so-called "open source" software licenses ("Open Source Components"). Open Source Components means any software licenses approved as open source licenses by the Open Source Initiative or any substantially similar licenses, including without limitation any license that, as a condition of distribution of the software licensed under such license, requires that the distributor make the software available in source code format. To the extent required by the licenses covering Open Source Components, the terms of such license will apply in lieu of the terms of this Agreement. To the extent the terms of the licenses applicable to Open Source Components prohibit any of the restrictions in this Agreement with respect to such Open Source Components, such restrictions will not apply to such Open Source Component.

6. Licensee Obligations. Licensee will not: (a) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (b) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products, Licensee Products or Third Party Products; or (c) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation. Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless from and against any and all claims, costs, damages, expenses (including reasonable attorney's fees), liabilities, and losses, including without limitation: (x) any claims directly or indirectly arising from or related to the use, modification, disclosure or distribution of the Software, Documentation, or any intellectual property rights related thereto; (y) the use, sale and distribution of Licensee Products or Third Party Products; and (z) breach of this Agreement.

7. Confidentiality. Licensee agrees that the Software (including but not limited to the Source Code, Object Code and library files) and its derivatives, Documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are proprietary information belonging to Microchip and its licensors ("Proprietary Information"). Except as expressly and unambiguously allowed herein, Licensee will hold in confidence and not use or disclose any Proprietary Information and will similarly bind its employees and Third Party(ies) in writing. Proprietary Information will not include information that: (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by the receiving party independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If Licensee is required to disclose Proprietary Information by law, court order, or government agency, License will give Microchip prompt notice of such requirement in order to allow Microchip to object or limit such disclosure. Licensee agrees that the provisions of this Agreement regarding unauthorized use and nondisclosure of the Software, Documentation and related Proprietary Rights are necessary to protect the legitimate business interests of Microchip and its licensors and that monetary damage alone cannot adequately compensate Microchip or its licensors if such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that Licensee or Third Party has breached or violated such provision then Microchip will have the right to injunctive relief, without the requirement for the posting of a bond, in addition to all other remedies at law or in equity.

8. Ownership of Proprietary Rights. Microchip and its licensors retain all right, title and interest in and to the Software and Documentation including, but not limited to all patent, copyright, trade secret and other intellectual property rights in the Software, Documentation, and underlying technology and all copies and derivative works thereof (by whomever produced). Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in this Agreement.

9. Termination of Agreement. Without prejudice to any other rights, this Agreement terminates immediately, without notice by Microchip, upon a failure by Licensee or Third Party to comply with any provision of this Agreement. Upon termination, Licensee and Third Party will immediately stop using the Software, Documentation, and derivatives thereof, and immediately

destroy all such copies.

10. Warranty Disclaimers. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. MICROCHIP AND ITS LICENSORS ASSUME NO RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-FREE. MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY DEFECTS IN THE SOFTWARE.

11. Limited Liability. IN NO EVENT WILL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER ANY LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Microchip and its licensors for damages hereunder will in no event exceed $1000 or the amount Licensee paid Microchip for the Software and Documentation, whichever is greater. Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement.

12. General. THIS AGREEMENT WILL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. Licensee agrees that any disputes arising out of or related to this Agreement, Software or Documentation will be brought exclusively in either the U.S. District Court for the District of Arizona, Phoenix Division, or the Superior Court of Arizona located in Maricopa County, Arizona. This Agreement will constitute the entire agreement between the parties with respect to the subject matter hereof. It will not be modified except by a written agreement signed by an authorized representative of Microchip. If any provision of this Agreement will be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision will be limited or eliminated to the minimum extent necessary so that this Agreement will otherwise remain in full force and effect and enforceable. No waiver of any breach of any provision of this Agreement will constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver will be effective unless made in writing and signed by an authorized representative of the waiving party. Licensee agrees to comply with all import and export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority. The indemnities, obligations of confidentiality, and limitations on liability described herein, and any right of action for breach of this Agreement prior to termination, will survive any termination of this Agreement. Any prohibited assignment will be null and void. Use, duplication or disclosure by the United States Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause of FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199.

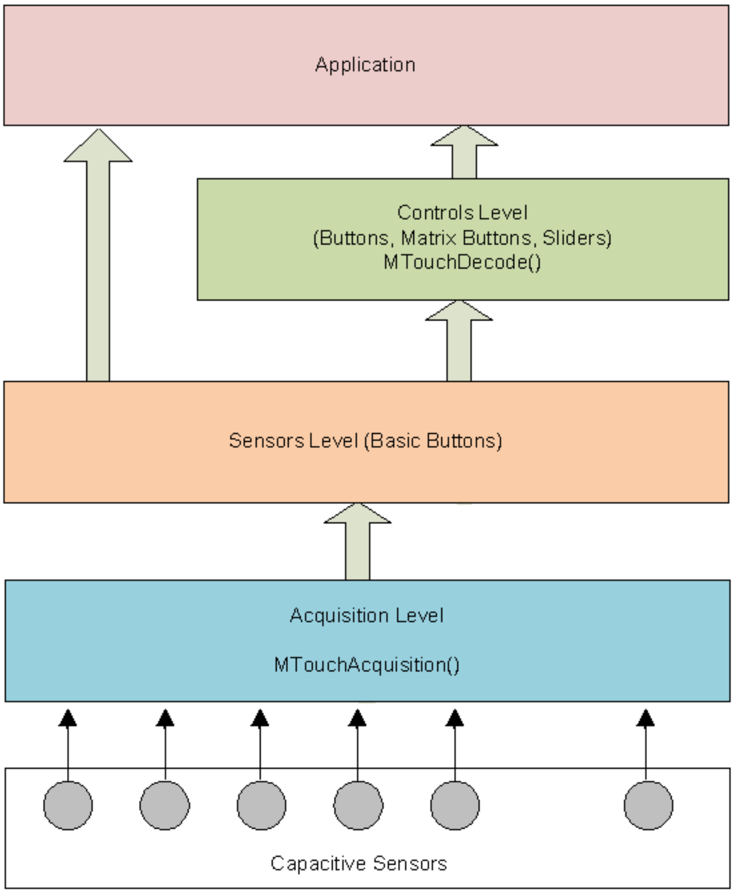
If Licensee has any questions about this Agreement, please write to Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA. ATTN: Marketing.

License Rev. No. 05-012412

3 Overview

This document describes capacitive touch library for PIC18F, PIC24F, PIC24H and dsPIC33 family of Microcontrollers.

The library has three levels: acquisition, sensors and controls. The acquisition level gets raw samples from the sensors. The sensors level allows initialization and press/release events detection for all sensors in the system. The controls level gets information from sensors level and contains implementation of different capacitive controls such as buttons, matrix buttons and sliders. Also there is a debug module helping adjustment of the sensors' settings.



3.1 Acquisition Level

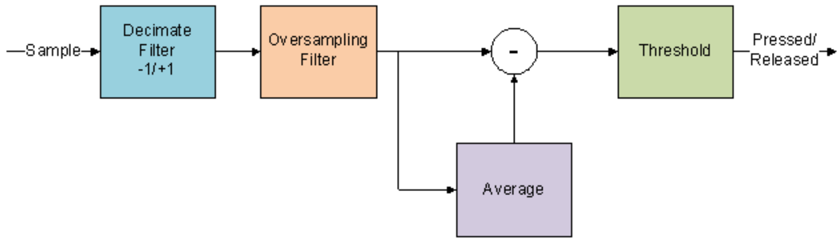
The acquisition level of the stack abstracts the hardware and acquires samples for the capacitive touch sensing.To perform the acquisition the MTouchAcquisition(…) function should be called periodically in the application. Depending on the hardware modules used on a PIC Microcontroller, the library supports two acquisition methods: capacitive voltage divider (CVD) and charging of the sensors using constant current source (CTMU).

- **CVD**: PIC Microcontroller's ADC holding capacitor (C_{hold}) is used for the measurements. Initially the capacitive sensor (C_{sensor}) is disconnected from C_{hold}. C_{hold} should be charged to Vdd and C_{sensor} should be discharged. Then both capacitors are connected together to divide a charge between them. Capacitance of C_{hold} is constant so the result voltage will depend on capacitance of C_{sensor}. When sensor is touched the capacitance is increased and voltage is decreased. When the sensor is released the capacitance is decreased and voltage is increased. The minimum number of sensors required for this acquisition method is 2.

- **CTMU**: If the capacitive sensor will be charged by a constant current source during a constant time then the voltage on the sensor after the charge will depend on the capacitance. When sensor is touched the capacitance is increased and voltage is decreased. When the sensor is released the capacitance is decreased and voltage is increased.

3.2 Sensors Level

To improve noise immunity the samples from sensors go through two filters: decimate and oversampling. If the sample is bigger than decimate filter value then the filter value is incremented otherwise it is decremented. Data from decimate filter go to oversampling filter. The oversampling filter performs averaging. Output from filters is used to form a long time average. Difference between value from filters and this average is used for comparison with threshold to detect state of the sensor.



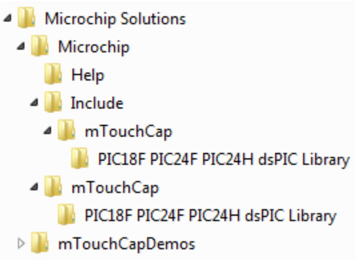
MTouchGetSensorState(…) function returns a current state of the sensor. The sensor acts as a basic button which can have two states: pressed or released.

3.3 Controls Level

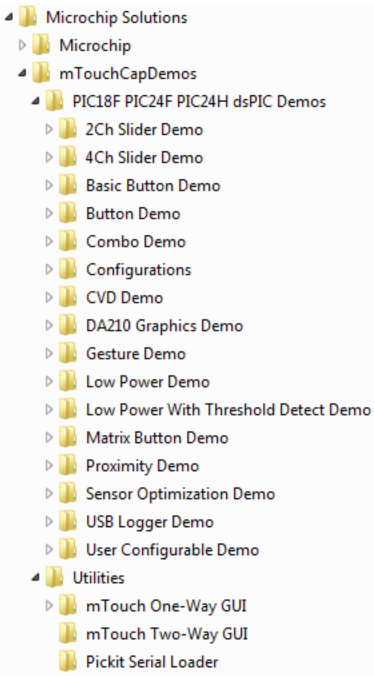
The control level contains implementations of the more complex capacitive controls. To decode states of the controls the MTouchDecode(…) function should be called periodically in the application. Some examples of these are matrix keys, sliders etc.

4 Getting Started

The folder structure of mTouchCap Software Library is shown below:



The folder structure of mTouchCapDemos are as follows:



You can add code and modules to the demo sub directories that will use and interact with the library. For example, you could add a folder named "Your Applications Directory" to the mTouchCapDemos folder that contains your application source code. The library specific folders are the following:

- The ..\Microchip folder will contain the library components.
- The Help sub-folder under ..\Microchip folder will contain this document (mTouch Cap Library Help.chm file).
- The ..\mTouchCap sub-folder under the ..\Microchip folder is where the C files, documentation related to mTouch stack are located.
- The ..\mTouchCap sub-folder under the Include folder is where the Header files related to the mTouch stack are located.

4.1 mTouch Library Files

The following files should be included in the project:

Common	
Compiler.h	Contains compiler specific definitions.
GenericTypeDefs.h	Standard MLA types definitions.
mTouch.h	This file joins all definitions, macros and functions prototypes related to mTouch library. To use the library API only this header can be included in the application code.
mTouchConfig.h	mTouch library configurations.
Acquisition	
mTouchAcquisitionMCU8.h , mTouchAcquisitionMCU16.h	Acquisition macros defining timing, CTMU and ADC operation.
mTouchAcquistion.c	Acquisition CVD and CTMU routines.
Sensors	
mTouchSensor.h , mTouchSensor.c	Sensors’ filtration and decoding. It provides basic button functionality.
Controls	
mTouchControl.h , mTouchControl.c	Common definitions and functions for all controls.
mTouchButton.h , mTouchButton.c	Definitions and functions for the button controls with different decoding methods.
mTouchMatrixButton.h, mTouchMatrixButton.c	Definitions and functions for the matrix button controls.
mTouch2ChSlider.h , mTouch2ChSlider.c	Definitions and functions for the 2 channel slider controls.

mTouch4ChSlider.h , mTouch4ChSlider.c	Definitions and functions for the 4 channel slider controls.
Debug	
mTouchDebug.h , mTouchDebug.c	This module contains means to log information from sensors and to calculate the optimal CTMU current, charge delay for the CTMU acquisition and press detection threshold.

4.2 mTouch Library Configuration

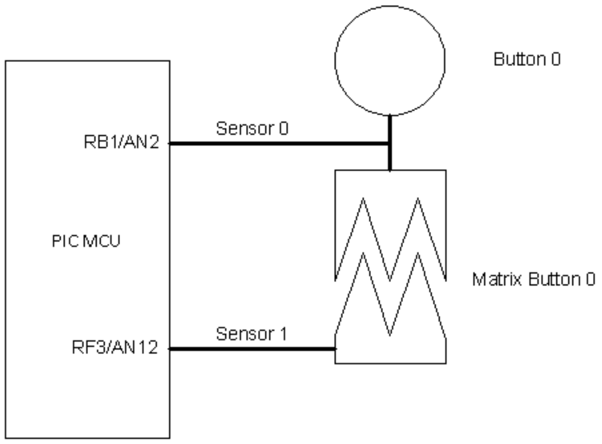
The following mTouch Library settings should be defined in mTouchConfig.h file:

MTOUCH_USE_10_BITS_ADC, MTOUCH_USE_12_BITS_ADC	ADC type (see PIC Microcontroller datasheet). Select (uncomment) only one: 10bits or 12bits.
MTOUCH_CTMU_HAS_CTMUCON2_REG, MTOUCH_CTMU_HAS_NO_CTMUCON2_REG	CTMU type (see PIC Microcontroller datasheet). Select (uncomment) only one: with CTMUCON2 register or without CTMUCON2 register.
MTOUCH_USE_CTMU, MTOUCH_USE_CVD	Acquisition method. Select (uncomment) only one: CTMU or CVD.
MTOUCH_DEBUG	Debugging. Uncomment to enable debug functions.
MTOUCH_SENSORS_NUMBER	Number of sensors (analog inputs connected to sensors). The minimum number of sensors required for CVD acquisition method is 2.
MTOUCH_BUTTONS_NUMBER	Number of button controls.
MTOUCH_MATRIXBUTTONS_NUMBER	Number of matrix button controls.
MTOUCH_2CHSLIDERS_NUMBER	Number of 2 channels slider controls.
MTOUCH_4CHSLIDERS_NUMBER	Number of 4 channels slider controls.
AVG_SLIDER_VALUE	The slider value is filtered. When the slider value is updated, this factor determines what weight is given in the calculation. Can be set to 0(100% new value/no averaging), 1(50% of new value), 2(25% of new value), 3(12.5% of new value) and so on.

MTOUCH_DEFAULT_CHARGE_DELAY	Default CTMU charge delay settings. This value is used in MTouchSetSensor(...) when "chargeDelay" is set to -1. Use MTouchDebugDelay(…) function to calculate CTMU charge delay value (to charge the sensor to about 75% of AVdd). If adjustment of this parameter gives a value less than 4 decrease CTMU current with MTOUCH_CTMU_CURRENT.
MTOUCH_DEFAULT_THRESHOLD	Default threshold for press event detection. This value is used when "threshold" is set to -1 in MTouchSetSensor(...) call. The optimal threshold value is about 20% of sensor signal (delta) amplitude. The sensor signal amplitude can be determined using debug module.
MTOUCH_DEFAULT_OVERSAMPLING	Default number of acquisitions for one sample of the sensor. This value is used when "oversampling" is set to -1 in MTouchSetSensor(...) call. The oversampling factor should be selected to maximize the amplitude of the signal from sensor and to provide fast enough response time (see “Acquisition time for one sensor” chapter for the response time estimation).
POWER_UP_SAMPLES	This is the number of total scans that should be taken for the sensor before it will be considered initialized. Allowable range is from 1 to 65535.
DEBOUNCE_COUNT	Number of consecutive scans a sensor must be seen as pressed or released before an updated state is declared. Allowable range is from 1 to 255.
MCONTROL_REPEAT_INITIAL_DELAY	Initial delay for the control DECODE_PRESS_REPEAT decoding method. Defines how many times the control decoding must be done before the control starts repeating CONTROL_PRESS/CONTROL_RELEASE events.Allowable range is from 1 to 65535.
MCONTROL_REPEAT_DELAY	Delay between CONTROL_PRESS/CONTROL_RELEASE events for the control DECODE_PRESS_REPEAT decoding method. Allowable range is from 1 to 65535.
AVG_UPDATE	When the average updates itself using a new sample, this value determines what weight is given to the new sample in the calculation of the new average. The new sample will have a weight of 1/AVG_UPDATE in the average calculation. Can be set to 2,4,8 or 16.
AVG_RATE_RELEASED	The update rate of the sensors' average values when sensor is released. Allowable Range from 1 to 65535.
AVG_RATE_PRESSED	The update rate of the sensors' average values when sensor is pressed. Allowable Range from 1 to 65535.
MTOUCH_CTMU_CURRENT	CTMU current settings. Bits 1-0 select the current source range (IRNG) and bits 7-2 select current trim value (ITRIM, signed). The current must be selected such way to get CTMU charge delay more than 4 (see MTOUCH_DEFAULT_CHARGE_DELAY).

4.3 Using API

Let's consider an application example for the following hardware configuration:



In the system there are 2 sensors, 1 button and 1 matrix button. Thus in mTouchConfig.h MTOUCH_SENSORS_NUMBER must be set to 2, MTOUCH_BUTTONS_NUMBER and MTOUCH_MATRIXBUTTONS_NUMBER must be set to 1. All IOs connected to sensors must be set as ANALOG in the application (see PCFGx, ANSx or ANSELx registers description in PIC Microcontroller datasheet).

The program should be started from MTouchInit(…) function call to initialize the mTouch Library. Then for each sensor in the system the sensors parameters must be set with MTouchSetSensor(…) function calls. From this point the mTouch library has all information about sensors and the application can get samples from them by calling MTouchAcquisition(…) function periodically. It can be done with a timer interrupt.

All controls in the application also must be initialized. In this example we have button and matrix button. Functions MTouchSetButton(…) and MTouchSetMatrixButton(…)assign sensors for these controls and define decoding methods. To get states of controls the MTouchDecode() must be run periodically. For this example the application code can be:

```
// Header file for mTouch library API.
#include "mTouch.h"

void main(void)
{
......................................................

    // STEP 1
    // mTouch library initialization.
    MTouchInit();

    // STEP 2
```

```
// Sensors initialization. All sensors must be initialized
// see MTOUCH_SENSORS_NUMBER in mTouchConfig.h).
// PLEASE READ "SENSOR OPTIMIZATION (DEBUG MODULE)" CHAPTER
// TO SELECT OPTIMAL PARAMETERS.

// Sensor #0 is connected to RB1/AN2 pin
MTouchSetSensor(0,      // sensor number
        &TRISB, // port B
        &LATB,
        1,      // IO bit number
        2,      // analog channel number
        -1,     // press detection threshold by default
           // (see MTOUCH_DEFAULT_THRESHOLD in mTouchConfig.h)
        -1,     // oversampling by default
          //(see MTOUCH_DEFAULT_OVERSAMPLING in mTouchConfig.h)
        -1 );   // CTMU charge delay by default
          //(see MTOUCH_DEFAULT_CHARGE_DELAY in mTouchConfig.h,
          // not used for CVD acquisition)

// Sensor #1 is connected to RF3/AN12 pin
MTouchSetSensor(1,      // sensor number
        &TRISF, // port F
        &LATF,
        3,      // IO bit number
        12,     // analog channel number
        -1,     // press detection threshold by default
           // (see MTOUCH_DEFAULT_THRESHOLD in mTouchConfig.h)
        -1,     // oversampling by default
          //(see MTOUCH_DEFAULT_OVERSAMPLING in mTouchConfig.h)
        -1 );   // CTMU charge delay by default
          //(see MTOUCH_DEFAULT_CHARGE_DELAY in mTouchConfig.h,
          // not used for CVD acquisition)

// STEP 3
// Buttons initialization. All buttons must be initialized
//(see MTOUCH_BUTTONS_NUMBER and MTOUCH_MATRIXBUTTONS_NUMBER in
// mTouchConfig.h).
// The button #0 is connected to sensor # 0
```

```
    MTouchSetButton(0,          // button number
            0,          // sensor number
            DECODE_TOGGLE); // decode method

    // The matrix button #0 is connected to sensor # 0 and sensor # 1
    MTouchSetMatrixButton(0,      // button number
            0,          // first sensor number
            1,          // second sensor number
            DECODE_PRESS_RELEASE); // decode method



     // STEP 4
    // Timer interrupt initialization to call mTouchAcquisition(...)
    // pereodically.
    TimerInterruptInitialization();



    while(1)
    {

      // STEP 4
      // Decode all controls periodically.
      MTouchDecode();

      // STEP 5
      // Get current states of the buttons.
      Led_ALLOff();
      // button #0
      if(MTouchGetButtonState(0) == CONTROL_PRESSED) { Led0On(); }
      // matrix button #0
      if(MTouchGetMatrixButtonState(0) == CONTROL_PRESSED) { Led1On(); }
    }

  }

  // Timer interrupt service routine.
  void __attribute__((interrupt, shadow, auto_psv)) _T4Interrupt(void)
```

```
{

  // STEP 6
   // Scan sensors periodically.
   MTouchAcquisition();

    // Clear timer interrupt flag.
TMR4 = 0; IFS1bits.T4IF = 0;
}
```

5 Sensor Optimization (Debug Module)

During initialization the application must pass a few parameters to MTouchSetSensor(…) for each sensor. This chapter describes how to select optimal values for a press detection threshold, oversampling factor, CTMU current and charge delay. If these parameters are not optimized then it can influence on the sensors' performance especially in a noisy environment. The optimization of sensors can be divided in a few steps:

- Step 1. Optimal CTMU current selection (MTOUCH_CTMU_CURRENT parameter in mTouchConfig.h).
- Step 2. Optimal CTMU charge delay selection.
- Step 3. Optimal oversampling factor selection.
- Step 4. Optimal press detection threshold selection.

5.1 Step 1. Optimal CTMU current selection

To achieve the maximum of sensitivity the sensors must be charged to the voltage level about 75% of AVdd . The rounding error depends on the charge delay parameter.The rounding error in percentage is (100/CTMU charge delay) of AVdd. The recommended minimum value for the CTMU charge delay is 8 (default charge delay) .This provides charge to the optimal level with rounding error about +-12.5% of AVdd. MTouchDebugCurrent(…) function returns the CTMU current source settings when the optimal charge delay value is 8. Assign this value to MTOUCH_CTMU_CURRENT parameter in mTouchConfig.h.

5.2 Step 2. Optimal CTMU charge delay selection

To achieve the maximum of sensitivity the CTMU charge delay must be set to charge the sensor to the voltage level about 75% of AVdd . This optimal delay value can be calculated with MTouchDebugDelay(…) function. The calculated optimal value should be passed for initialization to MTouchSetSensor(…).

5.3 Step 3. Optimal oversampling factor selection

The oversampling factor should be set as big as possible to get maximum of signal amplitude and to increase noise immunity. But this parameter is limited by the sensors response time. See time requirements for one acquisition to estimate how many samples can be used for one sample. The calculated optimal value should be passed for initialization to MTouchSetSensor(…).

5.4 Step 4. Optimal press detection threshold selection

The big noise can decrease sensitivity more than in 4 times. So the recommended value for the press detection threshold is 1/8th of the sensor signal amplitude (delta). To calculate the optimal threshold value MTouchDebugThreshold(…) function can be used. It waits for the user presses the sensor and returns the optimal threshold value as 1/8th of the detected amplitude. The calculated optimal value should be passed for initialization to MTouchSetSensor(…).

5.5 Optimization example

For the optimal parameters calculation the compiler optimization must be set to the required level. If the version of the compiler or optimization level is changed then the optimization process must be repeated again. The Debugger Watch Window can be used to see the result of the optimization. When the compiler optimization is on some variables can be optimized out and can be not available for the debugger. All variables displayed in the Watch Window must be global and declared as volatile. The code below calculates optimal parameters:

```
// Header file for mTouch library API
#include "mTouch.h"

//////////////////////////////////////////////////////
//                GLOBAL VARIABLES
//////////////////////////////////////////////////////
// This structure will contain the optimal CTMU current.
volatile DEBUGCURRENT* pOptimalCurrent;
// This structure will contain the optimal CTMU charge delay.
volatile DEBUGDELAY* pOptimalDelay;
// This variable will contain the optimal threshold.
volatile UINT16     optimalThreshold;

void main(void)
{
   // STEP 1
   // mTouch library initialization.
```

```
MTouchInit();

// STEP 2
// Sensors initialization. All sensors must be initialized
// see MTOUCH_SENSORS_NUMBER in mTouchConfig.h).
// Set default parameters.

// Sensor #0 is connected to RB0/AN0 pin
MTouchSetSensor(0,      // sensor number
            &TRISB, // port B
            &LATB,
            0,      // IO bit number
            0,      // analog channel number
            -1,     // press detection threshold by default
                   // (see MTOUCH_DEFAULT_THRESHOLD in
                   // mTouchConfig.h)
            -1,     // oversampling by default
                   //(see MTOUCH_DEFAULT_OVERSAMPLING in
                   // mTouchConfig.h)
            -1 );   // CTMU charge delay by default
                   //(see MTOUCH_DEFAULT_CHARGE_DELAY in
                   // mTouchConfig.h,
                   // not used for CVD acquisition)


// STEP 3
// MTouchDebugCurrent(sensorNumber) function calculates the optimal CTMU
// current value (optimal CTMU charge delay will be about 8).
// This will be a final value for MTOUCH_CTMU_CURRENT parameter in
// mTouchConfig.h.
// Before measurement set MTOUCH_CTMU_CURRENT to 0x01.
// Sensor #0 is tested.
pOptimalCurrent = MTouchDebugCurrent(0);
// Set adjusted CTMU current value.
MTouchSetCTMUCurrent(pOptimalCurrent->current);

// STEP 4
// MTouchDebugDelay(sensorNumber) function calculates the optimal
```

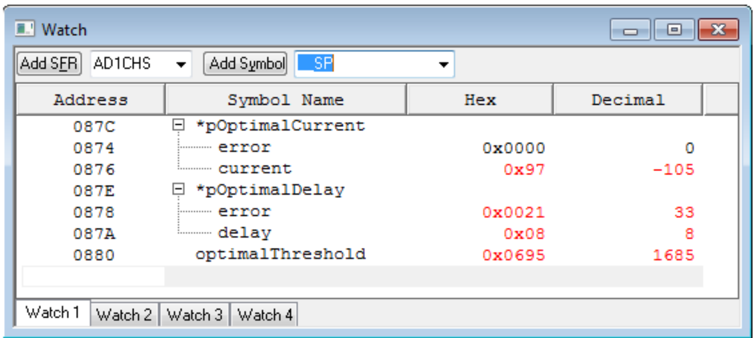
5

```
    // CTMU charge delay value to provide charging of sensor to
    // about 75% of AVdd.
    // Optimal delay for sensor #0.
    pOptimalDelay = MTouchDebugDelay(0);
    // Set adjusted CTMU charge delay value for the sensor # 0
    MTouchSetChargeDelay(0, pOptimalDelay->delay);

    // STEP 5
    // MTouchDebugThreshold(sensorNumber) function calculates the optimal
    // press detection threshold value. It waits for the sensor press event
    // from user to measure maximum signal amplitude (delta).
    // Optimal threshold for sensor #0.
    optimalThreshold = MTouchDebugThreshold(0);
    // Set adjusted threshold value for the sensor # 0
    MTouchSetThreshold(0, optimalThreshold);

    // STEP 6
    // Put break point here. Use Watch Window to see
    // pOptimalCurrent->current, pOptimalDelay->delay and optimalThreshold
    // values.
    while(1);
}
```

After the code execution the result in the Debugger Watch Window can be:



Address	Symbol Name	Hex	Decimal
087C	*pOptimalCurrent		
0874	error	0x0000	0
0876	current	0x97	-105
087E	*pOptimalDelay		
0878	error	0x0021	33
087A	delay	0x08	8
0880	optimalThreshold	0x0695	1685

pOptimalCurrent->error field shows an offset of the CTMU charge delay from nominal value (8). pOptimalDelay->error fiels shows an offset of the sample for the adjusted charge delay from the nominal value (should be less than 128 for 10-bit ADC and less than 512 for 12-bit ADC).

5.6 mTouch GUI

The library has capability to stream data from sensors to a text log file or to a special graphics tool – mTouch GUI. To use this functionality the LogChar(…) function must be implemented in the application. Usually this function should transmit a byte via PIC UART. The mTouch GUI utility is located in "**….\Microchip Solutions\mTouchCapDemos\Utilities\mTouch One-Way GUI**" folder. "**mTouch Library GUI Help.chm**" file in this folder contains all required information about setup, configuration and usage.

6 Sharing ADC between mTouch Library and Other Tasks

Often the ADC must be used for many different tasks. To share the ADC between mTouch Library and these tasks the state machine can be used. The code example below shows possible implementation for the system where ADC is shared between Touch Screen, mTouch Buttons and Battery Level Measurement. Battery_ADCInit() and TouchScreen_ADCInit() functions configure ADC and Vref as needed for these modules.

```
// This variable holds the state machine current state.
volatile int current_state = STATE_TOUCH_SCREEN;

// The state machine is run by the timer interrupt.
void __attribute__((interrupt, shadow, auto_psv)) _T4Interrupt(void)
{
switch (current_state) // The state machine main switch start.
 {

case  STATE_TOUCH_SCREEN:
   // If touch screen scan is finished then switch to mTouch Buttons task.
  // The TouchScreenDetectPosition() function runs the touch screen state machine.
  // A few calls of TouchScreenDetectPosition() are required to detect a touch on the touch screen.
  // When the position is detected this function returns non-zero.
if (TouchScreenDetectPosition()!= 0)
 {
  // Initialize ACD for mTouch Buttons.
 MTouchInit();
 current_state = STATE_MTOUCH_BUTTONS;
 }
break ;

case  STATE_MTOUCH_BUTTONS:
  // Get data from capacitive buttons.
 MTouchAcquisition();
  // Initialize ACD for Battery Level Measurement.
 Battery_ADCInit();
 current_state = STATE_BATTERY_LEVEL;
```

```
break ;

case  STATE_BATTERY_LEVEL:
 // BatteryLevelDetect() measures the battery level.
 BatteryLevelDetect();
 // Initialize ACD for touch screen.
 TouchScreen_ADCInit();
 current_state = STATE_TOUCH_SCREEN;
break ;

 } // The state machine main switch end.

 // Clear timer interrupt flag.
 TMR4 =0; IFS1bits.T4IF = 0;

} // End of timer interrupt.
```

7 Code and RAM Memories Size

In this section the required memory resources are listed.

7.1 RAM

Here is a list of RAM requirements per each sensor and control.

Object	Size less than
Sensor (basic button)	34 Bytes
Button (button with different decoding methods)	8 Bytes
Matrix Button	10 Bytes
2 Channel Slider	8 Bytes
4 Channel Slider	12 Bytes

7.2 Code

Here is a list of program memory requirements per each library module.

Module	Size for MPLAB C18 compiler less than	Size for MPLAB C30 compiler less than
CTMU Acquisition with Sensors (basic buttons)	2050 Bytes	1750 Bytes
CVD Acquisition with Sensors (basic buttons)	2700 Bytes	1850 Bytes
Button (buttons with different decoding methods)	680 Bytes	280 Bytes
Matrix Button	790 Bytes	330 Bytes
2 Channel Slider	1000 Bytes	280 Bytes

4 Channel Slider	1320 Bytes	430 Bytes

8 Acquisition Time for One Sensor

Acquisition method	Average time for MPLAB C18 compiler	Average time for MPLAB C30 compiler
CTMU	530 Instructions	160 Instructions
CVD	840 Instructions	290 Instructions

9 Demo Projects

The mTouch library demo projects are located in **"…\Microchip Solutions\mTouchCapDemos\PIC18F PIC24F PIC24H dsPIC Demos**". All hardware dependent settings, definitions, macros and functions for each demo project can be found in "**…\Microchip Solutions\mTouchCapDemos\PIC18F PIC24F PIC24H dsPIC Demos\Configurations**" folder. The **system.h** and **system.c** files in this folder contain the code specific for PIC Microcontroller device and development board used (such as configuration bits, ISRs, peripherals' initialization). There is one special demo project "**User Configurable Demo**". This project can be used as a start point for the custom application. This demo supports almost all PIC Microcontroller devices and all required mTouch library files are added to the project by default. The PIC Microcontroller device specific information for this demo project is placed in "**…\Microchip Solutions\mTouchCapDemos\PIC18F PIC24F PIC24H dsPIC Demos\Configurations\User_Board**" folder and **mTouchConfig.h** file.

Please read ReadMe.txt files in demo project folders to get more details about each demo.

10 API Reference

10.1 Common

In this section the common library functions are described .

10.1.1 void MTouchInit(void)

Description: this function initializes mTouch library.

10.1.2 MTouchSetCTMUCurrent(current)

Description: this macro sets CTMU current range and trim bits.

Parameters:

- **current** - current value. Bits 1-0 define the current source range (IRNG) and bits 7-2 define current trim value (ITRIM, signed).

10.2 Acquisition

In this section the acquisition level library functions are described .

10.2.1 MTouchAcquisition(void)

Description: this function performs an acquisition for all sensors (using CVD or CTMU).Contains decimate and oversampling filters. When oversampling is finished it decodes the sensor state. This function can be called periodically (for example by timer interrupt). The initialization should be done with MTouchInit() and MTouchSetSensor(...)functions.

10.3 Sensors

In this section the sensors level library functions are described.

10.3.1 void MTouchSetSensor(UINT8 sensorNumber, SFR tris, SFR lat, UINT8 ioBitNumber, UINT8 channelNumber, INT16 threshold, INT16 oversampling, INT8 chargeDelay)

Description: this function initializes a sensor. All sensors must be set before acquisition.

Parameters:

- **sensorNumber** - sensor number.
- **tris** - address of TRIS register for the sensor.
- **lat** - address of LAT register for the sensor.
- **ioBitNumber** - sensor IO bit number for LAT and TRIS registers.
- **channelNumber** - analog input number for the sensor.
- **threshold** - press detection threshold. Set this parameter to -1 to use default value MTOUCH_DEFAULT_THRESHOLD (mTouchConfig.h).
- **oversampling** - defines how many samples used for oversampling. Set this parameter to -1 to use default value MTOUCH_DEFAULT_OVERSAMPLING (mTouchConfig.h).
- **chargeDelay** - CTMU charge delay. Set this parameter to -1 to use default value MTOUCH_DEFAULT_CHARGE_DELAY (mTouchConfig.h).

10.3.2 MTouchSuspendSensor(sensorNumber)

Description: this macro excludes the sensor from scan. Use MTouchResumeSensor(...) to start the sensor scanning again.

Parameters:

- **sensorNumber** - sensor number.

10.3.3 MTouchResumeSensor(sensorNumber)

Description: this macro resumes the sensor scanning stopped by MTouchSuspendSensor(...).

Parameters:

- **sensorNumber** - sensor number.

10.3.4 MTouchSetChargeDelay(sensorNumber, delay)

Description: this macro sets charge delay value for sensor.

Parameters:

- **sensorNumber** - sensor number.
- **delay** - charge delay.

10.3.5 MTouchSetThreshold(sensorNumber, _threshold)

Description: this macro sets press detection threshold for sensor.

Parameters:

- **sensorNumber** - sensor number.
- **threshold** - press detection threshold.

10.3.6 MTouchSetOversampling(sensorNumber, oversampling)

Description: this macro sets oversampling factor for sensor.

Parameters:

- **sensorNumber** - sensor number.
- **oversampling** - oversampling factor.

10.3.7 MTouchGetSensorState(sensorNumber)

Description: this macro returns current state of sensor.

Parameters:

- **sensorNumber** - sensor number.

Returns: state of sensor (see MTOUCHSENSORSTATE enumeration in mTouchSensor.h).

10.3.8 MTouchInitializeSensor(sensorNumber)

Description: this macro starts the sensor's initialization.

Parameters:

- **sensorNumber** - sensor number.

10.4 Controls

In this section the controls level library functions are described.

10.4.1 void mTouchDecode(void)

Description: this function decodes states for all controls. It should be called periodically before reading of the controls states.

10.4.2 void MTouchSetButton(UINT8 buttonNumber, UINT8 sensorNumber, UINT8 decode)

Description: this function initializes button.

Parameters:

- **buttonNumber** - button number.
- **sensorNumber** - sensor number.
- **decode** - ORed combination of decode methods (see MTOUCHCONTROLDECODE union in mTouchControl.h).

10.4.3 MTouchGetButtonState(buttonNumber)

Description: this macro returns the button state.

Parameters:

- **buttonNumber** - button number.

Returns: button state flags (see MTOUCHCONTROLSTATE union in mTouchControl.h).

10.4.4 void MTouchSetMatrixButton(UINT8 buttonNumber, UINT8 ch1SensorNumber, UINT8 ch2SensorNumber, UINT8 decode)

Description: this function initializes matrix button.

Parameters:

- **buttonNumber** - button number.

- **ch1SensorNumber** - first sensor number (row or column).
- **ch2SensorNumber** - second sensor number (row or column).
- **decode** - ORed combination of decode methods (see MTOUCHCONTROLDECODE union in mTouchControl.h).

10.4.5 MTouchGetMatrixButtonState(buttonNumber)

Description: this macro returns the matrix button state.

Parameters:

- **buttonNumber** - matrix button number.

Returns: matrix button state flags(see MTOUCHCONTROLSTATE union in mTouchControl.h).

10.4.6 void MTouchSet2ChSlider(UINT8 sliderNumber, UINT8 ch1SensorNumber, UINT8 ch2SensorNumber)

Description: this function initializes 2 channels slider.

Parameters:

- **sliderNumber** - slider number.
- **ch1SensorNumber** - first sensor number.
- **ch2SensorNumber** - second sensor number.

10.4.7 MTouchGet2ChSliderState(sliderNumber)

Description: this macro returns the slider state.

Parameters:

- **sliderNumber** - slider number.

Returns: slider state (see MTOUCHCONTROLSTATE union in mTouchControl.h).

10.4.8 MTouchGet2ChSliderValue(sliderNumber)

Description: this macro returns the slider current position.

Parameters:

- **sliderNumber** - number of slider.

Returns: slider value (current position) from 0 to 1000.

10.4.9 void MTouchSet4ChSlider(UINT8 sliderNumber, UINT8 ch1SensorNumber, UINT8 ch2SensorNumber, UINT8 ch3SensorNumber, UINT8 ch4SensorNumber)

Description: this function initializes 4 channels slider.

Parameters:

- **sliderNumber** - slider number.
- **ch1SensorNumber** - sensor 1 number.
- **ch2SensorNumber** - sensor 2 number.
- **ch3SensorNumber** - sensor 3 number.
- **ch4SensorNumber** - sensor 4 number.

10.4.10 MTouchGet4ChSliderState(sliderNumber)

Description: this macro returns the slider state.

Parameters:

- **sliderNumber** - number of slider.

Returns: slider state (see MTOUCHCONTROLSTATE union in mTouchControl.h).

10.4.11 MTouchGet4ChSliderValue(sliderNumber)

Description: this macro returns the slider current position.

Parameters:

- **sliderNumber** - number of slider.

Returns: slider value (current position) from 0 to 1000.

10.5 Debug Module

In this section the debug module library functions are described.

10.5.1 void LogChar(char ch)

Description: This function outputs character to debug log. It MUST BE defined in application.

Parameters:

- **ch** - character to be transmitted.

10.5.2 DEBUGCURRENT* MTouchDebugCurrent(UINT8 sensorNumber)

Description: The function adjusts CTMU current to charge the sensor to 75% of AVdd for unpressed state when charge delay is 8. Before the adjustment MTOUCH_CTMU_CURRENT parameter in mTouchConfig.h must be set to 0x01 and sensor must be initialized with MTouchSetSensor(...). The CTMU current result can be set to MTOUCH_CTMU_CURRENT parameter directly.

Parameters:

- **sensorNumber** - sensor number.

Returns: the function returns a pointer to the structure with the CTMU current settings value and corresponding error in the sensor charge delay.

10.5.3 INT16 MTouchDebugThreshold(UINT8 sensorNumber)

Description: This function waits for the sensor press event and returns an optimal sensor threshold. The threshold should be about 12.5% percents of the signal(delta) amplitude. Before measurement the sensor must be initialized with MTouchSetSensor(...). Use the threshold result to intialize sensor (see parameter "threshold" in MTouchSetSensor(...) function).

Parameters:

- **sensorNumbe**r - sensor number.

Returns: the function returns an optimal sensor threshold value.

10.5.4 DEBUGDELAY* MTouchDebugDelay(UINT8 sensorNumber)

Description:this function adjusts CTMU charge delay to charge the unpressed sensor to 75% of AVdd. Sensor must be initialized with MTouchSetSensor(...). The charge delay result returned by this function can be used to intialize sensor (see parameter "chargeDelay" in MTouchSetSensor(...) function).

Parameters:

- **sensorNumber** - sensor number.

Returns: a pointer to the structure with the charge delay adjustment.

10.5.5 void MTouchDebugLogDeltas(void)

Description: This function sends deltas for all sensors to debug log as a semicolon delimited ASCII string of 5 digit decimal numbers. The first number in the string is the sensors’ states, other numbers are deltas.

10.5.6 void MTouchDebugLogAverages(void)

Description: This function sends averages values for all sensors to debug log as a semicolon delimited ASCII string of 5 digit decimal numbers. The first number in the string is the sensors’ states, other numbers are average values.

10.6 Structures and Enumerations

In this section the library structures and enumerations are described.

10.6.1 MTOUCHSENSORSTATE Enum

Enumeration: **MTOUCHSENSORSTATE**

This enumeration defines all possible states for sensor.

Values:

- **SENSOR_INITIALIZING** - sensor is still initializing (see POWER_UP_SAMPLES in mTouchConfig.h),
- **SENSOR_RELEASED** - sensor is currently released,
- **SENSOR_PRESSED** - sensor is currently pressed,
- **SENSOR_DISCONNECTED** = 0x80 - bit 7 shows that the sensor must be removed from scan.

10.6.2 MTOUCHCONTROLSTATE Enum

Enumeration: **MTOUCHCONTROLSTATE**

This enumeration defines possible state flags for controls.

Values:

- **CONTROL_IDLE** = 0x80 - bit 7 shows that control is in idle state (the state was not changed),
- **CONTROL_PRESSED** – control pressed,
- **CONTROL_RELEASED** – control released.

10.6.3 MTOUCHCONTROLDECODE Enum

Enumeration: **MTOUCHCONTROLDECODE**

This enumeration defines possible decode method flags for controls. These flags can be ORed.

Values:

- **DECODE_TOGGLE** - toggled button,
- **DECODE_PRESS_RELEASE** - simple button (reports pressed or released states),
- **DECODE_MOST_PRESSED** - looks through all pressed buttons having the decode method *DECODE_MOST_PRESSED* and reports "pressed" state only for one which has a bigger signal,
- **DECODE_PRESS_REPEAT** - if button is held pressed it starts to generate "pressed"/"released" events periodically. See MCONTROL_REPEAT_INITIAL_DELAY and MCONTROL_REPEAT_DELAY settings in mTouchConfig.h,
- **DECODE_ONE_EVENT** - if control's state is not changed CONTROL_IDLE state flag will be set.

10.6.4 DEBUGCURRENT Struct

Structure: **DEBUGCURRENT**

This structure contains results for the CTMU current adjustment. This resut can be used directly for MTOUCH_CTMU_CURRENT setting in mTouchConfig.h file. It is used by MTouchDebugCurrent(...) function.

Fields:

- INT16 **error** - charge delay error for the adjusted current from the nominal charge delay (equals 8).
- UINT8 **current** - settings for CTMU current. Bits 1-0 define the current source range (IRNG) and bits 7-2 define current trim value (ITRIM, signed).

10.6.5 DEBUGDELAY Struct

Structure: **DEBUGDELAY**

This structure contains results for CTMU charge delay adjustment. It is used by MTouchDebugDelay(...) function.

Fields:

- INT8 **delay** - settings for CTMU charge delay.

- INT16 **error** – sample error for the adjusted delay from the nominal value (75% of AVdd). The error should be less than 128 for 10-bit ADC and less than 512 for 12-bit ADC.

11 Known Limitations

The known limitations of mTouch^{TM} software library version 1.41 are listed below:

- For the PIC18 demos when HiTech PICC18 or XC8 compilers are used the optimization level should be set to STANDARD (LITE) option for successful operation.

12 Resources

To get more information about mTouch sensing solutions visit http://www.mirochip.com/mtouch and read the following articles:

- Capacitive Sensors by Larry K. Baxter ISBN 0-7803-5351-X
- AN1101, AN1102, AN1103, AN1104 – Covers Basic Cap Touch
- AN1250 – Cap Touch with CTMU
- AN1254 – Capacitive Touch Algorithm Simulation
- AN1298 – Capacitive Touch Using Only an ADC (CVD)
- AN1325 – mTouch™ Metal Over Cap Technology
- AN 1334 –Techniques for Robust Touch Sensing Design

Index

V