Touch Sensing Software

API Reference Manual

Document Number: TSSAPIRM
Rev. 7
08/2012

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

Document Number: TSSAPIRM
Rev. 7
08/2012

Chapter 1

Touch Sensing Software Library Overview

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, refer to:

http://www.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 for TSS 0.2.1
Rev. 2	10/2009	Updated for TSS 1.0 release
Rev. 3	11/2009	Updated for TSS 1.1 release
Rev. 4	06/2010	Updated for TSS 2.0 release
Rev. 5	04/2011	Updated for TSS 2.5 release
Rev. 6	03/2012	Updated for TSS 2.6 release Added a new text with the sections “TSI Lite low level sensor method”,“Simple low level routines”,“Default electrode level”,“Shielding function”,“Stuck-key function”,“Negative baseline drop”,“Control private data”,“TSS version information”,“Decoder interface”,“TSIL sample electrode control”,“Negative baseline drop”
Rev. 7	08/2012	Updated for TSS 3.0 release Added a new text with the sections “Signal normalization”,“Automatic sensitivity calibration”,“Baseline initialization”,“Electrodes groups”,“Analog slider and analog rotary decoder API”,“Reading the instant delta in the control”,“Proximity function”,“Automatic Sensitivity Calibration”,“Shielding function and Water tolerance”,“Water tolerance mode”,“Analog rotary”,“Analog Slider”,“Matrix”

Chapter 1
Touch Sensing Software Library Overview

This chapter describes features, architecture, and the software modules of the Touch Sensing Software (TSS) library. The following chapters cover the library API functions and use of resources.

1.1 Introduction

The TSS library enables capacitive sensing for all Freescale S08 and ColdFire® V1, ColdFire+, ARM®Cortex™-M4 based family microcontrollers (MCUs), and ARM®Cortex™-M0 based family microcontrollers (MCUs), providing the common touch sense decoding structures such as keypad, rotary, slider, analog slider, analog rotary, and matrix. It is implemented in a software layered architecture to enable easy integration into the application code and migration to other Freescale MCUs and customer customization.

1.2 Library features

The TSS library features include:

- Configurable number of electrodes from 1 to 64
- Configurable number of keypads, rotaries, sliders, analog sliders, analog rotaries, and matrixes
- Automatic electrode sensitivity calibration
- False detection prevention against external environment
- Electrode fault detection
- IIR and noise amplitude filters
- Shielding function
- Support for water-resistant touch systems
- Use of any standard MCU I/O as an electrode
- Touch Sense Input (TSI) module support
- One or more timer modules used with GPIO sampling algorithm
- Wakes from the MCU’s low power mode via the TSI module
- Three triggering modes supported (ALWAYS, SW, and AUTO when the TSI module is used)
- Library encoding is MISRA compliant
- Easy application integration achieved by a modular software layer architecture

1.3 Library architecture

The TSS library is implemented in a modular software layered approach shown in Figure 1-1. It provides a framework for Freescale touch sensing solutions, allowing further integration of new modules for application specific needs. It is structured only for linking the user application with the modules used in the application code. In addition, it permits the use of the modules in a user application for any layer in the library. Section 1.4, “System base modules” describes each module of the library.

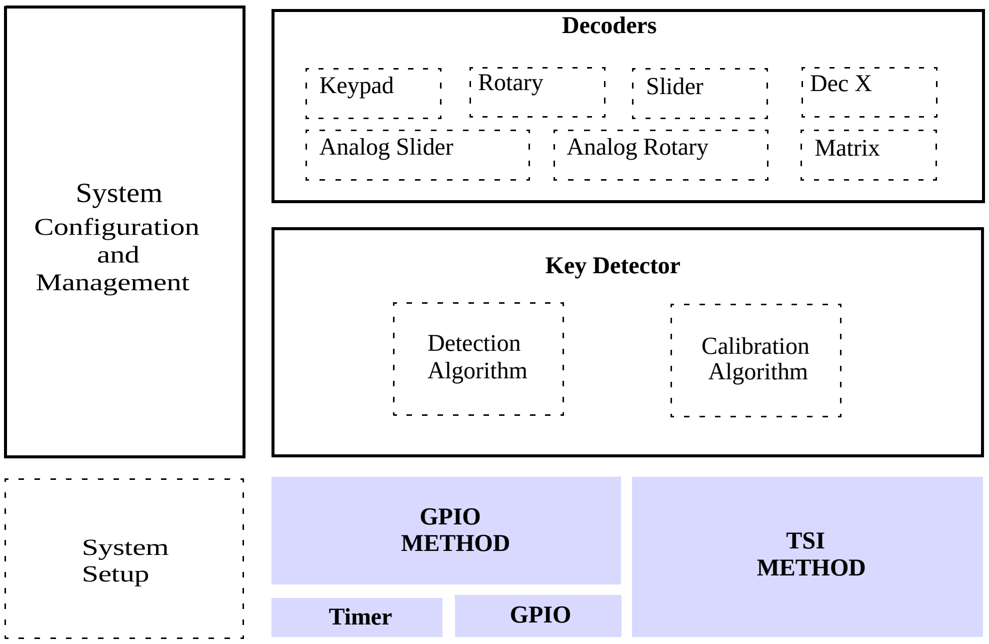
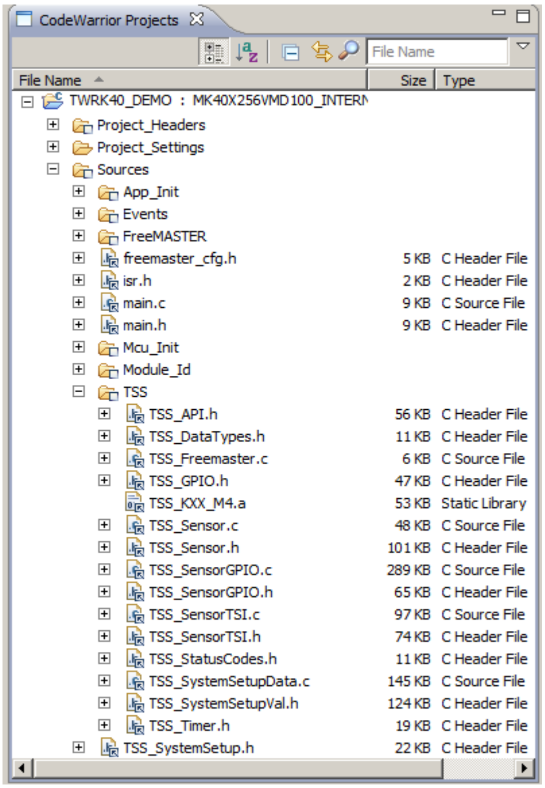


Figure 1-1. TSS library architecture

Figure 1-2 is an example of the application project using the ARM Cortex-M4 version of library files. As shown, the TSS folder contains all TSS library files.



```
CodeWarrior Projects
                                                File Name
File Name                                       Size    Type
⊟ TWRK40_DEMO : MK40X256VMD100_INTERN
   ⊞ Project_Headers
   ⊞ Project_Settings
   ⊟ Sources
      ⊞ App_Init
      ⊞ Events
      ⊞ FreeMASTER
      ⊞ freemaster_cfg.h                          5 KB  C Header File
      ⊞ isr.h                                     2 KB  C Header File
      ⊞ main.c                                    9 KB  C Source File
      ⊞ main.h                                    9 KB  C Header File
      ⊞ Mcu_Init
      ⊞ Module_Id
      ⊟ TSS
         ⊞ TSS_API.h                             56 KB  C Header File
         ⊞ TSS_DataTypes.h                       11 KB  C Header File
         ⊞ TSS_Freemaster.c                       6 KB  C Source File
         ⊞ TSS_GPIO.h                            47 KB  C Header File
           TSS_KXX_M4.a                          53 KB  Static Library
         ⊞ TSS_Sensor.c                          48 KB  C Source File
         ⊞ TSS_Sensor.h                         101 KB  C Header File
         ⊞ TSS_SensorGPIO.c                     289 KB  C Source File
         ⊞ TSS_SensorGPIO.h                      65 KB  C Header File
         ⊞ TSS_SensorTSI.c                       97 KB  C Source File
         ⊞ TSS_SensorTSI.h                       74 KB  C Header File
         ⊞ TSS_StatusCodes.h                     11 KB  C Header File
         ⊞ TSS_SystemSetupData.c                145 KB  C Source File
         ⊞ TSS_SystemSetupVal.h                 124 KB  C Header File
         ⊞ TSS_Timer.h                           19 KB  C Header File
      ⊞ TSS_SystemSetup.h                        22 KB  C Header File
```

Figure 1-2. Touch sensing software project

1.4 System base modules

This section provides an overview of the modules in the library and the source and header files that constitute these modules. The user configuration and API of each module is explained in Chapter 2, “Low-Level Interface.”

1.4.1 System setup module

Module description

The system setup module provides a compile time configuration interface to the user. In this module, a set of parameters that require a pre-compile definition, such as, electrode numbers, MCU pins to be used by each electrode, measurement method used by each electrode, and number of controls, can be configured.

Module files

This module contains the following files in open source:

- TSS_SystemSetup.h — Edit the TSS_SystemSetup.h file to customize it for the application particular electrode structure requirements.
- TSS_SystemSetupVal.h — Checks if the application configuration defined n the TSS_SystemSetup.h file is consistent. Do not edit the TSS_SystemSetupVal.h file.
- TSS_SystemSetupData.c — Creates the variables required for the TSS library that depend on the electrode structure of a particular application. Do not edit this file.

1.4.2 GPIO module

Module description

The GPIO module provides an interface between upper layers of the library and the MCU GPIO pins. This GPIO module is used by low-level routines for the default GPIO measurement method (detecting timeout condition). It specifically provides the following functionalities:

- Configure GPIO as input
- Configure GPIO as output
- Read GPIO value
- Set GPIO bit value (write a 1 to the GPIO bit on its specific data register)
- Clear GPIO bit value (write a 0 to the GPIO bit on its specific data register)
- Determine GPIO electrode port register address and mask of this pin
- Set GPIO pin output strength high or low
- Read GPIO pin output strength value
- Set GPIO pin slew rate high or low
- Read GPIO pin slew rate

Module files

The TSS_GPIO.h file implements the GPIO module. It defines macros to configure and access the MCU I/O pin values, pin output strength, and slew rate settings. The upper layer uses these macros to manipulate the MCU I/O pins.

1.4.3 Hardware timer module

Module description

The hardware timer module provides the interface between the upper layers and the hardware timer specified at compile time. This timer module is used by low-level routines for the default GPIO method to detect timeout condition.

This module covers the following functions:

- Timer configuration
  - Enable and disable interrupt
  - Set prescaler
  - Set and read timer module
- Timer start
- Timer stop
- Timer reset
- Set timer limit
- Read timer count
- Read 8 bits (low and high) of timer count

- Read 16 bits of timer count
- Hardware timer interrupt vector number assignment
- TPM, FTM, and MTIM8 timer modules are supported

Module files

The TSS_Timer.h file in the source code implements macros required to access the timer hardware. The hardware timer interrupt service routine is implemented in the TSS_Sensor.c file.

1.5 Capacitive sensing modules

This section provides an overview of the low-level capacitance measurement modules and the source and header files that constitute these modules.

1.5.1 GPIO low level sensor method

Method description

The GPIO low level sensing method measures the capacitance using the GPIO capacitive touch sensing method described in the Appendix A, “Touch Sensing Algorithms.” This method is in source code and provides the following functions:

- Configuration of all electrodes to a default state. The default state is output-high to achieve lower power consumption with external pull-up resistors.
- Initialization of the GPIO sensor method.
- Capacitance measurement of a specific electrode using GPIO method
- Hardware timer interrupt handling
- Driving an electrode pin as low-output state if a timer charge timeout is detected (probably electrode shorted to the ground)
- Noise amplitude filter function

To improve noise immunity and increase system sensitivity, the electrode sampling function can integrate several subsequent electrode sampling values.

The HCS08 and ColdFire V1 version of the TSS library automatically allocate the TSS_HWTimerIsr(void) interrupt service routine in the appropriate interrupt vector. The ColdFire+, ARM Cortex-M0, and ARM Cortex-M4 version of the TSS library needs manual allocation of this interrupt service routine.

Method files

The TSS_SensorGPIO.c and TSS_SensorGPIO.h files implement this method. The GPIO method uses the GPIO and timer module.

1.5.2 TSI low level sensor method

Method description

The TSI low level sensor method uses the TSI hardware (HW) peripheral module. Each TSI pin implements the capacitive measurement of an electrode having individual programmable detection thresholds and result registers. The TSI module can be functional in several low power modes and used to wake the CPU when a touch event or proximity event is detected. The TSI method provides an interface between the upper layers and the hardware. This module covers the following functions:

- TSI HW module initialization
- TSI HW module autocalibration
- MCU wakes from low power mode by a touch detection
- Triggering in all available modes (ALWAYS, software (SW), and AUTO)
- Starting the capacitance measurement on the pin
- Obtaining measured values

The HCS08 and ColdFire V1 version of the TSS library automatically allocate TSS_TSIxIsr(void) interrupt service routine in the appropriate interrupt vector. The ColdFire+, ARM Cortex-M0, and ARM Cortex-M4 version of the TSS library need manual allocation of this interrupt service routine.

Method files

The files TSS_SensorTSI.c and TSS_SensorTSI.h implement this method.

1.5.3 TSI Lite low level sensor method

Method Description

The TSI Lite Low Level Sensor method is a simplified algorithm for the second generation of Touch Sensing Input (TSI) HW peripheral module. Unlike the “full” TSI method and ARM Cortex-M0 version of TSI Lite, the HCS08 version does not enable to wake the device from the low power mode. The TSI method provides an interface between the upper layers and the hardware. This module covers the following functions:

- TSI HW module initialization
- TSI HW module autocalibration
- MCU wakes from low power mode by a touch detection on ARM Cortex-M0 version of TSI Lite
- Triggering in all available modes (ALWAYS, SW, and AUTO with external RTC, or LPTMR trigger source)
- Starting the capacitance measurement on the pin
- Obtaining measured values

The HCS08 version of the TSS library automatically allocates TSS_TSIxIsr(void) interrupt service routine in the appropriate interrupt vector. The ARM Cortex-M0 version of the TSS library needs manual allocation of this interrupt service routine.

Method Files

The files TSS_SensorTSIL.c and TSS_SensorTSIL.h implement this method.

1.6 Signal processing and decoding modules

This section provides an overview of the modules inside the precompiled library. The layer provides signal processing, touch detection, and decoding of events.

1.6.1 Key detector module

Module description

The key detector module determines what electrodes are pressed or released based on the values obtained by the capacitive sensing layer. It implements 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 baseline is obtained by a low pass IIR-based filter which removes slow environment changes that could cause a false electrode press detection, such as, air humidity, temperature, and other external variations. A de-bounce function is also implemented in this module to eliminate false detections caused by instantaneous noise.

The key detector module also contains an optional IIR filter for processing the capacitance signal. The filter processes values obtained from low-level routines and works with all low level sensor modules. Filtering may help to eliminate high frequency noise modulated on the capacitance signal and other external interference. The IIR equation is internally set to ratio 1/3 (current signal and previous signal).

The optional Shielding function is also processed in the module. The function enables to compensate signal drift on an electrode by shielding electrode which measures overall environment noise. The function may help to eliminate low frequency noise modulated on the capacitance signal, or protect the guarded system from detecting false touches caused by water drops.

The key detector also provides an automatic sensitivity calibration function. The function periodically adjusts the level of electrode sensitivity calculated according to the estimated noise level and touch tracking information. The user does not need to set the sensitivity anymore, but the settings are still availabe for more precise tuning.

The main output of this module is the indication of a finger presence or absence in each of the active electrodes.

The key detector module 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).

Main functions of the key detector module:

- Active electrodes detection pressed or not pressed
- Electrode detection debouncing
- Baseline generation
- IIR filtering of current capacitance signal
- Shielding function
- Sensitivity Autocalibration

- Electrode status reporting
- Fault reporting

Module files

The key detector module is implemented in the object code integrated inside the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, and TSS_KXX_M0.a/.lib library files.

For more information regarding the electrode touch detection method, refer to A.2, "TSI Module Based Touch Sensing Method."

1.6.2 Decoder module

Module Description

Decoders provide the highest level of abstraction in the library. In this layer, the information about touched-untouched electrodes is interpreted to present the status of a control in a behavioral way. Also, additional functionalities can be provided by the decoders, like the use of FIFOs. It is important to understand that the decoder-related code exists only once in memory. This implies that despite the number of, for example rotary controls present in the system, only one rotary decoder resides in the memory. Decoders can be seen as classes of an object oriented language. Each control has a decoder associated to it, so the control becomes an instance of the decoder (an object). However, not all decoders are necessarily instantiated in every system.

Decoder types supported by the library:

- Rotary
- Slider
- Keypad
- Analog rotary
- Analog slider
- Matrix

Module Files

The decoder module is implemented in the object code integrated in the TSS_S08.lib, TSS_CFV1.a, TSS_KXX_M4.a/.lib, and TSS_KXX_M0.a/.lib library files. For more information about the Decoders functionality, refer to A.4, "Decoder Functions."

1.7 System configuration and management module

Module description

This module implements the configuration and management functions required to integrate the library in the user application. The module contains the following functions:

- `TSS_Init()`— Initializes the TSS library.
- `TSS_Task()`— The function must be called periodically by the user application to provide CPU time to the TSS library. All electrodes are processed during a single execution of this function, but

the measured data are evaluated after at least two executions. The process status is reported by the return value.

- `TSS_TaskSeq()`— It is an alternative to `TSS_Task()` function. It must also be called periodically by the user application to provide the CPU time to the TSS library. The difference is that only a single electrode is processed in one function execution. When the function is executed periodically, it processes all electrodes and decoders. The status of the process is reported by the return value.
- `TSS_SetSystemConfig()`— Provides a way of configuring the TSS library during run-time.
- `TSS_GetSystemConfig()`— Provides reading of the TSS library registers during run-time.

This module can also perform integrity verification of the TSS RAM variables by executing a Checksum check, if enabled. The verification is embedded in all functions provided by this module.

Module files

The system configuration and management module is implemented in the object code integrated in the TSS_S08.lib, TSS_CFV1.a, TSS_KXX.a/.lib, and TSS_KXX_M0.a/.lib library files. The TSS_API.h file provides headers and macros required to integrate functions and variables defined in the library files.

Chapter 2 Low-Level Interface

The low-level interface is implemented by the system setup module. The low-level interface can be configured to specific application's requirements by editing the TSS_SystemSetup.h file. This file contains the system setup parameters that must be configured before application compilation and linking. The TSS_SystemSetupVal.h file checks these parameters for a consistency. If the parameters are not consistent, a compilation error message is issued. The TSS_SystemSetupData.c file creates the necessary variables according to the system setup parameters.

The GPIO module and timer module are also part of the low-level interface. The GPIO module implements macros and is implemented in a single TSS_GPIO.h file. The TSS timer module uses the timer overflow interrupt for timeout checking in the GPIO measurement method. This interrupt service routine is implemented in the TSS_Sensor.c file. The TSI module uses interrupts for measurement, autotriggering, and low power wake function management.

All interrupt service routines are implemented in the appropriate TSS_SensorXXX.c file. The Section 2.1, "System setup parameters" explains how to configure the TSS library in the user application.

2.1 System setup parameters

Table 2-1. System setup parameters

Parameter	Range	Description	Used
Features Configuration			
TSS_USE_SIMPLE_LOW_LEVEL	0-1	Enables the Simple Low Level routines option	Optional, default value depends on availability for the MCU
TSS_USE_DELTA_LOG	0–1	Enables the instant delta feature	Optional, default 0
TSS_USE_SIGNAL_LOG	0–1	Enables the instant signal feature	Optional, default 0
TSS_USE_FREEMASTER_GUI	0-1	Enables FreeMASTER GUI support	Optional, default 0
TSS_USE_CONTROL_PRIVATE_DATA	0-1	Enables Control Private Data feature	Optional, default 0
TSS_USE_AUTO_SENS_CALIBRATION	0-1	Enables automatic sensitivity calibration	Optional, default 0
TSS_USE_AUTO_SENS_CALIB_INIT_DURATION	0-255	Sets duration of an automatic sensitivity calibration initialization	Optional, default 100
TSS_USE_BASELINE_INIT_DURATION	0-255	Sets duration of a baseline initialization	Optional, default 0
Signal Control Options			

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_USE_GPIO_STRENGTH	0–1	Enables strength on usable pins for GPIO method.	Optional, default 0. Used only for the GPIO method.
TSS_USE_GPIO_SLEW_RATE	0–1	Enables slew rate on usable pins for GPIO method.	Optional, default 0. Used only for theGPIO method.
TSS_USE_IIR_FILTER	0–1	Enables IIR filter	Optional, default 0
TSS_USE_NOISE_AMPLITUDE_FILTER	0–1	Enables noise amplitude filter for GPIO method.	Optional, default 0. Used only for theGPIO method.
TSS_USE_SIGNAL_SHIELDING	0–1	Enables Signal Shielding function	Optional, default 0
TSS_USE_SIGNAL_DIVIDER	0–1	Enables signal divider	Optional, default 0
TSS_USE_SIGNAL_MULTIPLIER	0–1	Enables signal multiplier	Optional, default 0
TSS_USE_DEFAULT_ELECTRODE_LEVEL_LOW	0–1	Sets low electrode level between measurements for GPIO method.	Optional, default 0. Used only for the GPIO method.
Callbacks Definition			
TSS_ONFAULT_CALLBACK	Any valid function name. This callback function must be provided by the user.	This is the name of a function that matches the OnFault callback prototype	Optional. If the macro is not defined, the callback is disabled.
TSS_ONINIT_CALLBACK	Any valid function name. This callback function must be provided by the user.	This is the name of a function that matches the OnInit callback prototype	Optional. If macro is not defined, 'TSS_fOnInit' name is used.
TSS_ONPROXIMITY_CALLBACK	Any valid function name. This callback function must be provided by the user.	This is the name of a function that matches the OnProximity callback prototype	Optional. If macro is not defined, the proximity function is disabled.
Definition of Function Control Source			
TSS_USE_AUTOTRIGGER_SOURCE	RTC, LPTMR, TSI, TSI0, TSI1, UNUSED	Name of the hardware device used for management of TSS AUTO triggering	Optional, default UNUSED
TSS_USE_LOWPOWER_CONTROL_SOURCE	TSI, TSI0, TSI1, UNUSED	Name of hardware device used for management of the Low Power mode	Optional, default UNUSED
Code Size Reduction Options			
TSS_USE_DATA_CORRUPTION_CHECK	0–1	Enables compilation of TSS RAM data consistency check function	Optional, default 1

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_USE_TRIGGER_FUNCTION	0-1	Enables compilation of Triggering function	Optional, default 0
TSS_USE_STUCK_KEY	0-1	Enables compilation of Stuck Key detection function	Optional, default 1
TSS_USE_NEGATIVE_BASELINE_DROP	0-1	Enables compilation of Negative Baseline Drop function	Optional, default 1
TSS_USE_AUTO_HW_RECALIBRATION	0-1	Enables automatic hardware recalibration	Optional, default 1
Debug Options			
TSS_ENABLE_DIAGNOSTIC_MESSAGES	0–1	Enables diagnostic messages during compilation	Optional, default 0
Electrodes Configuration			
TSS_N_ELECTRODES	1–64	Sets the number of electrodes to be used by the application	Always
TSS_Ex_P	Any valid GPIO port on the MCU	Configures the MCU GPIO port to be used for each electrode	Always, except TSI module electrode
TSS_Ex_B	Any valid GPIO pin on the MCU	Configures the MCU GPIO pin to be used for each electrode	Always, except TSI module electrode
TSS_Ex_TYPE	GPIO, TSInCHm	Determines the measurement method for an electrode	Optional, default GPIO
TSS_Ex_NOISE_AMPLITUDE_FILTER_SIZE	2–255	Determines the size of the noise amplitude filter for an electrode	Optional, default 255
TSS_Ex_SHIELD_ELECTRODE	0-63	Determines the number of shielding electrode assigned to this electrode	Optional, if defined then the electrode is shielded.
TSS_Ex_SIGNAL_DIVIDER	0-255	Determines the value of signal divider for this electrode	Optional, if defined or non zero then the electrode uses divider.
TSS_Ex_SIGNAL_MULTIPLIER	0-255	Determines the value of signal multiplier for this electrode	Optional, if defined or non zero then the electrode uses multiplier.
Controls Configuration			
TSS_N_CONTROLS	0–16	Sets the number of controls to be used by the application	Always

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_Cn_TYPE	TSS_CT_KEYPAD TSS_CT_SLIDER TSS_CT_ROTARY TSS_CT_ASLIDER TSS_CT_AROTARY TSS_CT_MATRIX	Determines the type of the control	Always, if Controls > 0
TSS_Cn_ELECTRODES	1–16 for keypad 2–16 for slider 3–16 for rotary 2-16 for analog slider 3-16 for analog rotary	Determines the amount of electrodes that composes the control	Always, if Controls > 0 and matrix control is not used
TSS_Cn_ELECTRODES_X	2-16 for matrix	Determines the amount of electrodes that compose the matrix control on X axis	Always, if Controls > 0 and matrix control is used
TSS_Cn_ELECTRODES_Y	2-16 for matrix	Determines the amount of electrodes that compose the matrix control on Y axis	Always, if Controls > 0 and matrix control is used
TSS_Cn_STRUCTURE	Any valid name selected by the user	Indicates the name of the configuration and status structure of the control. It is used to create the configuration and status structure in a different file. This structure is not declared by users.	Always, if Controls > 0
TSS_Cn_CALLBACK	Any valid function name. This callback function is created by the user.	This is the name of a valid function that matches the callback prototype	Always, if Controls > 0
TSS_Cn_KEYS	Any valid array defined by an user	Defines the groups of electrodes for group decoder. Valid only for keypad control.	Optional.
Peripheral Specific Configuration			
TSS_HW_TIMER	TPMx, FTMx, MTIMx	Name of hardware timer used for GPIO method.	Always for GPIO method.
TSS_SENSOR_PRESCALER	NA, depends on used timer	Prescaler for all used timers	Optional, default 2. If the GPIO method is used.
TSS_SENSOR_TIMEOUT	128-65535, except HCS08 where 128-511(2047 if IIR filter is disabled) is used	Defines timeout for all used timers	Optional, default 511. If the GPIO method is used.

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_TSI_RESOLUTION	1–16 bits	Defines resolution of TSI in bits for autocalibration	Optional, default 11. If the TSI module is used.
TSS_TSI_EXTCHRG_LOW_LIMIT	0 — TSI module EXTCHRG range	Defines low limit of EXTCHRG for TSI autocalibration	Optional, default 0. If the TSI module is used.
TSS_TSI_EXTCHRG_HIGH_LIMIT	0 — TSI module EXTCHRG range	Defines high limit of EXTCHRG for TSI autocalibration	Optional, default 7. If theTSI module is used.
TSS_TSI_PS_LOW_LIMIT	0–7	Defines low limit of PS for TSI autocalibration	Optional, default 0. If the TSI module is used.
TSS_TSI_PS_HIGH_LIMIT	0– 7	Defines high limit of PS for TSI autocalibration	Optional, default 7. If the TSI module is used.
TSS_TSI_AMCLKS	0 Bus Clock 1 MCGIRCLK 2 OSCERCLK 3 Not valid	Defines TSI Active mode Clock Source	Optional if the TSI module supports AMCLKS function, default 0. If the TSI module is used.
TSS_TSI_AMCLKDIV	0 divider set to 1 1 divider set to 2048	Defines TSI Active Mode Clock Divider	Optional if the TSI module supports AMCLKDIV function, default 1. If the TSI module is used.
TSS_TSI_AMPSC	0 ~ divided by 1 1 ~ divided by 2 2 ~ divided by 4 3 ~ divided by 8 4 ~ divided by 16 5 ~ divided by 32 6 ~ divided by 64 7 ~ divided by 128	Defines TSI Active Mode Prescaler	Optional if TSI module supports AMPSC function, default 0. If the TSI module is used.
TSS_TSI_LPCLKS	0 LPOCLK 1 VLPOSCCLK	Defines TSI Low Power Mode Clock Source	Optional if TSI module supports LPCLKS function, default 0. If the TSI module is used.
TSS_USE_DVOLT	NA, depends on used TSI module	Defines TSI Delta Voltage value	Optional if TSI module provides DVOLT register, default is maximum value.
Proximity Specific Configuration			
TSS_SENSOR_PROX_PRESCALER	NA, depends on used timer	Prescaler for all used timers in proximity mode	Optional, default 2. If the GPIO method is used.

Table 2-1. System setup parameters (continued)

Parameter	Range	Description	Used
TSS_SENSOR_PROX_TIMEOUT	128-65535, except HCS08 where 128-511(2047 if IIR filter is disabled) is used	Defines timeout for all used timers in proximity mode	Optional, default 511. If the GPIO method is used.
TSS_TSI_PROX_RESOLUTION	1–16 bits	Defines resolution of TSI in bits for autocalibration in proximity mode	Optional, default 11. If the TSI module is used.
TSS_TSI_PROX_EXTCHRG_LOW_LIMIT	0 — TSI module EXTCHRG range	Defines low limit of EXTCHRG for TSI autocalibration in proximity mode	Optional, default 0. If the TSI module is used.
TSS_TSI_PROX_EXTCHRG_HIGH_LIMIT	0 — TSI module EXTCHRG range	Defines high limit of EXTCHRG for TSI autocalibration in proximity mode	Optional, default 7. If theTSI module is used.
TSS_TSI_PROX_PS_LOW_LIMIT	0–7	Defines low limit of PS for TSI autocalibration in proximity mode	Optional, default 0. If the TSI module is used.
TSS_TSI_PROX_PS_HIGH_LIMIT	0– 7	Defines high limit of PS for TSI autocalibration in proximity mode	Optional, default 7. If the TSI module is used.

2.1.1 Simple low level routines

The TSS library implements two versions of some low level measurement methods. Note that 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 only if TSI measurement method is used at least on one electrode.
- Simple low level routines are generally smaller but with limited functionality. The measurement is consecutive and the TSS_Task needs to wait for end of measurement. Auto triggering function is not available. The simple low level is also limited to just 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 2-2 shows availability of the feature on MCU families and measurement methods.

Table 2-2. 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, compilation is stopped with an error message.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.2 Instant delta values

The TSS library can also be configured to keep track of the instant delta values by storing them in an array. The instant delta function is automatically enabled if analog slider, analog rotary, or matrix control is used.

To enable this feature of the library 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 tss_ai8InstantDelta[TSS_N_ELECTRODES];
```

The library stores the instant delta value for each electrode every time the value is calculated. You can have access to these values by reading the tss_ai8InstantDelta array, specifying the number of electrodes, and using the standard C array addressing encoding. Access example of the tss_ai8InstantDelta array in C:

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

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.3 Instant signal values

The TSS library can also be configured to keep track of the instant signal values by storing them in an array.

To enable this feature of the library, 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 tss_au16InstantSignal[TSS_N_ELECTRODES];
```

The library stores the instant delta value for each electrode every time the value is calculated. You can have access to these values by reading the tss_ai8InstantDelta array, specifying the number of electrodes, and using the standard C array that addresses encoding.

Access example of the tss_au16InstantSignal array in C:

```
Temp = tss_au16InstantSignal[n];
```

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.4 GPIO strength

The TSS library can also enable the strength feature on the GPIO pins. When 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. The function is relevant only to the GPIO measurement method.

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 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.5 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. The function is relevant only to the GPIO measurement method.

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 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.6 Default electrode level

The TSS library can specify the default electrode pin voltage level in the idle state between each measurement. The option is relevant only to the GPIO measurement method.

Normally it is recommended to keep the idle state High. If from any reason a Low state is needed then use the following definition:

```
#define TSS_USE_DEFAULT_ELECTRODE_LEVEL_LOW  1
```

To see an example of the configuration, refer to Section 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.7 IIR filter

The TSS library can also enable the IIR filter feature. The filter processes current values obtained from the low-level routines and works with all 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 (current signal and previous signal).

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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.8 Noise amplitude filter

The TSS library provides the noise amplitude filter function. The function 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 filtered by the system, thus disregarding the noisy sample. 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. The function is relevant only to the GPIO measurement method. 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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.9 Shielding function

The TSS library provides the Shielding function. The function enables compensate 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.

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 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 and Negative Baseline Drop function are not performed on shielding, or the shielded electrode.

To see an example of the configuration, refer to Section 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.10 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 fit 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 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 then 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 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 equals 0, the multiplying of this electrode signal is disabled. The final electrode signal value is defined by the equation `En_SIGNAL = ( En_SIGNAL * TSS_En_SIGNAL_MULTIPLIER ) / TSS_En_SIGNAL_DIVIDER`.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.11 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 of the automatic sensitivity function, refer to Section A.3.5, "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 function is enabled the duration of initialization can be defined. To define the duration of initizalization, use the following definition:

```
#define TSS_USE_AUTO_SENS_CALIB_INIT_DURATION N
```

Where:

- *N* is the initizalization 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 the better estimation of noise level and the touch recognition reliability.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.12 Baseline initialization

The TSS library provides the setup of baseline initialization duration. The 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 touch recognition reliability.

To setup duration of baseline initialization, use the following definition:

```
#define TSS_USE_BASELINE_INIT_DURATION N
```

Where:

- *N* is the initizalization 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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.13 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 version 1.x. The OnFault callback function is available for all versions of libraries, HCS08, ColdFire V, Coldfire+, ARMCortex-M4, and ARMCortex-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 function in Section 3.4.4, “Faults register.”

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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.14 OnInit callback

The TSS library provides the OnInit Callback function. Use of this callback is mandatory. It must always be defined in an user application. The function is intended for placement of hardware initialization. The callback is executed during the TSS_Init function call, therefore every TSS reinitialization causes 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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”.

2.1.15 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. For more details about the proximity feature, refer to Section A.3.4, "Proximity function."

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 with low level sensor can be defined. Refer to Section 2.1.35, "Prescaler configuration of TSS hardware timer", Section 2.1.36, "Timeout configuration of TSS hardware timer" and Section 2.1.37, "TSI autocalibration settings".

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.16 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 A.3.2, "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 and Auto Trigger Modulo Value 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.

2.1.17 Low power control source

The TSS implements a 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 A.3.3, "Low power function."

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 Scan Period register, Low Power Electrode register, and 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.

2.1.18 Data corruption check

The TSS library provides the data corruption detection method. When 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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.19 Stuck-key function

The TSS library provides a Stuck-key detection feature. When 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.

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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.20 Negative baseline drop

The TSS library provides a negative baseline drop function. When enabled, the function adjusts the baseline level if the signal level is too low below the baseline. For more details about the negative baseline drop function, refer to Section A.3.6.1, “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 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h".

2.1.21 Automatic hardware recalibration

The TSS library provides an automatic hardware recalibration function. 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 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h".

2.1.22 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 observing signal behavior, tune sensitivities, setup the TSS system, controls 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 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.23 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 u8CntrlNum, void * pData)
```

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

```
void * TSS_GetControlPrivateData(UINT8 u8CntrlNum)
```

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h".

2.1.24 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 executed 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 executed in the application.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.25 Number of electrodes

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

For more details, refer to Table 2-1. To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.26 Electrode GPIO pin

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.

For more details, refer to Table 2-1. To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.27 Electrode type

The TSS library can also 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 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.28 Number of controls

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, and slider.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.29 Control type

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* can be replaced with 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.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.30 Number of electrodes assigned to control

To define the number of electrodes assigned to a given control except matrix control, use the following definition:

```
#define TSS_Cn_ELECTRODES      N
```

Where:

- *n* is the control number in the range 0 to `TSS_N_CONTROLS-1`
- *N* is the number of electrodes in the range 1 to 16 of control *n*

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

```
#define TSS_Cn_ELECTRODES_X      x
#define TSS_Cn_ELECTRODES_Y      y
```

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 16 for the X horizontal axis of control *n*
- *y* is the number of electrodes in the range 2 to `(16-TSS_Cn_ELECTRODES_X)` for the Y vertical axis of control *n*

Each control must have a definition specifying its number of electrodes. The electrodes are assigned to controls sequentially, starting from electrode 0.

To see an example of the configuration, refer to Section 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.31 Control configuration and status structure

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.

To see an example of the configuration, refer to Section 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

2.1.32 Application callback

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 see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.33 Electrodes groups

Keypad control provides an option to define electrodes 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* is one key

The element of the array forms one key. One key consists of the maximum 16 electrodes. 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. When not defined the default array of {0x01, 0x02, 0x04, 0x08, 0x10, ... } is used and forms each key with one electrode.

2.1.34 TSS hardware timer configuration

In case the application uses the GPIO measurement method, the TSS library requires the application to use a 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. For details refer to Section 2.1.35, "Prescaler configuration of TSS hardware timer" and Section 2.1.36, "Timeout configuration of TSS hardware timer".

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

To see an example of the configuration, refer to Section 2.2, "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.

2.1.35 Prescaler configuration of TSS hardware timer

In case the application uses the GPIO measurement method, the TSS library requires using the timer for measurement purposes. The timer prescaler may require to be adjusted to the application changing the counter speed that is used to measure the capacitance. As described in Section A.1, "GPIO Based Capacitive Touch Sensing Method," 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.

To see an example of the configuration, refer to Section 2.2, "Example of system setup parameters encoded in the TSS_SystemSetup.h."

2.1.36 Timeout configuration of TSS hardware timer

In case the application uses the GPIO measurement method, the TSS library requires using the timer for measurement purposes. 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.

2.1.37 TSI autocalibration settings

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 required value to the TSI module registers.

2.1.38 TSI active mode clock settings

In active mode, the TSI module has its full function 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

2.1.39 TSI low power mode clock settings

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 theTSI 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.

2.1.40 TSI delta voltage settings

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 2.2, “Example of system setup parameters encoded in the TSS_SystemSetup.h.”

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

```
/* TSS Features Configuration */
#define TSS_USE_SIMPLE_LOW_LEVEL              0
#define TSS_USE_DELTA_LOG                     1
#define TSS_USE_SIGNAL_LOG                    1
#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_BASELINE_INIT_DUARTION        15

/* Signal Control Options */
#define TSS_USE_GPIO_STRENGTH                 0
#define TSS_USE_SLEW_RATE                     0
#define TSS_USE_IIR_FILTER                    0
#define TSS_USE_NOISE_AMPLITUDE_FILTER        1
#define TSS_USE_SIGNAL_SHIELDING              1
#define TSS_USE_SIGNAL_DIVIDER                1
#define TSS_USE_SIGNAL_MULTIPLIER             1
#define TSS_USE_DEFAULT_ELECTRODE_LEVEL_LOW   0

/* Function Source Definition */
#define TSS_USE_AUTOTRIGGER_SOURCE            TSI0
#define TSS_USE_LOW_POWER_CONTROL_SOURCE      TSI0

/* 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_ELECTRODES   3        /* Number of electrodes in 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_ELECTRODES   2        /* Number of electrodes in the 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   3        /* Number of electrodes in the 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_ELECTRODES_X 3        /* Number of X electrodes in the control */
#define TSS_C3_ELECTRODES_Y 3        /* Number of Y 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*/

/* Low Power TSI definitions */
#define TSS_TSI_GENCS_LPCLKS      1 /* Set Low Power Mode Clock Source */

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

2.3 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 3 Application Interface

This section describes the TSS library initialization task function and how it is used by an application. It also presents the configuration and status structure of the library and how to manipulate it using the TSS_SetSystemConfig function.

Each control in the application has assigned configuration and status structure. Its parameters depend on control type. This section covers the configuration and status structure for each control type and information on how to manipulate them using the control configuration function. The section also describes callback functions for each control type.

The TSS_API.h file contains the function prototypes and variables declaration of the application interface. This file must be included in the applications source code that manages the TSS library.

3.1 TSS initialization function

This function initializes the data structures and low level routines of the library with default values. The OnInit callback is executed during the function call. It is important to have the MCU and peripherals clock configured before calling the `TSS_Init()`function or do it during the OnInit calback.

Function prototype

The TSS initialization function has the following prototype defined in the TSS_API.h file:

```
UINT8 TSS_Init(void);
```

Input parameters

None

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	TSS initialization has succesfully finished
TSS_STATUS_RECALIBRATION_FAULT	The fault occured during the recalibration of low level hardware

3.2 TSS task function

This function must be called periodically by the application to give the TSS library a CPU time. The `TSS_Task()` routine takes care of measurement initialization acquiring the sample and processing the capacitance signal values. Depending on the measurement method selected for given electrodes, the physical sampling is either performed by the HW module (for example,TSI) or directly by the `TSS_Task()` routine. This influences the duration of the `TSS_Task()` execution as well as the number of times it must be called before one full set of samples are taken and processed. Status of the processing is reported by the `TSS_Task()` return value.

Function prototype

The TSS task function has the following prototype defined in the TSS_API.h file:

```
UINT8 TSS_Task(void);
```

Input parameters

None

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	TSS task finished the measurement of all electrodes and has evaluated the values
TSS_STATUS_PROCESSING	Measurement of electrodes in progress or the TSS task has not finished the evaluation

NOTE

It is not recommended to perform any TSS register changes during electrode processing, that is until the `TSS_Task()` returns TSS_STATUS_OK.

3.3 TSS task sequenced function

The function `TSS_TaskSeq()` is a more “atomic” alternative to `TSS_Task()`. The `TSS_TaskSeq()`must also be called periodically by the application to provide the CPU time to the TSS library. Only one electrode is processed during a single call, the execution therefore takes less time and enables a smoother multitasking of other processes which are handled simultaneously with the `TSS_Task()`. The principle for two phase processing of electrodes is the same as in the `TSS_Task()`. As the function is periodically executed, it processes all electrodes and decoders.

Function prototype

The TSS task sequenced function has the following prototype defined in the TSS_API.h file:

```
UINT8 TSS_TaskSeq(void);
```

Input parameters

None

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	TSS sequenced task finished measurement of all electrodes and evaluated the values
TSS_STATUS_PROCESSING	Measurement of electrodes is in progress or the TSS sequenced task has not finished the evaluation

NOTE

It is not recommended to perform any TSS register changes during electrode array processing, that is until the `TSS_TaskSeq()` does not return TSS_STATUS_OK.

3.4 TSS Library Configuration and Status Registers

The TSS library Control and Status (C&S) registers are the main application interface to the TSS library. The application may use the registers to customize library operation and determine its status.

Although the C&S registers are located in RAM, they are read-only and declared as constant values for an application. This prevents a user from modifying status values or corrupting it with invalid configuration values.

3.4.1 Writing to the Configuration and Status Registers

To change values in the Configuration and Status register, the TSS_SetSystemConfig function must be called. The function may reinitialize low level hardware if needed. The function is declared in the TSS_API.h file.

Function prototype

```
UINT8 TSS_SetSystemConfig(UINT8 u8Parameter, UINT8 u8Value)
```

Input parameters

Type	Name	Valid Range and Values	Description
UINT8	u8Parameter	Any of the parameter codes provided by the configuration and status management	Code indicating the parameter configured
UINT8	u8Value	Depends on the specific parameter configured	The new desired value for the respective configuration register

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	Configuration was done successfully
TSS_ERROR_CONFSYS_NOT_IDLE	—

Return Value	Description
TSS_ERROR_CONFSYS_ILLEGAL_PARAMETER	Configuration was not successful due to illegal parameter number
TSS_ERROR_CONFSYS_READ_ONLY_PARAMETER	Configuration was not successful as the user attempted to modify a read-only parameter
TSS_ERROR_CONFSYS_OUT_OF_RANGE	Configuration was not successful as the new value was out of the range
TSS_ERROR_CONFSYS_LOWPOWER_SOURCE_NA	Configuration was not successful due to low power control source device is not selected
TSS_ERROR_CONFSYS_INCORRECT_LOWPOWER_EL	Configuration was not successful because the selected electrode does not belong to the low power control source device
TSS_ERROR_CONFSYS_TRIGGER_SOURCE_NA	Configuration was not successful because the trigger source device is not selected
TSS_ERROR_CONFSYS_PROXIMITY_CALLBACK_NA	Configuration was not successful because proximity callback was not defined
TSS_ERROR_CONFSYS_RECALIBRATION_FAULT	Configuration was not successful due to the recalibration fault
TSS_ERROR_CONFSYS_SHIELD_NA	Configuration was not successful because shielding functionality was not enabled

3.4.2 Reading the Configuration and Status registers

To read values from the Configuration and Status register, the application can either access the system structures directly or can use the TSS_GetSystemConfig function. The function is declared in the TSS_API.h file.

Function prototype

```
UINT8 TSS_GetSystemConfig(UINT8 u8Parameter)
```

Input parameters

Type	Name	Valid Range and Values	Description
UINT8	u8Parameter	Any of the parameter codes provided by the configuration and status management	Code indicating the parameter configured

Return value

The return value corresponds to an actual unsigned byte value of a register specified by u8Parameter.

The second option is a direct access to the TSS register structure. The Configuration and Status registers of the library are divided into two types. One set of registers is contained inside a structure, while the others are declared in global arrays. Register structure is defined as follows.

```
typedef struct{
         volatile const TSS_SYSTEM_FAULTS Faults;                         /Note 1
         volatile const TSS_SYSTEM_SYSCONF SystemConfig;                 //Note 2
         volatile const UINT8 Reserved;
         volatile const UINT8 NSamples;
         volatile const UINT8 DCTrackerRate;
         volatile const UINT8 ResponseTime;
         volatile const UINT8 StuckKeyTimeout;
         volatile const UINT8 LowPowerScanPeriod;
         volatile const UINT8 LowPowerElectrode;
         volatile const UINT8 LowPowerElectrodeSensitivity;
         volatile const TSS_SYSTEM_TRIGGER SystemTrigger;                //Note 3
         volatile const UINT8 AutoTriggerModuloValue;
} TSS_CSSystem;
```

NOTE

1. The TSS_SYSTEM_FAULTS type is an eight bit-field structure described in Section 3.4.4, "Faults register."
2. The TSS_SYSTEM_SYSCONF type is an 16 bit-field structure described in Section 3.4.5, "System Configuration register."
3. The TSS_SYSTEM_TRIGGER type is an eight bit-field structure described in Section 3.4.13, "System Trigger register."

The TSS library configuration and control structure instance is named:

tss_CSSys

To read any variable inside this structure, use the standard C structure read statements. Access example of the Fault register in C:

```
Temp = tss_CSSys.Faults;
```

To ensure a quick access to the electrode specific most critical configuration registers, the library defines the following registers as global arrays.

- `const unsigned char tss_au8Sensitivity[TSS_N_ELECTRODES];`
- `const unsigned char tss_au8ElectrodeEnablers[((TSS_N_ELECTRODES – 1)/8) + 1];`
- `const unsigned char tss_au8ElectrodeStatus[((TSS_N_ELECTRODES – 1)/8) + 1];`
- `const unsigned char tss_u8ConfigChecksum;`

To read any of these registers, use the standard C array read statements. Access an example of the au8Sensitivity register in C:

```
Temp = tss_au8Sensitivity[ Electrode ];
```

As mentioned above although the registers can be read directly as the part of the configuration structure or from global arrays, they all must be written using the TSS_SetSystemConfig() function. For more details, refer to Section 3.4.1, "Writing to the Configuration and Status Registers."

3.4.3 Configuration and Status registers list

Table 3-1. Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	Faults	Section 3.4.4, "Faults register"	0x00	RW — Shows the faults generated by the system.
0x01	2	SystemConfig	Section 3.4.5, "System Configuration register"	0x00	RW — Main configuration of the TSS library.
0x02	1	NSamples	Section 3.4.6, "Number of Samples registers"	0x08	RW — Determines the number of samples scanned for a single measurement.
0x03	1	DCTrackerRate	Section 3.4.7, "DC Tracker Rate register"	0x64	RW — Determines how often the recalibration function will occur. This number represents the number of calls to the TSS task function is required before recalibration.
0x04	1	ResponseTime	Section 3.4.8, "Response Time register"	0x04	RW — Configures the number of TSS task calls necessary to determine if a key is pressed or not.
0x05	1	StuckKeyTimeout	Section 3.4.9, "Stuck-key Timeout register"	0x00	RW — Configures the number of TSS task calls necessary to detect a key stuck.
0x06	1	LowPowerScanPeriod	Section 3.4.10, "Low Power Scan Period register	0x0F	RW — Determines value of the low power scan period.
0x07	1	LowPowerElectrode	Section 3.4.11, "Low Power Electrode register	0x00	RW — Number of the electrode scanned in low power mode and proximity mode.
0x08	1	LowPowerElectrodeSensitivity	Section 3.4.12, "Low Power Electrode Sensitivity register	0x3F	RW — Sensitivity of the electrode ¡'scanned in low power mode and proximity mode.
0x09	1	SystemTrigger	Section 3.4.13, "System Trigger register	0x01	RW — Configures TSS system triggering modes.
0x0A	1	AutoTriggerModuloValue	Section 3.4.14, "Auto Trigger Modulo Value register	0xFF	RW — Defines the modulo value of the Autotrigger clock source
0x0B	A	Sensitivity	Section 3.4.15, "Sensitivity Configuration register"	0x3F	RW — Each byte represents the sensitivity for each electrode. This value is the required difference between any instant value and the baseline to indicate a touch.

Table 3-1. Configuration and Status registers (continued)

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x0B + A	B	ElectrodeEnablers	Section 3.4.16, “Electrode enablers”	All electrodes enabled (all bytes in 0xFF)	RW — Each bit represents the enabler of an electrode. If a bit is 0, then the corresponding electrode will not be scanned.
0x0B + A + B	C	ElectrodeStatus	Section 3.4.17, “Electrode status”	All electrodes in 0x00	R — Each bit represents the current status of an electrode. 1 is electrode detected as touched, and 0 is electrode detected as untouched.
0x0B + A + B + C	1	ConfigChecksum	Section 3.4.18, “Configuration Checksum Register”	undetermined	R — This register contains the checksum value to guarantee the validity of the information contained in C&S registers.

3.4.4 Faults register

This register holds the information regarding why an electrode cannot be read. When this fault occurs, the library stores the information about the causes of the fault. The HCS08 version of the library allows you to enable a software interrupt to read the Faults register and determine what caused the fault. All HCS08, ColdFire V1, Coldfire+ , ARM Cortex-M0, and ARM Cortex-M4 version allows you to use the OnFault callback to handle the fault situation as indicated by the Fault register.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R W	▨	▨	▨	▨	SmallTriggerPeriod	Data Corruption	Small Capacitor	Charge Timeout
Reset:	0	0	0	0	0	0	0	0

▨	Unimplemented or reserved

Figure 3-1. Faults register

Table 3-2. Faults register description

Signal	Description
SmallTriggerPeriod	Indicates if the trigger period is long enough to cover needs of the time for the scanning of all active electrodes. This bit must be cleared by writing a 0 to it through the configuration function. If the SWI is enabled in the System Configuration register, an interrupt will be generated if this condition is met. If the OnFault callback is enabled, then the callback will be generated if this condition is met. The system will be disabled to prevent cyclic fault and needs 34to be reenabled. 1 — SmallTriggerPeriod detected 0 — No SmallTriggerPeriod detected
Data Corruption	Indicates if a change in the system configuration data has been detected through an invalid direct access to memory. This bit must be cleared by writing a 0 to it through the configuration function. If the SWI is enabled in the System Configuration register, an interrupt is produced if this condition is met. If the OnFault callback is enabled, then the callback will be generated if this condition is met. The system will be disabled to prevent cyclic fault and needs to be reenabled. If the data corruption check function is not enabled in TSS_SystemSetup.h the bit is not functional, refer to Section 2.1.18, “Data corruption check". 1 — Unaccounted change detected 0 — No invalid change detected
Small Capacitor	Indicates if the required voltage level change was detected too fast in one or more electrodes. This bit must be cleared by writing a 0 to it through the configuration function. If the SWI is enabled in the System Configuration register, an interrupt will be produced if this condition is met. If the OnFault callback is enabled, then the callback will be generated if this condition is met. The system will disable the electrode(s) that caused this fault. If automatic hardware recalibration is enabled in TSS_SystemSetup.h then the hardware recalibration is also executed in the case of electrode fault. 1 — Small capacitor detected 0 — No small capacitor detected
Charge Timeout	Indicates if the capacitor did not reach the required voltage level during the configured time an one or more electrodes. This bit must be cleared by writing a 0 to it through the configuration function. If the SWI is enabled in the System Configuration register, an interrupt will be generated if this condition is met. If the OnFault callback is enabled, then the callback will be generated if this condition is met. The system will disable the electrode(s) that caused this fault. If automatic hardware recalibration is enabled in TSS_SystemSetup.h then the hardware recalibration is also executed in the case of electrode fault. 1 — Charge timeout detected 0 — No timeout fault detected

3.4.5 System Configuration register

This register is used to enable the library features. You can change these features depending on your application needs.

Register Number = 0x01

	15	14	13	12	11	10	9	8
R W	System Enabler	SWI Enabler	DC-Tracker Enabler	Stuck-key Enabler	Reserved	Reserved	Reserved	Water Tolerance Enabler
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R W	Proximity Enabler	Low Power Enabler	Reserved	Reserved	Reserved	Hardware Recalibration Starter	System Reset	Manual Recalibration Starter

Figure 3-2. System Configuration register

Table 3-3. System Configuration register descriptions

Signal	Description
System Enabler	1 — System is enable 0 — System is disable
SWI Enabler	If any fault occurs (refer to Faults register), an SWI will be generated. This bit has effect only for the HCS08 version of the TSS library. The OnFault callback can be used for the same purpose on all HCS08, ColdfireV1, ARMCortex-M4 and ARMCortex-M0 versions of the TSS library. 1 — SWI is generated with any fault 0 — No SWI is generated by faults
DC-Tracker Enabler	1 — The DC Tracker filtering mechanism is enabled 0 — The DC Tracker filtering mechanism is disabled
Stuck-key Enabler	1 — Stuck-key is enabled 0 — Stuck-key is disabled
Water Tolerance Enabler	1 — Water tolerance is enabled 0 — Water tolerance is disabled
Proximity Enabler	1 — Proximity Mode is enabled 0 — Proximity Mode is disabled
Low Power Enabler	1 — Low Power Mode is enabled 0 — Low Power Mode is disabled
Hardware Recalibration Starter	Writing a 1 to this register schedules a low level hardware calibration for the next time the TSS task is executed. The recalibration is performed only on electrodes which are not touched. If more electrodes belong to the same hardware module then all these electrodes need to be released.
System Reset	Writing a 1 to this register resets the system immediately
Manual Recalibration Starter	Writing a 1 to this register schedules a baseline calibration for the next time the TSS task is executed

3.4.6 Number of Samples registers

Register Number = 0x02

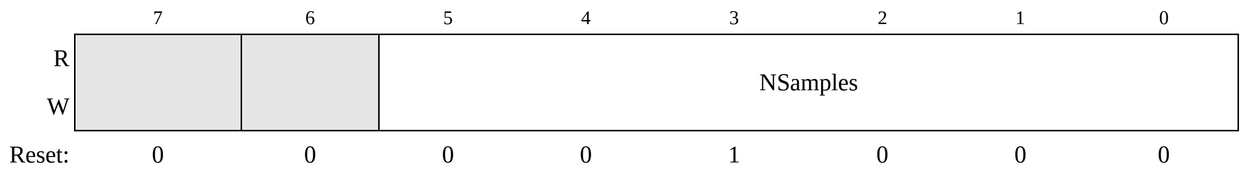


Figure 3-3. Number of Samples register

Table 3-4. Number of Samples register descriptions

Signal	Description
NSamples	Determines the number of capacitance samples acquired to obtain a single measurement. These samples are taken per electrode per task execution. 1–32 — *n* samples taken

3.4.7 DC Tracker Rate register

Register Number = 0x03

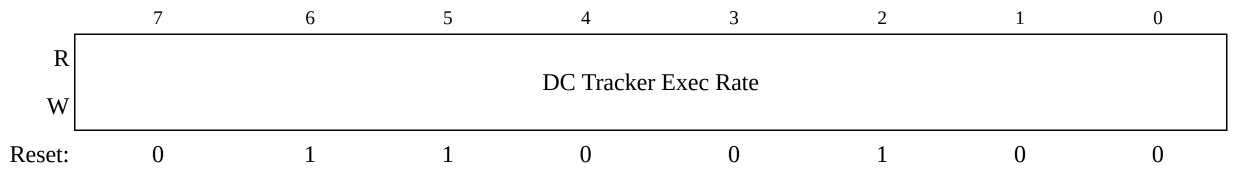


Figure 3-4. DC Tracker Rate register

Table 3-5. DC Tracker Rate register description

Signal	Description
DC Tracker Exec Rate	Determines how often the recalibration function occurs. This number represents the number of calls to The TSS task function required before recalibration. If this register is set to 0, the DC tracker feature will be disabled. 0 — DC tracker feature disabled 1–255 — DC tracker executed every *n* task executions

3.4.8 Response Time register

Register Number = 0x04

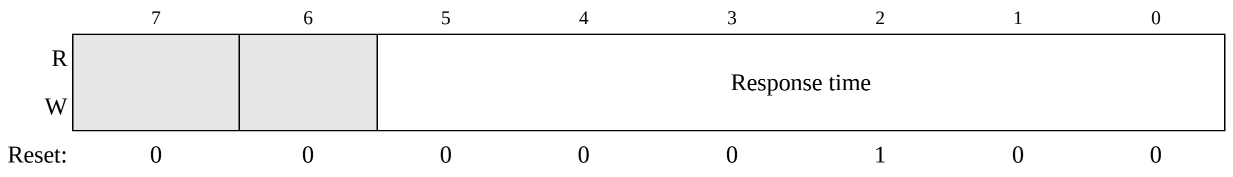


Figure 3-5. Response Time register

Table 3-6. Response Time register description

Signal	Description
Response Time	Determines the number of measurements required to detect the following: • Status change in any electrode • Initial recalibration • Capacitance Baseline shift Capacitance samples are taken per task execution, and therefore this value represents the number of calls to the TSS task function required for an electrode status change detection. 1–32 — *n* measurements used

3.4.9 Stuck-key Timeout register

This register configures how many times the library task must keep detecting a touch before it performs a recalibration task. This feature allows you to protect your application from false permanent touches caused by an external increase of the electrodes capacitance. The electrode is only considered touched until the recalibration occurs. If the stuck-key function is not enabled in TSS_SystemSetup.h then writing to this register is not possible, refer to Section 2.1.19, “Stuck-key function".

Register Number = 0x05

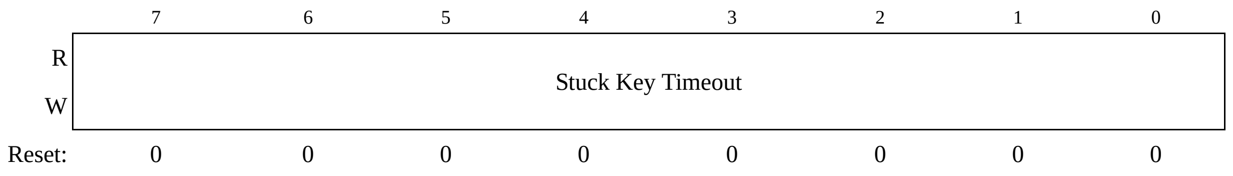


Figure 3-6. Stuck-key Timeout register

Table 3-7. Stuck-key Timeout register description

Signal	Description
Stuck Key Timeout	Determines how many task executions an electrode must be detected as touched before recalibrating that electrode and then detect it was untouched. If this value is set to 0, the stuck key feature is disabled. 0 — Stuck key feature disabled 1–255 — Electrode recalibrated after n task executions.

3.4.10 Low Power Scan Period register

This register represents the value of the scan period in the low power mode. The low power scan period value is selected from the list of values, see Table 3-8. If the low power control source is not selected in TSS_SystemSetup.h then writing to this register is not possible, refer to Section 2.1.17, “Low power control source".

If TSI is used for the low power mode control source, the register content is mirrored to the TSI_GENCS[LPSCNITV]. If the TSI does not provide the definition of the LPSCNITV then the low power scan register is not used.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Low Power Scan Period							
Reset:	0	0	0	0	1	1	1	1

Figure 3-7. Low Power Scan Period register

Table 3-8. Low Power Scan register description

Signal	Description
Low Power Scan Period	Determines scan period in low power mode. Only selection from the list of predefined values is allowed. 0000 -> 1 ms scan interval 0001 -> 5 ms scan interval 0010 -> 10 ms scan interval 0011 -> 15 ms scan interval 0100 -> 20 ms scan interval 0101 -> 30 ms scan interval 0110 -> 40 ms scan interval 0111 -> 50 ms scan interval 1000 -> 75 ms scan interval 1001 -> 100 ms scan interval 1010 -> 125 ms scan interval 1011 -> 150 ms scan interval 1100 -> 200 ms scan interval 1101 -> 300 ms scan interval 1110 -> 400 ms scan interval 1111 -> 500 ms scan interval

3.4.11 Low Power Electrode register

This register represents the electrode number scanned in the low power mode or in proximity mode. In low power mode and proximity mode, only one pin is active and is able to perform electrode capacitance measurements.

The low power electrode number must be selected from the module related to the low power control source defined by TSS_USE_LOWPOWER_CONTROL_SOURCE in TSS_SystemSetup.h. In case the electrode did not match this condition, the System Setup Config Error is generated. If the low power control source is not selected or neither proximity callback defined in TSS_SystemSetup.h then writing to this register is not possible, refer to Section 2.1.17, “Low power control source".

If the TSI module is used as a low power control source, the register content is used to setup the TSI_PEN[LPSP].

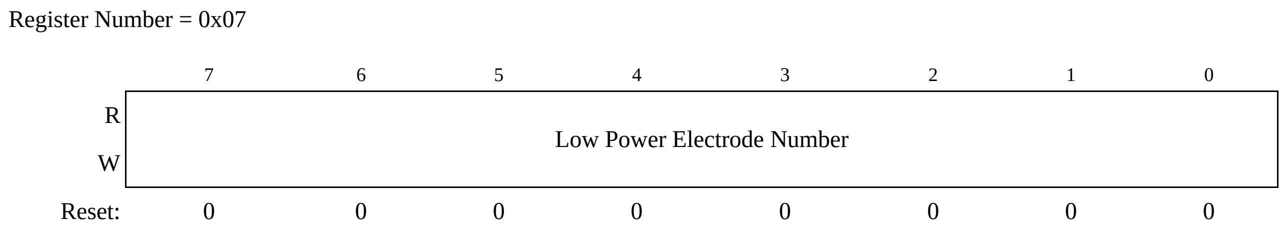


Figure 3-8. Low Power Electrode register

Table 3-9. Low Power Electrode register description

Signal	Description
Low Power Electrode Number	Number of the electrodes scanned in the low power mode or in proximity mode. The range of the electrode numbers is 0 – 63

3.4.12 Low Power Electrode Sensitivity register

The Low Power Sensitivity register defines the sensitivity value used to wake from the low power mode or sensitivity threshold for proximity detection. This sensitivity means relative delta threshold value from a baseline level.

The low power control source device manages scanning the selected electrode during the Low Power mode. When the electrode value exceeds a defined value the TSS wakes the device from the low power mode and enters active mode. If the low power control source is not selected or neither proximity callback defined in TSS_SystemSetup.h then writing to this register is not possible, refer to Section 2.1.17, “Low power control source".

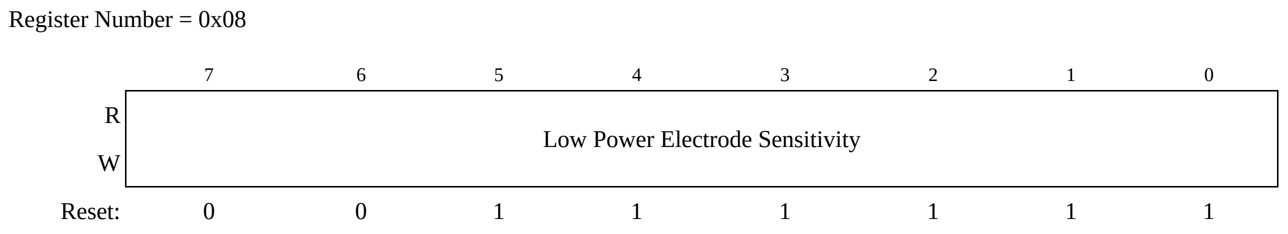


Figure 3-9. Low Power Electrode Sensitivity register

Table 3-10. Low Power Electrode Sensitivity register description

Signal	Description
Low Power Electrode Sensitivity	The sensitivity value for wakeup from low power mode. The range of the sensitivity is 1–255

3.4.13 System Trigger register

This register is used to enable and set up the library trigger features. If the trigger function is not enabled in TSS_SystemSetup.h then writing to this register is not possible, refer to Section 2.1.16, "Trigger function".

Register Number = 0x09

	7	6	5	4	3	2	1	0
R / W	Reserved	Reserved	Reserved	Reserved	Reserved	SWTrigger	TriggerMode	TriggerMode
Reset	0	0	0	0	0	0	0	1

Figure 3-10. System Trigger register

Table 3-11. System Trigger register descriptions

Signal	Description
TriggerMode	0x00 —> TSS_TRIGGER_MODE_AUTO 0x01 —> TSS_TRIGGER_MODE_ALWAYS 0x10 —> TSS_TRIGGER_MODE_SW 0x11 —> Reserved TriggerMode is not set if SWTigger bit is set in the same moment.
SWTrigger	Write only bit. This bit controls software triggering scan execution. In case the TSS_TRIGGER_MODE_SW is selected, write 1 to the software trigger performs one scan cycle execution, otherwise the bit is not functional.

3.4.14 Auto Trigger Modulo Value register

If TSI module is used as a trigger source and AUTO triggering mode is selected then the Auto Trigger Modulo Value register defines the period of regular scanning during the AUTO triggering mode. This is a modulo value of the TSI active mode clock defined in the TSS_SystemSetup.h. Refer to Section 2.1.38, "TSI active mode clock settings". If the trigger function is not enabled in TSS_SystemSetup.h then writing to this register is not possible, refer to Section 2.1.16, "Trigger function".

Register Number = 0x0A

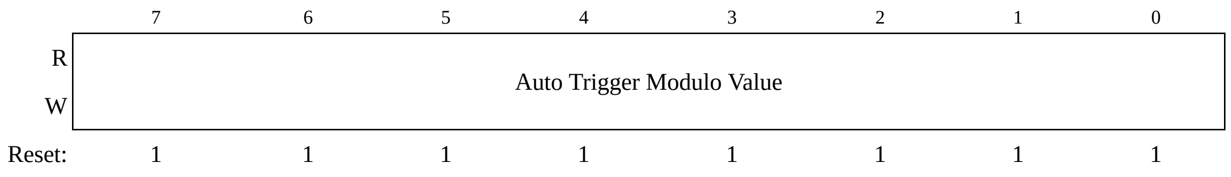


Figure 3-11. Auto Triger Modulo Value register

Table 3-12. Auto Trigger Modulo Value register description

Signal	Description
Auto Trigger modulo Value	The Auto Trigger Modulo Value register defines the period of scanning during the AUTO triggering mode

3.4.15 Sensitivity Configuration register

This set of registers contain a sensitivity value for each electrode. The sensitivity value is an 8-bit value that represents the minimum difference between an instant signal value versus the baseline required to indicate a touch. The sensitivity does not need to be set if Automatic Sensitivity Calibration is enabled, refer to Section A.3.5, “Automatic Sensitivity Calibration.”

Register Number = 0x0B

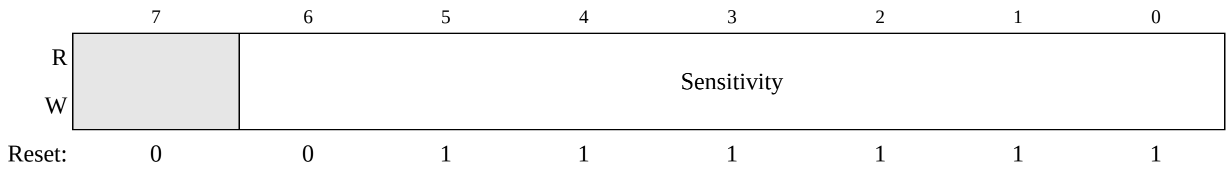


Figure 3-12. Sensitivity Configuration register

Table 3-13. Sensitivity Configuration register description

Signal	Description
Sensitivity	Represents the capacitance threshold for an electrode detected as touched, relative to the baseline value of that electrode. 2–127 — Electrode sensitivity

3.4.16 Electrode enablers

This set of registers contain the information about which electrode is enabled. Each bit represents one electrode. Bit value 1 represents enabled electrode, and a 0 represents disabled electrode. It is important to note that certain faults automatically disable electrodes and it is then up to the application to re-enable it. If automatic hardware recalibration is enabled in TSS_SystemSetup.h then the hardware recalibration bit is automatically set in the System Config register which leads to hardware recalibration in the next TSS_Task execution.

The arrangement of electrodes with bits is as follows:

- Register 0, bit 0 — Electrode 0
- Register 0, bit 1 — Electrode 1
- Register 0, bit 7 — Electrode 7
- Register 1, bit 0 — Electrode 8
- Register 1, bit 7 — Electrode 15
- And so on

If the electrode enabler settings are changed then hardware recalibration is scheduled for system adaptation to the new electrode configurations. The hardware recalibration starter bit in the System Configuration register is automatically set for this purpose, refer to Section 3.4.5, “System Configuration register".

3.4.17 Electrode status

This set of registers provides the electrode status (touched or untouched). Each bit represents one electrode. A 1 represents a touched electrode, while a 0 represents an untouched electrode.

The arrangement of electrodes with bits is as follows:

- Register 0, bit 0 — Electrode 0
- Register 0, bit 1 — Electrode 1
- Register 0, bit 7 — Electrode 7
- Register 1, bit 0 — Electrode 8
- Register 1, bit 7 — Electrode 15
- etc

3.4.18 Configuration Checksum Register

This read-only register contains the checksum value to guarantee the validity of the information contained in C&S registers. When the TSS_SetSystemConfig() function is called to update a C&S register, a checksum algorithm is performed and the result is stored in this register. The TSS_Task() function checks the integrity of this checksum. If it fails, the data corruption bit of the fault register is set. This function can be disabled in TSS_SystemSetup.h by define TSS_USE_DATA_CORRUPTION_CHECK, refer to Section 2.1.18, “Data corruption check.”

NOTE

If you write to the Configuration and Status registers directly without using the TSS_SetSystemConfig() function, the Checksum register will not be updated and the data corruption bit will be set on the next execution of the TSS_Task() function.

3.5 Keypad decoder API

This section describes the keypad decoder API. It describes the keypad Configuration and Status registers, the TSS_SetKeypadConfig() function used to write to these registers, and the data structure used for reading this register. This section also describes the callback function and its parameters for keypad control.

Each application keypad control has its respective Configuration and Status register and callback function.

3.5.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, use the TSS_SetKeypadConfig function declared in the TSS_API.h file.

Function prototype

```
UINT8 TSS_SetKeypadConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter, UINT8 u8Value);
```

Input parameters

Return valueReturn value

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control’s C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by keypad decoder	Code indicating the parameter configured
UINT8	u8Value	Depends on the specific parameter configured	New desired value, for the respective configuration register

Return value

The return value is an unsigned byte with the following possible return values defined in the file TSS_StatusCodes.h:

Return Value	Description
TSS_STATUS_OK	Configuration was executed successfully
TSS_ERROR_KEYPAD_ILLEGAL_CONTROL_TYPE	Configuration was not executed. The Id parameter did not match the control type structure the user was trying to modify.
TSS_ERROR_KEYPAD_READ_ONLY_PARAMETER	Configuration was not done as the user attempts to modify a read-only parameter.
TSS_ERROR_KEYPAD_OUT_OF_RANGE	Configuration was not done because the new value was out of the established boundaries.
TSS_ERROR_KEYPAD_ILLEGAL_PARAMETER	Configuration was not done due to an illegal parameter number.

3.5.2 Reading the Configuration and Status registers

There are two options how to read keypad Configuration and Status register structure. The first option uses TSS function declared in the TSS_API.h file.

Function prototype

```
UINT8 TSS_GetKeypadConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter);
```

Input parameters

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control’s C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds to the actual unsigned byte value of the keypad register specified by u8ControlId and u8Parameter.

The second option is to directly access the configuration and status structure for the specific data. The structure name is defined by the application in the TSS_SystemSetup.h file using the following definition:

```
#define TSS_Cn_STRUCTURE      cKey0
```

Where *n* is the Control number starting from 0 to the number of controls minus one.

The cKey0 name is just an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                             //Note 1
        const TSS_KEYPAD_CONTCONF ControlConfig;                    //Note 2
        UINT8 BufferReadIndex;
        const UINT8 BufferWriteIndex;
        const TSS_KEYPAD_EVENTS Events;                             //Note 3
        const UINT8 MaxTouches;
        const UINT8 AutoRepeatRate;
        const UINT8 AutoRepeatStart;
        const UINT8 * const BufferPtr;
} TSS_CSKeypad;
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.5.4, “Control ID register.”
2. The TSS_KEYPAD_CONTCONF type is an eight bit-field structure described in Section 3.5.5, “Control Configuration register.”
3. The TSS_KEYPAD_EVENTS type is an eight bit-field structure described in Section 3.5.9, “Event Control and Status register.”

Using the example of the structure *cKey0*, to read Control ID register use:

```
Temp = cKey0.ControlId;
```

3.5.3 Configuration and Status registers list

The table below describes the keypad control Configuration and Status registers, Control ID Register.

Table 3-14. Keypad Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.5.4, "Control ID register"	Application dependable	R — Displays the control type and control number
0x01	1	ControlConfig	Section 3.5.5, "Control Configuration register"	0x00	RW — Configures overall enablers of the object
0x02	1	BufferReadIndex	Section 3.5.7, "BufferReadIndex"	0x00	RW — Index of the first unread element of the events buffer
0x03	1	BufferWriteIndex	Section 3.5.8, "BufferWriteIndex"	0x00	R — Index of the first free element of the buffer
0x04	1	Event Control and Status Register	Section 3.5.9, "Event Control and Status register"	0x00	RW — Configures the events that will be reported by the controller. This Register holds the Max Key and Buffer Overflow Flags
0x05	1	MaxTouches	Section 3.5.10, "MaxTouches register"	0x00	RW — Configures the maximum number of keys that can be pressed at once.
0x06	1	AutoRepeatRate	Section 3.5.11, "Auto Repeat Rate register"	0x00	RW — Configures the rate at which keys will be reported when they are kept pressed and no movement is detected.
0x07	1	AutoRepeatStart	Section 3.5.12, "Auto Repeat Start register"	0x00	RW — Configures the time between a touch and an auto-repeat event when the key is kept pressed.
0x08	1	BufferPtr	Section 3.5.6, "Buffer Pointer register"	Assigned at precompile time	R — Address of the buffer where the events for each controller are stored

3.5.4 Control ID register

This read-only register contains the control's identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-13. Control ID Register

Encoding	**Control Type**
001	Keypad
010	Slider
011	Rotary
100	Analog slider
101	Analog rotary
110	Matrix
100-111	Reserved

NOTE

The control number is assigned in the system setup module. This value is unique for all controls, implying each control has a particular number regardless of its type. The first assigned control number is zero.

3.5.5 Control Configuration register

This register configures overall behavior of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R W	Control Enabler	Callback Enabler	Idle Enabler	Idle Scan Rate				
Reset:	0	0	0	0	0	0	0	0

Figure 3-14. Control Configuration register

Table 3-15. Control Configuration register description

Signal	Description
Control Enabler	Global enabler for the control. This determines if the electrodes of the control are scanned for detection. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function will be called when one of the active events in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled
Idle Enabler	Enables the possibility of the control to enter an Idle state when none of its electrodes are active. When enabled, the control electrodes are scanned for detection at the Idle Scan Rate, instead of each task execution. If disabled while in Idle state, the control immediately exits this state. 1 — Idle state enabled 0 — Idle state disabled
Idle Scan Rate	Determines the rate at which the electrodes of the control are scanned for detection while no touch is detected within the control. Once a touch is detected, this parameter has no effect. 1–31 — Electrodes are scanned for every *n* executions of the touch sensing task.

3.5.6 Buffer Pointer register

This register defines the base address of the event buffer for a specific control. The event buffer is a circular buffer with 16 elements. It stores every touch or release event that occurs in the control, this depends on the events enabled in the Events Register.

The Keypad Events Register has the following 8-bit format:

Register Number = 0x02

	7	6	5	4	3	2	1	0
R	Event Type				Key Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

1 Assigned at TSS_Init

Figure 3-15. Buffer Pointer register

Table 3-16. Buffer Pointer register description

Signal	Description
Event Type	Indicates the type of event registered in the buffer 1 — Release event 0 – Touch event
Key Number	Determines the key within keypad control on which the event has occurred 0–15 — Key presenting the event

3.5.7 BufferReadIndex

This register stores the index of the first unread element of the events buffer. This index is used to start reading the data from the buffer until it reaches the value of the Buffer Write Index. The user is responsible for updating this value after Read operation, which can be automatically done with the TSS_KEYPAD_BUFFER_READ macro.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R W					Index value			
Reset:	0	0	0	0	0	0	0	0

Figure 3-16. BufferReadIndex register

Table 3-17. BufferReadIndex Register description

Signal	Description
Index value	Determines the index of the first unread event in the events buffer 0–15 — Index of first unread element

Macro prototype

```
#define TSS_KEYPAD_BUFFER_READ(destvar,kpcsStruct)
```

This macro allows you to store the first unread element from the event buffer and update the value of the Buffer Write Index register. When using this macro provide two parameters; destvar, and kpcsStruct.

Input parameters

Type	Name	Description
UINT8	destvar	Name of the variable where the first unread element is stored.
CSKeypad	kpcsStruct	Name of the controller structure defined in the TSS_SystemSetup.h header file. Refer to Section 2.1.31, “Control configuration and status structure” for more details on the structures name

Return value

None

3.5.8 BufferWriteIndex

This register stores the index value of the first free element in the buffer that is located one element after the last event captured. This register is automatically updated when new events are stored in the buffer, and is provided for reference during the read operations. The user should read the buffer until this index is reached. Comparison made by the TSS_KEYPAD_BUFFER_EMPTY macro is explained later in this document.

Register Number = 0x04

	7	6	5	4	3	2	1	0
R					Index value			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-17. BufferWriteIndex register

Table 3-18. BufferWriteIndex register description

Signal	Description
Index value	Determines the index of the first free element in the events buffer. 0–15 — Index of the first free element

Macro prototype

```
#define TSS_KEYPAD_BUFFER_EMPTY(kpcsStruct)
```

This macro enables you to know when the events buffer is empty. The macro performs a comparison between the Buffer Read Index and the Buffer Write Index.

Input parameters

Type	Name	Description
CSKeypad	kpcsStruct	Name of the controller structure defined by the user in the TSS_SystemSetup.h header file. Refer to Section 2.1.31, “Control configuration and status structure” for more details on the structures name

Return value

The macro performs a comparison between the first unread element index in the buffer and the first free element index. It means that all the elements in the buffer have been read when the two indexes are equal. If all the elements have been read, it returns 1, if not, it returns 0.

3.5.9 Event Control and Status register

This register contains bits used to enable or disable events that call the control callback function. It also contains the maximun keys and buffer overflow status flags.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R W	Max Keys Flag	Buffer Overflow Flag		Keys Exceeded Enabler	Buffer Full/Overflow Enabler	Auto-Repeat Enabler	Release Event Enabler	Touch Event Enabler
Reset:	0	0	0	0	0	0	0	0

Figure 3-18. Event Control and Status register

Table 3-19. Event Control and Status register description

Signal	Description
Max Keys Fault	Flag that indicates if the limit of keys pressed at the same time has been exceeded. The limit is defined by the MaxTouches Register. In this case, any further key touches will be ignored until one key is at least released. This flag will be automatically cleared after the fault condition turns false. 1 — Limit exceeded 0 — No excess keys
Buffer Overflow Flag	This flag indicates if more events have been produced than the buffer's free space. In this case the newest events are lost and no more events can be logged. This flag is cleared by writing a 0 to this bit. 1 — Buffer overflowed 0 — Free space available
Keys Exceeded Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function after the limit for pressed keys is exceeded. 1 — Event enabled 0 — Event disabled
Buffer Overflow Enabler	If this bit is set and the callback function enabled, the decoder calls the control callback function after the circular buffer for events is full or has been overflown. 1 — Event enabled 0 — Event disabled
Auto-repeat Enabler	If this bit is set, the decoder stores a new touch event in the events buffer at the specified rate in case one key remains at least pressed for the time configured in the Auto-repeat Start register. If the control's callback function is enabled, it is called at the same rate. 1 — Auto-repeat enabled 0 — Auto-repeat disabled
Release Event Enabler	If this bit is set, the decoder will store released events in the events buffer. If the control's callback function is enabled, it will be called if an event occurs. 1 — Event enabled 0 — Event disabled
Touch Event Enabler	If this bit is set, the decoder stores touch events in the events buffer. If the control's callback function is enabled, it will be called if an event occurs. 1 — Event enabled 0 — Event disabled

NOTE

If touch and release events are both disabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.5.10 MaxTouches register

The MaxTouches register defines the maximum number of keys reported per touch. This feature can be used to control the function when bigger area than expected is touched. For example, a multiplexed array of electrodes where only two keys are needed to detect a value or function. In this case, the MaxTouches register is configured with a value of two.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	0	0	0	0	Max Number of Touches			
Reset:	0	0	0	0	0	0	0	0

Figure 3-19. MaxTouches register

Table 3-20. MaxTouches register description

Signal	Description
Max Number of Touches	Sets the limit for simultaneous active keys. If set to 0, no limit is established. 0 — No limit established 1–15 — Limit set to *n* keys

3.5.11 Auto Repeat Rate register

This register contains the rate value at which the pressed keys are reported when kept pressed.

Register Number = 0x07

	7	6	5	4	3	2	1	0
R W	Auto-repeat rate							
Reset:	0	0	0	0	0	0	0	0

Figure 3-20. Auto Repeat Rate register

Figure 3-21. Auto Repeat Rate register description

Signal	Description
Auto-repeat Rate	Sets the rate where the pressed keys are reported when kept pressed. This value is a multiplier of the touch sensing task execution rate. If set to 0, no auto-repeat occurs and the event will be automatically disabled in the Events Register. The user must manually re-enable this event if desired. 0 — Auto-repeat feature disabled 1–255 — Touch event reported every *n* execution times

3.5.12 Auto Repeat Start register

This register defines the time difference between a touch event and its auto-repeat, that is considering the key remains touched. During this time, no event will be generated. After this time elapses, a new event will be generated and the auto-repeat rate register will control the new events timing, generating new touch events at the specified rate. After this, the value of this register is not used by the decoder until a different touch event is detected.

Register Number = 0x08

	7	6	5	4	3	2	1	0
R W	Auto-repeat Start							
Reset:	0	0	0	0	0	0	0	0

Figure 3-22. Auto Repeat Start register

Table 3-21. Auto Repeat Start register description

Signal	Description
Auto-repeat Start	Sets the time a key must remain pressed before generating new events at the auto-repeat rate. This value is a multiplier of the touch sensing task execution rate. If set to 0, the auto-repeat state will be entered immediately after the last touch. 0–255 — Wait *n* execution times before auto-repeat.

3.5.13 Keypad Callback function

This function is called by the decoder module if an event occurs and the callback function is enabled. Callback functions are assigned to controls in the system setup module, and one callback may be assigned to different controls in the system.

Function prototype

```
void CallbackFuncName(UINT8 u8ControlId)
```

CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK      fCallBack1
```

Where:

- *n* is the number of the control
- fCallBack1 is a name example

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8ControlId	Any valid control ID of any control in the system	Indicates to the user the control that generated the event. This parameter matches the controlled field in the control C&S register.

Return Value

None

3.6 Slider and Rotary decoder API

This section describes the API for slider and also rotary decoder. The rotary decoder API is equal to slider so the section describes only the slider decoder with comments about differences. The

TSS_SetSliderConfig() and TSS_SetRotaryConfig() function are used to write to the decoder registers, as well as the TSS_GetSliderConfig() and TSS_GetRotaryConfig() are used for reading the registers. This section also describes the callback function and its parameters for a slider and rotary controls. Each application slider and rotary control has its respective Configuration and Status registers as well as callback function.

3.6.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, the TSS_SetSystemConfig function should be called. This is declared in the TSS_API.h file.

Function prototype for Slider

```
UINT8 TSS_SetSliderConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter, UINT8 u8Value);
```

Function prototype for Rotary

```
UINT8 TSS_SetRotaryConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter, UINT8 u8Value);
```

Input parameters

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control’s C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by slider decoder	Code indicating the parameter configured
UINT8	u8Value	Depends on the specific parameter configured	New desired value, for the respective configuration registers

Return value for Slider

The return value is an unsigned integer with the following possible return values defined in the file TSS_API.h:

Return Value	Description
TSS_STATUS_OK	Configuration was done successfully.
TSS_ERROR_ SLIDER _ILLEGAL_PARAMETER	Configuration was not done due to an illegal parameter number.
TSS_ERROR_ SLIDER _READ_ONLY_PARAMETER	Configuration was not done as the user attempts to modify a read-only parameter.
TSS_ERROR_ SLIDER _OUT_OF_RANGE	Configuration was not done because the new value was out of the established boundaries.
TSS_ERROR_ SLIDER _ILLEGAL_CONTROL_TYPE	Configuration was not done because the u8ControlId parameter did not match the control type structure the user was trying to modify.

Return value for Rotary

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration was done successfully.
TSS_ERROR_ROTARY_ILLEGAL_PARAMETER	Configuration was not done due to illegal parameter number.
TSS_ERROR_ROTARY_READ_ONLY_PARAMETER	Configuration was not done as the user attempts to modify a read-only parameter.
TSS_ERROR_ROTARY_OUT_OF_RANGE	Configuration was not done because the new value was out of the established boundaries
TSS_ERROR_ROTARY_ILLEGAL_CONTROL_TYPE	Configuration was not done because the u8ControlId parameter did not match the control type structure the user was trying to modify.

3.6.2 Reading the Configuration and Status registers

There are two options how to access Configuration and Status register structure. The first option uses TSS function declared in the TSS_API.h file.

Function prototype for Slider

```
UINT8 TSS_GetSliderConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter);
```

Function prototype for Rotary

```
UINT8 TSS_GetRotaryConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter);
```

Input parameters

Return value

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control's C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds to actual unsigned byte value of control register specified by u8ControlId and u8Parameter.

The second option is to directly access the configuration and status structure for the specific data. The structure name is defined by the application in the TSS_SystemSetup.h file using the following definition:

```
#define TSS_Cn_STRUCTURE      cStructure
```

Where *n* is the Control number starting from 0 going up to the number of controls minus one.

The *cStructure* name is just an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                                 //Note 1
        const TSS_SLIDER_CONTROL ControlConfig;                         //Note 2
```

```
         const TSS_SLIDER_DYN DynamicStatus;                       //Note 3
         const TSS_SLIDER_STAT StaticStatus;                       //Note 4
         const TSS_SLIDER_EVENTS Events;                           //Note 5
         const UINT8 AutoRepeatRate;
         const UINT8 MovementTimeout;
} TSS_CSSlider;

typedef struct{
         const TSS_CONTROL_ID ControlId;                           //Note 1
         const TSS_SLIDER_CONTROL ControlConfig;                   //Note 2
         const TSS_SLIDER_DYN DynamicStatus;                       //Note 3
         const TSS_SLIDER_STAT StaticStatus;                       //Note 4
         const TSS_SLIDER_EVENTS Events;                           //Note 5
         const UINT8 AutoRepeatRate;
         const UINT8 MovementTimeout;
} TSS_CSRotary;
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.6.4, “Control ID register.”
2. The TSS_SLIDER_CONTROL type is an eight bit-field structure described in Section 3.6.5, “Control configuration.”
3. The TSS_SLIDER_DYN type is an eight bit-field structure described in Section 3.6.6, “Dynamic Status register.”
4. The TSS_SLIDER_STAT type is an eight bit-field structure described in Section 3.6.7, “Static Status register.”
5. The SLIDER_EVENTS type is an eight bit-field structure described in Section 3.6.8, “Events Control register.”

Using the example of the structure named *cStructure*, to read Control ID register use:

```
Temp = cStructure.ControlId;
```

3.6.3 Configuration and Status registers list

The table below contains the control Configuration and Status registers.

Table 3-22. Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.6.4, “Control ID register”	Application dependable	R — Displays the control type and control number
0x01	1	ControlConfig	Section 3.6.5, “Control configuration”	0x00	RW — This register configures overall enablers of the object
0x02	1	DynamicStatus	Section 3.6.6, “Dynamic Status register”	0x00	R — Displays the movement, direction and displacement information

Table 3-22. Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x03	1	StaticStatus	Section 3.6.7, "Static Status register"	0x00	R — Displays the touch and absolute position information. Hold the invalid position status flag.
0x04	1	Events	Section 3.6.8, "Events Control register"	0x00	RW — Configures the events that will call the callback function.
0x05	1	AutoRepeatRate	Section 3.6.9, "Auto-repeat Rate register"	0x00	RW — Configures the rate at which keys will be reported when they are kept pressed and no movement is detected.
0x06	1	MovementTimeout	Section 3.6.10, "Movement Timeout register"	0x00	RW — Number of times the decoder must detect a no displacement before reporting a hold event and no movement.

3.6.4 Control ID register

This read-only register contains the control's identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-23. Control ID register

Encoding	Control Type
001	Keypad
010	Slider
011	Rotary
100	Analog Slider
101	Analog Rotary
110	Matrix
111	Reserved

NOTE

The control number is the one assigned in the system setup module. This value is unique for all controls, implying each control has a particular number regardless of its type. The first assigned control number is zero.

3.6.5 Control configuration

This register configures overall enablers of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R W	Control Enabler	Callback Enabler	Idle Enabler	Idle Scan Rate				
Reset:	0	0	0	0	0	0	0	0

Figure 3-24. Control Configuration register

Table 3-23. Control Configuration register description

Signal	Description
Control Enabler	Global enabler for the control. This determines if the electrodes of the control are scanned for detection. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function will be called when one of the active events in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled
Idle Enabler	Enables the possibility of the control to enter an Idle state when none of its electrodes are active. When enabled, the control electrodes will be scanned for detection at the Idle Scan Rate, instead of each task execution. If disabled while in Idle state, the control will immediately exit this state. 1 — Idle state enabled 0 — Idle state disabled
Idle Scan Rate	Determines the rate where the electrodes of the control will be scanned for detection while no touch is detected within the control. Once a touch is detected, this parameter has no effect. 1–31 — Electrodes will be scanned every *n* executions of the touch sensing task.

3.6.6 Dynamic Status register

This register contains the information about the movement over the electrodes assigned to the control.

Register Number = 0x02

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	0	0	Displacement			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-25. Dynamic Status register

Table 3-24. Dynamic Status register description

Signal	Description
Movement Flag	Indicates if movement is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of movement. This bit remains with its last value even if movement has stop and is no longer detected. 1 — Incremental (from Ex to Ey, where x < y) 0 — Decremental (from Ex to Ey, where x > y)
Displacement	This value indicates the difference in positions from the last status to the new one. This indicates how many positions have been advanced in the current direction of movement. 0–15 — Number of positions.

3.6.7 Static Status register

This register displays the events, flags, and status of the electrodes in the control when no movement over the electrodes is detected.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R	Touch Flag	Invalid Position Flag	0	Position				
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-26. Static Status register

Table 3-25. Static Status register description

Signal	Description
Touch Flag	Indicates if a touch in the control is being detected at the moment of reading. 1 — Touch detected 0 — Touch not detected

Signal	Description
Invalid Position Flag	Indicates if an invalid combination of touched electrodes is being detected. 1 — Invalid position detected 0 — Valid position detected
Position	This value indicates the absolute position within the control that is actuated. 0–31 — Absolute position

3.6.8 Events Control register

The register contains the bits used to enable or disable events that will call the control's callback function.

Register Number = 0x04

	7	6	5	4	3	2	1	0
R W				Release Event Enabler	Hold Auto-Repeat Enabler	Hold Event Enabler	Movement Event Enabler	Initial Touch Event Enabler
Reset:	0	0	0	0	0	0	0	0

Figure 3-27. Events Control register

Table 3-26. Events Control Register description

Signal	Description
Release Event Enabler	If this bit is set and the callback function is enabled, the callback will be called when all the touched electrodes in the control are released. 1 — Release event enabled 0 — Release event disabled
Hold Auto-repeat Enabler	If this bit is set, the Hold Event Enabler and the callback function are enabled. The callback is called at the rate specified in the Auto-repeat Rate register for as long as one valid position is detected as touched in the control, and no movement is detected. 1 — Auto-repeat feature enabled 0 — Auto-repeat disabled
Hold Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control's callback function when the movement stops and a constant touched position is detected after a certain period of time. This time is configurable in the Movement Timeout register. 1 — Event enabled 0 — Event disabled
Movement Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control's callback function every time a displacement is detected. 1 — Event enabled 0 — Event disabled
Initial Touch Event Enabler	If this bit is set and the callback function enabled, the decoder calls the control's callback function when the control transitions from an idle to an active state. 1 — Event enabled 0 — Event disabled

NOTE

If none of the events are enabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.6.9 Auto-repeat Rate register

The Auto-repeat Rate register contains the rate value where a hold event is reported when kept pressed without movement.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R W	Auto-repeat Rate							
Reset:	0	0	0	0	0	0	0	0

Figure 3-28. Auto-repeat Rate register

Table 3-27. Auto-repeat Rate register description

Signal	Description
Auto-repeat Rate	The decoder will call the control’s callback function at the specified rate as long as a valid position is continuously detected, the auto-repeat feature is enabled and no displacement occurs. If this register is set to 0, the auto-repeat feature will be disabled in the events register. 0 — Auto-repeat feature disabled 1–254 — Callback is called every *n* task executions

NOTE

The Auto-repeat function works only if hold event is presented. The hold event is delayed by the Movement Timeout function. If the Movement Timeout register is set to 0, the Auto-repeat function stops to generate callback, because the Hold event is no more reported.

3.6.10 Movement Timeout register

The register value defines how long the movement event stays reported after hold event is detected. The delay is defined in number of `TSS_Task()` executions. The hold event represents the situation when the movement stops and a constant touched position is detected.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Movement timeout							
Reset:	0	0	0	0	0	0	0	0

Figure 3-29. Movement Timeout register

Table 3-28. Movement Timeout register description

Signal	Description
Movement Timeout	Determines how many execution times the decoder must detect no displacement before reporting no movement and reporting a hold event. 0 — No limit established. Movement is reported until the control is detected as idle 1–254 — Report a hold event after *n* task executions

3.6.11 Callback function

The decoder module calls this function if an event occurs and the user did enable the callback function. Callback functions are assigned to controls in the system setup module, and one callback may be assigned to different controls in the system.

Function prototype

```
void CallbackFuncName(UINT8 u8ControlId)
```

The CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK      fCallBack3  /* Identifier of the user's callback */
```

Where:

- *n* is the number of the control
- *fCallBack3* is a name example

Input parameters

Type	Name	Valid Range/Values	Description
UINT8	u8ControlId	Any valid Control ID of any control in the system.	Indicates to the user the control that generated the event. This parameter matches the controlled field in the control C&S structure.

Return value

None

3.7 Analog slider and analog rotary decoder API

This section describes the API for the analog slider and also the analog rotary decoder. The analog rotary decoder API is equal to the analog slider so the section describes only the analog slider decoder with comments about differences. The TSS_SetASliderConfig() and TSS_ASetRotaryConfig() function are used to write to the decoder registers. And the TSS_GetASliderConfig() and TSS_GetARotaryConfig() are used for reading the registers. This section also describes the callback function and its parameters for an analog slider and analog rotary control. Each application analog slider and analog rotary control has its respective Configuration and Status registers as well as callback function.

3.7.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, the TSS_SetSystemConfig function should be called. This function is declared in the TSS_API.h file.

Function prototype for Analog Slider

```
UINT8 TSS_SetASliderConfig (TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter, UINT8 u8Value)
```

Function prototype for Analog Rotary

```
UINT8 TSS_SetARotaryConfig (TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter, UINT8 u8Value)
```

Input parameters

Type	Name	Valid range and values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control's C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by the decoder	Code indicating the parameter configured
UINT8	u8Value	Depends on the specific parameter configured	New desired value, for the respective configuration registers

Return value for Analog Slider

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration was executed successfully.
TSS_ERROR_ASLIDER_ILLEGAL_PARAMETER	Configuration was not executed due to illegal parameter number.
TSS_ERROR_ASLIDER_ILLEGAL_VALUE	Configuration was not executed due to illegal value
TSS_ERROR_ASLIDER_READ_ONLY_PARAMETER	Configuration was not done as the user attempts to modify a read-only parameter.
TSS_ERROR_ASLIDER_OUT_OF_RANGE	Configuration was not executed because the new value was out of the established boundaries
TSS_ERROR_ASLIDER_ILLEGAL_CONTROL_TYPE	Configuration was not executed because the u8ControlId parameter did not match the control type structure the user was trying to modify.

Return value for Analog Rotary

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration was executed successfully.
TSS_ERROR_AROTARY_ILLEGAL_PARAMETER	Configuration was not executed due to illegal parameter number.
TSS_ERROR_AROTARY_ILLEGAL_VALUE	Configuration was not executed due to illegal value
TSS_ERROR_AROTARY_READ_ONLY_PARAMETER	Configuration was not executed as the user attempts to modify a read-only parameter.
TSS_ERROR_AROTARY_OUT_OF_RANGE	Configuration was not executed because the new value was out of the established boundaries
TSS_ERROR_AROTARY_ILLEGAL_CONTROL_TYPE	Configuration was not executed because the u8ControlId parameter did not match the control type structure the user was trying to modify.

3.7.2 Reading the Configuration and Status registers

There are two options on how to access rotary Configuration and Status register structure. The first option uses the TSS function declared in the TSS_API.h file.

Function prototype for Analog Slider

```
UINT8 TSS_GetASliderConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter);
```

Function prototype for Analog Rotary

```
UINT8 TSS_GetARotaryConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter);
```

Input parameters

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control's C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds to an actual unsigned byte value for the control register specified by u8ControlId and u8Parameter.

The second option enables you to directly access the configuration and status structure for specific data. The structure name is defined by the application in the TSS_SystemSetup.h file using the following define:

```
#define TSS_Cn_STRUCTURE      cStructure
```

Where *n* is the control number starting from 0 going up to the number of controls minus one.

The *cStructure* name is just an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_ASLIDER_CONTROL ControlConfig;                 //Note 2
        const TSS_ASLIDER_DYN DynamicStatus;                     //Note 3
        const UINT8 Position;
        const TSS_ASLIDER_EVENTS Events;                         //Note 4
        const UINT8 AutoRepeatRate;
        const UINT8 MovementTimeout;
        const UINT8 Range;
} TSS_CSASlider;

typedef struct{
        const TSS_CONTROL_ID ControlId;                          //Note 1
        const TSS_ASLIDER_CONTROL ControlConfig;                 //Note 2
        const TSS_ASLIDER_DYN DynamicStatus;                     //Note 3
        const UINT8 Position;
        const TSS_ASLIDER_EVENTS Events;                         //Note 4
        const UINT8 AutoRepeatRate;
        const UINT8 MovementTimeout;
        const UINT8 Range;
  } TSS_CSARotary;
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.7.4, "Control ID register."
2. The TSS_ASLIDER_CONTROL type is an eight bit-field structure described in Section 3.7.5, "Control configuration."

3. The TSS_ASLIDER_DYN type is an eight bit-field structure described in Section 3.7.6, “Dynamic Status register.”
4. The TSS_ASLIDER_EVENTS type is an eight bit-field structure described in Section 3.7.8, “Events Control register.”

Using the example of the structure named *cStructure*, to read Control ID use:

```
Temp = cStructure.ControlId;
```

3.7.3 Configuration and Status registers list

The table below shows the control Configuration and Status registers.

Table 3-29. Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.7.4, “Control ID register”	Application dependable	R — Displays the control type and control number
0x01	1	ControlConfig	Section 3.7.5, “Control configuration”	0x00	RW — This register configures overall enablers of the object
0x02	1	DynamicStatus	Section 3.7.6, “Dynamic Status register”	0x00	R — Displays the movement, direction and displacement information
0x03	1	Position	Section 3.7.7, “Position register”	0x00	R — Displays absolute position information within a defined range.
0x04	1	Events	Section 3.7.8, “Events Control register”	0x00	RW — Configures the events that call the callback function.
0x05	1	AutoRepeatRate	Section 3.7.9, “Auto-repeat Rate register”	0x00	RW — Configures the rate where keys are reported when they are kept pressed and no movement is detected.
0x06	1	MovementTimeout	Section 3.7.10, “Movement Timeout register”	0x00	RW — Number of times the decoder must detect a no displacement before reporting a hold event and no movement.
0x07	1	Range	Section 3.7.11, “Range register”	0x40	RW - Defines the absolute range within the position is reported.

3.7.4 Control ID register

This read-only register contains the control's identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

[1] Application dependable.

Figure 3-30. Control ID register

Encoding	**Control Type**
001	Keypad
010	Slider
011	Rotary
100	Analog slider
101	Analog rotary
110	Matrix
111	Reserved

NOTE

The control number is the one assigned in the system setup module. This value is unique for all controls, implying each control has a particular number regardless of its type. The first assigned control number is zero.

3.7.5 Control configuration

This register configures overall enablers of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R / W	Control Enabler	Callback Enabler	Idle Enabler	Idle Scan Rate				
Reset:	0	0	0	0	0	0	0	0

Figure 3-31. Control Configuration register

Table 3-30. Control configuration description

Signal	Description
Control Enabler	Global enabler for the control. This determines if the electrodes of the control are scanned for detection. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function is called when one of the active events in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled
Idle Enabler	Enables the possibility of the control to enter an Idle state when none of its electrodes are active. When enabled, the control electrodes will be scanned for detection at the Idle Scan Rate, instead of each task execution. If disabled while in Idle state, the control will immediately exit this state. 1 — Idle state enabled 0 — Idle state disabled
Idle Scan Rate	This value determines the rate at which the electrodes of the control will be scanned for detection while no touch is detected within the control. Once a touch is detected, this parameter has no effect. 1–31 — Electrodes will be scanned every n executions of the touch sensing task.

3.7.6 Dynamic Status register

This register contains the information regarding the movement over the electrodes assigned to the controller.

Register Number = 0x02

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	Displacement					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-32. Dynamic Status register

Table 3-31. Dynamic Status register description

Signal	Description
Movement	Indicates if movement is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of movement. This bit remains with its last value even if movement has stopped and is no longer detected. 1 — Incremental (from PositionX to PositionY, where X< Y) 0 — Decremental (from PositionX to PositionY, where X> Y)
Displacement	This value indicates the difference in positions from the last status to the new one. This indicates how many positions have been advanced in the current direction of movement. 0–63 — Number of positions.

3.7.7 Position register

This Position register contains a absolute position within defined range.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R	Position							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-33. Static Status register

Table 3-32. Position register description

Signal	Description
Position	Indicates the calculated absolute position within defined range. 0—*Range* - Number of the actual position

3.7.8 Events Control register

The Event Control register contains the bits used to enable or disable events that will call the control's callback function.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R W	Touch	Invalid Position		Release Event Enabler	Hold Auto-Repeat Enabler	Hold Event Enabler	Movement Event Enabler	Initial Touch Event Enabler
Reset:	0	0	0	0	0	0	0	0

Figure 3-34. Events Control register

Table 3-33. Events Control register description

Signal	Description
Touch	This bit indicates if the touch has been detected on the control. 1 — Touch detected 0 — Touch not detected
Invalid Position	This bit indicates if the invalid position has been detected on the control. 1 — Invalid position detected 0 — Invalid position not detected
Release Event Enabler	If this bit is set and the callback function is enabled, the callback will be called when all the touched electrodes in the control are released. 1 — Release event enabled 0 — Release event disabled

Table 3-33. Events Control register description

Signal	Description
Hold Auto-repeat Enabler	If this bit is set, the hold event enabler and the callback function are enabled, the callback will be called at the rate specified in the Auto-repeat Rate register for as long as one valid position is detected as touched in the control and no movement is detected. 1 — Auto-repeat feature enabled 0 — Auto-repeat disabled
Hold Event Enabler	If this bit is set and the callback function enabled, the decoder will call the control’s callback function when the movement stops and a constant touched position is detected after a certain period of time. This time is configurable in the Movement Timeout Register. 1 — Event enabled 0 — Event disabled
Movement Event Enabler	If this bit is set and the callback function enabled, the decoder will call the control’s callback function every time a displacement is detected. 1 — Event enabled 0 — Event disabled
Initial Touch Event Enabler	If this bit is set and the callback function enabled, the decoder will call the control’s callback function when the control transitions from an idle to an active state. 1 — Event enabled 0 — Event disabled

NOTE

If no of the events is enabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.7.9 Auto-repeat Rate register

The Auto-repeat Rate register contains the rate value where hold event is reported when kept pressed without movement.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R W	Auto-repeat start							
Reset:	0	0	0	0	0	0	0	0

Figure 3-35. Auto-repeat Rate register

Table 3-34. Auto-repeat Rate register description

Signal	Description
Auto-repeat Rate	The decoder calls the control’s callback function at the specified rate as long as a valid position is continuously detected, the auto-repeat feature is enabled and no displacement occurs. If this register is set to 0, the auto-repeat feature will be disabled in the events register. 0 — Auto-repeat feature disabled 1–254 — Callback is called every *n* task executions

NOTE

The Auto-repeat function works only if hold event is presented. The hold event is delayed by the Movement Timeout function. If the Movement Timeout register is set to 0, the Auto-repeat function stops to generate callback, because the Hold event is no more reported.

3.7.10 Movement Timeout register

The register value defines how long the movement event stays reported after hold event is detected. The delay is defined in number of `TSS_Task()` executions. The hold event represents the situation when the movement stops and a constant touched position is detected.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Movement timeout							
Reset:	0	0	0	0	0	0	0	0

Figure 3-36. Movement Timeout register

Table 3-35. Movement Timeout register description

Signal	Description
Movement Timeout	Determines how many execution times the decoder must detect no displacement before reporting no movement and reporting a hold event. 0 — No limit established. Movement is reported until the control is detected as idle 1–254 — Report a hold event after n task executions

3.7.11 Range register

The range register value defines the absolute range within the position is reported.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Range							
Reset:	0	1	0	0	0	0	0	0

Figure 3-37. Range register

Table 3-36. Range register description

Signal	Description
Range	Defines the absolute range within the position is reported. 2— 255 for Analog slider 3— 255 for Analog rotary

3.7.12 Callback function

The decoder module calls this function if an event occurs and if the user enables the callback function. Callback functions are assigned to controls in the system setup module, and one callback may be assigned to different controls in the system.

Function prototype

```
void CallbackFuncName(UINT8 u8ControlId)
```

The CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK      fCallBack4
```

Where:

- *n* is the number of the control
- *fCallBack4* is a name example

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8ControlId	Any valid control ID of any control in the system	Indicates to the user the control that generated the event. This parameter matches the controlled field in the control C&S register.

Return value

None

3.8 Matrix decoder API

This section describes the Matrix Decoder API. It describes the Matrix C&S registers, the TSS_SetMatrixConfig() function used to write to these registers, and the TSS_GetMatrixConfig() function and data structure used for reading the register. This section also describes the callback function and its parameters for a Matrix control. Each Application Matrix control has its respective Configuration and Status registers and a callback function.

3.8.1 Writing to the Configuration and Status registers

To change values in the Configuration and Status register, the TSS_SetSystemConfig function should be called. This function is declared in the TSS_API.h file.

Function prototype

```
UINT8 TSS_SetMatrixConfig (TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter, UINT8 u8Value)
```

Input parameters

Type	Name	Valid range and values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control’s C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by matrix decoder	Code indicating the parameter configured
UINT8	u8Value	Depends on the specific parameter configured	New desired value, for the respective configuration registers

Return value

The return value is an unsigned integer with the following possible return values defined in the file TSS_StatusCodes.h.

Return Value	Description
TSS_STATUS_OK	Configuration was executed successfully.
TSS_ERROR_MATRIX_ILLEGAL_PARAMETER	Configuration was not executed due to illegal parameter number.
TSS_ERROR_MATRIX_ILLEGAL_VALUE	Configuration was not executed due to illegal value
TSS_ERROR_MATRIX_READ_ONLY_PARAMETER	Configuration was not executed as the user attempts to modify a read-only parameter.
TSS_ERROR_MATRIX_OUT_OF_RANGE	Configuration was not executed because the new value was out of the established boundaries
TSS_ERROR_MATRIX_ILLEGAL_CONTROL_TYPE	Configuration was not executed because the u8ControlId parameter did not match the control type structure the user was trying to modify.

3.8.2 Reading the Configuration and Status registers

There are two options how to access Configuration and Status register structure. The first option uses TSS function declared in the TSS_API.h file.

Function prototype

```
UINT8 TSS_GetMatrixConfig(TSS_CONTROL_ID u8ControlId, UINT8 u8Parameter);
```

Input parameters

Return value

Type	Name	Valid range/values	Description
TSS_CONTROL_ID	u8ControlId	Any valid control Id of the appropriate control type	Identifier of the control configured, stored in the first element of the control’s C&S structure
UINT8	u8Parameter	Any of the parameter codes provided by each decoder	Code indicating the parameter configured

Return value

The return value corresponds to actual unsigned byte value of register specified by u8ControlId and u8Parameter.

The second option enables you to directly access the configuration and status structure for specific data. The structure name is defined by the application in the TSS_SystemSetup.h file using the following define:

```
#define TSS_Cn_STRUCTURE      cMatrix
```

Where *n* is the control number starting from 0 going up to the number of controls minus one.

The *cMatrix* name is just an example. The user can define any other name for each control configuration and status structure.

```
typedef struct{
        const TSS_CONTROL_ID ControlId;                               //Note 1
        const TSS_MATRIX_CONTROL ControlConfig;                       //Note 2
        const TSS_MATRIX_EVENTS Events;                               //Note 3
        const UINT8 AutoRepeatRate;
        const UINT8 MovementTimeout;
        const TSS_MATRIX_DYN DynamicStatusX;                          //Note 4
        const TSS_MATRIX_DYN DynamicStatusY;
        const UINT8 PositionX;
        const UINT8 PositionY;
        const UINT8 GestureDistanceX;
        const UINT8 GestureDistanceY;
        const UINT8 RangeX;
        const UINT8 RangeY;
  } TSS_CSMatrix
```

NOTE

1. The TSS_CONTROL_ID type is an eight bit-field structure described in Section 3.8.4, “Control ID register.”
2. The TSS_MATRIX_CONTROL type is an eight bit-field structure described in Section 3.8.5, “Control configuration.”
3. The TSS_MATRIX_EVENTS type is an eight bit-field structure described in Section 3.8.6, “Events Control register.”
4. The TSS_MATRIX_DYN type is an eight bit-field structure described in Section 3.8.9, “Dynamic Status X register.”

Using the example of the structure named *cMatrix*, to read Control ID use:

```
Temp = cMatrix.ControlId;
```

3.8.3 Configuration and Status registers list

The table below presents the matrix control Configuration and Status registers.

Table 3-37. Matrix Control Configuration and Status registers

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x00	1	ControlId	Section 3.8.4, “Control ID register”	Application dependable	R — Displays the control type and control number
0x01	1	ControlConfig	Section 3.8.5, “Control configuration”	0x00	RW — This register configures overall enablers of the object
0x02	1	Events	Section 3.8.6, “Events Control register”	0x00	RW — Configures the events that call the callback function.
0x03	1	AutoRepeatRate	Section 3.8.7, “Auto-repeat Rate register”	0x00	RW — Configures the rate where keys are reported when they are kept pressed and no movement is detected.
0x04	1	MovementTimeout	Section 3.8.8, “Movement Timeout register”	0x00	RW — Number of times the decoder must detect a no displacement before reporting a hold event and no movement.
0x05	1	DynamicStatusX	Section 3.8.9, “Dynamic Status X register”	0x00	R — Displays the movement, direction and displacement information on X axis
0x06	1	DynamicStatusY	Section 3.8.10, “Dynamic Status Y register”	0x00	R — Displays the movement, direction and displacement information on Y axis
0x07	1	PositionX	Section 3.8.11, “Position X register”	0x00	R — Displays absolute position information within a defined range on X axis.
0x08	1	PositionY	Section 3.8.12, “Position Y register”	0x00	R — Displays absolute position information within a defined range on Y axis.
0x09	1	GestureDistanceX	Section 3.8.13, “Gesture distance X register”	0x00	R — Displays relative distance between two or more positions within a defined range on X axis.
0x0A	1	GestureDistanceY	Section 3.8.14, “Gesture distance Y register”	0x00	R — Displays relative distance between two or more positions within a defined range on Y axis.

Table 3-37. Matrix Control Configuration and Status registers (continued)

Register Number	Size [bytes]	Register Name	Section	Initial value	Brief Description
0x0B	1	RangeX	Section 3.8.15, “Range X register”	0x40	R — Defines the absolute range on X axis.
0x0C	1	RangeY	Section 3.8.15, “Range X register”	0x40	R — Defines the absolute range on Y axis

3.8.4 Control ID register

This read-only register contains the control identifier code.

Register Number = 0x00

	7	6	5	4	3	2	1	0
R	Control Type				Control Number			
W								
Reset:	—[1]	—	—	—	—	—	—	—

1 Application dependable.

Figure 3-38. Control ID register

Encoding	Control Type
001	Keypad
010	Slider
011	Rotary
100	Analog slider
101	Analog rotary
100	Matrix
111	Reserved

NOTE

The control number is the one assigned in the system setup module. This value is unique for all controls, implying each control has a particular number regardless of its type. The first assigned control number is zero.

3.8.5 Control configuration

This register configures overall enablers of the object.

Register Number = 0x01

	7	6	5	4	3	2	1	0
R W	Control Enabler	Callback Enabler	Idle Enabler	Idle Scan Rate				
Reset:	0	0	0	0	0	0	0	0

Figure 3-39. Control Configuration register

Table 3-38. Control configuration description

Signal	**Description**
Control Enabler	Global enabler for the control. This determines if the electrodes of the control are scanned for detection. 1 — Control enabled 0 — Control disabled
Callback Enabler	Enables or disables the use of the callback function. If it is enabled, the function is called when one of the active events in the Events Configuration registers occurs. 1 — Callback enabled 0 — Callback disabled
Idle Enabler	Enables the possibility of the control to enter an Idle state when none of its electrodes are active. When enabled, the control electrodes will be scanned for detection at the Idle Scan Rate, instead of each task execution. If disabled while in Idle state, the control will immediately exit this state. 1 — Idle state enabled 0 — Idle state disabled
Idle Scan Rate	This value determines the rate at which the electrodes of the control will be scanned for detection while no touch is detected within the control. Once a touch is detected, this parameter has no effect. 1–31 — Electrodes will be scanned every n executions of the touch sensing task.

3.8.6 Events Control register

The Event Control register contains the bits used to enable or disable events that will call the control’s callback function.

Register Number = 0x02

	7	6	5	4	3	2	1	0
R W	Touch	Gesture	Gestures Enabler	Release Event Enabler	Hold Auto-Repeat Enabler	Hold Event Enabler	Movement Event Enabler	Initial Touch Event Enabler
Reset:	0	0	0	0	0	0	0	0

Figure 3-40. Events Control register

Table 3-39. Events Control register description

Signal	Description
Touch	This bit indicates if the touch has been detected on the control. 1 — Touch detected 0 — Touch not detected
Gesture	This bit indicates if the gesture has been detected on the control. The gesture is reported if at least two isolated touches are detected within the control. 1 — Gesture detected 0 — Gesture not detected
Gestures Enabler	If this bit is set, the gestures evaluation is enabled and the information about gesture starts write into the Gesture Distance registers.This option also enables to generate callbacks on all enabled events, but the event source is Gesture Distance register instead of Position register. 1 — Gestures enabled 0 — Release event disabled
Hold Auto-repeat Enabler	If this bit is set, the hold event enabler and the callback function are enabled, the callback will be called at the rate specified in the Auto-repeat Rate register for as long as one valid position is detected as touched in the control and no movement is detected. 1 — Auto-repeat feature enabled 0 — Auto-repeat disabled
Hold Event Enabler	If this bit is set and the callback function enabled, the decoder will call the control’s callback function when the movement stops and a constant touched position is detected after a certain period of time. This time is configurable in the Movement Timeout Register. 1 — Event enabled 0 — Event disabled
Movement Event Enabler	If this bit is set and the callback function enabled, the decoder will call the control’s callback function every time a displacement is detected. 1 — Event enabled 0 — Event disabled
Initial Touch Event Enabler	If this bit is set and the callback function enabled, the decoder will call the control’s callback function when the control transitions from an idle to an active state. 1 — Event enabled 0 — Event disabled

NOTE

If no events are enabled, the control is automatically disabled and must be re-enabled by the user in the Control Configuration Register.

3.8.7 Auto-repeat Rate register

The Auto-repeat Rate register contains the rate value where a hold event is reported when kept pressed without movement.

Register Number = 0x03

	7	6	5	4	3	2	1	0
R W	Auto-repeat start							
Reset:	0	0	0	0	0	0	0	0

Figure 3-41. Auto-repeat Rate register

Table 3-40. Auto-repeat Rate register description

Signal	Description
Auto-repeat Rate	The decoder calls the control's callback function at the specified rate as long as a valid position is continuously detected, the auto-repeat feature is enabled and no displacement occurs. If this register is set to 0, the auto-repeat feature will be disabled in the events register. 0 — Auto-repeat feature disabled 1–254 — Callback is called every *n* task executions

NOTE

The Auto-repeat function works only if hold event is presented. The hold event is delayed by the Movement Timeout function. If the Movement Timeout register is set to 0, the Auto-repeat function stops to generate callback, because the Hold event is no longer reported.

3.8.8 Movement Timeout register

The register value defines how long the movement event stays reported after hold event is detected. The delay is defined in number of `TSS_Task()` executions. The hold event represents the situation when the movement stops and a constant touched position is detected.

Register Number = 0x04

	7	6	5	4	3	2	1	0
R W	Movement timeout							
Reset:	0	0	0	0	0	0	0	0

Figure 3-42. Movement Timeout register

Table 3-41. Movement Timeout register description

Signal	Description
Movement Timeout	Determines how many execution times the decoder must detect no displacement before reporting no movement and reporting a hold event. 0 — No limit established. Movement is reported until the control is detected as idle 1–254 — Report a hold event after n task executions

3.8.9 Dynamic Status X register

This register contains the information regarding the movement on X horizontal axis over the electrodes assigned to the control. The information is based on Position X or Gesture Distance X register information.

Register Number = 0x05

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	Displacement					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-43. Dynamic Status X register

Table 3-42. Dynamic Status X register description

Signal	Description
Movement	Indicates if movement on X axis is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of movement on X axis. This bit remains with its last value even if movement has stopped and is no longer detected. 1 — Incremental (from PositionN to PositionM, where N < M) 0 — Decremental (from PositionN to PisitionM, where N > M)
Displacement	This value indicates the difference in positions on X axis from the last status to the new one. This indicates how many positions have been advanced in the current direction of movement. 0–63 — Number of positions.

3.8.10 Dynamic Status Y register

This register contains the information regarding the movement on Y vertical axis over the electrodes assigned to the control. The information is based on Position Y or Gesture Distance Y register information.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R	Movement Flag	Direction	Displacement					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-44. Dynamic Status Y register

Table 3-43. Dynamic Status Y register description

Signal	Description
Movement	Indicates if movement on Y axis is being detected at the moment of reading. 1 — Movement detected 0 — Movement not detected
Direction	Indicates the direction of movement on Y axis. This bit remains with its last value even if movement has stopped and is no longer detected. 1 — Incremental (from PositionN to PositionM, where N < M) 0 — Decremental (from PositionN to PisitionM, where N > M)
Displacement	This value indicates the difference in positions on Y axis from the last status to the new one. This indicates how many positions have been advanced in the current direction of movement. 0–63 — Number of positions.

3.8.11 Position X register

This Position X register contains absolute analog position within defined range on X horizontal axis.

Register Number = 0x07

	7	6	5	4	3	2	1	0
R	Position X							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-45. Position X register

Table 3-44. Position X register description

Signal	Description
Position	Indicates the calculated absolute analog position within defined range. 0—*Range* - Number of the actual position

3.8.12 Position Y register

This Position Y register contains absolute analog position within defined range on Y vertical axis.

Register Number = 0x08

	7	6	5	4	3	2	1	0
R	Position Y							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 3-46. Position Y register

Table 3-45. Position Y register description

Signal	Description
Position	Indicates the calculated absolute analog position within defined range. 0—*Range* - Number of the actual position

3.8.13 Gesture distance X register

This Gesture Distance X register contains a calculated maximum distance between gesture analog positions on X horizontal axis. The gesture event is reported if at least two isolated touches are detected within control range.

Register Number = 0x09

	7	6	5	4	3	2	1	0
R	Gesture Distance X							
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

Figure 3-47. Gesture Distance X register

Table 3-46. Gesture Distance X register description

Signal	Description
Position	Indicates calculated maximum distance between gesture analog positions on X horizontal axis. 0—*Range* - Number of the actual position

3.8.14 Gesture distance Y register

This Gesture Distance Y register contains a calculated maximum distance between gesture analog positions on Y vertical axis. The gesture event is reported if at least two isolated touches are detected within control range.

Register Number = 0x09

	7	6	5	4	3	2	1	0
R	Gesture Distance Y							
W	▨	▨	▨	▨	▨	▨	▨	▨
Reset:	0	0	0	0	0	0	0	0

Figure 3-48. Gesture Distance Y register

Table 3-47. Gesture Distance Y register description

Signal	Description
Position	Indicates calculated maximum distance between gesture analog positions on Y vertical axis. 0—*Range* - Number of the actual position

3.8.15 Range X register

The range register value defines the absolute range within the position on X horizontal axis and is reported.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Range X							
Reset:	0	1	0	0	0	0	0	0

Figure 3-49. Range X register

Table 3-48. Range X register description

Signal	Description
Range	Defines the absolute range within the position on X horizontal axis reported. 2— 255 - Range value

3.8.16 Range Y register

The range register value defines the absolute range within the position on Y vertical axis and is reported.

Register Number = 0x06

	7	6	5	4	3	2	1	0
R W	Range Y							
Reset:	0	1	0	0	0	0	0	0

Figure 3-50. Range Y register

Table 3-49. Range Y register description

Signal	Description
Range	Defines the absolute range within the position on Y vertical axis is reported. 2— 255 - Range value

3.8.17 Matrix callback function

The decoder module calls this function if an event occurs and the user enables the callback function. Callback functions are assigned to controls in the system setup module, and one callback may be assigned to different controls in the system.

Function prototype

```
void CallbackFuncName(UINT8 u8ControlId)
```

The CallbackFuncName for each control is defined in the following TSS_SystemSetup.h file macro:

```
#define TSS_Cn_CALLBACK      fCallBack5
```

Where:

- *n* is the number of the control
- *fCallBack5* is a name example

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8ControlId	Any valid control ID of any control in the system	Indicates to the user the control that generated the event. This parameter matches the controlled field in the control C&S register.

Return value

None

Chapter 4
Library Intermediate Layer Interfaces

The TSS library architecture allows you to implement your modules depending on your application requirements. You can customize your applications using a suitable library layer. To implement modules, you need to understand how modules exchange information and communicate with each other. This section describes the parameters and functions used by the library to establish communication between layers. For more details on the TSS library architecture, refer to Section 1.3, “Library architecture.”

4.1 Capacitive sensing and key detector interface

The capacitive sensing layer senses each electrode declared in the system. Based on the information provided by the capacitive sensing layer, the key detector layer determines if an electrode is being pressed or not. Table 4-1 shows the global value where the charging time is stored.

Table 4-1. Global variable description

Parameter	Description
tss_u16CapSample	Global 16 bits variable used to store the electrode charging time

The tss_u16CapSample global variable passes the acquired sensing data from the capacitive sensing layer to the key detector layer. When the capacitive sensing layer senses an electrode, it stores the charging time information into the tss_u16CapSample global variable. If you want to use your own capacitive sensing module, the acquired value from the sensing module is passed to the upper layer by storing this value into the tss_u16CapSample global variable.

4.1.1 Electrode sampling

The exchange of information between the capacitive sensing and key detector layers is performed using the sensing function.

Function prototype

```
UINT8 (* const tss_faSampleElectrode[])(UINT8 u8ElecNum, UINT8 u8Command)
```

Function description

The key detector layer calls the sensing function each time it needs to sense an electrode. The function

```
UINT8 (* const tss_faSampleElectrode[])(UINT8 u8ElecNum, UINT8 u8Command)
```

is used to select the appropriate sensing function according to the defined low level method (GPIO, ~~CTS~~ ....). The sensing function allows both layers to exchange parameters. When the key detector layer calls this function, it provides the electrode number that needs to be scanned and a command for a measurement routine. When the sensing is completed, the capacitive sensing layer returns the sensing state to the upper layer.

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8ElecNum	Any valid Electrode number	Number of electrodes to be scanned by the capacitive sensing layer
UINT8	u8Command	Any valid command: TSS_SAMPLE_COMMAND_DUMMY TSS_SAMPLE_COMMAND_RESTART TSS_SAMPLE_COMMAND_PROCESS TSS_SAMPLE_COMMAND_RECALIB TSS_SAMPLE_COMMAND_GET_NEXT_ELECTRODE TSS_SAMPLE_COMMAND_ENABLE_ELECTRODE	These commands are used for management of the measurement process and setting of the electrode from the upper level

Return value

The function returns the resulting status of the electrode sensing to the key detector layer.

Possible return values are described below:

Return Value	Description
TSS_SAMPLE_STATUS_OK	Electrode sensing was successfully finished.
TSS_SAMPLE_STATUS_PROCESSING	The electrode sensing process is still running.
TSS_SAMPLE_STATUS_CALIBRATION_CHANGED	Electrode was recalibrated.
TSS_SAMPLE_RECALIB_REQUEST_LOCAP	Electrode sensing was not successful done due to small capacitance measured value. Recalibration is requested.
TSS_SAMPLE_RECALIB_REQUEST_HICAP	Electrode sensing was not successful done due to high capacitance measured value. Recalibration is requested.
TSS_SAMPLE_ERROR_SMALL_TRIGGER_PERIOD	Small trigger period was detected
TSS_SAMPLE_ERROR_CHARGE_TIMEOUT	Electrode sensing was not successful done due to charge timeout.
TSS_SAMPLE_ERROR_SMALL_CAP	Electrode sensing was not successful done due to small capacity

When a fault occurs, the key detector layer determines if the sense capacitance was too small, too large, recalibration needed, or if another not standard state is detected.

4.1.2 Low-level initialization

The initialization of low level hardware from the Key Detector layers is performed by the Sensor Init function.

Function prototype

```
UINT8 TSS_SensorInit(UINT8 u8Command)
```

Function description

The key detector layer calls the Sensor Init function each time it needs to change the configuration of the low level hardware, or electrode. The function `UINT8 TSS_SensorInit(UINT8 u8Command)` is used to reconfigurate the entire low level, that is all electrodes regardless of measurement method. The Sensor Init function allows both layers to exchange parameters. When the key detector layer calls this function, it provides a command for this function. When the configuration of low level is completed, the Sensor Init function returns the state to the upper layer.

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8Command	Any valid command: • TSS_INIT_COMMAND_DUMMY • TSS_INIT_COMMAND_INIT_MODULES • TSS_INIT_COMMAND_ENABLE_ELECTRODES • TSS_INIT_COMMAND_SET_NSAMPLES • TSS_INIT_COMMAND_INIT_TRIGGER • TSS_INIT_COMMAND_SW_TRIGGER • TSS_INIT_COMMAND_INIT_LOWPOWER • TSS_INIT_COMMAND_GOTO_LOWPOWER • TSS_INIT_COMMAND_RECALIBRATE	These commands are used to set the entire low level hardware from the upper level

Return value

The function returns the resulting status of the configuration to the key detector layer.

Possible return values are described below.

Return Value	Description
TSS_INIT_STATUS_OK	Configuration was successful.
TSS_INIT_STATUS_LOWPOWER_SET	Configuration of Low Power function was successful.
TSS_INIT_STATUS_LOWPOWER_ELEC_SET	Configuration of Low Power electrode was successful.
TSS_INIT_STATUS_TRIGGER_SET	Configuration of Trigger was successful.
TSS_INIT_STATUS_AUTOTRIGGER_SET	Configuration of Auto Trigger mode was successful.
TSS_INIT_STATUS_CALIBRATION_CHANGED	Calibration was changed.
TSS_INIT_ERROR_RECALIB_FAULT	Calibration was unsuccessful.

4.2 Decoder interface

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 to present the status of a control in a behavioral way. It is important to understand that decoder-related code exists only once in memory. Decoders can be seen as classes of an object oriented language. Each control has a decoder associated to it, so the control becomes an instance of the decoder (an object). However, not all decoders are necessarily instantiated in every system. The TSS library supports Rotary, Slider, Analog rotary, Analog slider, Matrix, and Keypad inside of the precompiled library files, but there is interface which allows to implement other decoders externally. For more information about the Decoders functiona, refer to A.4, “Decoder Functions.”

4.2.1 Decoder main function

The exchange of information between the decoder and key detector layer is performed using the following function.

Function prototype

```
UINT8 (* const tss_faDecoders[])(UINT8 u8CtrlNum, const UINT16 *pu16Buffer,
UINT8 u8Command)
```

Function description

The key detector layer calls the decoder function everytime electrodes are measured. The decoder function allows both layers to exchange parameters. When the key detector layer calls this function, it provides the control number which is processed, pointer to the data which provides the electrode state information with signal change, and a command. When the decoder function finishes processing, it returns the status to the lower layer.

The electrode state buffer always provides a pointer to the data of the electrodes related to the control. The data consists of two 16 bit variables. The first variable provides information about the touch state of the electrodes in the form of bit flags relative to the control (logic one is touch state, zero is release state). The second variable provides information about the signal change of the electrodes in the form of bit flags relative to the control (logic one represents signal change).

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8CtrlNum	Any valid control number	Number of control to be processed
UINT16	*pu16Buffer	Pointer to the electrode state data	The pointer to the data which provides electrode state information with signal change information
UINT8	u8Command	Any valid command: TSS_DECODER_COMMAND_DUMMY TSS_DECODER_COMMAND_PROCESS TSS_DECODER_COMMAND_GET_TOUCH_STATUS TSS_DECODER_COMMAND_GET_IDLE_SCAN_RATE TSS_DECODER_COMMAND_INIT TSS_DECODER_COMMAND_GET_ENABLE_STATUS TSS_DECODER_COMMAND_GET_THRSHLD_RATIO	These commands are used for decoder process management and decoder setting from lower level

Return value

The function returns the resulting status of the decoder to the key detector layer.

Possible return values are described below:

Return Value	Description
TSS_DECODER_STATUS_OK	Decoder processing successfully finished
TSS_DECODER_STATUS_TOUCHED	The answer to the command TSS_DECODER_COMMAND_GET_TOUCH_STATUS if at least one electrode of the control is touched.

Return Value	Description
TSS_DECODER_ERROR_ILLEGAL_CONTROL_TYPE	Called decoder type does not match with control type
TSS_DECODER_STATUS_BUSY	Decoder is not ready to perform processing.
TSS_DECODER_STATUS_ENABLED	The answer to the command TSS_DECODER_COMMAND_GET_ENABLE_STATUS if de-coder is enabled by the user.

4.2.2 Writing to the decoder schedule counter

The Key Detector provides an internal counter that can schedule decoder calls periodically after a defined number of TSS_Task measurement cycles. The initialization of the control schedule counter is performed by this function.

Function prototype

```
void TSS_KeyDetectorSetControlScheduleCounter(UINT8 u8CntrlNum, UINT8 u8Value)
```

Function description

The key detector layer decrements the control schedule counter after each electrode measurement cycle is finished.

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8CntrlNum	Any valid control number	Control index
UINT8	u8Value	1-255	Number of TSS_Task measurement cycles

Return value

None

4.2.3 Reading the decoder schedule counter

The key detector provides an internal counter that can schedule a decoder call after a defined number of TSS_Task measurement cycles. The status of this counter can be read by this function.

Function prototype

```
UINT8 TSS_KeyDetectorGetControlScheduleCounter(UINT8 u8CntrlNum)
```

Function description

The key detector layer decrements control schedule counter after each measurement cycle. The function returns actual state of the counter

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8CntrlNum	Any valid control number	Control index

Return value

The status of the control schedule counter.

4.2.4 Reading the electrode boundaries in the control

The key detector provides information about the maximum system electrode number assigned to the control, refer to Section 2.1.30, "Number of electrodes assigned to control". This can be helpful for finding the electrode status buffer end provided to a decoder. The electrode boundary for the control can be read by this function.

Function prototype

```
UINT8 TSS_KeyDetectorGetCntrlElecBound(UINT8 u8CntrlNum)
```

Function description

The function returns boundary of the last system electrode number assigned to the control.

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8CntrlNum	Any valid control number	Number of control which is required electrode boundary for

Return value

The boundary of the last system electrode number assigned to the control.

4.2.5 Reading the instant delta in the control

The key detector provides information about the electrode instant delta value.

Function prototype

```
INT8 TSS_KeyDetectorGetInstantDelta(UINT8 u8ElecNum)
```

Function description

The function returns instant delta value for a defined electrode.

Input parameters

Type	Name	Valid range/values	Description
UINT8	u8ElecNum	Any valid electrode number	Number of instant delta electrode

Return value

The instant delta value of a defined electrode.

Appendix A Touch Sensing Algorithms

A.1 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 A-1 shows the electric diagram of the circuit.

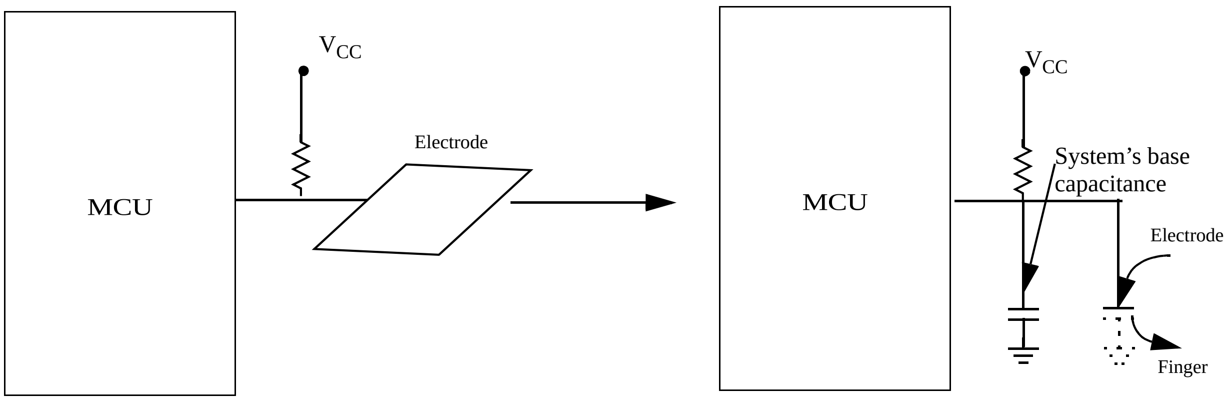


Figure A-1. Electrode equivalent circuit

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:

$$\tau = RC \qquad \textbf{\textit{Eqn. A-1}}$$

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. Figure A-2 shows the charging time of the electrode circuit.

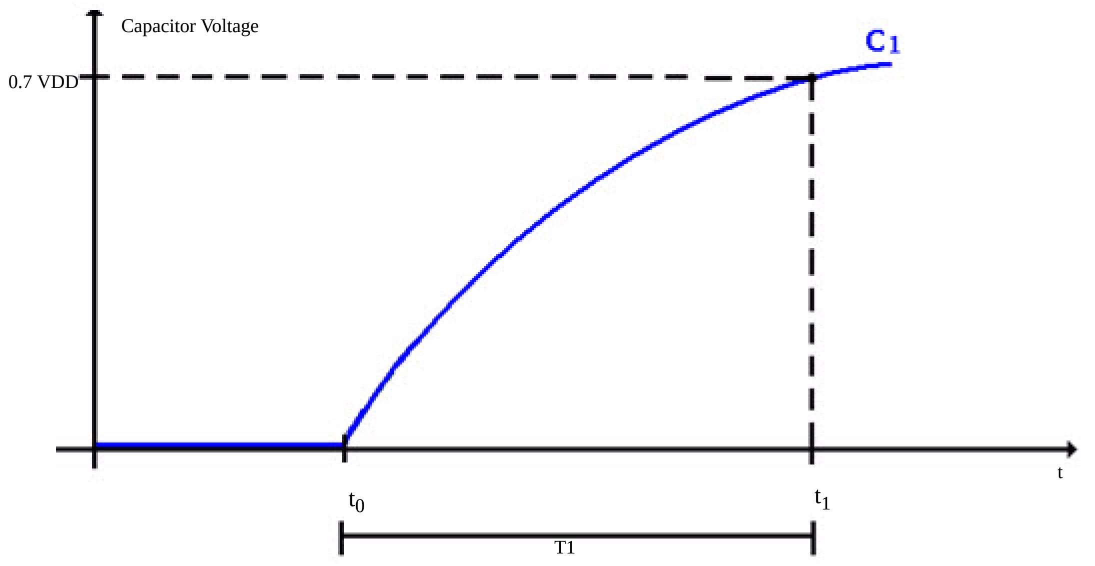


Figure A-2. RC capacitor charge curve

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 A.1.1, “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.

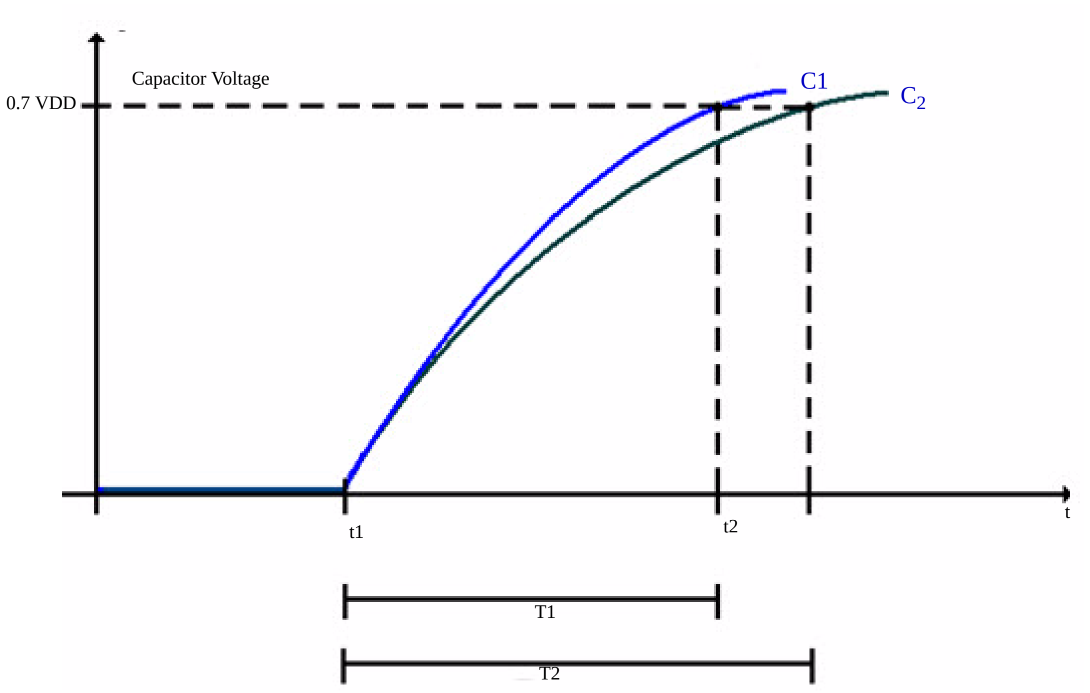


Figure A-3. Charge Time of Capacitor with Finger Added Capacitance

As shown in Figure A-3:

- 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.

Figure A-3 shows that when the capacitance increases due to the finger capacitance in the electrode, the charging time increases. The TSS library uses the charging time difference to determine if the electrode is touched or not.

A.1.1 Electrode capacitive sensing algorithm

Figure A-4 shows the Capacitive Sensing algorithm for the GPIO measurement method. The program starts with recognition of command from Keydetector.

In the case of the RESTART command the function just reports the PROCESSING state and escapes, because the GPIO based method does not need to do anything, but it is important for a unified API.

In the PROCESS command the function, sets the returning value as OK, stops the counter and resets it, and then clears the sample interrupt flag. 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 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 4.2.5.1, "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, the timeout return value is set and the number of remaining samples is set to zero.
   — If the sample is valid, the charging time value is stored 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. The number of remaining samples is read. If the number of samples is zero, the status value is returned and the program ends. Otherwise the algorithm goes back the discharge step repeating the sampling process.

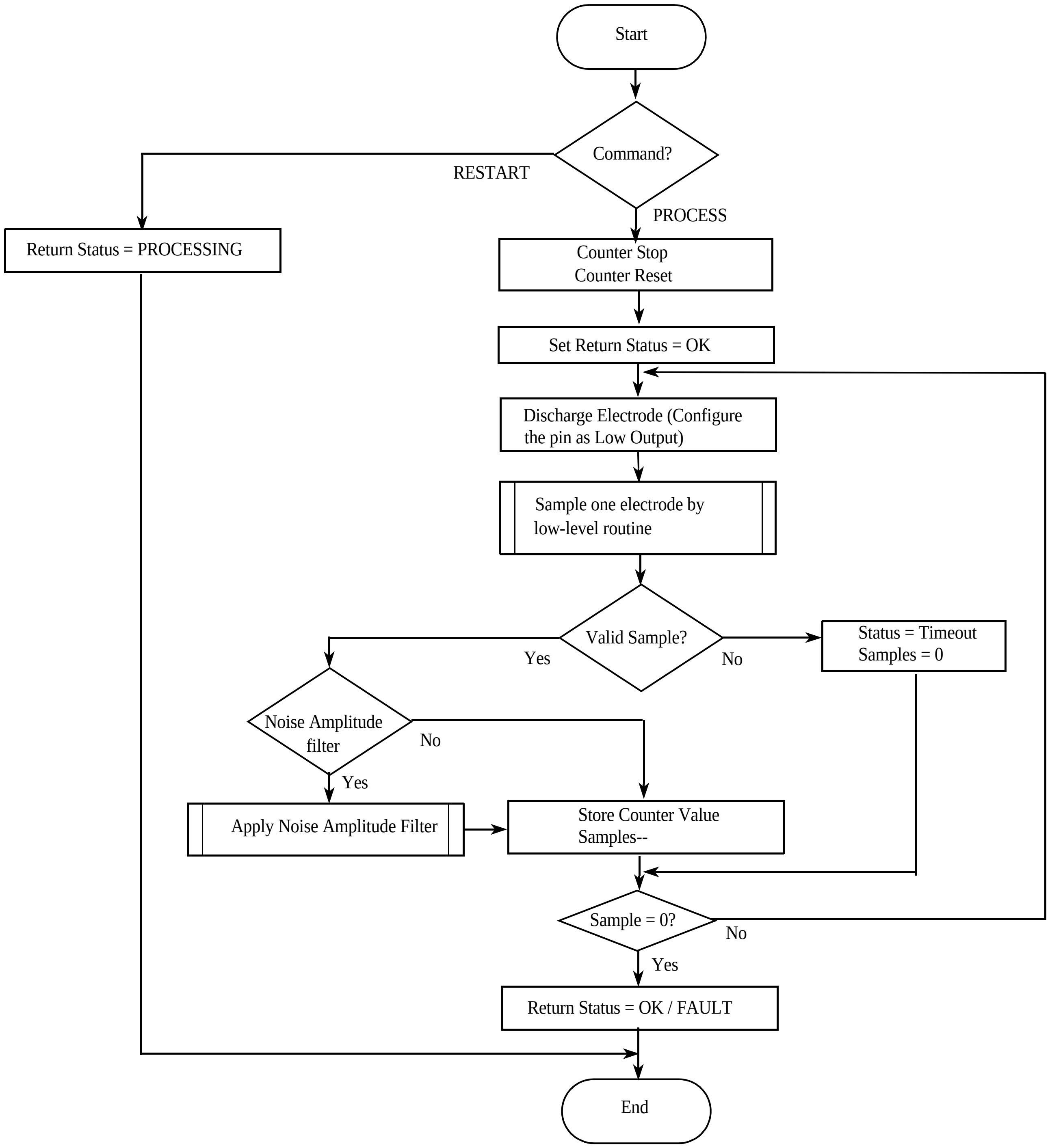


Figure A-4. Capacitive sensing algorithm

4.2.5.1 GPIO based low level Routine

Figure A-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 TSS_SampleElectrodeLowEx(void)`, or it can be directly programmed code in the GPIO Electrode Sensing Algorithm, see Section A.1.1, “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.

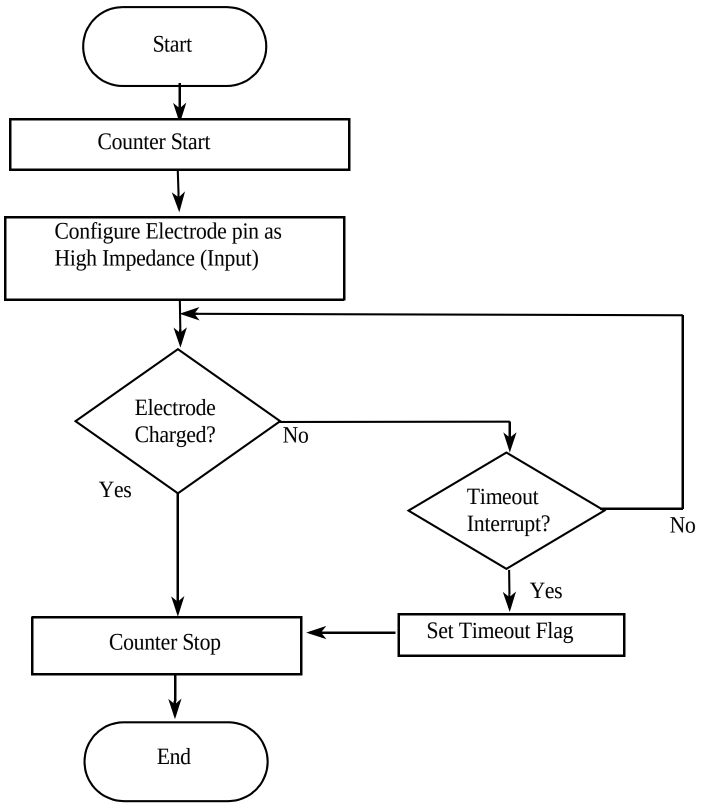


Figure A-5. GPIO based low level routine

A.1.2 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.

A.1.3 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 MHz = 400ns

Maximum amount of time is:

255 * 400ns = 102us (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

A.1.4 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 A-1.

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

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

A.1.5 Sensitivity and range

As explained in Section A.1, “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 A-2 shows the maximum capacitance range and capacitance resolution at different pull-up resistance values from 500 k to 2 M Ohms.

Table A-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 V	500k	169.44	0.6645
V_{DD} > 2.3 V	680k	124.59	0.4886
V_{DD} > 2.3 V	810k	104.59	0.4102
V_{DD} > 2.3 V	1M	84.72	0.3322
V_{DD} > 2.3 V	1.5M	56.48	0.2215
V_{DD} > 2.3 V	2M	42.36	0.1661
1.8 V < V_{DD} < 2.3 V	500k	107.53	0.4217
1.8 V < V_{DD} < 2.3 V	680k	79.07	0.3101
1.8 V < V_{DD} < 2.3 V	810k	66.38	0.2603
1.8 V < V_{DD} < 2.3 V	1M	53.77	0.2108
1.8 V < V_{DD} < 2.3 V	1.5M	35.84	0.1406
1.8 V < V_{DD} < 2.3 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.

A.2 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 kind of TSI module can wake the CPU in a touch event, measures all enabled electrodes in one automatic cycle, provides automatic triggering inside the module, and so on. For more details on the TSI module features, refer to an arbitrary reference manual of the MCU containing the TSI module inside. Figure A-6 is the block diagram for the TSI module.

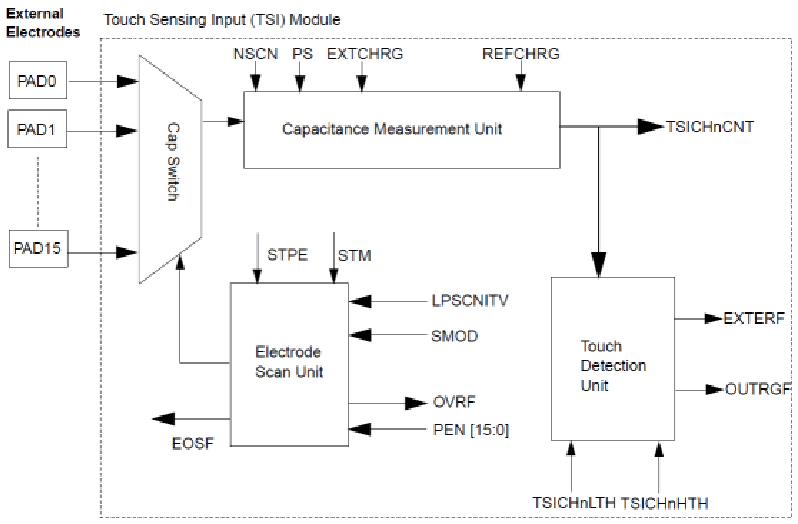


Figure A-6. TSI module version 1 block diagram

The second version of the TSI module is simplified version of the first generation. 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 A-7 shows the block diagram of the TSI module.

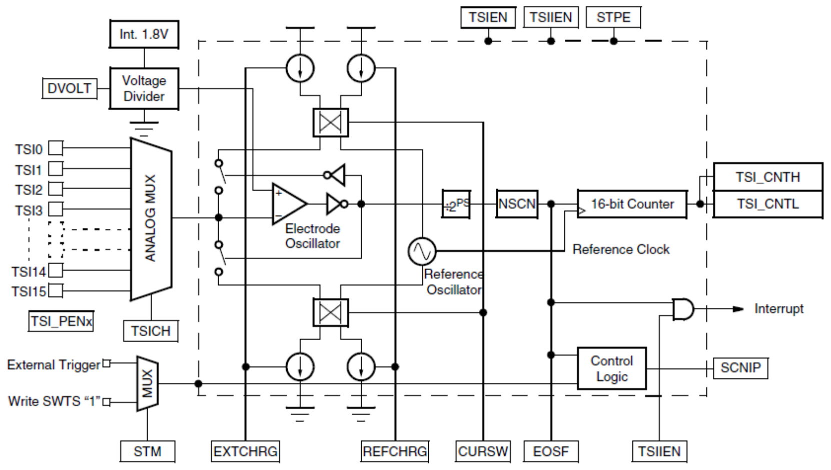


Figure A-7. TSI module version 2 block diagram

A.2.1 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 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 2.1.1, "Simple low level routines."

A.3 Key Detect Method

The key detector module determines if an electrode has been touched. If the sample is reported as a valid sample, the module compares the stored value of the charging time with the threshold value. 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. Along with this comparison, the key detector module uses a de-bounce algorithm that prevents the library from false touches.

A.3.1 Key press detection

Figure A-8 shows the four states in which the electrodes can be found by the key detector module.

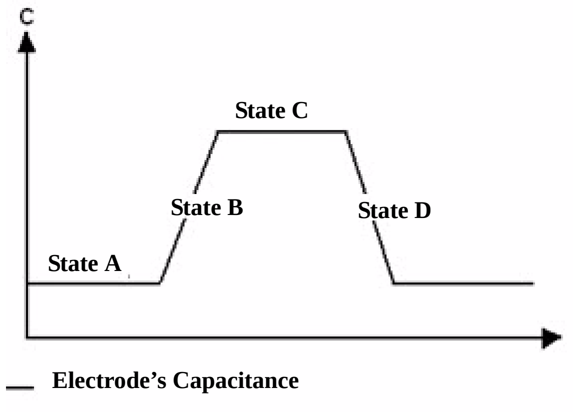


Figure A-8. Electrode states

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 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.

A.3.2 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 A-3 below.

Table A-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 Auto Trigger Modulo Value register 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. For more details on how to setup the trigger function refer to Section 2.1.16, “Trigger function.”

If the trigger function is enabled, writing to the System Trigger and Auto Trigger Modulo Value 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 Auto Trigger Modulo Value register defines the auto trigger period only for the TSIx auto trigger source.

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 2.1.38, “TSI active mode clock settings.”

If the 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.

A.3.3 Low power function

The low power function 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. Then 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 A.3.4, “Proximity function.”

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 2.1.17, “Low power control source.”
2. The user defines LowPowerScanPeriod, LowPowerElectrode, and LowPowerElectrodeSensitivity registers by the TSS_SetSystemConfig. For more details about these registers, refer to Section 3.4.10, “Low Power Scan Period register”, Section 3.4.11, “Low Power Electrode register” and Section 3.4.12, “Low Power Electrode Sensitivity register”.
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. For more details about this register, refer to Section 3.4.5, “System Configuration register.” 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.

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, so the macro TSS_TSI_LPCLKS in the TSS_SystemSetup.h file needs to be defined. For more details about this macro setting, refer to Section 2.1.39, “TSI low power mode clock settings.” The LPSCNITV register will be set directly from LowPowerScanPeriod register. 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 reposnsible for initialization of an external clock source for this purpose, for example the LPTMR timer on ARM®Cortex™-M0 family.

A.3.4 Proximity function

The proximity function 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 electrode can also wake the MCU, if the proximity function is combined with the low power function described in Section A.3.3, “Low power function”. In this case, the LowPowerElectrode, and LowPowerElectrodeSensitivity registers are used for Low Power electrode and also for Proximity electrode settings. The dc-tracker function is disabled in proximity mode.

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 2.1.15, “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 2.1.37, “TSI autocalibration settings”, Section 2.1.35, “Prescaler configuration of TSS hardware timer”, or Section 2.1.36, “Timeout configuration of TSS hardware timer”.
3. The user defines LowPowerElectrode, and LowPowerElectrodeSensitivity registers by the TSS_SetSystemConfig. For more details about these registers, refer to Section 3.4.11, “Low Power Electrode register.” and Section 3.4.12, “Low Power Electrode Sensitivity register.”.
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 3.4.5, “System Configuration register.” This action switches to TSI and GPIO proximity configuration and initates 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.

A.3.5 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 estimated noise level and touch tracking information. The user does not need to set the electrode sensitivity anymore, but manual settings are still availabe in the case for precise tuning. For more information on how to configure the ASC, refer to Section 2.1.11, “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 A-10 shows a description of 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.

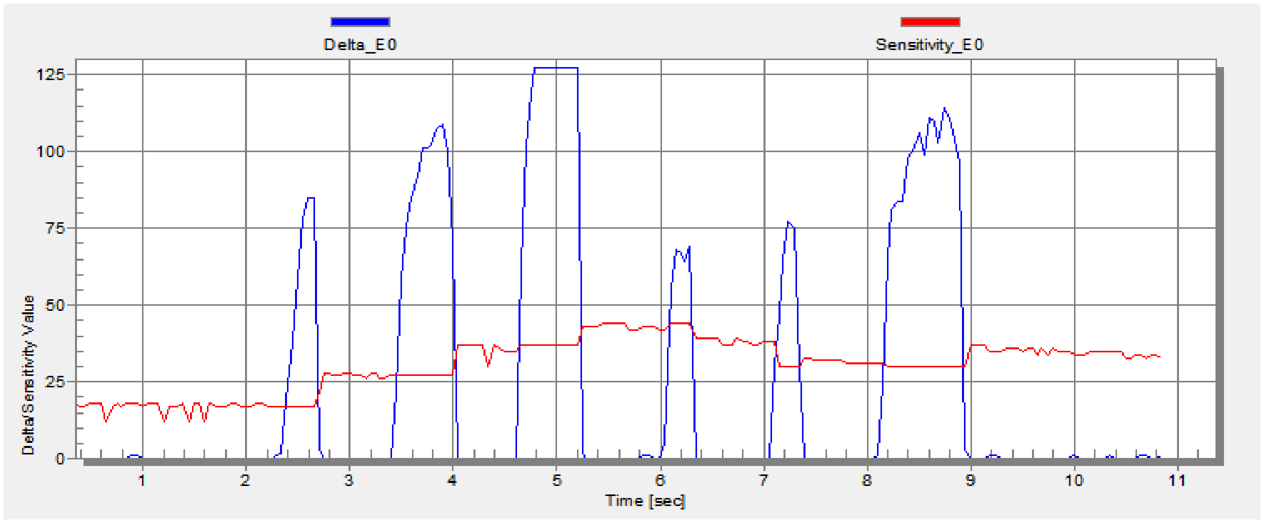


Figure A-9. Automatic Sensitivity Calibration function

A.3.6 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. 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. For more information on how to configure the baseline-tracking rate, refer to Section 3.4.7, “DC Tracker Rate register.”

A.3.6.1 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 A-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 Section 3.4.7, "DC Tracker Rate register." The function can be 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 2.1.20, "Negative baseline drop."

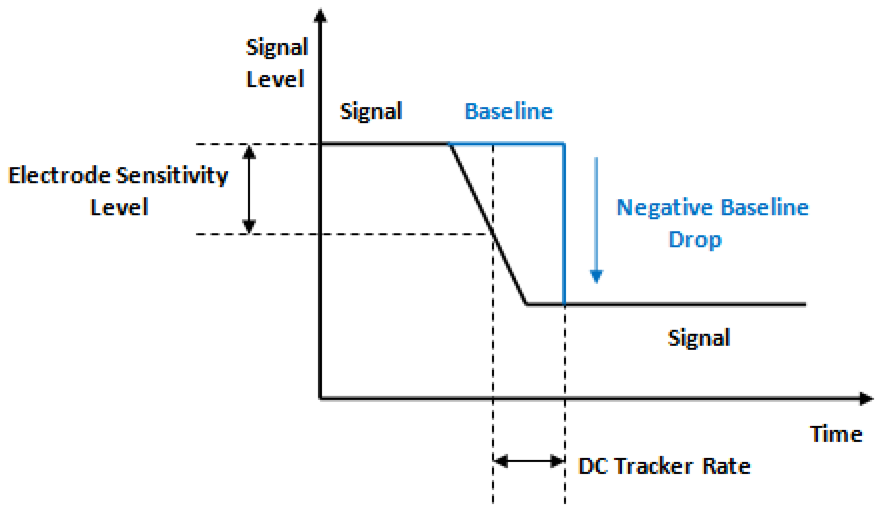


Figure A-10. Negative baseline drop

A.3.7 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 the key detector layer. The debouncing rate is modified with the NSamples register that defines how many samples need to be valid before considering a measured value as valid. Refer to Section 3.4.6, "Number of Samples registers."

A.3.8 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). For more information on how to enable the IIR filter, refer to Section 2.1.7, “IIR filter.”

A.3.9 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 2.1.8, “Noise amplitude filter.”

A.3.10 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 A.3.10.1, “Standard shielding function.” and water tolerance mode described Section A.3.10.2, “Water tolerance mode.” It is important to note that the DC-Tracker function and Negative Baseline Drop function are not performed on shielding, or shielded electrode. For more information on how to enable the shielding function, refer to Section 2.1.9, “Shielding function.”

A.3.10.1 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.

A.3.10.2 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. 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.

A.4 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 A-11.

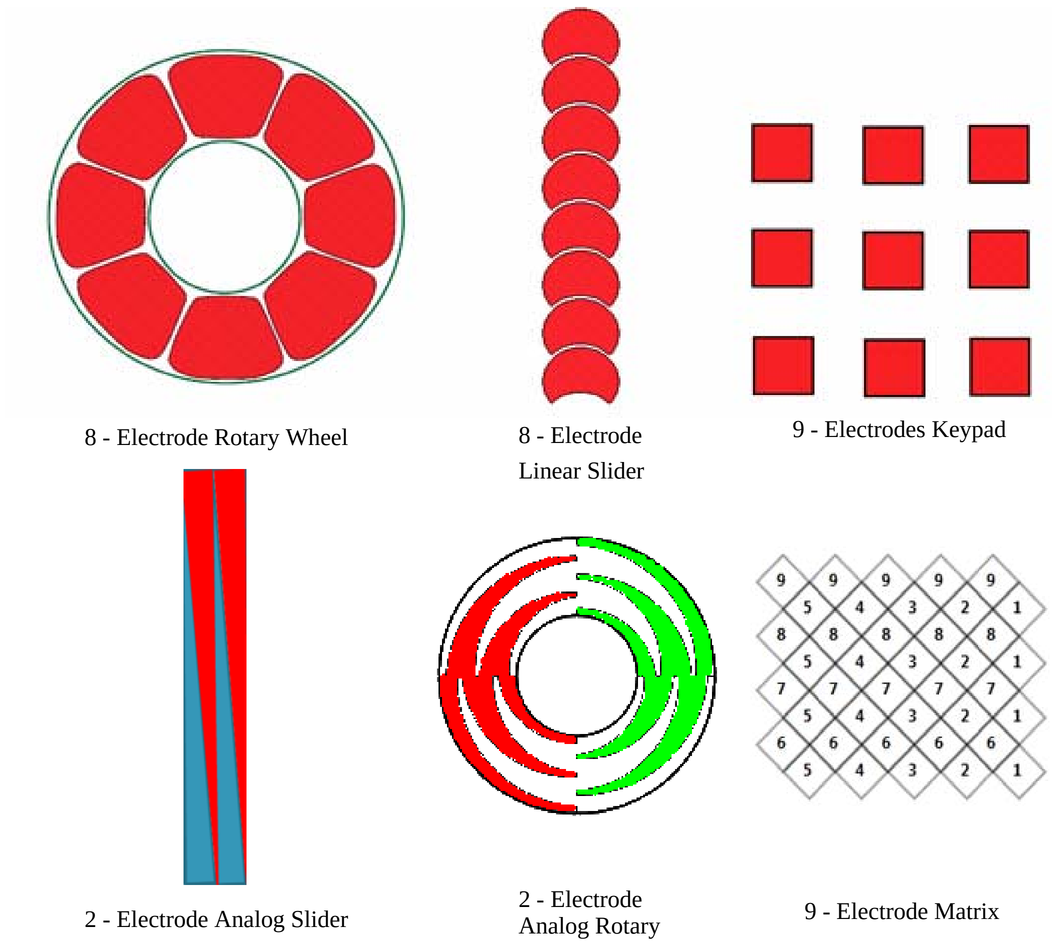

8 - Electrode Rotary Wheel
8 - Electrode Linear Slider
9 - Electrodes Keypad
2 - Electrode Analog Slider
2 - Electrode Analog Rotary
9 - Electrode Matrix

Figure A-11. Linear slider, rotary sliders, and keypad configuration

Figure A-11 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.

A.4.1 Keypad

Keypad is a basic configuration for the arranged electrodes shown in Figure A-11 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 A-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.

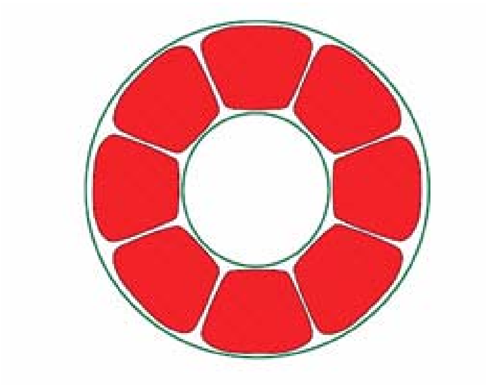
The Keypad decoder is capable of using groups of electrodes that need to be simultaneously touched for reporting the defined key. This allows to create a control interface with more user inputs than the number of physical electrodes.

Table A-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

A.4.2 Rotary

As mentioned in Section A.4, “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 A-12 shows the electrode configuration needed to implement a rotary slider.



8 - Electrode Linear Slider

Figure A-12. 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 A-5 along with a brief explanation.

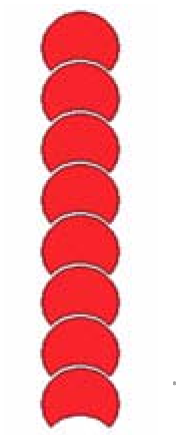
The Rotary Decoder determines the parameters listed in Table A-5 based on which electrodes have been pressed and which are being pressed.

Table A-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.

A.4.3 Slider

A linear slider control works in a similar way as the rotary slider . The same parameters must be reported in both. Figure A-13 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 A-13.



8 - Electrode Linear Slider

Figure A-13. Linear slider configuration

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

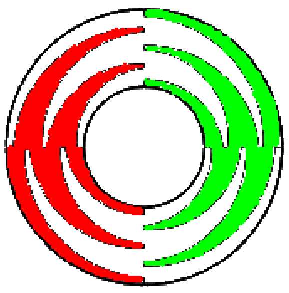
Table A-6. Events reported by the slider decoder

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 A-6 by the interpretation of the past and current touch and release events.

A.4.4 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 A-14 shows an arrangement of electrodes used for a typical analog rotary.



4 - Electrode Analog Rotary

Figure A-14. 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 A-7 based on which electrodes have been pressed and which are being pressed.

Table A-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.

A.4.5 Analog Slider

An analog slider control works similiar 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 A-15 shows an arrangement of electrodes used for a typical analog slider.

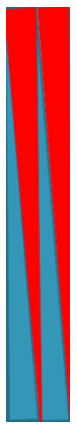


Figure A-15. Analog slider configuration

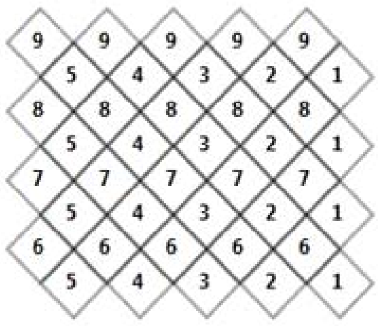
The Analog Slider Decoder determines the parameters listed in Table A-8. This based on which electrodes have been pressed and which are being pressed.

Table A-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.

A.4.6 Matrix

An analog matrix works similarly as the analog slider, but the position is calculated in 2 dimensions horizontal X, and vertical Y. Eeach axis has a defined range of postion 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 A-16 shows an arrangement of electrodes used for a typical matrix.



9 - Electrode Matrix

Figure A-16. Matrix configuration

The Matrix Decoder determines the parameters listed in Table A-9. This based on which electrodes have been pressed and which are being pressed.

Table A-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.

A.5 Glossary

API	Application Programming Interface
MCU	Microcontroller
TSS	Touch Sensing Software

TSIL	Touch Sensing Input Lite
ASC	Automatic Sensitivity Calibration