AN205062

F²MC-8FX Family MB95200H/210H Series Capacitance Touch Sensor

Associated Part Family: MB95200H/210H Series

This Application Note describes Cypress TSC solution, and explains how to use TSC library and TSC GUI.

Contents

1 Introduction....1
2 Cypress Capacitance Touch Sensor Solution....2
3 Library....4
3.1 Library Functions....4
3.2 Cypress TSC Performance....8
3.3 How to Add Cypress TSC.lib....9
3.4 How to use Cypress TSC.lib....13
4 LIB Usage Notice....14
5 TSC GUI....16
5.1 Overview....16
5.2 Open Software....17
5.3 Check Threshold....18
5.4 Test Threshold....23
5.5 Graph....27
6 Additional Information....27
A Appendix....28
A.1 Sample Code....28
Document History....34
Worldwide Sales and Design Support....35
Products....35
PSoC® Solutions....35
Cypress Developer Community....35
Technical Support....35

1 Introduction

This application note describes Cypress TSC solution, and describes how to use TSC library and TSC GUI.

Section 2 explains the working principles of TSC solution.

Section 3 explains how to use TSC library.

Section 4 explains LIB usage notice.

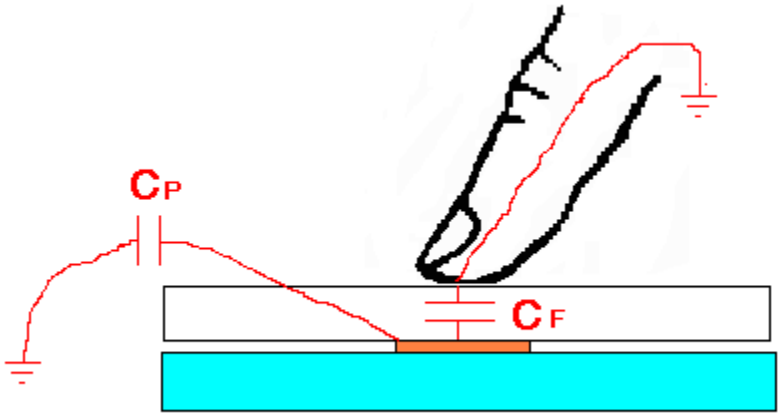
Section 5 explains how to use TSC GUI.

2 Cypress Capacitance Touch Sensor Solution

This section introduces working principles of TSC solution.

The theory of capacitance touch sensor is to check capacitance increment. As follows, when finger not touch the C = Cp, when touching the C = Cp +Cғ, and the increment is ΔC = Cғ.

Figure 2-1. Capacitance Touch Sensor Theory



According to the characteristics of capacitor, the deposited charge of capacitor increases along with the increase of capacitance. Cypress uses A/D+GPIO method to check the capacitance change. The method includes the following steps: GPIO output ‘H’, duration 20us, then GPIO input and A/D sampling.

Figure 2-2. Operation Flowchart

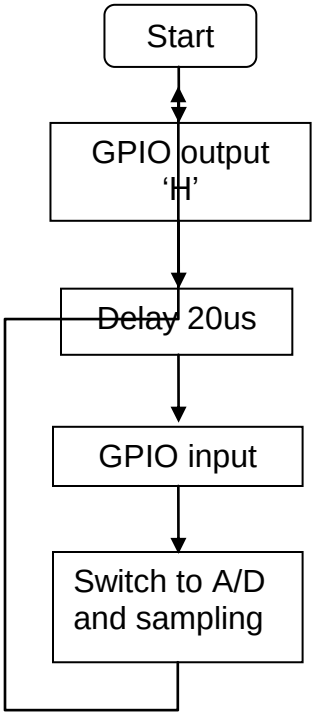
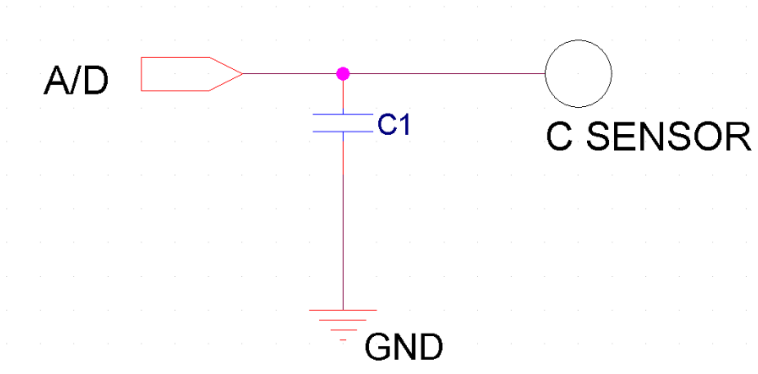
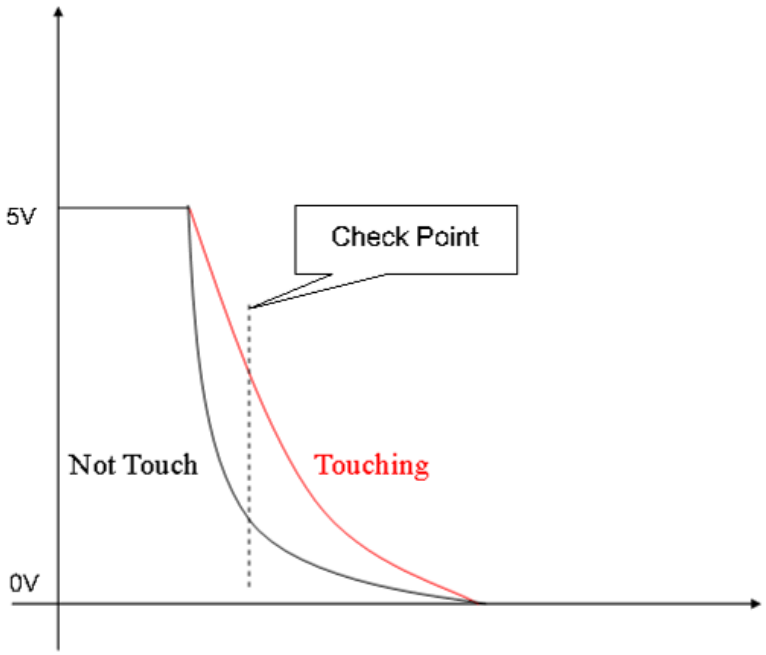


Figure 2-3. Hardware Connection



During GPIO input, the voltage when C sensor is touched is higher than the case when it is not touched.

Figure 2-4. Sketch Map



3 Library

This section introduces how to use TSC library.

Basing on MB95F204K, Cypress designs a 4-buttons TSC solution which provides a library for user to use and a TSC GUI to test.

3.1 Library Functions

The library has 12 functions which are listed as follow.

Table 3-1. Functions

Function Name	Function
void TSC_init(void)	Initialize TSC
void button1_init(unsigned int threshold)	Set button1 threshold
void button2_init(unsigned int threshold)	Set button2 threshold
void button3_init(unsigned int threshold)	Set button3 threshold
void button4_init(unsigned int threshold)	Set button4 threshold
void UART_init(void)	Initialize and start serial port
void TBT_init(void)	Initialize and start Timebase Timer
unsigned char Get_Data(void)	Get check data
void Transmit(void)	Transmit data

Function Name	Function
unsigned char get_mode(void)	Get sensor mode
void set_mode(unsigned char set_mode)	Set sensor mode
void clear_check_data(void)	Clear check data

- void TSC_init(void)

Table 3-2. TSC_init

Function Name	TSC_init
Description	Initialize IO which connect sensor, and initialize A/D.
Input	None
Return	None

- void button1_init(unsigned int threshold)

Table 3-3. button1_init

Function Name	button1_init
Description	Set button1 threshold
Input	threshold: threshold value, and the type data from 1000 to 1120
Return	None

- void button2_init(unsigned int threshold)

Table 3-4. button2_init

Function Name	button2_init
Description	Set button2 threshold
Input	threshold: threshold value, and the type data from 1000 to 1120
Return	None

- void button3_init(unsigned int threshold)

Table 3-5. button3_init

Function Name	button3_init
Description	Set button3 threshold
Input	threshold: threshold value, and the type data from 1000 to 1120
Return	None

- void button4_init(unsigned int threshold)

Table 3-6. button4_init

Function Name	button4_init
Description	Set button4 threshold
Input	threshold: threshold value, and the normal value from 1000 to 1120
Return	None

- void UART_init(void)

Table 3-7. UART_init

Function Name	UART_init
Description	Initialize and start serial port, the data length is 8-bit, stop bit length is 1-bit, parity is even parity, and the baud rate is 115200.
Input	None
Return	None

- void TBT_init(void)

Table 3-8. TBT_init

Function Name	TBT_init
Description	Initialize and start Timebase Timer, the interval time is 2^{12} x 1/FCRH
Input	None
Return	None

- void Transmit(void)

Table 3-9. Transmit

Function Name	Transmit
Description	Transmit check data to TSC GUI
Input	None
Return	None

- unsigned char get_mode(void)

Table 3-10. get_mode

Function Name	get_mode
Description	Get current check mode
Input	None
Return	Return check mode: **0:** highest mode, this mode only returns sensor number which the maximum increment of voltage and over threshold. **1:** normal mode, this mode return all over threshold sensor. **2:** slippage mode, this mode checks slippage direction, right or left.

- unsigned char Get_Data(void)

Table 3-11: Get_Data

Function Name	Get_Data
Description	Get check data
Input	None
Return	Return sensor number or slippage direction. The returned value varies with the mode. **highest mode:** 1->button1, 2->button2, 3->button3, 4->button4, 0-> no button. **normal mode:** bit0-> button1, bit1-> button2, bit2-> button3, bit3-> button4, 1: touching, 0: not touch. **slippage mode:** 1: right, 2: left, 0: checking.

- void set_mode(unsigned char set_mode)

Table 3-12. set_mode

Function Name	set_mode
Description	Set current check mode
Input	Set mode: **0:** highest mode, this mode only returns sensor number which the maximum increment of voltage and over threshold. **1:** normal mode, this mode return all over threshold sensor. **2:** slippage mode, this mode checks slippage direction, right or left.
Return	None

- void clear_check_data(void)

Table 3-13. clear_check_data

Function Name	clear_check_data
Description	Clear check data when checking
Input	None
Return	None

3.2 Cypress TSC Performance

Table 3-14. Performance

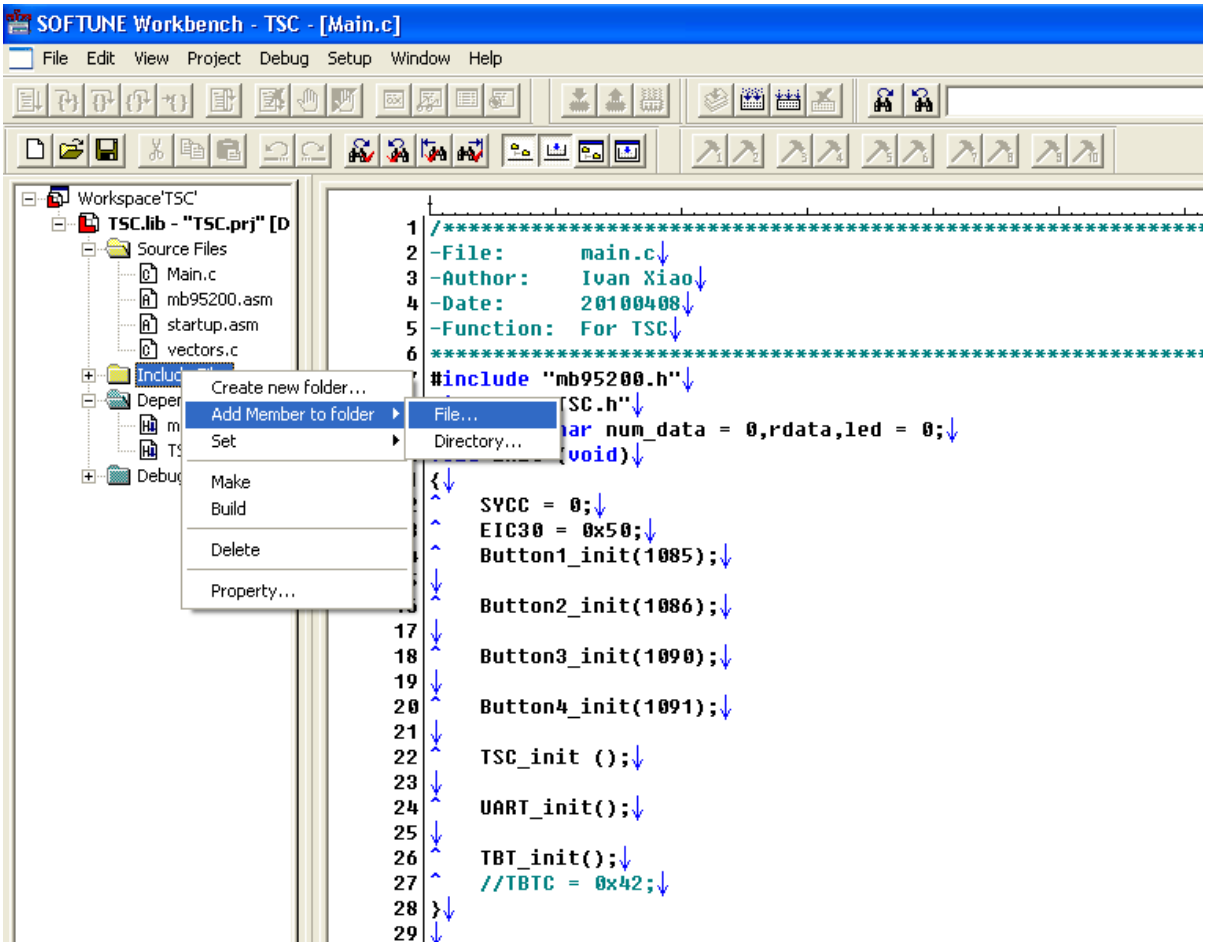
Resource	Amount	Description
ROM	1272 Byte	ROM space: If just need support 1 sensor, ROM of lib can be reduced to: 300 Byte If just need support 2 sensors, ROM of lib can be reduced to: 960 Byte If just need support 3 sensors, ROM of lib can be reduced to: 1120 Byte If need support 4 sensors, ROM of Lib is: 1272 Byte
RAM	36 Byte	RAM space: If just need support 1 sensor, RAM of lib can be reduced to: 11 Byte If just need support 2 sensors, RAM of lib can be reduced to: 26 Byte If just need support 3 sensors, RAM of lib can be reduced to: 31 Byte If need support 4 sensors, RAM of Lib is: 36 Byte
IO Port	4	P00~P02,P03
A/D	4	Channel 1,2,3,6(P00~P02,P03)
Timer	1	Timebase Timer
Serial Port	1	Serial Port(P04,P05),only use in check threshold mode
Machine Clock	>=8M	
Scan 1 Key	2 ms	

3.3 How to Add Cypress TSC.lib

3.3.1 Add Cypress TSC.lib to User's Project

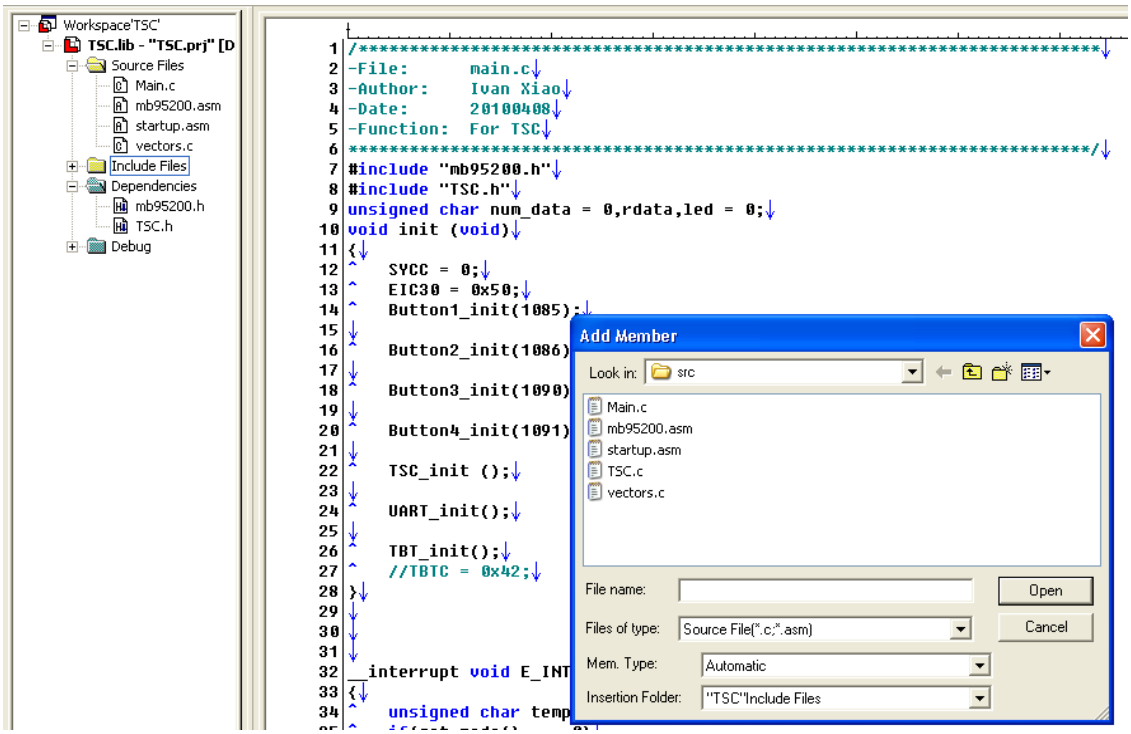
1. In Softune, right click **Include Files** and select **Add member to folder** > **File** from the shortcut menu.

Figure 3-1. Select File



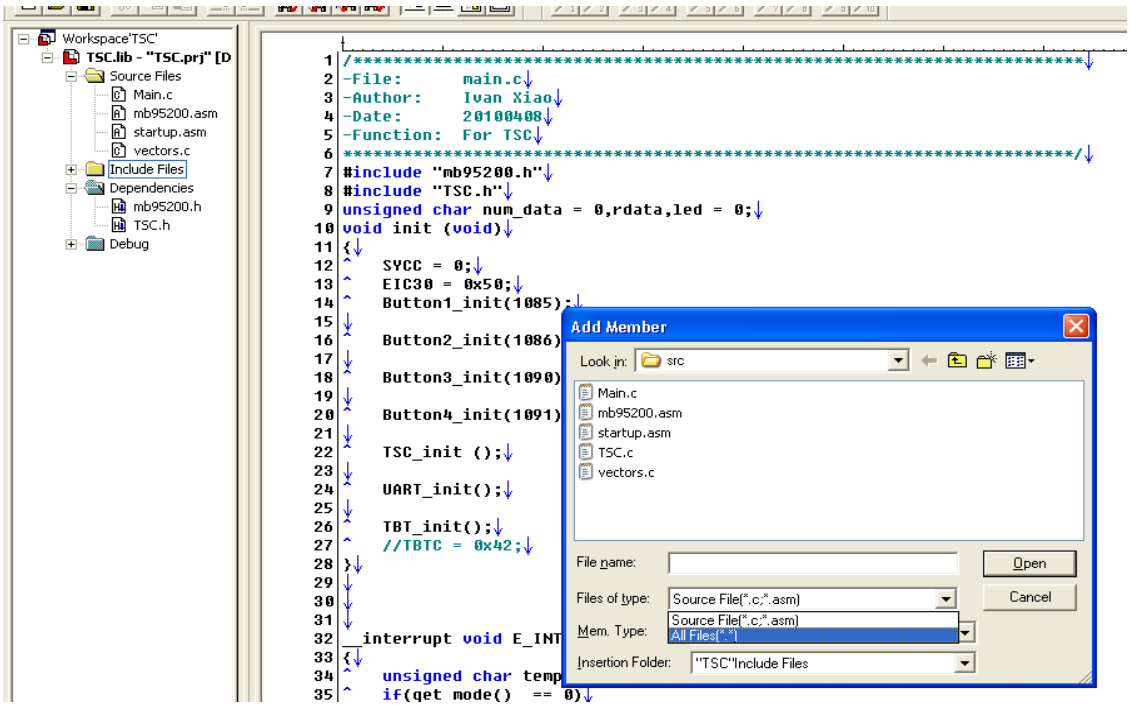
2. The TSC.lib can't be found in window of **Add Member**.

Figure 3-2. open Add Member Window



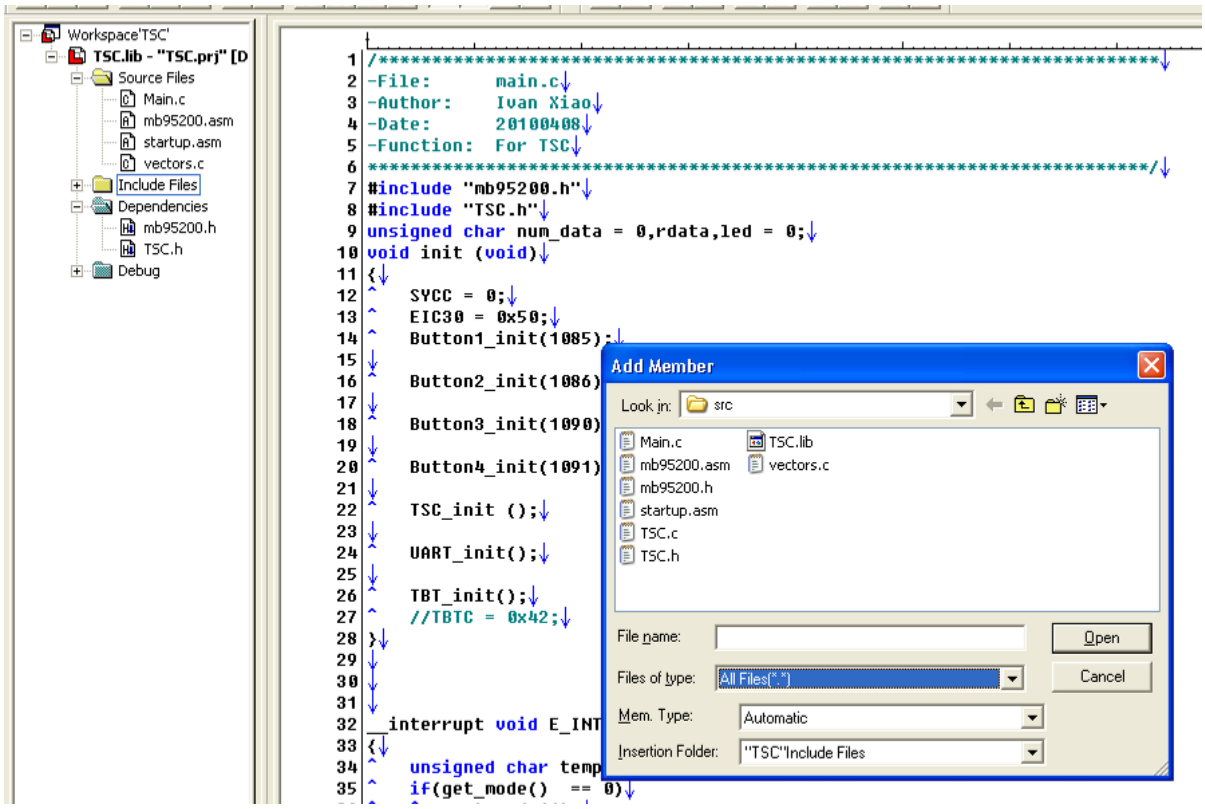
3. In **Add Member** window, select **ALL Files** from **Files of Type**, and then you will find the *TSC.lib*.

Figure 3-3. Select File Type



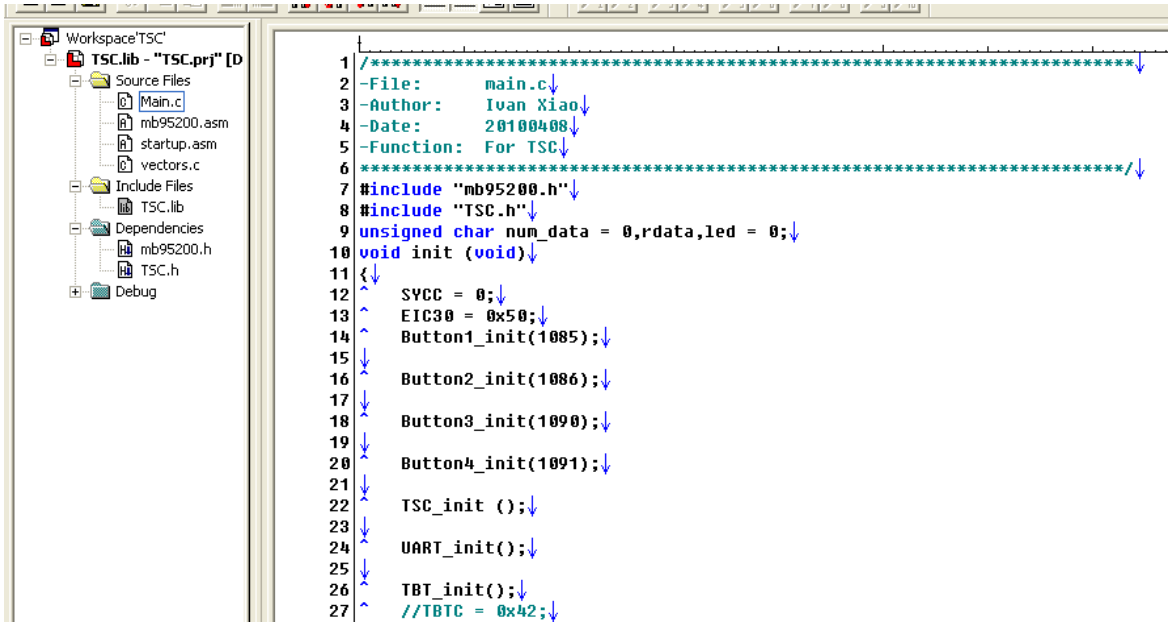
4. Then the *TSC.lib* could be found.

Figure 3-4. Show TSC.lib



5. Double click **TSC.lib**, and the *TSC.lib* will be added to Source Files.

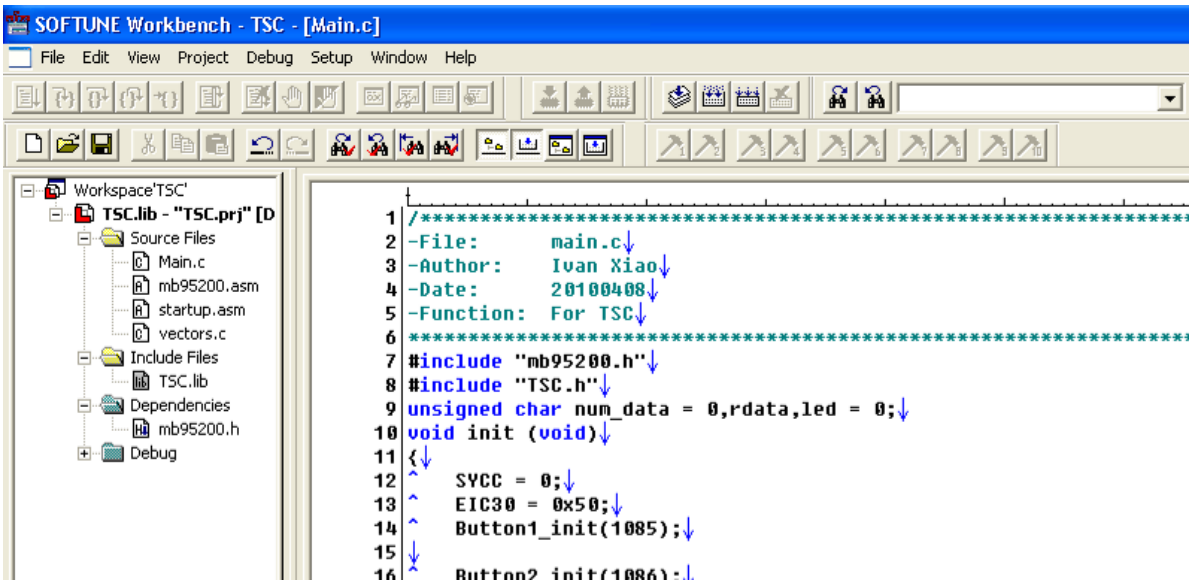
Figure 3-5. Add TSC.lib



3.3.2 Add "#include 'TSC.h'" in c File

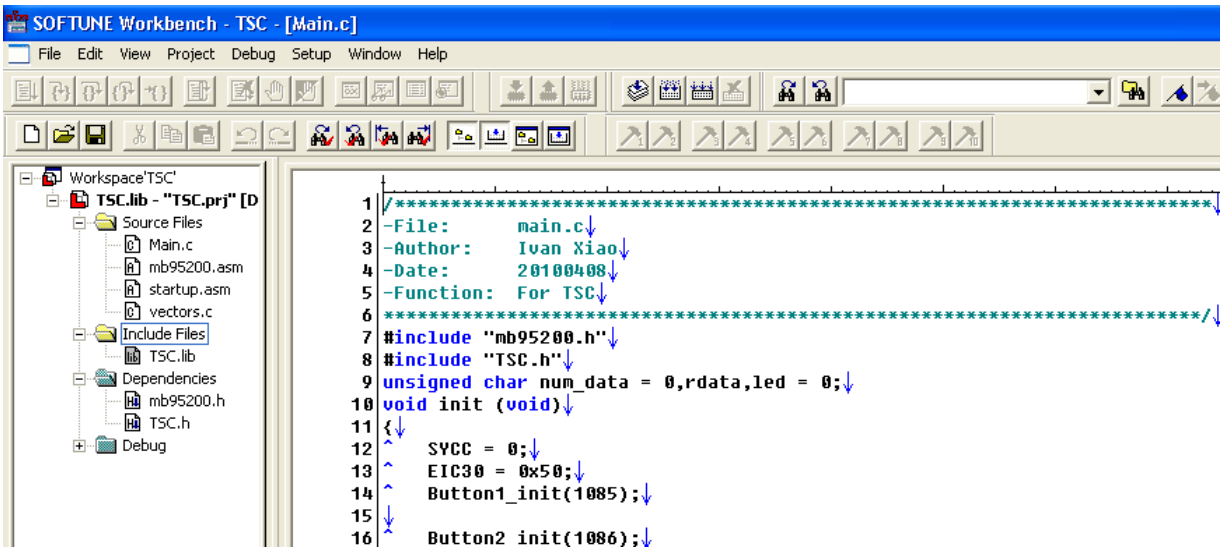
1. Add "`#include 'TSC.h'`" in *c* file, such as in "`main.c`".

Figure 3-6. Add "#include "TSC.h "" in "main.c"



2. Compile the whole project, "`TSC.h`" will link TSC.lib to *c* file, so that user program can use API functions in TSC.lib.

Figure 3-7: '`#include 'TSC.h'`' is Added Successfully

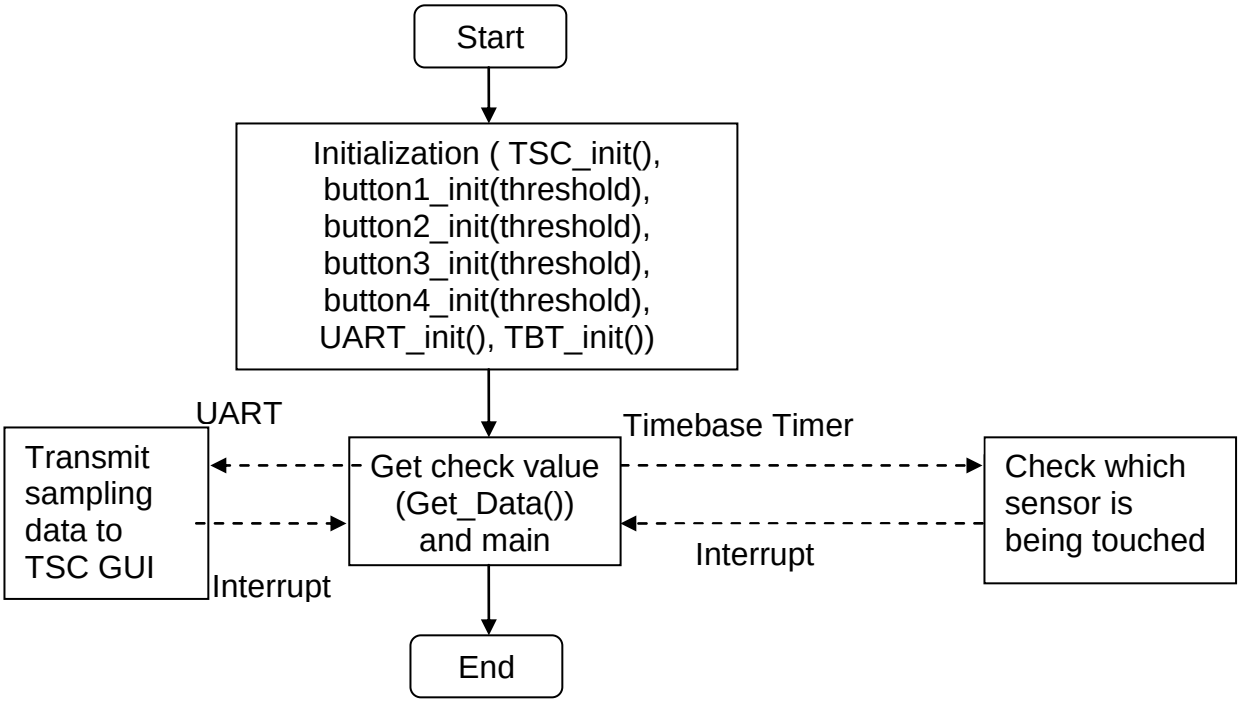


3.4 How to use Cypress *TSC.lib*

Using the library needs following step. First, initialize TSC main function, threshold and Timebase Timer. If need to get the threshold, the UART needs to be initialized. Second, use function "*Get_data()"* to get the check data. Timebase Timer interrupt checks sensor function, and UART transmits the check data to TSC GUI.

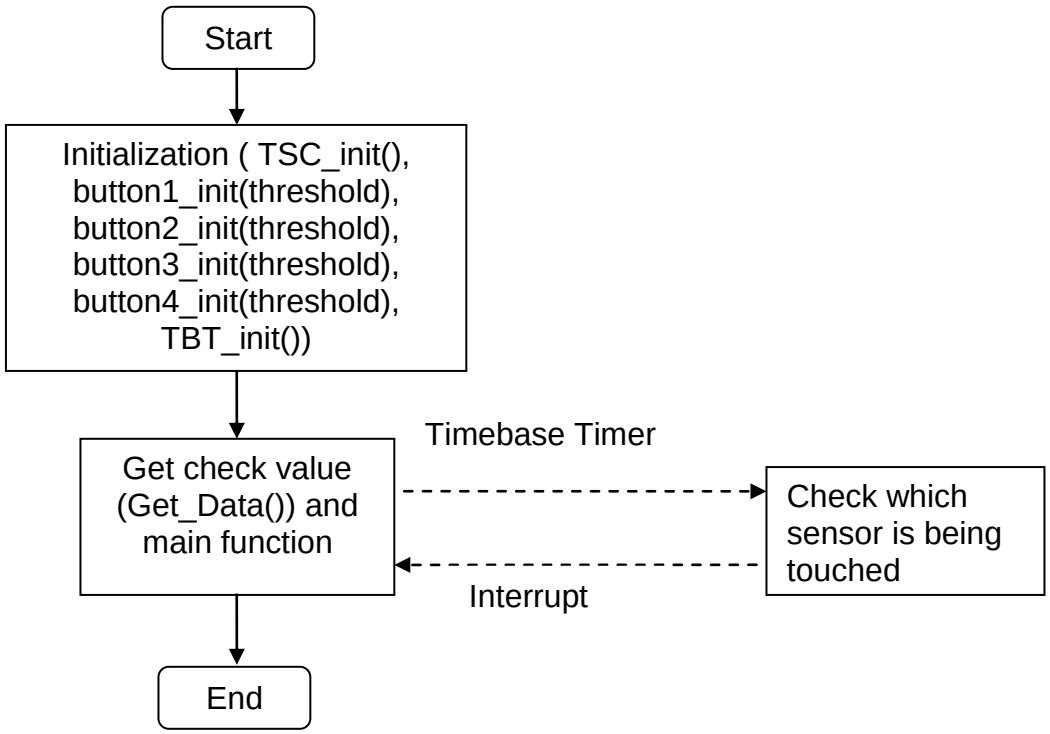
- Connect TSC GUI software flowchart

Figure 3-8. Get Threshold with GUI Flowchart



- Not connect TSC GUI software flowchart

Figure 3-9. Working Flowchart



4 LIB Usage Notice

This section introduces LIB usage notice.

- Machine clock

The machine clock should be set to 8M or above. If the machine clock is less than 8M, the sensor response is slow.

- Interrupt

This solution needs to use Timebase Timer interrupt to check sensor, and the interval time is 512us. It is unnecessary to use UART if the threshold is not gotten or tested with TSC GUI, and the interrupt setting in "`vector.c`" is as following:

```
#include "mb95200.h"
#include "TSC.h"
…
void InitIrqLevels(void)
{
…
      #ifdef enableUART
            ILR2 = 0xfd;     //  IRQ8:  LIN-UART (transmission)
                              //  IRQ9:  8/16-bit PPG ch1 (lower) |
UART/SIO ch1
                              //  IRQ10: 8/16-bit PPG ch1 (upper) | I2C
ch1
                              //  IRQ11: 16-bit reload timer ch0
      #else
      ILR2 = 0xff;      //  IRQ8:  LIN-UART (transmission)
                              //  IRQ9:  8/16-bit PPG ch1 (lower) |
UART/SIO ch1
                              //  IRQ10: 8/16-bit PPG ch1 (upper) | I2C
ch1
                              //  IRQ11: 16-bit reload timer ch0
     #endif
…
            ILR4 = 0xbf;    //  IRQ16: 16-bit reload timer ch1 | I2C ch0
                              //  IRQ17: 16-bit PPG ch1
                              //  IRQ18: 10-biat A/D-converter
                              //  IRQ19: Timebase timer
..
}
…
#ifdef enableUART
__interrupt void UART_T (void);
#endif
```

```
__interrupt void TBT (void);
..
#ifdef enableUART
#pragma intvect UART_T 8   //  IRQ8:  LIN-UART (transmission)
#else
#pragma intvect DefaultIRQHandler 8   //  IRQ8:  LIN-UART (transmission)

#endif…
#pragma intvect TBT           19          //  IRQ19: Timebase timer
…
```

- Mode

When slippage mode is selected, the check data will be cleared during checking. Use "*clear_check_data*" function to clear check data.

- Button number

The corresponding relationship between button number and pins are as follows.

Button1-P00, Button2-P01, Button3-P02, Button4-P05,

- Sample code

For more operation please refer to Appendix: Sample Code.

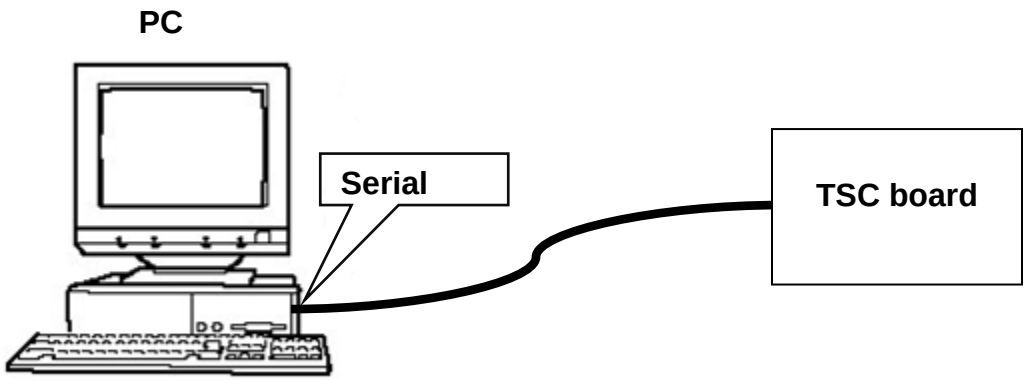
5 TSC GUI

This section introduces how to use TSC library.

5.1 Overview

The hardware needs to use Serial Port to connect PC with TSC board, as follows:

Figure 5-1. GUI Connects TSC Board



TSC GUI is used to get and test threshold. It includes four files, as shown in Figure 5-2

Figure 5-2. GUI Software



5.2 Open Software

Double click **TSC GUI.exe** to open the software, and the software includes 4 parts, Menu Bar, Check module, Test module and Graph module, as following:

Menu Bar: this `module 'File→exit', 'Tools→language', 'Help→help';'Help→about'.`

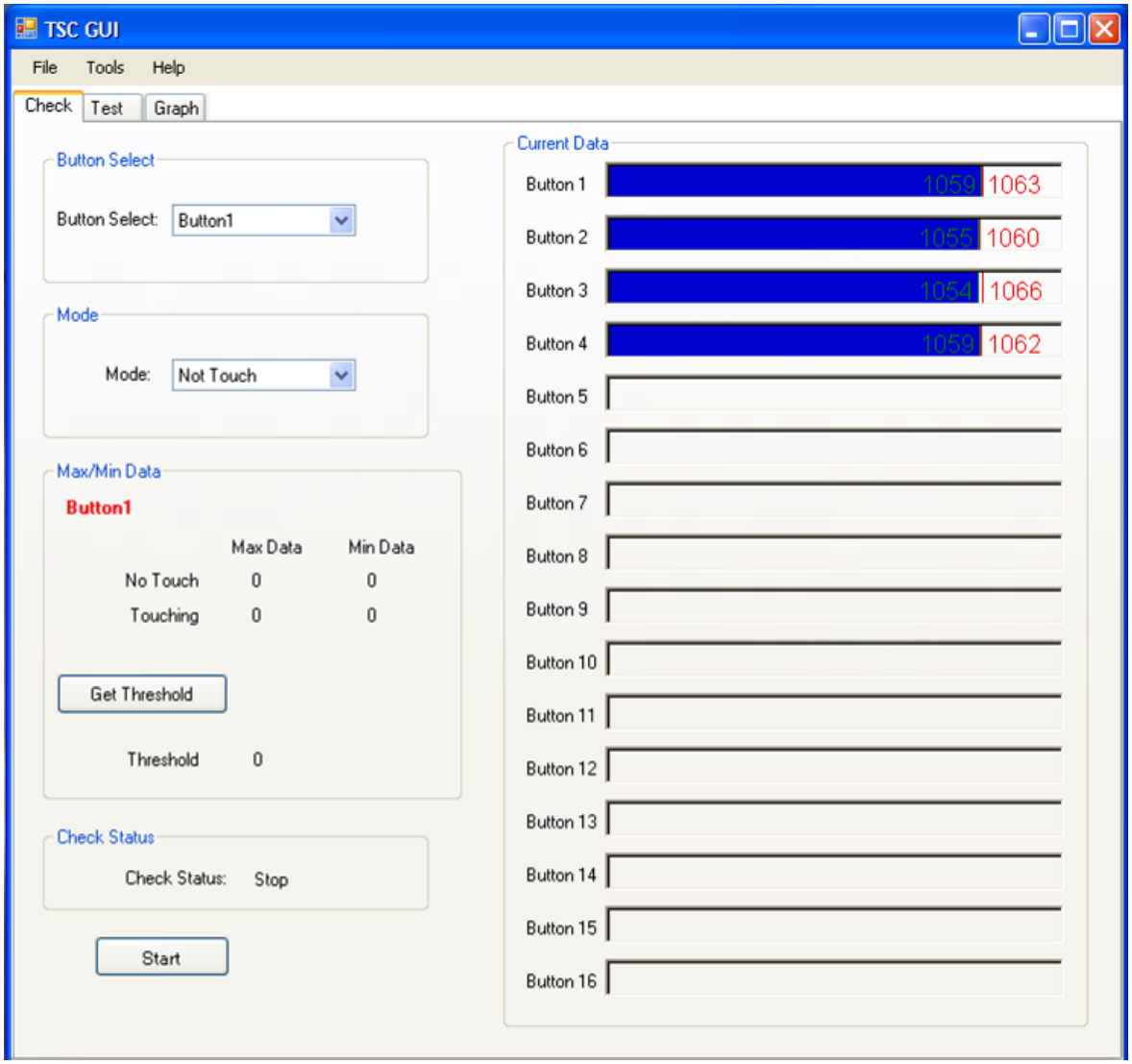
Check: this module includes 'Button Select', 'Mode', 'Max/Min Data', 'Check Status', 'Current Data' and 'Start' Button.

Test: this module includes 'Mode', 'Threshold', 'Test Result', 'Test Status', 'Current Data', and 'Start' Button.

Graph: this module includes 'Button Select' and graph display area.

The main interface is shown as follow:

Figure 5-3. TSC GUI Interface

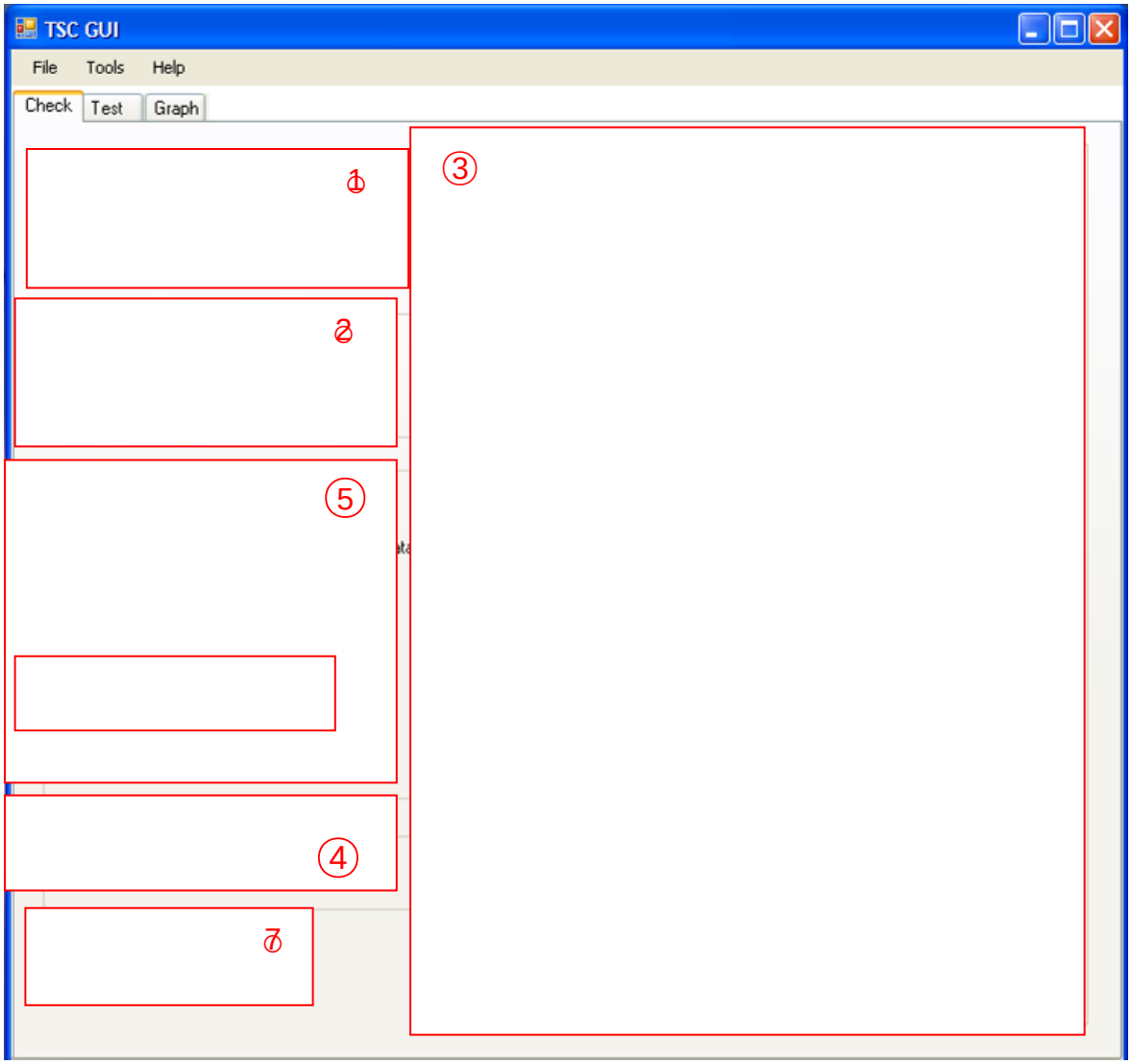


5.3 Check Threshold

5.3.1 Overview

1. **Button Select**: This item is used to select which button to check. It has 16 options: “button 1” to “button 16”.
2. **Mode**: This item is used to set current check mode, it include 2 modes, “Not Touch”, “Touching”.
3. **Current Data**: This item displays current data.
4. **Check Status**: This item displays current status, “checking” or “stop”.
5. **Max/Min Data**: This item displays max data and min data when check is stopped.
6. **Get Threshold**: Get the threshold.
7. **Start/Stop**: Clicking this item can start or stop check.

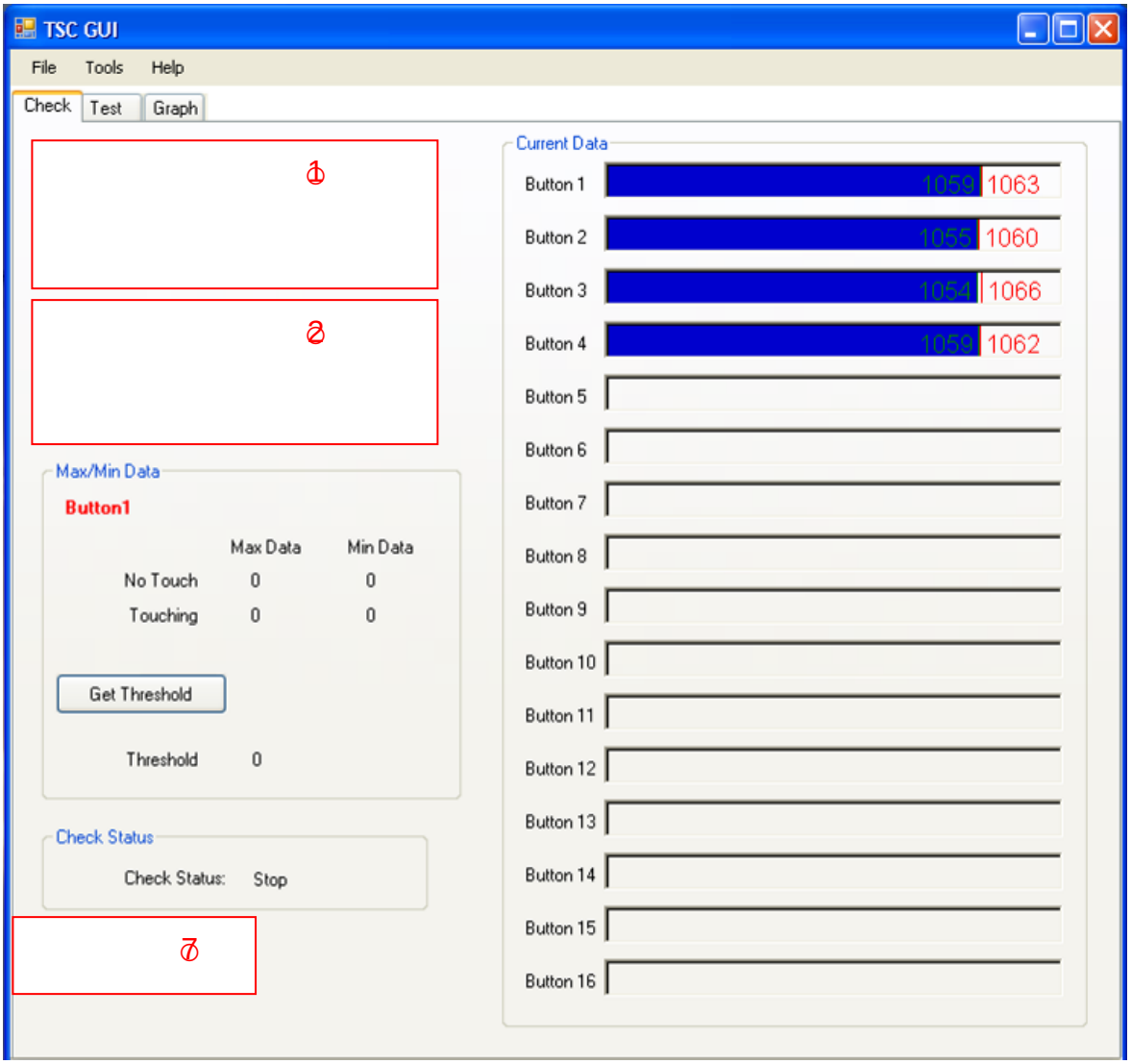
Figure 5-4. Check Main Interface



5.3.2 Check Start

Select the button to be checked from **Button Select** field and select the right mode from **Mode** field. Click **Start**, and the button text changes to **Stop**. The button will be checked for a moment.

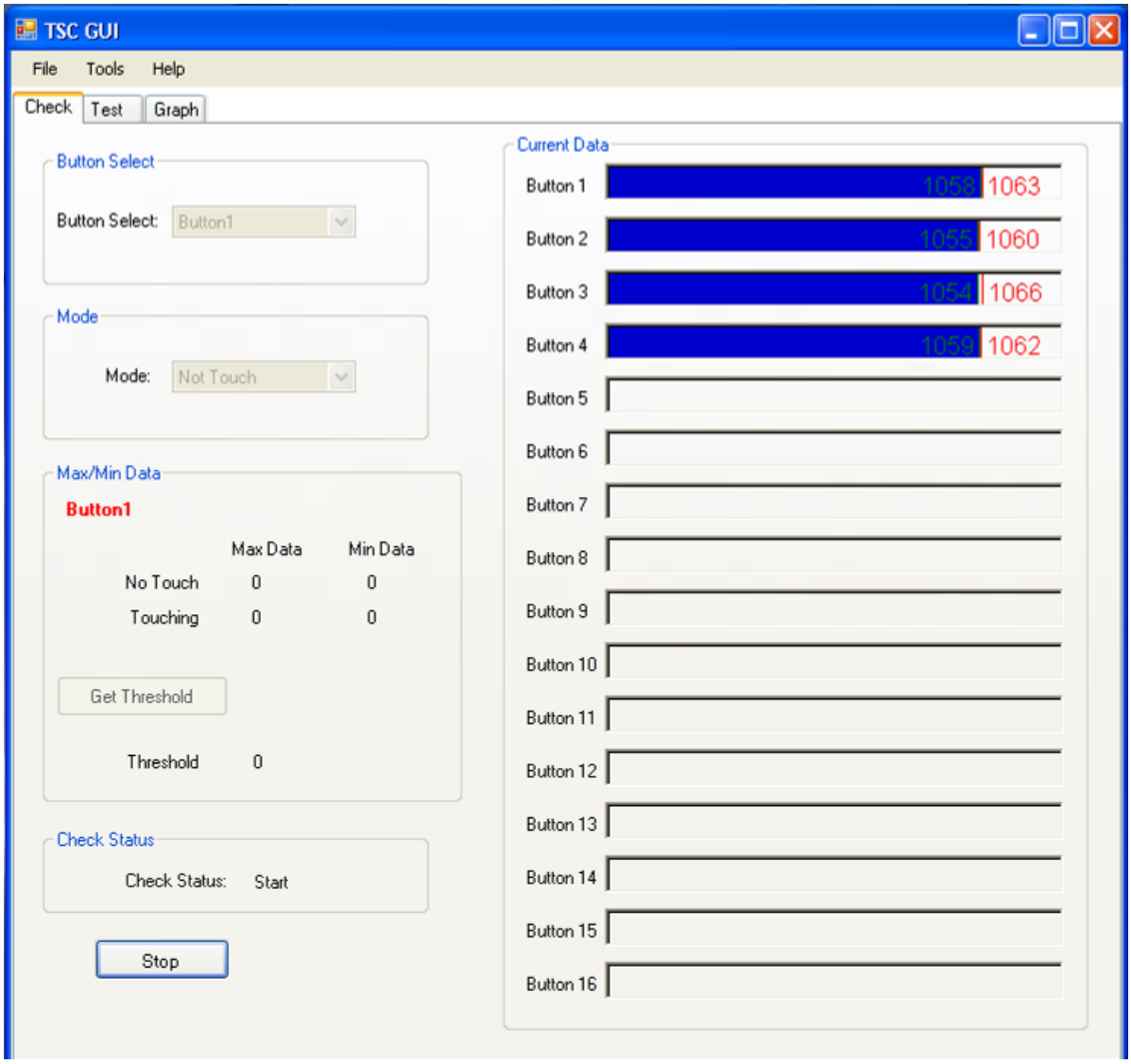
Figure 5-5. Check Start



5.3.3 Checking

In this status, all the fields in GUI Check Module cannot be operated except ‘Stop’. If **Touching** is selected in **Mode**, the sensor should be kept in touching state.

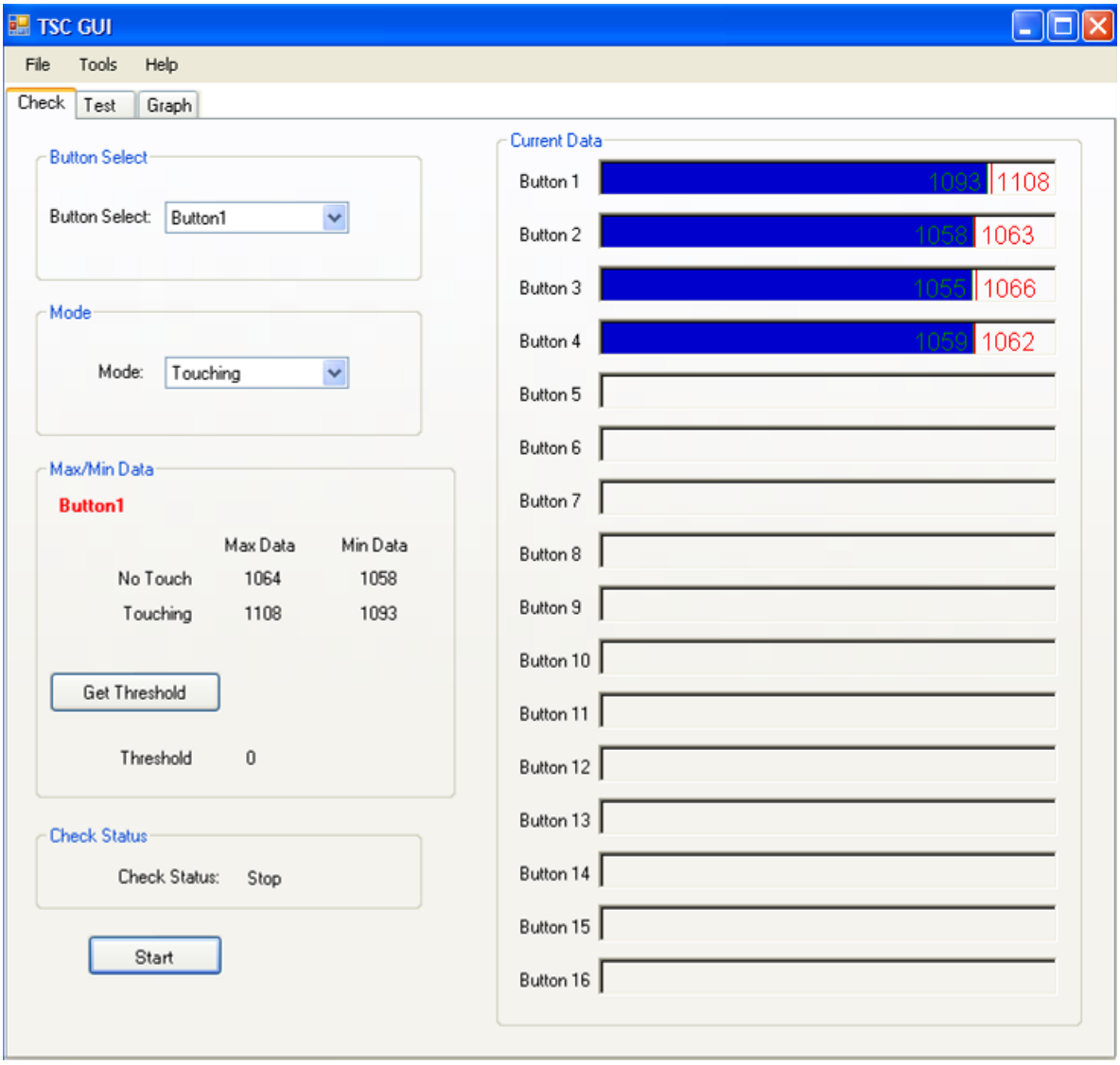
Figure 5-6. Checking



5.3.4 Check Stop

Click **Stop** button to stop check, and the button text changes to **Start**. The max data and min data are generated in **Max/Min Data** field.

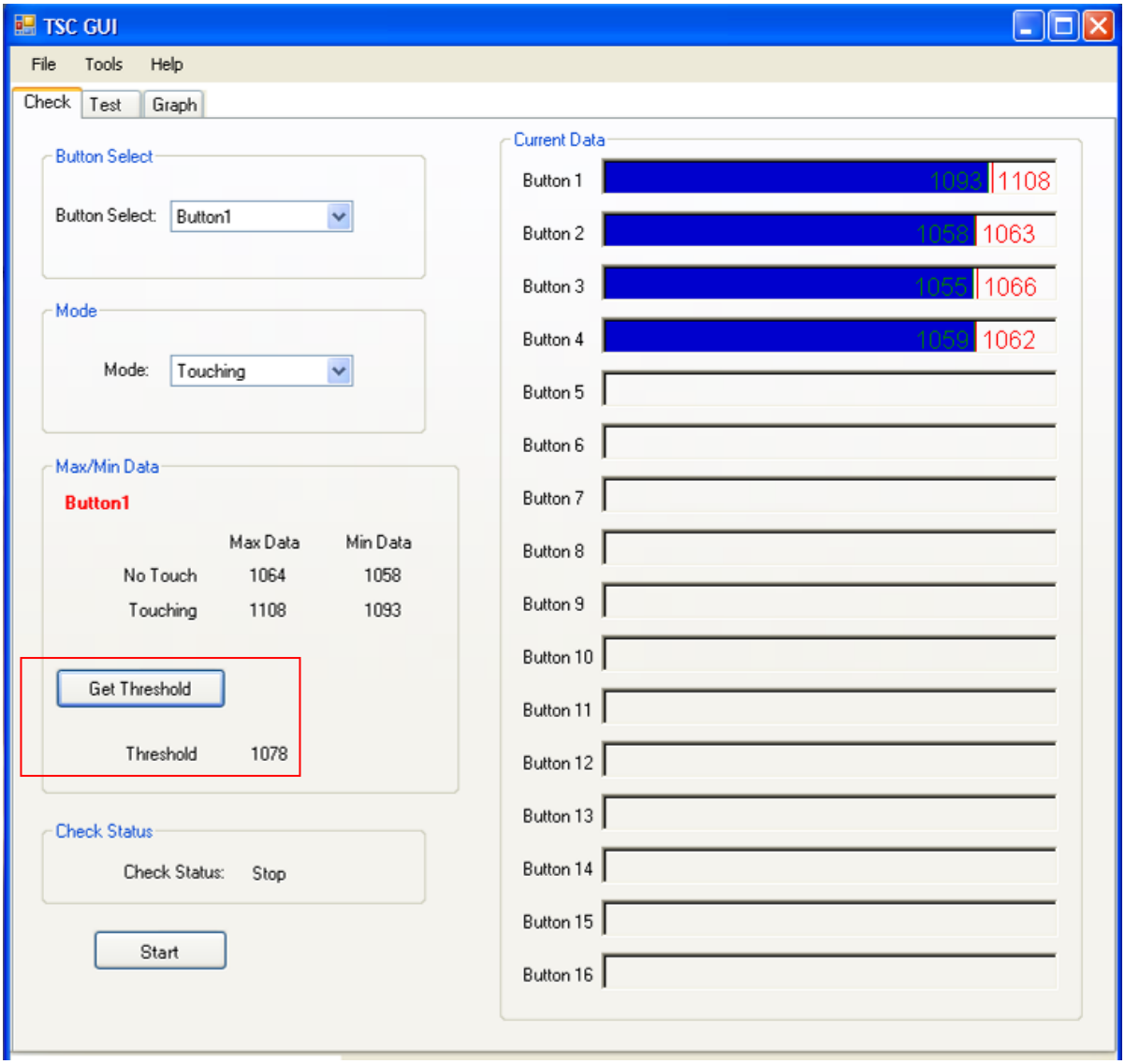
Figure 5-7. Check Stop



5.3.5 Calculate Threshold

When **Not Touch** and **Touching** statuses are all checked, click **Get Threshold** button to calculate the threshold.

Figure 5-8. Calculate Threshold

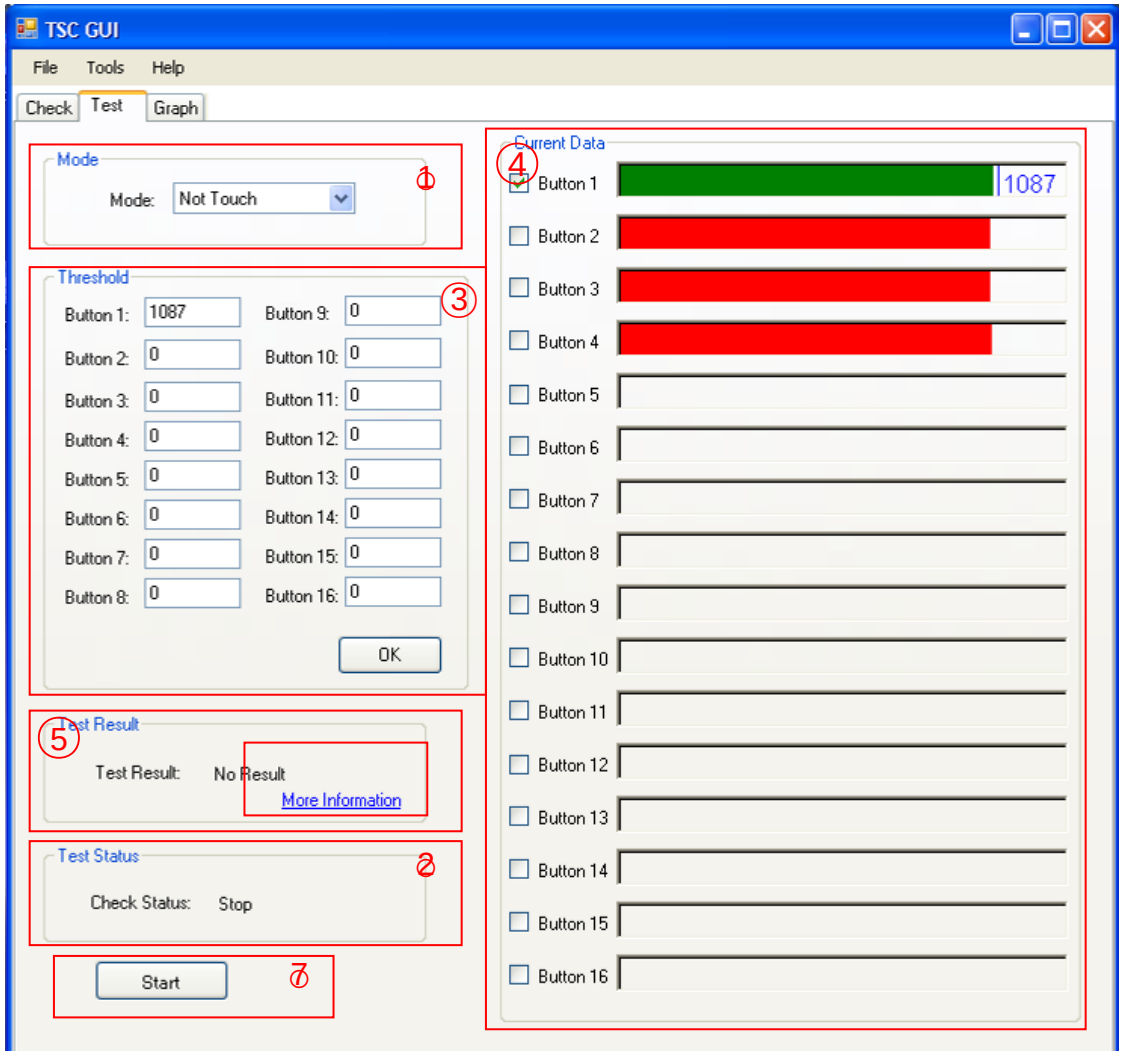


5.4 Test Threshold

5.4.1 Overview

1. **Mode**: This item is used to set current test mode, it include 3 modes, “Not Touch”, “Touching”.
2. **Test Status**: This item displays current status, “checking” or “stop”.
3. **Threshold**: This item sets the threshold to be tested.
4. **Current Data**: This item displays current data.
5. **Test Result**: This item displays test result, “pass” and “not pass”.
6. **More Information**: Display more information on test.
7. **Start/Stop**: Clicking this item can start or stop test.

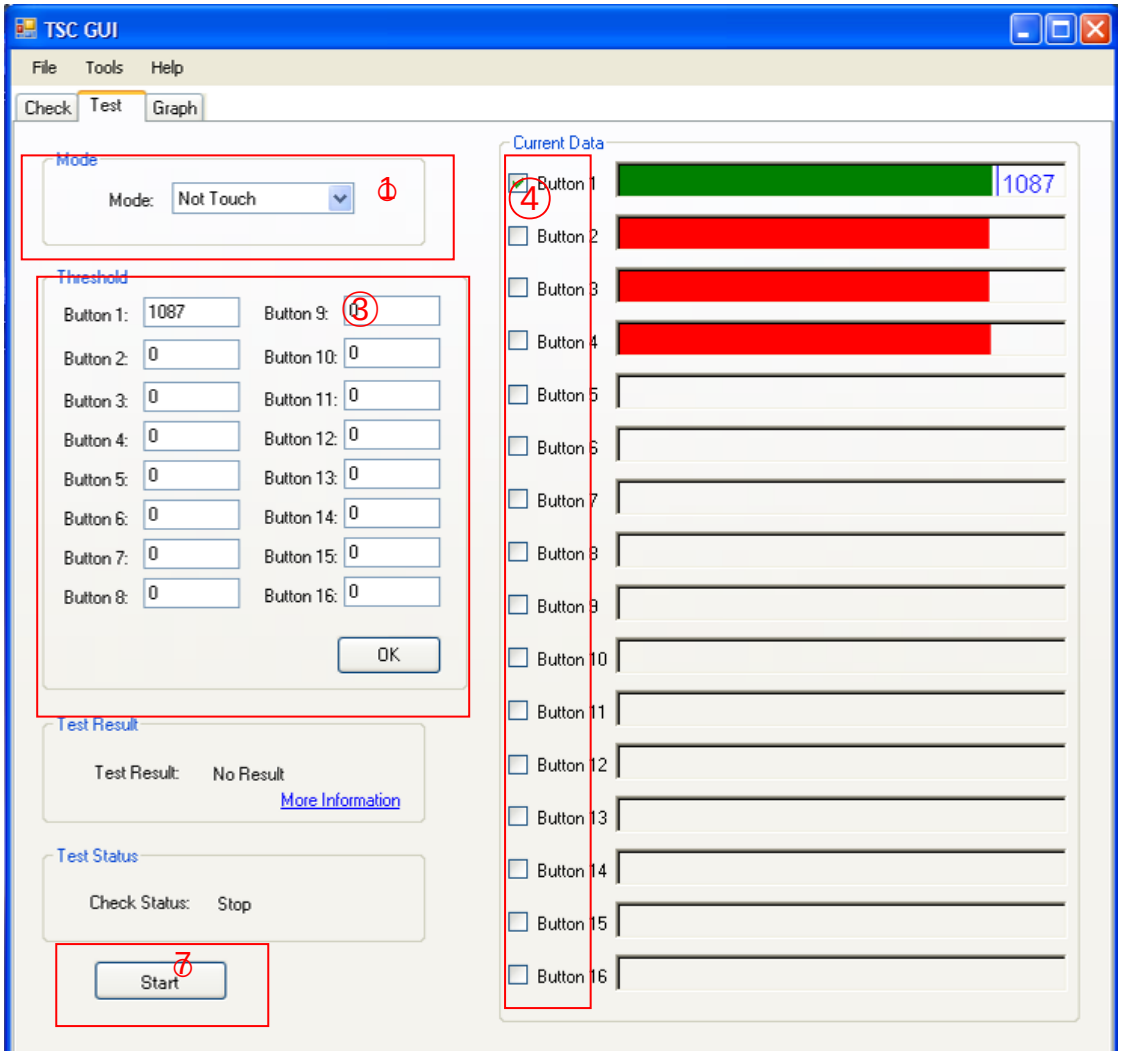
Figure 5-9. Test Main Interface



5.4.2 Test Start

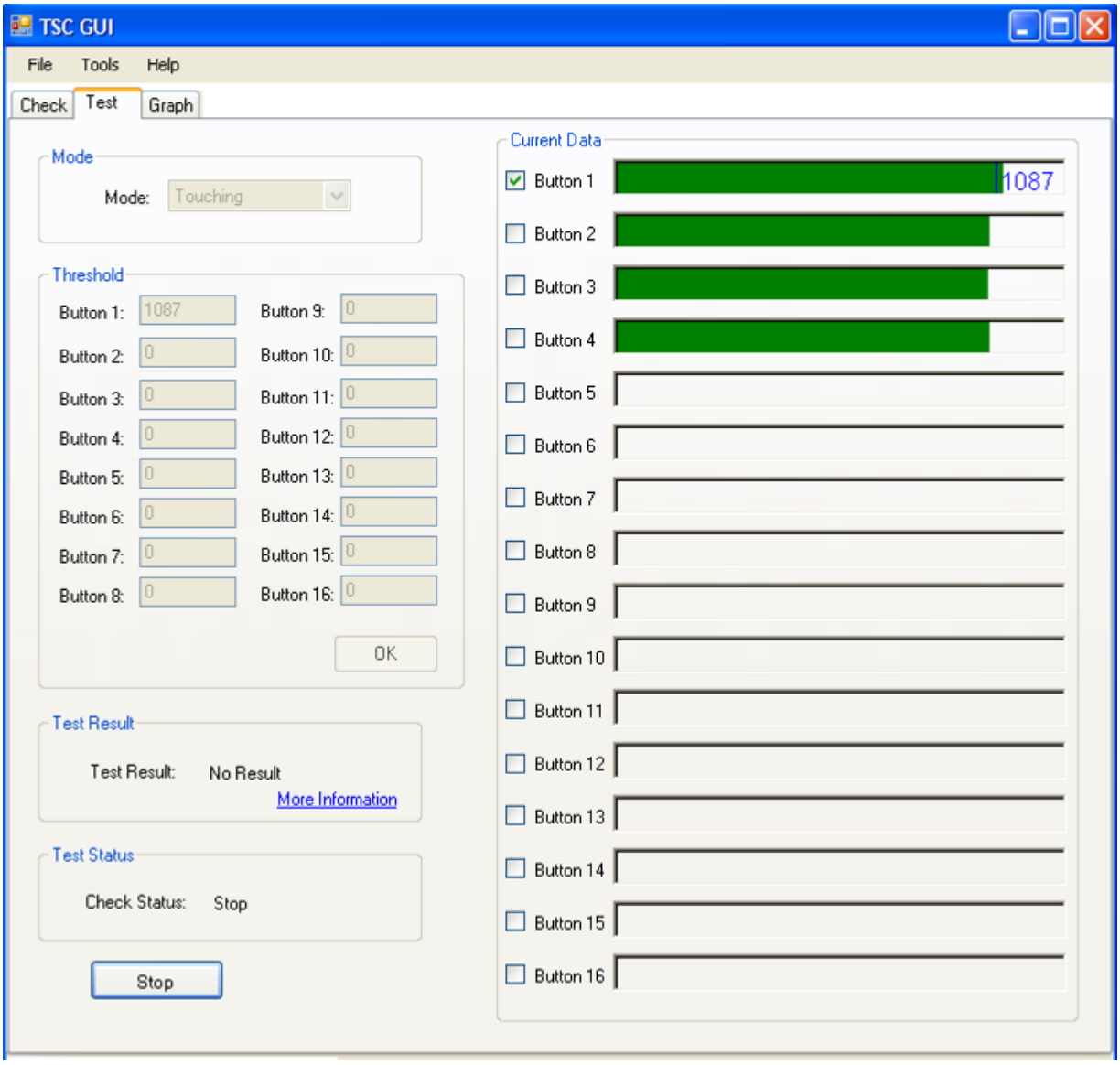
Select a mode from **Mode** field, select the button to be tested in **Current Data** field, and input the threshold in **Threshold** field. If the threshold is already gotten in check module, it will be automatically displayed in **Threshold** field. Click **OK** to submit the threshold.

Figure 5-10. Test Start



5.4.3 Testing

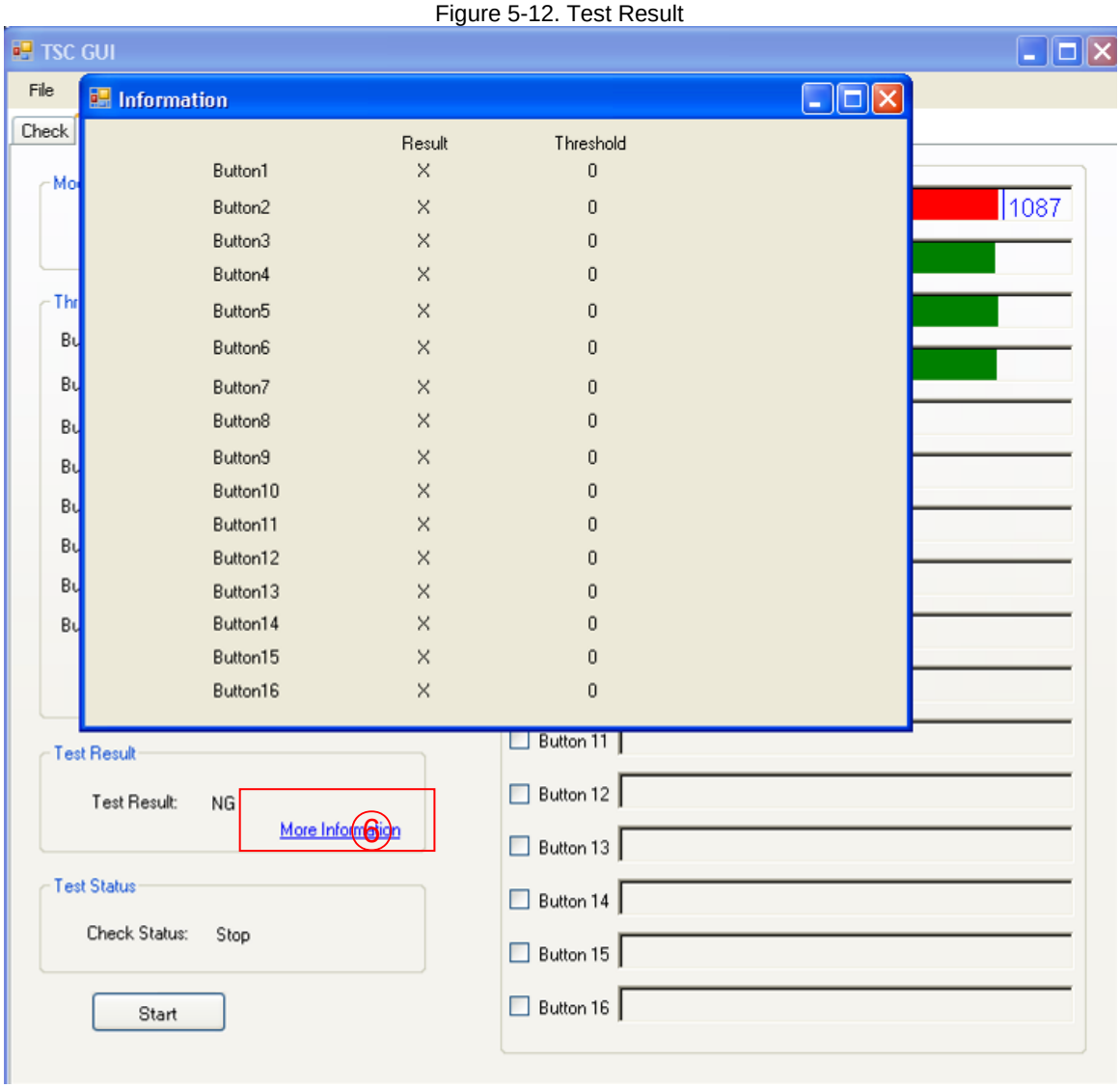
Figure 5-11. Testing



5.4.4 Test Stop

Click **Stop** to stop test, and the current button test result will be displayed in **Test Result**. Click **More Information** to display all button test results.

Figure 5-12. Test Result



5.5 Graph

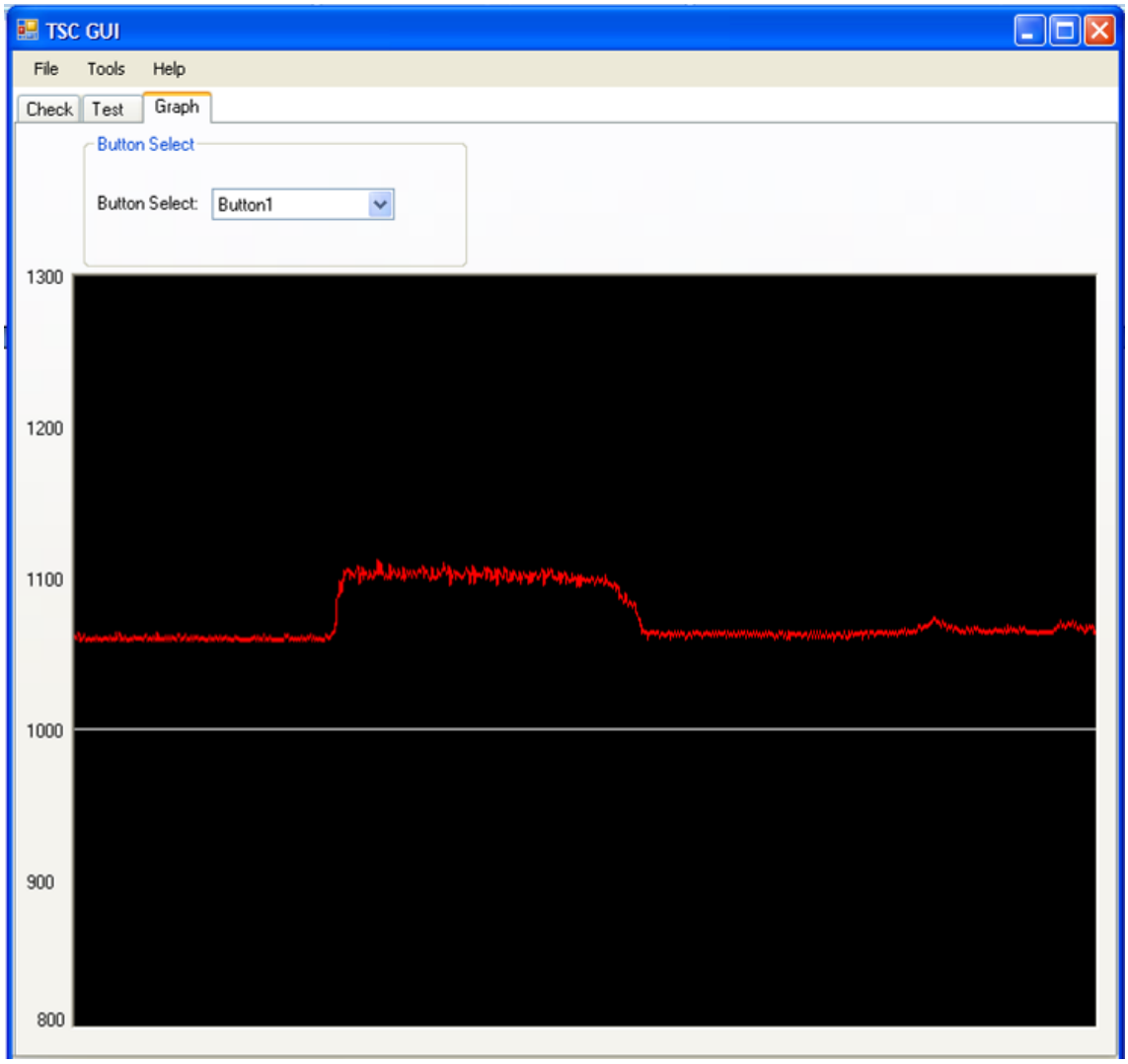
5.5.1 Overview

1. **Button Select**: This item is used to select which button to display. It has 16 items, “button 1” to “button 16”.

5.5.2 Graph

This module displays A/D sampling data graph. Select the button to be displayed from **Button Select** field.

Figure 5-13. Graph



6 Additional Information

Please contact your local support team for any technical question.

A Appendix

A.1 Sample Code

Project Name: TSC

Function: TSC

```
/*****************************************************************************
-File:      main.c
-Author:    Ivan Xiao
-Date:      20100408
-Function:  For TSC
*****************************************************************************/
#include "mb95200.h"
#include "TSC.h"
unsigned char led = 0;
/*****************************************************************************
-FunctionName:  init
-Description:   initialization system and
-Input:         None
-Return:        None
*****************************************************************************/
void init (void)
{
        SYCC = 0;                       //machine frequence is 8M
        EIC30 = 0x50;                   //external interrupt 7
        DDR1 = 0xff;
        PDR1 = 0xFF;
        DDR6 = 0xff;
        PDR6 = 0xFF;
        Button1_init(1077);             //button1 threshold initialization
        Button2_init(1080);             //button2 threshold initialization
        Button3_init(1086);             //button3 threshold initialization
        Button4_init(1080);             //button4 threshold initialization
        TSC_init ();                    //TSC initialization
#ifdef enableUART
        UART_init();                    //use serial port and initialization
#endif
        TBT_init();                     //Timebase Timer initialization
}
```

```
/*****************************************************************************************
-FunctionName:  main
-Description:   main function
-Input:         None
-Return:        None
*****************************************************************************************/
void main(void)
{
        unsigned char temp;
        init ();
        InitIrqLevels();
        __EI();
        while(1)
        {
                if(get_mode() == Highest_Mode)      //judge current mode
                {
                        switch(Get_Data())          //get check result
                        {
                                case HighestMode_Button1:
                                        PDR6 = 0xf7;  //button1 touching
                                        temp = PDR0;
                                        temp |= 0x40;
                                        PDR0 = temp;
                                        PDR1 = 0xff;
                                        break;
                                case HighestMode_Button2:
                                        PDR6 = 0xef;  //button2 touching
                                        temp = PDR0;
                                        temp |= 0x40;
                                        PDR0 = temp;
                                        PDR1 = 0xff;
                                        break;
```

```
                    case HighestMode_Button3:
                        PDR6 = 0xff;   //button3 touching
                        temp = PDR0;
                        temp &= 0xbf;
                        PDR0 = temp;
                        PDR1 = 0xff;
                        break;
                    case HighestMode_Button4:
                        PDR6 = 0xff;   //button4 touching
                        temp = PDR0;
                        temp |= 0x40;
                        PDR0 = temp;
                        PDR1 = 0x00;
                        break;
                    case Checking:
                        PDR6 = 0xff;   //no button touch
                        temp = PDR0;
                        temp |= 0x40;
                        PDR0 = temp;
                        PDR1 = 0xFF;break;
                }

            }
            else
                if(get_mode() ==Normal_Mode)      //judge current mode
                {
                if((Get_Data() & BIT00_Button1)== NormalMode_Button1)
                                                  //judge button1 touching?
                    {
                        temp = PDR6;
                        temp &= 0xf7;
                        PDR6 = temp;
                    }
                    else
                    {
                        temp = PDR6;
                        temp |= 0x08;
```

```
                PDR6 = temp;
        }
    if((Get_Data() & BIT01_Button2) == NormalMode_Button2)
                                //judge button1 touching?
        {
                temp = PDR6;
                temp &= 0xef;
                PDR6 = temp;
        }
        else
        {
                temp = PDR6;
                temp |= 0x10;
                PDR6 = temp;
        }
    if((Get_Data() & BIT02_Button3) == NormalMode_Button3)
                                //judge button1 touching?
        {
                temp = PDR0;
                temp &= 0xbf;
                PDR0 = temp;
        }
        else
        {
                temp = PDR0;
                temp |= 0x40;
                PDR0 = temp;
        }
    if((Get_Data() & BIT03_Button4) == NormalMode_Button4)
                                //judge button1 touching?
        {
                PDR1 = 0x00;
        }
        else
        {
                PDR1 = 0xff;
        }
}
```

```
                                    PDR0 = temp;
                                    PDR1 = 0xff;
                                    break;
                                case 2:                 //open 2 leds
                                    temp = PDR6;
                                    temp &= 0xE7;
                                    PDR6 = temp;
                                    temp = PDR0;
                                    temp |= 0x40;
                                    PDR0 = temp;
                                    PDR1 = 0xff;
                                    break;
                                case 3:                 //open 3 leds
                                    temp = PDR6;
                                    temp &= 0xE7;
                                    PDR6 = temp;
                                    temp = PDR0;
                                    temp &= 0xBF;
                                    PDR0 = temp;
                                    PDR1 = 0xff;
                                    break;
                                case 4:                 //open 4 leds
                                    temp = PDR6;
                                    temp &= 0xE7;
                                    PDR6 = temp;
                                    temp = PDR0;
                                    temp &= 0xBF;
                                    PDR0 = temp;
                                    PDR1 = 0x00;
                                    break;
                            }
                        }


    }
}
```

```
/*****************************************************************************
-FunctionName:  E_INT
-Description:   external interrupt process, change mode by hard key.
-Input:         None
-Return:        None
*****************************************************************************/
__interrupt void E_INT (void)
{
        unsigned char temp;
        if(get_mode()  == Highest_Mode)             //judge current mode to change mode
                set_mode(Normal_Mode);
        else
                if(get_mode() == Normal_Mode)
                {

                        set_mode(Slippage_Mode);
                }
                else
                        if(get_mode() == Slippage_Mode)
                                set_mode(Highest_Mode);
        temp = EIC30;
        temp &= 0x7f;
        EIC30 = temp;
}

/*****************************************************************************
-FunctionName:  UART_T
-Description:   serial port transmit interrupt
-Input:         None
-Return:        None
*****************************************************************************/
#ifdef enableUART
__interrupt void UART_T (void)
{
        Transmit();             //transmit check data to TSC GUI
}
#endif
```

Document History

Document Title: AN205062 - F²MC-8FX Family MB95200H/210H Series Capacitance Touch Sensor

Document Number: 002-05062

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	—	HUAL	04/14/2010	Initial release
*A	5260584	HUAL	06/24/2016	Migrated Spansion Application Note “MCU-AN-500084-E-10” to Cypress format.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corporation. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2010-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.