
Touch Sensing Software

User Guide

TSSUG
Rev. 6
08/2013



How to Reach Us:**Home Page:**

freescale.com

Web Support:

freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions

Freescale, the Freescale logo, ColdFire, and Kinetis are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM Cortex-M0, and ARM Cortex-M4 are registered trademarks of ARM Limited.

© 2013 Freescale Semiconductor, Inc.



Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web are the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, see: freescale.com.

The following revision history table summarizes changes in this document.

Revision Number	Revision Date	Description of Changes
Rev. 1	07/2009	Launch Release
Rev. 2	06/2010	TSS Component Installation chapter added Creating a New TSS Project section updated TSS_SystemSetup.h File section updated Chapter 5 Integrating the TSS using TSS Component added Chapter 6.3.3 OnFault Callback
	07/2010	Added Appendix A, "Evaluation Board Options"
Rev. 3	04/2011	Update for TSS 2.5 release
Rev. 4	03/2012	Update for TSS 2.6 release
Rev. 5	08/2012	Update for TSS 3.0 release
Rev. 6	08/2013	Update for TSS 3.1 release



Chapter 1 Before You Begin

1.1	About touch sensing software solutions	1-1
1.2	TSS product installation	1-1
1.3	About this book	1-1
1.4	Reference material	1-2
1.5	Acronyms and abbreviations	1-2

Chapter 2 TSS IDE Setup

2.1	CodeWarrior for MCU v10.x	2-1
2.1.1	TSS libraries for CodeWarrior	2-1
2.1.2	Integrate the TSS Project	2-1
2.2	IAR Embedded Workbench	2-4
2.2.1	TSS libraries for IAR	2-5
2.2.2	Integrate the TSS project	2-5
2.3	MDK-ARM uVision for ARM	2-7
2.3.1	TSS libraries	2-7
2.3.2	Integrate the TSS project	2-7
2.4	Cosmic IdeaCPU08 for HCS08	2-8
2.4.1	TSS libraries	2-8
2.4.2	Integrate the TSS project	2-8

Chapter 3 The first TSS Application

3.1	Introduction	3-1
3.2	Project header files	3-1
3.3	TSS configuration	3-1
3.3.1	GPIO measurement method configuration	3-2
3.3.2	Decoders	3-2
3.3.3	Control-specific parameters (decoders)	3-3
3.4	MCU Bus initialization	3-3
3.5	TSS interrupt service routines installation	3-4
3.6	TSS initialization	3-4
3.6.1	OnInit callback function	3-4
3.7	Main loop	3-4
3.8	TSS system runtime configuration	3-5
3.8.1	System response time	3-5
3.8.2	Electrode sensitivity	3-6
3.8.3	Enabling electrodes	3-6
3.8.4	System activation	3-6
3.9	TSS Controls runtime configuration	3-6
3.9.1	Callback function	3-7

Chapter 4 TSS Features

4.1	C++ wrapper	4-1
4.2	System setup parameters	4-1
4.2.1	Keydetector selection	4-1

4.2.2	Simple low level routines	4-2
4.2.3	Instant signal values	4-3
4.2.3.1	Basic keydetector signals	4-4
4.2.3.2	AFID keydetector signals	4-4
4.2.3.3	Noise keydetector signals	4-4
4.2.4	Signal control settings	4-5
4.2.4.1	IIR filter	4-5
4.2.4.2	Shielding function	4-5
4.2.4.3	Signal normalization	4-6
4.2.5	Automatic sensitivity calibration	4-6
4.2.6	Baseline initialization	4-7
4.2.7	OnFault callback	4-7
4.2.8	OnInit callback	4-8
4.2.9	OnProximity callback	4-8
4.2.10	Trigger function	4-9
4.2.11	Low power function	4-9
4.2.12	Data corruption check	4-10
4.2.13	Stuck-key function	4-10
4.2.14	Negative baseline drop	4-10
4.2.15	Automatic hardware recalibration	4-11
4.2.16	Noise mode	4-11
4.2.17	FreeMASTER GUI support	4-11
4.2.18	Control private data	4-12
4.2.19	Diagnostic messages	4-12
4.2.20	Electrodes configuration	4-12
4.2.21	Controls configuration	4-13
4.2.21.1	Matrix-specific parameters	4-14
4.2.21.2	Keypad electrode groups	4-15
4.2.22	GPIO measurement method settings	4-15
4.2.22.1	GPIO strength	4-15
4.2.22.2	GPIO slew rate	4-15
4.2.22.3	Default electrode level	4-15
4.2.22.4	Noise amplitude filter	4-16
4.2.22.5	Hardware timer configuration	4-16
4.2.23	TSI measurement method settings	4-18
4.2.23.1	TSI autocalibration	4-18
4.2.23.2	TSI active mode clock	4-19
4.2.23.3	TSI low power mode clock	4-20
4.2.23.4	TSI delta voltage	4-21
4.2.23.5	TSI noise mode	4-21
4.3	Store and Load System configuration	4-22
4.4	Example of system setup parameters encoded in the TSS_SystemSetup.h	4-23
4.5	TSS version information	4-26

Chapter 5 Touch Sensing Algorithms

5.1	TSI Module Based Touch Sensing Method	5-1
5.1.1	TSIL sample electrode control	5-2
5.2	GPIO Based Capacitive Touch Sensing Method	5-2
5.2.1	Electrode capacitive sensing algorithm	5-4
5.2.1.1	GPIO based low level routine	5-5
5.2.2	Selecting the proper timer frequency and external pull-up resistor	5-6
5.2.3	MCU frequency	5-6
5.2.4	Voltage trip point (Vih)	5-7
5.2.5	Sensitivity and range	5-7
5.3	Key Detect Method	5-8
5.3.1	Basic key detector	5-9
5.3.2	AFID key detector	5-10
5.3.3	Noise key detector	5-11
5.4	Triggering	5-12
5.5	Low power feature	5-13
5.5.1	Low power calibration (Kinetis ARM Cortex-M4 silicon 2, 3)	5-14
5.6	Proximity feature	5-14
5.7	Automatic Sensitivity Calibration	5-15
5.8	Baseline tracking	5-16
5.8.1	Negative baseline drop	5-17
5.9	Debouncing	5-17
5.10	IIR filter	5-18
5.11	Noise amplitude filter	5-18
5.12	Shielding function and Water tolerance	5-18
5.12.1	Standard shielding function	5-18
5.12.2	Water tolerance mode	5-19
5.13	Decoder Functions	5-19
5.13.1	Keypad	5-20
5.13.2	Rotary	5-21
5.13.3	Slider	5-22
5.13.4	Analog rotary	5-23
5.13.5	Analog Slider	5-23
5.13.6	Matrix	5-24

Chapter 6 TSS Component

6.1	Introduction	6-1
6.1.1	Integrating TSS Component	6-1
6.2	Configuring TSS Component	6-2
6.2.1	TSS component properties	6-2
6.2.2	TSS component Events	6-4
6.2.3	TSS component configuration related to Configure method	6-5
6.2.4	Generating TSS component code	6-8

Chapter 7 Using TSS FreeMASTER GUI

7.1	FreeMASTER Introduction	7-1
7.1.1	FreeMASTER integration	7-1
7.1.2	Communication interfaces	7-2
7.2	Signal visualization	7-3
7.3	Configuration panel	7-5
7.3.1	System config	7-6
7.3.2	Electrodes configuration	7-6
7.3.3	Controls configuration	7-7

Appendix A Evaluation Board Options

Chapter 1 Before You Begin

1.1 About touch sensing software solutions

The Touch Sensing Software (TSS) solution transforms a standard Freescale microcontroller into a proximity capacitive touch sensor controller. The touch sensor controller has an ability to manage multiple touch pad configurations and mechanical keys while maintaining its standard Microcontroller (MCU) control function. The touch sense library enables capacitive sensing for the entire Freescale S08, ColdFire V1 family, Kinetis L family of microcontrollers based on ARM[®]Cortex[™]-M4 Kinetis and ARM[®]Cortex[™]-M0+ cores. The library provides commonly used touch sense decoding structures such as a keypad, a rotary, a slider, an analog slider, an analog rotary, and a matrix. It is implemented in a software-layered architecture to enable seamless integration into the application code, migration to other Freescale MCUs, and to support customization.

For MCUs that do not contain a hardware Touch Sense Input (TSI) module, the TSS does not require any MCU peripheral modules except one or more timer modules, leaving all other MCU hardware peripherals for the main application. In certain configurations, it also enables the operation without the need for a timer module.

This document describes how to create and set up a touch sensing project by using the CodeWarrior Development Studio for microcontrollers, IAR Embedded Workbench[®] for ARM or MDK-ARM uVision and the Touch Sensing Library, or to integrate the Touch Sensing software into an existing application. This document also describes using an optional TSS Processor Expert Component in the CodeWarrior environment.

See the *Touch Sensing Software API Reference Manual* (TSSAPIRM) for more information regarding the Touch Sensing library interface and registers.

1.2 TSS product installation

The TSS library files are distributed as a binary library and partly in source code for compile-time configuration and optimization. To install TSS on your host PC:

1. Download the latest version of the TSS setup file by navigating to freescale.com and searching for Xtrinsic Touch-Sensing Software.
2. Run the self-extracting setup executable and follow the on-screen instructions.

When the system prompts for the destination folder, either accept the default value or provide another path.

1.3 About this book

The following table shows the summary of chapters in this document.

Table 1-1. TSSUG Description

Chapter Title	Description
Before you begin	This chapter provides the preliminary steps required before reading the document.
TSS IDE Setup	This chapter describes how to integrate the TSS project into an existing project.
The first TSS application	This chapter introduces the basic features of TSS.
TSS features	This chapter discusses the available TSS features.
Touch Sensing Algorithms	This chapter describes TSS algorithms and their use.
TSS Component	This chapter describes the TSS component.
Using TSS FreeMASTER GUI	This chapter describes how to use the TSS FreeMASTER GUI along with a TSS project.

1.4 Reference material

Use this document with:

- *Touch Sensing Software API Reference Manual* (document TSSAPIRM)

For better understanding, see the following documents.

- *Designing Proximity Sensing Electrodes* Application Note (AN3863)
- *Touch Panel Applications Using the MC34940/MC33794 E-Field IC* Application Note (AN1985)
- *Writing Touch Sensing Software Using TSI Module* Application Note (AN4330)
- *Pad Layout* Application Note (AN3747)
- *How to Implement a Human Machine Interface Using the Touch Sensing Software Library* Application Note (AN3934)
- *TSI module application on the S08PT family* Application Note (AN4431)
- CodeWarrior Help

1.5 Acronyms and abbreviations

AFID	Advanced Filtering and Integration Detection
API	Application Programming Interface
ASC	Automatic Sensitivity Calibration
EVB	Evaluation Board
FreeMASTER	PC to MCU communication and visualization tool
MCU	Microcontroller
TSI	Touch Sensing Interface — A microprocessor peripheral module
TSIL	Touch Sensing Input Lite
TSS	Touch Sensing Software

Chapter 2 TSS IDE Setup

2.1 CodeWarrior for MCU v10.x

The CodeWarrior for MCU v10.x supports the TSS project development for all supported MCU families. To get the latest CodeWarrior version, navigate to freescale.com and search for CodeWarrior.

2.1.1 TSS libraries for CodeWarrior

The binary library files must match the platform for which they were built. The lib folder contains the binary files generated by CodeWarrior.

Folder (build tools)	Name of the library	Freescall MCU family
lib_cw (CodeWarrior Freescale ARM/CFV/HCS08 build tools)	TSS_CV1.a	ColdFire V1 and ColdFire+
	TSS_KXX_M4.a	ARM Cortex-M4
	TSS_S08.lib	HCS08
lib_cw_gcc (CodeWarrior GCC ARM build tools)	libTSS_KXX_M0.a	ARM Cortex-M0+
	libTSS_KXX_M4.a	ARM Cortex-M4
	libTSS_KXX_M4_FPU.a	ARM Cortex-M4 with FPU support

Figure 2-1. TSS libraries

2.1.2 Integrate the TSS Project

To integrate the TSS project into an existing project:

1. Select **Start > Programs > Freescale CodeWarrior > CW for MCU v10.x > CodeWarrior**.
2. Select a workspace with the selected project in the workspace launcher window or create a new project in workspace.
3. The **CodeWarrior Projects** window should display a project structure as shown in [Figure 2-2](#).

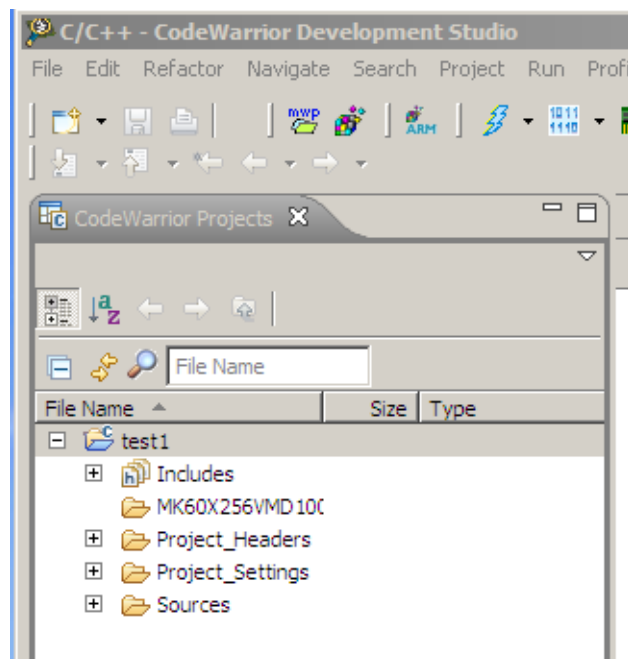


Figure 2-2. Existing project structure

4. When adding TSS to an existing project, create a new folder for TSS files. From the CodeWarrior file menu, select **File > New > Folder**; enter a name for the folder and click **Finish**.

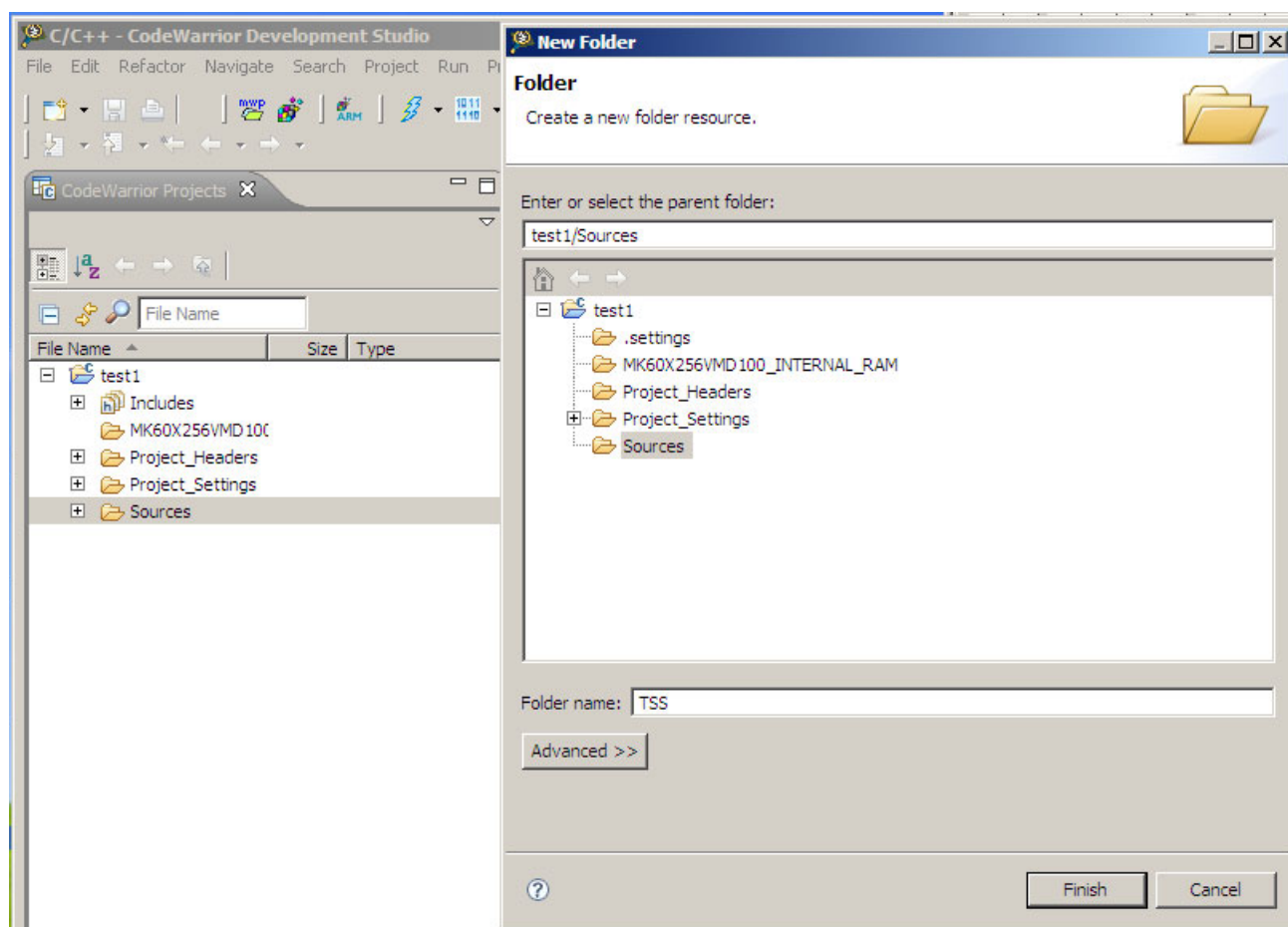


Figure 2-3. Creating a folder

5. In the CodeWarrior projects window, click the right mouse button on the newly created folder to select this item.
 - From the context menu, select **Add Files**. The **Open** dialog box appears.
 - Locate the TSS files in the lib\shared directory of the TSS installation folder.
 - Select the files related to the selected MCU family, and click **Open**. The File and Folder Import dialog box will open.
 - Select how files and folders should be imported in the project and click **OK**.

All files from the lib\shared directory must be used for all S08, ColdFire V1, ColdFire+, ARM Cortex-M4, and ARM Cortex-M0+ projects. The binary library files must match the platform for which they were built. For more details, see [Section 2.1.1, “TSS libraries for CodeWarrior.”](#)

The CodeWarrior links or adds selected files into the project.

- Add all files from the directory lib/shared and TSS binary library file for a target platform.
- Alternatively, you can copy the library files into the application folder and add files to the project from this location. Because the files should not be modified in the user application, both processes are acceptable and provide the following advantages:

- Copying the files to the project, keeps the project isolated and unaffected by external resources.
- Referencing the files at the install location by multiple different application projects facilitates subsequent upgrades to the new TSS library versions.

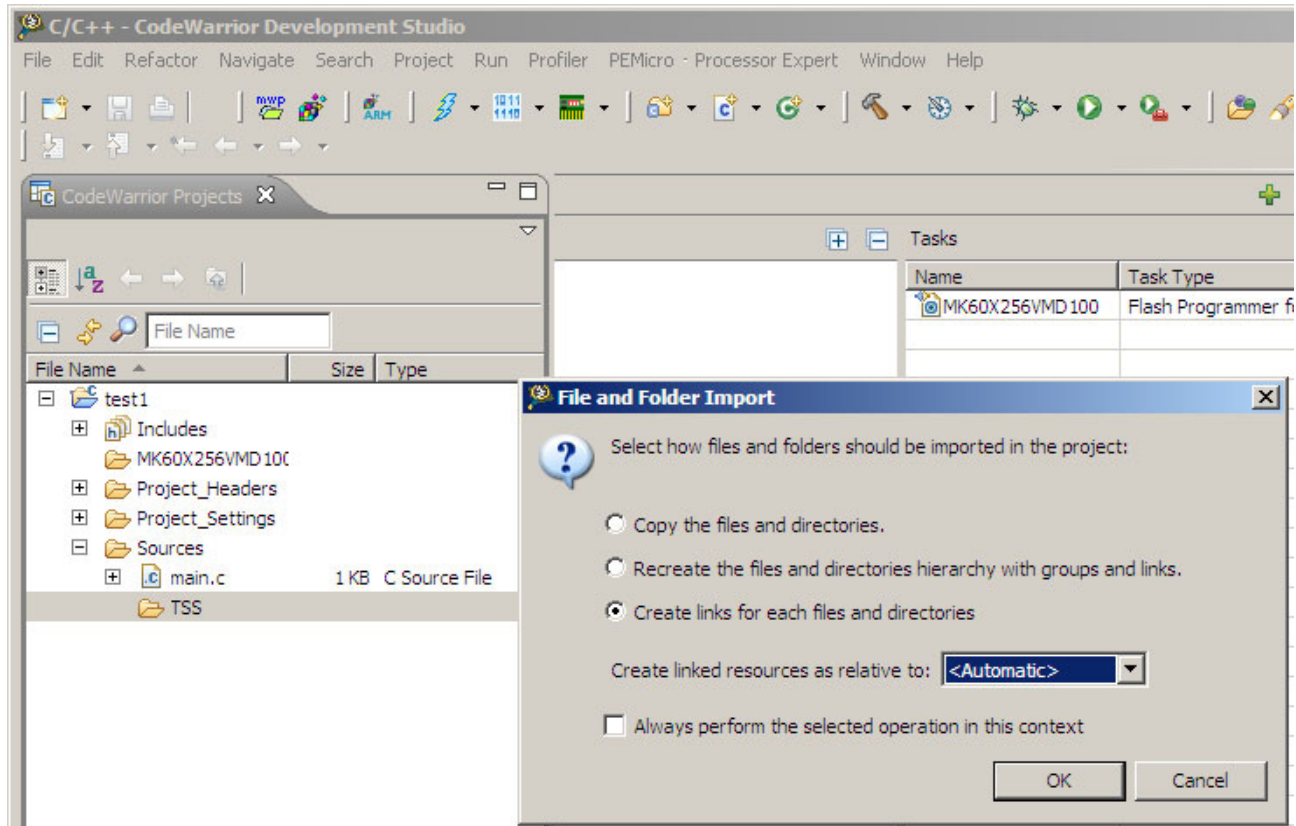


Figure 2-4. Adding files to the group

6. After library files are added to the project, create the TSS_SystemSetup.h configuration header file in the application folder either by copying an example file provided in the examples folder to your project folder and modifying its content, by creating the file manually, or by using the TSS Component for Processor Expert. For more details, see [Section 3.3, “TSS configuration.”](#)
7. If everything is ready for compilation, check whether the following paths are correctly defined in the **Project Properties > C/C++ Build > Settings**:
 - Include Search Paths in the TSS directory e.g. "\${ProjDirPath}/../LIB/shared"
 - Additional Library Paths in the selected library e.g. "\${ProjDirPath}/../LIB/lib_cw/TSS_KXX_M4.a"

2.2 IAR Embedded Workbench

Kinetis MCUs based on ARM Cortex-M4 and ARM Cortex-M0+ cores and Kinetis L family of microcontrollers require the IAR Embedded Workbench for ARM. The latest version of the IAR Embedded Workbench for ARM can be obtained at www.iar.com.

2.2.1 TSS libraries for IAR

The binary library files must match the platform for which they are built. Binary files provided for IAR Embedded Workbench are listed in the the following table.

Folder (build tools)	Name of the library	Freescal MCU family
lib_iar (IAR ARM compiler)	TSS_KXX_M0.a	ARM Cortex-M0+
	TSS_KXX_M4.a	ARM Cortex-M4

Figure 2-5. TSS libraries

2.2.2 Integrate the TSS project

To integrate the TSS project into an existing project do the following:

1. Select **Start > Programs > IAR Systems> IAR Embedded Workbench for ARM > IAR Embedded Workbench**
2. From the IAR IDE menu bar, select **File > Open > Workspace**.
3. On the **Open Workspace** dialog box, find the project file (*.eww) and click **Open**. The IDE opens the Project.eww as shown in the following figure.

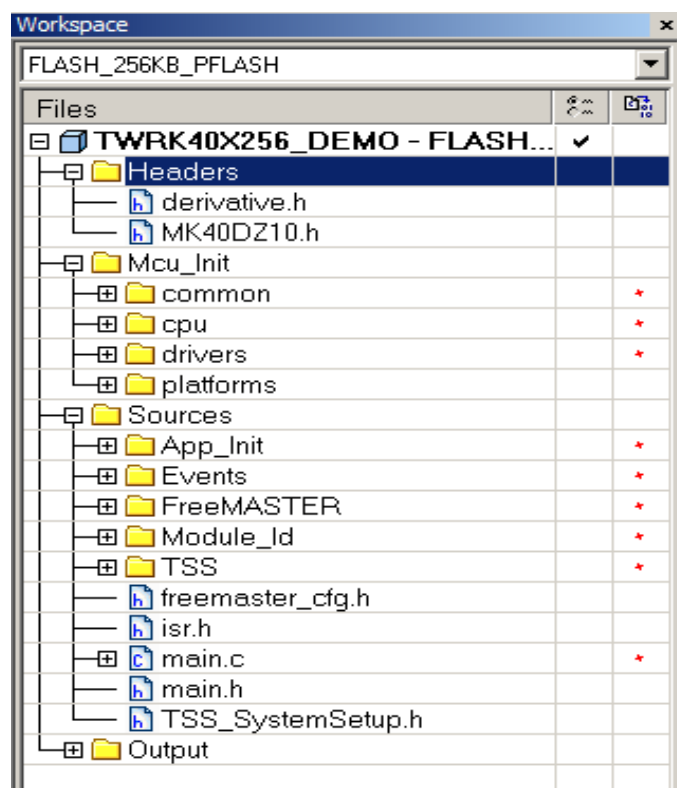


Figure 2-6. Existing project structure

4. When adding TSS to an existing project, create a group to store logically grouped TSS files. From the IAR main menu, select **Project > Add Group**. Enter a name for the group in the dialog box and click OK.
5. In the workspace window, click the file group created to select this item.
 - From the IAR main menu bar, select **Project > Add Files**. **Add File** dialog box appears.
 - Locate the TSS files in the lib\shared directory of the TSS installation folder.
 - Select the files related to the appropriate MCU family and click Open.

All files from the lib\shared directory can be used for ARM Cortex-M4 and ARM Cortex-M0+ projects. The binary library files must match the platform for which they are built. For more details see [Section 2.2.1, “TSS libraries for IAR.”](#)

 - The IAR IDE adds the selected files into the project.

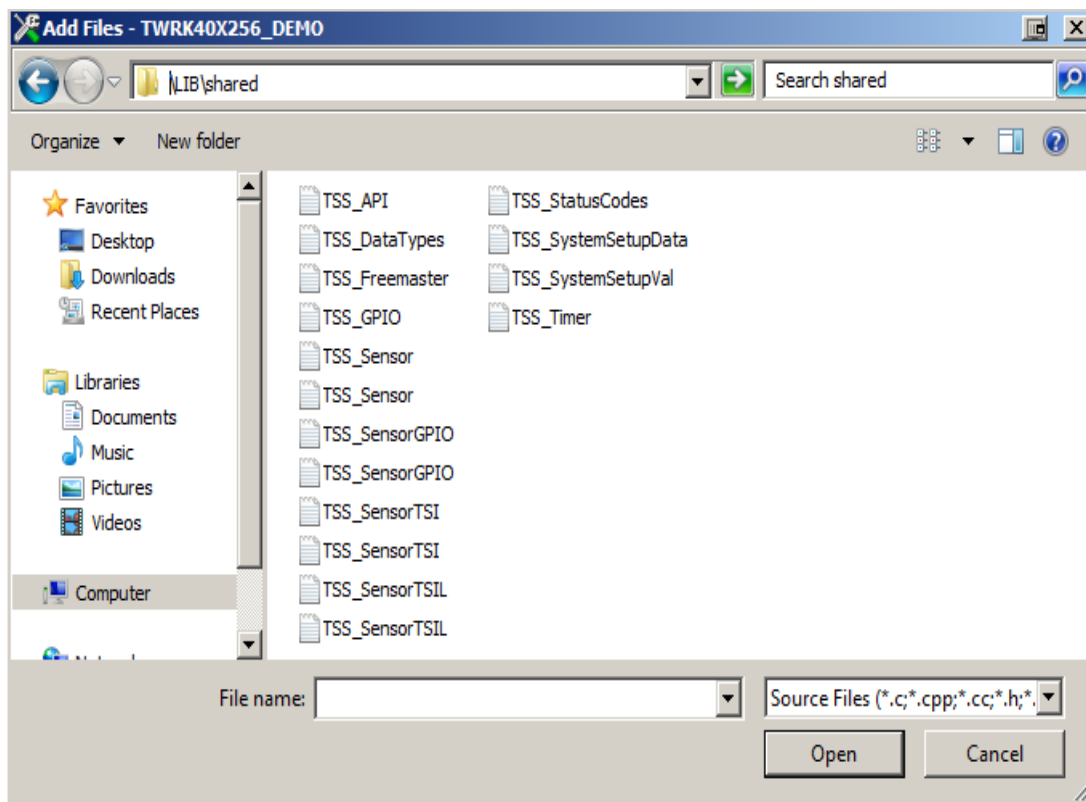


Figure 2-7. Adding files to the group

6. After the library files are added to the project, create the TSS_SystemSetup.h configuration header file in the application folder either by copying an example file provided in the examples folder to your project folder and modify its content, or by creating the file manually. For more details, see [Section 3.3, “TSS configuration.”](#)
7. If everything is ready for compilation, check whether the following paths are correctly defined in the **Project Options > C/C++ Compiler > Preprocessor**:
 - Additional Include Directory to the TSS directory e.g. "\$PROJ_DIR\$..\LIB\shared"

- Additional Include Directory to the library TSS_KXX_M4.a or TSS_KXX_M0.a e.g.
"\$PROJ_DIR\$\\..\\LIB\\lib_iar"

2.3 MDK-ARM uVision for ARM

Kinetis MCUs based on ARM Cortex-M4 and ARM Cortex-M0+ cores and Kinetis L family of microcontrollers require MDK-ARM uVision. Get the latest version of MDK-ARM uVision at www.keil.com.

2.3.1 TSS libraries

The binary library files must match the platform for which they are built. Binary files provided for MDK-ARM uVision are listed in the following table.

Table 2-1. TSS libraries

Folder (build tools)	Name of the library	Freescal MCU family
lib_uv4 (MDK-ARM compiler)	TSS_KXX_M0.lib	ARM Cortex-M0+
	TSS_KXX_M4.lib	ARM Cortex-M4

2.3.2 Integrate the TSS project

To integrate the TSS project into an existing project:

1. Select **Start > Keil uVision4**
2. From the uVision IDE menu bar, select **Project > Open Project**.
3. In the **Open Project** dialog box, find the project file (*.uvproj) and click **Open**. The IDE opens the Project.uvproj as shown in the following figure.

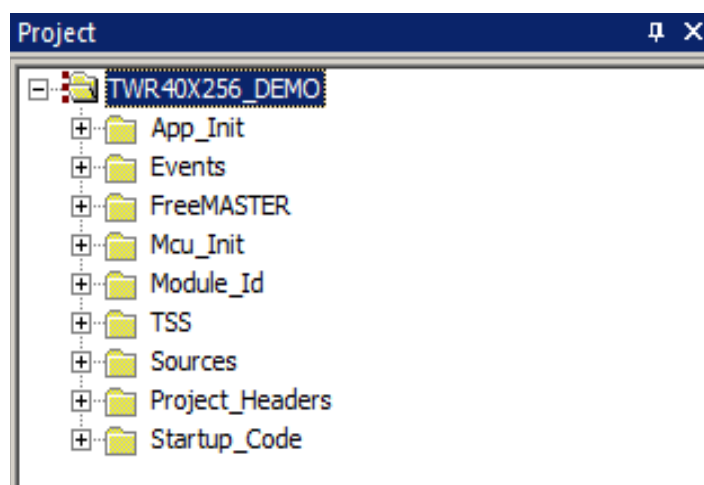


Figure 2-8. Existing project structure

4. When adding the TSS to an existing project, create a group to store logically grouped TSS files. Bring up the context menu in the Project Window and choose **Add Group**.
5. In the project window, click the selected file group.
 - Open the Context Menu and select **Add Files to Group**.
 - Locate the TSS files in the lib\shared directory of the TSS installation folder.
 - Select the files related to the selected MCU family, and click **Add**.

All files from the lib\shared directory can be used for ARM Cortex-M4 and ARM Cortex-M0+ projects. The binary library files must match the platform for which they are built.

The uVision IDE adds the files chosen into the project.
6. When the library files are added to the project, create the TSS_SystemSetup.h configuration header file in the application folder either by copying an example file provided in the examples folder to your project folder and modifying its content, or by creating the file manually. For more details see [Section 3.3, “TSS configuration.”](#)
7. If everything is ready for compilation, check whether the following paths are correctly defined in the **Options for Target ‘Project’ > C/C++** :
 - Additional Include Directory to the TSS directory e.g `"..\..\LIB\shared"`
 - Additional Include Directory to the library TSS_KXX_M4.lib or TSS_KXX_M0.lib e.g. `"..\..\LIB\lib_uv4"`

2.4 Cosmic IdeaCPU08 for HCS08

HCS08 family of microcontrollers requires the installation of the Cosmic IdeaCPU08. Get the latest version at www.cosmic-software.com.

2.4.1 TSS libraries

The binary library files must match the platform for which they are built. Binary files provided for Cosmic IdeaCPU08 are listed in the following table.

Table 2-2. TSS libraries

Folder (build tools)	Name of the library	Freescale MCU family
lib_idea (IdeaCPU08 compiler)	TSS_S08.lib	HCS08

2.4.2 Integrate the TSS project

To integrate TSS project into an existing project:

1. Select **Start > Cosmic Cx6808**
2. From the IdeaCPU08 IDE menu bar, select **Project > Load**.

3. In the **Open Project** dialog box, find the project file (*.prj) and click **Open**. The IDE opens the Project.prj as shown in the following figure.

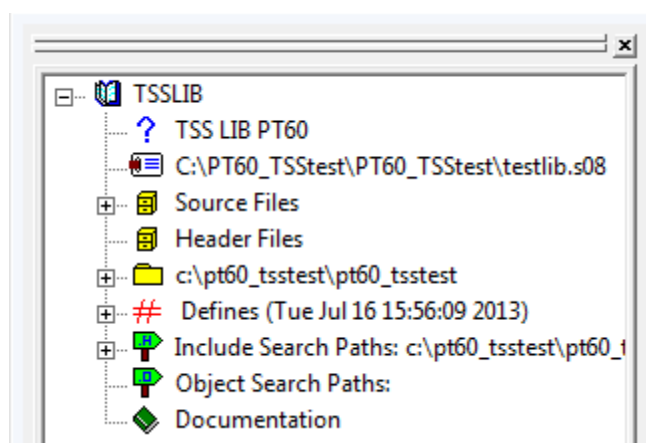
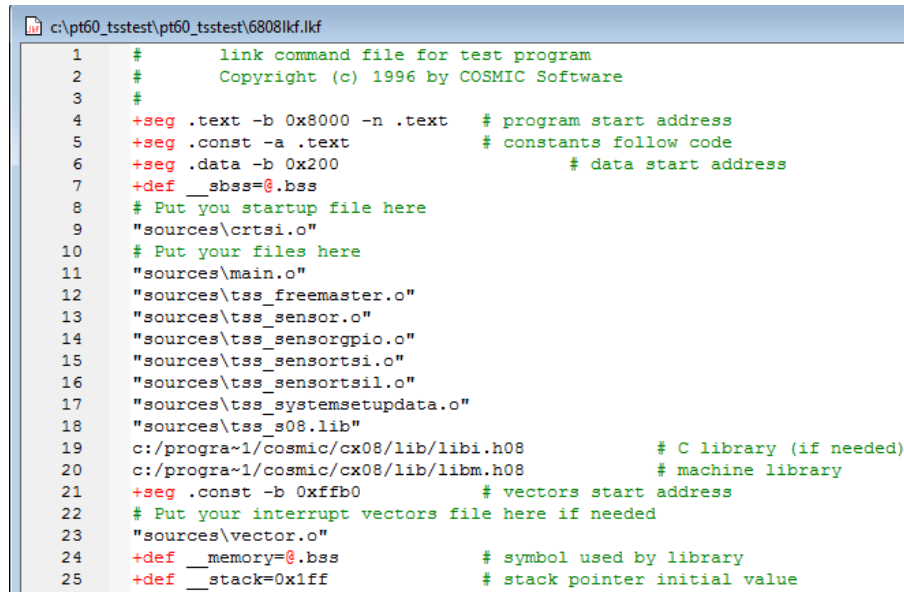


Figure 2-9. Existing project structure

4. When adding TSS to an existing project, create a group to store logically grouped TSS files. In the project window, click the Source Files to select this item. Open the Context Menu and select **Add Group**.
5. In the project window, click the file group created to select this item.
 - Open the Context Menu and select **Add File**.
 - Locate the TSS files in the lib\shared directory of the TSS installation folder.
 - Select the files related to the selected MCU family and click **Open**.

All files from the lib\shared directory can be used for HCS08 projects. The Cosmic IdeaCPU08 IDE adds the selected files to the project.
6. Create the link command file such as 6808LKF.LKF. The file defines memory areas and paths to object files and library files as shown in the [Figure 2-10](#). The TSS binary files must match the platform for which they are built.



```

1  #      link command file for test program
2  #      Copyright (c) 1996 by COSMIC Software
3  #
4  +seg .text -b 0x8000 -n .text      # program start address
5  +seg .const -a .text              # constants follow code
6  +seg .data -b 0x200                # data start address
7  +def __sbss=@.bss
8  # Put you startup file here
9  "sources\crtssi.o"
10 # Put your files here
11 "sources\main.o"
12 "sources\tss_freemaster.o"
13 "sources\tss_sensor.o"
14 "sources\tss_sensorgpio.o"
15 "sources\tss_sensortsi.o"
16 "sources\tss_sensortsil.o"
17 "sources\tss_systemsetupdata.o"
18 "sources\tss_s08.lib"
19 c:/progra~1/cosmic/cx08/lib/libi.h08      # C library (if needed)
20 c:/progra~1/cosmic/cx08/lib/libm.h08      # machine library
21 +seg .const -b 0xffb0              # vectors start address
22 # Put your interrupt vectors file here if needed
23 "sources\vector.o"
24 +def __memory=@.bss                # symbol used by library
25 +def __stack=0x1ff                # stack pointer initial value

```

Figure 2-10. Link command file

7. After the library files are added to the project, create the TSS_SystemSetup.h configuration header file in the application folder either by copying an example file provided in the examples folder to your project folder and modifying its content, or by create the file manually. For more details, see [Section 3.3, “TSS configuration.”](#)
8. If everything is ready for compilation, specify the global project defines. In the project window, click the item **Defines**. Open the Context Menu and add defines `__HCS08__` and `TSS_HCS08_MCU`.

Chapter 3 The first TSS Application

3.1 Introduction

This chapter describes the steps to configure and initialize a basic project by using the TSS library.

3.2 Project header files

After adding all files into the project, customize the project as needed for an application. To use the TSS functionalities in the application, the TSS_API.h file must be included by the C code files.

Additionally, it is necessary to place a derivative.h header in the application. The file is used to add a processor-specific header and for an automatic detection of the MCU platform.

The TSS_API.h file contains the function prototypes and variable declaration of the Application Interface. TSS is configured by editing the TSS_SystemSetup.h file.

3.3 TSS configuration

Create the TSS_SystemSetup.h file in one of the following ways:

- Copy an example file from the examples directory, which is located in the TSS installation folder, to your project directory. After the file is copied, edit it to meet the specific application needs.
- Copy the default configuration file, which is located in examples/default_config directory, and edit it.
- Create the file manually and provide the configuration options for the library.
- Use the TSS component in the Processor Expert-enabled project. The component generates the header file based on the settings selected in the graphical interface.

Configure all necessary parameters to set up the TSS for a new application in the TSS_SystemSetup.h file:

#define TSS_ENABLE_DIAGNOSTIC_MESSAGES

This macro is optional. If diagnostic messages or recommendations display during compile time, such as which interrupt service routine needs to be assigned to the vector table, set the macro to 1. Whenever code is recompiled, the TSS_SystemSetupData.c and TSS_SystemSetupVal.h files verify whether values provided in the TSS_SystemSetup.h file are valid for all parameters. If a parameter is out of range, the compiler displays an error pointing to an out of range parameter.

#define TSS_ONINIT_CALLBACK

When using the TSS library, special initialization requirements are handled by the application. Define the TSS_ONINIT_CALLBACK macro as the name of your application callback function. The TSS library calls this user-defined function whenever the TSS_Init() function is called. The

function performs peripheral initializations. Processor Expert TSS component generates an initialization function which, in turn, automatically enables the clock for all peripherals used by the TSS library. Call this function from the OnInit callback. For more information, see [3.6.1, “OnInit callback function”](#).

#define TSS_N_ELECTRODES

The TSS_N_ELECTRODES macro defines the electrode count in the application. A valid count is between 1 and 64.

#define TSS_Ex_TYPE

To select a measurement method for electrode number *x*, define the TSS_Ex_TYPE macro to one of the values described below:

- TSI_nCH_m — Where *n* represents the TSI module number and *m* represents a channel number.
- GPIO — For a general-purpose I/O pins. The GPIO type is defined as default for each electrode; a macro is not required.

3.3.1 GPIO measurement method configuration

If the GPIO method is selected for any electrode, declare a pin and a bit number for the electrode. The first electrode is zero and the last electrode is TSS_N_ELECTRODES–1. The TSS uses this information to set up all electrodes as needed by the software. The format for the declarations is:

- TSS_Ex_P — Where X is the electrode number and P refers to the Port letter where the electrode is located.
- TSS_Ex_B — Where X is the electrode number and B refers to the bit number of the port where the electrode is located.

For the GPIO algorithm, select the hardware timer, which is used by the TSS library sampling algorithm, by using a macro TSS_HW_TIMER. You can select a TPM timer, MTIM, or FTM, provided that the selected timer is available on the MCU. To get the optimal capacitance sensing results, tune the timer input clock in the application.

To set up the timer prescaler factor, define the TSS_SENSOR_PRESCALER macro accordingly. To prevent the TSS sensor from blocking the application in the event of a large capacitance or a short-to-ground, set the TSS sensor timeout and define the value as the TSS_SENSOR_TIMEOUT macro.

3.3.2 Decoders

An electrode can be read either with the decoders, or with a direct access to an electrode status.

- The decoders provide the highest level of abstraction in the library. In this layer, the key detector information about touched and untouched electrodes is interpreted in the form of controls. The controls provide the electrode status in a behavioral way, such as callbacks on the defined finger touch or movement events. This approach has a higher memory footprint, but is easier to use in an application.

- Direct access to electrode status (touched or untouched) is provided by `tss_au8ElectrodeStatus[]` array where each bit represents one electrode, a 1 represents a touched electrode, and a 0 represents an untouched electrode.

3.3.3 Control-specific parameters (decoders)

#define TSS_N_CONTROLS

Set the number of decoder controls available in the system by using the `TSS_N_CONTROLS` macro. The TSS supports up to 16 controls.

#define TSS_Cx_TYPE

Indicates which type of control is configured. Each control type has a different function. There are six different control types available:

- keypad (`TSS_CT_KEYPAD`)
- slider (`TSS_CT_SLIDER`)
- rotary (`TSS_CT_ROTARY`)
- analog slider (`TSS_CT_ASLIDER`)
- analog rotary (`TSS_CT_AROTARY`)
- matrix (`TSS_CT_MATRIX`)

#define TSS_Cx_INPUTS

Indicates which electrodes are assigned to a given control. The electrodes are defined in the form of array. For details about inputs definitions, see [4.2.21, “Controls configuration”](#).

If Cx is a matrix control — `TSS_Cn_INPUTS_NUM_X` indicates how defined electrodes are used as X axis. A valid range is 2-14 for the axis.

If Cx is a keypad control — The `TSS_Cx_KEYS` defines the group keys whenever more than one electrode forms one key. It is only valid for a keypad control.

#define TSS_Cx_STRUCTURE

Creates the control and status structure for the configured control.

#define TSS_Cx_CALLBACK

The touch sensing software library uses this function to call a user-defined function when an event is configured in the control and the callback enabler is active for that control. This function must be included in one of the higher layers of the application (in `main.c` or another application file). For details about control callback see [3.9.1, “Callback function”](#).

3.4 MCU Bus initialization

The TSS library does not perform any device initialization other than the selected GPIO pins and timer modules. Any other initialization relevant to the TSS, such as peripheral clock enabling, pin multiplexers and so on, should be handled in the `TSS_fOnInit()` callback function. This callback function is called by the TSS library code during the TSS initialization. For Processor Expert TSS component, the user callback function may call the `TSS_InitDevices()` method of the TSS component which contains the necessary

initialization code generated automatically. For more information about the OnInit callback function, see [Section 4.2.8, “OnInit callback.”](#)

3.5 TSS interrupt service routines installation

- TSI Module

If an application uses TSI module, configure the TSI interrupt service routine manually. For instance, for CodeWarrior, an isr.h file is included by the startup code file. Isr header file contains the following code:

```
#undef VECTOR_042
#define VECTOR_042 TSS_TSI0Isr
```

- Timer Module (GPIO method only)

The timer configuration is executed by the macro values defined in the TSS_SystemSetup.h file. This option is used only with the GPIO methods. Configure the timer interrupt service routine manually.

For instance, for CodeWarrior, an isr.h file is included by the startup code file. Isr header file contains the following code:

```
#undef VECTOR_033
#define VECTOR_033 TSS_HWTimerIsr
```

3.6 TSS initialization

To initialize the TSS, call the TSS_Init() function after initializing the MCU device. Typically, you initialize the TSS library early after device's start-up in the main() function.

3.6.1 OnInit callback function

The TSS library provides the OnInit Callback function which is not optional and must always be defined in the application. The function should contain initialization code for the hardware related to the TSS (clock gating, pin multiplexers and so on). The callback is executed during the TSS_Init function call. The default callback name is TSS_fOnInit. To rename it, use the following definition:

```
#define TSS_ONINIT_CALLBACK TSS_fOnInit
```

Where:

- TSS_fOnInit is the callback function name as defined in the application code.

For the Processor Expert TSS component, the function code is generated automatically and enables clocks and pin multiplexers for all peripherals used by the TSS.

3.7 Main loop

To ensure that the TSS is operating properly, the application must call either the TSS_Task() or TSS_TaskSeq() function periodically within the main loop. The difference between the TSS_Task() and TSS_TaskSeq() is in how the execution time is spent within the main loop.

- The `TSS_Task()` function performs all the sensing tasks for all active electrodes at once. The execution time depends on the number of active electrodes, the sensing time of each electrode, and the duration of the TSI module measurement which is processed in parallel with other measurement methods. The function reports the status `TSS_STATUS_PROCESSING` if sensing all electrodes is not complete and `TSS_STATUS_OK` if the sensing all electrodes is complete.
- The `TSS_TaskSeq()` function performs the sensing task for active electrodes sequentially. Each single call of the function performs the sensing task for one electrode. The execution time depends only on the sensing time of one electrode. The function reports the status `TSS_STATUS_PROCESSING` if the sensing electrodes is not complete and `TSS_STATUS_OK` if the sensing all electrodes is complete.

While `TSS_Task` or `TSS_TaskSeq` are running, do not change the system configuration. Perform any TSS register settings only after the reported status is `TSS_STATUS_OK`.

3.8 TSS system runtime configuration

TSS is controlled and configured in the application runtime by writing values into virtual TSS registers. The registers are stored in data structures that are also accessible from the application code, but must not be modified directly. All write accesses must be done by calling the `TSS_SetSystemConfig()` function. Direct modification of the internal TSS structures are evaluated as a memory corruption fault (provided the `TSS_USE_DATA_CORRUPTION_CHECK` configuration option is enabled).

The easiest way to configure TSS is to use the TSS Processor Expert component that provides the `Configure()` method. This method configures all necessary registers according to properties set in the TSS Component Inspector window. For more details about the `Configure` method, see [Section 6.2.3, “TSS component configuration related to `Configure` method”](#).

To manually change the TSS configuration from the application code, call the `TSS_SetSystemConfig()` function with the register that needs to be changed. The new value for that register is passed as the second parameter to the `TSS_SetSystemConfig()` function. The TSS then writes the new value to the configuration register, and updates the checksum covering all configuration registers. A function prototype of `SetSystemConfig` is:

```
uint8_t TSS_SetSystemConfig(uint8_t u8Parameter, uint16_t u16Value);
```

All available TSS registers are defined in the `TSS_API` header file in enum `TSS_CSSystem_REGISTERS`.

Typically, the TSS configuration starts by configuring the System Response Time register that determines how fast the TSS responds to any input signal changes. Next, the `DCTrackerRate` must be set (if enabled).

3.8.1 System response time

The system response time register determines how fast the TSS responds to input signal changes. To set this register, use the following code:

```
TSS_SetSystemConfig(System_ResponseTime_Register, 5);
```

3.8.2 Electrode sensitivity

If the macro `TSS_USE_AUTO_SENS_CALIBRATION` is set, the function instantly controls levels of sensitivity for each electrode according to noise and touch tracking information. If the feature is enabled you do not need to set up sensitivity registers.

Otherwise, use the `System_Sensitivity_Register` base constant and add an electrode number to it:

```
TSS_SetSystemConfig(System_Sensitivity_Register + 0u, 0x40);
```

The sensitivity is set for each electrode individually. If the sensitivity registers are set while the Automatic Sensitivity Calibration is enabled, the values are used as initial sensitivity values.

3.8.3 Enabling electrodes

To enable an electrode, use `System_ElectrodeEnablers_Register` as shown in the code example. Enable the first electrode in the first enablers register and so on.

```
TSS_SetSystemConfig(System_ElectrodeEnablers_Register + 0u, 0x01);
```

Note that enabling control automatically enables all electrodes assigned to the control. However, manual settings of the registers is supported.

3.8.4 System activation

After all registers are properly configured, TSS can be activated by writing the Enable bit in the `System_SystemConfig_Register` and also optionally by enabling the DCTracker algorithm as shown in the following code:

```
TSS_SetSystemConfig(System_SystemConfig_Register, (TSS_SYSTEM_EN_MASK |  
TSS_DC_TRACKER_EN_BIT));
```

This process enables only the DC tracker feature globally because each electrode has a distinct enabler bit. To set the DC tracker enabler, use the `System_DCTrackerEnablers_Register` as the `SetSystemConfig` first parameter of the function:

```
TSS_SetSystemConfig(System_DCTrackerEnablers_Register, 0x01);
```

3.9 TSS Controls runtime configuration

Each control object also has a configuration structure associated to it. Similar to the system setup, configuring the control behavior is accomplished by writing to control-specific virtual registers.

For example:

```
TSS_KeypadConfig(cKey0.ControlId, Keypad_Events_Register,  
(TSS_KEYPAD_TOUCH_EVENT_EN_MASK));  
TSS_KeypadConfig(cKey0.ControlId, Keypad_ControlConfig_Register,  
(TSS_KEYPAD_CALLBACK_EN_MASK | TSS_KEYPAD_CONTROL_EN_MASK));
```

The code shows how the keypad structure is configured to enable touch events for this control. The control and the callback are enabled whenever there is a change in events.

3.9.1 Callback function

After the system configuration code is complete, the callback routines can be coded. The callbacks are called from the TSS library whenever there is a new event detected by the system. The callback function is declared as follows:

```
void fCallback1(TSS_CONTROL_ID u8ControlId)
```

The parameter `u8ControlId` is the ID of the control that generated the callback. The callback code reads and processes the generated events.

Chapter 4 TSS Features

This chapter provides a summary of TSS features.

4.1 C++ wrapper

This chapter describes the C++ wrapper to interact with TSS and its configuration from within C++ code. It is designed to be built on top of the TSS to simplify application development and facilitate the use of TSS.

The C++ wrapper consists of the following files:

```
TSS_cpp.cpp - TSS System implementation, TSS Factory implementation
TSS_cpp.h - the interface for entire C++ API
TSS_cpp_arotary.cpp - ARotary implementation
TSS_cpp_aslider.cpp - ASlider implementation
TSS_cpp_const.h - TSS System constants and TSS Control constants
TSS_cpp_keypad.cpp - Keypad implementation
TSS_cpp_matrix.cpp - Matrix implementation
TSS_cpp_rotary.cpp - Rotary implementation
TSS_cpp_slider.cpp - Slider implementation
```

The interface is defined in a single header file TSS_cpp.h. All definitions are contained in a “tss” namespace. For detailed API description, refer to the TSS reference manual.

4.2 System setup parameters

This section describes static TSS configuration parameters in detail. Static parameters are configured in the TSS_SystemSetup.h file. This file contains the system setup parameters that must be configured before application compilation and linking.

4.2.1 Keydetector selection

The TSS library implements three versions of the keydetector. Note that some MCUs may provide only some of these implementations.

- **Basic key detector** uses algorithms for identification of touched and released electrodes by obtaining a baseline value, which is the "DC" value of the capacitance when no finger is present, and comparing it to the immediate capacitance value. The keydetector supports all TSS features without limitations.
- **AFID (Advanced Filtering and Integrating Detection) key detector** is based on using two IIR filters with different depths (short/fast and long/slow) and on integrating the difference between two filtered signals. This keydetector is more immune to noise, but it is not compatible with the shielding noise avoidance method and with the under-water operation.
- **Noise key detector** processes a noise signal from the TSI module supporting a noise mode. The noise mode feature is an alternative hardware measurement method intended for touch detection in an extremely noisy environment. The noise key detector cannot be selected as the main keydetector, but it extends Basic or AFID key detector. The noise keydetector is disabled by default. For more information about enabling noise mode, see [Section 4.2.16, “Noise mode”](#).

For more information about keydetector algorithms, see [Section 5.3, “Key Detect Method”](#).

To define the main keydetector version, use the following macros:

```
#define TSS_USE_KEYDETECTOR_VERSION X
```

Where *X* is the identifier of the main keydetector version and designates the following values:

- 1 — For the Basic keydetector
- 2 — For the AFID keydetector

If the keydetector version is not defined, the Basic keydetector is selected by default.

If AFID keydetector is selected, a weight of the signal filters is defined in the form of ratio $f_{\text{sample}}/f_{\text{cutoff}}$. To define filter weights, use the following definitions:

```
#define TSS_USE_AFID_FAST_FILTER_RATIO m
#define TSS_USE_AFID_SLOW_FILTER_RATIO n
```

Where:

- *m* defines ratio for the fast AFID filter.
- *n* defines ratio for the slow AFID filter. The value *m* must always be lower than *n*.

Each ratio value is defined as result of $f_{\text{sample}}/f_{\text{cutoff}}$ equation. The value f_{sample} is frequency of signal sampling and the value f_{cutoff} is cutoff frequency. Allowed values are: 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 220, 240, 260, 280, 300, 350, 400, 500, 600, 700, 800, 900, 1000.

[Table 4-1](#) shows keydetector availability against MCU families.

Table 4-1. Keydetector availability

MCU Family	Available Keydetectors
HCS08	Basic
Coldfire V1	Basic/AFID
Coldfire+	Basic/AFID
ARM Cortex-M0+	Basic/AFID/Noise
ARM Cortex-M4	Basic/AFID

If a selected keydetector is not available for the MCU, compilation is stopped with an error message.

To view a configuration example, see [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.2 Simple low level routines

The TSS library implements two low level measurement methods. Some MCUs may provide only one of these implementations.

- Standard low level routines provide full TSS functionality with background measurement. All triggering modes are available. Auto triggering is available only if TSI measurement method is used on, at least, one electrode.
- Simple low level routines require smaller code, but also provide limited functionality. The measurement is consecutive and the TSS_Task needs to wait for the end of measurement. Auto triggering function is not available. The simple low level is limited to one TSI module per device.

To enable simple low level routines, use the following macro in the TSS_SystemSetup.h file:

```
#define TSS_USE_SIMPLE_LOW_LEVEL 1
```

Table 4-2 shows feature availability against the MCU families and measurement methods.

Table 4-2. Simple low level availability

MCU Family	Available Measurement Methods	Low Level Options
HCS08	GPIO/TSI	Simple/Standard
Coldfire V1	GPIO	Simple/Standard
Coldfire+	GPIO/TSI	Standard
ARM Cortex-M0+	GPIO/TSI	Standard
ARM Cortex-M4	GPIO/TSI	Standard

If a configured feature is not available for the MCU, the compilation is stopped with an error message.

To view a configuration example, see [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.3 Instant signal values

The TSS library can also be configured to keep track of the instant signal values by storing them in an array. Note that each type of keydetector provides a different kind of tracking signals. The arrays are updated for each electrode each time the value is calculated. For a configuration example of the, see [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

Table 4-3 shows availability of the signals in keydetector types.

Table 4-3. Signals availability

Keydetector	Available Signals
Basic	Instant Delta / Instant Signal
AFID	Instant Delta / Instant Signal / Integration Delta
Noise	Instant Noise

4.2.3.1 Basic keydetector signals

To enable tracking of the instant delta value use the following macro in the TSS_SystemSetup.h file:

```
#define TSS_USE_DELTA_LOG 1
```

If enabled, the TSS library instantiates an array declared as follows:

```
int8_t tss_ai8InstantDelta[TSS_N_ELECTRODES];
```

The instant delta function is automatically enabled if analog slider, analog rotary, or matrix control is used, because the instant values are also required internally by the library. User application can have access to these values by reading the array, specifying the electrode number, and using the standard C array addressing encoding. A reading of the array in C:

```
Temp = tss_ai8InstantDelta[n];
```

To enable tracking of the instant signal value, use the following macro in the TSS_SystemSetup.h file:

```
#define TSS_USE_SIGNAL_LOG 1
```

If enabled, the TSS library instantiates an array declared as follows:

```
uint16_t tss_aul6InstantSignal[TSS_N_ELECTRODES];
```

The library stores the instant signal value for each electrode every time the value is calculated. User application can have access to these values by reading the array, specifying the electrode number, and using the standard C array that addresses encoding. A reading of the array in C:

```
Temp = tss_aul6InstantSignal[n];
```

4.2.3.2 AFID keydetector signals

To enable tracking of the instant integration delta value use the following macro in the TSS_SystemSetup.h file:

```
#define TSS_USE_INTEGRATION_DELTA_LOG 1
```

If enabled, the TSS library instantiates an array declared as follows:

```
int8_t tss_ai8IntegrationDelta[TSS_N_ELECTRODES];
```

An access can be obtained by reading the array, specifying the electrode number, and using the standard C array addressing encoding. A reading of the array in C:

```
Temp = tss_ai8IntegrationDelta[n];
```

4.2.3.3 Noise keydetector signals

To enable tracking of the instant noise signal value use the following macro in the TSS_SystemSetup.h file:

```
#define TSS_USE_NOISE_SIGNAL_LOG 1
```

If enabled, the TSS library instantiates an array declared as follows:

```
uint8_t tss_au8InstantNoise[TSS_N_ELECTRODES];
```

You can have access to these values by reading of the array, specifying the electrode number, and using the standard C array addressing encoding. A reading of the array in C:

```
Temp = tss_au8InstantNoise[n];
```


4.2.4 Signal control settings

The following chapters describe parameters which control the way how the raw signal is filtered and decoded.

4.2.4.1 IIR filter

The TSS library can enable the IIR filter feature. The filter processes current capacitance or noise values obtained from the low-level routines. It is functional with all kinds of low level sensor modules. Filtering may help to eliminate high-frequency noise modulated on the input signal and other external interferences. The IIR equation is internally set to ratio 1/3 (the current signal and the previous signal). For more information about IIR filter, refer to [Section 5.10, “IIR filter”](#).

To enable the IIR filter feature of the library, use the following definition:

```
#define TSS_USE_IIR_FILTER 1
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.4.2 Shielding function

The TSS library provides the shielding function. The function enables a compensation of a signal drift on the common electrode by the special shielding electrode. The shielding electrode is not intended to be touched. It measures overall environmental noise affecting the system. The function may help eliminate low frequency noise modulated on the capacitance signal, or protect the guarded system from detecting false touches caused by water drops. The function is relevant only for Basic keydetector.

To enable the shielding feature of the library, use the following definition:

```
#define TSS_USE_SIGNAL_SHIELDING 1
```

If the function is enabled then each electrode may be assigned its shielding electrode. To assign shielding electrode to an electrode, use the following definition:

```
#define TSS_En_SHIELD_ELECTRODE N
```

Where:

- n is the electrode number
- N is the number of shielding electrode in the range 0-63

If an electrode does not have a shielding electrode assigned, the shielding of this electrode is disabled. It is important to note that the DC-Tracker function on shielding and shielded electrode is executed in rate defined in SlowDcTrackerFactor register in cooperation with DcTrackerRate register. For more information on shielding function, refer to [Section 5.12, “Shielding function and Water tolerance”](#).

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.4.3 Signal normalization

The TSS library provides the signal normalization function. This function enables to resize the raw capacitive signal to the required level, which also affects the signal delta amplitude. This function is useful for analog controls like an analog slider, analog rotary, and matrix where it is important to normalize the signal delta into the 127 range for the best performance of an analog position calculation.

The required signal level can be reached with the combination of a signal divider and a multiplier value for each electrode separately.

To enable signal divider feature of the library, use the following definition:

```
#define TSS_USE_SIGNAL_DIVIDER 1
```

If the function is enabled the signal divider value of each electrode must be defined. To define the signal divider value of electrodes, use the following definition:

```
#define TSS_En_SIGNAL_DIVIDER N
```

Where:

- n is the electrode number
- N is the divider value in the range 0-255

If an electrode does not have a divider defined or it equals 0, the dividing of this electrode signal is disabled.

To enable signal multiplier feature of the library, use the following definition:

```
#define TSS_USE_SIGNAL_MULTIPLIER 1
```

If the function is enabled then the signal multiplier value of each electrode must be defined. To define the signal multiplier value for electrodes, use the following definition:

```
#define TSS_En_SIGNAL_MULTIPLIER N
```

Where:

- n is the electrode number
- N is the multiplier value in the range 0-255

If an electrode does not have a multiplier defined or it equals 0, the multiplying of this electrode signal is disabled. The final electrode signal value is defined by the equation:

$$\text{En_SIGNAL} = (\text{En_SIGNAL} * \text{TSS_En_SIGNAL_MULTIPLIER}) / \text{TSS_En_SIGNAL_DIVIDER}.$$

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.5 Automatic sensitivity calibration

The TSS library provides the automatic sensitivity calibration function. The function periodically adjusts the level of electrode sensitivity calculated according to the estimated noise level and touch tracking information. For more details on the automatic sensitivity function, refer to [Section 5.7, “Automatic Sensitivity Calibration”](#).

To enable the Automatic Sensitivity Calibration feature of the library, use the following definition:

```
#define TSS_USE_AUTO_SENS_CALIBRATION 1
```

If the Automatic Sensitivity Calibration is enabled the duration of initialization can be defined. To define the duration of initialization, use the following definition:

```
#define TSS_USE_AUTO_SENS_CALIB_INIT_DURATION N
```

Where:

- *N* is the initialization duration value in the range 0-255. The number designates the number of the executed TSS task.

The TSS does not evaluate touch during this initialization time. The longer duration can improve an estimation of the noise level and the touch recognition reliability.

TSS enables to configure the minimum level of sensitivity value controlled by Automatic Sensitivity Calibration. The limit value is defined in runtime by setting an initial value of Electrode Sensitivity register. To enable this feature, use the following definition:

```
#define TSS_USE_AUTO_SENS_CALIB_LOW_LIMIT 1
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.6 Baseline initialization

The TSS library provides the dc-tracker feature which is initialized after TSS is started. The initialization duration parameter value defines how long the electrode signal is averaged until it is used as initial baseline value. The TSS does not evaluate touch during this initialization time. The longer duration can improve the noise immunity and the touch recognition reliability. The function is not available for the noise keydetector. To setup duration of baseline initialization, use the following definition:

```
#define TSS_USE_BASELINE_INIT_DURATION N
```

Where:

- *N* is the initialization duration value in the range 0-255. The number designates the number of the executed TSS task.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.7 OnFault callback

The TSS library enables the OnFault Callback function. This is an alternative to the SWI function of the HCS08 version of the library used in TSS versions 1.x. The OnFault callback function is available for all versions of libraries, HCS08, ColdFire V1, Coldfire+, ARM Cortex-M4, and ARM Cortex-M0+. It is recommended to handle fault events detected by the TSS code. The callback function is executed every time the TSS detects a fault. The fault is also reported in the Fault register. Numeric parameter of the callback function identifies the fault electrode. See more details about the fault register in the TSS reference manual.

To enable this feature of the library and definition of `TSS_fOnFault` callback name, use the following definition:

```
#define TSS_ONFAULT_CALLBACK TSS_fOnFault
```

Where `TSS_fOnFault` is the callback function name defined in the application code

When the macro is not defined, no callback is invoked and this feature is disabled.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.8 OnInit callback

The TSS library provides the OnInit Callback function. It must always be defined in the user application. The function is intended for hardware initialization. The callback is executed during the `TSS_Init` function call, therefore every TSS reinitialization causes also this hardware reinitialization. The function must perform initialization of peripherals used by the TSS, for example enabling peripheral clock, setup pin multiplexers, and so on.

The standard callback name is `TSS_fOnInit`. To rename this callback function name use the following definition:

```
#define TSS_ONINIT_CALLBACK TSS_fOnInit
```

Where `TSS_fOnInit` is the callback function name defined in the application code.

When the macro is not defined, the name `TSS_fOnInit` is used as the name of this callback function.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.9 OnProximity callback

The TSS library provides the OnProximity Callback function. This callback is invoked when a touch is detected on the proximity electrode while the proximity mode is enabled. The function is not available in the Noise keydetector. For more details about the proximity feature, refer to [Section 5.6, “Proximity feature”](#).

To enable the proximity function and to define this callback function name use the following definition:

```
#define TSS_ONPROXIMITY_CALLBACK TSS_fOnProximity
```

Where `TSS_fOnProximity` is the callback function name defined in the application code.

When the macro is not defined, the proximity function is globally disabled in the TSS. If the proximity function is enabled then the proximity-specific options related to low level sensor can be defined. Refer to [Section 4.2.23, “TSI measurement method settings”](#) and [Section 4.2.22, “GPIO measurement method settings”](#).

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.10 Trigger function

The TSS enables the use of the triggering function that allows you to control time when the capacitance measurement is performed. Three Triggering modes are provided—ALWAYS, SW, and AUTO. For more details of triggering function, refer to [Section 5.4, “Triggering”](#).

Enable triggering feature in TSS_SystemSetup.h.

```
#define TSS_USE_TRIGGER_FUNCTION 1
```

When not enabled, only the ALWAYS triggering method is available. The AUTO and SW triggering method are further configured in runtime by writing to the System Trigger register by the TSS_SetSystemConfig function.

To enable the AUTO trigger feature, the following definition must also be used in the TSS_SystemSetup.h.

```
#define TSS_USE_AUTOTRIGGER_SOURCE source
```

Where `source` is the name of the device that controls the auto trigger function. The possible source options are RTC, LPTMR, TSIx, or UNUSED.

If the auto trigger option is not available on the MCU, the TSS_SetSystemConfig function returns TSS_ERROR_CONFSYS_TRIGGER_SOURCE_NA error code.

4.2.11 Low power function

The TSS implements the low power function that enables to wake the MCU from Low Power mode if the defined source device detects a touch event. For more details of the low power function, refer to [Section 5.5, “Low power feature”](#).

The peripheral module that is responsible for LowPower wakeup control and synchronization is defined by the TSS_USE_LOWPOWER_CONTROL_SOURCE defined in the TSS_SystemSetup.h.

```
#define TSS_USE_LOWPOWER_CONTROL_SOURCE source
```

Where `source` is the name of the device that controls the low power function. The possible options are TSIx, or UNUSED.

The low power functionality is further configured in runtime by writing to the Low Power Electrode register, Low Power Electrode Sensitivity register by the TSS_SetSystemConfig function.

If the low power control source is not available on the MCU, the TSS_SetSystemConfig function returns TSS_ERROR_CONFSYS_LOWPOWER_SOURCE_NA error code.

If the low power source is defined then method of wake up threshold calculation can be selected. The threshold is calculated from the stable signal value without a touch plus the threshold defined in Low Power Electrode Sensitivity register. The stable signal value is either measured once during system initialization, or the dc-tracker baseline can be used for this purpose. To enable calculation of wake-up threshold from dc-tracker baseline, use the following definition:

```
#define TSS_USE_LOWPOWER_THRESHOLD_BASELINE 1
```

TSI module in Kinetis ARM Cortex-M4 silicon version 2 and 3 has errata related to an incorrect wake up signal offset. To enable compensation of the wake up threshold, use the following definition:

```
#define TSS_USE_LOWPOWER_CALIBRATION 1
```

For more details about low power and how to use low power calibration runtime, refer to [Section 5.5, “Low power feature”](#).

4.2.12 Data corruption check

When Data corruption check is enabled, the interval data structures are protected with checksum so any illegal modifications of the structures is detected and reported by Data Corruption bit in the Faults register.

This feature is enabled by default. When disabled by defining

```
#define TSS_USE_DATA_CORRUPTION_CHECK 0
```

the feature is not available and the library code size is decreased.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.13 Stuck-key function

When Stuck-key detection feature is enabled, the application is protected from false permanent touches caused by external increase of the electrodes capacitance. The electrode is only considered touched until the recalibration occurs. The function is configured by Stuck-key Enabler bit in the System Configuration register and Stuck-key Timeout register. The function is not available in the Noise keydetector.

This feature is enabled by default. When disabled by defining

```
#define TSS_USE_STUCK_KEY 0
```

the feature is not available and the library code size is decreased.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.14 Negative baseline drop

When enabled, the function adjusts the baseline level if the signal level is too low, below the baseline. The function is relevant only for Basic keydetector and it is disabled in shielding and proximity mode. For more details about the negative baseline drop function, refer to [Section 5.8.1, “Negative baseline drop”](#).

This feature is enabled by default. When disabled by defining

```
#define TSS_USE_NEGATIVE_BASELINE_DROP 0
```

the feature is not available and the library code size is decreased.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.15 Automatic hardware recalibration

If an electrode fault is detected or electrode enablers are changed, hardware recalibration is automatically executed for the system to adapt to this situation. The electrode fault reason can be: capacitive signal overflow, electrode short to ground, or peripheral module recalibration request.

This feature is enabled by default. When disabled by defining

```
#define TSS_USE_AUTO_HW_RECALIBRATION 0
```

the automatic hardware recalibration is not executed unless the hardware recalibration bit in the System Config register is set.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.16 Noise mode

TSS supports automatic measurement of noise provided by the latest version of the TSI module. The noise mode is measured always once during the response time period to verify noise presence. Once the noise value reaches the threshold, TSS switches from the capacitive measurement to the permanent noise measurement. The noise mode is active as long as the maximum noise value is greater or at least equal to the defined threshold. Once it falls below the defined threshold, TSS switches back to the capacitive mode. If noise was previously measured on any electrode, all electrodes are reset when switching back to the capacitive mode.

To enable noise keydetector based on TSI noise mode, use the following definition:

```
#define TSS_USE_NOISE_MODE 1
```

To see an more configuration options of noise mode, refer to [Section 4.2.23.5, “TSI noise mode”](#).

A status of the active mode can be read in the fault register by NoiseMode flag. The noise key detector is calibrated only during TSS initialization phase where noise presence threshold is obtained. Refer to [Section 5.3.3, “Noise key detector”](#) for more details.

4.2.17 FreeMASTER GUI support

The TSS library can use the FreeMASTER Graphical User Interface (GUI) tool for visualization and configuration of internal library variables. This tool also enables to observe signal behavior, tune sensitivities, setup the TSS system, control registers, and so on.

To enable support of the FreeMASTER GUI, use the following definition:

```
#define TSS_USE_FREEMASTER_GUI 1
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.18 Control private data

The TSS library enables the application code to assign user data to control objects. The assignment is represented by the data pointer which can be retrieved anytime by providing an index of the control.

To enable Control Private Data, use the following definition:

```
#define TSS_USE_CONTROL_PRIVATE_DATA 1
```

To assign user data to the control, use the following function:

```
void TSS_SetControlPrivateData(uint8_t u8CntrlNum, void *pData)
```

To retrieve pointer to the data, use the following function:

```
void* TSS_GetControlPrivateData(uint8_t u8CntrlNum)
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.19 Diagnostic messages

The TSS code tries to define as many compiler and infrastructure declarations as possible to achieve the desired function. For example interrupt vector assignments are done automatically in compiler tools which enable it. When a compiler tool does not enable the library source code to make all necessary declarations, it may be tricky to understand what needs to be done manually in the application code.

By defining the

```
#define TSS_ENABLE_DIAGNOSTIC_MESSAGES 1
```

the library issues the warning messages with instructions about all declarations that need to be made in the application.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.20 Electrodes configuration

To specify the number of electrodes in the TSS_SystemSetup.h file, use the following definition:

```
#define TSS_N_ELECTRODES N
```

Where:

- N is the number of electrodes in the range 1-64

The TSS library needs to specify the measurement method for each electrode. This setting is applied for all low-level routines.

To configure the electrode measurement method type, use the following macros:

```
#define TSS_En_TYPE X
```

Where:

- n is the electrode number. It can have a value from 0 to TSS_N_ELECTRODES-1.
- X is the identifier of the measurement method and it can hold the following values:

- GPIO — For the GPIO based measurement method, default if type is not specified
- TSInCHm — For the TSI module based measurement method, where n is TSI module number and m is channel number
- To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

If GPIO measurement method is defined for the electrode, GPIO pin must be specified for the electrode. To configure the electrode GPIO pin, use the following macros:

```
#define TSS_EnP X
#define TSS_EnB Y
```

Where:

- n is the electrode number. It can have a value from 0 to TSS_N_ELECTRODES-1.
- X is the port letter of the electrode pin. For instance, A, B, C, D, ...
- Y is the port pin number of the electrode

These macros are relevant only to the GPIO measurement method and need to be defined for each electrode pin.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.21 Controls configuration

To specify the number of controls in the application, use the following definition:

```
#define TSS_N_CONTROLS N
```

Where:

- N is the number of controls

Each control is an instance of a decoder object. The decoder objects supported are keypad, rotary, slider, analog rotary, analog slider, and matrix.

To specify the control type, use the following definition:

```
#define TSS_Cn_TYPE ControlType
```

Where:

- n is the control number in the range 0 to TSS_N_CONTROLS-1
- *ControlType* is the specification of control type

One definition is required for each control to specify its type. The *ControlType* is one of: TSS_CT_KEYPAD, TSS_CT_SLIDER, TSS_CT_ROTARY, TSS_CT_ASLIDER, TSS_CT_AROTARY, or TSS_CT_MATRIX. One macro is required for each control to specify its type, starting with n equal to 0 to the TSS_N_CONTROLS-1.

Each control must have its configuration and status (C&S) structure. This definition sets the name of the control C&S structure. You can define any name for each control structure. However, you cannot have the

same name for different controls. `TSS_SystemSetupData.c` creates each control structure using the name specified.

```
#define TSS_Cn_STRUCTURE Structname
```

Where:

- *n* is the number of the control in the range 0 to `TSS_N_CONTROLS-1`. All defined controls must have a configuration macro of its control C&S structure name.
- *Structname* is the specific structure name.

Each control has its own callback function. It is called by the TSS library in case of an event occurrence in the control. To configure the callback function name for each control, use the following definition:

```
#define TSS_Cn_CALLBACK CallBackFunc
```

Where:

- *n* is the control number in the range 0 to `TSS_N_CONTROLS-1`
- *CallBackFunc* is the callback function

To assign electrodes to a given control, use the following definition:

```
#define TSS_Cn_INPUTS {first el.,second el.,....}
```

Where:

- *n* is the control number in the range 0 to `TSS_N_CONTROLS-1`
- *each element in the array* is one electrode index

Matrix control needs to setup specific parameters, to see the description, refer to [Section 4.2.21.1, “Matrix-specific parameters”](#)

Keypad control provides to configure electrodes groups, to see the description, refer to [Section 4.2.21.2, “Keypad electrode groups”](#)

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.21.1 Matrix-specific parameters

To define the number of electrodes assigned to the matrix control on X axis, use the following definitions:

```
#define TSS_Cn_INPUTS_NUM_X x
```

Where:

- *n* is the control number in the range 0 to `TSS_N_CONTROLS-1`
- *x* is the number of electrodes in the range 2 to 14 for the X horizontal axis of control *n*

Each matrix control must have a definition specifying number of electrodes on X axis. This number specifies what amount of electrodes is assigned to the X axis from the beginning of defined control inputs array. The rest of the electrodes is assigned to the Y axis.

4.2.21.2 Keypad electrode groups

Keypad control provides an option to define electrode groups. One key can be formed with more than one electrode. To configure the groups, use the following definition:

```
#define TSS_Cn_KEYS {first,second,...}
```

Where:

- *n* is the control number in the range 0 to TSS_N_CONTROLS-1
- *each element in the array* defines a bitmask of one key

Each element of the array forms one key in the form of 16-bit value. One key consists of the maximum 16 electrodes as assigned to the control with TSS_Cn_INPUTS macro. Each bit represents one electrode. For instance number 0x55 (0b01010101) represents key formed by first, third, fifth, and seventh electrode assigned to the keypad control. If not defined, the default array of {0x01, 0x02, 0x04, 0x08, 0x10, ... } is used which makes each electrode one key (no groups are used).

4.2.22 GPIO measurement method settings

The following subchapters describe parameters relevant only to the GPIO measurement method.

4.2.22.1 GPIO strength

The TSS library can enable the strength feature on the GPIO pins. If this option is enabled, the TSS library enables the strength setting on each pin that provides this function. If it is not possible to use this feature for a pin, a warning message is returned.

To enable the strength feature, use the following definition:

```
#define TSS_USE_GPIO_STRENGTH 1
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.22.2 GPIO slew rate

The TSS library can enable the slew rate feature on the GPIO pins. If this option is enabled, then the TSS library enables the slew rate setting on each pin that provides this function. If it is not possible to use this feature for a pin, a warning message is returned.

To enable the slew rate feature, use the following definition:

```
#define TSS_USE_GPIO_SLEW_RATE 1
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.22.3 Default electrode level

The TSS library can specify the default electrode pin voltage level in the idle state between each measurement. Normally it is recommended to keep the idle state High. If a low state is needed, use the following definition:

```
#define TSS_USE_DEFAULT_ELECTRODE_LEVEL_LOW 1
```

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.22.4 Noise amplitude filter

The noise amplitude filter processes each sample measured by TSS low level sensor method. You can define the noise amplitude to be filtered. Noise peaks greater than the defined amplitude are filtered by the system, thus disregarding the noisy samples. A new sample is taken to replace the rejected one. The function helps to eliminate high-frequency noise modulated on the input signal and other external interference. To enable this feature of the library, use the following definition:

```
#define TSS_USE_NOISE_AMPLITUDE_FILTER 1
```

If the function is enabled then the Noise Amplitude Filter Size of each electrode must be defined. To define the noise amplitude filter size of electrodes, use the following definition:

```
#define TSS_En_NOISE_AMPLITUDE_FILTER_SIZE N
```

Where:

- n is the electrode number
- N is the size of the noise amplitude filter for the electrode in the range of 2–255

If the noise amplitude filter size is not defined for an electrode, the default value used is 255, and disables the amplitude filter algorithm for the electrode.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.22.5 Hardware timer configuration

In case the application uses the GPIO measurement method, the TSS library requires the application to use one of the following timer modules: TPM, FTM, or MTIM timer module. The timer selected for use by the TSS library is internally called the hardware timer and needs to be defined in the TSS_SystemSetup.h. It is used for all electrodes of either type.

If the hardware timer is needed, the following configuration must be performed by the user application:

- Configure the hardware timer name.
- Include the `TSS_SET_SAMPLE_INTERRUPTED()` macro in all the interrupt service routines anywhere in the user application.
- Configure the timer prescaler and timer timeout, if needed. Configure the proximity timer prescaler and proximity timer timeout, if the proximity function is enabled.
- The Coldfire+, ARM Cortex-M4, and ARM Cortex-M0+ version of the TSS library need manual allocation of `void TSS_HWTimerIsr(void)` interrupt service routine.

To configure the timer name, use the following macro in the TSS_SystemSetup.h

```
#define TSS_HW_TIMER timername
```

Where `timername` is the name of the TPMx, FTMx, or MTIMx timer to be configured by the application

The timer prescaler may require to be adjusted to the application by changing the counter speed that is used to measure the capacitance. The timer frequency depends on the MCU bus frequency. The timer configuration uses a prescaler value to adjust the time frequency relative to the MCU bus frequency. This adjustment is made using the following define present in the TSS_SystemSetup.h file:

```
#define TSS_SENSOR_PRESCALER X
```

Where:

- X is the value used to implement the 2^X timer prescaler. The value of 0 means no prescaler is used.

When the proximity function is enabled, the TSS configuration can be adjusted to proximity configuration which is distinct to a normal TSS mode. The following macro is available for proximity feature configuration:

```
#define TSS_SENSOR_PROX_PRESCALER X
```

Where:

- X is the value used to implement the 2^X timer prescaler for proximity mode. The value of 0 means no prescaler is used.

Specifying the timer prescaler is optional. If not specified, the default value 2 (prescaler = 4) is used.

The timer overflow timeout may require to be adjusted for the application using a non-standard electrode size with specific values of capacitance. The touch sense timer interrupt provides an error handling if an electrode is never charged and the code to exit the electrode charge loop in the event of a timeout. This ensures that the capacitive sensing module does not block the application execution. The adjustment of the timer overflow timeout value is made using the following macro present in the TSS_SystemSetup.h file:

```
#define TSS_SENSOR_TIMEOUT X
```

Where:

- X is the desired timeout value

When the proximity function is enabled, the TSS configuration can be adjusted to proximity configuration which is distinct to a normal TSS mode. The following macro is available for proximity feature configuration:

```
#define TSS_SENSOR_PROX_TIMEOUT X
```

Where:

- X is the desired timeout value for proximity mode

Specification of the timer overflow timeout is optional. If the timer overflow timeout is not specified, then the default value of 511 is used. The range for the timeout value on the HCS08 family also depends on the use of the IIR filter feature preventing the cumulated value overflow. If a slower time is needed, the prescaler can be adjusted.

The HCS08 and ColdFire V1 version of the TSS library automatically allocates the timer overflow interrupt vector for all the used TSS timers. The Coldfire+, ARM Cortex-M4, and ARM Cortex-M0+ version of the TSS library need manual allocation of these interrupt service routines. It defines the interrupt handler function as:

```
void TSS_HWTimerIsr(void)
```

for the hardware timer. The hardware timer interrupt handler function is implemented in the TSS_Sensor.c. To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

NOTE

If the hardware timer is used, then any interrupt service routine of the application must call the `TSS_SET_SAMPLE_INTERRUPTED()` macro, which signals the interrupt occurrence to the capacitive sensing module. If that happens during the capacitive sensing method execution, the measured value is considered invalid and is discarded. This limitation does not affect the TSI based methods.

4.2.23 TSI measurement method settings

The following subchapters describe parameters relevant only to the TSI module based measurement method.

4.2.23.1 TSI autocalibration

The TSI module uses the electrode internal capacitance measurement unit that senses the capacitance of a TSI pin and outputs a 16-bit result. This module is based on dual oscillator architecture. One oscillator has its frequency depending on the electrode capacitance and the second one has a stable reference frequency. Both oscillators have configuration parameters to ensure the best application performance.

To provide a simple setup procedure there is an autocalibration algorithm calculating the best configuration of an External Oscillator Charge Current (EXTCHRG register value) and configuration of an Electrode Oscillator Prescaler (PS register value). The aim of the autocalibration is reaching the defined bit-resolution of signal values set by the `TSS_TSI_RESOLUTION` configuration parameter:

```
#define TSS_TSI_RESOLUTION n
```

Where:

- *n* is the requested resolution in bits, the default is 11.

Searching the best TSI register configuration by autocalibration algorithm can be controlled by limiting the considered values. These limits can be configured by the following macros in the `TSS_SystemSetup.h` file:

```
#define TSS_TSI_EXTCHRG_LOW_LIMIT      X
#define TSS_TSI_EXTCHRG_HIGH_LIMIT    Y
#define TSS_TSI_PS_LOW_LIMIT          Z
#define TSS_TSI_PS_HIGH_LIMIT         W
```

Where:

- X — Low Limit of External Oscillator Charge Current register, default is 0
- Y — High Limit of External Oscillator Charge Current register. Range depends on TSI module version used. Default value is set to maximum register value.
- Z — Low Limit of Electrode Oscillator Prescaler register value, default is 0

- W — High Limit of Electrode Oscillator Prescaler register. Range depends on TSI module version used. Default is set to maximum register value.

When the proximity function is enabled, the TSS configuration can be adjusted to proximity configuration which is distinct to a normal TSS mode. The following macros for configuration of TSI autocalibration in the proximity mode are available:

```
#define TSS_TSI_PROX_RESOLUTION          n
#define TSS_TSI_PROX_EXTCHRG_LOW_LIMIT  X
#define TSS_TSI_PROX_EXTCHRG_HIGH_LIMIT Y
#define TSS_TSI_PROX_PS_LOW_LIMIT       Z
#define TSS_TSI_PROX_PS_HIGH_LIMIT      W
```

Where:

- *n* is the requested resolution in bits for proximity mode, the default is 11.
- X — Low Limit of External Oscillator Charge Current register for proximity mode, default is 0
- Y — High Limit of External Oscillator Charge Current register for proximity mode. Range depends on TSI module version used. Default value is set to maximum register value.
- Z — Low Limit of Electrode Oscillator Prescaler register value for proximity mode, default is 0
- W — High Limit of Electrode Oscillator Prescaler register for proximity mode. Range depends on TSI module version used. Default is set to maximum register value.

NOTE

If High and Low limit value of each range parameter is set to the same value the autocalibration algorithm sets exactly the defined values to the TSI module registers. This is a way how to set required configuration manually to the TSI module and disable TSI autocalibration algorithm. The manual configuration of TSI is recommended in the case of proximity configuration for reaching of precise detection range.

4.2.23.2 TSI active mode clock

In active mode, the TSI module has its full functionality and is able to scan up to 16 electrodes. Some versions of the TSI module can select active mode clock options. Most of the TSI modules implemented in Coldfire+, ARM®Cortex™-M4, and ARM®Cortex™-M0+ provide this option.

The following active clock source configuration options must be defined:

- Select the active mode clock source

```
#define TSS_TSI_AMCLKS clocksource
```

clocksource is the number of the active mode clock source placed into the TSI_SCANC register.

- 0 —> Bus Clock.
- 1 —> MCGIRCLK.
- 2 —> OSCERCLK.
- 3 —> Not valid.

- Set up active mode clock source divider

```
#define TSS_TSI_AMCLKDIV divider
```

`divider` is the number of the active mode clock divider placed into the `TSI_SCANC` register.

- 0 —> Divider set to 1
- 1 —> Divider set to 2048

- Set up active mode prescaler

```
#define TSS_TSI_AMPSC prescaler
```

`prescaler` is the number of the active mode clock prescaler placed into the `TSI_SCANC` register.

- 0 —> Input clock source divided by 1
- 1 —> Input clock source divided by 2
- 2 —> Input clock source divided by 4
- 3 —> Input clock source divided by 8
- 4 —> Input clock source divided by 16
- 5 —> Input clock source divided by 32
- 6 —> Input clock source divided by 64
- 7 —> Input clock source divided by 128

- Set up scan modulo

```
#define TSS_TSI_SMOD smod
```

`smod` is the value of scan modulo placed into the `TSI_SCANC` register. If TSI module is used as a trigger source and AUTO triggering mode is selected then the value defines the period of regular scanning during the AUTO triggering mode.

- 0 -> Continue Scan.
- Others -> Scan Period Modulus.

4.2.23.3 TSI low power mode clock

The TSI module is able to enter low power mode if the MCU enters one of the power saving modes. Some versions of the TSI module can even wake the MCU from the low power mode. In low power mode, only one electrode may be active and able to perform capacitance measurements. Most of the TSI modules implemented in Coldfire+, ARM[®]Cortex[™]-M4, and ARM[®]Cortex[™]-M0+ may provide a selection of low power clock sources. The TSS does not use definitions if they are not applicable.

To use the low power mode feature, the low power source clock must be selected:

```
#define TSS_TSI_LPCLKS clocksource
```

Where `clocksource` is the number of the low power mode clock source placed into the `TSI_GENCS` register.

In the low power mode the electrode scan unit is always configured to a periodical low power scan.

To setup low power mode scan interval, use the following definition:

```
#define TSS_TSI_LPSCNITV lpscnitv
```

Where `lpscnitv` is the number of the low power scan mode interval placed into the `TSI_GENCS` register.

- 0 —> 1 ms scan interval
- 1 —> 5 ms scan interval
- 2 —> 10 ms scan interval
- 3 —> 15 ms scan interval
- 4 —> 20 ms scan interval
- 5 —> 30 ms scan interval
- 6 —> 40 ms scan interval
- 7 —> 50 ms scan interval
- 8 —> 75 ms scan interval
- 9 —> 100 ms scan interval
- 10 —> 125 ms scan interval
- 11 —> 150 ms scan interval
- 12 —> 200 ms scan interval
- 13 —> 300 ms scan interval
- 14 —> 400 ms scan interval
- 15 —> 500 ms scan interval

4.2.23.4 TSI delta voltage

Some versions of the TSI module provide configuration of the charge and discharge difference voltage called Delta Voltage. The value is defined by the macro definition:

```
#define TSS_TSI_DVOLT dvoltvalue
```

Where:

- `dvoltvalue` is the delta voltage value placed into the `TSI_GENCS` register. For more details about the feature refer to the manual of target MCU.

To see an example of the configuration, refer to [Section 4.4, “Example of system setup parameters encoded in the TSS_SystemSetup.h”](#).

4.2.23.5 TSI noise mode

Some versions of the TSI module provide noise mode. The noise mode has different TSI settings in `TSS_SystemSetup.h` file than the capacitive mode.

The following noise mode configuration options may be defined:

- Set up the electrode oscillator prescaler for noise mode

```
#define TSS_TSI_NOISE_PS prescaler
```

`prescaler` is the prescaler value of electrode oscillator placed into the PS register in TSI module.

- 0 —> Electrode oscillator frequency divided by 1
- 1 —> Electrode oscillator frequency divided by 2
- 2 —> Electrode oscillator frequency divided by 4

- 3 —> Electrode oscillator frequency divided by 8
- 4 —> Electrode oscillator frequency divided by 16
- 5 —> Electrode oscillator frequency divided by 32
- 6 —> Electrode oscillator frequency divided by 64
- 7 —> Electrode oscillator frequency divided by 128
- Select number of filter bits for noise mode


```
#define TSS_TSI_NOISE_FILTER filter
```

`filter` is the number of filter bits placed into the EXTCHRG[2:1] register in TSI module.

 - 0 —> 3 filter bits
 - 1 —> 2 filter bits
 - 2 —> 1 filter bit
 - 3 —> no filter bits
- Select series resistance for noise mode


```
#define TSS_TSI_NOISE_RS rs
```

`rs` is the value of series resistance placed into the EXTCHRG[0] register in TSI module.

 - 0 —> $R_s = 32\text{ k}\Omega$
 - 1 —> $R_s = 187\text{ k}\Omega$
- Set up reference oscillator charge current for noise mode


```
#define TSS_TSI_NOISE_REFCHRG refchrg
```

`refchrg` is the value of reference oscillator current placed into the REFCHRG register in TSI module.

 - 0 —> 500 nA
 - 1 —> 1 μA
 - 2 —> 2 μA
 - 3 —> 4 μA
 - 4 —> 8 μA
 - 5 —> 16 μA
 - 6 —> 32 μA
 - 7 —> 64 μA

4.3 Store and Load System configuration

This functionality is intended for restoring TSS after low power wakeup. TSS save and restore API provides:

- Save and restore electrode data
- Save and restore modules data
- Save and restore all system data

A use case how this functionality can be used to restore entire TSS configuration:

```

size = TSS_StoreAllSystemData(application_buffer, buffer_size);
/* Check if all data was stored and no error reported */
if ((size <= buffer_size) && (size > 0))
{
    sleep();

    /* After POR caused by wakeup unit, the TSS configuration can be restored */
    TSS_LoadAllSystemData(application_buffer);
}

```

For further information, see chapter TSS Library Configuration Save and Restore Functions in the TSSAPIRM.

4.4 Example of system setup parameters encoded in the TSS_SystemSetup.h

```

/* TSS Features Configuration */
#define TSS_USE_KEYDETECTOR_VERSION          1
#define TSS_USE_NOISE_MODE                   0
#define TSS_USE_SIMPLE_LOW_LEVEL             0
#define TSS_USE_DELTA_LOG                    1
#define TSS_USE_SIGNAL_LOG                   1
#define TSS_USE_INTEGRATION_DELTA_LOG        0
#define TSS_USE_NOISE_SIGNAL_LOG             0
#define TSS_USE_FREEMASTER_GUI               1
#define TSS_USE_CONTROL_PRIVATE_DATA         0
#define TSS_USE_AUTO_SENS_CALIBRATION        1
#define TSS_USE_AUTO_SENS_CALIB_INIT_DURATION 100
#define TSS_USE_AUTO_SENS_CALIB_LOW_LIMIT    0
#define TSS_USE_BASELINE_INIT_DURATION      15
#define TSS_USE_LOWPOWER_THRESHOLD_BASELINE  0
#define TSS_USE_LOWPOWER_CALIBRATION         0

/* GPIO Method Options */
#define TSS_USE_GPIO_STRENGTH                 0
#define TSS_USE_GPIO_SLEW_RATE               0
#define TSS_USE_NOISE_AMPLITUDE_FILTER        1
#define TSS_USE_DEFAULT_ELECTRODE_LEVEL_LOW  0

/* Signal Control Options */
#define TSS_USE_IIR_FILTER                    0
#define TSS_USE_SIGNAL_SHIELDING              1
#define TSS_USE_SIGNAL_DIVIDER                1
#define TSS_USE_SIGNAL_MULTIPLIER             1

/* Function Source Definition */
#define TSS_USE_AUTOTRIGGER_SOURCE             TSIO
#define TSS_USE_LOW_POWER_CONTROL_SOURCE      TSIO

/* Code Size Reduction Options */
#define TSS_USE_DATA_CORRUPTION_CHECK         1
#define TSS_USE_TRIGGER_FUNCTION              1
#define TSS_USE_STUCK_KEY                     1
#define TSS_USE_NEGATIVE_BASELINE_DROP        1
#define TSS_USE_AUTO_HW_RECALIBRATION         1

/* Callback Definition */
#define TSS_ONFAULT_CALLBACK                  TSS_fOnFault

```

```

#define TSS_ONINIT_CALLBACK                TSS_fOnInit
#define TSS_ONPROXIMITY_CALLBACK           TSS_fOnProximity

/* Debug Options */
#define TSS_ENABLE_DIAGNOSTIC_MESSAGES    1

/* Electrode Configuration */
#define TSS_N_ELECTRODES                  14    /* Number of electrodes in the system */

/* Electrode's GPIOs configuration */
#define TSS_E0_P                          A      /* Electrode port */
#define TSS_E0_B                          0      /* Electrode bit */
#define TSS_E1_P                          A      /* Electrode port */
#define TSS_E1_B                          2      /* Electrode bit */
#define TSS_E2_P                          B      /* Electrode port */
#define TSS_E2_B                          0      /* Electrode bit */
#define TSS_E3_P                          B      /* Electrode port */
#define TSS_E3_B                          1      /* Electrode bit */
#define TSS_E4_P                          A      /* Electrode port */
#define TSS_E4_B                          1      /* Electrode bit */
#define TSS_E5_P                          A      /* Electrode port */
#define TSS_E5_B                          3      /* Electrode bit */
#define TSS_E6_P                          B      /* Electrode port */
#define TSS_E6_B                          4      /* Electrode bit */
#define TSS_E7_P                          B      /* Electrode port */
#define TSS_E7_B                          3      /* Electrode bit */

/* Electrode measurement method specification */
#define TSS_E8_TYPE                       TSI0_CH5 /* Measurement Method for E8 */
#define TSS_E9_TYPE                       TSI0_CH0 /* Measurement Method for E9 */
#define TSS_E10_TYPE                      TSI0_CH1 /* Measurement Method for E10 */
#define TSS_E11_TYPE                      TSI0_CH2 /* Measurement Method for E11 */
#define TSS_E12_TYPE                      TSI0_CH8 /* Measurement Method for E12 */
#define TSS_E13_TYPE                      TSI0_CH9 /* Measurement Method for E13 */

/* Electrode's noise amplitude filter size configuration */
#define TSS_E0_NOISE_AMPLITUDE_FILTER_SIZE 100 /* Amplitude Filter size for E0 */
#define TSS_E1_NOISE_AMPLITUDE_FILTER_SIZE 150 /* Amplitude Filter size for E1 */
#define TSS_E2_NOISE_AMPLITUDE_FILTER_SIZE 50  /* Amplitude Filter size for E2 */

/* Shield Configuration */
#define TSS_E0_SHIELD_ELECTRODE           11    /* Assign shield electrode 11 to E0 */
#define TSS_E1_SHIELD_ELECTRODE           11    /* Assign shield electrode 11 to E1 */
#define TSS_E5_SHIELD_ELECTRODE           11    /* Assign shield electrode 11 to E5 */

/* Signal Divider Configuration */
#define TSS_E0_SIGNAL_DIVIDER              2      /* Assign signal divider to E0 */
#define TSS_E3_SIGNAL_DIVIDER              3      /* Assign signal divider to E3 */

/* Signal Multiplier Configuration */
#define TSS_E0_SIGNAL_MULTIPLIER           3      /* Assign signal multiplier to E0 */
#define TSS_E9_SIGNAL_MULTIPLIER           2      /* Assign signal multiplier to E9 */

/* Controls configuration */
#define TSS_N_CONTROLS                     4
#define TSS_C0_TYPE                       TSS_CT_SLIDER /* Control type */
#define TSS_C0_INPUTS                     {1,0,2} /* Electrodes assigned to the control */

```

```

#define TSS_C0_STRUCTURE      cVolSlider    /* Name of the C&S struct to create */
#define TSS_C0_CALLBACK      VolCbK        /* Identifier of the user's callback */
#define TSS_C1_TYPE          TSS_CT_KEYPAD  /*Control type */
#define TSS_C1_INPUTS        {3,4}         /* Electrodes assigned tothe control */
#define TSS_C1_KEYS          {0x3,0x1,0x2}  /* Keypad groups definition*/
#define TSS_C1_STRUCTURE     cKey1         /* Name of the C&S struct to create */
#define TSS_C1_CALLBACK      KeyCbK        /* Identifier of the user's callback */
#define TSS_C2_TYPE          TSS_CT_AROTARY /* Control type */
#define TSS_C2_ELECTRODES    {5,6,7}       /* Electrodes assigned tothe control */
#define TSS_C2_STRUCTURE     cARotary      /* Name of the C&S struct to create */
#define TSS_C2_CALLBACK      ARotCbK      /* Identifier of the user's callback */
#define TSS_C3_TYPE          TSS_CT_MATRIX  /* Control type */
#define TSS_C3_INPUTS3        {8,9,10,11,12,13} /* Electrodes assigned tothe control */
#define TSS_C3_INPUTS_NUM_X  3             /* Number of X electrodes in the control */
#define TSS_C3_STRUCTURE     cMatrix       /* Name of the C&S struct to create */
#define TSS_C3_CALLBACK      MatCbK       /* Identifier of the user's callback */

/* Timer Specific parameters */
#define TSS_HW_TIMER          FTM1          /* Hardware Timer name */
#define TSS_SENSOR_PRESCALER  2            /* Prescaler for HW Timer */
#define TSS_SENSOR_TIMEOUT    511         /* Timeout for HW Timer */
#define TSS_SENSOR_PROX_PRESCALER 1        /* Prescaler in proximity mode */
#define TSS_SENSOR_PROX_TIMEOUT 1024      /* Timeout in proximity mode */

/* TSI Specific parameters */

/* Required TSI Configuration */
#define TSS_TSI_RESOLUTION    10          /* TSI Resolution in bits */
#define TSS_TSI_EXTCHRG_LOW_LIMIT 1
#define TSS_TSI_EXTCHRG_HIGH_LIMIT 31
#define TSS_TSI_PS_LOW_LIMIT  0
#define TSS_TSI_PS_HIGH_LIMIT 7

/* Required TSI configuration in Proximity mode */
#define TSS_TSI_PROX_RESOLUTION 12        /* TSI Resolution in bits */
#define TSS_TSI_PROX_EXTCHRG_LOW_LIMIT 10
#define TSS_TSI_PROX_EXTCHRG_HIGH_LIMIT 10
#define TSS_TSI_PROX_PS_LOW_LIMIT 7
#define TSS_TSI_PROX_PS_HIGH_LIMIT 7

/* Active Mode Clock Settings */
#define TSS_TSI_SCANC_AMCLKS   2 /*Set Input Active Mode Clock Source */
#define TSS_TSI_SCANC_AMCLKDIV 1 /*Set Input Active Mode Clock Divider */
#define TSS_TSI_SCANC_AMPSC    7 /*Set Input Active Mode Clock Prescaler */
#define TSS_TSI_SMOD           0 /* Set Scan Modulo */

/* Low Power TSI definitions */
#define TSS_TSI_GENCS_LPCLKS   1 /* Set Low Power Mode Clock Source */
#define TSS_TSI_LPSCNITV      8 /* Set Low Power Mode Scan Interval */

/* Delta Voltage configuration */
#define TSS_TSI_DVOLT          7 /* Set TSI Delta Voltage */

```

4.5 TSS version information

The TSS_API.h file provides information about the TSS release version. This may help to manage versions of the TSS product in custom applications.

The information about TSS release version is provided by following macros in TSS_API.h file:

```
#define __TSS__          X
#define __TSS_MINOR__    Y
#define __TSS_PATCHLEVEL__ Z

#define __TSS_VERSION__  (__TSS__ * 10000 + __TSS_MINOR__ * 100 + __TSS_PATCHLEVEL__)
```

Where:

- X — Major version of the TSS release
- Y — Minor version of the TSS release
- Z — Patch version of the TSS release

Macro `__TSS_VERSION__` can be used directly for explicit detection of the latest software version.

Chapter 5 Touch Sensing Algorithms

5.1 TSI Module Based Touch Sensing Method

The Touch Sensing Input (TSI) module provides capacitive touch sensing detection with high sensitivity and enhanced robustness. Each TSI pin implements the capacitive measurement of an electrode having individual result registers. The TSI module can be functional in several low power modes with an ultra low current adder. The Freescale provides two versions of TSI modules at the moment.

The first version of the TSI module is implemented in Coldfire+ and ARM®Cortex®-M4 MCUs. This TSI module can wake the CPU in a touch event, measures all enabled electrodes in one automatic cycle and provides automatic triggering inside the module. For more details on the TSI module features, refer to an arbitrary reference manual of the MCU containing the TSI module inside. [Figure 5-1](#) is the block diagram for the TSI module.

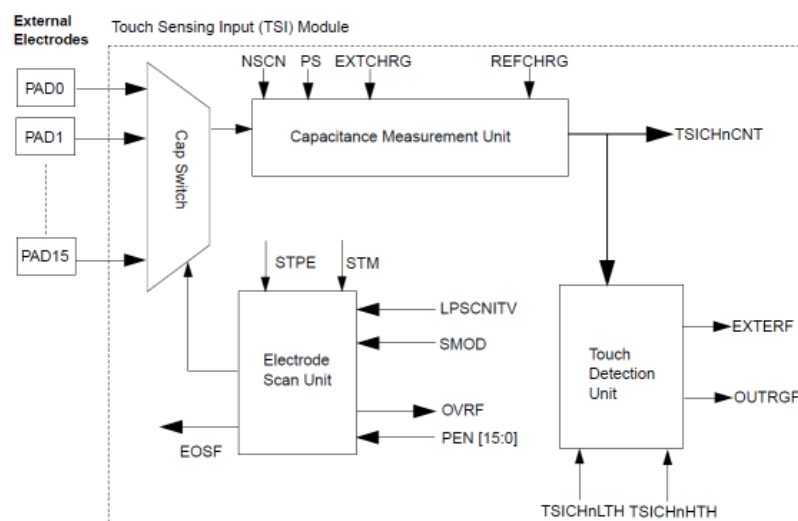


Figure 5-1. TSI module version 1 block diagram

The second version of the TSI module is simplified version of the first generation with some additional features. It is implemented in the S08PTxx and ARM®Cortex®-M0+ MCUs at the moment. This kind of TSI module measures just one enabled electrode in one measurement cycle and provides automatic triggering externally by the RTC, or LPTMR timer. For more information on the TSI module features, refer to a reference manual of the MCU containing the TSI module.

[Figure 5-2](#) shows the block diagram of the TSI module.

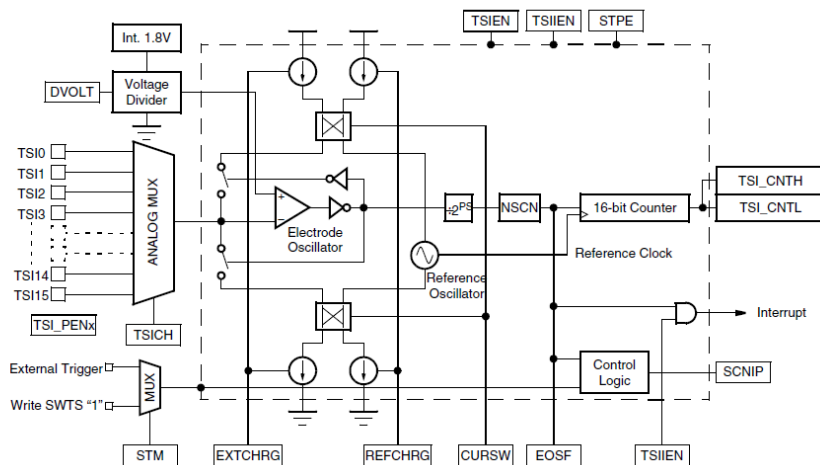


Figure 5-2. TSI module version 2 block diagram

5.1.1 TSIL sample electrode control

Touch Sensing Input Lite (TSIL) is an internal name of the TSI module version 2 as it is implemented in the TSS library. The TSIL algorithm is placed into the separated file TSS_SensorTSIL.c. The TSI module version 2 is implemented in the HCS08 MCU families with a smaller flash memory footprint. For this purpose the TSS has implemented two versions of the TSIL low level routines.

- Standard Low Level routines provide full TSS functionality with background measurement. There are three triggering modes available (Auto triggering only if the TSI measurement method is used at least on one electrode).
- Simple Low Level routines are simpler with smaller size code. The measurement is consecutive, but TSS_Task needs to wait for an end of measurement. The auto triggering function is not available. The simple low level is limited to a single TSI module only.

For more details how to select simple low level routines, refer to [Section 4.2.2, “Simple low level routines.”](#)

5.2 GPIO Based Capacitive Touch Sensing Method

To measure the capacitance of an electrode, a Freescale MCU with GPIO and time measurement capabilities is required. [Figure 5-3](#) shows the electric diagram of the circuit.

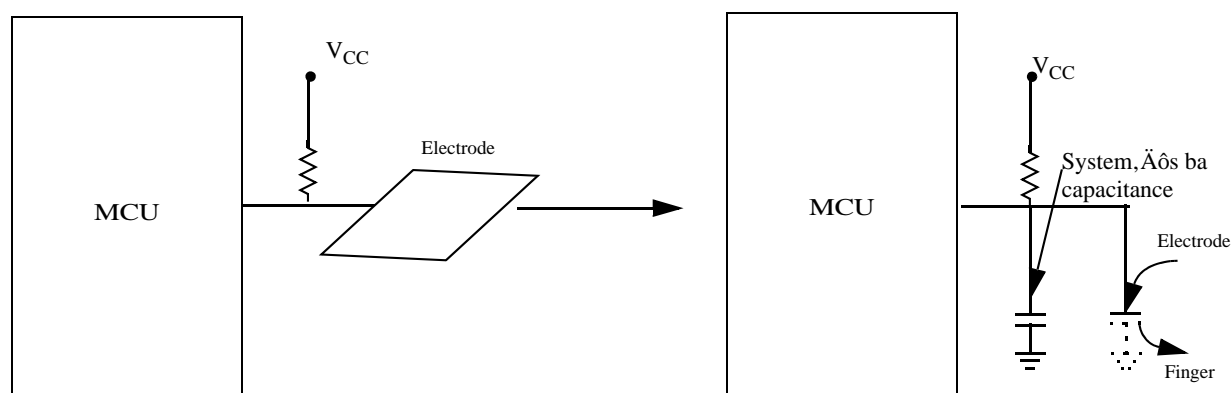


Figure 5-3. Electrode equivalent circuit

The electrode connected to the MCU acts like a capacitor, and the external pull-up resistor limits the current to charge the electrode.

The RC circuit charging time constant (τ) is defined by the following equation:

$$\tau = RC \quad \text{Eqn. 5-1}$$

According to the equation, if the pull-up resistor remains the same, an increase in the capacitance will increase the circuit charging time. The MCU measures the charging time and uses this value to determine if the electrode has been touched or not.

By default, the electrode is in output high state. When the measurement starts, the MCU sets the electrode pin as output low to discharge the capacitor. Then, it sets the electrode pin as high impedance state, making the capacitor start charging. This depends on the selected measurement method.

For the GPIO method, it is the input state of the GPIO pin. As the capacitor charges, the MCU enables a counter and counts the time required to reach the pin threshold value, which is $0.7 V_{DD}$. Detection of this state depends on the module function of the selected measurement method. After the threshold is reached, the counter stops, stores the value and discharges the electrode. For more details on the electrode sensing algorithm, refer to [Section 5.2.1, “Electrode capacitive sensing algorithm.”](#)

As the electrode is touched, the finger capacitance is added to the capacitance of the electrode. This increases the circuit capacitance, which increases the charge time measured by the timer as is shown on [Figure 5-4](#). The TSS library uses the charging time difference to determine if the electrode is touched or not.

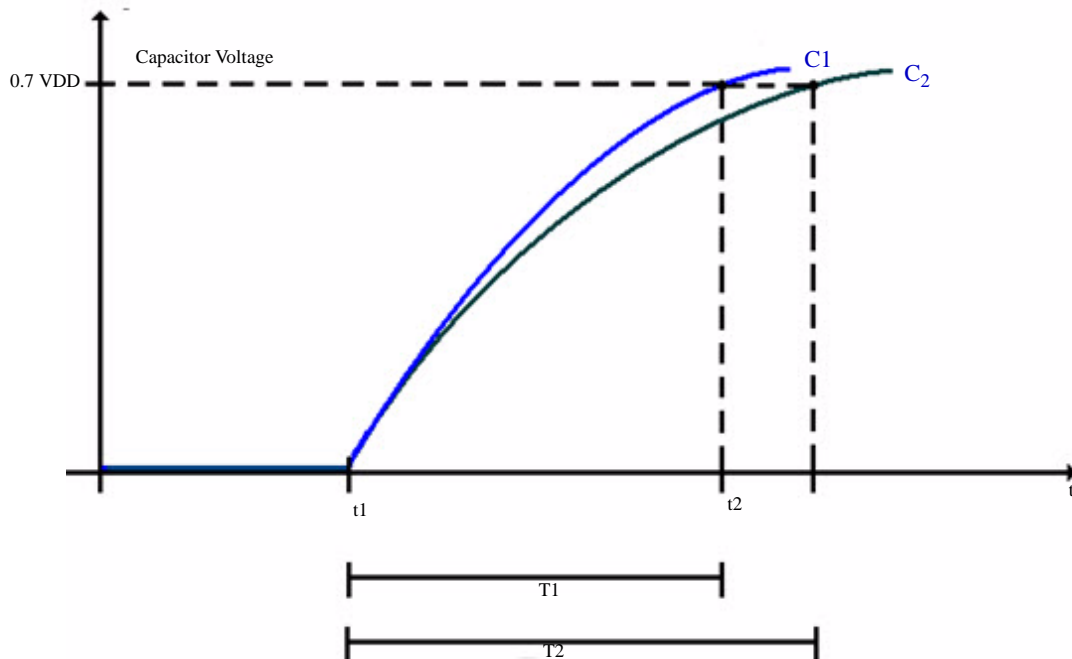


Figure 5-4. Charge Time of Capacitor with Finger Added Capacitance

As shown in [Figure 5-4](#):

- C1 — The charging time curve when there is no extra capacitance or when the electrode has not been touched.
- C2 — The charging time curve when the electrode has been touched.
- T1 — The charging time when the electrode has not been touched.
- T2 — The charging time when the electrode has been touched.

5.2.1 Electrode capacitive sensing algorithm

The Capacitive Sensing algorithm for the GPIO measurement method starts with initialization of the hardware timer and then the function continues in the main steps described below.

1. The Electrode Capacitive Sensing Algorithm sets up the used modules. Then the related low-level measurement routine is called either in the form of optimized external `uint16_t TSS_SampleElectrodeLowEx(void)` low-level routine, or directly programmed code. In the case of external low-level routine its type depends on the measurement method. For details about the GPIO-based external low-level routine, refer to [Section 5.2.1.1, “GPIO based low level routine.”](#)
2. The algorithm reads the timeout flag to determine if it is a valid sample. This step may lead to the following outcomes:
 - If the sample is not valid due to charge timeout or short situation, the error return value is set and the number of remaining samples is set to zero.
 - If the sample is not valid due to interruption of sampling by interrupt from user application, the sample is discarded and the number of remaining samples stays the same. The interruption of

the sampling needs to be indicated by `TSS_SET_SAMPLE_INTERRUPTED()` macro placed to the interrupt service routines in the user application.

- If the sample is valid, the charging time value is accumulated to previous sampled signal and the number of remaining samples is decremented by one.
 - If the noise amplitude filter is enabled then the sample is additionally processed by this algorithm.
3. If the number of remaining samples is zero, the status value is returned and the program ends. Otherwise the algorithm goes back the discharge step repeating the sampling process.

5.2.1.1 GPIO based low level routine

Figure 5-5 shows the GPIO based low-level algorithm. This low-level routine can be realized either in the form of the optimized external low level routine `uint16_t TSS_SampleElectrodeLowEx(void)`, or it can be directly programmed code in the GPIO Electrode Sensing Algorithm, see [Section 5.2.1, “Electrode capacitive sensing algorithm.”](#) It uses the hardware timer defined by the user. These instructions are the same for all electrodes using the GPIO based measurement method.

The main steps of the GPIO based low-level routine are as follows.

1. Start the hardware timer.
2. Start charging the electrode by setting the GPIO pin to input state.
3. Wait for the electrode to be charged. This step may lead to the following results:
 - If the electrode is not charged and the timeout value has not been reached, the algorithm remains in this step.
 - If the electrode is not charged and the timeout value has been reached, an interrupt occurs. The timeout flag is set, and the program comes out of the waiting loop and stops the counter.
 - If the electrode is charged before the timeout value is reached, the program comes out of the waiting step and stops the counter.
4. The function returns the integer value of the hardware timer counter.

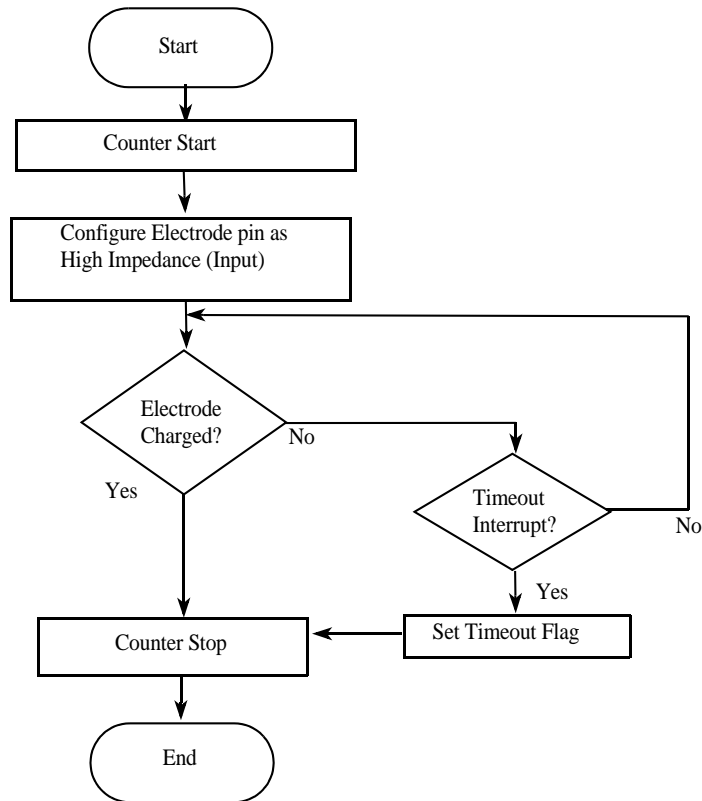


Figure 5-5. GPIO based low level routine

5.2.2 Selecting the proper timer frequency and external pull-up resistor

When using the TSS library and GPIO based measurement method, it is important to consider the following parameters:

- Timer frequency
- Pull-up resistor value
- MCU power voltage

Any variation in these parameters will modify the library performance. A variation may also result in the improvement of one aspect compromising the other. A commitment must be made when selecting the value of these parameters so that the performance of the library meets the application needs. The sections below describe the effect of these parameters on the library performance and provide comparison tables so the user can choose the most suitable value.

5.2.3 MCU frequency

If the GPIO based measurement method is selected, then the library uses at least one timer module from the MCU to count the charging time of the electrodes. The timer module depends on the frequency of the clock used, therefore the frequency at which the MCU is configured is important. Because the frequency

determines the minimum capacitance value detected per count, the most desirable frequency is the maximum allowed value.

The S08 microcontroller series are able to run at 10 MHz when using the internal source clock oscillator, having the timer module run at 2.5 Mhz.

Minimum amount of time to measure is:

$$1/2.5 \text{ MHz} = 400\text{ns}$$

Maximum amount of time is:

$$255 * 400\text{ns} = 102\mu\text{s (because it is an 8-bit timer machine)}$$

Therefore, using the 10 MHz frequency configuration, the minimum detected capacitance value per count is:

Minimal — 0.6645 pF/count

Maximal — 0.1054 pF/count

5.2.4 Voltage trip point (V_{ih})

The equivalent value of capacitance measured in an electrode depends on the value where the MCU detects the input as logic 1 (V_{ih}). This value depends on the power voltage (V_{DD}). Therefore, the voltage used to power the MCU affects the time the MCU detects the electrode as charged. According to the datasheet of the 9S08QG8 microcontroller, the value of V_{ih} is as shown in [Table 5-1](#).

Table 5-1. Relationship between V_{dd} and V_{ih}

Parameter	Symbol	Min	Typical	Max	Unit
Input high voltage ($V_{DD} > 2.3 \text{ V}$)	V_{ih}	$0.7 \times V_{DD}$			V
Input high voltage ($1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$) (all digital inputs)		$0.85 \times V_{DD}$			V

5.2.5 Sensitivity and range

As explained in [Section 5.2, “GPIO Based Capacitive Touch Sensing Method,”](#) the capacitance measurement depends on the charging time of the RC circuit formed by the electrode and the pull-up resistor. Any variation in the capacitance translates into the variation of the charging time of the equivalent RC circuit. In addition, a variation in the resistor translates into a variation in the charging time. With the proper resistor value, you can modify the capacitance range and the sensitivity to a range suitable for application. [Table 5-2](#) shows the maximum capacitance range and capacitance resolution at different pull-up resistance values from 500 k to 2 M Ohms.

Table 5-2. Maximum capacitance range and resolution at different pull-up resistance values

Voltage Level	Pull-up Resistor	Cap Range (max) (pF)	C Resolution (min) (pF)
$V_{DD} > 2.3 \text{ V}$	500k	169.44	0.6645
$V_{DD} > 2.3 \text{ V}$	680k	124.59	0.4886
$V_{DD} > 2.3 \text{ V}$	810k	104.59	0.4102
$V_{DD} > 2.3 \text{ V}$	1M	84.72	0.3322
$V_{DD} > 2.3 \text{ V}$	1.5M	56.48	0.2215
$V_{DD} > 2.3 \text{ V}$	2M	42.36	0.1661
$1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$	500k	107.53	0.4217
$1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$	680k	79.07	0.3101
$1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$	810k	66.38	0.2603
$1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$	1M	53.77	0.2108
$1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$	1.5M	35.84	0.1406
$1.8 \text{ V} < V_{DD} < 2.3 \text{ V}$	2M	26.88	0.1054

The table shows that the maximum capacitance range and the minimum capacitance resolution decrease when the pull-up resistance value is increased. Given that, the most desirable scenario occurs when the capacitance resolution is the smallest possible value and the maximum capacitance range is the biggest possible value. A commitment between these two values must be made to obtain the values that meet your application.

5.3 Key Detect Method

The key detector module determines if an electrode has been touched or released based on the values obtained by the capacitive sensing layer. Along with this detection, the key detector module uses a de-bounce algorithm that prevents the library from false touches.

The key detector also detects, reports, and acts on fault conditions during the scanning process. Two main fault conditions are identified electrode short-circuited to supply voltage or ground. The same conditions can be caused by a small capacitance (equal to a short circuit to supply voltage) or by a big capacitance (equals to a short circuit to ground).

The current version of TSS provides these key detector methods:

- Basic key detector
- AFID key detector
- Noise key detector

Figure 5-6 shows the four states in which the electrodes can be found in the key detector module.

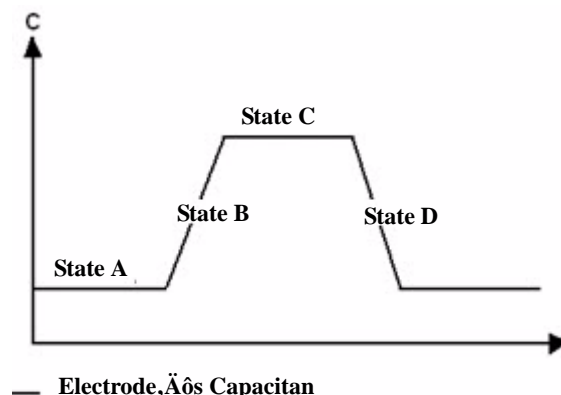


Figure 5-6. Electrode states

State A

In this state, the electrode is not being touched and has not been touched for some time. This state can be referred to as a release state.

State B

In this state, the electrode changes from a release state to the touch state. In this state, the algorithm to prevent false touches is used. If the required conditions are not met, the key detector module does not report the touch event.

State C

In this state, the electrode has met all the conditions required to be considered as touch. Therefore, the key detector module reports the electrode's status as touched to the event buffer or the event register depending on the controller used.

State D

In this state, the electrode is making the transition from touched to release. This state uses algorithms to avoid false releases.

5.3.1 Basic key detector

If the sample is reported as a valid sample, the module calculates delta value from actual signal value and baseline value. The delta value is compared with the threshold value defined in the sensitivity register. Based on the result, it determines the electrode's state, which can be released, touched, changing from released to touch, and changing from touched to release. [Figure 5-9](#) shows example of the basic key detector function.

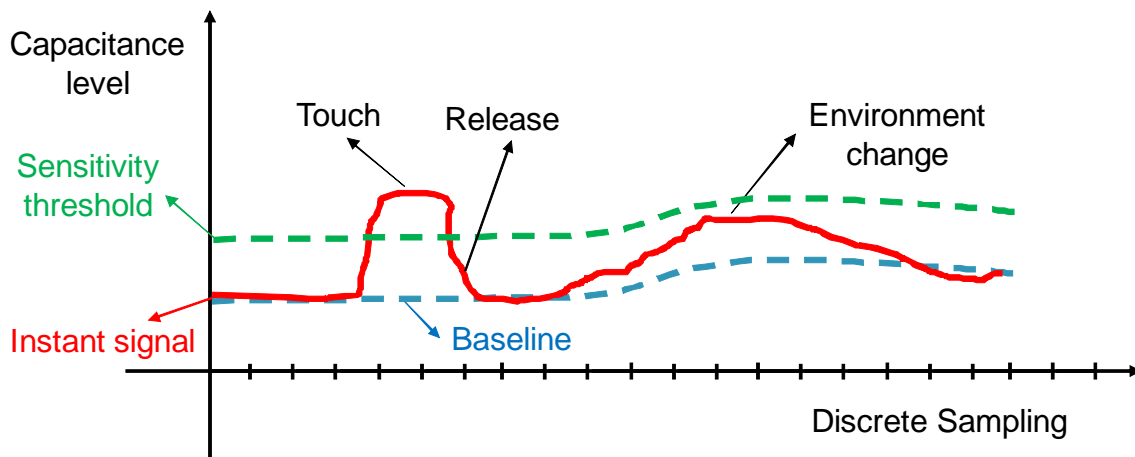


Figure 5-7. Basic key detector signal diagram

5.3.2 AFID key detector

The AFID (Advanced Filtering and Integrating Detection) key detector is based on using two IIR filters with different depths (one short/fast, the other long/slow) and on integrating the difference between the two filtered signals. The algorithm uses two thresholds, touch threshold and release threshold. The touch threshold is defined in sensitivity register. The release threshold has twice lower level than touch threshold. If integrated signal is higher than touch threshold, or lower than release threshold then integrated signal is reset. Touch state is reported for the electrode if the first touch reset is detected. Release state is reported if so many release resets is detected as many touch resets was detected during the previous touch state.

Weight of the signal filters is defined in the form of ratio $f_{\text{sample}}/f_{\text{cutoff}}$. To define filter weights, use the TSS_USE_AFID_FAST_FILTER_RATIO and TSS_USE_AFID_SLOW_FILTER_RATIO macros. For more details about these macro settings, refer to [Section 4.2.1, “Keydetector selection”](#). The filters are set to capacitive measurement period ~1ms by default. For example, it is Kinetis ARM Cortex-M4 MCU with 96 MHz core clock and ALWAYS trigger mode. Weight of the filters should be adjusted in the case of different MCU bus clock for optimal performance.

The integration value is periodically reset in the case of long term release state every 1000 executions of TSS_Task. This helps to adapt to long term signal drifts caused by temperature, RF noise, or other external sources.

This algorithm is more immune against noise, but is not compatible with other noise-cancellation techniques like shielding. User can manually select this key detector in TSS configuration. The standard baseline based on low pass IIR filter is also calculated for analog decoders and proximity function.

[Figure 5-8](#) shows example of the AFID key detector function.

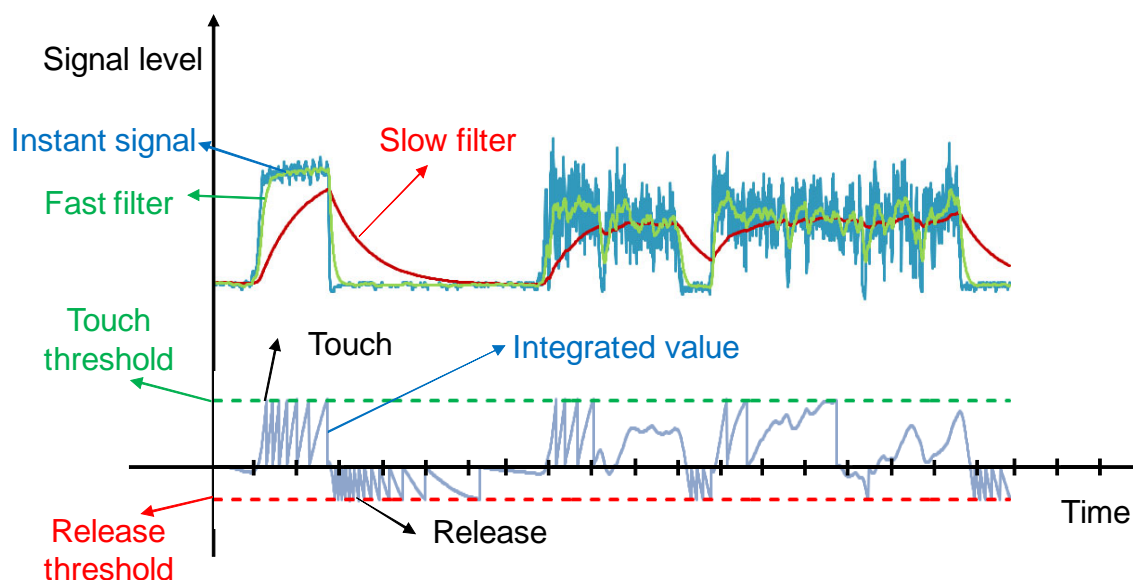


Figure 5-8. AFID key detector signal diagram

5.3.3 Noise key detector

This key detector processes only noise signal from TSI module supporting noise mode. This feature is available only in TSI modules from Kinetis L ARM Cortex-M0+ family. The noise mode feature is an alternative hardware measurement method intended for touch detection in an extremely noisy environment. The noise key detector cannot be selected as standalone key detector, but it works as supplement of Basic or AFID key detector. We strongly advice to use it with AFID key detector whenever possible.

The algorithm provides only touch information and it is not compatible with analog decoders, low power, shielding, or proximity features. A de-bounce function is also implemented in this module to eliminate false detections caused by instantaneous noise.

The noise signal is also processed by IIR filter enabled by TSS_USE_IIR_FILTER macro. The key detector also provides only initial sensitivity calibration function and then sensitivity is not updated.

The noise key detector works in these phases:

1. The key detector measures initial maximum level of noise on all electrodes during the first execution. This value multiplied by 2 is used as noise touch threshold.
2. The standard capacitance measurement is executed during each call of TSS_Task() function except one time during the Response Time period when the single noise measurement is performed.
3. The noise signal information from the measurement is compared with noise touch threshold. If noise level is lower than noise touch threshold then algorithm goes back to point 2. If noise level is higher than noise touch threshold on at least one electrode then algorithm switches to permanent touch detection mode based on noise information.

4. Permanent noise detection mode finds maximum noise level on all electrodes. Multiple touch is reported on all electrodes in the range 2 from maximum noise level.
5. If noise level is lower than noise touch threshold then algorithm goes back to point 2.

5.4 Triggering

The standard measurement processing method runs in an asynchronous way, which means that electrodes are measured one by one in a defined order without time synchronization. The triggering function allows the user to control time when the measurement is performed. Three triggering modes of the capacitance measurement process are provided by the TSS:

- **ALWAYS** — Common measurement process that starts the next measurement sequence of all electrodes immediately after the last electrode finishes measurement. Immediately means as soon as the TSS_Task is called after the previous measurement cycle finishes.
- **SW** — The user application starts the measurement sequence of all electrodes by toggling the SW Trigger bit in the System trigger register. If the TSI method is used at least on one electrode, then precision of measurement start is higher due to background management by the TSI hardware module. The user needs to enable the trigger function in the TSS_SystemSetup.h file.
- **AUTO** — The selected triggering source will control automatic start of the next measurement execution in the defined auto trigger period. The user needs to define auto triggering source and also enable trigger function in the TSS_SystemSetup.h file.

The TSS supports several options on how to configure the triggering function, as shown in the [Table 5-3](#) below.

Table 5-3. Summary of Triggering function configuration

Trigger Function Enabling	Auto Trigger Source	Triggering Mode	Measurement Period	Note
Disabled	UNUSED (or not defined)	ALWAYS	N/A	Writing to Trigger registers is disabled
Enabled	UNUSED (or not defined)	ALWAYS	N/A	—
		SW	Period defined by SW Trigger Bit toggling	If TSI method is used in the application then precision of measurement start is higher.
	RTC, LPTMR, TSIx	ALWAYS	N/A	—
		SW	Period defined by SW Trigger Bit toggling	—
		AUTO	If TSI auto trigger source is used, the TSS_TSI_SMOD macro defines the measurement period. In the case of RTC, or LPTMR auto trigger source, the measurement period is defined by the timer setup.	If the RTC, or LPTMR is used the user needs to manage the timer initialization and interrupt service routine.

A compilation of trigger function code needs to be manually enabled in `TSS_SystemSetup.h`. To enable the AUTO trigger function, the auto triggering source needs to be defined in the `TSS_SystemSetup.h`.

If the trigger function is enabled, writing to the System Trigger registers by the `TSS_SetSystemConfig` function is then possible. The triggering mode can be selected by the Trigger Mode selector in the System Trigger register. The `TSS_TSI_SMOD` macro defines the auto trigger period only for the TSIx auto trigger source and only for some versions of TSI modules which provides SMOD register. Most of the TSI modules implemented in Coldfire+ and ARM®Cortex™-M4 may provide this option.

If the SW or AUTO trigger mode is selected, the `TSS_Task` function should be called as often as possible. This period should be shorter than the used trigger period, otherwise the measured data will be lost and not processed. This situation is reported as a fault by the Small Trigger Period bit in the Fault register. Also the OnFault callback can be enabled to indicate this problematic condition.

Some versions of TSI modules can select active mode clock options. Most of the TSI modules implemented in Coldfire+ and ARM®Cortex™-M4 may provide this option. If this kind of TSI module is used as an auto trigger source and auto triggering mode is selected then the scan period will depend on selection of TSI active mode clock settings. These settings are defined by `TSS_TSI_AMCLKS`, `TSS_TSI_AMCLKDIV`, and `TSS_TSI_AMPSC` macros in `TSS_SystemSetup.h`. For more details about these macro settings, refer to [Section 4.2.23.2, “TSI active mode clock.”](#)

If RTC, or LPTMR timer is selected as auto trigger source for the TSI module then the period of the triggering depends on the timer setup.

5.5 Low power feature

The TSS low power functionality enables to wake the MCU from low power mode if the defined source device detects a touch.

The TSS does not fully manage the MCU low power mode. The TSS just prepares the TSS system and the selected lower power control source device for entering the MCU's low power mode. The user initiates entering into low power mode by himself. If low power control source device detects a touch then it wakes the MCU. The user is then responsible for reconfiguring the TSS to a standard run mode, or again initiates entering the LowPower mode. This function can be combined with the Proximity function, refer to [Section 5.6, “Proximity feature.”](#)

The following steps show an example of typical use of low power function:

1. The peripheral module which is responsible for LowPower wake control and synchronization is defined by the `TSS_USE_LOWPOWER_CONTROL_SOURCE` defined in `TSS_SystemSetup.h`. For more details about this macro setting, refer to [Section 4.2.11, “Low power function.”](#)
2. The user defines `LowPowerElectrode`, and `LowPowerElectrodeSensitivity` registers by the `TSS_SetSystemConfig` function. For more details about these registers, refer to the TSS reference manual.
3. If the user wants to enter to MCU low power mode, the `LowPowerEn` bit has to be enabled in the `SystemConfig` register by the `TSS_SetSystemConfig` function. This action enables the TSS and a selected low power control source device to wake from the low power mode. No execution of the `TSS_Task` is now allowed.

4. The user may now force the MCU to enter low power mode by instructions related to the used MCU platform.
5. If the selected lower power control source device detects a touch, the MCU wakes and the program continues to run. The LowPowerEn bit is automatically disabled.
6. The user can now enter the low power mode as in step 3, or set the TSS to common running mode.

It is important to note that low power threshold is not updated during the low power mode. For eventual recalibration of the TSI threshold is needed manual wake up of MCU by different source than TSI. Then user have to initiate low power enter procedure again. If macro `TSS_USE_LOWPOWER_THRESHOLD_BASELINE` is disabled the `HWRecalStarter` bit must be enabled during re-entering to low power mode.

Some versions of TSI module can wake the MCU from low power mode. Most of TSI modules implemented in Coldfire+, ARM®Cortex™-M4, and ARM®Cortex™-M0+ provide this feature. If this kind of TSI module is used, then the TSI module needs to be defined as source control of the low power mode.

Some TSI modules provide definition of a low power mode clock source and low power scan interval, so the macro `TSS_TSI_LPCLKS` and `TSS_TSI_LPSCNITV` in the `TSS_SystemSetup.h` file needs to be defined. For more details about this macro setting, refer to [Section 4.2.23.3, “TSI low power mode clock”](#). The longer scan period during the low power mode contributes to smaller average power consumption.

Some TSI modules need an external clock for low power functionality. The `TSS_TSI_LPCLKS` is not defined in that case and the user is responsible for an initialization of an external clock source for this purpose, for example the LPTMR timer on ARM®Cortex™-M0+ family.

5.5.1 Low power calibration (Kinetis ARM Cortex-M4 silicon 2, 3)

The macro `TSS_USE_LOWPOWER_CALIBRATION` must be enabled. An application must invoke a function `TSS_LowPowerCalibrationEnable` before entering a sleep mode. This sets TSS low power calibration. The MCU must wake up twice. The first time is a dummy wake up to set the proper wake up counter and the second one is the correct wake up. An example:

```
/* Calibration wake up cycle */
TSS_SetSystemConfig(System_SystemConfig_Register, (TSS_SYSTEM_EN_MASK |
TSS_LOWPOWER_EN_MASK));
TSS_LowPowerCalibrationEnable();
stop();
/* Correct calibrated low power function (after automatic MCU wake-up by TSI) */
TSS_SetSystemConfig(System_SystemConfig_Register, (TSS_SYSTEM_EN_MASK |
TSS_LOWPOWER_EN_MASK));
stop();
/* User's code after wake-up */
```

5.6 Proximity feature

The proximity feature enables a detection of a finger presence even without a direct finger touch to the electrode. The detection distance can be a few centimeters. To enable proximity function you need to have a special designed electrode shape with a bigger electrode area.

The proximity feature is available in two distinct modes.

- proximity run mode can set own proximity TSI or GPIO configuration which can be different from normal run TSI or GPIO configuration. The proximity mode is enabled by ProximityEn bit in system configuration register.
- proximity low power mode is combined with the low power function described in [Section 5.5, “Low power feature”](#). It can wake the MCU. In this case, the LowPowerElectrode, and LowPowerElectrodeSensitivity registers are used for Low Power electrode and also for Proximity electrode settings.

The feature is available in Basic and AFID key detector. The dc-tracker is executed in a rate defined in SlowDcTrackerFactor register in cooperation with DcTrackerRate register.

The following steps show an example of typical use of the proximity function:

1. The OnProximity callback must be defined in TSS_SystemSetup.h file. This enables the proximity feature in TSS. For more details about this callback definition, refer to [Section 4.2.9, “OnProximity callback”](#)
2. If the TSI measurement method is used, the user needs to define proximity configuration of TSI autocalibration. In the case of the GPIO method the user must define proximity configuration of the timer prescaler and timeout. For more details, refer to [Section 4.2.23, “TSI measurement method settings”](#), or [Section 4.2.22, “GPIO measurement method settings”](#).
3. The user defines LowPowerElectrode, and LowPowerElectrodeSensitivity registers by the TSS_SetSystemConfig. For more details about these registers, refer to [Section 4.2.11, “Low power function”](#).
4. If the user wants to enter proximity mode, the ProximityEn bit has to be enabled in the SystemConfig register by the TSS_SetSystemConfig function. For more details about this register, refer to [Section 3.8, “TSS system runtime configuration.”](#) This action switches to TSI and GPIO proximity configuration and initiates hardware recalibration. Then it enables only the proximity electrode and disables the others.
5. If low power mode is also enabled by the LowPowerEn bit then the user must enter the low power mode at this moment. Otherwise, the application needs to execute periodically TSS_Task for proximity detection.
6. If the proximity event is detected, then the OnProximity callback is called.
7. The user can now disable proximity mode by ProximityEn bit, or continue in the next proximity event detection.

5.7 Automatic Sensitivity Calibration

The TSS library provides the automatic sensitivity calibration (ASC) function. The function periodically adjusts the level of electrode sensitivity calculated according to the signal behaviour. The user does not need to set the electrode sensitivity anymore, but manual settings are still available in the case for precise tuning. For more information on how to configure the ASC, refer to [Section 4.2.5, “Automatic sensitivity calibration.”](#)

The three modes of sensitivity management:

1. **The ASC enabled without Sensitivity configuration** — The feature automatically manages sensitivity for each electrode during the overall run of the application. [Figure 5-9](#) shows the ASC operation.
2. **The ASC enabled with Sensitivity configuration** — The first sensitivity configuration is used as an initial value of sensitivity for the ASC feature. Then the feature automatically manages sensitivity for each electrode. This approach can help ASC with better initial estimation of the noise in the system.
3. **The ASC disabled** — The user needs to setup sensitivity manually and this value will be used until the user changes the sensitivity register again.

There are following differences in algorithm between key detector versions:

1. **Basic key detector** - The function periodically adjusts the level of electrode sensitivity calculated according to the estimated noise level and touch tracking information. The algorithm sets sensitivity threshold for common keypad electrodes to 60% between the noise value and maximum delta value. The algorithm sets lower sensitivity to 30% on electrodes used in analog based decoders.
2. **AFID key detector** - The function periodically evaluates number of resets done by integrating delta value and it moves sensitivity threshold for reaching requested number of resets. The algorithm holds sensitivity threshold for common keypad electrodes on 6 resets. The algorithm uses higher sensitivity 12 resets on electrodes used in analog based decoders.
3. **Noise key detector** - The algorithm provides only initial sensitivity calibration function after TSS start and then sensitivity is not updated during the function.

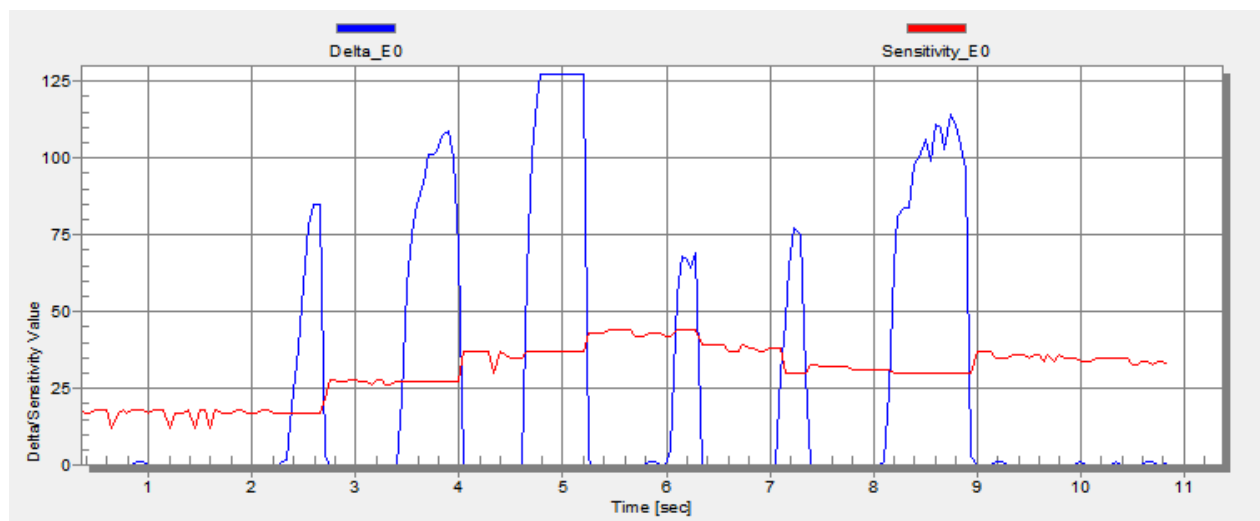


Figure 5-9. Automatic Sensitivity Calibration function

5.8 Baseline tracking

The electrode capacitance constantly varies due to environmental factors. These variations are usually small, but sometimes can lead to spurious detections of the electrodes. Therefore, an algorithm is required to avoid erratic behavior in the electrodes. Temperature changes, noise, humidity, are some factors that can influence the electrode's capacitance. To ensure a robust performance, the library implements the baseline tracking algorithm called dc-tracker. The algorithm determines if the change in the electrode's capacitance was caused by a touch or an environmental factor. The library also allows you to configure the rate at which the baseline is updated. The feature is available in Basic and AFID key detector.

5.8.1 Negative baseline drop

Negative baseline drop is an internal part of the standard baseline tracking algorithm. The function adjusts baseline level if the signal level drops below the baseline. When a signal level is lower than the sensitivity value in a negative direction below the baseline, then the baseline is set to a signal level. [Figure 5-10](#) shows a description of the function. The function helps to adapt the system to not have a predictable drop of the signal caused by noise, or dynamic system recalibration. The refresh rate of negative baseline drop is defined by the baseline-tracking rate. For more information how to configure the baseline-tracking rate, refer to the TSS reference manual - DC tracker rate register. The feature is available only in Basic key detector and it is disabled in shielding and proximity mode. The function can be globally enabled or disabled by the `TSS_USE_NEGATIVE_BASELINE_DROP` macro defined in the `TSS_SystemSetup.h`. For more details about this macro setting, refer to [Section 4.2.14](#), “Negative baseline drop.”

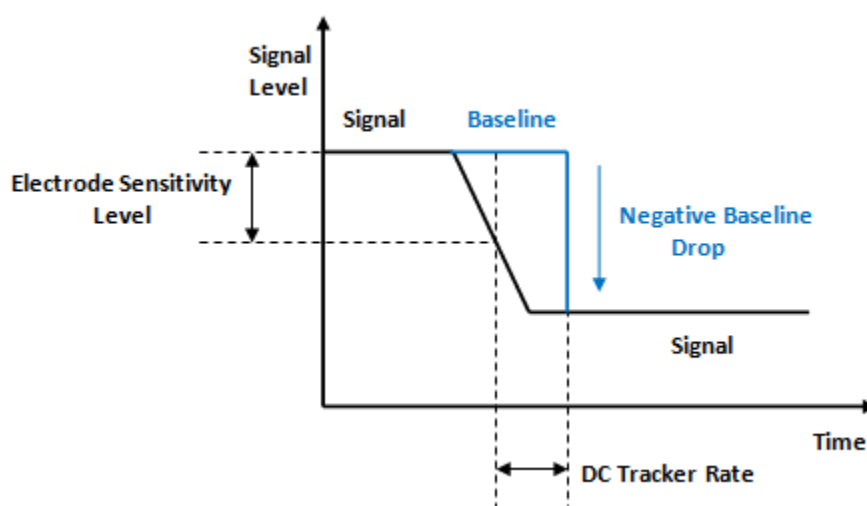


Figure 5-10. Negative baseline drop

5.9 Debouncing

Because the electrodes are implemented as traces or planes of conductive material, they can behave like antennas. Therefore, electric noise is induced in the electrodes. The noise induced, in an extreme scenario, can lead to spurious touches or releases. The TSS library implements a configurable de-bounce algorithm to ensure that the electrodes are properly detected even in noisy environments. The de-bounce algorithm

is implemented in all versions of key detector. The debouncing rate is modified with the ResponseTime register that defines how many samples need to be valid before considering a measured value as valid. The feature is available in all key detector versions. Refer to the TSS reference manual - Number of samples register.

5.10 IIR filter

The electrodes may induct the high frequency noise from the environment. In an extreme situation, the noisy variances on the capacitive signal may lead to spurious touches or releases. The IIR filtration implemented in the TSS library helps to eliminate this high frequency noise modulated on the capacitance signal and other external interferences. The filter processes current capacitance values obtained from low-level routines and works with all low Level Sensor modules. The IIR equation is internally set to ratio 1/3 (current signal and previous signal). The feature is available in all key detector versions. For more information on how to enable the IIR filter, refer to [Section 4.2.4.1, “IIR filter.”](#)

5.11 Noise amplitude filter

In addition to the IIR filter mentioned in the previous chapter, the TSS library also implements the noise amplitude filtering function that may help to eliminate high-frequency noise modulated on the electrode's capacitance signal. The function is available only for the GPIO measurement method. The function processes each sample measured by all low level sensor modules except the TSI module. Each sample value is limited by the amplitude filter with a defined size. If a sample noise is bigger than the defined amplitude, it is ignored. The function helps to eliminate high-frequency noise modulated on the input signal and other external interference. For more information on how to enable the noise amplitude filter function, refer to [Section 4.2.22.4, “Noise amplitude filter.”](#)

5.12 Shielding function and Water tolerance

The shielding function may further improve the TSS noise immunity and water tolerance. The function compensates signal drift on a regular electrode by a special shielding electrode signal. The function can be used in the standard shielding mode described in [Section 5.12.1, “Standard shielding function.”](#) and water tolerance mode described [Section 5.12.2, “Water tolerance mode.”](#) It is important to note that the DC-Tracker function on shielding, or shielded electrode is executed in rate defined in SlowDcTrackerFactor register in cooperation with DcTrackerRate register. Shielding and Water tolerance function is available only in Basic key detector.

For more information on how to enable the shielding function, refer to [Section 4.2.4.2, “Shielding function.”](#)

5.12.1 Standard shielding function

The standard shielding function is intended mainly for suppression of RF noise interference. The shielding electrode is not intended for a touch and measures overall environmental noise or unwanted interference to the system. The user needs to avoid to touch the shield electrode mechanically during the design of touchpad, for example place shielding electrode to the opposite side of the board. It may help to eliminate low frequency noise modulated on the capacitance signal.

5.12.2 Water tolerance mode

The shielding function may also work in Water tolerance mode. The function is enabled by the WaterToleranceEn bit in the System Config register. If Water Tolerance mode is enabled then the shielding electrode compensates the signal drift on regular electrodes, and only to the threshold defined by the Sensitivity register of the shielding electrode. The sensitivity auto calibration is disabled on shielding electrode in the water tolerance mode. If the signal drift is higher than the Sensitivity register value then shield compensation is disabled and the regular electrodes provide a standard touch detection function.

The guarded system enables to eliminate influence of water droplets, but also to detect a touch on regular electrodes if continuous water surface covers all electrodes. Tuning of the shielding Sensitivity register is important for good performance. The user must design the shielding electrode to surround the area of regular electrodes intended to be touched. The shielding electrode must not be directly touched.

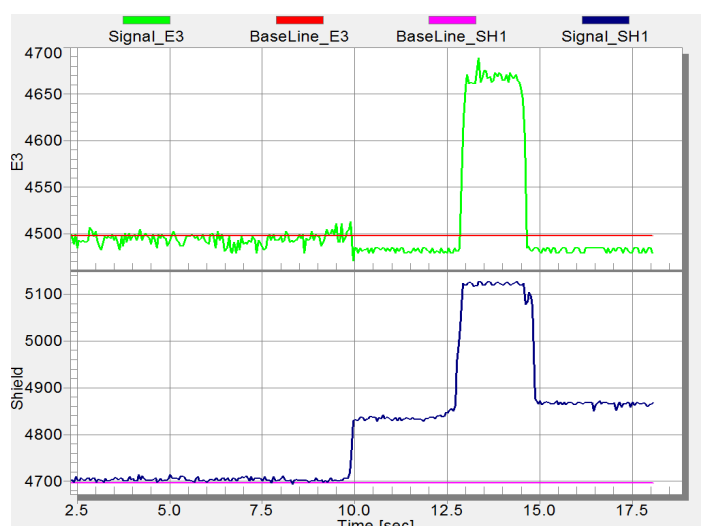


Figure 5-11. Water film on electrodes, E3 electrode's signal unchanged.

5.13 Decoder Functions

Capacitive sensors provide the possibility to replace the traditional on-off buttons and other devices like potentiometers. Along with buttons, potentiometers are one of the commonly used mechanisms that control electronic devices. However, unlike buttons, linear rotational sliders, or matrix require a specific footprint as shown in [Figure 5-12](#).

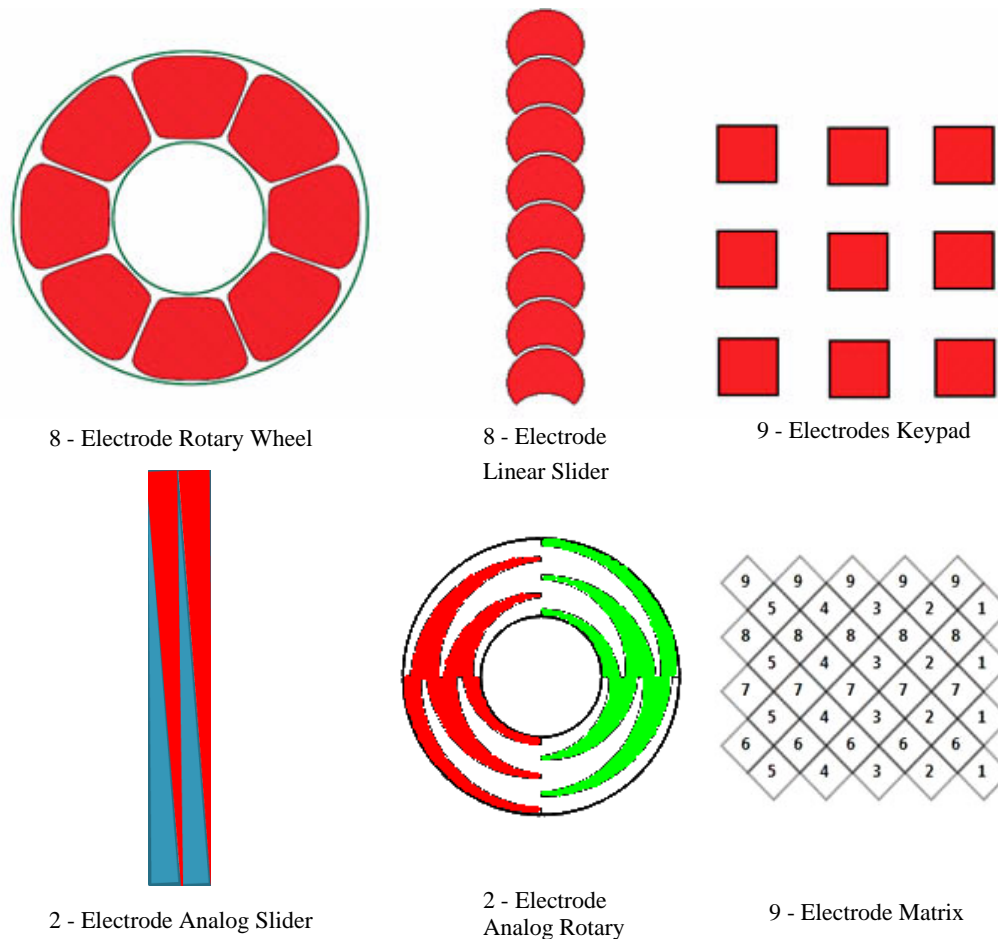


Figure 5-12. Linear slider, rotary sliders, and keypad configuration

Figure 5-12 shows the different ways in which electrodes can be arranged depending on the application needs. After using one of the configurations shown above, the information obtained from sensing the electrode must be interpreted by the decoders. For instance, if a movement has occurred in one of the sliders, the decoder must be able to present the related information from that movement. The interpretation of the sensing information depends on the configuration. This means that the interpretation varies depending on whether it is a rotary slider, linear slider, or a keypad.

5.13.1 Keypad

Keypad is a basic configuration for the arranged electrodes shown in Figure 5-12 since all that matters is to determine which one of the electrodes has been touched. The Keypad Decoder is the module in charge of handling the boundary checking, control of the events buffer, and report of events depending on the user's configuration. The information needed when using a Keypad Decoder is presented in Table 5-4.

The Keypad Decoder must be used when the user's application needs the electrodes to have like keyboard keys. If the user needs to detect movement, another type of decoder must be used.

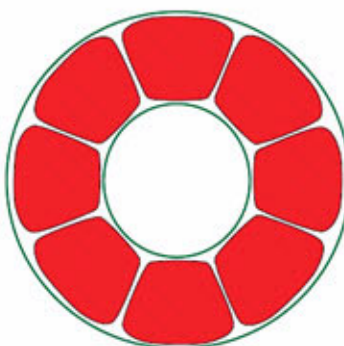
The Keypad decoder is capable of using groups of electrodes that need to be simultaneously touched for reporting the defined key. This allows to create a control interface with more user inputs than the number of physical electrodes.

Table 5-4. Events reported by the keypad decoder

Information	Description
Touch	When using a keypad type decoder, it must provide information regarding when a touch occurred in one of the keypad's electrodes.
Release	The decoder must detect and inform the user when one of the electrodes that were pressed has been released.
Electrode	The keypad decoder must provide information regarding the electrode where the event occurred

5.13.2 Rotary

As mentioned in [Section 5.13, “Decoder Functions”](#) capacitive sensors provide the opportunity to control a device like a potentiometer . To achieve this, a special electrode configuration must be used. [Figure 5-13](#) shows the electrode configuration needed to implement a rotary slider.



8 - Electrode Linear Slider

Figure 5-13. Rotary slider configuration

The shape of the electrodes can be different, but the configuration must be the same. In other words, the electrodes intended to form a rotary slider must be placed one after another forming a circle.

For this type of electrodes, a keyboard decoder cannot be used, even when you still need to detect when a touch has occurred there is more information that needs to be reported. The information needed is listed in [Table 5-5](#) along with a brief explanation.

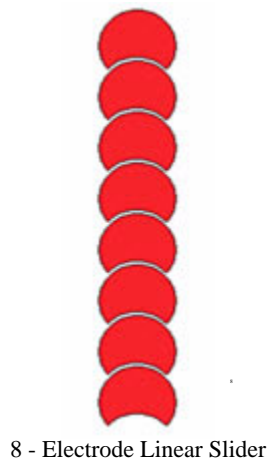
The Rotary Decoder determines the parameters listed in [Table 5-5](#) based on which electrodes have been pressed and which are being pressed.

Table 5-5. Events reported by the rotary decoder

Information	Description
Direction	Probably the most important feature that a slider must have is the ability to report the direction in which a displacement has occurred. This is important because it allows detecting either a decrease or an increase in your application.
Position	The slider must be capable to report which electrode has been or is touched therefore it reports the position.
Displacement	It is important to know the increment in positions when a displacement has occurred. In other words, how many electrodes have been advanced in some displacement.

5.13.3 Slider

A linear slider control works in a similar way as the rotary slider. The same parameters must be reported in both. [Figure 5-14](#) shows an arrangement of electrodes used for a typical linear slider. Like the rotary slider, the shape of the electrodes can be changed but their position must remain as shown in the [Figure 5-14](#).



8 - Electrode Linear Slider

Figure 5-14. Linear slider configuration

The parameters that the Slider Decoder must determine and report are listed in the table below.

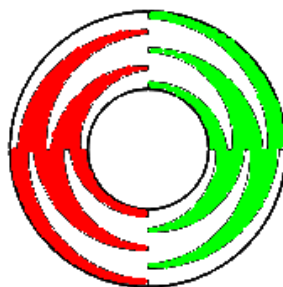
Table 5-6. Events reported by the slider decoder

Information	Description
Direction	Probably the most important feature that a slider must have is the ability to report the direction in which a displacement has occurred. This is important because it allows detecting either a decrement or a increment in your application.
Position	The slider must be capable to report which electrode has been or is touched hence the report of the position.
Displacement	It is important to know the increment in positions when a displacement has occurred, in other words, how many electrodes have been advanced in some displacement.

The slider decoder reports the parameters listed in [Table 5-6](#) by the interpretation of the past and current touch and release events.

5.13.4 Analog rotary

An analog rotary control works similarly as the standard rotary, but with less electrodes and the calculated position has a higher resolution. For example a 4 electrode analog rotary can provide an analog position in the range 64. The shape of the electrodes needs to meet the condition that increases and decreases the signal during the finger movement which needs to be linear. [Figure 5-15](#) shows an arrangement of electrodes used for a typical analog rotary.



4 - Electrode Analog Rotary

Figure 5-15. Analog rotary configuration

The configuration must be the same. In other words, the electrodes intended to form a rotary slider must be placed one after another forming a circle.

The Analog Rotary Decoder determines the parameters listed in [Table 5-7](#) based on which electrodes have been pressed and which are being pressed.

Table 5-7. Events reported by the analog rotary decoder

Information	Description
Direction	Probably the most important feature that analog rotary must have is the ability to report the direction in which a displacement has occurred. This is important because it allows detecting either a decrease or an increase in your application.
Position	The analog rotary must be capable to report the actual analog position
Displacement	It is important to know the increment in positions when a displacement has occurred. In other words, how the position has changed from the last report.

5.13.5 Analog Slider

An analog slider control works similar as the standard slider, but works with less electrodes and the calculated position has a higher resolution. For example a 2 electrode analog slider can provide an analog position in the range 128. The shape of the electrodes need to meet the condition that increases and decreases the signal during the finger movement which needs to be linear. [Figure 5-16](#) shows an arrangement of electrodes used for a typical analog slider.



2 - Electrode Analog Slider

Figure 5-16. Analog slider configuration

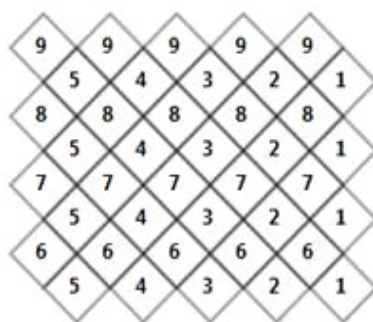
The Analog Slider Decoder determines the parameters listed in [Table 5-8](#). This based on which electrodes have been pressed and which are being pressed.

Table 5-8. Events reported by the analog slider decoder

Information	Description
Direction	Probably the most important feature that a analog slider must have is the ability to report the direction in which a displacement has occurred. This is important because it allows detecting either a decrease or an increase in your application.
Position	The analog slider must be able to report actual analog position.
Displacement	It is important to know the increment in positions when a displacement has occurred. In other words, how the position was changed from last report.

5.13.6 Matrix

An analog matrix works similarly as the analog slider, but the position is calculated in 2 dimensions horizontal X, and vertical Y. Each axis has a defined range of position calculation. The shape of the electrodes need to meet the condition that increases and decreases the signal during the finger movement, in any axis this needs to be linear. The next condition is that the columns and rows needs to be formed from the electrodes. [Figure 5-17](#) shows an arrangement of electrodes used for a typical matrix.



9 - Electrode Matrix

Figure 5-17. Matrix configuration

The Matrix Decoder determines the parameters listed in [Table 5-9](#). This based on which electrodes have been pressed and which are being pressed.

Table 5-9. Events reported by the matrix decoder

Information	Description
Direction X, Direction Y	Probably the most important feature that a matrix must have is the ability to report the direction a displacement has occurred. This is important because it allows detecting either a decrease or an increase in your application. The information is reported for X and Y axis.
PositionX, PositionY	The matrix must be capable to report actual analog position. The information is reported for X and Y axis.
DisplacementX, DisplacementY	It is important to know the increment in positions when a displacement has occurred. That is, how the position was changed from last report. The information is reported for X and Y axis.
Gesture	The matrix is able to detect gesture presence. The gesture is situation at least two isolated touches are detected on any axis.
GestureDistanceX, GestureDistanceY	if the gesture event is detected then the maximum gesture distance is calculated. The information is reported for X and Y axis.

Chapter 6 TSS Component

6.1 Introduction

This chapter describes the TSS Component provided in Processor Expert available in CodeWarrior for MCU v10.x and DriverSuite v 10.x product.

6.1.1 Integrating TSS Component

To integrate the TSS Component into a new or existing project, enable the Processor Expert engine first.

1. Open or create a project with Processor Expert support enabled.
2. Display the Processor Expert components library by choosing **Processor Expert > Show Views > Components Library**. In the **SW > Tools Library** folder, right-click the TSS_Library, and then select **Add to project**. See [Figure 6-1](#).

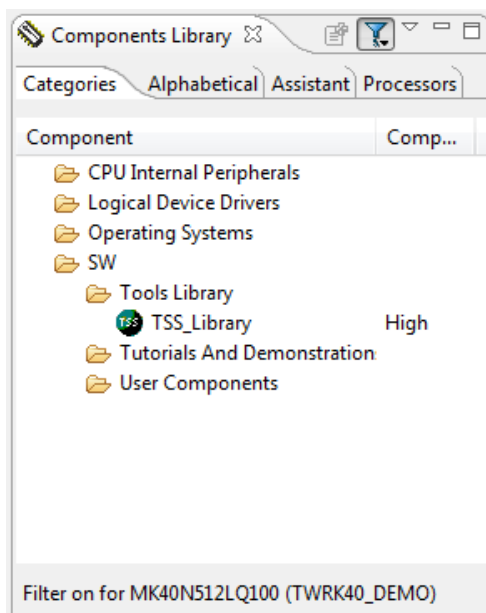


Figure 6-1. Components library window

3. [Figure 6-2](#) shows how the TSS Component appears in the Components window of the Processor Expert project. You are now ready to configure the TSS Component.

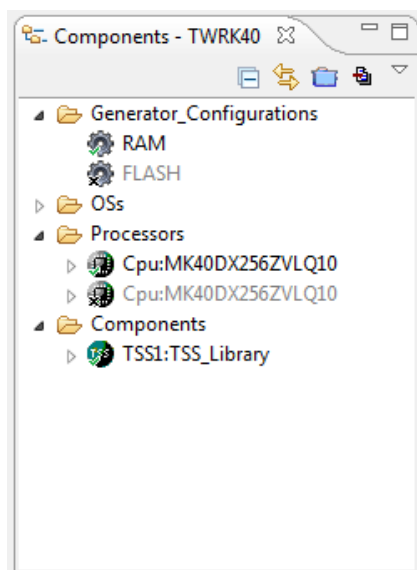


Figure 6-2. Processor Expert project Components window

6.2 Configuring TSS Component

The main tasks of the TSS component is to configure TSS, generate a standard TSS_SystemSetup.h configuration file and add all necessary files to the project.

Compared to manual editing of the TSS_SystemSetup.h file, the use of the Processor Expert TSS component is a more convenient way of configuring the TSS library. The Processor Expert has mechanisms to prevent conflicting settings of the TSS or the settings invalid for a particular MCU.

To configure the TSS component, click on the TSS_Library component in the Components window of the existing project to display the Component Inspector window.

6.2.1 TSS component properties

The main component configuration interface is Properties tab in Component Inspector Window (see [Figure 6-3](#)). The most of the properties is related to configuration of static parameters which are generated to TSS_SystemSetup.h.

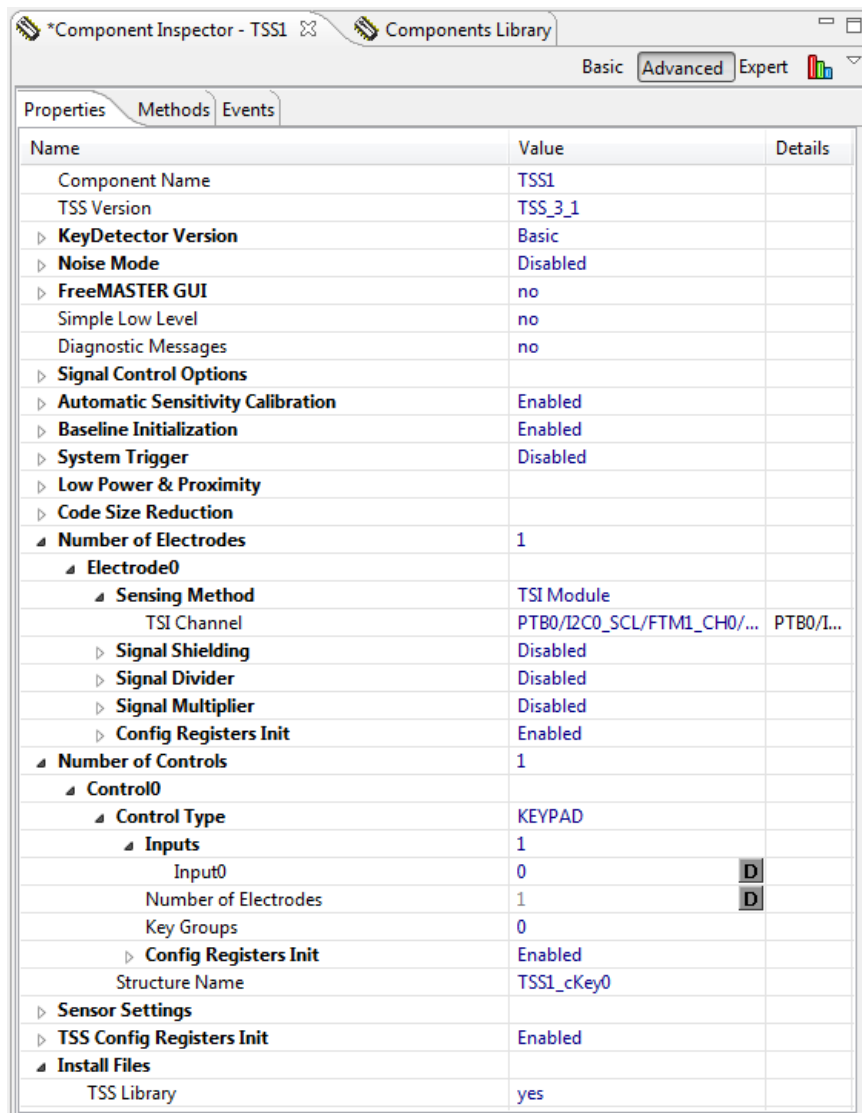


Figure 6-3. TSS component

For detailed help regarding the **TSS Component**, select **Help on Component**. It provides a detailed description of all the properties. In addition, each property in the component provides detailed description in the form of hint (see Figure 6-4). The hint is shown after a moment if cursor is placed above the property. For a detailed description of the settings related with TSS_SystemSetup.h, refer to Section 4.2, “System setup parameters”. For more information about runtime configuration options, refer to the *TSSAPIRM Reference Manual*.

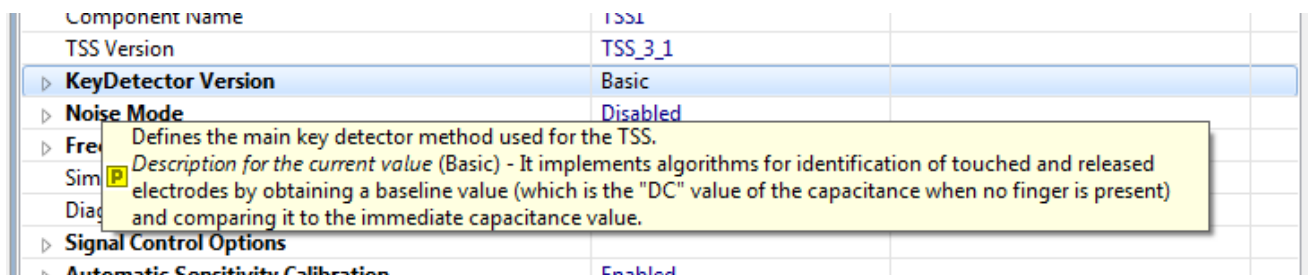


Figure 6-4. Property hint

6.2.2 TSS component Events

The Processor Expert TSS component defines the TSS callback functions as **Events**. On the **Component Inspector** window, switch to the **Events** tab as shown in Figure 6-5. The TSS component automatically generates the empty handler bodies of all callback functions in the Events.c file. The default callback name is generated as the Component Name used as prefix and the name set in the **Event procedure name** property. You can override this and use your own callback name.

The Events tab enables to define these kinds of callbacks:

- fOnFault — Callback for handling fault events detected by the TSS library. If the code generation is disabled then the fOnFault callback function is also disabled.
- fOnInit — Callback for implementation of peripheral initialization. Called automatically during the TSS_Init() function execution. This callback must be always defined.
- fOnProximity — Definition of the proximity callback enables proximity feature. The proximity callback is invoked only if the proximity mode is enabled.
- fCallbackX control — Callbacks for handling events generated by controls. Number of Event callbacks available depends on the number of enabled controls set in the Properties tab.

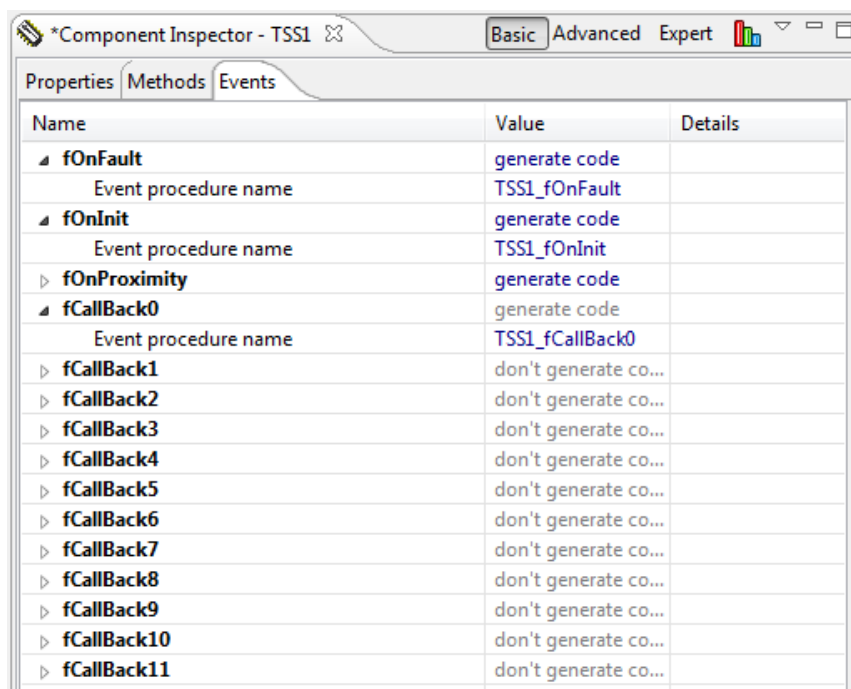


Figure 6-5. Callback specification

6.2.3 TSS component configuration related to Configure method

You can configure the TSS component to generate the `xxx_Configure()` method to initialize and configure internal registers and variables of the TSS library. Generally, the configure method performs the following library calls:

- `TSS_Init()`
- `TSS_SetSystemConfig(...)`
- `TSS_SetKeypadConfig(...)`
- `TSS_SetRotaryConfig(...)`
- `TSS_SetSliderConfig(...)`
- `TSS_SetASliderConfig(...)`
- `TSS_SetARotaryConfig(...)`
- `TSS_SetMatrixConfig(...)`

When the method code generation is enabled on the **Methods** tab, the function is named `<ComponentName>_Configure()` and is placed in the `ComponentName.c` file (see [Figure 6-6](#)).

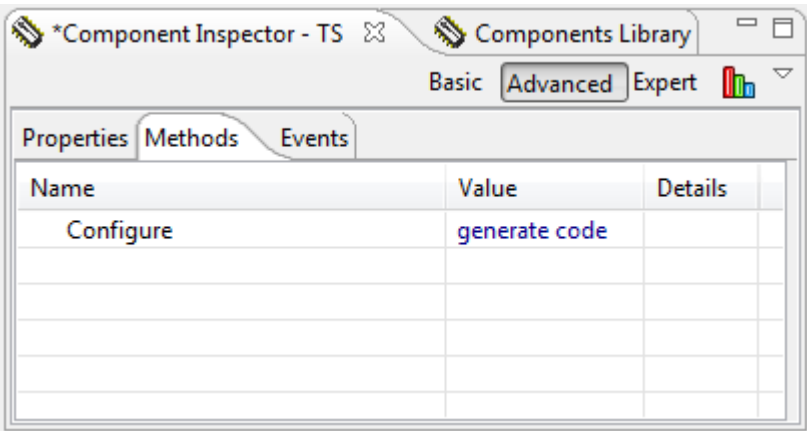


Figure 6-6. Configure method

NOTE

It is the user’s responsibility to call this function during the initialization of the application code.

The following TSS runtime parameters are initialized through the Configure call.

- The TSS system configuration registers are initialized when the *TSS Config Register Init* property is enabled as in Figure 6-7. For more information about this register configuration, refer to hints or help available with the TSS component or to the *TSSAPIRM Reference Manual*.

▲ TSS Config Registers Init	Enabled		
▲ System Configuration Register			
System Enabled	yes		
SWI Enabled	no		
DC Tracker Enabled	yes		
Stuck-key Enabled	no		
Water Tolerance Enabled	no		
Proximity Enabled	no		
LowPower Enabled	no		
Number of Samples	8	D	
DC Tracker Rate	100	D	
Slow DC Tracker Factor	100	D	
Response Time	4	D	
Stuck-keyTimeout	0	D	

Figure 6-7. TSS System Config Registers Initialization

- Control-specific configuration registers are initialized when the *Config Registers Init* property is enabled in the context of each specific control. See Figure 6-8 for keypad control example. The control callback is defined in **Events** tab, refer to Section 6.2.2, “TSS component Events” for more details. For more information about this register configuration, refer to the *TSSAPIRM Reference Manual*.

▲ Number of Controls	1	
▲ Control0		
▲ Control Type	KEYPAD	
▲ Inputs	1	
Input0	0	D
Number of Electrodes	1	D
Key Groups	0	
▲ Config Registers Init	Enabled	
▲ Control Configuration Register		
Control Enabled	yes	
Callback Enabled	yes	
Idle Enabled	no	
Idle Scan Rate	0	D
▲ Event Control and Status Register		
Keys Exceeded Enabled	no	
Buffer Overflow Enabled	no	
Auto-repeat Enabled	no	
Release Event Enabled	no	
Touch Event Enabled	yes	
Max Number of Touches	0	D
Auto Repeat Rate	0	D
Auto Repeat Start	0	D
Structure Name	TSS1_cKey0	

Figure 6-8. Control Config Registers Initialization

- Registers specific to low-power and proximity settings are initialized if the **Config Register Init** property is enabled. The **Proximity & Wake-Up Electrode** property specifies what electrode is used to wake the MCU from low power mode and proximity. **Proximity & Wake-Up Electrode Sensitivity** defines sensitivity of the electrode assigned to a wakeup event and proximity. See Figure 6-9.

▲ Low Power & Proximity		
▲ Low Power	Enabled	
Source	TSIO	TSIO
Threshold From Baseline	no	
▲ Config Registers Init	Enabled	
Proximity & Wake-Up Electrode	0	D
Proximity & Wake-Up Electrode Sensitivity	63	D
Low Power Scan Period	500 ms	

Figure 6-9. Low Power & Proximity Config Registers Initialization

- When the **Config Registers Init** property is enabled, select the mode for triggering by the **System Trigger Mode**.

▲ System Trigger	Enabled	
▲ Auto Trigger	TSI module	
Source	TSIO	TSIO
▲ Config Registers Init	Enabled	
System Trigger Mode	ALWAYS	

Figure 6-10. System Trigger Config Register Initialization

- Electrode-specific registers (general electrode enable and disable state and sensitivity) are initialized when the **Config Registers Init** property is enabled in a context of each specific electrode. See Figure 6-11. For more information about this register configuration, refer to the *TSSAPIRM Reference Manual*.

▲ Number of Electrodes	1	
▲ Electrode0		
▲ Sensing Method	TSI Module	
TSI Channel	PTB0/I2C0_SCL...	PTB0/I...
▷ Signal Shielding	Disabled	
▷ Signal Divider	Disabled	
▷ Signal Multiplier	Disabled	
▲ Config Registers Init	Enabled	
Electrode Enabled	yes	
Electrode Sensitivity	63	D
DC Tracker Enabled	yes	

Figure 6-11. Electrode Config Registers Initialization

6.2.4 Generating TSS component code

The final code can be generated when the TSS component is fully configured. The code generation starts when choosing the menu item **Project > Generate Processor Expert Code**.

If no error occurs, the component automatically creates or overwrites the generated code with the following files related to the TSS:

- derivative.h — Defines the IO_Map.h file as the main header file for mapping the I/O peripherals. This file is needed for compatibility between the Processor Expert and the standard TSS Library code.
- TSS_SystemSetup.h — The main TSS configuration header file set accordingly to the parameters specified in the TSS component.
- <ComponentName>.c — The TSS initialization and configuration of registers.
- <ComponentName>.h — The TSS initialization and configuration of registers header file.
- TSS folder
 - TSS binary file and all support files needed — The TSS library together with all the required support files are automatically copied to the project directory and are added to the project structure if the property **Install Files>TSS Library** is set. The correct TSS binary library file is automatically selected and added to the project based on the platform used.

- FreeMASTER
 - If **FreeMASTER GUI** and **Integrate FreeMASTER** property are enabled, a user can initialize FreeMASTER using inherited component and all necessary files are installed to the project.
- Events folder
 - events.c, events.h — The source and header files with all callback service implementations.

After the code is generated, check if the following paths are correctly defined in the **Project Properties > C/C++ Build > Settings** as needed:

- Include search paths to the TSS directory, for example "\${ProjDirPath}/Sources/TSS"
- Additional library paths to the selected library, for example "\${ProjDirPath}/Sources/TSS_KXX_M4.a"

Chapter 7 Using TSS FreeMASTER GUI

7.1 FreeMASTER Introduction

This chapter describes the steps to integrate the FreeMASTER component into the TSS application project. The FreeMASTER software is one of the off-chip drivers, which supports communication between the target microcontroller and PC. This tool allows the programmer to remotely control an application using a user friendly graphical environment that is running on a PC. The FreeMASTER GUI designed for the TSS allows to control, test and validate the TSS parameters, and display capacitance signals from the electrodes on PC in real time. The latest FreeMASTER version is available for download from at freescale.com/freemaster.

7.1.1 FreeMASTER integration

To use the FreeMASTER function with the TSS project, the FreeMASTER source files need to be included into the project.

1. Copy FreeMASTER files

- Manual files installation. The `TSS_USE_FREEMASTER_GUI` macro must be set to 1 in the `TSS_SystemSetup.h` file and files added manually to the project. See [Figure 7-1](#).



Figure 7-1. FreeMASTER group in CodeWarrior 10.x

- For installing and enabling FreeMASTER in the PE component, the **FreeMASTER GUI** property must be enabled. If **Integrate FreeMASTER** property is enabled, a user can initialize FreeMASTER and necessary files are installed to the project.
2. Additionally, the `freemaster.h` must be included in the `main.c` file or wherever the main application loop is located. If using the SCI communication, the SCI module must be initialized first before the FreeMASTER is initialized. The user is responsible for initialization of SCI, CAN, or other module.

- The FreeMASTER communication can take place with or without using interrupts. In case it is configured for one of the polled modes, the main loop must call the FMSTR_Poll() function periodically. You can see an example of the main in [Figure 7-2](#).

```
int main (void)
{
    /* Init HW */
    InitPorts(); /* Init application ports */
    /* Default TSS init */
    TSS_Init_Keypad0();
    /* Initialization of Touchpad Daughter Boards detection */
    MODULE_ID_Init();
    /* Init FreeMASTER resources */
    FreeMASTER_Init();
    /* Enable Interrupts globally */
    EnableInterrupts();
    /* Reset SW flag */
    LPSWflag = 0u;
    /* Main Loop */
    for(;;)
    {
        /* FreeMASTER */
        FMSTR_Poll();
        /* TSS Task */
        if (TSS_Task() == TSS_STATUS_OK)
        {
            /* Control of Touchpad Daughter Boards detection */
            (void) MODULE_ID_Check_Modules();
            /* Low Power Control */
            LowPowerControl();
        }
        /* Write your code here ... */
    }
}
```

Figure 7-2. Main loop with FreeMASTER service example

7.1.2 Communication interfaces

To set up the parameters related to communication between FreeMASTER application and the target board, open the Options dialog in the FreeMaster.

- RS232

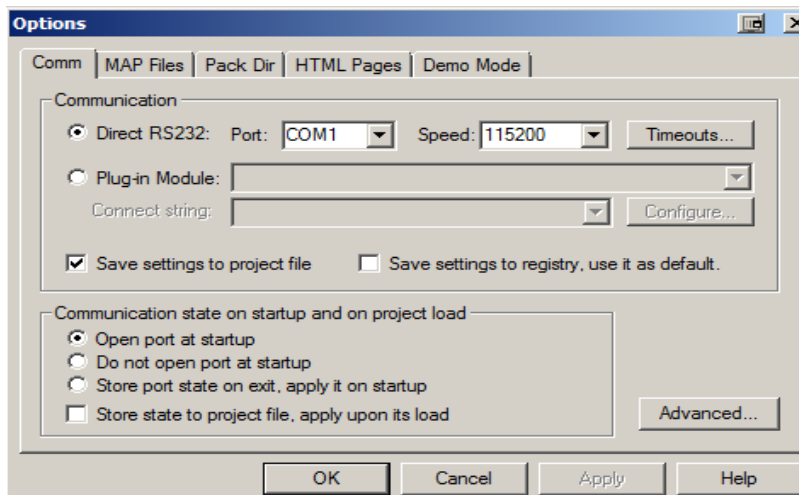


Figure 7-3. Select RS232 in Communication options

- User-defined communication via the FreeMASTER plug-in module. Tool supports P&E BDM Communication Plug-in for HCS08 and CFV1 (as shown in [Figure 7-4](#)) and JTAG communication for Kinetis platforms.

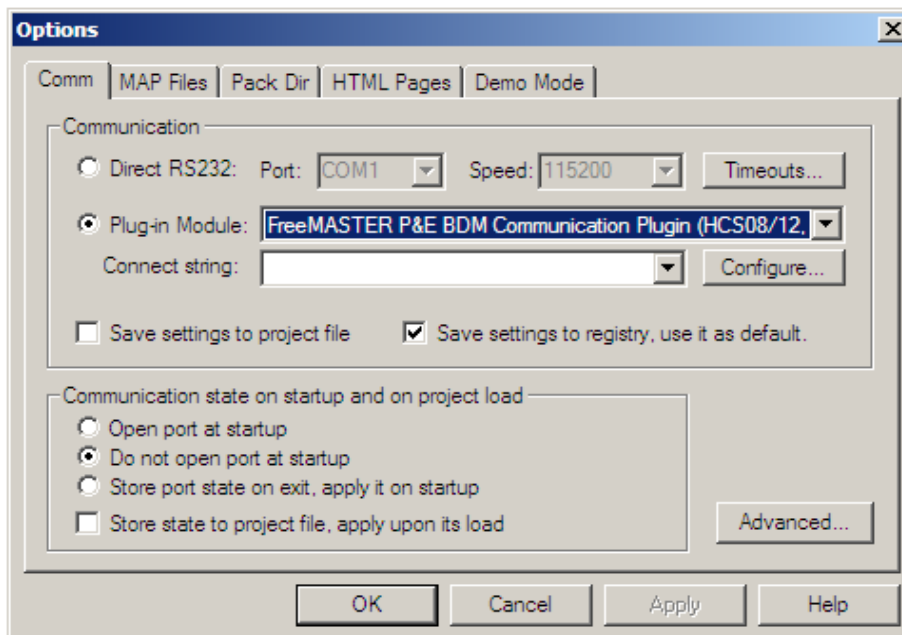


Figure 7-4. Plug-in Module in Communication options

The communication settings are easy and intuitive. Detailed information about communication settings can be found in the FreeMASTER User Guide document. Also refer to TSS example applications.

Note that the application ELF executable file must also be configured in the FreeMaster tool.

7.2 Signal visualization

The TSS signals can be visualized by using the FreeMASTER oscilloscope feature. It shows selected variables graphically in real-time. The variable values are obtained from the board application in real-time through the selected communication line. The oscilloscope function shows the behavior in real time of all the electrodes present in the system. There are some pre-defined configurations of the oscilloscope window, choose from the example FreeMASTER projects distributed within the TSS installation. See [Figure 7-5](#). Electrode signals can be displayed with or without the Baseline signals or electrode Delta signals.

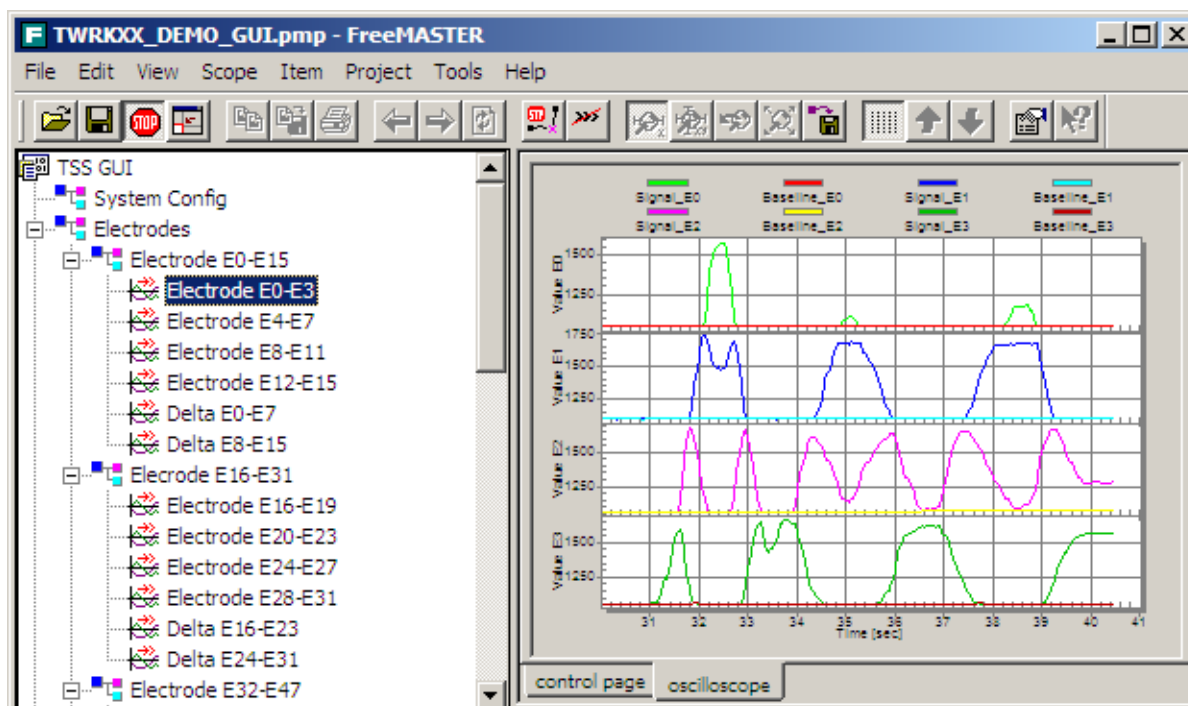


Figure 7-5. GUI visualization property

The scopes can be modified easily to display the electrode data but also any arbitrary variables from the target application. Find more details about the Oscilloscope feature and settings in the *FreeMASTER User Guide*.

7.3 Configuration panel

The TSS configuration pages enable reading the virtual registers of the TSS library. It also allows to modify the registers in real time without a need to reload the program in the MCU. You can modify the register values to test and debug the performance and behavior of the electrodes under different configuration parameters. The TSS configuration page also allows you to fine tune the sensitivity parameters for each electrode. The TSS configuration page displays three tabs—System Config, Electrodes, and Controls. See [Figure 7-6](#).

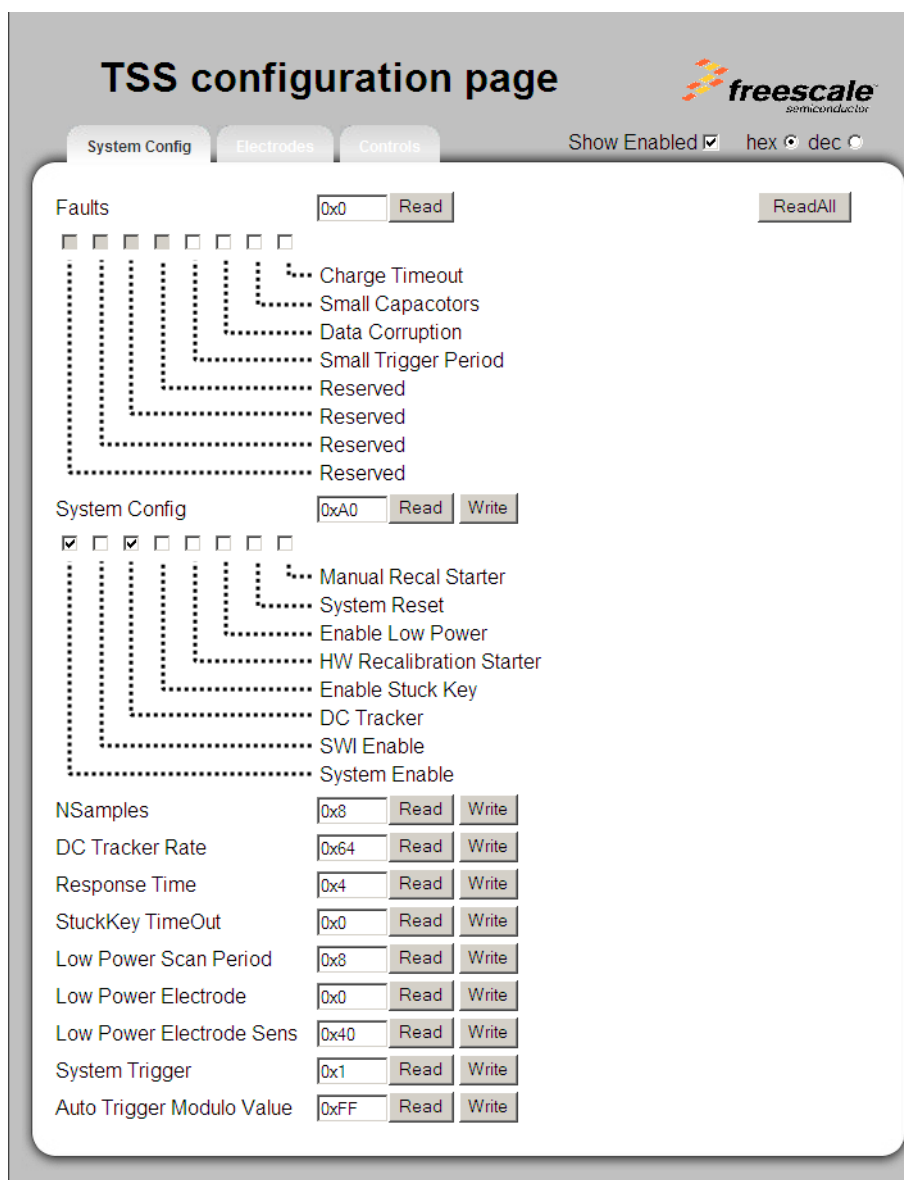


Figure 7-6. TSS configuration page

7.3.1 System config

The System config tab allows you to read or modify the TSS Configuration and Status registers. These registers allow to customize the TSS library operation and read back its status. To modify any value, change the value in the text box of each parameter, and click the Write button. You can use the Read all button to read all the parameters at once. See [Figure 7-6](#) for the TSS Configuration Page design. For more details about TSS Configuration and Status registers refer to the Touch Sensing Software API Reference Manual.

7.3.2 Electrodes configuration

The Electrodes tab allows you to read or modify the TSS registers associated with the electrodes. For each enabled electrode you can display—Signal, Baseline, Delta and Electrode state values. You can read and modify the sensitivity value for each enabled electrode and read or modify the electrode enabler register status or dc tracker enabler register.

To modify any sensitivity value, change the value in the text box and click the Write button. See [Figure 7-6](#) for the TSS Electrodes page design. For more details about TSS registers related to the electrode refer the *Touch Sensing Software API Reference Manual*.

TSS configuration page freescale
semiconductor

System Config **Electrodes** Controls Show Enabled ☒ hex ☒ dec ☐

El. Index	Signal	Baseline	Delta	Status	Sensitivity	Enabler	DcTracker
5	0x463	0x463	0x0		<input type="text" value="0x7"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
6	0x47E	0x47E	0x0		<input type="text" value="0x7"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
7	0x4BE	0x4BE	0x0		<input type="text" value="0x7"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
8	0x48B	0x48B	0x0		<input type="text" value="0x7"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Update ☒ Write

Figure 7-7. TSS electrodes configuration

NOTE

With Automatic Sensitivity Calibration feature enabled, the sensitivity values are controlled also by the autocalibration algorithms of the TSS.

7.3.3 Controls configuration

The Controls tab allows you to read or modify the TSS registers associated with the TSS controls. Use the drop down list to select one of the controls enabled in the target application. For each control the configuration page is automatically refreshed and displayed. There are different pages designed for Keypad control (see [Figure 7-8](#)), Slider and Rotary control (see [Figure 7-9](#)), and Matrix control.

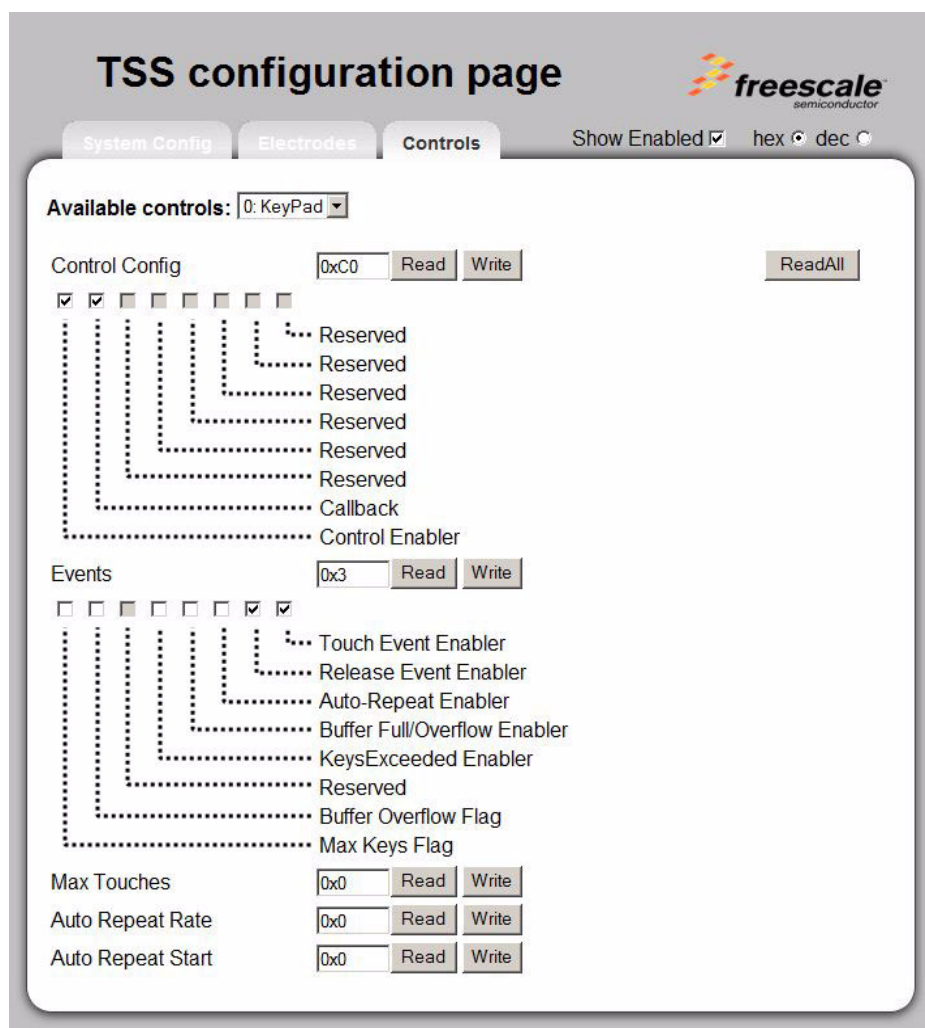


Figure 7-8. TSS keypad control configuration

For Keypad you can read and modify all available Control Config registers. You can use the Read all button to read all the parameters at once. To modify any value, change the value in the text box and click the Write button.

The screenshot shows the 'TSS configuration page' with the 'Controls' tab selected. The 'Available controls' dropdown is set to '3: Rotary'. The 'Control Config' section shows a value of '0xC0' with 'Read' and 'Write' buttons, and a 'ReadAll' button. Below this are several checkboxes, some of which are checked, and a 'Control Enabler' label. The 'Dynamic Status' section shows a value of '0x0' with a 'Read' button, and a 'Direction' icon. The 'Static Status' section shows a value of '0x0' with a 'Read' button, and a 'Touch state' icon. The 'Events' section shows a value of '0x2' with 'Read' and 'Write' buttons, and several checkboxes, some of which are checked. The 'Auto Repeat Rate' and 'Movement Timeout' sections show values of '0x0' with 'Read' and 'Write' buttons.

Figure 7-9. TSS Slider/Rotary Control Configuration

You can read and modify the following Slider and Rotary control registers: Control Config, Events enablers, Auto repeat Rate, and Movement Timeout.

It is possible to read and modify Control Config, Events enablers, Auto repeat Rate, Movement Timeout and Range registers for the analog slider, the analog rotary and the matrix . All other available registers are displayed as read-only. You can use the Read all button to read all the parameters at the same time. To modify any value, change the value in the text box and click the **Write** button.

For more details about TSS registers related to the controls refer to *TSSAPIRM Reference Manual*.

Appendix A Evaluation Board Options

A.1 Selecting the Evaluation Board for TSS Application

Freescall provides several evaluation board options to create demo applications before creating touch sensing enabled products. [Table A-1](#) describes the evaluation board options and their main features. This information will help you identify the best evaluation board option. [Table A-2](#) describes the currently supported evaluation touch pad boards with various numbers of electrodes and control configurations.

Table A-1. Evaluation boards

Board	Electrode types	Main MCU	Main features
TSSEVB	<ul style="list-style-type: none">• 8 in slider configuration• 8 in rotary configuration• 4 with LED backlighting• 4 multiplexed buttons• 4 single in different sizes• 6 multiplexed electrodes forming a 9 position keypad	MC9S08LG32	<ul style="list-style-type: none">• Includes a demonstration application• Includes a Communication MCU (MC9S08JM60 Comm MCU) that serves as a bridge between the application and the PC to evaluate the Electrode Graphing Tool (EGT) along with TSS.• Includes all the decoding structures supported by TSS along with special electrodes, such as different size electrodes and multiplexed electrodes supported by TSS.• Contains a custom on-board display that allows you to explore the software development combining the integrated LCD driver with TSS.• Includes an MC9S08LG32• Includes an OSBDM module that allows programming of the MC9S08LG32 MCU and the MC9S08JM60 Comm MCU, eliminating the need of an external BDM module
KITPROXIMITYEVM	8 electrodes configured in 3 ways—keypad, slider and rotary	Any demo board with "MCU_PORT" connector	<ul style="list-style-type: none">• Evaluation module, which connects to any demo board that has the MCU_PORT connector. Many S08 and V1 MCUs include this connector. So, the owners of these boards can buy TSSELECTRODEEVM as an add-on to their current evaluation tools.• Eight electrodes are common in the slider, the rotary, or the keypad, which implies only 8 GPIOs are needed to evaluate the three types of controls.• Can be used with the ColdFire® V1 Family demo
TWRPI-TOUCH-STR	2 daughter boards with 12 electrodes in keypad and rotary configuration.	Any TWR board with "TWRPI" connector	<ul style="list-style-type: none">• TWRPI-ROTARY• TWRPI-KEYPAD

Table A-1. Evaluation boards (continued)

Board	Electrode types	Main MCU	Main features
TWRPITSS-SHIELD	2 daughter boards with 4-6 electrodes in shield configuration.	Any TWR board with "TWRPI" connector	<ul style="list-style-type: none"> • TWRPITSS-SHIELD1 • TWRPITSS-SHIELD2
TWRPITSS-SLIDERS	3 daughter boards with 6-12 electrodes in analog slider, analog rotary and matrix configuration.	Any TWR board with "TWRPI" connector	<ul style="list-style-type: none"> • TWRPITSS-ROTARY3 • TWRPITSS-SLIDER2 • TWRPI-TOUCHPAD
KWIKSTIK	6 electrodes in keypad configuration (not multiplexed)	K40X256	<ul style="list-style-type: none"> • TSI and GPIO measurement method used • Touch status is indicated by onboard LCD and buzzer beep • FreeMASTER via serial connection by TWR-SER board and elevator modules • Also MQX demo available
FRDM-KL02Z	2 electrodes in analog slider configuration	KL02Z32	<ul style="list-style-type: none"> • GPIO method used • C++ demo available
FRDM-KL05Z	2 electrodes in analog slider configuration	KL05Z32	<ul style="list-style-type: none"> • TSI method used • C++ demo available
FRDM-KL25Z	2 electrodes in analog slider configuration	KL25Z128	<ul style="list-style-type: none"> • TSI method used • C++ demo available
FRDM-KL26Z	2 electrodes in analog slider configuration	KL26Z128	<ul style="list-style-type: none"> • TSI method used • C++ demo available
TWR-S08PT60	4 electrodes in keypad configuration (not multiplexed)	MC9S8PT60	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-MCF51JF	2 electrodes in keypad configuration (not multiplexed)	MCF51JF128	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-MCF51QM	2 electrodes in keypad configuration (not multiplexed)	MCF51QM128	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-KL05Z48	2 electrodes in keypad configuration (not multiplexed)	KL05Z32	<ul style="list-style-type: none"> • TSI method used • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-KL25Z48	2 electrodes in keypad configuration (not multiplexed)	KL25Z48	<ul style="list-style-type: none"> • TSI method used • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.

Table A-1. Evaluation boards (continued)

Board	Electrode types	Main MCU	Main features
TWR-KL46Z48	2 electrodes in keypad configuration (not multiplexed)	KL46Z128	<ul style="list-style-type: none"> • TSI method used • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-K15EN256	2 electrodes in keypad configuration (not multiplexed)	K15E256	<ul style="list-style-type: none"> • TSI method used • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-K20DX50	2 electrodes in keypad configuration (not multiplexed)	K20X128	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-K20D72M	2 electrodes in keypad configuration (not multiplexed)	K20X256	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-K40X256	4 electrodes in keypad configuration (not multiplexed)	K40X256	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector. • Also MQX demo available
TWR-K53N512	2 electrodes in keypad configuration (not multiplexed)	K53N512	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector. • Also MQX demo available
TWR-K60N512	4 electrodes in keypad configuration (not multiplexed)	K60N512VMD100	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector. • Also MQX demo available
TWR-K60D100	4 electrodes in keypad configuration (not multiplexed)	K60DN512VMD100	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector.
TWR-K70FN1M	4 electrodes in keypad configuration (not multiplexed)	K70FN1M	<ul style="list-style-type: none"> • Suitable for the owners of Tower System modules. • Other electrodes in different configurations (TWRPI_XXX boards) are possible to connect through the TWRPI connector. • Also MQX demo available

Table A-2. Evaluation touch pad boards

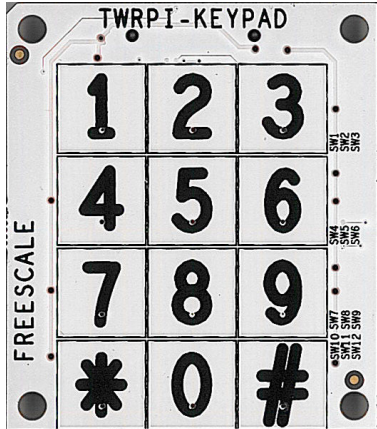
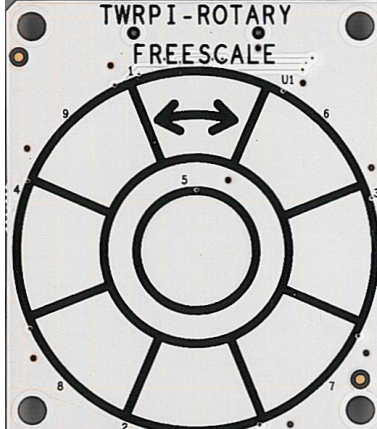
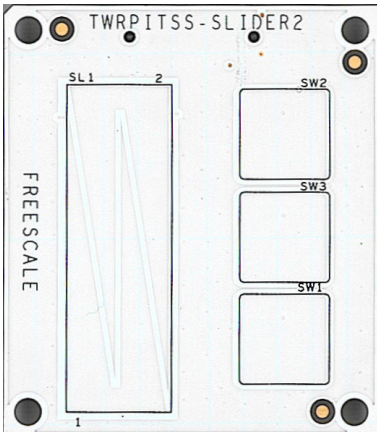
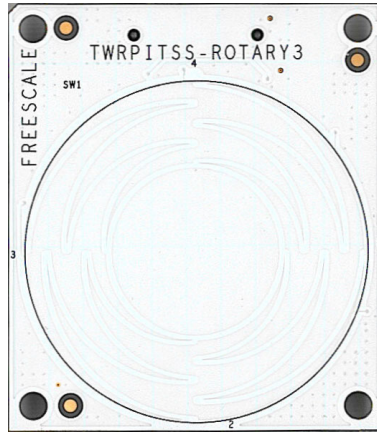
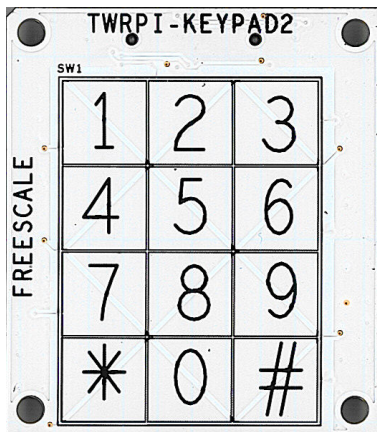
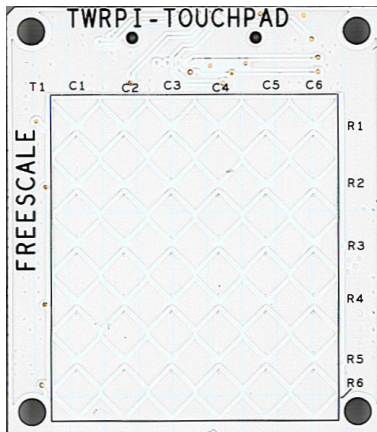
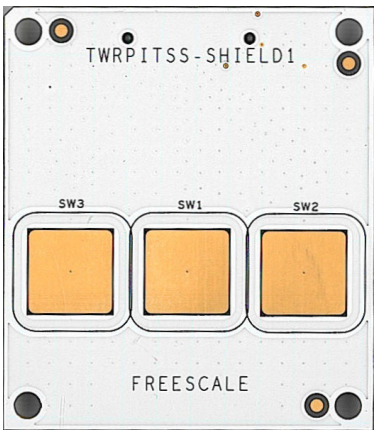
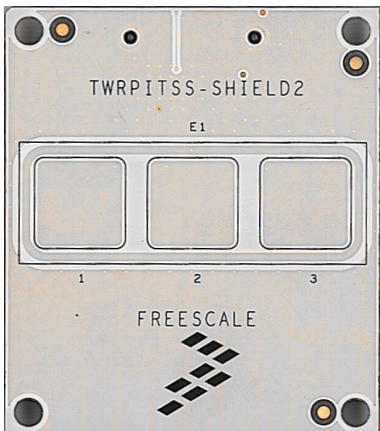
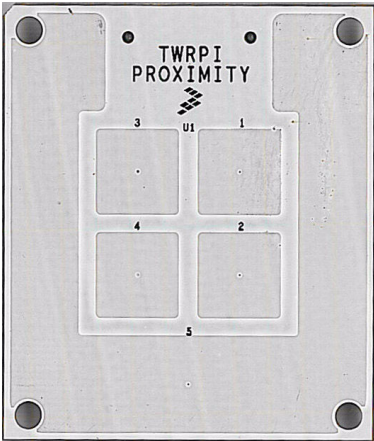
Board	Description	Board	Description
TWRPI-KEYPAD 12 electrodes in keypad configuration		TWRPI-ROTARY 9 electrodes in rotary configuration	
TWRPITSS-SLIDER2 2 electrodes in analog slider configuration		TWRPITSS-ROTARY3 4 electrodes in analog rotary configuration	
TWRPI-KEYPAD2 6 electrodes in group keypad configuration		TWRPI-TOUCHPAD 12 electrodes in matrix configuration	

Table A-2. Evaluation touch pad boards (continued)

Board	Description	Board	Description
TWRPITSS-SHIELD1 6 electrodes in keypad configuration	 The image shows the TWRPITSS-SHIELD1 board, a white PCB with three orange square electrodes labeled SW3, SW1, and SW2. The text 'TWRPITSS-SHIELD1' and 'FREESCALE' are printed on the board.	TWRPITSS-SHIELD2 4 electrodes in keypad configuration	 The image shows the TWRPITSS-SHIELD2 board, a white PCB with three orange square electrodes labeled 1, 2, and 3. The text 'TWRPITSS-SHIELD2', 'E1', and 'FREESCALE' are printed on the board.
TWRPI-PROXIMITY 4 electrodes in keypad configuration and one electrode for proximity feature	 The image shows the TWRPI-PROXIMITY board, a white PCB with four orange square electrodes labeled 1, 2, 3, and 4, and one electrode for proximity feature labeled 5. The text 'TWRPI PROXIMITY' and 'FREESCALE' are printed on the board.		

A.2 Links for TWRPI packs

TWRPI-TOUCH-STR, TWRPITSS-SHIELD and TWRPITSS-SLIDERS kit

- Navigate to [freescale.com/tower](https://www.freescale.com/tower) and search for "Tower System Plug In Modules".

