Atmel QTouch Safety Library

Peripheral Touch Controller

USER GUIDE

Description

Atmel® QTouch® Peripheral Touch Controller (PTC) offers built-in hardware for buttons, sliders, and wheels. PTC supports both mutual and self capacitance measurement without the need for any external component. It offers superb sensitivity and noise tolerance, as well as self-calibration and minimizes the sensitivity tuning effort by the user.

The PTC is intended for acquiring capacitive touch sensor signals. The external capacitive touch sensor is typically formed on a PCB, and the sensor electrodes are connected to the analog charge integrator of the PTC using the device I/O pins. The PTC supports mutual capacitance sensors organized as capacitive touch matrices in different X-Y configurations, including Indium Tin Oxide (ITO) sensor grids. In mutual capacitance mode, the PTC requires one pin per X line (drive line) and one pin per Y line (sense line). In self capacitance mode, the PTC requires only one pin with a Y-line driver for each self-capacitance sensor.

The PTC supports two sets of libraries, the QTouch Library and the QTouch Safety Library. The QTouch Library supports both mutual and self capacitance methods. The QTouch Safety Library is available for both GCC and IAR. The QTouch Safety Library also supports both the mutual capacitance method and self capacitance method along with the additional safety features.

1. Development Tools 8
1.1 Device Variants Supported 8

2. QTouch Safety Library 9
2.1 API Overview. 9
2.2 Sequence of Operation 11
2.3 Program Flow 12
2.4 Configuration Parameters 13
2.4.1 Pin Configuration 16
2.4.2 Sensor Configuration 18
2.4.3 Acquisition Parameters 18
2.4.4 Sensor Global Parameters 20
2.4.5 Common Parameters 20
2.4.6 Noise Immunity Global Parameters 21
2.4.7 Noise Immunity Features 22
2.4.8 Sensor Lockout 25
2.4.9 Frequency Auto Tune 25
2.5 Touch Library Error Reporting Mechanism 27
2.6 Touch Library Program Counter Test 27
2.6.1 Logical Program Flow Test 27
2.6.2 Program Counter Test 29
2.7 CRC on Touch Input Configuration 30
2.8 Double Inverse Memory Check 32
2.8.1 Application to Touch Library 32
2.8.2 Touch Library to Application 33
2.9 Application Burst Again Mechanism 36
2.10 Memory Requirement 36
2.10.1 Memory Requirement for IAR Safety library 37
2.11 API Execution Time. 38
2.11.1 Mutual Capacitance API Execution Time 38
2.11.2 Self Capacitance API Execution Time 40
2.12 Error Interpretation 43
2.12.1 Error Codes Returned Synchronously 43
2.12.2 Error Codes Returned Through Callback 45
2.13 Data and Function Protection 46
2.14 Moisture Tolerance 46
2.14.1 Moisture Tolerance Group 46
2.14.2 Multi-touch Group 46
2.15 Quick Re-burst 47
2.15.1 Synchronizing Quick Re-burst and Application Burst again 48
2.16 Reading Sensor States 48
2.17 Touch Library Suspend Resume Operation 48
2.18 Drifting on Disabled Sensors. 50

3. QTouch Safety Library API 51
3.1 Typedefs 51
3.2 Macros. 51
3.2.1 Touch Library Acquisition Status Bit Fields. 51
3.2.2 Sensor State Configurations. 52
3.3 Enumerations 53
3.3.1 Touch Library GAIN Setting(`tag_gain_t`) 53

3.3.2 Filter Level Setting (tag_filter_level_t) 53
3.3.3 Touch Library AUTO OS Setting (tag_auto_os_t) 53
3.3.4 Library Error Code (tag_touch_ret_t) 54
3.3.5 Sensor Channel (tag_channel_t) 55
3.3.6 Touch Library State (tag_touch_lib_state_t) 55
3.3.7 Sensor Type (tag_sensor_type_t) 55
3.3.8 Touch Library Acquisition Mode (tag_touch_acq_mode_t) 55
3.3.9 AKS Group (tag_aks_group_t) 56
3.3.10 Channel Gain Setting (tag_gain_t) 56
3.3.11 Sensor Recalibration Threshold (tag_recal_threshold_t) 57
3.3.12 Rotor Slider Resolution (tag_resolution_t) 57
3.3.13 Auto Tune Setting (tag_auto_tune_type_t) 57
3.3.14 PTC Clock Prescale Setting (tag_prsc_div_sel_t) 58
3.3.15 PTC Series Resistor Setting (tag_rsel_val_t) 58
3.3.16 PTC Acquisition Frequency Delay Setting (freq_hop_sel_t) 59
3.3.17 PTC Acquisition Frequency Mode Setting (tag_freq_mode_sel_t) 59
3.3.18 PTC Sensor Lockout Setting (nm_sensor_lockout_t) 60
3.3.19 Moisture Group Setting (moisture_grp_t) 60
3.3.20 Multi-touch Group Setting (mltch_grp_t) 60
3.4 Data Structures 62
3.4.1 Touch Library Configuration Type 62
3.4.2 Touch Library Safety Type 63
3.4.3 Touch Library Double Inverse Type 64
3.4.4 Touch Library Parameter Type 65
3.4.5 Touch Library Measurement Data Type 67
3.4.6 Touch Library Filter Data Type 67
3.4.7 Touch Library Time Type 68
3.4.8 Touch Library Info Type 68
3.4.9 Touch Library Version 69
3.5 Global Variables 69
3.5.1 touch_lib_fault_test_status 69
3.5.2 touch_error_app_cb 69
3.5.3 touch_suspend_app_cb 69
3.6 Functions 70
3.6.1 Touch Library Initialization 70
3.6.2 Touch Library Sensor Configuration 70
3.6.3 Touch Library Sensor Calibration 70
3.6.4 Touch Library Sensor Measurement 71
3.6.5 Touch Library Sensor Specific Touch Delta Read 71
3.6.6 Touch Library Sensor Specific Parameter Configuration Read-write 72
3.6.7 Touch Library Sensor Specific Acquisition Configuration Read-write 72
3.6.8 Touch Library Sensor Global Parameter Configuration Read-write 73
3.6.9 Touch Library Info Read 74
3.6.10 Touch Library Program Counter 74
3.6.11 Touch Library CRC Configuration Check 74
3.6.12 Touch Library Double Inverse check 75
3.6.13 Touch Library Enable Disable Sensor 75
3.6.14 Touch Library Version Information 76
3.6.15 Touch Library Moisture Tolerance 76
3.6.16 Touch PTC Peripheral Enable Disable 77
3.6.17 Touch Library Suspend Resume 78

3.6.18 Touch Library Re-Initialization . . . 78

4. FMEA . . . 80

4.1 Double Inverse Memory Check . . . 80
4.1.1 Application to FMEA . . . 80
4.1.2 FMEA to Application . . . 80
4.2 Memory Requirement . . . 80
4.2.1 Memory Requirement for IAR Library . . . 80
4.3 API Execution Time . . . 82
4.3.1 Mutual Capacitance API Execution Time . . . 82
4.3.2 Self Capacitance API Execution Time . . . 83
4.4 Error Interpretation . . . 84
4.5 Data and Function Protection . . . 85
4.6 FMEA Considerations . . . 85

5. FMEA API . . . 86

5.1 Typedefs . . . 86
5.2 Enumerations . . . 86
5.2.1 sf_fmea_faults_t . . . 86
5.3 Data Structures . . . 87
5.3.1 sf_xxxxcap_fmea_open_test_config_t . . . 87
5.3.2 sf_xxxxcap_fmea_input_config_t . . . 88
5.3.3 sf_mutlcap_fmea_fault_report_t . . . 88
5.3.4 sf_selfcap_fmea_fault_report_t . . . 90
5.4 Global Variables . . . 93
5.4.1 sf_xxxxcap_fmea_fault_report_var . . . 93
5.5 Functions . . . 93
5.5.1 sf_xxxxcap_fmea_init . . . 93
5.5.2 sf_xxxxcap_fmea_test . . . 93
5.5.3 sf_xxxcap_fmea_test_open_pins_per_channel . . . 95
5.5.4 sf_xxxxcap_fmea_stop . . . 100
5.6 Macros . . . 101

6. System . . . 102

6.1 Relocating Touch Library and FMEA RAM Area . . . 102
6.1.1 Modifying the IAR Linker File . . . 102
6.1.2 Modifying GCC Linker File . . . 103
6.2 API Rules . . . 105
6.3 Safety Firmware Action Upon Fault Detection . . . 106
6.4 System Action Upon Fault Detection . . . 106
6.5 Touch Library and FMEA Synchronization . . . 106
6.6 Safety Firmware Package . . . 108
6.7 SAMDSafety Firmware Certification Scope . . . 108
6.8 Hazard Time . . . 109
6.9 ASF Dependency . . . 109
6.10 Robustness and Tuning . . . 109
6.11 Standards compliance . . . 109
6.12 Safety Certification . . . 110

7. Known Issues . . . 111

8. References . . . 112

9. Revision History 113

Features

- Implements low-power, high-sensitivity, environmentally robust capacitive touch buttons, sliders, and wheels
- Supports mutual capacitance and self capacitance sensing
- Upto 256 channels in mutual-capacitance mode
- Upto 16 channels in self-capacitance mode
- Two pin per electrode in mutual capacitance mode - with no external components
- One pin per electrode in self capacitance mode - with no external components
- Load compensating charge sensing
- Parasitic capacitance compensation for mutual capacitance mode
- Adjustable gain for superior sensitivity
- Zero drift over the temperature and VDD range
- No need for temperature or VDD compensation
- Hardware noise filtering and noise signal desynchronization for high conducted immunity
- Supports moisture tolerance
- Atmel provided QTouch Safety Library firmware
- Supports Sensor Enable and Disable at Runtime
- Supports Quick Reburst Feature for Faster Response Time

The following features are available only in the QTouch Safety Library:

- CRC protection
- Logical program flow sequence
- Memory protection using double inverse mechanism
- Library RAM relocation and
- Compile-time and Run-time check

For more information about the capacitance related technological concepts, Refer Chapter 4 in *Atmel QTouch Library Peripheral Touch Controller User Guide [42195]* available at www.atmel.com.

Product Support

For assistance related to QTouch capacitive touch sensing software libraries and related issues, contact your local Atmel sales representative or log on to myAtmel Design Support portal to submit a support request or access a comprehensive knowledge base.

If you don't have a myAtmel account, please visit http://www.atmel.com/design-support/ to create a new account by clicking on "Create Account" in the myAtmel menu at the top of the page.

Once logged in, you will be able to access the knowledge base, submit new support cases from the myAtmel page or review status of your ongoing cases.

1. Development Tools

The following development tools are required for QTouch Safety Library development using SAMD20/SAMD21 devices:

Development Environment:

- IAR Embedded Workbench for ARM® 7.40.3.8938 for IAR Compiler
- Atmel Software Framework 3.26.0.1381.
- Atmel Studio 6.2.1563- Service Pack 2 for GCC Compiler

1.1 Device Variants Supported

QTouch Safety Library for SAMDevices is available for the following device variants:

Series	Variant
SAM D20 J Series	ATSAMD20J18, ATSAMD20J17, ATSAMD20J16, ATSAMD20J15
SAM D20 G Series	ATSAMD20G18, ATSAMD20G17, ATSAMD20G16, ATSAMD20G15, ATSAMD20G17U, ATSAMD20G18U
SAM D20 E Series	ATSAMD20E18, ATSAMD20E17, ATSAMD20E16, ATSAMD20E15
SAM D21 J Series	ATSAMD21J18A, ATSAMD21J17A, ATSAMD21J16A, ATSAMD21J16B, ATSAMD21J15A, ATSAMD21J15B
SAM D21 G Series	ATSAMD21G18A, ATSAMD21G18AU, ATSAMD21G17A, ATSAMD21G17AU, ATSAMD21G16A, ATSAMD21G16B, ATSAMD21G15A, ATSAMD21G15B
SAM D21 E Series	ATSAMD21E18A, ATSAMD21E17A, ATSAMD21E16A, ATSAMD21E16B, ATSAMD21E16BU, ATSAMD21E15A, ATSAMD21E15B, ATSAMD21E15BU

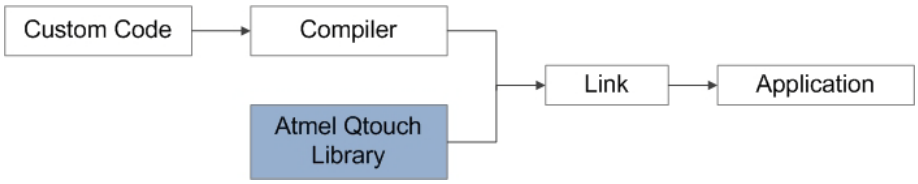
Note: `PIN_PB04` is not supported for Touch operation in ATSAMD21G17AU and ATSAMD21G18AU devices.

2. QTouch Safety Library

Atmel QTouch Safety Library makes it simple for developers to embed capacitive-touch button, slider, wheel functionality into general-purpose Atmel SAMD20/SAMD21 microcontroller applications. The royalty-free QTouch Safety Library provides library files for each device and supports different numbers of touch channels, enabling both flexibility and efficiency in touch applications.

QTouch Safety Library can be used to develop single-chip solutions for many control applications, or to reduce chip count in more complex applications. Developers have the latitude to implement buttons, sliders, and wheels in a variety of combinations on a single interface.

Figure 2-1. Atmel QTouch Safety Library



2.1 API Overview

QTouch Safety Library API for PTC can be used for touch sensor pin configuration, acquisition parameter setting as well as periodic sensor data capture and status update operations. The QTouch Safety Library interfaces with the PTC module to perform the required actions. The PTC module interfaces with the external capacitive touch sensors and is capable of performing mutual and self capacitance method measurements.

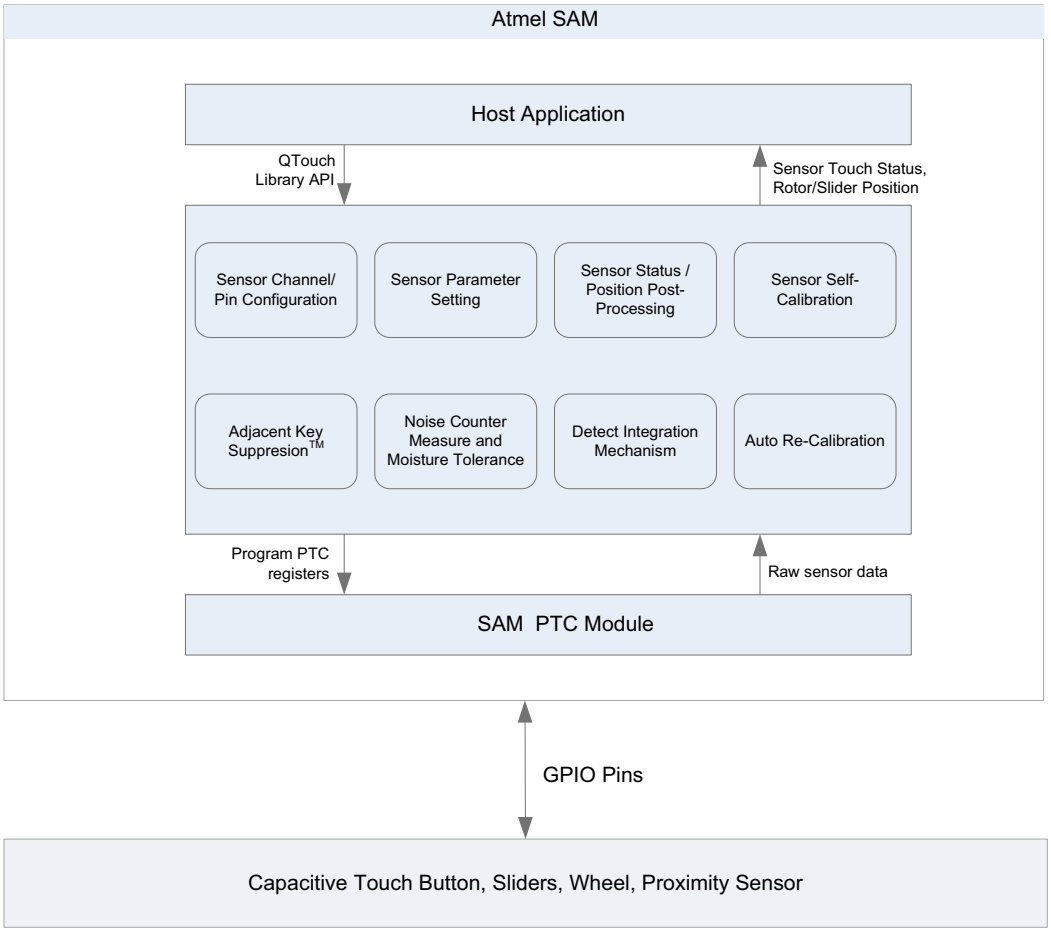
Note: From this section onwards, the program elements that are common to both mutual and self capacitance technologies are represented using `XXXXCAP` or `xxxxcap`.

For normal operation, it is sufficient to use the Regular APIs. The Helper APIs provides additional flexibility to the user application. The available APIs are listed in the following table.

Table 2-1. Regular and Helper APIs

Regular API	Helper API
touch_xxxxcap_sensors_init touch_xxxxcap_di_init touch_xxxxcap_sensor_config touch_xxxxcap_sensors_calibrate touch_xxxxcap_sensors_measure touch_xxxxcap_sensors_deinit	touch_xxxxcap_sensor_get_delta touch_xxxxcap_sensor_update_config touch_xxxxcap_sensor_get_config touch_xxxxcap_update_global_param touch_xxxxcap_get_global_param touch_xxxxcap_update_acq_config touch_xxxxcap_get_acq_config touch_xxxxcap_get_libinfo touch_xxxxcap_calibrate_single_sensor touch_xxxxcap_sensor_disable touch_xxxxcap_sensor_reenable touch_lib_pc_test_magic_no_1 touch_lib_pc_test_magic_no_2 touch_lib_pc_test_magic_no_3 touch_lib_pc_test_magic_no_4 touch_calc_xxxcap_config_data_integrity touch_test_xxxcap_config_data_integrity touch_xxxxcap_cnfg_mois_mltchgrp touch_xxxxcap_cnfg_mois_threshold touch_xxxxcap_mois_tolrnce_enable touch_xxxxcap_mois_tolrnce_disable touch_library_get_version_info touch_disable_ptc touch_enable_ptc touch_suspend_ptc touch_resume_ptc

Figure 2-2. QTouch Safety Library Overview

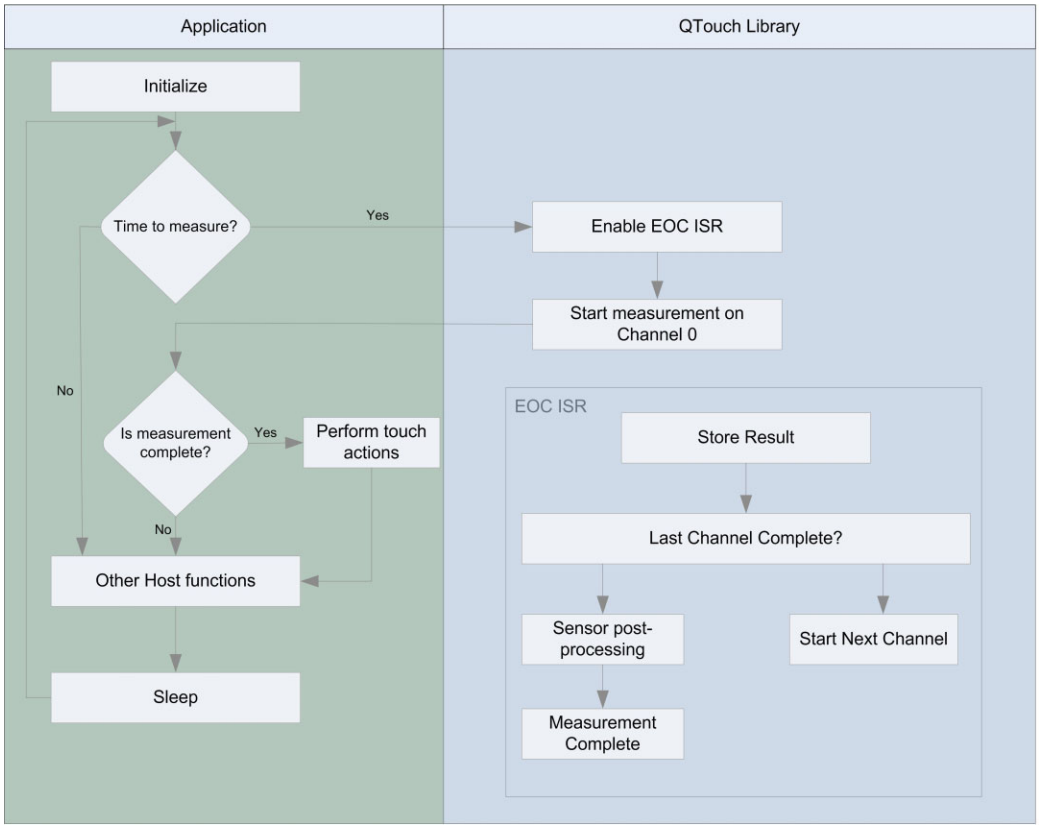


2.2 Sequence of Operation

The application periodically initiates a touch measurement on either mutual capacitance or self capacitance sensors. At the end of each sensor measurement, the PTC module generates an end of conversion (EOC) interrupt. The touch measurement is performed sequentially until all the sensors are measured. Additional post-processing is performed on the measured sensor data to determine the touch status of the sensors (keys/rotor/slider) position. The post processing determines the position value of the sensors and the callback function is then triggered to indicate completion of measurement.

The recommended sequence of operation facilitates the CPU to either sleep or perform other functions during touch sensor measurement.

Figure 2-3. QTouch Application Sequence



2.3 Program Flow

Before using the QTouch Safety Library API, configure the PTC module clock generator source. The PTC module clock can be generated using one of the eight generic clock generators (GCLK0-GCLK7). Configure the corresponding generic clock multiplexer such that the PTC module clock is set between 400 kHz and 4 MHz.

The `touch_xxxxcap_sensors_init` API initializes the QTouch Safety Library as well as the PTC module. Additionally, it initializes the capacitance method specific pin, register, and global sensor configuration.

The `touch_xxxxcap_di_init` API initializes the memory for different pointers in the `touch_lib_xxxcap_param_safety` structure.

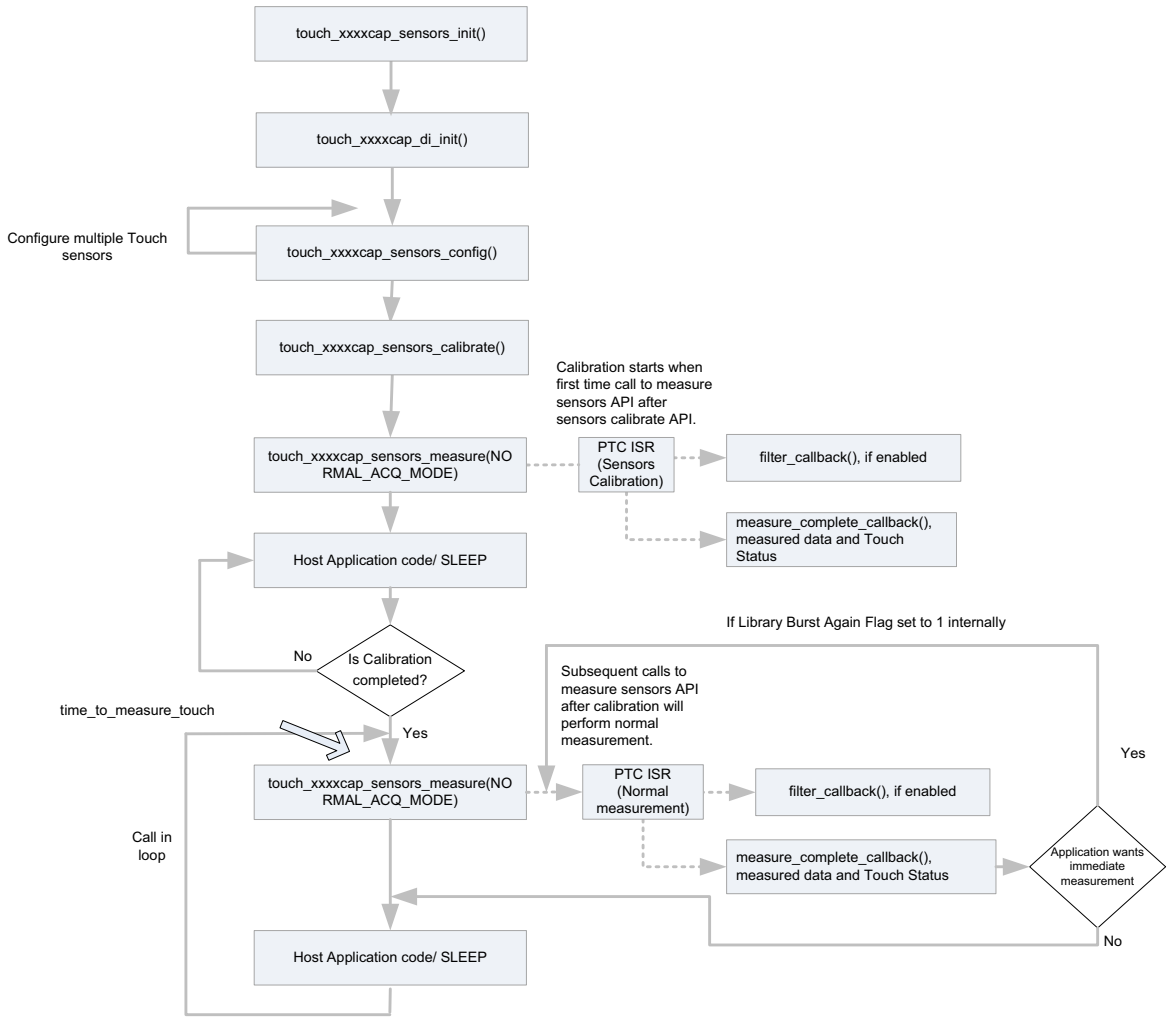
The `touch_xxxxcap_sensor_config` API configures the individual sensor. The sensor specific configuration parameters can be provided as input arguments to this API.

The `touch_xxxxcap_sensors_calibrate` API calibrates all the configured sensors and prepares the sensors for normal operation. The auto tuning type parameter is provided as input argument to this API.

The `touch_xxxxcap_sensors_measure` API initiates a touch measurement on all the configured sensors.

The sequence of the mandatory APIs are depicted in the following illustration.

Figure 2-4. API Usage



For configuring multiple sensors, `touch_xxxxcap_config_sensor` must be called every time to configure each sensor.

2.4 Configuration Parameters

The following parameters are available in the QTouch Safety Library for configuring capacitance.

Table 2-2. Configuration Parameters for Mutual and Self capacitance Methods

Parameter	Parameter Macros	Description
Pin Configuration	DEF_MUTLCAP_NODES DEF_SELFCAP_LINES	Number of Mutual Capacitance nodes. Number of Self Capacitance lines.
Sensor Configuration	DEF_XXXXCAP_NUM_CHANNELS	Number of Channels.
	DEF_XXXXCAP_NUM_SENSORS	Number of Sensors.
	DEF_XXXXCAP_NUM_ROTORS_SLIDERS	Number of Rotor/Sliders.

Table 2-2. Configuration Parameters for Mutual and Self capacitance Methods

Parameter	Parameter Macros	Description
Acquisition Parameters	DEF_XXXXCAP_FILTER_LEVEL_PER_NODE	The filter level setting controls the number of samples collected to resolve each acquisition. This is applicable for individual channel.
	DEF_XXXXCAP_GAIN_PER_NODE	Gain is applied for an individual channel to allow a scaling-up of the touch delta.
	DEF_XXXXCAP_AUTO_OS_PER_NODE	Auto oversample controls the automatic oversampling of sensor channels when unstable signals are detected. This is applicable for individual channel.
	DEF_XXXXCAP_FREQ_MODE	Frequency mode setting allows users to configure the bursting waveform characteristics to get better noise performance for the system.
	DEF_XXXXCAP_CLK_PRESCALE_PER_NODE	This method is used to select the PTC prescaler. This is applicable for individual channel.
	DEF_XXXXCAP_SENSE_RESISTOR_PER_NODE	This method is used to select the sense resistor value. This is applicable for individual channel.
	DEF_XXXXCAP_CC_CAL_CLK_PRESCALE_PER_NODE	This method is used to select the PTC prescalar for CC calibration. This is applicable for individual channel.
	DEF_XXXXCAP_CC_CAL_SENSE_RESISTOR_PER_NODE	This method is used to select the sense resistor for CC calibration. This is applicable for individual channel.
	DEF_XXXXCAP_HOP_FREQS	Frequency hops to be performed. Maximum three frequency hops is possible.

Table 2-2. Configuration Parameters for Mutual and Self capacitance Methods

Parameter	Parameter Macros	Description
Sensor Global Parameters	DEF_XXXXCAP_DI	Capacitance sensor detect integration (DI) limit. Range: 0u to 255u.
	DEF_XXXXCAP_TCH_DRIFT_RATE	Capacitance sensor towards touch drift rate. Range: 1u to 127u.
	DEF_XXXXCAP_ATCH_DRIFT_RATE	Capacitance sensor away from touch drift rate. Range: 1u to 127u.
	DEF_XXXXCAP_MAX_ON_DURATION	Capacitance sensor maximum ON time duration. Range: 0u to 255u.
	DEF_XXXXCAP_DRIFT_HOLD_TIME	Capacitance Sensor drift hold time. Range: 1u to 255u.
	DEF_XXXXCAP_ATCH_RECAL_DELAY	Capacitance sensor away from touch recalibration delay. Range: 0u to 255u. Specifying a value of 0u would disable the away from touch recalibration feature.
	DEF_XXXXCAP_ATCH_RECAL_THRESHOLD	Capacitance sensor away from touch recalibration threshold.
	DEF_XXXXCAP_CAL_SEQ1_COUNT	Software calibration sequence counter 1.
	DEF_XXXXCAP_CAL_SEQ2_COUNT	Software calibration sequence counter 2.
	DEF_XXXXCAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT	Defines the stability of the signals for noise measurement. Range: 1u to 1000u.
	DEF_XXXXCAP_NOISE_LIMIT	This parameter is used to select the noise limit value to trigger sensor lockout functionality. Range: 1u to 255u.
Sensor Global Parameters	DEF_XXXXCAP_LOCKOUT_SEL	This parameter is used to select the lockout functionality method. Range: 0u to 2u.
	DEF_XXXXCAP_LOCKOUT_CNTDOWN	Defines the number of measurements after which the sensor is unlocked for touch detection. Range: 1u to 255u.
	DEF_XXXXCAP_FREQ_AUTO_TUNE_SIGNAL_STABILITY_LIMIT	Defines the stability limit of signals for frequency auto tune decision making. Range: 1u to 1000u.
	DEF_XXXXCAP_FREQ_AUTO_TUNE_IN_CNT	This parameter is used to trigger the frequency auto tune. Range: 1u to 255u.

Table 2-2. Configuration Parameters for Mutual and Self capacitance Methods

Parameter	Parameter Macros	Description
Common Parameters	DEF_TOUCH_MEASUREMENT_PERIOD_MS	User for Touch measurement periodicity.
	DEF_TOUCH_PTC_ISR_LVL	PTC Module interrupt level.
	DEF_XXXXCAP_NOISE_MEAS_ENABLE	This parameter is used to enable or disable the noise measurement. Range: 0 or 1.
	DEF_XXXXCAP_FREQ_AUTO_TUNE_ENABLE	This parameter is used to enable and disable the frequency auto tune functionality. Range: 0 or 1.
	DEF_XXXXCAP_NOISE_MEAS_BUFFER_CNT	This parameter is used to select the buffer count for noise measurement buffer. Range: 3 to 10.
Moisture Tolerance and Quick re-burst Parameters	DEF_XXXXCAP_NUM_MOIS_GROUPS	This parameter is used to configure the number of moisture groups.
	DEF_XXXXCAP_MOIS_TOLERANCE_ENABLE	This parameter is used to enable or disable the Moisture tolerance feature.
	DEF_XXXXCAP_QUICK_REBURST_ENABLE	This parameter id used to enable or disable the Quick re-burst feature.

2.4.1 Pin Configuration

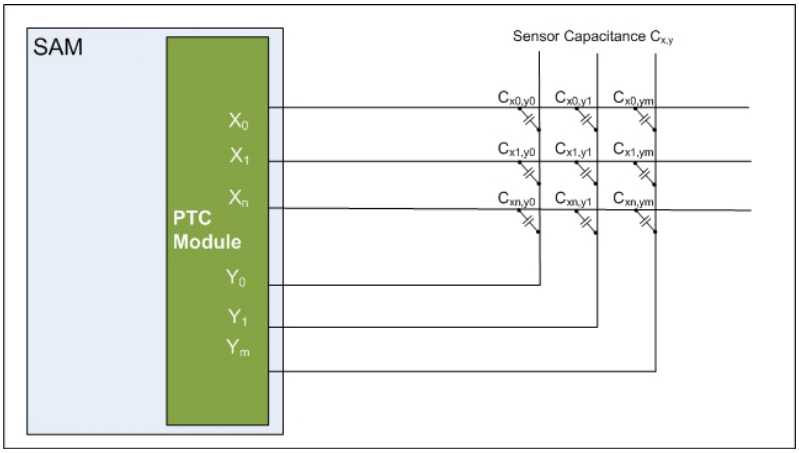
2.4.1.1 Mutual Capacitance

Mutual capacitance method uses a pair of sensing electrodes for each touch channel. These electrodes are denoted as X and Y lines. Capacitance measurement is performed sequentially in the order in which touch (X-Y) nodes are specified.

Mutual capacitance channel (X-Y channels)

- SAMD20/SAMD21 J (64 pin): up to 16(X) x 16(Y) channels
- SAMD20/SAMD21 G (48 pin): up to 12(X) x 10(Y) channels
- SAMD20/SAMD21 E (32 pin): up to 10(X) x 6(Y) channels

Figure 2-5. Mutual Capacitance Sensor Arrangement



To reduce noise issues due to EMC, use a series resistor with value of 1kΩ on X and Y lines.

2.4.1.2 Self Capacitance

Self capacitance method uses a single sense electrode for each touch channel, denoted by a Y line. Capacitance measurement is performed sequentially in the order in which Y lines are specified in the `DEF_SELFCAP_LINES` configuration parameter. Self capacitance touch button sensor is formed using a single Y line channel, while a touch rotor or slider sensor can be formed using three Y line channels.

Self capacitance channel (Y sense lines)

- SAMD20/SAMD21 J (64 pin): up to 16 channels
- SAMD20/SAMD21 G (48 pin): up to 10 channels
- SAMD20/SAMD21 E (32 pin): up to 6 channels

Figure 2-6. Self Capacitance - Sensor Arrangement

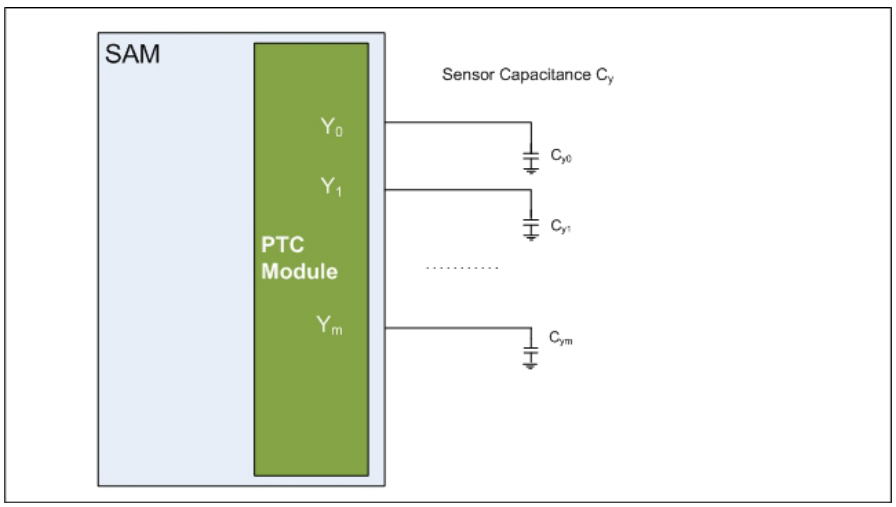
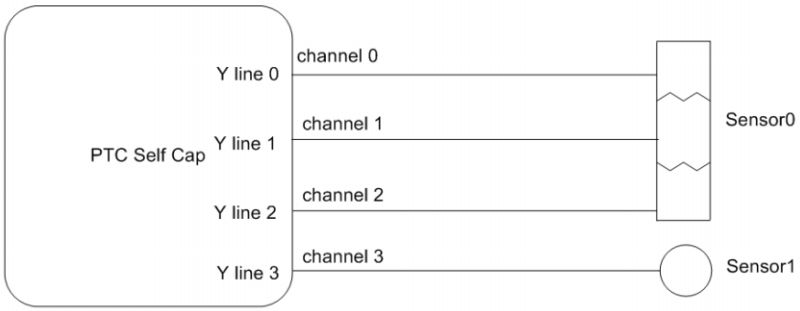


Figure 2-7. Self Capacitance - Channel to Sensor Mapping



Y sense line can be specified using the configuration parameter `DEF_SELFCAP_LINES` in non-sequential order.

The touch sensors should be enabled in the sequential order of the channels specified using the `touch_xxxxcap_sensor_config()` API.

For improved EMC performance, a series resistor with value of 1kΩ should be used on X and Y lines. For more information about designing the touch sensor, refer to *Buttons, Sliders and Wheels Touch Sensor Design Guide* available at www.atmel.com.

2.4.2 Sensor Configuration

A mutual capacitance button is formed using a single X-Y channel, while a rotor or slider can be formed using three to eight X-Y channels.

A self capacitance button is formed using a single Y channel, while a rotor or slider can be formed using three Y channels.

For more information about designing the touch sensor, refer to *Buttons, Sliders and Wheels Touch Sensor Design Guide* [QTAN0079] (www.atmel.com).

2.4.3 Acquisition Parameters

Filter Level Setting

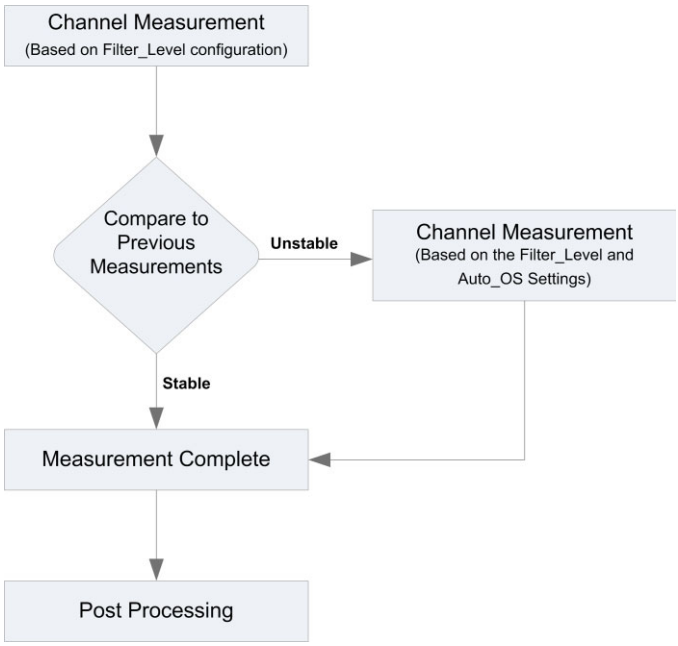
The filter level setting controls the number of samples acquired to resolve each acquisition. A higher filter level setting provides improved signal to noise ratio even under noisy conditions. However, it increases the total time for measuring the signal, which results in increased power consumption. This is applicable for individual channel.

Auto Oversample Setting

Auto oversample controls the automatic oversampling of sensor channels when unstable signals are detected with the default `Filter level` setting. Enabling Auto oversample results in `Filter level` x `Auto Oversample` number of samples measured on the corresponding sensor channel when an unstable signal is observed. In a case where `Filter level` is set to `FILTER_LEVEL_4` and `Auto Oversample` is set to `AUTO_OS_4`, 4 oversamples are collected with stable signal values and 16 oversamples are collected when unstable signal is detected. Auto Oversampling Signal Stability will be determined by the `nm_sig_stability_limit` variable.

A higher Auto oversample setting provides improved signal to noise ratio under noisy conditions, while increasing the total time for measurement resulting in increased power consumption and response time. Auto oversamples can be disabled to obtain best power consumption. Auto oversamples should be configured for individual channel.

Figure 2-8. Auto Oversamples



Auto Tuning Options

Auto tuning parameter passed to the calibration API allows users to trade-off between power consumption and noise immunity. Following auto tuning options are available:

- `AUTO_TUNE_NONE` - Auto tuning disabled
- `AUTO_TUNE_PRSC` - Auto tuning of the PTC prescaler
- `AUTO_TUNE_RSEL` - Auto tuning of the series resistor

When *Auto tuning of the series resistor* is selected the PTC is optimized for fastest operation or lowest power operation. The PTC runs at user defined speed and the series resistor is set to the optimum value which still allows full charge transfer. Auto tuning will be performed on individual channel series resistor settings. `DEF_XXXXCAP_SENSE_RESISTOR_PER_NODE` will be tuned by the QTouch Safety Library.

When *Auto tuning of PTC prescaler* is selected the performance is optimized for best noise immunity. During calibration, the QTouch Safety Library carries out auto tuning to ensure full charge transfer for each sensor, by adjusting either the internal series resistor or the PTC clock prescaler. The internal series resistor is set to user defined value and the PTC prescaler is adjusted to slow down the PTC operation to ensure full charge transfer. Auto tuning will be performed on individual channel PTC prescaler settings.`DEF_XXXXCAP_CLK_PRESCALE_PER_NODE` will be tuned by the QTouch Safety Library.

Manual tuning can also be performed by passing `AUTO_TUNE_NONE` as parameter to the calibration function. When manual tuning option is selected, the user defined values of PTC prescaler and series resistor on individual channels are used for PTC operation.

Frequency Mode Setting

Frequency mode allows users to configure the bursting waveform characteristics for better noise performance in the system. Following frequency modes are available:

- `FREQ_MODE_NONE` - Frequency mode is disabled
- `FREQ_MODE_HOP` - Frequency mode hopping
- `FREQ_MODE_SPREAD` - Frequency mode spread
- `FREQ_MODE_SPREAD_MEDIAN` - Frequency mode spread median

When *frequency mode none* option is selected, the PTC runs at constant speed selected by the user (in manual tuning mode) or auto tuned frequency (in PTC rescale tune mode). In this case, the median filter is not applied.

When *frequency mode hopping* option is selected, the PTC runs at a frequency hopping cycle selected by the user (in manual tuning mode) or auto tuned frequency cycle (in PTC prescaler tune mode). In this case, the median filter is applied.

When *frequency mode spread spectrum* option is selected, the PTC runs with spread spectrum enabled on frequency selected by the user (in manual tuning mode) or auto tuned frequency (in PTC prescaler tune mode). In this case, the median filter is not applied.

When *frequency mode spread spectrum median* option is selected, the PTC runs with spread spectrum enabled on frequency selected by the user (in manual tuning mode) or auto tuned frequency (in PTC prescaler tune mode). In this case, the median filter is applied.

Gain Setting

Gain setting is applied for an individual channel to allow a scaling-up of the touch delta upon contact.

2.4.4 Sensor Global Parameters

For an overview of the sensor global and sensor specific parameters, refer Section 4.2.2 and Section 4.3 of the QTouch General Library User Guide (www.atmel.com).

Table 2-3. Global Parameters Naming Changes

QTouch Safety Library Name	Conventional QTouch Library Name
`DEF_XXXXCAP_TCH_DRIFT_RATE` Towards Touch Drift	Negative Drift
`DEF_XXXXCAP_ATCH_DRIFT_RATE` Away From Touch Drift	Positive Drift
`DEF_XXXXCAP_ATCH_RECAL_THRESHOLD` Away From Touch Recalibration Threshold	Recalibration Threshold
`DEF_XXXXCAP_ATCH_RECAL_DELAY` Away From Touch Recalibration delay	Positive Recalibration Delay
`DEF_XXXXCAP_CAL_SEQ1_COUNT` Calibration Sequence Counter 1	Software Calibration Counter 1
`DEF_XXXXCAP_CAL_SEQ2_COUNT` Calibration Sequence Counter 2	Software Calibration Counter 2

Note: Ensure that the value of `DEF_XXXXCAP_CAL_SEQ2_COUNT` is always less than the value specified in `DEF_XXXXCAP_CAL_SEQ1_COUNT`. Refer Section 2.4.6 for more information about noise immunity global parameter.

2.4.5 Common Parameters

Interrupt Priority Level Setting

The Nested Vectored Interrupt Controller (NVIC) in the SAMD20/SAMD21 has four different priority levels. The priority level of the PTC end of conversion ISR can be selected based on application requirements to accommodate time critical operations.

To avoid stack overflow, ensure that adequate stack size has been set in the user application.

2.4.5.1 Measurement Period Setting

The measurement period setting is used to configure the periodic interval for touch measurement.

2.4.6 Noise Immunity Global Parameters

2.4.6.1 Noise Measurement Parameters

Noise Measurement Enable Disable

The `DEF_XXXXCAP_NOISE_MEAS_ENABLE` parameter is used to enable or disable the noise measurement.

- 1 - Noise measurement will be enabled.
- 0 - Noise measurement will be disabled and lockout functionality will not be available.

Noise Measurement Signal Stability Limit

The parameter `DEF_XXXXAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT` defines the stability of the signals for noise measurement.

Signal values can change from sample to sample during a window buffer period. The difference between adjacent buffer value is compared to the user configured stability limit.

Noise is reported only when two changes occur within the specified window period and only if both of which exceed the stability limit.

Range: 1 to 1000

Noise Measurement Limit

The `DEF_XXXXCAP_NOISE_LIMIT` parameter is used to select the noise limit value to trigger sensor lockout functionality.

There are two purposes for this parameter:

- If the noise level calculated during a running window exceeds `DEF_XXXXCAP_NOISE_LIMIT`, then the corresponding sensors are declared noisy and sensor global noisy bit is set as '1'.
- If the lockout is enabled, and the noise level calculated during a running window exceeds `DEF_XXXXCAP_NOISE_LIMIT`, then system triggers the sensor lockout functionality.

Range: 1 to 255

Noise Measurement Buffer Count

The `DEF_XXXXCAP_NOISE_MEAS_BUFFER_CNT` parameter is used to select the buffer count for noise measurement buffer.

Range: 3 to 10 (If `N` number of samples differences have to be checked, define this parameter as "`N` + 1")

If N = 4 then set `DEF_XXXXCAP_NOISE_MEAS_BUFFER_CNT` 5u

2.4.6.2 Sensor LockOut Parameters

Sensor Lockout Selection

The `DEF_XXXXCAP_LOCKOUT_SEL` parameter is used to select the lockout functionality method.

- If `DEF_XXXXCAP_LOCKOUT_SEL` is set to `SINGLE_SENSOR_LOCKOUT` and a sensor's noise level is greater than `DEF_XXXXCAP_NOISE_LIMIT`, then corresponding sensor is locked out from touch detection and drifting is disabled.
- If `DEF_XXXXCAP_LOCKOUT_SEL` is set to `GLOBAL_SENSOR_LOCKOUT` and any sensor's noise level is greater than `DEF_XXXXCAP_NOISE_LIMIT`, then all sensors are locked out from touch detection and drifting is disabled.
- If `DEF_XXXXCAP_LOCKOUT_SEL` is set to `NO_LOCKOUT`, then lockout feature is disabled.

Note:

- Global sensors noisy bit will be available for SINGLE_SENSOR_LOCKOUT and GLOBAL_SENSOR_LOCKOUT.
- Global sensors noisy bit will not be available for NO_LOCK_OUT.

Range: 0 to 2

Sensor Lockout Countdown

If the sensor signal moves from noisy to a good condition and stays there for a DEF_XXXXCAP_LOCKOUT_CNTDOWN number of measurements, the sensor is unlocked and sensors are ready for touch detection and drifting is enabled.

Note: This parameter is valid only for global lockout.

Range: 1 to 255

2.4.6.3 Frequency Auto Tune Parameters

Frequency Auto Tune Enable Disable

The DEF_XXXXCAP_FREQ_AUTO_TUNE_ENABLE parameter will enable and disable the frequency auto tune functionality.

This feature is applicable only for FREQ_MODE_HOP.

- 1 - Frequency auto tune will be enabled
- 0 - Frequency auto tune will be disabled

Frequency Auto Tune Signal Stability

The DEF_XXXXCAP_FREQ_AUTO_SIGNAL_STABILITY_LIMIT parameter defines the stability limit of signals for deciding the Frequency auto tune.

Range: 1 to 1000

Frequency Auto Tune In Counter

The DEF_XXXXCAP_FREQ_AUTO_TUNE_IN_CNT parameter is used to trigger the frequency auto tune.

If sensor signal change at each frequency exceeds the value specified as DEF_XXXXCAP_FREQ_AUTO_SIGNAL_STABILITY_LIMIT for DEF_XXXXCAP_FREQ_AUTO_TUNE_IN_CNT, then frequency auto tune will be triggered at this frequency.

Range: 1 to 255

Note: The Frequency Auto Tune feature and related parameters are available only in FREQ_MODE_HOP mode.

2.4.7 Noise Immunity Features

Noise Measurement

Noise is measured on a per-channel basis after each channel acquisition, using historical data on a rolling window of successive measurements. Reported noise to exclude the instance of an applied or removed touch contact, but the noise indication must react sufficiently fast that false touch detection before noise lockout is prevented.

Signal change from sample to sample during the window buffer is compared to the stability limit. Noise is reported only when two changes occur within the window period and both of which exceed the DEF_XXXXCAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT limit.

Noise is calculated using the following algorithm:

```
if (swing count > 2)
{
  Nk = ∑  ((|Sn – Sn-1| > DEF_XXXXCAP_NOISE_MEAS_SIGNAL_STABILITY))? (0): (|Sn-Sn-1|
  -    DEF_XXXXCAP_NOISE_MEAS_SIGNAL_STABILITY)).
}
else
```

```
{
   Nk = 0
}
```

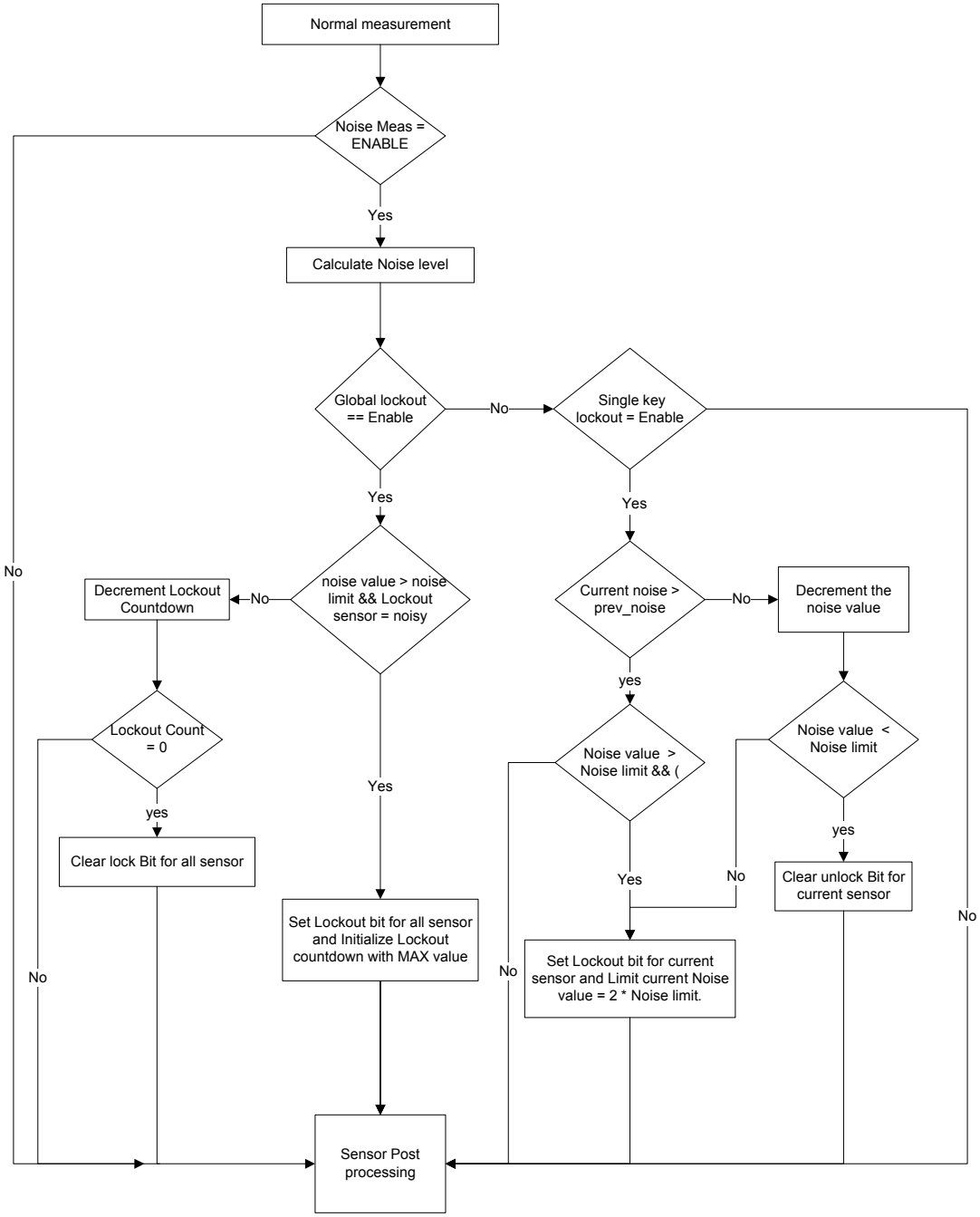
The swing count is number of signal changes that exceed `DEF_XXXXCAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT` limit during buffer window period.

When the measured noise exceeds `DEF_XXXXCAP_NOISE_LIMIT`, the touch library locks out sensors, reports no touch detection and drifting is stopped. Noise measurement is provided for all the channels. Each byte in `p_xxxxcap_measure_data-> p_nm_ch_noise_val` provides the noise level associated with that channel. Noise indication is provided for all the sensors configured by the application. A bit is available in `p_xxxxcap_measure_data-> p_sensor_noise_status` for each sensor to determine whether the sensor is noisy or not. The following code snippet provides the sample code to read the noise status of a particular sensor.

```
If (Double_Inverse_Check is passed on p_xxxxcap_measure_data->
p_sensor_noise_status)
{
   If((GET_XXXXCAP_SENSOR_NOISE_STATUS(SENSOR_NUMBER) ==  0 )
   {
         /* Sensor is stable  */
   }
   Else
   {
         /* Sensor is Unstable */
}
else
{
/* Take fault action on Double inverse check failure */
}
```

Note: Double inverse check must be performed on `p_xxxxcap_measure_data-> p_sensor_noise_status` variable before using those variables.

Figure 2-9. Noise Calculation



2.4.8 Sensor Lockout

This feature locks out the sensors when the measured noise exceeds `DEF_XXXXCAP_NOISE_LIMIT` and does not report a touch. This prevents post-processing. So, the high level of noise cannot cause the channel to drift or recalibrate incorrectly.

Safety library presents two types of lockout features:

Global sensor lockout

When the noise level of a sensor is greater than `DEF_XXXXCAP_NOISE_LIMIT`, all the sensors are locked out from touch detection and drifting is disabled. Sensor signal changes from noisy to a good condition and stays there for a `DEF_XXXXCAP_LOCKOUT_CNTDOWN` number of measurements, the sensor is unlocked for touch detection and also available for post processing.

Single sensor lockout

When the noise level of a sensor is greater than `DEF_XXXXCAP_NOISE_LIMIT`, corresponding sensor is locked out from touch detection and drifting is disabled. Sensor's signal moves from noisy to a good condition and the noise value itself becomes the count-down to clear lockout. The count-out time after a noise spike is proportional to the size of the spike.

2.4.9 Frequency Auto Tune

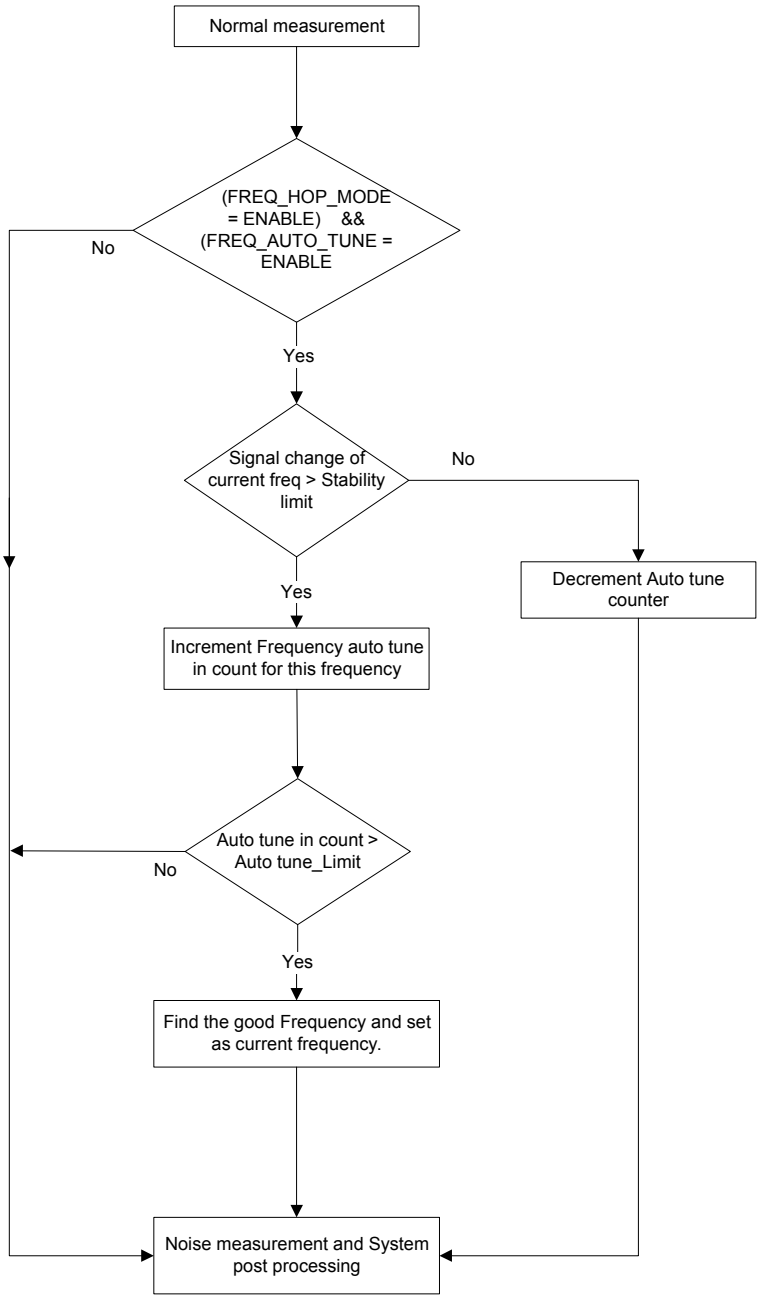
The frequency auto tune feature provides the best quality of signal data for touch detection by automatically selecting acquisition frequencies showing the best SNR in `FREQ_MODE_HOP` mode. During each measurement cycle, the signal change since the last acquisition at the same frequency is recorded for each sensor. After the cycle, when all sensors have been measured at the present acquisition frequency, the largest signal variation of all sensors is stored as the variance for that frequency stage.

The variance for each frequency stage is compared to the `DEF_XXXXCAP_FREQ_AUTO_TUNE_SIGNAL_STABILITY_LIMIT` limit, and if the limit is exceeded, a per-stage counter is incremented. If the measured variance is lower than the limit, the counter is decremented, if it has not been set as zero. If all frequencies display noise exceeding the stability limit, only the counter for the specific frequency stage with the highest variance is incremented after its cycle.

When a frequency counter reaches the `DEF_XXXXCAP_FREQ_AUTO_TUNE_IN_CNT` (auto-tune count in variable), that frequency stage is selected for auto-tuning. A new frequency selection is applied and the counters and variances for all frequencies are reset. After a frequency has been selected for auto-tuning, the count-in for that frequency stage is set to half the original count-in and the process is repeated until either all frequencies have been measured or a frequency is selected which does not re-trigger auto-tuning is determined.

If all frequencies have been tested, and the variation exceeds the `DEF_XXXXCAP_FREQ_AUTO_TUNE_SIGNAL_STABILITY_LIMIT` limit then the frequency with the lowest variance is selected for the frequency stage currently under tuning. The auto-tune process is re-initialized and further tuning does not take place until a frequency stage's high variance counter again reaches the count in limit.

Figure 2-10. Frequency Auto-Tune



2.5 Touch Library Error Reporting Mechanism

The application reports the Touch library errors using one of the two mechanisms:

- Touch Library Error Application Callback mechanism
- API Return Type mechanism

Touch Library Error Application Callback

If any touch library error is generated due to failure in the logical program counter flow or internal library checks, the touch library calls the error callback function registered by the application. If error callback is not registered by the application, the touch library will lock the system in an infinite loop.

The following sample code block registers the touch library error callback:

```
/* registering the callback */
touch_error_app_cb = touch_lib_error_callback;
```

Note: Before calling any touch library API, register the touch library error callback.

For the list of APIs that calls the error call back function, see Section 2.12.2, “Error Codes Returned Through Callback”

API Return Type Mechanism

Few Touch library APIs can return the error synchronously through function call return. For the list of APIs that return the error synchronously, see Section 2.12.1, “Error Codes Returned Synchronously”.

2.6 Touch Library Program Counter Test

The touch library implements two types of tests to verify if the program counter is functioning properly. The logical program tests verifies that the logical sequence of the APIs and processes are appropriate. The program counter test ensures that the program counter is working as expected.

2.6.1 Logical Program Flow Test

There are two sub tests. One test ensures that the mandatory sequence of APIs is followed as illustrated in Figure 2-4. The second test tracks various internal processes by maintaining a unique counter for each process. Any error in the logical sequence causes error callback function to be called with error status as `TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR`.

Figure 2-11. Example Sequence for Logical Program Flow Error

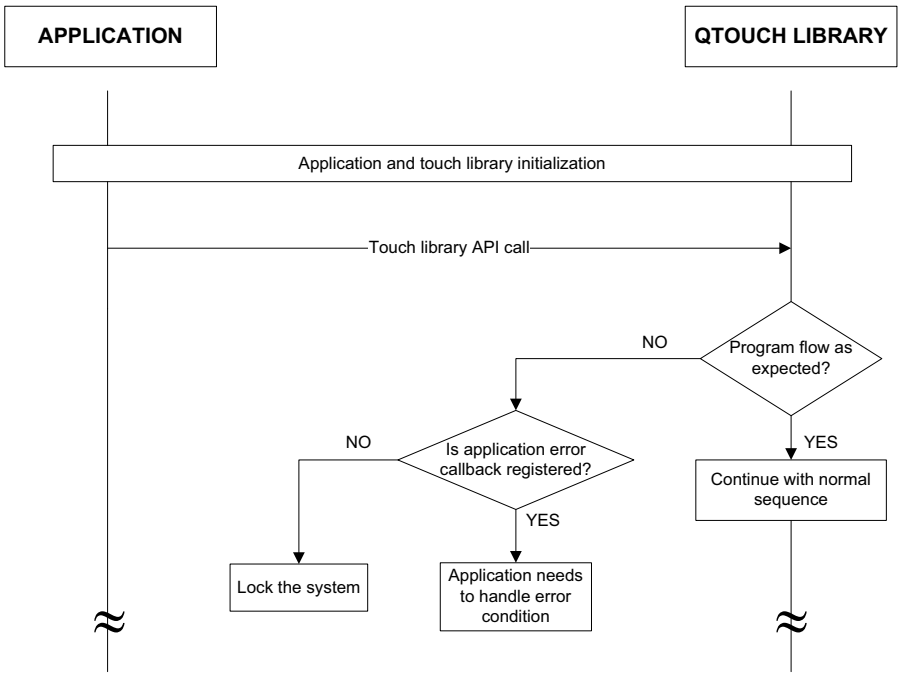
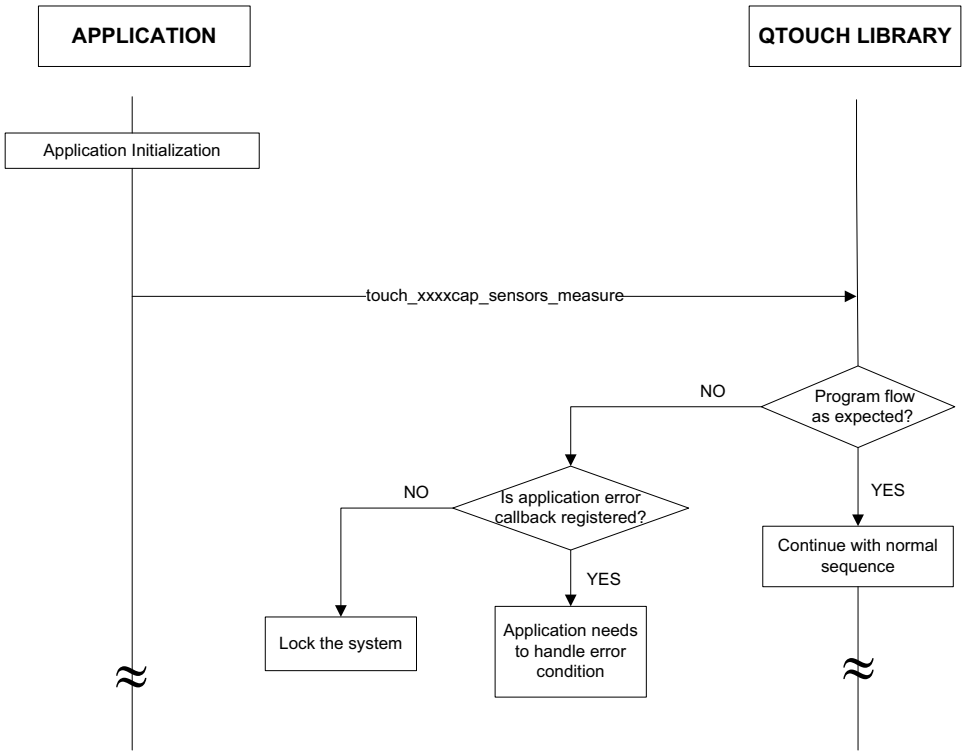


Figure 2-12. Example of a Wrong API Sequence



2.6.2 Program Counter Test

This is another mechanism using which Program Counter can tested. To test the branching, the following program counter API are provided within the touch library at different flash locations:

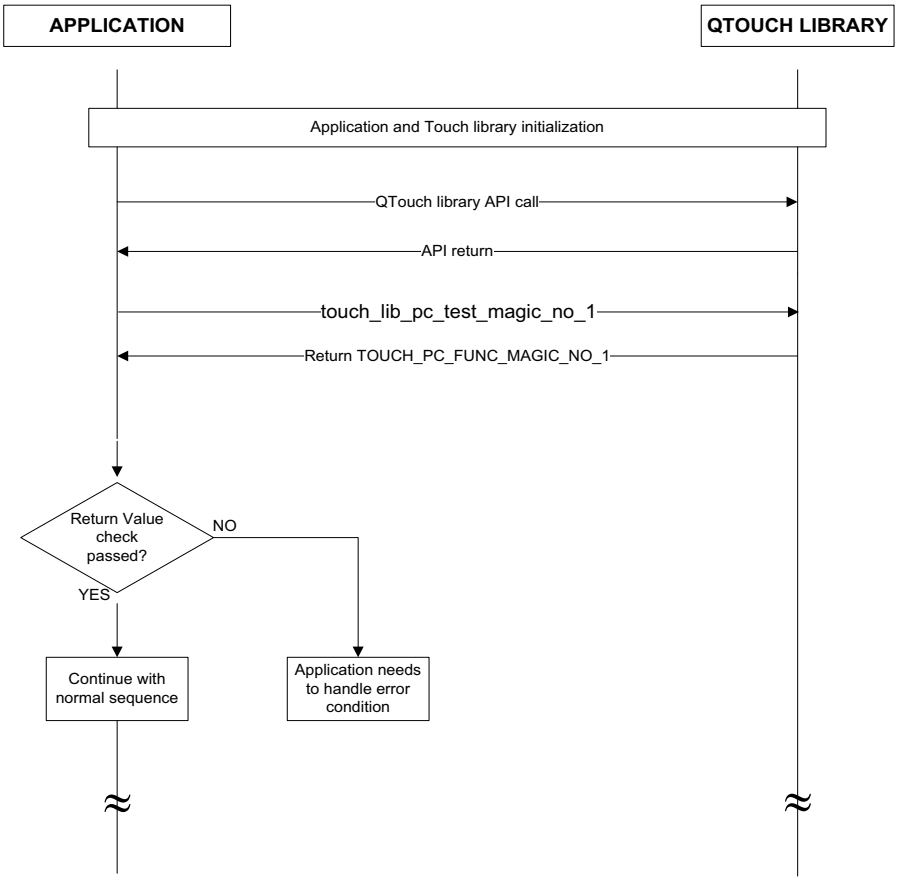
- `touch_lib_pc_test_magic_no_1`
- `touch_lib_pc_test_magic_no_2`
- `touch_lib_pc_test_magic_no_3`
- `touch_lib_pc_test_magic_no_4`

The application calls these API and check the returned value. Each of these API returns a unique value. Hence it is possible to check if the program counter has jumped to the correct address within the touch library by verifying the unique value it returns. If the expected return value is not returned the application must handle error condition.

Note: Ensure that the program counter can branch throughout the touch library. This program counter test is applicable only for checking the program counter validity within the touch library.

The following figure illustrates the implementation of the program counter APIs.

Figure 2-13. Program Counter Test Using Program Counter APIs



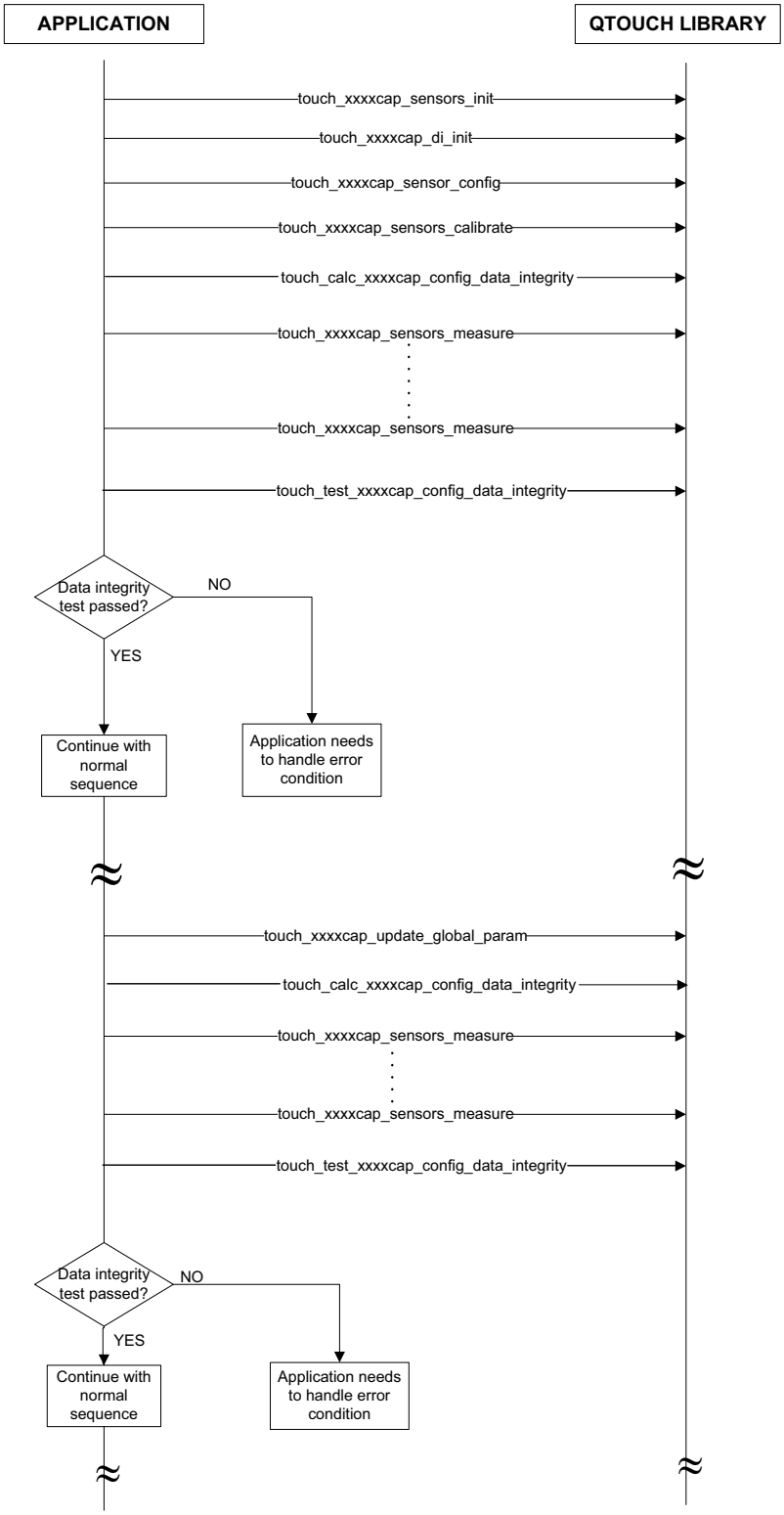
2.7 CRC on Touch Input Configuration

The data integrity check is performed on the input configuration variables from application to Touch Library.

The application calls the `touch_calc_xxxxcap_config_data_integrity` API, if the input configuration variables has been modified. The `touch_test_xxxxcap_config_data_integrity` API must be called to test the input configuration data integrity. The periodicity of calling this API can be decided by the application.

Note: The `touch_calc_xxxxcap_config_data_integrity` API must be called after initialization sequence. The following illustration depicts the sequence for verifying the data integrity.

Figure 2-14. Data Integrity Check Sequence



The following APIs modifies the input configuration and hence `touch_calc_xxxxcap_config_data_integrity` must be called only after calling these APIs.

- `touch_xxxxcap_update_global_param`
- `touch_xxxxcap_sensor_update_acq_config`
- `touch_xxxxcap_sensor_update_config`
- `touch_xxxxcap_cnfg_mois_threshold`
- `touch_xxxxcap_cnfg_mois_mltchgrp`
- `touch_xxxxcap_mois_tolrnce_enable`
- `touch_xxxxcap_mois_tolrnce_disable`

Notes:
1. `touch_calc_xxxxcap_config_data_integrity` and `touch_test_xxxxcap_config_data_integrity` should be called only when touch library state is `TOUCH_STATE_INIT` or `TOUCH_STATE_READY`.
2. If calibration of all channels is requested by application with `AUTO_TUNE_PRSC` or `AUTO_TUNE_RSEL` option, QTouch Safety Library will automatically recalculate the CRC at the end of auto tuning calibration process. If there is any fault, library will report error as `TOUCH_LIB_CRC_FAIL` through errror callback, even before application calls `touch_test_xxxxcap_config_data_integrity` API.

2.8 Double Inverse Memory Check

It is important to check the critical safety data before the application uses such data. Checking each critical data before using it prevents any system malfunction.

Double inverse memory check is a mechanism that stores and retrieve data with additional redundancy. Reading and writing redundant data requires some processing and additional memory requirement. Hence, this mechanism is suggested only for the most important safety critical data in the FMEA and QTouch Safety Library.

The inverse of all the critical data interface variables used among the application and touch library is stored in the structure variable `touch_lib_xxxxcap_param_safety`. The mechanism stores the inverse of the critical data in this structure. Before reading and writing the critical data, the authenticity of the critical data is verified.

All double inverse variables are part of the `touch_lib_param_safety_t` structure. These double inverse variables are inverse value of various variables selected from different structure variables. The application must perform the double inverse check whenever it attempts to read or write a critical data interface variables.

2.8.1 Application to Touch Library

The application must calculate the inverse for all the variables listed in the column *Variable* and store it as the corresponding inverse variable listed in the column *Inverse Variable*.

Touch library checks for double inversion between the variables listed in the *Inverse Variable* column and *Variable* column. If the verification is successful, touch library operation continues as expected.

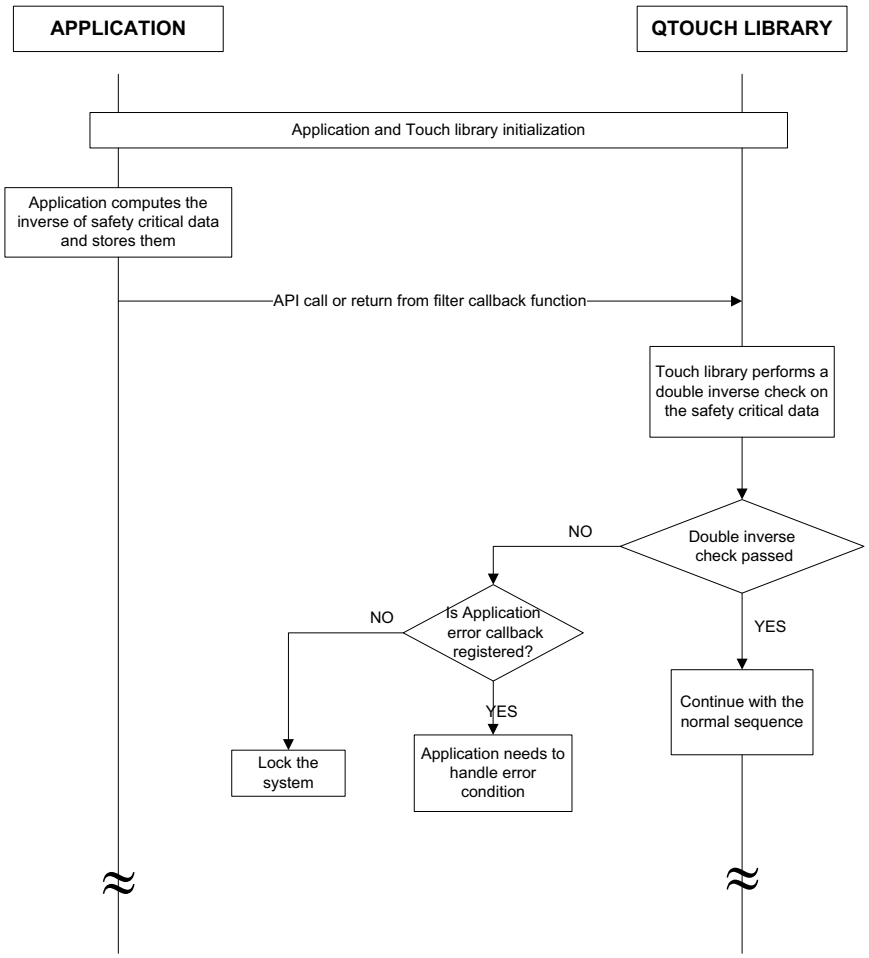
If the verification is unsuccessful, the touch library calls the error callback function `touch_error_app_cb` indicating the reason `TOUCH_LIB_DI_CHECK_FAIL.`

The following table provides the list of variables and the corresponding inverse variable for which the application must add double inverse protection.

Table 2-4. Inverse Variable Details - Application to Touch Library

Variable	Inverse Variable	Description
`p_channel_signals`	`P_inv_channel_signals`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`current_time_ms`	`inv_current_time_ms`	Refer Section 3.4.7 for variable and Section 3.4.2 for corresponding inverse variable.
`burst_again`	`inv_burst_again`	Refer Section 2.9 for variable and Section 3.4.2 for corresponding inverse variable.
`acq_mode`	`inv_acq_mode`	Refer Section 3.3.8 for variable and Section 3.4.2 for corresponding inverse variable.

Figure 2-15. Example Sequence for Processing Double Inverse Variable (Application to QTouch Safety Library)



2.8.2 Touch Library to Application

The touch library must calculate the inverse for all the variables listed in the column *Variable* and store it as the corresponding inverse variable listed in the column *Inverse Variable*.

Application must check for double inversion between the variables listed in the *Inverse Variable* column and *Variable* column. Appropriate action must be performed by the application if double inversion check fails.

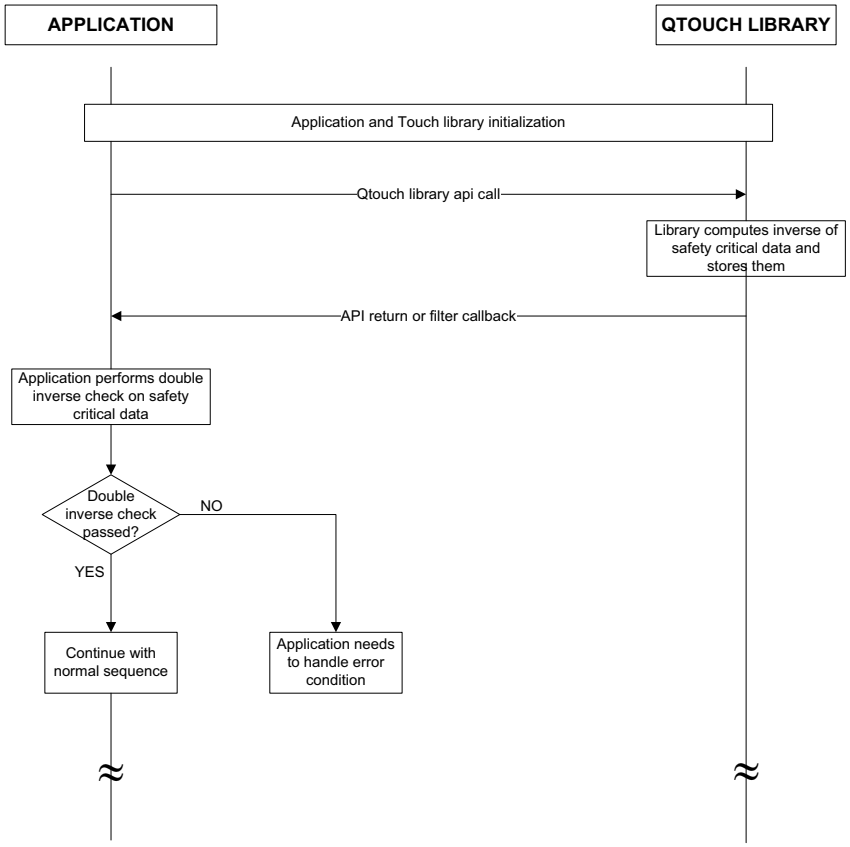
The following table lists the variables and the corresponding inverse variable for which the touch library will add double inverse protection.

Table 2-5. Inverse Variable Details Touch Library to Application

Variable	Inverse Variable	Description
`p_channel_signals`	`p_inv_channel_signals`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`acq_status`	`inv_acq_status`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`num_channel_signals`	`inv_num_channel_signals`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`num_sensor_states`	`p_inv_sensor_states`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`p_sensor_states`	`inv_num_sensor_states`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`num_rotor_slider_values`	`inv_num_rotor_slider_values`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`p_rotor_slider_values`	`p_inv_rotor_slider_values`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`lib_state`	`inv_lib_state`	Refer Section 3.4.8 for variable and Section 3.4.2 for corresponding inverse variable.
`delta`	`inv_delta`	Refer Section 3.4.8 for variable and Section 3.4.2 for corresponding inverse variable.
`sf_ptc_error_flag`	`inv_sf_ptc_error_flag`	This variable is used by FMEA and should not be used by the application.
`cc_cal_open_calibration_vals`	`inv_cc_cal_open_calibration_vals`	This variable is used by FMEA and should not be used by the application.
`p_sensor_noise_status`	`p_inv_sensor_noise status`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`p_sensor_mois_status`	`p_inv_sensor_mois_status`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.
`p_auto_os_status`	`p_inv_chan_auto_os_status`	Refer Section 3.4.5 for variable and Section 3.4.2 for corresponding inverse variable.

Note: `p_channel_signals` variable must be double inversed by both the application and the touch library. The application can apply filtering mechanism on the channel signals in the filter callback function. The application must check for the double inversion before modifying the channel signals. After modifying the channel signals, the application would store the value of the channel signals into the `p_inv_channel_signals` variable. The Touch Library after returning from the filter callback function, would re-check for double inversion on the channel signals.

Figure 2-16. Example Sequence for Processing Double Inverse Variable



2.9 Application Burst Again Mechanism

The completion of a touch measurement is indicated by the touch library by calling the function `touch_xxxxcap_measure_complete_callback()`.The complete callback function will be called on completion of the previously initiated touch measurement.

The application can call the touch measurement again based on touch measurement periodicity or initiate the next measurement immediately by returning a value `1` in the `touch_xxxxcap_measure_complete_callback()` function. The touch library will initiate the next measurement immediately if application returns a value 1 when the complete callback function is called and the internal burst again flag is set by the library.

If the application returns `0`, the touch library waits for another touch measurement to be initiated by the application by calling `touch_xxxxcap_sensors_measure()` to perform another touch measurement. Refer Figure 2-4 for more information.

2.10 Memory Requirement

The table provided in this section provides the typical code and data memory required for QTouch Safety Library.

Mutual and self capacitance measurement method requires additional data memory for the application to store the signals, references, sensor configuration information, and touch status. This data memory is provided by the application as *data block* array. The size of this data block depends on the number of Channels, sensors and rotor sliders configured.

Default Configuration Used For Memory Requirement Calculations:

Apart from the various combinations mentioned in Section 2.10.1. The default configuration details used in all the cases applicable for memory calculation in Section 2.10.1 are mentioned in the following table.

Table 2-6. Default Configuration

CONFIGURATION	MUTLCAP	SELFCAP
DEF_XXXXCAP_NOISE_MEAS_ENABLE	1	1
DEF_XXXXCAP_FREQ_AUTO_TUNE_ENABLE	1	1
DEF_XXXCAP_NOISE_MEAS_BUFFER_CNT	5	5
DEF_XXXCAP_MOIS_TOLERANCE_ENABLE	1	1
DEF_XXXCAP_NUM_MOIS_GROUPS	8	8

2.10.1 Memory Requirement for IAR Safety library

Table 2-7. Memory Requirement for Mutual Capacitance

Total No of Channels	No of Keys	No of rotor/slider	Total Code Memory	Total Data Memory
1	1	0	23275	1720
10	10	0	23296	2180
10	2	2	24925	2172
20	20	0	23286	2664
20	10	2	24913	2640
40	40	0	23255	3664
256	20	5	24884	3620
256	256	0	23185	14568
256	200	14	24812	14400

Table 2-8. Memory Requirement Self Capacitance

Total No of Channels	No of Keys	No of rotor/slider	Total Code Memory	Total Data Memory
1	1	0	22884	1700
2	2	0	22887	1744
11	11	0	22880	2188
11	2	3	24412	2232
16	16	0	22868	2412
16	4	4	24398	2468

Table 2-9. Memory Requirement (Self + Mutual) Capacitance

Total No of Mutual Cap Channels	Total No of SelfCap Channels	Total No of Mutual Cap Keys	Total No of Self Cap Keys	Total No of Mutual Cap Rotor Sliders	Total No of Self Cap Rotor Sliders	Total Code Memory	Total Data Memory
1	1	1	1	0	0	28556	2080
40	8	40	8	0	0	28510	4356
40	8	40	2	0	2	30042	4392
40	8	24	8	3	0	30138	4300
40	8	8	2	3	2	31669	4336
80	11	80	11	0	0	28522	6548
80	11	80	2	0	3	30054	6592
80	11	48	11	6	0	30148	6412
80	11	48	2	6	3	31680	6456

2.11 API Execution Time

2.11.1 Mutual Capacitance API Execution Time

This section provides the time required for various mutual capacitance APIs. The values provided are based on the following system configuration:

- CPU Frequency: 48MHz
- PTC Frequency: 4MHz
- No of Channels: 20
- No of Sensors: 10

- No of Keys: 8
- No of Rotor/sliders: 2

Table 2-10. Default Configuration - Mutual Capacitance

CONFIGURATION	MUTLCAP
DEF_XXXXCAP_NOISE_MEAS_ENABLE	1
DEF_XXXXCAP_FREQ_AUTO_TUNE_ENABLE	1
DEF_XXXCAP_NOISE_MEAS_BUFFER_CNT	5
DEF_XXXCAP_MOIS_TOLERANCE_ENABLE	1
DEF_XXXCAP_NUM_MOIS_GROUPS	8

Table 2-11. Execution Time for Various QTouch Safety Library APIs - Mutual Capacitance

API	Time	Units
touch_mutlcap_sensors_init	412.7	µs
touch_mutlcap_di_init	14.04	µs
touch_mutlcap_sensor_config	19.83	µs
touch_mutlcap_sensors_calibrate	210.6*	ms
touch_mutlcap_calibrate_single_sensor	24.84*	ms
touch_mutlcap_sensors_measure	17.1*	ms
touch_calc_mutlcap_config_data_integrity	1250	µs
touch_test_mutlcap_config_data_integrity	1250	µs
touch_mutlcap_sensor_get_delta	10.79	µs
touch_mutlcap_sensor_update_config	8.48	µs
touch_mutlcap_sensor_get_config	6.59	µs
touch_mutlcap_sensor_update_acq_config	60.05	µs
touch_mutlcap_sensor_get_acq_config	34.55	µs
touch_mutlcap_update_global_param	8.04	µs
touch_mutlcap_get_global_param	6.25	µs
touch_mutlcap_get_libinfo	6.5	µs
touch_lib_pc_test_magic_no_1	3.67	µs
touch_lib_pc_test_magic_no_2	3.39	µs
touch_lib_pc_test_magic_no_3	3.39	µs
touch_lib_pc_test_magic_no_4	3.39	µs
touch_mutlcap_cnfg_mois_mltchgrp	6.06	µs
touch_mutlcap_cnfg_mois_threshold	6.19	µs

Table 2-11. Execution Time for Various QTouch Safety Library APIs - Mutual Capacitance

API	Time	Units
touch_mutlcap_mois_tolrnce_enable	4.56	µs
touch_mutlcap_mois_tolrnce_disable	7.72	µs
touch_mutlcap_sensor_reenable	24.17	µs
touch_mutlcap_sensor_disable	14.67	µs
touch_library_get_version_info	4.35	µs
touch_suspend_ptc	2.0	ms
touch_resume_ptc	8.0	µs
touch_disable_ptc	5.0	µs
touch_enable_ptc	5.0	µs
touch_mutlcap_sensors_deinit	183.8	µs

Notes:
1. The Table 2-11 provides the maximum time required for the touch_mutlcap_sensors_calibrate, touch_mutlcap_calibrate_single_sensor, touch_mutlcap_sensors_measure, and touch_suspend_ptc API to complete the procedure. The time required for the API to return control to the application will be much shorter than the time specified in the Table 2-11. After the control is returned back to the application, the application can execute other non-touch related tasks.
2. API Execution time marked as * are calculated for sensors mentioned in Section 2.11.1 with typical sensor capacitance values.

Table 2-12. Timings for APIs to Return Control to the Application

API	Time	Units
touch_mutlcap_sensors_calibrate	151.9	µs
touch_mutlcap_calibrate_single_sensor	17.79	µs
touch_mutlcap_sensors_measure	85.7	µs
touch_suspend_ptc	4.2	µs

2.11.2 Self Capacitance API Execution Time

This section provides the time required for various self capacitance APIs. The values provided are based on the following system configuration:

- CPU Frequency: 48MHz
- PTC Frequency: 4MHz
- No of Channels: 16
- No of Sensors: 8
- No of Keys: 4
- No of Rotor/sliders: 4

Table 2-13. Default Configuration - Self Capacitance

CONFIGURATION	SELFCAP
DEF_XXXXCAP_NOISE_MEAS_ENABLE	1
DEF_XXXXCAP_FREQ_AUTO_TUNE_ENABLE	1
DEF_XXXCAP_NOISE_MEAS_BUFFER_CNT	5
DEF_XXXCAP_MOIS_TOLERANCE_ENABLE	1
DEF_XXXCAP_NUM_MOIS_GROUPS	8

Table 2-14. Execution Time for Various QTouch Safety Library APIs - Self Capacitance

API	Time	Units
touch_selfcap_sensors_init	313.7	µs
touch_selfcap_di_init	12	µs
touch_selfcap_sensor_config	18.35	µs
touch_selfcap_sensors_calibrate	566.7*	ms
touch_selfcap_calibrate_single_sensor	73.89*	ms
touch_selfcap_sensors_measure	47.95*	ms
touch_calc_selfcap_config_data_integrity	1041	µs
touch_test_selfcap_config_data_integrity	1040	µs
touch_selfcap_sensor_get_delta	10.54	µs
touch_selfcap_sensor_update_config	7.47	µs
touch_selfcap_sensor_get_config	6.1	µs
touch_selfcap_sensor_update_acq_config	19.62	µs
touch_selfcap_sensor_get_acq_config	29.54	µs
touch_selfcap_update_global_param	8.18	µs
touch_selfcap_get_global_param	5.97	µs
touch_selfcap_get_libinfo	6.39	µs
touch_lib_pc_test_magic_no_1	3.67	µs
touch_lib_pc_test_magic_no_2	3.53	µs
touch_lib_pc_test_magic_no_3	3.39	µs
touch_lib_pc_test_magic_no_4	3.39	µs
touch_selfcap_cnfg_mois_mltchgrp	5.59	µs
touch_selfcap_cnfg_mois_threshold	6.1	µs
touch_selfcap_mois_tolrnce_enable	4.74	µs
touch_selfcap_mois_tolrnce_disable	7.28	µs

Table 2-14. Execution Time for Various QTouch Safety Library APIs - Self Capacitance

API	Time	Units
`touch_selfcap_sensor_reenable`	24.17	µs
`touch_selfcap_sensor_disable`	14.63	µs
`touch_library_get_version_info`	4.2	µs
`touch_selfcap_sensors_deinit`	155.8	µs

Notes:
1. The Table 2-14 provides the maximum time required for the `touch_selfcap_sensors_calibrate`, `touch_selfcap_calibrate_single_sensor`, and `touch_selfcap_sensors_measure` API to complete the procedure. The time required for the API to return control to the application will be much shorter than the time specified in the Table 2-14. After the control is returned back to the application, the application can execute other non-touch related tasks.
2. API Execution Time marked as * are calculated for sensors mentioned in Section 2.11.2, “Self Capacitance API Execution Time” with typical sensor capacitance values.

Table 2-15. Timings for APIs to Return Control to the Application

API	Time	Units
touch_selfcap_sensors_calibrate	138.2	µs
touch_selfcap_calibrate_single_sensor	13.73	µs
touch_selfcap_sensors_measure	139.9	µs

2.12 Error Interpretation

This section provides information about the error bits that indicate the errors and the specific reason that causes the errors.

2.12.1 Error Codes Returned Synchronously

The following table provides the error codes returned by various touch APIs synchronously through function call return.

Table 2-16. Error Codes Returned Synchronously

API	Error Bit	Reason
touch_xxxxcap_sensors_init	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_MUTLCAP_CONFIG_PARAM	Configuration parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
	TOUCH_INVALID_RECAL_THRESHOLD	Recalibration threshold is invalid.
touch_xxxxcap_di_init	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
touch_xxxxcap_sensor_config	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
	TOUCH_INVALID_SENSOR_TYPE	Sensor type is invalid.
	TOUCH_INVALID_CHANNEL_NUM	Channel number is invalid.
	TOUCH_INVALID_RS_NUM	Invalid rotor slider number.
touch_xxxxcap_sensors_calibrate	TOUCH_INVALID_LIB_STATE	Library state is invalid.
	TOUCH_CNFG_MISMATCH	Configuration mismatch error.
touch_xxxxcap_calibrate_single_sensor	TOUCH_INVALID_LIB_STATE	Library state is invalid.
	TOUCH_INVALID_SENSOR_ID	Sensor ID is invalid.
touch_xxxxcap_sensors_measure	TOUCH_ACQ_INCOMPLETE	Acquisition is in progress.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
	TOUCH_ALL_SENSORS_DISABLED	All sensors are disabled.
touch_xxxxcap_sensor_get_delta	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.

Table 2-16. Error Codes Returned Synchronously

API	Error Bit	Reason
touch_xxxxcap_sensor_update_config	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_sensor_get_config	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
	TOUCH_INVALID_SENSOR_ID	Sensor ID is invalid.
touch_xxxxcap_update_global_param	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_RECAL_THRESHOLD	Recalibration threshold is invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_get_global_param	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_sensor_update_acq_config	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_sensor_get_acq_config	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_get_libinfo	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
touch_xxxxcap_sensor_reenable	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_sensor_disable	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_INVALID_LIB_STATE	Library state is invalid.
touch_xxxxcap_cnfg_mois_mltchgrp	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
touch_xxxxcap_cnfg_mois_threshold	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
touch_xxxxcap_mois_tolrnce_enable	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_CNFG_MISMATCH	Configuration mismatch error
touch_xxxxcap_mois_tolrnce_disable	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid.
	TOUCH_CNFG_MISMATCH	Configuration mismatch error
touch_library_get_version_info	TOUCH_INVALID_INPUT_PARAM	Input parameters are invalid
touch_suspend_ptc	TOUCH_INVALID_INPUT_PARAM	Suspend application callback is not registered
	TOUCH_INVALID_LIB_STATE	PTC is already suspended
touch_resume_ptc	TOUCH_INVALID_LIB_STATE	PTC is already resumed

Table 2-16. Error Codes Returned Synchronously

API	Error Bit	Reason
touch_calc_xxxxcap_config_data_integrity	TOUCH_INVALID_LIB_STATE	Library state is invalid.Should be called when library state is TOUCH_STATE_INIT or TOUCH_STATE_READY.
touch_test_xxxxcap_config_data_integrity	TOUCH_INVALID_LIB_STATE	Library state is invalid.Should be called when library state is TOUCH_STATE_INIT or TOUCH_STATE_READY.
touch_xxxxcap_sensors_deinit	TOUCH_INVALID_LIB_STATE	Library state is invalid.Should be called when library state is TOUCH_STATE_INIT or TOUCH_STATE_READY.

2.12.2 Error Codes Returned Through Callback

The following table provides the list of APIs and the associated error codes that results in touch_library_error_callback being called.

Table 2-17. API Error Codes Returned through Callback

API	Error Bit	Reason
touch_xxxxcap_sensor_init	TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR	Logical program counter flow error.
	TOUCH_PINS_VALIDATION_FAIL	Touch library Pins Invalid.
touch_xxxxcap_sensor_config	TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR	Logical program counter flow error.
touch_xxxxcap_di_init	TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR	Logical program counter flow error.
touch_xxxxcap_sensors_calibrate	TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR	Logical program counter flow error.
touch_xxxxcap_calibrate_single_sensor	TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR	Logical program counter flow error.
touch_xxxxcap_sensors_measure	TOUCH_LOGICAL_PROGRAM_CNT_FAIL	Logical program counter flow error.
	TOUCH_LIB_DI_CHECK_FAIL	Double inverse check failed.
	TOUCH_LIB_CRC_FAIL	CRC failure during Calibration process.
touch_test_xxxxcap_config_data_integrity	TOUCH_LIB_CRC_FAIL	CRC check failed.

2.13 Data and Function Protection

The functions and global variables that are used only by Touch Library are marked as static. The user / application must not change these variable to non-static.

The header file `touch_fmea_api_samd.h` file is used only by FMEA. Hence, the application should not include the same in any file.

Table 2-18. API Header File Details

Header File	Availability for Application
`touch_safety_api_samd.h`	Yes
`touch_fmea_api_samd.h`	Yes

2.14 Moisture Tolerance

Moisture tolerance check executes at the end of each measurement cycle and compares the sum of delta of all sensors in a moisture tolerance group against pre-configured threshold. If delta sum is greater than sensor moisture lock threshold and less than system moisture lock threshold, then the ON-state sensors within moisture tolerance group will be considered as moisture affected.

If delta sum is greater than system moisture lock threshold, all sensors within the moisture tolerance group will be considered as moisture affected. This condition is referred as moisture global lock out. The safety library will come out of the moisture global lock out state when delta sum is less than threshold for 5 consecutive measurements. Self cap and mutual cap sensors cannot be configured in a single moisture group, Self cap moisture tolerance and mutual cap Moisture tolerance features can be enabled or disabled separately.

2.14.1 Moisture Tolerance Group

This feature enables the customer application to group a set of sensors in to single moisture tolerance group. If moisture on one sensor might affect other sensors due to physical proximity, they must be grouped together into one Moisture tolerance group.

Using this feature the application can disable moisture tolerance detection for a set of sensors, Multiple Moisture tolerance groups can be formed by the customer application. The library supports up to a maximum of 8 moisture groups.

Note: Changing the moisture tolerance group configuration during runtime is not recommended. However, muti-touch group configuration can be changed during runtime.

2.14.2 Multi-touch Group

If the user wants to touch multiple sensors within the moisture tolerance group simultaneously to indicate a specific request, then the application should configure those sensors into single multi-touch group. Multiple multi-touch groups can be formed by the customer application. The library supports a maximum of 8 multi-touch groups within a single moisture tolerance group.

Moisture tolerance feature improves a system's performance under the following scenarios:

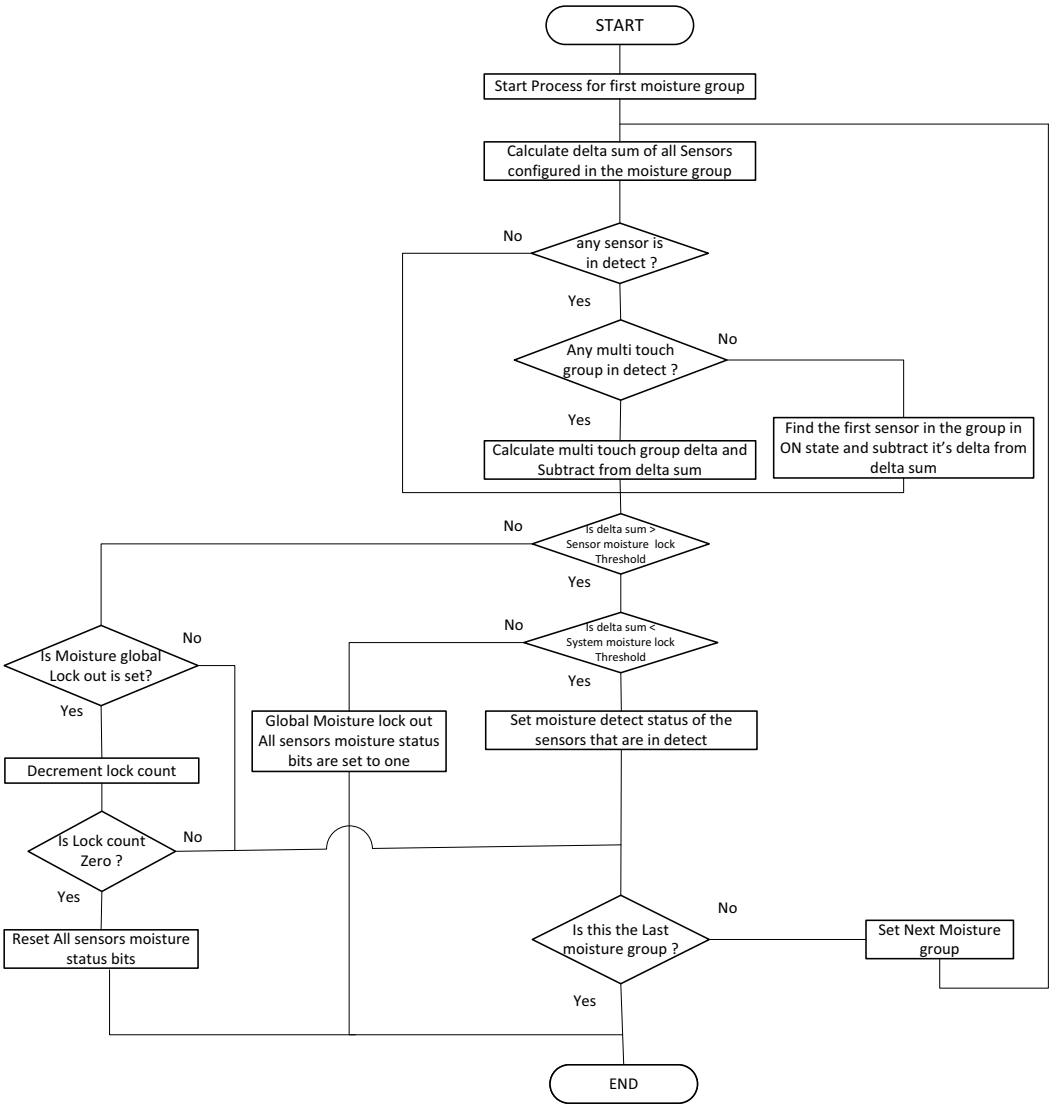
- Droplets of water sprayed on the front panel surface
- Heavy water poured on the front panel surface
- Large water puddle on multiple sensors
- Trickling water on multiple sensors

Moisture tolerance feature is not expected to offer any significant performance improvement under the following scenarios:

- Large isolated puddle on single sensor
- Direct water pour on single sensor

Within the same moisture group, user should not configure all the sensors to the single multi-touch group.

Figure 2-17. Moisture Tolerance Algorithm



2.15 Quick Re-burst

This feature allows faster resolution of a sensor's state during DI filtering. If Sensor-N is touched by the user, then any other sensor that meets one of the following criteria is selected for the measurement in the next cycle:

- Same AKS group as Sensor-N
- Same Moisture tolerance group Sensor-N

If quick re-burst feature is disabled, then all sensors would be measured in every measurement cycle.

2.15.1 Synchronizing Quick Re-burst and Application Burst again

Table 2-19. Quick Re-burst - Triggers and Sensors

Quick Re-burst Status	Measurement Trigger	List of Sensors Measured
Enabled	`touch_xxxxcap_sensors_measure()`	All
Enabled	Application Burst Again	Sensors that are touched and their AKS and moisture tolerance group sensors
Disabled	`touch_xxxxcap_sensors_measure()`	All
Disabled	Application burst again	All

2.16 Reading Sensor States

When noise immunity and moisture tolerance features are enabled the validity of the sensor sate is based on the moisture status and noise status. Refer to Section 2.4.7 and Section 2.14 for information on noise immunity and moisture tolerance status of sensors. The state of a sensor is valid only when the sensor is not affected by noise and moisture. If a sensor is noisy or affected by moisture, then the state of sensor must be considered as OFF. The code snippet below depicts the same for mutual-cap sensors.

When a sensor is touched or released during DI, library will burst on channels corresponding to sensors whose state is other than OFF or DISABLED. If any sensor in an AKS group is in a state other than OFF or DISABLED, the library will burst channels corresponding sensors belong to that AKS group. If a sensor in any moisture group is in a state other than OFF or DISABLED, the library will burst on channels corresponding to sensors belonging to that moisture group.

```
If(! (GET_MUTLCAP_SENSOR_NOISE_STATUS(SENSOR_NUMBER)))
{
      If(! (GET_MUTLCAP_SENSOR_MOIS_STATUS (SENSOR_NUMBER)))
      {
            /*Sensor state is valid Read sensor state */
      } else
      {
            /* Sensor is Moisture affected*/
      }

}else
{
      /* Sensor is noisy */
}
```

2.17 Touch Library Suspend Resume Operation

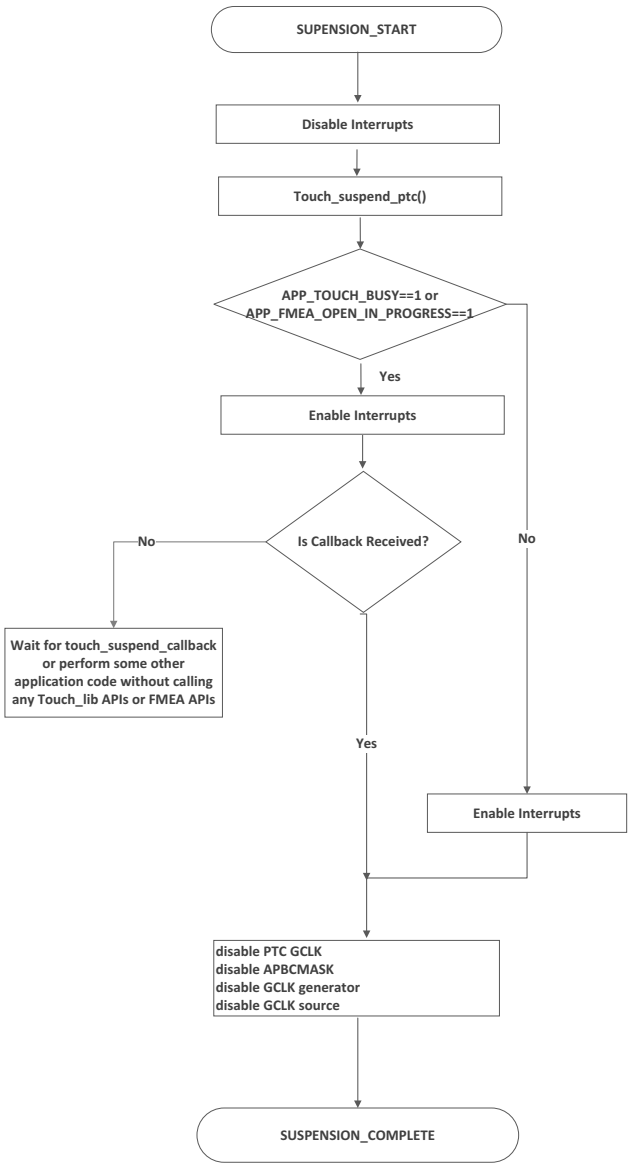
The touch library provides `touch_suspend_ptc`,`touch_resume_ptc` API to suspend and resume the PTC.

When suspend API is called, the touch library initiates the suspend operation and return to the application.After completing the current PTC conversion, the touch library will initiate suspend operation and call the application touch suspend callback function pointer. The suspend complete callback function pointer has to be registered by the application (Refer Section 3.5.3 for more details).

Note: The application then should disable the corresponding PTC clock to reduce the power consumption.`APP_TOUCH_BUSY` and `APP_FMEA_OPEN_IN_PROGRESS` needs to be maintained by the application. The `APP_TOUCH_BUSY` will be set to 1 until the completion of following APIs as mentioned in Table 2-15. The `APP_FMEA_OPEN_IN_PROGRESS` will be set to 1 until the completion of API mentioned in Table 4-4.

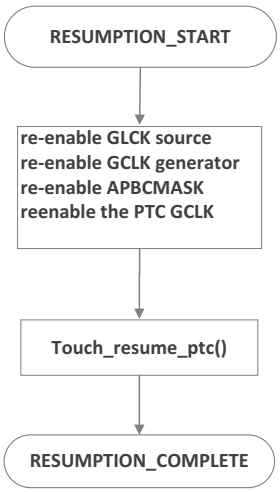
The following flowchart depicts the suspend sequence.

Figure 2-18. Suspension Sequence



The following flowchart depicts the resume sequence.

Figure 2-19. Resumption Sequence



Note: The suspend and resume operation must be followed as specified in Section 2.17, otherwise the touch library may not behave as expected.
Once the suspend API is called, the touch library resumption should happen before calling any other API's.

2.18 Drifting on Disabled Sensors

Touch Safety library performs drifting on disabled sensors. Drifting for disabled sensors would function in a same way, as drifting happens on a sensor which is in 'OFF' state. Hence, drift configuration settings which are applicable for 'OFF' state sensors would be applicable for disabled sensors also.

When a sensor is touched, it goes to 'ON' state and if it is disabled in this condition, drifting will adjust the reference to unintentional signal value. Hence for drifting on disabled sensor to function properly, following conditions has to be ensured before that sensor is disabled.

- The state of that particular sensor should be 'OFF'.
- `TOUCH_BURST_AGAIN`' bit field in '`p_xxxxcap_measure_data->acq_status`' should be '0'. Refer Section 3.6.13, “Touch Library Enable Disable Sensor”.

Note:

a. It is recomended to re-enable the sensors periodically so that drifting could be done with respect to latest signal values and reference would be adjusted with respect to latest signal values. In other case, if sensors are re-enabled after a long duration, they can be re-enabled with calibration option(`no_calib` = 0).
b. Drifting on Disabled sensors functionality would be applicable if sensors are re-enabled without calibration.If sensors are re-enabled with calibration, then reference would be adjusted as part of calibration process itself.

3. QTouch Safety Library API

3.1 Typedefs

Keyword	Type	Description
threshold_t	uint8_t	An unsigned 8-bit number setting a sensor detection threshold.
sensor_id_t	uint8_t	Sensor number type.
touch_current_time_t	uint16_t	Current time type.
touch_delta_t	int16_t	Touch sensor delta value type.
touch_acq_status_t	uint16_t	Status of touch measurement.

3.2 Macros

3.2.1 Touch Library Acquisition Status Bit Fields.

Keyword	Type	Description
TOUCH_NO_ACTIVITY	0x0000u	No touch activity
TOUCH_IN_DETECT	0x0001u	At least one touch channel is in detect.
TOUCH_STATUS_CHANGE	0x0002u	Change in touch status of at least one Touch channel.
TOUCH_ROTOR_SLIDER_POS_CHANGE	0x0004u	Change in the position of at least one rotor or slider.
TOUCH_CHANNEL_REF_CHANGE	0x0008u	Change in the reference value of at least one touch channel.
TOUCH_BURST_AGAIN	0x0100u	Indicates that re-burst is required to resolve filtering or calibration state.
TOUCH_RESOLVE_CAL	0x0200u	Indicates that re-burst is required to resolve calibration process.
TOUCH_RESOLVE_FILTERIN	0x0400u	Indicates that re-burst is required to resolve calibration.
TOUCH_RESOLVE_DI	0x0800u	Indicates that re-burst is needed to resolve Detect Integration.
TOUCH_RESOLVE_POS_RECAL	0x1000u	Indicates that re-burst is needed to resolve away from touch recalibration.
TOUCH_CC_CALIB_ERROR	0x2000u	Indicates that CC Calibration error has occurred.
TOUCH_AUTO_OS_IN_PROGRESS	0x4000u	Indicates that Auto Oversample process is going on.

DEF_TOUCH_MUTLCAP must be set to 1 in the application to enable the Mutual Capacitance touch technology. DEF_TOUCH_SELFCAP must be set to 1 in the application to enable the Self Capacitance touch technology.

TOUCH_SAFETY_COMPILE_CHECK must be set to 1 to enable the compile time check feature.

3.2.2 Sensor State Configurations.

GET_SENSOR_STATE (SENSOR_NUMBER)

To get the sensor state (whether detect or not). These values are valid for parameter that corresponds to the sensor specified using the SENSOR_NUMBER. The macro returns either 0 or 1. If the bit value is 0, the sensor is not in detect. If the bit value is 1, the sensor is in detect.

```
#define GET_XXXXCAP_SENSOR_STATE(SENSOR_NUMBER) p_xxxxcap_measure_data->
p_sensor_states [(SENSOR_NUMBER / 8)] & (1 << (SENSOR_NUMBER % 8))) >>
(SENSOR_NUMBER % 8)
```

GET_ROTOR_SLIDER_POSITION (ROTOR_SLIDER_NUMBER)

To get the rotor angle or slider position. These values are valid only when the sensor state for corresponding rotor or slider state is in detect. ROTOR_SLIDER_NUMBER is the parameter for which the position is being obtained. The macro returns rotor angle or sensor position.

```
#define GET_XXXXCAP_ROTOR_SLIDER_POSITION(ROTOR_SLIDER_NUMBER)
p_xxxxcap_measure_data->p_rotor_slider_values [ROTOR_SLIDER_NUMBER]
```

GET_XXXXCAP_SENSOR_NOISE_STATUS (SENSOR_NUMBER)

To get the noise status of a particular sensor. The return value is 1 in case of sensor is noisy and returns 0 if sensor is not noisy.

```
#define GET_XXXXCAP_SENSOR_NOISE_STATUS (SENSOR_NUMBER)
(p_xxxxcap_measure_data->p_sensor_noise_status [(SENSOR_NUMBER / 8)] & (1 <<
(SENSOR_NUMBER % 8))) >> (SENSOR_NUMBER % 8)
```

GET_XXXXCAP_SENSOR_MOIS_STATUS (SENSOR_NUMBER)

To get the moisture status of a particular sensor. The return value is 1 in case of sensor is moisture affected and returns 0 if sensor is not moisture affected.

```
#define GET_XXXXCAP_SENSOR_MOIS_STATUS (SENSOR_NUMBER)
(p_xxxxcap_measure_data-> \p_sensor_mois_status [(SENSOR_NUMBER / 8)] & (1 <<
(SENSOR_NUMBER % 8))) >> (SENSOR_NUMBER % 8))
```

GET_XXXXCAP_AUTO_OS_CHAN_STATUS(CHAN_NUM)

To get the auto oversample status of a particular channel. The return value is 1 in case of channel auto oversample is going on and returns 0 if channel auto oversample process is not going on.

```
#define GET_XXXXCAP_AUTO_OS_CHAN_STATUS (CHAN_NUM)
(p_xxxxcap_measure_data->p_auto_os_status [(CHAN_NUM / 8)] & (1 <<
(CHAN_NUM % 8))) >> (CHAN_NUM % 8))
```

3.3 Enumerations

3.3.1 Touch Library GAIN Setting (tag_gain_t)

Detailed Description

Gain per touch channel. Gain is applied for an individual channel to allow a scaling-up of the touch delta. Delta on touch contact is measured on each sensor. The resting signal is ignored.

Range: GAIN_1 (no scaling) to GAIN_32 (scale-up by 32).

Data Fields

- GAIN_1
- GAIN_2
- GAIN_4
- GAIN_8
- GAIN_16
- GAIN_32

3.3.2 Filter Level Setting (tag_filter_level_t)

Detailed Description

Touch library FILTER LEVEL setting.

The filter level setting controls the number of samples acquired to resolve each acquisition. A higher filter level setting provides improved signal to noise ratio under noisy conditions, while increasing the total time for measurement which results in increased power consumption. The filter level should be configured for each channel.

Refer filter_level_t in touch_safety_api_samd.h

Range: FILTER_LEVEL_1 (one sample) to FILTER_LEVEL_64 (64 samples).

Data Fields

- FILTER_LEVEL_1
- FILTER_LEVEL_2
- FILTER_LEVEL_4
- FILTER_LEVEL_8
- FILTER_LEVEL_16
- FILTER_LEVEL_32
- FILTER_LEVEL_64

3.3.3 Touch Library AUTO OS Setting (tag_auto_os_t)

Detailed Description

Auto oversample controls the automatic oversampling of sensor channels when unstable signals are detected with the default setting of filter level. Each increment of Auto Oversample doubles the number of samples acquired from the corresponding sensor channel when an unstable signal is observed. The auto oversample should be configured for each channel.

For example, when filter level is set to FILTER_LEVEL_4 and Auto Oversample is set to AUTO_OS_4, 4 oversamples are collected with stable signal values and 16 oversamples are collected when unstable signal is detected.

Refer auto_os_t in touch_safety_api_samd.h

Range: AUTO_OS_DISABLE (oversample disabled) to AUTO_OS_128 (128 oversamples).

Data Fields

- AUTO_OS_DISABLE
- AUTO_OS_2
- AUTO_OS_4
- AUTO_OS_8
- AUTO_OS_16
- AUTO_OS_32
- AUTO_OS_64
- AUTO_OS_128

3.3.4 Library Error Code (tag_touch_ret_t)

Detailed Description

Touch Library error codes.

Data Fields

- TOUCH_SUCCESS Successful completion of touch operation.
- TOUCH_ACQ_INCOMPLETE Library is busy with pending previous touch measurement.
- TOUCH_INVALID_INPUT_PARAM Invalid input parameter.
- TOUCH_INVALID_LIB_STATE Operation not allowed in the current touch library state.
- TOUCH_INVALID_SELFCAP_CONFIG_PARAM Invalid self capacitance configuration input parameter.
- TOUCH_INVALID_MUTLCAP_CONFIG_PARAM Invalid mutual capacitance configuration input parameter.
- TOUCH_INVALID_RECAL_THRESHOLD Invalid recalibration threshold input value.
- TOUCH_INVALID_CHANNEL_NUM Channel number parameter exceeded total number of channels configured.
- TOUCH_INVALID_SENSOR_TYPE Invalid sensor type. Sensor type must NOT be SENSOR_TYPE_UNASSIGNED.
- TOUCH_INVALID_SENSOR_ID Invalid sensor number parameter.
- TOUCH_INVALID_RS_NUM Number of rotor/sliders set as 0, while trying to configure a rotor/slider.
- TOUCH_INTERNAL_TOUCH_LIB_ERR Touch internal library error
- TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR Touch logical flow error
- TOUCH_LIB_CRC_FAIL Touch library data CRC error
- TOUCH_LIB_DI_CHECK_FAIL Touch library double inverse check field
- TOUCH_PC_FUNC_MAGIC_NO_1 Program counter magic number 1
- TOUCH_PC_FUNC_MAGIC_NO_2 Program counter magic number 2
- TOUCH_PC_FUNC_MAGIC_NO_3 Program counter magic number 3
- TOUCH_PC_FUNC_MAGIC_NO_4 Program counter magic number 4
- TOUCH_PINS_VALIDATION_FAIL Touch pins are not valid
- TOUCH_ALL_SENSORS_DISABLED All sensors are disabled
- TOUCH_CNFG_MISMATCH Number of sensors defined in DEF_XXXXCAP_NUM_SENSORS are not equal to the number of sensors configured using touch_xxxcap_sensor_config() or Number of moisture groups defined In DEF_XXXXCAP_NUM_MOIS_GROUPS are not equal to the number of groups configured using touch_xxxxcap_cnfg_mois_mltchgrp or If moisture group threshold is not configured for all moisture groups.

3.3.5 Sensor Channel (`tag_channel_t`)

Detailed Description

Sensor start and end channel type of a Sensor. Channel number starts with value 0.

Data Fields

`CHANNEL_0` to `CHANNEL_255`

3.3.6 Touch Library State (`tag_touch_lib_state_t`)

Detailed Description

Touch library state.

Data Fields

- `TOUCH_STATE_NULL` Touch library is un-initialized. All sensors are disabled.
- `TOUCH_STATE_INIT` Touch library has been initialized.
- `TOUCH_STATE_READY` Touch library is ready to start a new capacitance measurement on enabled sensors.
- `TOUCH_STATE_CALIBRATE` Touch library is performing calibration on all sensors.
- `TOUCH_STATE_BUSY` Touch library is busy with on-going capacitance measurement.

3.3.7 Sensor Type (`tag_sensor_type_t`)

Detailed Description

Sensor types available.

Data Fields

- `SENSOR_TYPE_UNASSIGNED` Sensor is not configured yet.
- `SENSOR_TYPE_KEY` Sensor type key.
- `SENSOR_TYPE_ROTOR` Sensor type rotor.
- `SENSOR_TYPE_SLIDER` Sensor type slider.
- `MAX_SENSOR_TYPE` Max value of enum type for testing.

3.3.8 Touch Library Acquisition Mode (`tag_touch_acq_mode_t`)

Detailed Description

Touch library acquisition mode.

Data Fields

`RAW_ACQ_MODE`

When raw acquisition mode is used, the `measure_complete_callback` function is called immediately once a fresh value of signals are available. In this mode, the Touch Library does not perform any post processing. So, the references, sensor states or rotor/slider position values are not updated in this mode.

NORMAL_ACQ_MODE

When normal acquisition mode is used, the measure_complete_callback function is called only after the Touch Library completes processing of the signal values obtained. The references, sensor states and rotor/slider position values are updated in this mode.

3.3.9 AKS Group (tag_aks_group_t)

Detailed Description

It provides information about the sensors that belong to specific AKS group.

NO_AKS_GROUP indicates that the sensor does not belong to any AKS group and cannot be suppressed.

AKS_GROUP_x indicates that the sensor belongs to the AKS group x.

Data Fields

- NO_AKS_GROUP
- AKS_GROUP_1
- AKS_GROUP_2
- AKS_GROUP_3
- AKS_GROUP_4
- AKS_GROUP_5
- AKS_GROUP_6
- AKS_GROUP_7
- MAX_AKS_GROUP Max value of enum type for testing

3.3.10 Channel Gain Setting (tag_gain_t)

Detailed Description

A sensor detection hysteresis value. This is expressed as a percentage of the sensor detection threshold.

HYST_x = hysteresis value is x% of detection threshold value (rounded down).

Note: A minimum threshold value of 2 is used.

Example: If detection threshold = 20,

HYST_50= 10 (50% of 20)

HYST_25 = 5 (25% of 20)

HYST_12_5 = 2 (12.5% of 20)

HYST_6_25= 2 (6.25% of 20 = 1, but value is hard limited to 2)

Data Fields

- HYST_50
- HYST_25
- HYST_12_5
- HYST_6_25
- MAX_HYST Maximum value of enum type for testing

3.3.11 Sensor Recalibration Threshold (`tag_recal_threshold_t`)

Detailed Description

This is expressed as a percentage of the sensor detection threshold.

`RECAL_x` = recalibration threshold is x% of detection threshold value (rounded down).

Note: A minimum value of 4 is used.

Example: If detection threshold = 40,

`RECAL_100` = 40 (100% of 40)

`RECAL_50` = 20 (50% of 40)

`RECAL_25` = 10 (25% of 40)

`RECAL_12_5` = 5 (12.5% of 40)

`RECAL_6_25` = 4 (6.25% of 40 = 2, but value is hard limited to 4)

Data Fields

- `RECAL_100`
- `RECAL_50`
- `RECAL_25`
- `RECAL_12_5`
- `RECAL_6_25`
- `MAX_RECAL` Maximum value of enum type for testing

3.3.12 Rotor Slider Resolution (`tag_resolution_t`)

Detailed Description

For rotors and sliders, the resolution of the reported angle or position. `RES_x_BIT` = rotor/slider reports x-bit values.

Example: If slider resolution is `RES_7_BIT`, then reported positions are in the range 0..127.

Data Fields

- `RES_1_BIT`
- `RES_2_BIT`
- `RES_3_BIT`
- `RES_4_BIT`
- `RES_5_BIT`
- `RES_6_BIT`
- `RES_7_BIT`
- `RES_8_BIT`
- `MAX_RES` Maximum value of enum type for testing

3.3.13 Auto Tune Setting (`tag_auto_tune_type_t`)

Detailed Description

Touch library PTC prescaler clock and series resistor auto tuning setting

Data Fields

- `AUTO_TUNE_NONE` Auto tuning mode disabled. This mode uses the user defined PTC prescaler and series resistor values.
- `AUTO_TUNE_PRSC` Auto tune PTC prescaler for best noise performance. This mode uses the user defined series resistor value.
- `AUTO_TUNE_RSEL` Auto tune series resistor for least power consumption. This mode uses the user defined PTC prescaler value.

3.3.14 PTC Clock Prescale Setting (`tag_prsc_div_sel_t`)

Detailed Description

Refer `touch_configure_ptc_clock()` API in `touch.c`. PTC Clock Prescale setting is available for each channel.

Example:

If generic clock input to PTC = 4 MHz,

`PRSC_DIV_SEL_1` sets PTC Clock to 4 MHz.

`PRSC_DIV_SEL_2` sets PTC Clock to 2 MHz.

`PRSC_DIV_SEL_4` sets PTC Clock to 1 MHz.

`PRSC_DIV_SEL_8` sets PTC Clock to 500 KHz.

Data Fields

- `PRSC_DIV_SEL_1`
- `PRSC_DIV_SEL_2`
- `PRSC_DIV_SEL_4`
- `PRSC_DIV_SEL_8`

3.3.15 PTC Series Resistor Setting (`tag_rsel_val_t`)

Detailed Description

For mutual capacitance mode, this series resistor is switched internally on the Y-pin. For self capacitance mode, the series resistor is switched internally on the sensor pin. PTC Series Resistance setting is available for individual channel.

Example:

`RSEL_VAL_0` sets internal series resistor to `0` Ohms.

`RSEL_VAL_20` sets internal series resistor to `20` Kohms.

`RSEL_VAL_50` sets internal series resistor to `50` Kohms.

`RSEL_VAL_100` sets internal series resistor to `100` Kohms.

Data Fields

- `RSEL_VAL_0`
- `RSEL_VAL_20`
- `RSEL_VAL_50`
- `RSEL_VAL_100`

3.3.16 PTC Acquisition Frequency Delay Setting (`freq_hop_sel_t`)

Detailed Description

The PTC acquisition frequency is dependent on the generic clock input to PTC and PTC clock prescaler setting. This delay setting inserts n PTC clock cycles between consecutive measurements on a given sensor, thereby changing the PTC acquisition frequency. `FREQ_HOP_SEL_1` setting inserts 0 PTC clock cycle between consecutive measurements. `FREQ_HOP_SEL_16` setting inserts 15 PTC clock cycles. Hence, higher delay setting will increase the total time required for capacitance measurement on a given sensor as compared to a lower delay setting.

An optimal setting avoids noise in the same frequency as the acquisition frequency.

Data Fields

- `FREQ_HOP_SEL_1`
- `FREQ_HOP_SEL_2`
- `FREQ_HOP_SEL_3`
- `FREQ_HOP_SEL_4`
- `FREQ_HOP_SEL_5`
- `FREQ_HOP_SEL_6`
- `FREQ_HOP_SEL_7`
- `FREQ_HOP_SEL_8`
- `FREQ_HOP_SEL_9`
- `FREQ_HOP_SEL_10`
- `FREQ_HOP_SEL_11`
- `FREQ_HOP_SEL_12`
- `FREQ_HOP_SEL_13`
- `FREQ_HOP_SEL_14`
- `FREQ_HOP_SEL_15`
- `FREQ_HOP_SEL_16`

3.3.17 PTC Acquisition Frequency Mode Setting (`tag_freq_mode_sel_t`)

Detailed Description

The frequency mode setting option enables the PTC acquisition to be configured for the following modes.

- Frequency hopping and spread spectrum disabled.
- Frequency hopping enabled with median filter.
- Frequency spread spectrum enabled without median filter.
- Frequency spread spectrum enabled with median filter.

Range: `FREQ_MODE_NONE` (no frequency hopping & spread spectrum) to `FREQ_MODE_SPREAD_MEDIAN` (spread spectrum with median filter).

Data Fields

- `FREQ_MODE_NONE` 0u
- `FREQ_MODE_HOP` 1u
- `FREQ_MODE_SPREAD` 2u
- `FREQ_MODE_SPREAD_MEDIAN` 3u

3.3.18 PTC Sensor Lockout Setting (nm_sensor_lockout_t)

Detailed Description

The sensor lockout setting option allows the system to be configured in the following modes.

- SINGLE_SENSOR_LOCKOUT Single sensor can be locked out.
- GLOBAL_SENSOR_LOCKOUT All the sensors are locked out for touch detection.
- NO_LOCK_OUT All the sensors are available for touch detection.

Range: SINGLE_SENSOR_LOCKOUT to NO_LOCK_OUT.

Data Fields

- SINGLE_SENSOR_LOCKOUT 0u
- GLOBAL_SENSOR_LOCKOUT 1u
- NO_LOCK_OUT 2u

3.3.19 Moisture Group Setting (moisture_grp_t)

Detailed Description

Sensor can be configured in the moisture group using this type.

- MOIS_DISABLED Indicates that the sensor does not belong to any moisture group.
- MOIS_GROUP_X Indicates that the sensor belongs to the moisture group x.

Range: MOIS_DISABLED=0 to MOIS_GROUP_7.

Data Fields

- MOIS_DISABLED=0
- MOIS_GROUP_0
- MOIS_GROUP_1
- MOIS_GROUP_2
- MOIS_GROUP_3
- MOIS_GROUP_4
- MOIS_GROUP_5
- MOIS_GROUP_6
- MOIS_GROUP_7
- MOIS_GROUPN

3.3.20 Multi-touch Group Setting (mltch_grp_t)

Detailed Description

Sensor can be configured in the multi-touch group using this type

- MLTCH_NONE Indicates that the sensor does not belong to any multi-touch group.
- MLTCH_GROUP_X Indicates that the sensor belongs to the multi-touch group x.

Range: MLTCH_NONE=0 to MOIS_GROUP_7.

Data Fields

- MLTCH_NONE=0
- MLTCH_GROUP_0
- MLTCH_GROUP_1
- MLTCH_GROUP_2

- MLTCH_GROUP_3
- MLTCH_GROUP_4
- MLTCH_GROUP_5
- MLTCH_GROUP_6
- MLTCH_GROUP_7
- MLTCH_GROUPN

3.4 Data Structures

3.4.1 Touch Library Configuration Type

touch_config_t Struct Reference

Touch library Input Configuration Structure.

Data Fields

Field	Unit	Description
p_mutlcap_config p_selfcap_config	touch_mutlcap_config_t touch_selfcap_config_t	Pointer to mutual capacitance configuration structure. Pointer to self capacitance configuration structure.
ptc_isr_lvl	uint8_t	PTC ISR priority level

touch_mutlcap_config_t Struct Reference

Touch Library mutual capacitance configuration input type.

Data Fields

Field	Unit	Description
num_channels	uint16_t	Number of channels.
num_sensors	uint16_t	Number of sensors
num_rotors_and_sliders	uint8_t	Number of rotors/sliders.
global_param	touch_global_param_t	Noise measurement enable/disable
touch_xxxxcap_acq_param	touch_xxxxcap_acq_param_t	Sensor acquisition parameter info.
* p_data_blk	uint8_t	Pointer to data block buffer.
buffer_size	uint16_t	Size of data block buffer.
* p_mutlcap_xy_nodes	uint16_t	Pointer to xy nodes
mutl_quick_reburst_enable	uint8_t	Quick re-burst enable
(touch_filter_data_t *p_filter_data)	void(* filter_callback)	Mutual capacitance filter callback
enable_freq_auto_tune	uint8_t	Frequency auto tune enable
enable_noise_measurement	uint8_t	Noise measurement enable
nm_buffer_cnt	uint8_t	Memory allocation buffer
mutl_mois_tlrnce_enable	uint8_t	Mutual capacitance moisture tolerance enable flag
mutl_mois_groups	uint8_t	Number of mutual capacitance moisture groups

touch_selfcap_config_t Struct Reference

Touch Library self capacitance configuration input type.

Data Fields

Field	Unit	Description
num_channels	uint16_t	Number of channels.
num_sensors	uint16_t	Number of sensors
num_rotors_and_sliders	uint8_t	Number of rotors/sliders.
global_param	touch_global_param_t	Global sensor configuration information.
touch_selfcap_acq_param	touch_selfcap_acq_param_t	Sensor acquisition parameter information.
* p_data_blk	uint8_t	Pointer to data block buffer.
buffer_size	uint16_t	Size of data block buffer.
* p_selfcap_y_nodes	uint16_t	Pointer to selfcap nodes
self_quick_reburst_enable	uint8_t	Quick re-burst enable
(touch_filter_data_t *p_filter_data)	void(* filter_callback)	Self capacitance filter callback
enable_freq_auto_tune;	uint8_t	Frequency auto tune enable
enable_noise_measurement	uint8_t	Noise measurement enable
nm_buffer_cnt	uint8_t	Memory allocation buffer
self_mois_tlrnce_enable	uint8_t	Self cap moisture tolerance enable flag
self_mois_groups	uint8_t	Number of self-cap moisture groups

3.4.2 Touch Library Safety Type

touch_lib_fault_t Struct Reference

Detailed Description

This structure holds the inverse values of various touch library parameters.

Data Fields

Field	Unit	Description
inv_touch_ret_status	touch_ret_t	Holds the inverse value of the touch return status.

touch_lib_param_safety_t Struct Reference

Detailed Description

This structure holds the pointer to the data block for double inverse safety variables.

Data Fields

Field	Unit	Description
*p_inv_channel_signals	uint16_t	Pointer to the channel signals which hold the inverse value of different channel signals.
inv_acq_status	touch_acq_status_t	Holds the inverse value of the touch acquisition status.
inv_num_channel_signals	uint8_t	Holds the inverse value of the total number of channel signals.
inv_num_sensor_states	uint8_t	Holds the inverse value of the number of sensor states bytes.
*p_inv_sensor_states	uint8_t	Pointer to the sensor states that holds the inverse value of different sensor states.
inv_num_rotor_slider_values	uint8_t	Holds the inverse value of the number of rotor slider.
*p_inv_rotor_slider_values	uint8_t	Pointer to the rotor slider values that holds the inverse value of different rotor slider values
inv_lib_state	uint8_t	Holds the inverse value of the touch library state.
p_inv_delta	Int16_t	Holds the inverse value of the touch delta.
inv_current_time_ms	uint16_t	Holds the inverse value of current time millisecond variable.
inv_burst_again	uint8_t	Holds the inverse value of the burst again flag.
inv_acq_mode	touch_acq_mode_t	Holds the inverse value of the touch acquisition mode.
inv_sf_ptc_error_flag	uint8_t	Holds the inverse value of the PTC error flag.
inv_cc_cal_open_calibration_vals	uint16_t	Holds the inverse value of the CC calibration value.
*p_inv_sensor_noise_status	uint8_t	Holds the inverse value of the sensor noise status.
*p_inv_sensor_mois_status	uint8_t	Holds the inverse value of the Sensor moisture status.
*p_inv_chan_auto_os_status	uint8_t	Holds the inverse value of the channel auto os status.

3.4.3 Touch Library Double Inverse Type

touch_lib_di_data_block_t Struct Reference

Detailed Description

This structure holds the pointer to the data block for the double inverse safety variables.

Data Fields

Field	Unit	Description
p_di_data_block	uint8_t	Holds the pointer to the data block allocated by the application for double inverse check for the safety variables.
di_data_block_size	uint16_t	Holds the size of the data block allocated by the application of safety variables.

3.4.4 Touch Library Parameter Type

tag_touch_global_param_t Struct Reference

Detailed Description

Touch library global parameter type.

Data Fields

Field	Unit	Description
di Sensor	uint8_t	Detect Integration (DI) limit.
atch_drift_rate	uint8_t	Sensor away from touch drift rate.
tch_drift_rate	uint8_t	Sensor towards touch drift rate.
max_on_durationSensor	uint8_t	Maximum ON time duration.
drift_hold_time	uint8_t	Sensor drift hold time.
atch_recal_delay	uint8_t	Sensor away from touch recalibration delay.
recal_threshold	recal_threshold_t	Sensor away from touch recalibration threshold.
cal_seq_1_count	uint8_t	Sensor calibration dummy burst count.
cal_seq_2_count	uint8_t	Sensor calibration settling burst count.
auto_tune_sig_stability_limit	uint16_t	Stability limit for frequency auto tune feature.
auto_freq_tune_in_cnt	uint8_t	Frequency auto tune In counter.
nm_sig_stability_limit	uint16_t	Stability limit for noise measurement.
nm_noise_limit	uint8_t	Noise limit.
nm_enable_sensor_lock_out	nm_sensor_lockout_t	Sensor lockout feature variable.
nm_lockout_countdown	uint8_t	Lockout countdown for noise measurement.

tag_touch_xxxxcap_param_t Struct Reference

Detailed Description

Touch library capacitance sensor parameter type.

Data Fields

Field	Unit	Description
aks_group	aks_group_t	Which AKS group, the sensor belongs to.
detect_threshold	threshold_t	An unsigned 8-bit number setting a sensor detection threshold.
detect_hysteresis	hysteresis_t	A sensor detection hysteresis value. This is expressed as a percentage of the sensor detection threshold. • HYST_x = hysteresis value is x% of detection threshold value (rounded down). A minimum value of 2 is used. • Example: If detection threshold = 20, • HYST_50 = 10 (50% of 20) • HYST_25 = 5 (25% of 20) • HYST_12_5 = 2 (12.5% of 20) • HYST_6_25 = 2 (6.25% of 20 = 1, but value is hard limited to 2)
position_resolution	resolution_t	For rotors and sliders, the resolution of the reported angle or position. RES_x_BIT = rotor/slider reports x-bit values. Example: If slider resolution is RES_7_BIT, then reported positions are in the range 0..127
position_hysteresis	uint8_t	Sensor position hysteresis. This is valid only for a rotor or slider. bits 1..0: hysteresis. This parameter is valid only for mutual cap.

tag_touch_xxxxcap_acq_param_t Struct Reference

Detailed Description

Capacitance sensor acquisition parameter type.

Data Fields

Field	Unit	Description
p_xxxxcap_gain_per_node	gain_t	Pointer to gain per node.
touch_xxxxcap_freq_mode	uint8_t	Setup acquisition frequency mode.
p_xxxxcap_ptc_prsc	prsc_div_sel_t	Pointer to PTC clock prescaler value per node.
p_xxxxcap_resistor_value	rsel_val_t	Pointer to PTC series resistor value per node.
p_xxxxcap_hop_freqs	freq_hop_sel_t	Pointer to acquisition frequency settings.
p_xxxxcap_filter_level	filter_level_t	Pointer to Filter level per node..
p_xxxxcap_auto_os	auto_os_t	Pointer to Auto oversampling per node.
p_xxxxcap_ptc_prsc_cc_cal	prsc_div_sel_t	Pointer to PTC clock prescale value during CC cal
p_xxxxcap_resistor_value_cc_cal	rsel_val_t	Pointer to PTC series resistor value during CC cal..

3.4.5 Touch Library Measurement Data Type

tag_touch_measure_data_t Struct Reference

Detailed Description

Touch library measurement parameter type.

Data Fields

Field	Unit	Description
measurement_done_touch	volatile uint8_t	Flag set by touch_xxxxcap_measure_complete_callback( ) function when a latest Touch status is available.
acq_status	touch_acq_status_t	Status of touch measurement.
num_channel_signals	uint16_t	Length of the measured signal values list.
*p_channel_signals	uint16_t	Pointer to measured signal values for each channel.
num_channel_references	uint16_t	Length of the measured reference values list.
* p_channel_references	uint16_t	Touch status of each sensor.
num_sensor_states	uint8_t	Number of sensor state bytes.
num_rotor_slider_values	uint8_t	Length of the rotor and slider position values list.
*p_rotor_slider_values	uint8_t	Pointer to rotor and slider position values.
num_sensors	uint16_t	Length of the sensors data list.
* p_cc_calibration_vals	uint16_t	Pointer to calibrated compensation values for a given sensor channel.
p_sensors	sensor_t	Pointer to sensor data.
*p_sensor_noise_status	uint8_t	Pointer to noise status of the sensors.
*p_nm_ch_noise_val	uint16_t	Pointer to noise level value of each channel.
p_sensor_mois_status	uint8_t	Pointer to moisture status.
* p_auto_os_status	uint8_t	Pointer to Per channel Auto Oversample status.

3.4.6 Touch Library Filter Data Type

tag_touch_filter_data_t Struct Reference

Detailed Description

Touch library filter data parameter type.

Data Fields

Field	Unit	Description
num_channel_signals	uint16_t	Length of the measured signal values list.
p_channel_signals	uint16_t	Pointer to measured signal values for each channel.

3.4.7 Touch Library Time Type

tag_touch_time_t Struct Reference

Detailed Description

Touch library time parameter type.

Data Fields

Field	Unit	Description
measurement_period_ms	uint16_t	Touch measurement period in milliseconds. This variable determines how often a new touch measurement must be done.
current_time_ms	volatile uint16_t	Current time, set by timer ISR.
mutl_time_to_measure_touch	volatile uint8_t	Flag set by timer ISR when it is time to measure touch - Mutual capacitance method.
self_time_to_measure_touch	volatile uint8_t	Flag set by timer ISR when it is time to measure touch - Self capacitance method.

3.4.8 Touch Library Info Type

tag_touch_info_t Struct Reference

Detailed Description

Touch library Info type.

Data Fields

Field	Unit	Description
tlib_state	touch_lib_state_t	Touch library state.
num_channels_in_use	uint16_t	Number of channels currently in use.
num_sensors_in_use	uint16_t	Number of sensors in use irrespective of the sensor is enable or disable.
num_rotors_sliders_in_use	uint8_t	Number of rotor sliders in use, irrespective of the rotor/slider being disabled or enabled.
max_channels_per_rotor_slider	uint8_t	Max possible number of channels per rotor or slider.
fw_version	uint16_t	Touch library version.

3.4.9 Touch Library Version

touch_libver_info_t Struct Reference

Detailed Description

Touch library version information.

Product id for Safety Library is 202. Firmware version is formed of major, minor and patch version as given below:

`TLIB_MAJOR_VERSION = 5`

`TLIB_MINOR_VERSION = 1`

`TLIB_PATCH_VERSION = 4`

`fw_version` = (`TLIB_MAJOR_VERSION` << 8) | (`TLIB_MINOR_VERSION` << 4) | (`TLIB_PATCH_VERSION`)

Data Fields

Field	Unit	Description
`chip_id`	`uint32`	Chip identification number.
`product_id`	`uint16_t`	Product identification number.
`fw_version`	`uint16_t`	Library version number.

3.5 Global Variables

3.5.1 touch_lib_fault_test_status

Type

`touch_lib_fault_t`

Detailed Description

This structure holds the inverse value of the touch return status.

3.5.2 touch_error_app_cb

Type

`void (*)(touch_ret_t lib_error)`

Detailed Description

Callback function pointer that must be initialized by the application before a touch library API is called. Touch library would call the function pointed by this variable under certain error conditions.

3.5.3 touch_suspend_app_cb

Type

`void (* volatile touch_suspend_app_cb) (void)`

Detailed Description

Callback function pointer that must be initialized by the application before a touch library API is called.Touch library would call the function pointed by this function when suspension operation has to be carry on by the application.

If suspend operation is requested by application and touch library is not in `TOUCH_STATE_BUSY` state, then application will not receive suspend callback from the library. The application should continue the suspend operation in that case without waiting for the suspend callback.

3.6 Functions

3.6.1 Touch Library Initialization

The following API is used to initialize the Touch Library with capacitance method pin, register and sensor configuration provided by the user.

```
touch_ret_t touch_xxxxcap_sensors_init (touch_config_t * p_touch_config)
```

Fields	Description
`p_touch_config`	Pointer to touch configuration structure.

Returns:

`touch_ret_t:` Touch Library status.

3.6.2 Touch Library Sensor Configuration

The following API configures a capacitance sensor of type key, rotor or slider.

```
touch_ret_t touch_xxxxcap_sensor_config (sensor_type_t sensor_type, channel_t
from_channel, channel_t to_channel, aks_group_t aks_group, threshold_t detect_threshold,
hysteresis_t detect_hysteresis, resolution_t position_resolution, uint8_t
position_hysteresis, sensor_id_t * p_sensor_id)
```

Fields	Description
`sensor_type`	Sensor type key, rotor or slider.
`from_channel`	First channel in the slider sensor.
`to_channel`	Last channel in the slider sensor.
`aks_group`	AKS group (if any) the sensor belongs to.
`detect_threshold`	Sensor detection threshold.
`detect_hysteresis`	Sensor detection hysteresis value.
`position_resolution`	Resolution of the reported position value.
`position_hysteresis`	Hysteresis level of the reported position value.
`p_sensor_id`	Sensor id value of the configured sensor is updated by the Touch Library.

Returns:

`touch_ret_t:` Touch Library status.

3.6.3 Touch Library Sensor Calibration

The following API is used to calibrate the capacitance sensors for the first time before starting a touch measurement. This API can also be used to force calibration of capacitance sensors during runtime.

`touch_ret_t touch_xxxxcap_sensors_calibrate (auto_tune_type_t auto_tune_type )`

Fields	Description
`auto_tune_type`	Specify auto tuning parameter mode.

Returns:

`touch_ret_t:` Touch Library status.

Note: Call `touch_xxxxcap_sensors_measure` API after executing this API.

The following API calibrates the single sensor.

`touch_ret_t touch_xxxxcap_calibrate_single_sensor(sensor_id_t sensor_id)`

Fields	Description
`sensor_id`	Sensor number to calibrate.

Returns:

`touch_ret_t:` Touch Library status.

Note: Call `touch_xxxxcap_sensors_measure` API after executing this API. If calibration of a disabled sensor is required, `touch_xxxxcap_sensor_reenable` API should be used with calibration option. `touch_xxxxcap_calibrate_single_sensor` API should not be used for calibrating a disabled sensor. Otherwise it may lead to `TOUCH_LOGICAL_PROGRAM_CNTR_FLOW_ERR`.

3.6.4 Touch Library Sensor Measurement

The following API starts a touch measurement on capacitance sensors.

`touch_ret_t touch_xxxxcap_sensors_measure (touch_current_time_t current_time_ms,touch_acq_mode_t xxxxcap_acq_mode, uint8_t(*measure_complete_callback)(void))`

Fields	Description
`current_time_ms`	Current time in millisecond.
`xxxxcap_acq_mode`	Normal or raw acquisition mode.
`measure_complete_callback`	Callback function to indicate that a single touch measurement is completed.

Returns:

`touch_ret_t:` Touch Library status.

3.6.5 Touch Library Sensor Specific Touch Delta Read

The following API can be used retrieve the delta value corresponding to a given sensor for capacitance sensors respectively.

`touch_ret_t touch_xxxxcap_sensor_get_delta(sensor_id_t sensor_id, touch_delta_t * p_delta)`

Fields	Description
`sensor_id`	The sensor id for which delta value is being seeked.
`p_delta`	Pointer to the delta variable to be updated by the touch library.

Returns:

`touch_ret_t:` Touch Library status.

3.6.6 Touch Library Sensor Specific Parameter Configuration Read-write

The following API sets the individual sensor specific configuration parameters for capacitance sensors.

`touch_ret_t touch_xxxxcap_sensor_update_config (sensor_id_t sensor_id, touch_xxxxcap_param_t * p_touch_sensor_param)`

Fields	Description
`p_sensor_id`	The sensor id for which configuration parameter information is being set.
`p_touch_sensor_param`	The touch sensor parameter structure that will be used by the touch library to update.

Returns:

`touch_ret_t:` Touch Library status.

The following API reads the sensor configuration parameters for capacitance sensors.

`touch_ret_t touch_xxxxcap_sensor_get_config (sensor_id_t sensor_id, touch_xxxxcap_param_t * p_touch_sensor_param)`

Fields	Description
`p_sensor_id`	The sensor id for which configuration parameter information is being set.
`p_touch_sensor_param`	The touch sensor parameter structure that will be used by the touch library to update.

Returns:

`touch_ret_t:` Touch Library status.

3.6.7 Touch Library Sensor Specific Acquisition Configuration Read-write

The following API sets the sensor specific acquisition configuration parameters for capacitance sensors respectively.

`touch_ret_t touch_xxxxcap_sensor_update_acq_config (touch_xxxxcap_acq_param_t * p_touch_xxxxcap_acq_param)`

Fields	Description
`p_touch_xxxxcap_acq_param`	The touch sensor acquisition parameter structure that will be used by the touch library to update.

Returns:

`touch_ret_t:` Touch Library status.

Note: `touch_xxxxcap_sensor_update_acq_config` API must be called after the `touch_xxxxcap_sensors_init` API.

The following API gets the sensor specific acquisition configuration parameters for cap sensors respectively.

`touch_ret_ttouch_xxxxcap_sensor_get_acq_config (touch_xxxxcap_acq_param_t * p_touch_xxxxcap_acq_param)`

Fields	Description
`p_touch_xxxxcap_acq_param`	The touch sensor acquisition parameter structure that will be used by the touch library to update.

Returns:

`touch_ret_t:` Touch Library status.

3.6.8 Touch Library Sensor Global Parameter Configuration Read-write

The following API updates the global parameter for cap sensors respectively.

`touch_ret_t touch_xxxxcap_update_global_param (touch_global_param_t * p_global_param)`

Fields	Description
`p_global_param`	The pointer to global sensor configuration.

Returns:

`touch_ret_t:` Touch Library status.

Note: `touch_xxxxcap_update_global_param` API must be called after the `touch_xxxxcap_sensors_init` API.

The following API reads back the global parameter for cap sensors respectively.

touch_ret_t touch_xxxxcap_get_global_param (touch_global_param_t * p_global_param)

Fields	Description
`p_global_param`	The pointer to global sensor configuration.

Returns:

`touch_ret_t:` Touch Library status.

3.6.9 Touch Library Info Read

The following API gets the Touch Library status information for cap sensors respectively.

touch_ret_t touch_xxxxcap_get_libinfo (touch_info_t * p_touch_info)

Fields	Description
`p_touch_info`	Pointer to the touch info data structure that will be updated by the touch library.

Returns:

`touch_ret_t:` Touch library status.

3.6.10 Touch Library Program Counter

touch_ret_t touch_lib_pc_test_magic_no_1 (void)

The following API tests the program counter inside the touch library. This function returns the unique magic number `TOUCH_PC_FUNC_MAGIC_NO_1` to the application.

Returns: `touch_ret_t`

touch_ret_t touch_lib_pc_test_magic_no_2 (void)

This function tests the program counter inside the touch library. This function returns the unique magic number `TOUCH_PC_FUNC_MAGIC_NO_2` to the application.

Returns: `touch_ret_t`

touch_ret_t touch_lib_pc_test_magic_no_3 (void)

This function tests the program counter inside the touch library.This function returns the unique magic number `TOUCH_PC_FUNC_MAGIC_NO_3` to the application.

Returns: `touch_ret_t`

touch_ret_t touch_lib_pc_test_magic_no_4 (void)

This function tests the program counter inside the touch library.This function returns the unique magic number `TOUCH_PC_FUNC_MAGIC_NO_4` to the application.

Returns: `touch_ret_t`

3.6.11 Touch Library CRC Configuration Check

touch_ret_t touch_calc_xxxxcap_config_data_integrity

This function computes 16 bit CRC for the touch configuration data and stores it in a global variable internal to the library.

Returns: `touch_ret_t`.

touch_ret_t touch_test_xxxxcap_config_data_integrity(void)

This function performs a test to verify the integrity of the touch configuration data. It computes the CRC value and tests it against the previously stored CRC value. The result of the comparison is passed back to the application.

Returns: Returns the result of the test integrity check. If CRC check passes, it returns `TOUCH_SUCCESS`, else it returns `TOUCH_LIB_CRC_FAIL.`

3.6.12 Touch Library Double Inverse check

touch_ret touch_xxxxcap_di_init (touch_lib_di_data_block_t *p_dblk)

This function initializes the memory from inverse data block allocated by the application for different pointers in the `touch_lib_param_safety_t`.

Data Fields

Fields	Description
* `p_dblk`	Pointer to the starting address of the data block allocated by the application for double inverse check.

Returns: `touch_ret_t`

This API must be called after the `touch_xxxxcap_sensors_init` API and before any other API is called.

3.6.13 Touch Library Enable Disable Sensor

touch_ret touch_xxxxcap_sensor_disable (sensor_id_t sensor_id)

This function disable the sensor.

Data Fields

Fields	Description
`sensor_id`	Sensor which needs to be disabled.

Returns : touch_ret_t

touch_ret touch_xxxxcap_sensor_reenable (sensor_id_t sensor_id, uint8_t no_calib)

This function will enable the sensor.

Data Fields

Fields	Description
`sensor_id`	Sensor which needs to be re-enabled.
`no_calib`	Re-enable of sensor would be done with calibration or not. If value is 1, sensor would be re-enable without calibration else if value is 0, sensor would be re-enable with calibration.

Returns : `touch_ret_t`

Notes: 1. Call `touch_xxxxcap_sensors_measure` API after executing this API.
2. It is recommended to re-enable the sensors with calibration (`no_calib` = 0), if sensors are re-enabled after a long duration. Refer Section 2.18, “Drifting on Disabled Sensors”.

3.6.14 Touch Library Version Information

This function will provide the library version information.

touch_ret_t touch_library_get_version_info(touch_libver_info_t *p_touch_libver_info);

Data Fields

Fields	Description
`p_touch_libver_info`	Pointer to touch library version information structure.

Returns : `touch_ret_t`

3.6.15 Touch Library Moisture Tolerance

This function can be used to Configure sensor in the moisture group and multi touch group.

touch_ret_t touch_xxxxcap_cnfg_mois_mltchgrp (sensor_id_t snsr_id,moisture_grp_t mois_grpid,mltch_grp_t mltch_grpid);

Data Fields

Fields	Description
`snsr_id`	Sensor to configure.
`mois_grpid`	Sensor to be configured in this moisture group.
`mltch_grpid`	Sensor to be configured in this multi touch group.

Returns : `touch_ret_t`

This function can be used to configure moisture group sensor moisture lock and system moisture lock threshold.

touch_ret_t touch_xxxxcap_cnfg_mois_threshold (moisture_grp_t,mois_snsr_threshold_t snsr_threshold,mois_system_threshold_t system_threshold);

Data Fields

Fields	Description
`mois_grpid`	Moisture group id.
`snsr_threshold`	Sensor moisture lock threshold.
`system_threshold`	System moisture lock threshold.

Returns : touch_ret_t

This function can be used to enable the moisture tolerance feature.

touch_ret_t touch_xxxxcap_mois_tolrnce_enable (void);

Data Fields

None

Returns : `touch_ret_t`

This function can be used to disable the moisture tolerance feature.

touch_ret_t touch_xxxxcap_mois_tolrnce_disable (void);

Data Fields

None

Returns : `touch_ret_t`

3.6.16 Touch PTC Peripheral Enable Disable

touch_ret_t touch_disable_ptc(void)

This function disable the PTC module

Data Fields

None

Returns : `touch_ret_t`

Note: Refer Section 2.17 and Section 4.6 for use cases associated with `touch_disable_ptc`.

touch_ret_t touch_enable_ptc(void)

This function enable the PTC module.

Data Fields

None

Returns : `touch_ret_t`

Note: Refer Section 2.17 for use cases associated with `touch_enable_ptc`.

3.6.17 Touch Library Suspend Resume

touch_ret_t touch_suspend_ptc(void)

This function suspends the PTC library's current measurement cycle. The completion of the operation is indicated through callback pointer that must be initialized by the application. Refer section Section 3.5.3 and Section 2.17.

Data Fields

None

Returns : `touch_ret_t`

touch_ret_t touch_resume_ptc(void)

This function resumes the PTC library's current measurement which was suspended using `touch_suspend_ptc.` After the `touch_resume_ptc` is called by the application, the `touch_xxxxcap_sensors_measure` API should be called only after the measurement complete callback function is received. Refer section Section 3.5.3 and Section 2.17.

Data Fields

None

Returns : `touch_ret_t`

Note: The APIs related to touch suspend operation must be used in accordance with the safety requirements of the product and must be taken care by the customer application.

3.6.18 Touch Library Re-Initialization

touch_ret_t touch_xxxxcap_sensors_deinit(void)

This function deinitializes the touch library.This API should be called only when the library state is in `TOUCH_STATE_INIT` or `TOUCH_STATE_READY` state.

After calling deinit API,no other API should be called apart from `touch_xxxxcap_sensors_init` to reinitialize the touch library.

Data Fields

None

Returns : `touch_ret_t`

Note:

a. If one module(self-cap or mutual-cap touch library) is de-initialized, then all other modules should be de-initialized as well. For eg., if mutual-cap touch library is de-initialized, then mutual-cap FMEA, self-cap touch library and self-cap FMEA should be de-initialized or stopped.
b. When touch library or FMEA has to be re-initialized, the application has to follow the initialization sequence as done during power-up.

4. FMEA

This section provides information about the FMEA component. The FMEA library supports the rotor/slider built with spatially interpolated design. FMEA component is further categorized into mutual and self capacitance FMEA component. FMEA will be performed on all the touch pins including sensor disabled pins.

For more information about designing the touch sensor, refer to *Buttons, Sliders and Wheels Touch Sensor Design Guide* (www.atmel.com).

4.1 Double Inverse Memory Check

4.1.1 Application to FMEA

No variable is interfaced from the application to FMEA. Hence, Double Inverse mechanism need not be used for protection.

4.1.2 FMEA to Application

The following variable must be protected using the specified inverse variable.

Table 4-1.

Variable	Inverse Variable
`faults_to_report`	`faults_to_report_inv` (Refer Section 5.3.3)

4.2 Memory Requirement

The following table provides the Flash and the RAM memory required for various configurations using different number of channels.

Default Configuration:

The following Macros are defined for all the cases mentioned for the Memory Calculation in Section 4.2.1

- `SELFCAP_FMEA_MAP_FAULT_TO_CHANNEL`
- `MUTLCAP_FMEA_MAP_FAULT_TO_CHANNEL`

4.2.1 Memory Requirement for IAR Library

4.2.1.1 Memory Requirement for Mutual Capacitance

Total No of Mutual Cap Channels	Total Code Memory	Total Data Memory
1	2224	88
10	2268	108
20	2268	124
40	2268	164
256	2300	596

4.2.1.2 Memory Requirement Self Capacitance

Total No of Self Cap Channels	Total Code Memory	Total Data Memory
1	2056	80
2	2124	84
11	2148	124
16	2174	144

4.2.1.3 e

Total No of Mutual Cap Channels	Total No of Self Cap Channels	Total Code Memory	Total Data Memory
1	1	4538	168
40	8	4689	276
80	11	4692	368

4.3 API Execution Time

4.3.1 Mutual Capacitance API Execution Time

The following table provides information about the execution time required for various FMEA APIs.

System Clock Frequency: 48MHz

PTC Clock Frequency: 4MHz

Table 4-2. Mutual Capacitance FMEA API Execution Time

API	Input Value	Time (in µs)	
		1 Channel (PORT A)	20 Channels (PORT A & B) (5 x4)
`sf_mutlcap_fmea_init`	Any value	84.8	125.6
`sf_mutlcap_fmea_test`	`0x01` (short to Vcc)	114	303.5
	`0x02` (short to Vss)	115.2	309
	`0x04` (short between pins)	744	3656
	`0x08` (PTC register test)	214	352.9
	`0x10` (input configuration data integrity check)	122	265.1
	`0x1F` (all test)	1000	4032
`sf_mutlcap_fmea_test_open_pins_per_channel`	Any value	13200	12830

Notes:
1. For the `sf_mutlcap_fmea_test_open_pins_per_channel` API, the preceding table provides the maximum time required to complete the procedure. After the control is returned back to the application, the application can execute any other tasks.
2. API Execution Time marked as * are calculated for sensors with typical sensor capacitance values.

The time for the Mutual capacitance FMEA API to return the control to the application is as follows:

API	Input Value	Time (in µs)	
		1 Channel (PORT A)	20 Channels (PORT A & B) (5 x4)
`sf_mutlcap_fmea_test_open_pins_per_channel`	Any value	50.2	50.3

4.3.2 Self Capacitance API Execution Time

The following table provides information about the APIs and their corresponding execution time.

Table 4-3. Self Capacitance FMEA API Execution Time

API	Input Value	Time (in µs)	
		1 Channel (PORT A)	16 Channels (PORT A & B)
`sf_selfcap_fmea_init`	Any value	79.4	218
`sf_selfcap_fmea_test`	`0x01` (short to Vcc)	99.4	290
	`0x02` (short to Vss)	99.8	298
	`0x04` (short between pins)	612	3540
	`0x08` (PTC register test)	212	348
	`0x10` (input configuration data integrity check)	116	329
	`0x1F` (all test)	848	3946
`sf_selfcap_fmea_test_open_pins_per_channel`	Any value	10800	10700

Notes:
1. For the `sf_selfcap_fmea_test_open_pins_per_channel` API, the preceding table provides the maximum time required to complete the procedure. After the control is returned back to the application, the application can execute any other tasks.
2. API Execution Time marked as * are calculated for sensors with typical sensor capacitance values.

The time for the Self capacitance FMEA API to return the control to the application is as follows:

Table 4-4. Self Capacitance FMEA Asynchronous API Execution Time

API	Input Value	Time (in µs)	
		1 Channel (PORT A)	16 Channels (PORT A & B)
`sf_selfcap_fmea_test_open_pins_per_channel`	Any value	48.7	48.9

4.4 Error Interpretation

Table 4-5. Error Interpretation

List of API	Error Bit	Reason	Error Coverage
sf_xxxxcap_fmea_init	FMEA_ERR_INIT	CRC value computed by touch library has failed double inverse check	Not applicable
	FMEA_ERR_INIT	Input pointer is NULL	Not applicable
	FMEA_ERR_INIT	Input values are not within limit	Not applicable
sf_xxxxcap_fmea_test	FMEA_ERR_PRE_TEST	Undefined test bits are set	Not applicable
	FMEA_ERR_PRE_TEST	This function is called before calling sf_xxxxcap_fmea_init()	Not applicable
	FMEA_ERR_SHORT_TO_VCC	Any one touch pin is short to Vcc	XXXXCAP enabled pins
	FMEA_ERR_CONFIG_CHECK CRC	CRC check has failed Not applicable	Not applicable
	FMEA_ERR_SHORT_TO_VSS	Any one touch pin is short to Vss	XXXXCAP enabled pins
	FMEA_ERR_SHORT_TO_PINS	Any two touch pins are shorted to each other	XXXXCAP enabled pins
	FMEA_ERR_PTC_REG	PTC register test failed or the PTC test status returned by touch library failed double inverse check	Not applicable
	FMEA_ERR_SHORT_TO_VCC	At least one test has failed	
	FMEA_ERR_SHORT_TO_VSS		
	FMEA_ERR_SHORT_TO_PINS		
	FMEA_ERR_PTC_REG		
	FMEA_ERR_CONFIG_CHECK		

Table 4-5. Error Interpretation

List of API	Error Bit	Reason	Error Coverage
`sf_xxxxcap_fmea_test_open_pins_per_channel`	`FMEA_ERR_PRE_TEST`	This function is called before calling `sf_xxxxcap_fmea_init()`	Not applicable
	`FMEA_ERR_PRE_TEST`	Channel number passed is more than the maximum possible	Not applicable
	`FMEA_ERR_OPEN_PINS`	There is a disconnect between sensor electrode and device pin for the given channel number	One channel per call

4.5 Data and Function Protection

The functions and global variables which are used only by FMEA are marked as static. The user / application should not change the same to non-static.

The header file `sf_fmea_samd_int.h` file is used only by FMEA. The user/application should not include this header file in any other files.

Table 4-6. Header File Availability for Application

Header File	Availability for Application	Configurable Fields
`sf_fmea_samd_int.h`	No	Not applicable
`sf_fmea_samd_api.h`	Yes	• `FMEA_VAR_LOCATION` • `MUTLCAP_FMEA_MAP_FAULT_TO_CHANNEL` • `SELFCAP_FMEA_MAP_FAULT_TO_CHANNEL`

4.6 FMEA Considerations

FMEA Short Between Pins, Short to VSS, Short to VCC can be detected on the MCU pins. The periodicity of Short to VSS test should be much lesser than the Short between Pins test. The `touch_disable_ptc` could be called after `sf_xxxxcap_fmea_test` API and also after the open pin test callback is received for each channel.This should be done to reduce the power consumption.

5. FMEA API

5.1 Typedefs

None

5.2 Enumerations

5.2.1 sf_fmea_faults_t

This enumeration describes the types of FMEA faults or errors such as short to Vcc, short to Vss, and short between pins that occur in a system. The test results of FMEA tests are stored in global fault report structure. The generic test result of FMEA test is stored in `faults_to_report` field of `sf_xxxxcap_fmea_fault_report_var`. Each bit of the field `faults_to_report` field represents the test status for each FMEA test.

Table 5-1. FMEA Fault Details

Values	Description
`FMEA_ERR_SHORT_TO_VCC`	Short to Vcc
`FMEA_ERR_SHORT_TO_VSS`	Short to Vss
`FMEA_ERR_SHORT_TO_PINS`	Short between pins
`FMEA_ERR_PTC_REG`	PTC register test
`FMEA_ERROR_CONFIG_CHECK`	Checks the input configuration integrity
`FMEA_ERR_OPEN_PINS`	Open connection between device pin and sensor
`FMEA_ERROR_PRE_TEST`	Pre-test failure
`FMEA_ERR_INIT`	Initialization

For example, `FMEA_ERR_SHORT_TO_VCC` bit represents short to Vcc test status, the `FMEA_ERR_SHORT_TO_VSS` bit represents short to Vss test status.

Note: If multiple FMEA tests are conducted in a single API call, sf_xxxxcap_fmea_fault_report_var will hold the consolidated results of all the requested tests.

In other case, when FMEA tests are conducted one after other by the application, sf_xxxxcap_fmea_fault_report_var will hold only the latest test results(previous results will be cleared each time by FMEA component).In such cases, it is recommenced that application should keep track of fault report variable.

5.3 Data Structures

5.3.1 sf_xxxxcap_fmea_open_test_config_t

The configuration parameters required for FMEA open pin test are passed through this structure.

Fields	Type	Description

`cc_cal_valid_min_val`
`For Mutual capacitance`
`cc_cal_valid_min_val[DEF_SELFCAP_NUM_CHANNELS]`
`For Self capacitance`
	`uint16_t`	CC value should be provided for each selfcap channel.In case of mutual cap, single cc calibration value needs to be provided.
		Maximum value: 16000

`cc_cal_val_min_no_error`	`uint8_t`	Open errors are declared only if CC calibration values of a particular channel is out of range in N1 samples out of N2 samples. For example, if N2 is set to 4 and N1 is set to 2, then CC calibration values are compared with the `cc_cal_valid_min_val` low and high limits, for continuous 4 samples. The channels whose CC calibration values are in error for more than 2 samples are declared error. Whenever an open pin test function is called, a sample counter corresponding to the channel is incremented. If an error is found among the samples, the error count for the channel is incremented. If the error count reaches N1, the error is reported and the error count and sample count are reset. If sample count reaches N2 value (it indicates that the error count has not reached N1) the error count and sample count is reset. In the previous example, `cc_cal_val_min_no_error` represents N1. Maximum value: `cc_cal_val_no_of_samples` Minimum value: 1
`cc_cal_val_no_of_samples`	`uint8_t`	In the previous example, `cc_cal_val_no_of_samples` represents N2. Maximum value: 15 Minimum value: 1
`xxxxcap_open_pin_test_callback`	`void (*)(uint16_t)`	After completing the open pin test, the open pin test function calls the `xxxxcap_open_pin_test_callback` function and indicates the completion of the open pin test. The application can pick the test status in this complete callback functions.

Note: The open pin test is performed indirectly by measuring the capacitance of the sensor electrode. If the sensor electrode is disconnected from the device pin, the measured capacitance value will be less when compared to that of the sensor electrode connected to the device pin.

During design stage, the application developer must monitor the equivalent capacitance value for all the channels under normal (all the sensors are connected and un-touched) condition. User can read the equivalent capacitance value as shown in the following example:

```
/* channel 0's equivalent capacitance */
p_xxxxcap_measure_data->p_cc_calibration_vals[0]
/* channel 1's equivalent capacitance */
p_xxxxcap_measure_data->p_cc_calibration_vals[1]
```

Although not mandatory, it is recommended to set `cc_cal_valid_min_val` as 30% of the lowest value observed in `p_cc_calibration_vals` array.

For example, if 415 is the lowest value observed in the `p_cc_calibration_vals` array, set `cc_cal_valid_min_val` as 124.

Note: The CC values would differ based on the value of series resistance (internal or external) connected to the touch pins.

5.3.2 sf_xxxxcap_fmea_input_config_t

The Open CC values will change based on the resistance added on the touch lines.Proper value of CC has to given as input to the `sf_xxxxcap_fmea_test_open_pins_per_channel` function. The FMEA test input configuration data are passed through this structure.

```
typedef struct
tag_sf_xxxxcap_fmea_input_config_t
{
 sf_xxxxcap_fmea_open_test_config_t *xxxxcap_open_test_config;
}sf_xxxxcap_fmea_input_config_t;
```

Values	Description
`xxxxcap_open_test_config`	Refer `sf_xxxxcap_fmea_open_test_config_t` description in Section 5.3.1

5.3.3 sf_mutlcap_fmea_fault_report_t

The Mutual capacitance FMEA test API status is updated in this structure.

```
typedef struct tag_sf_mutlcap_fmea_fault_report_t
{
uint16_t faults_to_report;
uint16_t faults_to_report_inv;
uint16_t x_lines_fault_vcc;
uint16_t y_lines_fault_vcc;
uint16_t x_lines_fault_vss;
uint16_t y_lines_fault_vss;
uint16_t x_lines_fault_short;
uint16_t y_lines_fault_short;
#ifdef MUTLCAP_FMEA_MAP_FAULT_TO_CHANNEL
uint8_t fmea_channel_status[DEF_MUTLCAP_NUM_CHANNELS];
```

```
#endif
}sf_mutlcap_fmea_fault_report_t;
```

Values	Description
`faults_to_report`	If a bit is set to `1` in `fault_to_report`, then corresponding fault has occurred. If a bit is set to `0` in `fault_to_report`, then the corresponding fault has not occurred. The X/Y lines and channels that are affected are provided in other fields. FMEA fault status • Bit 0 represents the short to Vcc. • Bit 1 represents the short to Vss. • Bit 2 represents the short to PINS. • Bit 3 represents the PTC register test. • Bit 4 represents the Configuration data integrity. • Bit 5 represents the Open pin fault. • Bit 6 represents the fault pre-test failure condition. • Bit 7 represents the fault init failed condition. The bit 0 is set if at least one of the touch pin (X or Y) is short to Vcc. The bit 1 is set if at least one of the touch pin (X or Y) is short to Vss. The bit 2 is set if at least two touch pins are shorted to each other. The bit 3 is set if, • a fault is found in PTC register test • the test result passed by touch library fails double inversion check The bit 4 is set if, • a fault is found in the input configuration data integrity • the CRC value computed by touch library fails double inversion check The bit 5 is set if at least one touch pin is not connected with the sensor electrode. The bit 6 is set if, • the `sf_mutlcap_fmea_test()` function is called before executing the initialization function • if the channel number passed to `sf_mutlcap_fmea_test_open_pins_per_channel()` function is greater than `DEF_MUTLCAP_NUM_CHANNELS`. The bit 7 is set if, • invalid parameters are passed to the FMEA initialization function • when the CRC value computed by the touch library for the input configuration data fails the double inverse check • the input pointer is NULL.
`faults_to_report_inv`	Compliment value of field `faults_to_report`
`x_lines_fault_vcc`	If bit n is set, then Xn pin is short to Vcc
`y_lines_fault_vcc`	If bit n is set, then Yn pin is short to Vcc

Values	Description
`x_lines_fault_vss`	If bit n is set, then Xn pin is short to Vss
`y_lines_fault_vss`	If bit n is set, then Yn pin is short to Vss
`y_lines_fault_short`	If bit n is set, then Xn pin is short to other touch pin
`x_lines_fault_short`	If bit n is set, then Yn pin is short to other touch pin
`fmea_channel_status [DEF_MUTLCAP_NUM_CHANNELS]`	This array maps FMEA faults to individual channel numbers. This variable is applicable only if `MUTLCAP_FMEA_MAP_FAULT_TO_CHANNEL` macro is defined in `sf_fmea_samd_api.h` file. This is used to map FMEA faults to individual channel numbers. Each byte in the array corresponds to the FMEA faults in the particular channel number. Example: `FMEA_CHANNEL_STATUS[0]` represents the fault of the channel number 0. Each bit in the byte represents the FMEA test status. Example: • Bit `0` represents the short to Vcc. • Bit `1` represents the short to Vss. • Bit `2` represents the short to PINS. • Bit `5` represents the open pin fault. If X or Y pin corresponding to a channel is shorted to Vcc then the Bit 0 position of that specific byte will be set to 1. If X or Y pin corresponding to the channel is shorted to Vss then the Bit 1 position of that specific byte will be set to 1. If X or Y pin corresponding to the channel is shorted to other X or Y pins, the Bit 2 of all the channel which uses the faulty X or Y will be set to 1. Bit `5` of all the channels whose sensor electrode is not connected to the device pin is set to `1`. Since PTC register test, configuration data integrity, pre-test failure and initialization failure are common for all the channels, `fmea_channel_status` will not contain those information.

5.3.4 sf_selfcap_fmea_fault_report_t

The self capacitance FMEA test API status is updated in this structure.

```
typedef struct tag_sf_selfcap_fmea_fault_report_t
{
uint16_t faults_to_report;
uint16_t faults_to_report_inv;
uint16_t y_lines_fault_vcc;
uint16_t y_lines_fault_vss;
uint16_t y_lines_fault_short;
#ifdef SELFCAP_FMEA_MAP_FAULT_TO_CHANNEL
uint8_t fmea_channel_status[DEF_SELFCAP_NUM_CHANNELS];
#endif
```

```
}sf_selfcap_fmea_fault_report_t;
```

Values	Description
`faults_to_report`	If a bit is set to `1` in `fault_to_report`, then corresponding fault has occurred. If a bit is set to `0` in `fault_to_report`, then the corresponding fault has not occurred. The Y lines and channels that are affected are provided in other fields. FMEA fault status: • Bit 0 represents the short to Vcc. • Bit 1 represents the short to Vss. • Bit 2 represents the short to PINS. • Bit 3 represents the PTC register test. • Bit 4 represents the Configuration data integrity. • Bit 5 represents the Open pin fault. • Bit 6 represents the fault pre-test failure condition. • Bit 7 represents the fault init failed condition. The bit 0 is set if at least one of the touch pin (Y) is short to Vcc. The bit 1 is set if at least one of the touch pin (Y) is short to Vss. The bit 2 is set if at least two touch pins are shorted to each other. The bit 3 is set if, • a fault is found in PTC register test • the test result passed by touch library fails double inversion check The bit 4 is set if, • a fault is found in the input configuration data integrity • the CRC value computed by touch library fails double inversion check The bit 5 is set if at least one touch pin is not connected with the sensor electrode. The bit 6 is set if, • the `sf_selfcap_fmea_test()` function is called before executing the initialization function • if the channel number passed to `sf_selfcap_fmea_test_open_pins_per_channel()` function is greater than `DEF_SELFCAP_NUM_CHANNELS`. The bit 7 is set if, • invalid parameters are passed to the FMEA initialization function • when the CRC value computed by the touch library for the input configuration data fails the double inverse check • the input pointer is NULL.
`faults_to_report_inv`	Compliment value of field `faults_to_report`

Values	Description
`y_lines_fault_vcc`	If bit n is set, then Yn pin is short to Vcc
`y_lines_fault_vss`	If bit n is set, then Yn pin is short to Vss
`fmea_channel_status [DEF_SELFCAP_NUM_CHANNELS]`	This array maps FMEA faults to individual channel numbers. This variable is applicable only if `SELFCAP_FMEA_MAP_FAULT_TO_CHANNEL` macro is defined in `sf_fmea_samd_api.h` file. This is used to map FMEA faults to individual channel numbers. Each byte in the array corresponds to the FMEA faults in the particular channel number. Example: `FMEA_CHANNEL_STATUS[0]` represents the fault of the channel number 0. Each bit in the byte represents the FMEA test status. Example: • Bit `0` represents the short to Vcc. • Bit `1` represents the short to Vss. • Bit `2` represents the short to PINS. • Bit `5` represents the open pin fault. If Y pin corresponding to a channel is shorted to Vcc then the Bit 0 position of that specific byte will be set to 1. If Y pin corresponding to the channel is shorted to Vss then the Bit 1 position of that specific byte will be set to 1. If Y pin corresponding to the channel is shorted to other Y pins, the Bit 2 of all the channel which uses the faulty Y will be set to 1. Bit 5 of all the channels whose sensor electrode is not connected to the device pin is set to 1. Since PTC register test, configuration data integrity, pre-test failure and initialization failure are common for all the channels, `fmea_channel_status` will not contain those information.

Note: The application must validate the field `faults_to_report` by performing the double inversion check on `faults_to_report` variable using the `faults_to_report_inv` variables.

5.4 Global Variables

5.4.1 sf_xxxxcap_fmea_fault_report_var

Type	Description
`sf_xxxxcap_fmea_fault_report_t`	Holds the test status from the latest `sf_xxxxcap_fmea_test()` call. Refer Section 5.3.3 for mutual capacitance and Section 5.3.4 for self capacitance related information. The members, `faults_to_report` and `faults_to_report_inv` of `sf_xxxxcap_fmea_fault_report_var` variable must be verified for double inversion before using any other member of this variable.

5.5 Functions

5.5.1 sf_xxxxcap_fmea_init

This function initializes all the FMEA related variables and verifies if the input parameters are within predefined range. If the values are outside the predefined range, the `faults_to_report` field of `sf_xxxxcap_fmea_fault_report_var` global structure is updated with an `FMEA_ERROR_INIT` error. If the values are within the range, the touch library computes the CRC for the input configuration data. The FMEA validates the CRC value passed by the touch library by performing double inverse check. If the double inverse check fails, the `FMEA_ERROR_INIT` is reported in the variable `sf_xxxxcap_fmea_fault_report_var`. This function must be called after performing the touch initialization. The application should check the variable `sf_xxxxcap_fmea_fault_report_var` after calling this function and ensure that the initialization has not failed.

```
void sf_xxxxcap_fmea_init(sf_xxxxcap_fmea_config_t sf_xxxxcap_fmea_input_config)
```

Fields	Type	Description
`sf_xxxxcap_fmea_input_config`	`sf_xxxxcap_fmea_input_config_t`	The input parameters are passed through this structure

Return: None.

5.5.2 sf_xxxxcap_fmea_test

This function performs various FMEA tests based on the input parameter and updates the global structure `sf_xxxxcap_fmea_fault_report_var` which contains the FMEA fault status.

```
void sf_xxxxcap_fmea_test(uint16_t select_checks)
```

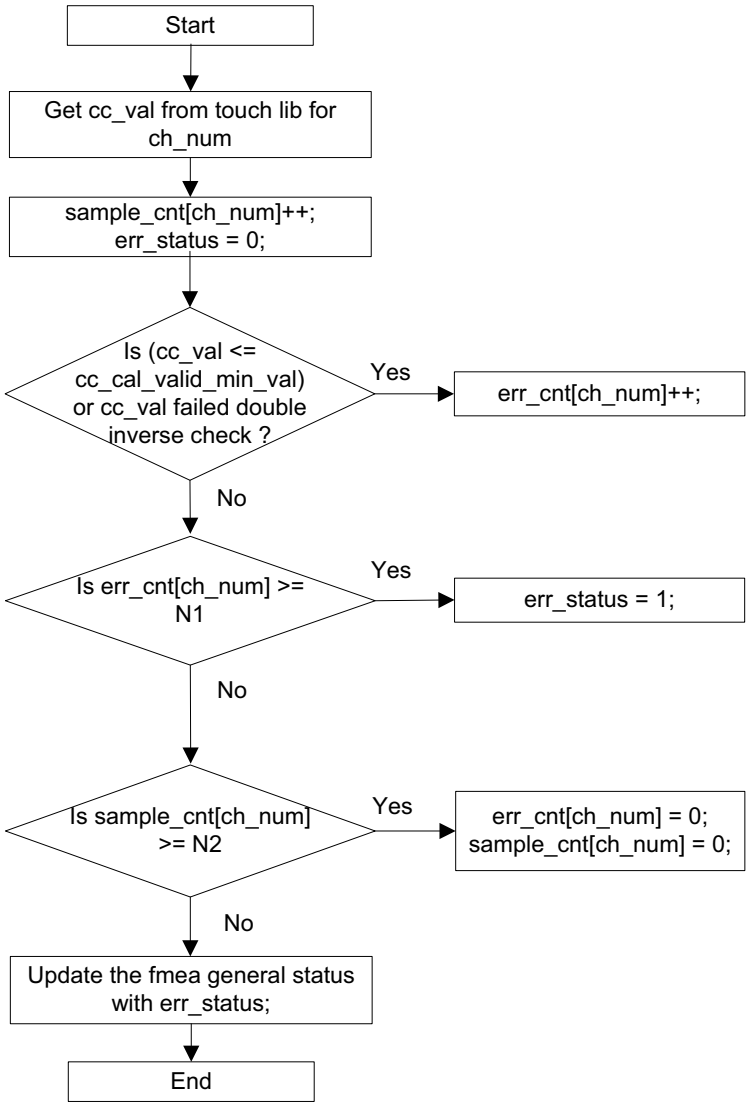
Fields	Type	Description
`select_checks`	`uint16_t`	Bit masks of the tests that must be performed. • If bit 0 is set as 1, Short to Vcc test is performed. • If bit 1 is set as 1, Short to Vss test is performed. • If bit 2 is set as 1, Short to Pins test is performed. • If bit 3 is set as 1, PTC register test is performed. • If bit 4 is set as 1, input configuration data integrity test is performed. • If any bit is set to 0, the corresponding FMEA test is not performed. • Bit 5 to 15 are reserved in this field. The application should not call this function by setting them.

Return: None.

5.5.3 sf_xxxcap_fmea_test_open_pins_per_channel

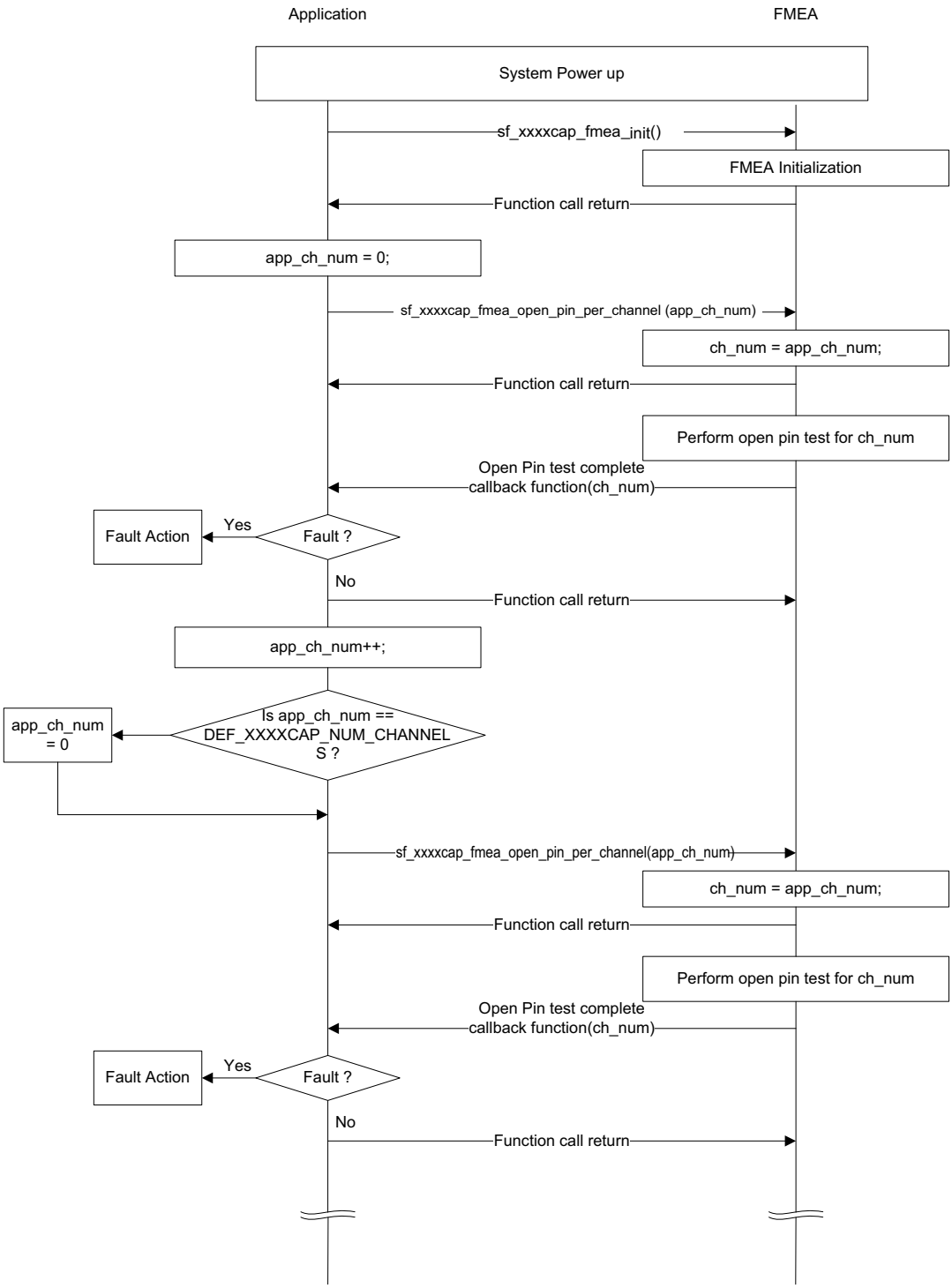
Open pin test is performed by receiving the CC value for the current channel number from touch library. If the CC value received from the touch library is less than or equal to the configured minimum value, then the error counter for that channel is incremented. Error counter will also be incremented if double inverse check of the CC value is failed. If the error counter reaches the configured minimum number of error count, then the `FMEA_ERR_OPEN_PINS` error is updated in `sf_xxxxcap_fmea_fault_report_var` and the sample and error counter of that channel is reset to zero. If the sample counter reaches the configured maximum number of channels, then the error counter and sample counter are reset to zero.

Figure 5-1. Working Mechanism of the Error and Sample Counter



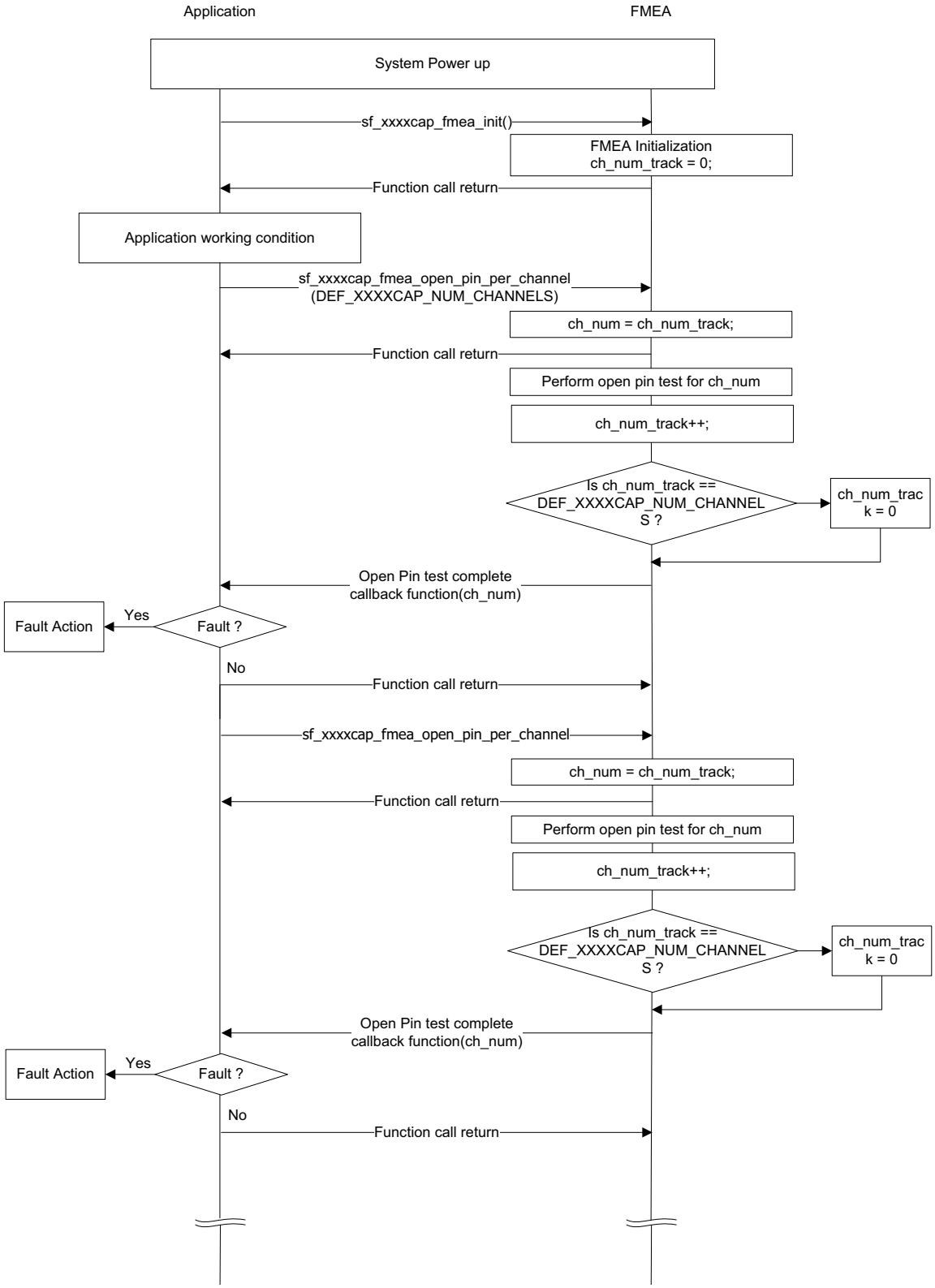
This API can be called using one of the three modes.

Mode 1: Application Tracking the Next Channel Number



If the channel number passed as parameter is less than `DEF_XXXXCAP_NUM_CHANNELS`, this function performs open pin test for the specified channel number. In this mode, the application can decide the channel number to be tested during each run.

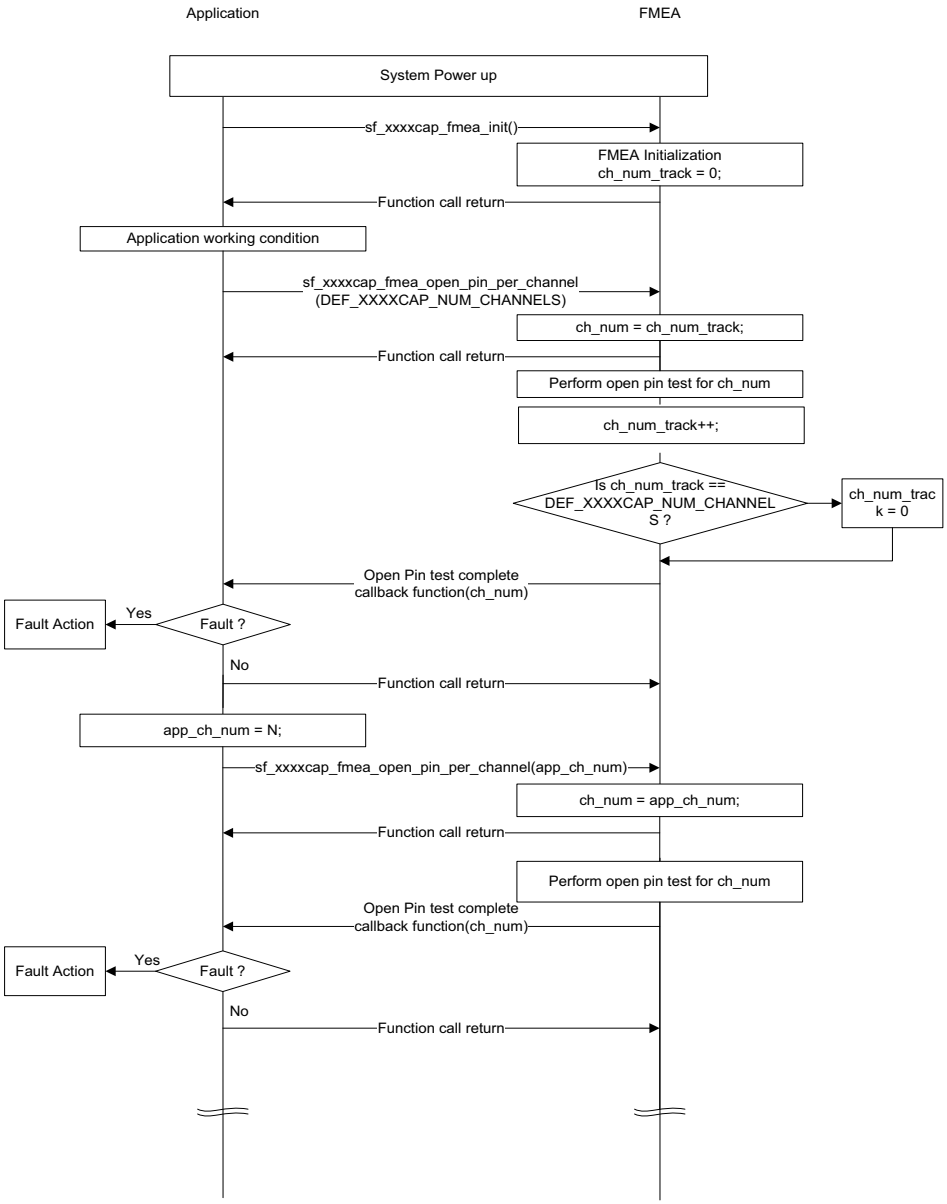
Mode 2: FMEA Tracking the Next Channel Number



The application can let the FMEA to track the channel number by passing `DEF_XXXXCAP_NUM_CHANNELS` as the input value. For each call to `sf_xxxxcap_fmea_open_pins_per_channel` with `DEF_XXXXCAP_NUM_CHANNELS` as the

input value, open pin test will be performed on one channel (referred as sf_xxxxcap_fmea_open_test_ch_track). At FMEA initialization, sf_xxxxcap_fmea_open_test_ch_track is initialized to 0. After each test, sf_xxxxcap_fmea_open_test_ch_track is incremented by 1. When sf_xxxxcap_fmea_open_test_ch_track reaches DEF_XXXXCAP_NUM_CHANNELS, it is reset to 0.

Mode 3: Both FMEA and Application tracking the channel number



In mode 3, sf_xxxxcap_fmea_test_open_pins_per_channel() can be called with input parameter value in the range of 0 to DEF_XXXXCAP_NUM_CHANNELS.

Whenever the input parameter value is in the range of 0 to DEF_XXXXCAP_NUM_CHANNELS-1, this function performs open pin test for the specified channel number.

Whenever the input parameter value is equal to DEF_XXXXCAP_NUM_CHANNELS, open pin test will be performed on one channel number sf_xxxxcap_fmea_open_test_ch_track. sf_xxxxcap_fmea_open_test_ch_track is

incremented by after performing the test. If the `sf_xxxxcap_fmea_open_test_ch_track` is equal to or greater than `DEF_XXXXCAP_NUM_CHANNELS`, then `sf_xxxxcap_fmea_open_test_ch_track` reset to 0.

In all these modes, the application should initiate the next open pin test only after receiving the callback function for the previously initiated open pin test.

void sf_xxxxcap_fmea_test_open_pins_per_channel (uint16_t ch_num)

If the channel number passed is greater than `DEF_XXXXCAP_NUM_CHANNELS`, then the `sf_xxxxcap_fmea_fault_report_var` is updated with `FMEA_ERR_PRE_TEST` error.

Return:

None.

> The `sf_xxxxcap_fmea_test_open_pins_per_channel()` calls the open pin test complete callback function after performing open pin test for the specified channel. The application should check the open pin test status only after the open pin test complete callback function is called.

void sf_xxxxcap_fmea_test_open_pins_per_channel (uint16_t ch_num)

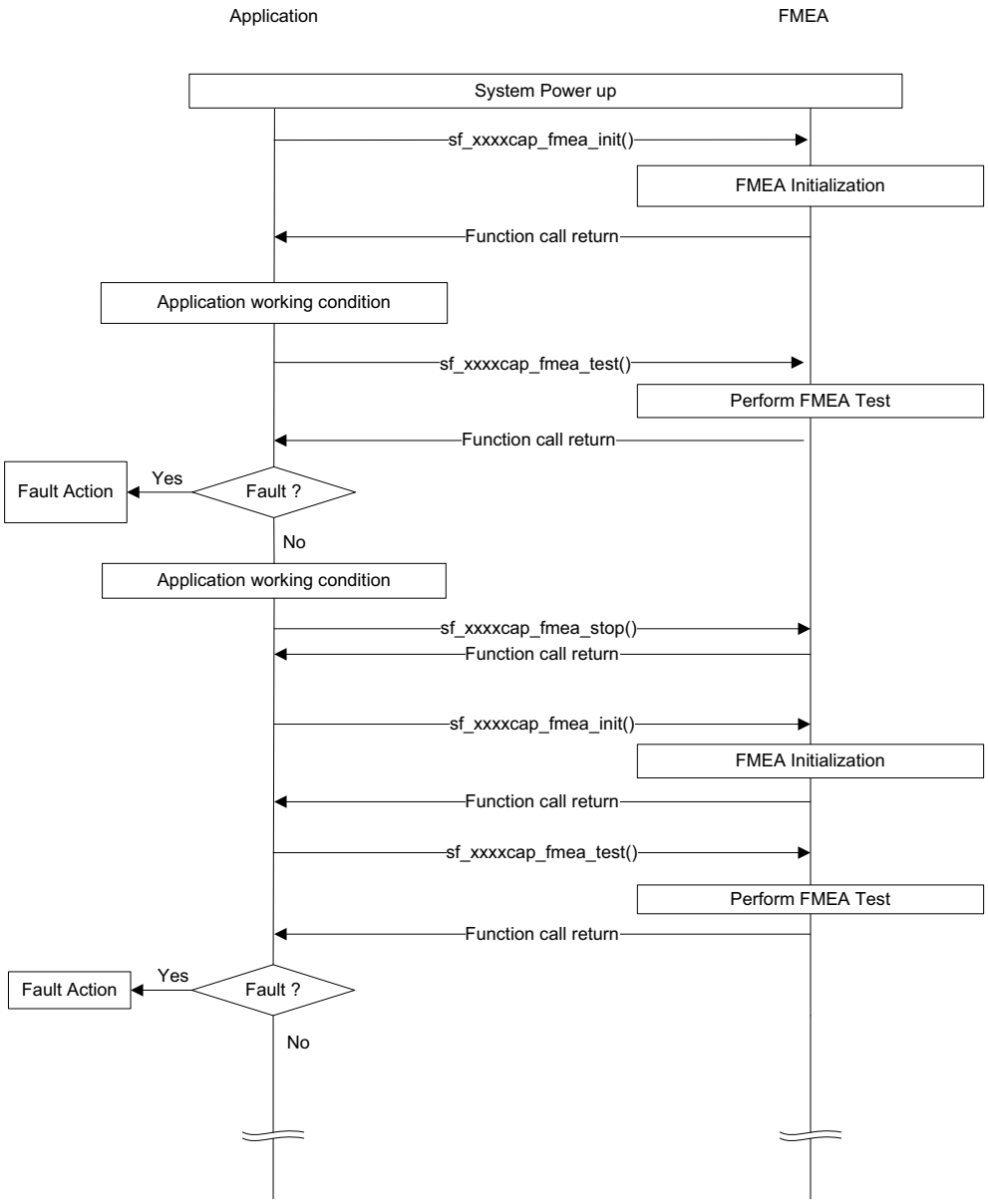
Data Fields

Arguments	Type	Description
`ch_num`	`unint16_t`	Channel number for which the open pin test must be performed

Return: None**.**

The `sf_xxxxcap_fmea_test_open_pins_per_channel()` calls the open pin test complete callback function after performing open pin test for the specified channels. The application should check the open pin test status only after the open pin test complete callback function is being called for the respective touch acquisition technology.

5.5.4 sf_xxxxcap_fmea_stop



This function stops the FMEA component operation and change the FMEA init status to uninitialized state. The global variables used by the FMEA are reset to default value. The application cannot execute further FMEA tests without re-initializing the FMEA component.

```
void sf_xxxxcap_fmea_stop (void)
```

Arguments	Type	Description
None	None	None

Return: None.

5.6 Macros

`DEF_TOUCH_FMEA_MUTLCAP_ENABLE` and `DEF_TOUCH_FMEA_SELFCAP_ENABLE` must be set to 1 to enable mutual cap and self cap FMEA respectively.

6. System

6.1 Relocating Touch Library and FMEA RAM Area

The data corresponding to the touch library and FMEA are placed at specific sections in the RAM.

This is done so that the customer application can perform the static memory analysis test on the touch and FMEA RAM area as per the Class B safety requirements.

To create these two RAM sections (Touch and FMEA), the linker file must be modified as per the description in the following sections.

Notes:
1. All the variables related to touch sensing (filter callback, touch input configuration, gain variables and others) in `touch.c` application file must be re-located to touch library RAM section.
2. Following warning may be displayed in IAR IDE:
   Warning[Be006]: possible conflict for segment/section.
   This warning is thrown due to relocation of configuration variables in `touch.c` and FMEA variables which contains both initialized and zero initialized data to the `TOUCH_SAFETY_DATA_LOCATION` and `TOUCH_FMEA_DATA_LOCATION` sections, respectively.
   This warning will not affect the safe operation of the system.
   This warning can be safely discarded or if required the same can be suppressed using diagnostic tab in IAR project options.

6.1.1 Modifying the IAR Linker File

Touch Library RAM Section

The changes should be done in `<devicevariant>_flash.icf` file as follows:

Linker symbols should be added in linker file to denote the start and size of the touch library RAM section. The size of touch RAM section (`SIZE_OF_TOUCH_SAFETY_DATA_LOCATION`) should be calculated as per Section 2.10.

Table 6-1. IAR Linker Symbols for Touch RAM Data

Symbol in Linker File	Description
`TOUCH_SAFETY_DATA_LOCATION_region`	Touch Library Data Memory Region to be created in linker file.
`TOUCH_SAFETY_DATA_LOCATION`	Touch library Data Section to be created in linker file
`SIZE_OF_TOUCH_SAFETY_DATA_LOCATION`	Size of Touch Library RAM data
`TOUCH_SAFETY_DATA_LOCATION_START`	The absolute address of RAM from where touch library RAM variables would be placed in `TOUCH_SAFETY_DATA_LOCATION` section
`TOUCH_SAFETY_DATA_LOCATION_END`	End location of the `TOUCH_SAFETY_DATA_LOCATION` section

An example setting is as follows:

```
define symbol TOUCH_SAFETY_DATA_LOCATION_START = 0x20004000;

define symbol SIZE_OF_TOUCH_SAFETY_DATA_LOCATION = 0x05DC;

define symbol TOUCH_SAFETY_DATA_LOCATION_END = (TOUCH_SAFETY_DATA_LOCATION_START +
SIZE_OF_TOUCH_SAFETY_DATA_LOCATION -1);
```

FMEA RAM Section

Linker symbols should be added in linker file to denote the start and size of the FMEA library RAM section. The size of FMEA RAM section (`SIZE_OF_FMEA_SAFETY_DATA_LOCATION`) should be calculated as per section Section 4.2.

Table 6-2. IAR Linker Symbols for FMEA RAM Data

Symbol in Linker File	Description
`FMEA_SAFETY_DATA_LOCATION_region`	FMEA Library Data Memory Region to be created in linker file
`FMEA_SAFETY_DATA_LOCATION`	FMEA library Data Section to be created in linker file
`SIZE_OF_FMEA_SAFETY_DATA_LOCATION`	Size of FMEA Library RAM data
`FMEA_SAFETY_DATA_LOCATION_START`	The absolute address of RAM from where FMEA library RAM variables would be placed in `FMEA_SAFETY_DATA_LOCATION` section
`FMEA_SAFETY_DATA_LOCATION_END`	End location of the `FMEA_SAFETY_DATA_LOCATION` section

An example setting is as follows:

```
define symbol FMEA_SAFETY_DATA_LOCATION_START = 0x20004000;

define symbol SIZE_OF_FMEA_SAFETY_DATA_LOCATION = 0x05DC;

define symbol FMEA_SAFETY_DATA_LOCATION_END = (FMEA_SAFETY_DATA_LOCATION_START +
SIZE_OF_FMEA_SAFETY_DATA_LOCATION -1);
```

Note: More information can be found at page 85, Linking Your Application in [3]. Refer [4] for the version of IAR tool-chain used.

6.1.2 Modifying GCC Linker File

The changes should be done in `<devicevariant>_flash.ld` file as follows:

Touch Library RAM Section

Symbol in Linker File	Description
TOUCH_SAFETY_DATA_LOCATION_region	Touch Library Data Memory Region to be created in linker file. The ORIGIN field in the memory region should be the starting address of the touch library RAM data and LENGTH field should be the size of the touch library RAM data.
TOUCH_SAFETY_DATA_LOCATION	Touch library Data Section to be created in linker file
SIZE_OF_TOUCH_SAFETY_DATA_LOCATION	Size of Touch Library RAM data
TOUCH_SAFETY_DATA_LOCATION_START	The absolute address of RAM from where Touch library RAM variables would be placed in TOUCH_SAFETY_DATA_LOCATION section
TOUCH_SAFETY_DATA_LOCATION_END	End location of the TOUCH_SAFETY_DATA_LOCATION section
_sTOUCH_SAFETY_DATA_LOCATION	It holds the start address of the TOUCH_SAFETY_DATA_LOCATION in FLASH
_eTOUCH_SAFETY_DATA_LOCATION	It holds the end address of the TOUCH_SAFETY_DATA_LOCATION in FLASH

The TOUCH_SAFETY_DATA_LOCATION_START, _sTOUCH_SAFETY_DATA_LOCATION, TOUCH_SAFETY_DATA_LOCATION_END and _eTOUCH_SAFETY_DATA_LOCATION variables would be used in the startup_samd20.c or startup_samd21.c file to initialize the Touch library RAM section from FLASH.

FMEA Library RAM Section

Symbol in Linker File	Description
`FMEA_SAFETY_DATA_LOCATION_region`	FMEA Library Data Memory Region to be created in linker file. The ORIGIN field in the memory region should be the starting address of the FMEA library RAM data and LENGTH field should be the size of the FMEA library RAM data.
`FMEA_SAFETY_DATA_LOCATION`	FMEA library Data Section to be created in linker file
`SIZE_OF_FMEA_SAFETY_DATA_LOCATION`	Size of FMEA Library RAM data
`FMEA_SAFETY_DATA_LOCATION_START`	The absolute address of RAM from where FMEA library RAM variables would be placed in `TOUCH_SAFETY_DATA_LOCATION` section
`FMEA_SAFETY_DATA_LOCATION_END`	End location of the `FMEA_SAFETY_DATA_LOCATION` section
`_sFMEA_SAFETY_DATA_LOCATION`	It holds the start address of the `FMEA_SAFETY_DATA_LOCATION` in FLASH
`_eFMEA_SAFETY_DATA_LOCATION`	It holds the end address of the `FMEA_SAFETY_DATA_LOCATION` in FLASH

The `FMEA_SAFETY_DATA_LOCATION_START`, `_sFMEA_SAFETY_DATA_LOCATION`, `FMEA_SAFETY_DATA_LOCATION_END` and `_eFMEA_SAFETY_DATA_LOCATION` variables would be used in the `startup_samd20.c` or `startup_samd21.c` file to initialize the FMEA library RAM section from FLASH.

Note: More information can be found on linker script at page 37 in [6].

6.2 API Rules

All safety APIs must be incorporated in to a system as per the following rules:

a. Both FMEA and Touch library must be initialized at least once after power-up. FMEA can be initialized again after stopping the FMEA.
b. The periodicity for calling safety test APIs is controlled by the application.
c. Few safety test APIs will lock interrupts during the test period since interrupts could potentially disrupt the safety functionality. Refer Section 2.11 for information about Touch Library.

FMEA component is functionally dependent on Atmel touch library. Hence FMEA test must be performed only after the touch library is initialized. Touch library is a pre-requisite for FMEA firmware, Include the FMEA firmware, only when the Touch library is included in the system.

6.3 Safety Firmware Action Upon Fault Detection

On detection of a fault within an IEC safety test API, the safety firmware can perform the corrective action.

A. **Touch library action upon fault detection.**

B. **FMEA library action upon fault detection.** If a fault is detected by the FMEA library, it will update the fault in the global structure `sf_xxxxcap_fmea_fault_report_var`.

6.4 System Action Upon Fault Detection

The fault action routine must be designed by the user and will be system dependent. The following options can be considered for the fault actions routines:

a) Application may inform the host about the failure, provided the failure does not impact the communication with the host controller.

b) Lock the system by disabling interrupt. Perform other possible clean-up actions and lock the system.

c) The system can clean-up and shutdown other safety systems and reset the system.

6.5 Touch Library and FMEA Synchronization

The following entities are mutually exclusive and cannot be executing an activity (touch measurement or FMEA test) simultaneously.

- Self-cap touch library
- Mutual-cap touch library
- Self-cap FMEA
- Mutual-cap FMEA

The customer application should establish a synchronization mechanism to manage the exclusivity of the entities.

The following tables provides the information about the FMEA APIs, Touch library APIs and their corresponding action to indicate completion.

Table 6-3. FMEA API Execution Completion Indicators

API Name	Completion Indication
`sf_xxxxcap_fmea_init`	Function call return
`sf_xxxxcap_fmea_test`	Function call return
`sf_xxxxcap_fmea_test_open_pin_per_channel`	Open pin test complete callback function call
`sf_xxxxcap_fmea_stop`	Function call return

Table 6-4. API Execution Completion Indicators

API	Completion Indication
touch_xxxxcap_sensors_init	Function call return
touch_xxxxcap_di_init	Function call return
touch_xxxxcap_sensor_config	Function call return
touch_xxxxcap_sensors_calibrate	Measure complete callback function call
touch_xxxxcap_calibrate_single_sensor	Measure complete callback function call
touch_xxxxcap_sensors_measure	Measure complete callback function call with Application burst again set to zero
touch_xxxxcap_sensor_get_delta	Function call return
touch_xxxxcap_sensor_update_config	Function call return
touch_xxxxcap_sensor_get_config	Function call return
touch_xxxxcap_sensor_update_acq_config	Function call return
touch_xxxxcap_sensor_get_acq_config	Function call return
touch_xxxxcap_update_global_param	Function call return
touch_xxxxcap_get_global_param	Function call return
touch_xxxxcap_get_libinfo	Function call return
touch_lib_pc_test_magic_no_1	Function call return
touch_lib_pc_test_magic_no_2	Function call return
touch_lib_pc_test_magic_no_3	Function call return
touch_lib_pc_test_magic_no_4	Function call return
touch_xxxxcap_sensor_disable	Function call return
touch_xxxxcap_sensor_reenable	Function call return
touch_library_get_version_info	Function call return
touch_xxxxcap_cnfg_mois_mltchgrp	Function call return
touch_xxxxcap_cnfg_mois_threshold	Function call return
touch_xxxxcap_mois_tolrnce_enable	Function call return
touch_xxxxcap_mois_tolrnce_disable	Function call return
touch_calc_xxxxcap_config_data_integrity	Function call return
touch_test_xxxxcap_config_data_integrity	Function call return

6.6 Safety Firmware Package

The following files corresponding to the safety component.

Safety Component	Files

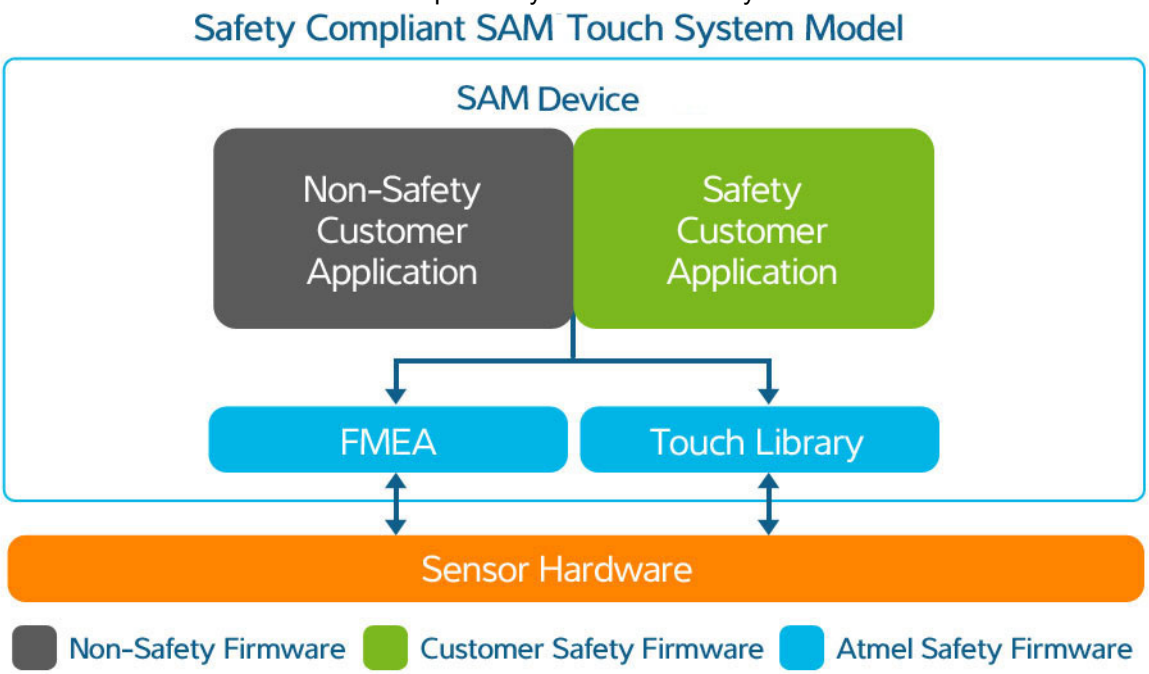
FMEA	`sf_mutlcap_fmea_samd.c`
	`sf_selfcap_fmea_samd.c`
	`touch_fmea_api_samd.h`
	`sf_fmea_samd_api.h`
	`sf_fmea_samd_int.h`

Touch Library	`libsamd20_safety_iar.a`
	`libsamd20_safety_gcc.a`
	`libsamd21_safety_iar.a`
	`libsamd21_safety_gcc.a`
	`touch_safety_api_samd.h`

6.7 SAMDSafety Firmware Certification Scope

The Class-B IEC certification of the following modules are supported and compiled by FMEA and Safety Touch Library. The following activities must be performed by the user to achieve IEC certification for the overall system:

- Risk analysis for the system
- IEC certification for the critical and supervisory sections of the system



6.8 Hazard Time

It is the responsibility of the application to ensure that the optimal configuration is selected for the individual test components (FMEA) to achieve the hazard time requirement of the end user system as per the [1] and [2].

Note: The hazard time for various types of failure is not defined by Atmel. It is based on the test configuration and periodicity selected by the user designing the end user system or application.

6.9 ASF Dependency

The Atmel® Software Framework (ASF) is a MCU software library providing a large collection of embedded software for different Atmel MCUs. It simplifies the usage of microcontrollers, providing an abstraction to the hardware and high-value middle wares. The Touch Library and FMEA is dependent on the ASF.

ASF is available as standalone package for IAR compilers and can be downloaded from Atmel website. For more information and an overview about ASF visit: http://www.atmel.com/tools/ AVRSOFTWAREFRAMEWORK.aspx.

The latest ASF standalone package is available for download in the download page in the *Software Category* in www.atmel.com.

6.10 Robustness and Tuning

Please refer *AT08578: SAM D20 QTouch Robustness Demo* User Guide and *AT09363: PTC Robustness Design Guide.*

6.11 Standards compliance

Atmel Safety Library is compliant with the following list of IEC, EN and UL standards.

UL Compliance

- UL 60730-1, IEC 60730-1 and CSA E60730-1, Automatic electrical controls
- UL 60335-1 and IEC 60335-1, Household and similar electrical appliances
- UL 60730-2-11 and IEC 60730-2-11, Energy Regulators
- UL 1017 and IEC 60335-2-2, Vacuum Cleaners and Water-Suction Cleaning Appliances
- UL 749, UL 921, and IEC 60335-2-5, Dishwashers
- UL 858 and IEC 60335-2-6, Stationary Cooking Ranges, Hobs, Ovens, and Similar Appliances
- UL 1206, UL 2157, and IEC 60335-2-7, Washing Machines
- UL 1240, UL 2158, and IEC 60335-2-11, Tumble Dryers
- UL 1083 and IEC 60335-2-13, Deep Fat Fryers, Frying Pans, and Similar Appliances
- UL 982 and IEC 60335-2-14, Kitchen Machines
- UL 1082 and IEC 60335-2-15, Appliances for Heating Liquids
- UL 923 and IEC 60335-2-25, Microwave Ovens, Including Combination Microwave Ovens
- UL 197 and IEC 60335-2-36, Commercial Electric Cooking Ranges, Ovens, Hobs, and Hob Elements
- UL 197 and IEC 60335-2-37, Commercial Electric Doughnut Fryers and Deep Fat Fryers
- UL 73, UL 499, and IEC 60335-2-54, Surface-Cleaning Appliances for Household Use Employing Liquids or Steam
- UL 499, UL 1776, and IEC 60335-2-79, High Pressure Cleaners and Steam Cleaners
- UL 507 and IEC 60335-2-80, Fans

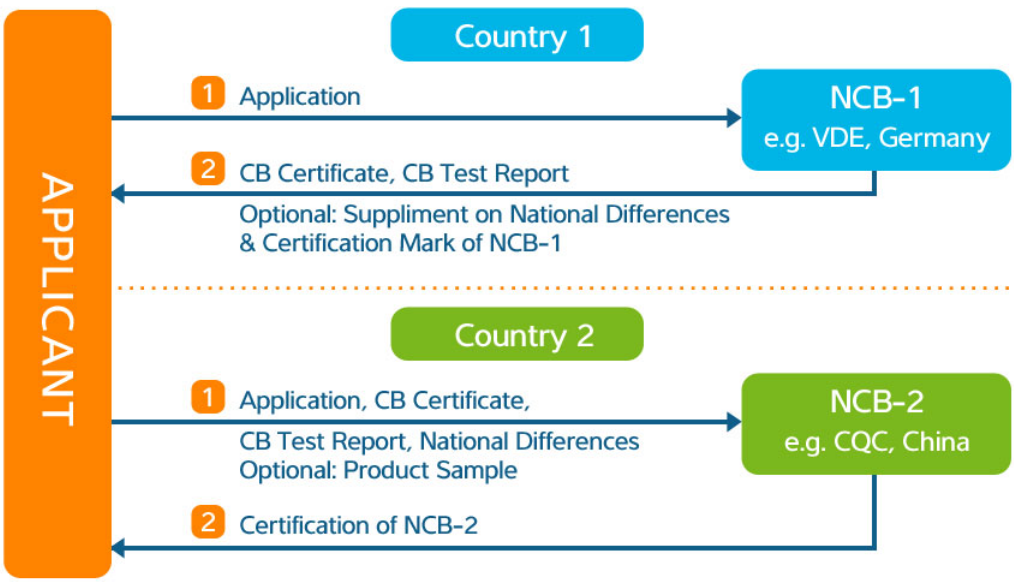
VDE Compliance

- IEC/EN 60730-1, Automatic electrical controls
- IEC/EN 60335-2-11, Energy regulators
- IEC/EN 60335-1, Safety of household appliances
- IEC/EN 60335-2-5, Dishwashers
- IEC/EN 60335-2-6, Hobs, ovens and cooking ranges
- IEC/EN 60335-2-7, Washing machines
- IEC/EN 60335-2-9, Grills, toasters and similar portable cooking appliances
- IEC/EN 60335-2-14 Kitchen machines
- IEC/EN 60335-2-15,Heating liquids
- IEC 60335-2-25 Microwave ovens including combination micro wave ovens
- IEC 60335-2-33 Coffeemills and coffee
- IEC 60335-2-36 Commercial electric cooking ranges, ovens, hobs and hob elements
- IEC 60730-2-11 Energy regulators

6.12 Safety Certification

A Safety Certification "mark" on a product indicates that it has been tested against the applicable safety in a certain region and found to be in compliance. A National Certification Body (NCB) is an organization that grants nationally recognized conformity certificates and marks to products such asVDE and UL are NCBs in Germany and USA, respectively.

The IECEE CB Scheme is an international system for mutual acceptance of test reports and certificates dealing with the safety of electrical and electronic components, equipment and products. The tests performed by one national NCB and the resulting CB-certificates / test reports are the basis for obtaining the national certification of other participating NCBs, subject to any National Differences being met. The following diagram illustrates the typical CB scheme flow.



7. Known Issues

The following errata is applicable for the QTouch Safety Library versions up to 5.1.4.

Issue:

Touch acquisition may fail and stop working.

Description:

In QTouch applications, where either a single interrupt or a chain of nested non-PTC interrupts has duration longer than the total touch measurement time, the touch acquisition may fail and stop working.

This issue occurs most likely in applications with few touch channels (2-3 channels) and a low level of noise handling (filter level 16 or lower and no frequency hopping).

Fix/workaround:

1. Always ensure that the duration of a single interrupt or a chain of nested non-PTC interrupts does not exceed the total touch measurement time. (or)
2. Add a critical section by disabling interrupts for the `touch_xxxxcap_sensors_measure()` function as shown in the following code snippet.

```
Disable_global_interrupt();
touch_ret = touch_xxxxcap_sensors_measure(current_time, NORMAL_ACQ_MODE,
touch_xxxxcap_measure_complete_callback);
Enable_global_interrupt();
```

The Interrupt Blocking Time while executing touch_xxxxcap_sensors_measure API for various CPU frequencies are as follows.

CPU Frequency (in MHz)	Interrupt Blocking Time (in µs)
48	~96
24	~162
16	~229
12	~295

The Interrupt Blocking Time varies based on the PTC_GCLK frequency, CPU frequency, and the library version. The actual blocking time can be measured by toggling a GPIO pin before and after calling the touch_xxxxcap_sensors _measure function.

If you are using an IAR compiler, then use `system_interrupt_enable_global()` and `system_interrupt_disable_global()` functions to enable and disable the global interrupts, respectively.

8. References

For more information and knowledge about the safety component for SAM devices, refer the following:

- [1]: IEC 60730-1: IEC60730-1 Standard for Safety for Software in Programmable Components
- [2]: SAMD20 device data sheet (http://www.atmel.com/Images/Atmel-42129-SAM-D20_Datasheet.pdf)
- [3]: IAR C/C++ Compiler Guide (http://supp.iar.com/FilesPublic/UPDINFO/004916/arm/doc/EWARM_DevelopmentGuide.ENU.pdf)
- [4]: IAR Embedded Workbench for ARM – Version 7.40
- [5]: Buttons, Sliders and Wheels Touch Sensor Design Guide (http://www.atmel.com/Images/doc10752.pdf)
- [6]: GCC Linker pdf (https://sourceware.org/binutils/docs/ld/)
- [7]: SAMD21 device data sheet (http://www.atmel.com/Images/Atmel-42181-SAM-D21_Datasheet.pdf)

9. Revision History

Doc. Rev.	Date	Comments
G	07/2016	Added Section Known Issues
F	10/2015	Minor non-technical updates
E	09/2015	Added AutoOversample, Filter level, PTC Resistor selection and PTC Pre-scaler on a per channel basis, and Touch De-init APIs
D	05/2015	Included Suspend and Resume PTC Operation Feature
C	11/2014	Enhanced release with Noise Immunity and Moisture Control Features
B	04/2014	Enhanced release with Noise Immunity Features
A	01/2014	Initial document release

Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**

© 2016 Atmel Corporation. / Rev.: Atmel-42230G-SAM-Atmel-SAMD-QTouch-Safety-Library-Datasheet_07/2016.

Atmel®, Atmel logo, Enabling Unlimited Possibilities®, and combinations thereof, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. ARM®, and Cortex®, are registered trademarks of ARM Limited. Other terms and product names may be trademarks of others.

DISCLAIMER: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and products descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

SAFETY-CRITICAL, MILITARY, AND AUTOMOTIVE APPLICATIONS DISCLAIMER: Atmel products are not designed for and will not be used in connection with any applications where the failure of such products would reasonably be expected to result in significant personal injury or death (“Safety-Critical Applications”) without an Atmel officer's specific written consent. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Atmel products are not designed nor intended for use in military or aerospace applications or environments unless specifically designated by Atmel as military-grade. Atmel products are not designed nor intended for use in automotive applications unless specifically designated by Atmel as automotive-grade.