Description

Atmel® QTouch® Peripheral Touch Controller (PTC) offers built-in hardware for capacitive touch measurement on sensors that function as buttons, sliders, and wheels. The 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 autonomously performing capacitive touch sensor measurements. 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.

Features

- Implements low-power, high-sensitivity, environmentally robust capacitive touch buttons, sliders, and wheels
- Supports mutual capacitance and self-capacitance sensing
- Up to 32 buttons in self-capacitance mode
- Up to 256 buttons in mutual capacitance mode
- Supports lumped mode configuration
- One pin per electrode - 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 de-synchronization for high conducted immunity
- Atmel provided QTouch Library firmware and QTouch Composer tool
**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 do not have a myAtmel account, please visit [http://www.atmel.com/design-support/](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.

When 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.
# Table of Contents

Description................................................................................................................................1

Features........................................................................................................................................1

1. Development Tools .......................................................................................... 5

2. Device Variants Supported ........................................................................ 6

3. Capacitive Touch Technology................................................................. 8
   3.1. Capacitive Touch Sensors.......................................................................... 8
   3.2. Capacitance Measurement Methods..................................................... 8
   3.3. Self-capacitance Measurement Method.............................................. 8
   3.4. Mutual Capacitance Measurement Method..................................... 9
   3.5. Capacitive Touch Lumped Sensors.................................................... 9
   3.6. Capacitive Touch Low Power Sensor ................................................. 11
   3.7. PTC and its Benefits.............................................................................. 13
   3.8. PTC Block Diagram for Self-capacitance and Mutual Capacitance Method ...................................... 13
   3.9. Design Approach with PTC............................................................... 15
   3.10. Capacitive Touch Development Cycle.......................................... 16

4. Touch Sensor Debug and Status Information......................................... 17
   4.1. Signal..................................................................................................... 17
   4.2. Reference................................................................................................. 17
   4.3. Delta........................................................................................................ 18
   4.4. Touch Status & Slider/Wheel Position................................................. 19

5. QTtouch Library......................................................................................... 20
   5.1. Overview............................................................................................... 20
   5.2. Library Parameters............................................................................... 21
   5.3. Moisture Tolerance............................................................................... 42
   5.4. Reading Sensor States........................................................................... 44
   5.5. Application Flow.................................................................................... 44
   5.6. API Sequence......................................................................................... 46
   5.7. State Machine......................................................................................... 47
   5.8. Operation Modes................................................................................... 50
   5.9. Touch Library API Error........................................................................ 52

6. Tuning for Noise Performance................................................................. 54
   6.1. Noise Sources....................................................................................... 54
   6.2. Noise Counter Measures....................................................................... 54

7. Application Design....................................................................................... 60
   7.1. Touch Library and Associated Files................................................. 60
   7.2. Code and Data Memory Considerations.......................................... 60

8. Example Applications................................................................................... 63
1. Development Tools

The following development tools are required for developing QTouch library using PTC:

- Development Environment for GCC Compiler:
  - QTouch Composer 5.9.116 or later versions
  - QTouch Library 5.9.211 or later versions
  **Note:** The QTouch Library and Composer extensions work only with Atmel Studio 7 which can be downloaded from [http://www.atmel.com/](http://www.atmel.com/)
  - Dependent Atmel Studio Extensions
    - Atmel Software Framework 3.30.1 or later versions
    - Atmel Kit Extension 7.0.70 or later versions
- Development Environment for IAR Compiler:
  - IAR Embedded Workbench® for ARM® 7.50.1.10273 or later
  - IAR Embedded Workbench for Atmel AVR® 6.70.1 or later
  - Atmel Software Framework 3.29.0 or later (optional)
## Device Variants Supported

QTouch Library for SAM and ATmega devices are available for the following device variants:

<table>
<thead>
<tr>
<th>Series</th>
<th>Variant</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAM D20 E Series</td>
<td>ATSAMD20E18, ATSAMD20E17, ATSAMD20E16, ATSAMD20E15, ATSAMD20E14</td>
</tr>
<tr>
<td>SAM D10 C Series</td>
<td>ATSAMD10C14A</td>
</tr>
<tr>
<td>SAM D10 D Series</td>
<td>ATSAMD10D14AM, ATSAMD10D14AS, ATSAMD10D14AU</td>
</tr>
<tr>
<td>SAM D11 C Series</td>
<td>ATSAMD11C14A</td>
</tr>
<tr>
<td>SAM D11 D Series</td>
<td>ATSAMD11D14AM, ATSAMD11D14AS, ATSAMD11D14AU</td>
</tr>
<tr>
<td>SAM L21 E Series</td>
<td>ATSAML21E15B, ATSAML21E16B, ATSAML21E17B, ATSAML21E18B</td>
</tr>
<tr>
<td>SAM L21 G Series</td>
<td>ATSAML21G16B, ATSAML21G17B, ATSAML21G18B</td>
</tr>
<tr>
<td>SAM L21 J Series</td>
<td>ATSAML21J16B, ATSAML21J17B, ATSAML21J18B</td>
</tr>
<tr>
<td>SAM R21 E Series</td>
<td>ATSAMR21E16A, ATSAMR21E17A, ATSAMR21E18A, ATSAMR21E19A</td>
</tr>
<tr>
<td>SAM R21 G Series</td>
<td>ATSAMR21G16A, ATSAMR21G17A, ATSAMR21G18A</td>
</tr>
<tr>
<td>SAM DA1 E Series</td>
<td>ATSAMDA1E14A, ATSAMDA1E15A, ATSAMDA1E16A</td>
</tr>
<tr>
<td>SAM DA1 G Series</td>
<td>ATSAMDA1G14A, ATSAMDA1G15A, ATSAMDA1G16A</td>
</tr>
<tr>
<td>SAM DA1 J Series</td>
<td>ATSAMDA1J14A, ATSAMDA1J15A, ATSAMDA1J16A</td>
</tr>
<tr>
<td>SAM C20 E Series</td>
<td>ATSAMC20E15A, ATSAMC20E16A, ATSAMC20E17A, ATSAMC20E18A</td>
</tr>
<tr>
<td>SAM C20 J Series</td>
<td>ATSAMC20J16A, ATSAMC20J17A, ATSAMC20J18A</td>
</tr>
<tr>
<td>SAM L22 G Series</td>
<td>ATSAML22G16A, ATSAML22G17A, ATSAML22G18A</td>
</tr>
<tr>
<td>SAM L22 J Series</td>
<td>ATSAML22J16A, ATSAML22J17A, ATSAML22J18A</td>
</tr>
<tr>
<td>Series</td>
<td>Variant</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>SAM L22 N Series</td>
<td>ATSAML22N16A, ATSAML22N17A, ATSAML22N18A</td>
</tr>
<tr>
<td>ATmega Series</td>
<td>ATmega328PB, ATmega324PB</td>
</tr>
</tbody>
</table>
3. Capacitive Touch Technology

3.1. Capacitive Touch Sensors
Capacitive touch sensors replace conventional mechanical interfaces and operate with no mechanical wear and are closed to the environment. They provide greater flexibility in industrial design and result in differentiating end product design. For more information, refer Capacitive Touch Lumped Sensors and Capacitive Touch Low Power Sensor.

Figure 3-1. Sensor Types

3.2. Capacitance Measurement Methods
Self-capacitance measurement method involves charging a sense electrode of unknown capacitance to a known potential. The resulting charge is transferred into a measurement circuit. By measuring the charge with one or more charge-and-transfer cycles, the capacitance of the sense plate can be determined.

Figure 3-2. Capacitance Measurement Principle

Mutual capacitance measurement method uses a pair of sensing electrodes. One electrode acts as an emitter into which a charge consisting of logic pulses is driven in burst mode. The other electrode acts as a receiver that couples to the emitter using the overlying panel dielectric. When a finger touches the panel, the field coupling is reduced, and touch is detected.

3.3. Self-capacitance Measurement Method
- Uses a single sense electrode (Y-line)
  - Self-capacitance button can be formed using one channel
  - Self-capacitance slider and wheel is formed using 3 channels
- Robust and easy to use, ideal for low sensors count
3.4. Mutual Capacitance Measurement Method

- Uses a pair of sense electrodes (X-Y lines)
  - Mutual capacitance buttons use one X-Y channel
  - Mutual capacitance sliders and wheels can be configured to use 3 to 8 X-Y channels, depending on the sensor size
- Suitable for high sensor count
- Better moisture tolerance

3.5. Capacitive Touch Lumped Sensors

Lumped sensor configuration is a combination of multiple sense lines (Self-capacitance measurement) or multiple drive and sense lines (Mutual capacitance measurement) to act as one single sensor. Lumped mode acts as a tool for application developers to improve overall system performance.

**Improved Power Efficiency**

When multiple sensors are lumped together and treated as one single sensor the time taken to perform scans is reduced. For battery powered applications using multiple buttons, a group of touch sensors can be lumped to form a single lumped sensor and this sensor alone can be scanned, thereby resulting in reduced power consumption. Upon user presence detection on the lumped sensor all configured sensors in the system can then be scanned individually.

**Improved Response Time**
In high key-count applications, there can be a significant latency between touching a sensor and the detection of a touch contact. This is due to the time taken to sequentially measure the capacitance of each key on each measurement cycle. With a Lumped mode implementation this latency can be reduced by arranging the sensors into groups. When one of those lumped groups shows touch detection, only the keys within that group are individually measured to determine which is touched.

E.g. A keyboard consisting 64 keys may be divided into 8 lumped groups of 8.

Thus, each measurement cycle is reduced to measure only the 8 lumped sensors. When a touch contact is applied, first the lump sensor shows touch delta, then the 8 component keys are scanned and the location is resolved. Only 16 measurements are required to resolve the touch status of all keys, compared to 64 measurements in the traditional sequential scan of all keys.

It offers an additional edge during low power acquisition as a group of keys [in lumped configuration] can be scanned thus reducing the power consumed drastically. Each sensor has its own pre-scaled clock and series resistor for improved noise immunity.

Figure 3-5. Self-capacitance Sensors connected to PTC

Figure 3-6. Lumped Self-capacitance Sensors connected to PTC

In the preceding figures, individual buttons are shown along with the lumped equivalent for self-capacitance arrangement.

**Lumped Mode Pin and Sensor Configuration for Self-capacitance Method:**

```c
#define DEF_SELFCAP_LINES Y(5), Y(4), Y(11), Y(10), Y(13), Y(7), Y(12), Y(6), LUMP_Y(5,4)
touch_ret = touch_selfcap_sensor_config(SENSOR_TYPE_LUMP, CHANNEL_8, CHANNEL_8, NO_AKS_GROUP, 40u, HYST_6_25, RES_8_BIT, &sensor_id);
```
In the preceding figure, mutual capacitance lumped sensor configuration is presented.

**Lumped Mode Pin and Sensor Configuration for Mutual Capacitance Method:**

```c
#define DEF_MUTLCAP_NODES X(8), Y(10), X(9), Y(10), X(2), Y(12), X(3), Y(12), \X(8), Y(12), 
X(9), Y(12), X(2), Y(13), X(3), Y(13), \X(8), Y(13), X(9), Y(13), LUMP_X(2,3,8,9), 
LUMP_Y(10,13)

touch_ret = touch_mutlcap_sensor_config(SENSOR_TYPE_LUMP, CHANNEL_10, CHANNEL_10, 
NO_AKS_GROUP, 20u, HYST_6_25, RES_8_BIT, 0, &sensor_id);
```

**Limitations of Use**

Lumped sensor capacitive load should not exceed the maximum sensor load for individual sensors in either mutual or self-capacitance modes. Lumped mode treats the larger sensors as one single sensor therefore the maximum lumped sensor load should also observe this specification, else this will result in calibration error.

In mutual capacitance measurement mode the capacitive load of each sensor is normally much lower than that of the self-capacitance method. It is therefore possible as a general rule to use more mutual sensors together as a single lumped sensor.

The user can ensure that the lumped sensor does not result in a calibration error (value of 0x80) using `p_xxxxcap_measure_data->p_sensors[<SENSOR>].state variable.`

### 3.6. Capacitive Touch Low Power Sensor

The QTouch Library may be configured to operate PTC touch sensing autonomously using the Event System. In this mode, a single sensor is designated as the ‘Low Power’ key and may be periodically measured for touch detection without any CPU action. The CPU may be held in deep sleep mode throughout the operation, minimizing power consumption.

The low power key may be a discrete electrode with one Y (Sense) line for Self-capacitance or One X (Drive) plus one Y (Sense) for mutual capacitance, or it may be a combination of multiple Drive and/or Sense lines as a Lumped mode sensor.
In the active measurement mode all configured sensors are measured at \( \text{DEF_TOUCH_MEASUREMENT_PERIOD_MS} \) millisecond scan interval.

The user application arrangement could be designed such that when no touch activity is detected on any of the configured sensors for \( \text{NO_ACTIVITY_TRIGGER_TIME} \) milliseconds, then the application switches to low power measurement mode.

In the low power measurement mode, a designated sensor or a lumped sensor can be scanned as a single sensor. In this mode, the system is in standby sleep mode, the CPU and other peripherals are in sleep, excepting for the event system, the RTC and the PTC module / WDT and PTC module in SAM / Mega devices. A user touch on the designated low power sensor will cause the CPU to wake up and perform active measurement in order to resolve the touch. To keep reference tracking of the designated low power sensor, the RTC/WDT is configured to periodically wake up the CPU every \( \text{DEF_LOWPOWER_SENSOR_DRIFT_PERIODICITY_MS} \) millisecond to perform one active measurement.

Switching between Active Mode and Low Power Mode
When switching from active to low power mode, all sensors except the lumped sensor are disabled. So, no reference tracking is performed on these sensors during the low power mode. When a touch is detected on the lumped sensor, all disabled sensors shall now be re-enabled and measurement is initiated on the sensors. If the device is in sleep for a very long time, then it is recommended to force calibration on the re-enabled sensors to ensure proper reference values on these sensors.

3.7. **PTC and its Benefits**

- Mixed Hardware + Firmware solution, allows user to define sensor configuration
  - Peripheral Touch Controller + QTouch library
- PTC runs data acquisition autonomously, resulting in low CPU utilization and power consumption
  - User controlled power-performance trade-off
  - CPU can sleep during acquisition to save power
  - Alternatively, CPU can perform other time critical operations during touch acquisition
- Robust noise performance

*Figure 3-9. User Application with PTC Device*

3.8. **PTC Block Diagram for Self-capacitance and Mutual Capacitance Method**

The PTC block diagram for self-capacitance measurement is shown in the following figure. Only Y-lines can be connected to self-capacitance sensors and are selected using the Input control. X-lines remain unused and can be used for any other GPIO functionality. The acquisition module along with the compensation circuit helps in measuring the change in capacitance due to user touch.
The PTC block diagram for mutual capacitance measurement is as shown in the following figure. Both X-lines and Y-lines should be connected to mutual capacitance sensors and are selected using the Input control.

The tag_touch_measure_data_t structure contains the p_cc_calibration_vals parameter which represents the current channel's compensation circuit value. For more information, refer Measure Data Type (tag_touch_measure_data_t).

Compensation circuit value used in pF = (p_cc_calibration_vals[channel_no] & 0x0F) * 0.00675 + ((p_cc_calibration_vals[channel_no] >> 4) & 0x0F) * 0.0675 + ((p_cc_calibration_vals[channel_no] >> 8) & 0x0F) * 0.675 + ((p_cc_calibration_vals[channel_no] >> 12) & 0x3) * 6.75

Also, the touch_xxxx_cap_sensors_calibrate function helps the user to calibrate the compensation circuit according to the sensors used. If the routine fails to calibrate the compensation circuit due to saturation, the measurement will return TOUCH_CC_CALIB_ERROR. The compensation circuit could have...
exceeded its limit. The specific sensor that has failed can be determined using `p_xxxxxcap_measure_data->p_sensors[<sensor>].state` when it contains the value of `SENSOR_CALIBRATION_ERROR(0x80u)`.

- Typical compensation circuit value for the self-capacitance mode ranges from 10 to 25 pF and for the mutual capacitance mode it is around 2 pF.
- The compensation circuit value is affected by sensor size and the ground surrounding the sensor or trace. The compensation circuit value ranges from 0.00675 pF to 31.48 pF.
- If the compensation circuit value exceeds the limit, to reduce the value, use a mesh instead of a solid plane in the sensor and ground plane.

### 3.9. Design Approach with PTC

Two design approaches are possible when using Atmel MCU along with PTC. The Atmel MCU could be predominantly used as an MCU for touch measurement. Else, the Atmel MCU can function as a Host MCU utilizing peripherals such as the USB, ADC, DAC, SERCOM, DMA and GPIO along with the PTC used for "on-chip" touch functionality.

The design approaches are:

- **Atmel MCU with PTC predominantly functioning as a touch MCU**
  - Used for touch sensor status and rotor/slider position detection
  - Additionally used to indicate touch status using LED, buzzer etc
  - Sends touch status and rotor/slider position information to a Host MCU

- **Atmel MCU functions as a Host MCU with on-chip touch functionality**
  - Can be a cost saving design as a single chip solution with on-chip touch functionality
  - Utilizes other on-chip peripheral for a desired user application

![PTC Design Approach](image-url)
3.10. Capacitive Touch Development Cycle

The capacitive touch development cycle involves PCB board design to develop the user interface hardware as well as firmware application development. The QTouch Composer PC software available as part of Atmel Studio extension gallery allows for PTC QTouch Library projects to be generated automatically with a desired user configuration for touch sensors. The QTouch Composer also allows for touch sensor data analysis and performance tuning for sensitivity and noise.

Figure 3-13. Capacitive Touch Development Cycle
4. **Touch Sensor Debug and Status Information**

The touch sensor debug information necessary for tuning of the sensors are signal, reference, delta, and compensation capacitance. While the signal, reference and delta help in sensitivity and noise tuning the sensor parameters, the compensation capacitance is an indicator for extreme sensor design. The sensor status and position information are parameters that must be judged by the user application to initiate the relevant touch action.

4.1. **Signal**

Signal value is the raw measurement data on a given touch channel. The value increases upon touch.

*Figure 4-1. Channel Signal*

4.2. **Reference**

Reference value of a touch channel is the long term average measurement on a specific channel. It represents:

- Resting signal when there is no touch
- Initial value obtained during the calibration process
- Reference is adapted by Drift Compensation algorithm
4.3. **Delta**

Delta value of a touch channel represents touch strength.

- Delta = (signal - reference)
- Deltas increase with touch

**Figure 4-3. Sensor Delta**
4.4. **Touch Status & Slider/Wheel Position**

The sensor touch status is the primary touch sensor information utilized by a user application. The sensor state can either be ON or OFF. For sliders and wheel, additionally the touch position is of interest. For an 8-bit resolution, the touch position ranges from 0 to 255 end-to-end. It is possible to configure with a lower resolution by configuring setting in the touch library. The sensor touch status and slider/wheel position must always be used once the library completes the measurements.

The touch sensor state for mutual capacitance or self-capacitance sensor can be obtained by reading the following boolean variables.

```c
bool sensor_state_self = GET_SELFCAP_SENSOR_STATE(SENSOR_NUMBER);
bool sensor_state_mutl = GET_MUTLCAP_SENSOR_STATE(SENSOR_NUMBER);
```

The touch sensor rotor or slider position information for mutual capacitance or self-capacitance sensor can be obtained using the following parameters.

```c
uint8_t rotor_slider_position_self = GET_SELFCAP_ROTOR_SLIDER_POSITION(ROTOR_SLIDER_NUMBER);
uint8_t rotor_slider_position_mutl = GET_MUTLCAP_ROTOR_SLIDER_POSITION(ROTOR_SLIDER_NUMBER);
```

The touch sensor noise status for mutual capacitance or self-capacitance sensor can be obtained using the following parameters.

```c
bool sensor_noise_state_self = GET_SELFCAP_SENSOR_NOISE_STATUS(SENSOR_NUMBER);
bool sensor_noise_state_mutl = GET_MUTLCAP_SENSOR_NOISE_STATUS(SENSOR_NUMBER);
```
5. **QTouch Library**

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

QTouch 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 5-1. QTouch Library**

5.1. **Overview**

QTouch 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 Library in turn interfaces with the PTC module to perform the necessary action. The PTC module interfaces with the external capacitive touch sensors and is capable of performing self and mutual capacitance method measurements. The library features low power and lumped mode configuration.

**Figure 5-2. QTouch Library Overview**

The QTouch Library API is arranged such that the user application can use standalone self-capacitance or mutual capacitance method or both methods, simultaneously. The following table captures the APIs available for each method. For normal operation, it is sufficient to use the set of Regular APIs for each method. The Helper APIs provides additional flexibility to the user application.
### 5.2. Library Parameters

The QTouch Library configuration parameters are listed in the following table:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Mutual capacitance</th>
<th>Self-capacitance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Configuration</td>
<td>DEF_MUTLCAP_NODES</td>
<td>DEF_SELFCAP_LINES</td>
</tr>
<tr>
<td>Sensor Configuration</td>
<td>DEF_MUTLCAP_NUM_CHANNELS DEF_MUTLCAP_NUM_SENSORS DEF_MUTLCAP_NUM_ROTORS_SLIDERS DEF_MUTLCAP_PTC_GPIO_STATE DEF_MUTLCAP_QUICK_REBURST_ENABLE</td>
<td>DEF_SELFCAP_NUM_CHANNELS DEF_SELFCAP_NUM_SENSORS DEF_SELFCAP_NUM_ROTORS_SLIDERS DEF_SELFCAP_PTC_GPIO_STATE DEF_SELFCAP_QUICK_REBURST_ENABLE</td>
</tr>
<tr>
<td>Sensor Individual Parameters</td>
<td>Detect Threshold Detect Hysteresis Position Resolution Position Hysteresis AKS group</td>
<td>Detect Threshold Detect Hysteresis Position Resolution AKS group</td>
</tr>
</tbody>
</table>
### Configuration

<table>
<thead>
<tr>
<th>Sensor Acquisition Parameters</th>
<th>Mutual capacitance</th>
<th>Self-capacitance</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_MUTLCAP_FILTER_LEVEL_PER_NODE</td>
<td>DEF_SELFCAP_FILTER_LEVEL_PER_NODE</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_AUTO_OS_PER_NODE</td>
<td>DEF_SELFCAP_AUTO_OS_PER_NODE</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_GAIN_PER_NODE DEF_MUTLCAP_FREQ_MODE</td>
<td>DEF_SELFCAP_GAIN_PER_NODE DEF_SELFCAP_FREQ_MODE</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_HOP_FREQS</td>
<td>DEF_SELFCAP_HOP_FREQS</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_CLK_PRESCALE_PER_NODE</td>
<td>DEF_SELFCAP_CLK_PRESCALE_PER_NODE</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_SENSE_RESISTOR_PER_NODE</td>
<td>DEF_SELFCAP_SENSE_RESISTOR_PER_NODE</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor Calibration Auto Tune Setting</th>
<th>AUTO_TUNE_FRSC, AUTO_TUNE_RSEL, AUTO_TUNE_NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_MUTLCAP_FILTER_LEVEL_PER_NODE</td>
<td>DEF_SELFCAP_FILTER_LEVEL_PER_NODE</td>
</tr>
<tr>
<td>DEF_MUTLCAP_AUTO_OS_PER_NODE</td>
<td>DEF_SELFCAP_AUTO_OS_PER_NODE</td>
</tr>
<tr>
<td>DEF_MUTLCAP_GAIN_PER_NODE DEF_MUTLCAP_FREQ_MODE</td>
<td>DEF_SELFCAP_GAIN_PER_NODE DEF_SELFCAP_FREQ_MODE</td>
</tr>
<tr>
<td>DEF_MUTLCAP_HOP_FREQS</td>
<td>DEF_SELFCAP_HOP_FREQS</td>
</tr>
<tr>
<td>DEF_MUTLCAP_CLK_PRESCALE_PER_NODE</td>
<td>DEF_SELFCAP_CLK_PRESCALE_PER_NODE</td>
</tr>
<tr>
<td>DEF_MUTLCAP_SENSE_RESISTOR_PER_NODE</td>
<td>DEF_SELFCAP_SENSE_RESISTOR_PER_NODE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor Noise measurement and Lockout Parameters</th>
<th>DEF_MUTLCAP_NOISE_MEAS_ENABLE</th>
<th>DEF_SELFCAP_NOISE_MEAS_ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_MUTLCAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT</td>
<td>DEF_SELFCAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_NOISE_LIMIT</td>
<td>DEF_SELFCAP_NOISE_LIMIT</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_NOISE_BUFFER_CNT</td>
<td>DEF_SELFCAP_NOISE_BUFFER_CNT</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_LOCKOUT_SEL</td>
<td>DEF_SELFCAP_LOCKOUT_SEL</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_LOCKOUT_CNTLDOWN</td>
<td>DEF_SELFCAP_LOCKOUT_CNTLDOWN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sensor Acquisition Frequency Auto-tuning Parameters</th>
<th>DEF_MUTLCAP_FREQ_AUTO_TUNE_ENABLE</th>
<th>DEF_SELFCAP_FREQ_AUTO_TUNE_ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_MUTLCAP_FREQ_AUTO_TUNE_SIGNAL_STABILITY_LIMIT</td>
<td>DEF_SELFCAP_FREQ_AUTO_TUNE_SIGNAL_STABILITY_LIMIT</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_FREQ_AUTO_TUNE_IN_CNT</td>
<td>DEF_SELFCAP_FREQ_AUTO_TUNE_IN_CNT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Common Parameters</th>
<th>DEF_TOUCH_MEASUREMENT_PERIOD_MS, DEF_TOUCH_PTC_ISR_LVL</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Low Power Parameters</th>
<th>DEF_LOWPOWER_SENSOR_EVENT_PERIODICITY, DEF_LOWPOWER_SENSOR_DRIFT_PERIODICITY_MS, DEF_LOWPOWER_SENSOR_ID</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Moisture Parameters</th>
<th>DEF_MUTLCAP_MOIS_TOLERANCE_ENABLE</th>
<th>DEF_SELFCAP_MOIS_TOLERANCE_ENABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_MUTLCAP_NUM_MOIS_GROUPS</td>
<td>DEF_SELFCAP_NUM_MOIS_GROUPS</td>
<td></td>
</tr>
<tr>
<td>DEF_MUTLCAP_MOIS_QUICK_REBURST_ENABLE</td>
<td>DEF_SELFCAP_MOIS_QUICK_REBURST_ENABLE</td>
<td></td>
</tr>
</tbody>
</table>

5.2.1. Pin, Channel, and Sensor Parameters

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 in the DEF_MUTLCAP_NODES configuration parameter. A mutual capacitance touch button sensor is formed using a single X-Y channel, while a touch rotor or slider sensor is formed using three to eight X-Y channels.

**Mutual Capacitance Channel (X-Y Sense Node)**

- SAM D20J and SAM D21J (64 pins): up to 256 touch channels, 16 X and 16 Y-lines
- SAM D20G and SAM D21G (48 pins): up to 120 touch channels, 12 X and 10 Y-lines
- SAM D20E and SAM D21E (32 pins): up to 60 touch channels, 10 X and 6 Y-lines
- SAM R21E (32 pins): up to 12 touch channels, 6 X and 2 Y-lines
- SAM R21G (48 pins) up to 48 touch channels, 8 X and 6 Y-lines
- SAM DA1J (64 pins): up to 256 touch channels, 16 X and 16 Y-lines
- SAM DA1G (48 pins): up to 120 touch channels, 12 X and 10 Y-lines
- SAM DA1E (32 pins): up to 60 touch channels, 10 X and 6 Y-lines
- SAM D21G17AU and SAM D21G18AU (45 pins): up to 132 touch channels, 12 X and 11 Y-lines
- SAM D21E15BU and SAM D21E16BU (35 pins): up to 60 touch channels, 10 X and 6 Y-lines

The following devices have X and Y multiplexing option.

- SAM D10C14A and SAM D11C14A (14 pins): up to 12 touch channels, 4 X and 3 Y-lines
- SAM D10D14 AS/AU and SAM D11D14 AS/AU (20 pins): up to 42 touch channels, 7 X and 6 Y-lines
- SAM D10D14AM and SAM D11D14AM (24 pins): up to 72 touch channels, 9 X and 8 Y-lines
- SAM L21E (32 pins): up to 42 touch channels, 7 X and 6 Y-lines
- SAM L21G (48 pins): up to 81 touch channels, 9 X and 9 Y-lines
- SAM L21J (64 pins): up to 169 touch channels, 13 X and 13 Y-lines
- SAM L22G (48 pins): up to 132 touch channels, 11 X and 12 Y-lines
- SAM L22J (64 pins): up to 182 touch channels, 13 X and 14 Y-lines
- SAM L22N (100 pins): up to 256 touch channels, 16 X and 16 Y-lines
- SAM C21E and SAM C20E (32 pins): up to 60 touch channels, 10 X and 6 Y-lines
- SAM C21G and SAM C20G (48 pins): up to 120 touch channels, 12 X and 10-Y lines
- SAM C21J and SAM C20J (64 pins): up to 256 touch channels, 16 X and 16 Y-lines
- ATmega328PB (32 pins): up to 144 touch channels, 12 X and 12 Y-lines
- ATmega324PB (44 pins): up to 256 touch channels, 16 X and 16 Y-lines

A few pins can be used either as X-line or Y-line. The datasheets of individual devices provide more information about this multiplexing option.

**Figure 5-3. Mutual Capacitance Sensor Arrangement**

**Figure 5-4. Mutual Capacitance - Channel to Sensor Mapping**

X-Y node pair can be specified using the configuration parameter `DEF_MUTLCAP_NODES` in a non-sequential order. The channel numbering is done in the same order as the X-Y node pair specified in the configuration parameter `DEF_MUTLCAP_NODES`.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual Cap Touch Channel Nodes</td>
<td>DEF_MUTLCAP_NODES</td>
<td>uint16_t</td>
<td>None</td>
<td>1 X-Y node</td>
<td>256 X-Y nodepair</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>array</td>
<td></td>
<td>pair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutual Cap Number of Channels</td>
<td>DEF_MUTLCAP_NUM_CHANNELS</td>
<td>uint16_t</td>
<td>None</td>
<td>1</td>
<td>256 X-Y nodepair</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutual Cap Number of Sensors</td>
<td>DEF_MUTLCAP_NUM_SENSORS</td>
<td>uint16_t</td>
<td>None</td>
<td>1</td>
<td>256 X-Y nodepair</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutual Cap Number of Rotors and Sliders</td>
<td>DEF_MUTLCAP_NUM_ROTORS_SLIDERS</td>
<td>uint8_t</td>
<td>None</td>
<td>0</td>
<td>85 nodepair</td>
<td></td>
</tr>
</tbody>
</table>

Self-capacitance method uses a single sense electrode, 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 - line channel, while a touch rotor or slider sensor can be formed using three Y-line channels.

**Self-capacitance Channel (Y-sense line)**

- SAM D20J and SAM D21J (64 pins): up to 16 channels
- SAM D20G and SAM D21G (48 pins): up to 10 channels
- SAM D20E and SAM D21E (32 pins): up to 6 channels
- SAM D10C14A and SAMD 11C14A (14 pins): up to 7 touch channels
- SAM D10D14 AS/AU and SAMD 11D14 AS/AU (20 pins): up to 13 touch channels
- SAM D10D14AM and SAMD 11D14AM (24 pins): up to 16 touch channels
- SAM L21E (32 pins): up to 7 touch channels
- SAM L21G (48 pins): up to 10 touch channels
- SAM L21J (64 pins): up to 16 touch channels
- SAMR21E (32 pins): up to 2 touch channels
- SAMR21G (48 pins): up to 6 touch channels
- SAM DA1J (64 pins): up to 16 channels
- SAM DA1G (48 pins): up to 10 channels
- SAM DA1E (32 pins): up to 6 channels
- SAM C21E and SAM C20E (32 pins): up to 16 touch channels
- SAM C21G and SAM C20G (48 pins): up to 22 touch channels
- SAM C21J and SAM C20J (64 pins): up to 32 touch channels
- SAM L22G (48 pins): up to 15 touch channels
- SAM L22J (64 pins): up to 19 touch channels
- SAM L22N (100 pins): up to 24 touch channels
- ATmega328PB (32 pins): up to 24 touch channels
- ATmega324PB (44 pins): up to 32 touch channels

Figure 5-5. Self-capacitance Sensor Arrangement

Figure 5-6. Self-capacitance Channel to Sensor Mapping

Y sense line can be specified using the configuration parameter `DEF_SELFCAP_LINES` in non-sequential order. The channel numbering is done in the same order as the Y sense line specified in the configuration parameter `DEF_SELFCAP_LINES`.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self Cap touch channel nodes</td>
<td><code>DEF_SELFCAP_NODES</code></td>
<td><code>uint16_t</code> array</td>
<td>None</td>
<td>1 Y-line</td>
<td>32 Y-line</td>
<td>-</td>
</tr>
<tr>
<td>Self Cap number of channels</td>
<td><code>DEF_SELFCAP_NUM_CHANNELS</code></td>
<td><code>uint16_t</code></td>
<td>None</td>
<td>1 Y-line</td>
<td>32 Y-line</td>
<td>-</td>
</tr>
<tr>
<td>Self Cap number of Sensors</td>
<td><code>DEF_SELFCAP_NUM_SENSORS</code></td>
<td><code>uint16_t</code></td>
<td>None</td>
<td>1 Y-line</td>
<td>32 Y-line</td>
<td>-</td>
</tr>
<tr>
<td>Self Cap number of Rotors and Sliders</td>
<td><code>DEF_SELFCAP_NUM_ROTORS_SLIDERS</code></td>
<td><code>uint8_t</code></td>
<td>None</td>
<td>0 Y-line</td>
<td>10 Y-line</td>
<td>-</td>
</tr>
</tbody>
</table>
The touch sensors must be enabled in the sequential order of the channels specified using the `touch_xx_sensor_config()` API. For improved EMC performance, a series resistor with value of 1 Kilo-ohm must 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.

### 5.2.2. Sensor Individual Parameters

This section explains the settings that are specific to the individual sensor.

#### Detect Threshold

A sensor’s detect threshold defines how much its signal must increase above its reference level to qualify as a potential touch detect. However, the final detection confirmation must satisfy the Detect Integrator (DI) limit. Larger threshold values desensitize sensors since the signal must change more (i.e. requires larger touch) to exceed the threshold level. Conversely, lower threshold levels make sensors more sensitive.

Threshold setting depends on the amount of signal swing when a sensor is touched. Usually, thicker front panels or smaller electrodes have smaller signal swing on touch, thus require lower threshold levels. Typically, detect threshold is set to 50% of touch delta. Desired touch delta for a buttons is ~30 to 80 counts and for wheels or sliders is ~50 to 120 counts.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threshold</td>
<td>detect_threshold</td>
<td>threshold_t</td>
<td>Counts</td>
<td>3</td>
<td>255</td>
<td>20-50(For buttons)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>30-80(For sliders and wheels)</td>
</tr>
</tbody>
</table>

#### Detect Hysteresis

This setting is sensor detection hysteresis value. It is expressed as a percentage of the sensor detection threshold setting. Once a sensor goes into detect its threshold level is reduced (by the hysteresis value) in order to avoid the sensor dither in and out of detect if the signal level is close to original threshold level.

- Setting of 0 = 50% of detect threshold value (**HYST_50**)
- Setting of 1 = 25% of detect threshold value (**HYST_25**)
- Setting of 2 = 12.5% of detect threshold value (**HYST_12_5**)
- Setting of 3 = 6.25% of detect threshold value (**HYST_6_25**)

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hysteresis</td>
<td>detect_threshold</td>
<td>uint8_t</td>
<td>(2bits)</td>
<td>HYST_6_25</td>
<td>HYST_50</td>
<td>HYST_6_25</td>
</tr>
</tbody>
</table>

#### Position Resolution

The rotor or slider needs the position resolution (angle resolution in case of rotor and linear resolution in case of slider) to be set. Resolution is the number of bits needed to report the position of rotor or slider. It can have values from 2 bits to 8 bits.
Position Hysteresis

In case of Mutual Cap, the rotor or slider needs the position hysteresis (angle hysteresis in case of rotor and linear hysteresis in case of slider) to be set. It is the number of positions the user has to move back, before touch position is reported when the direction of scrolling is changed and during the first scrolling after user press.

Hysteresis can range from 0 (1 position) to 7 (8 positions). The hysteresis is carried out at 8 bits resolution internally and scaled to desired resolution; therefore at resolutions lower than 8 bits there might be a difference of 1 reported position from the hysteresis setting, depending on where the touch is detected. At lower resolutions, where skipping of the reported positions is observed, hysteresis can be set to 0 (1 position). At Higher resolutions (6 to 8bits), it would be recommended to have a hysteresis of at least 2 positions or more.

**Note:** It is not valid to have a hysteresis value more than the available bit positions in the resolution. For instance, a hysteresis value of 5 positions with a resolution of 2 bits (4 positions) is invalid. Position hysteresis is invalid (unused) in case of self-capacitance method sensors.

### Adjacent Key Suppression (AKS®)

In designs where the sensors are close together or configured for high sensitivity, multiple sensors might report a detect simultaneously. To allow applications to determine the intended single touch, the touch library provides the user the ability to configure a certain number of sensors in an AKS group.

When a group of sensors are in the same AKS group, only the first strongest sensor will report detection. The sensor reporting detection will continue to report detection even if another sensor’s delta becomes stronger. The sensor stays indetect until its delta falls lower than its detection threshold. If any more sensors in the AKS group are still in detect only the strongest will report detection. At a given time point, only one sensor from each AKS group is reported to be indetect.

AKS feature can be enabled or disabled using a macro `DEF_XXXXCAP_AKS_ENABLE`

- 1u = AKS grouping functionality is enabled
- 0u = AKS grouping functionality is disabled

The library provides the ability to configure a sensor to belong to one of the Adjacent Key Suppression Groups (AKS Group).
5.2.3. Sensor Global Parameters

This section explains the settings that are common all sensors. For instance, if recalibration threshold (one of the global settings) of mutual cap sensors is set as RECAL_100, all mutual capacitance sensors will be configured for a recalibration threshold of 100%. These sensor global parameter settings can be independently set to self-capacitance and mutual capacitance sensors.

Detect Integration

The QTouch Library features a detect integration mechanism, which confirm detection in a robust environment. The detect integrator (DI) acts as a simple signal filter to suppress false detections caused by spurious events such as electrical noise.

A counter is incremented each time the sensor delta has exceeded its threshold and stayed there for a specific number of acquisitions, without going below the threshold levels. When this counter reaches a preset limit (the DI value) the sensor is finally declared to be touched. If on any acquisition the delta is below the threshold level, the counter is cleared and the process has to start from the beginning. The DI process is applicable to a ‘release’ (going out of detect) event as well.

For example, if the DI value is 10, the device has to exceed its threshold and stay there for \((10 + 2)\) successive acquisitions without going below the threshold level, before the sensor is declared to be touched.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>DEF_MUTLCAP_DI, DEF_SELFCAP_DI</td>
<td>uint8_t</td>
<td>Cycles</td>
<td>0</td>
<td>255</td>
<td>4</td>
</tr>
</tbody>
</table>

Max-ON Duration

If an object unintentionally contacts a sensor resulting in a touch detection for a prolonged interval it is usually desirable to recalibrate the sensor in order to restore its function, after a time delay of a few seconds.

The Maximum ON duration timer monitors such detections; if detection exceeds the timer's settings, the sensor is automatically recalibrated. After a recalibration has taken place, the affected sensor once again functions normally even if it still in contact with the foreign object.

Max-ON duration can be disabled by setting it to zero (infinite timeout) in which case the channel never recalibrates during a continuous detection (but the host could still command it).

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum ON Duration</td>
<td>DEF_MUTLCAP_MAX_ON_DURATION, DEF_SELFCAP_MAX_ON_DURATION</td>
<td>uint8_t</td>
<td>200ms</td>
<td>0</td>
<td>255</td>
<td>30(6s)</td>
</tr>
</tbody>
</table>

Away from Touch and Towards Touch Drift Rate

Drift in a general sense means adjusting reference level (of a sensor) to allow compensation for temperature (or other factor) effect on physical sensor characteristics. Decreasing reference level for such compensation is called Negative drift & increasing reference level is called Positive drift. Specifically, the drift compensation should be set to compensate faster for increasing signals than for decreasing signals.

Signals can drift because of changes in physical sensor characteristics over time and temperature. It is crucial that such drift be compensated for; otherwise false detections and sensitivity shifts can occur.
Drift compensation occurs only while there is no detection in effect. Once a finger is sensed, the drift compensation mechanism ceases since the signal is legitimately detecting an object. Drift compensation works only when the signal in question has not crossed the 'Detect threshold' level.

The drift compensation mechanism can be asymmetric. It can be made to occur in one direction faster than it does in the other simply by changing the appropriate setup parameters.

Signal values of a sensor tend to increase when an object (touch) is approaching it or a characteristic change of sensor over time and temperature. Increasing signals should not be compensated quickly, as an approaching finger could be compensated for partially or entirely before even touching the channel (towards touch drift).

However, an object over the channel which does not cause detection, and for which the sensor has already made full allowance (over some period of time), could suddenly be removed leaving the sensor with an artificially suppressed reference level and thus become insensitive to touch. In the latter case, the sensor should compensate for the object's removal by raising the reference level relatively quickly (away from touch drift).

### Setting Configuration Name Data Type Unit Min Max Typical

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Towards touch</td>
<td>DEF_MUTLCAP_TCH_DRIFT_RATE, DEF_SELFCAP_TCH_DRIFT_RATE</td>
<td>uint8_t</td>
<td>200ms</td>
<td>0</td>
<td>127</td>
<td>20(4s)</td>
</tr>
<tr>
<td>Drift</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Away from touch</td>
<td>DEF_MUTLCAP_ATCH_DRIFT_RATE, DEF_SELFCAP_ATCH_DRIFT_RATE</td>
<td>uint8_t</td>
<td>200ms</td>
<td>0</td>
<td>127</td>
<td>5(1s)</td>
</tr>
<tr>
<td>Drift</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Drift Hold Time**

Drift Hold Time (DHT) is used to restrict drift on all sensors while one or more sensors are activated. It defines the length of time the drift is halted after a key detection. This feature is useful in cases of high density keypads where touching a key or floating a finger over the keypad would cause untouched keys to drift, and therefore create a sensitivity shift, and ultimately inhibit any touch detection.

### Setting Configuration Name Data Type Unit Min Max Typical

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drift Hold Time</td>
<td>DEF_MUTLCAP_DRIFT_HOLD_TIME, DEF_SELFCAP_DRIFT_HOLD_TIME</td>
<td>uint8_t</td>
<td>200ms</td>
<td>0</td>
<td>255</td>
<td>20(4s)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Away From Touch Recalibration Threshold**

Recalibration threshold is the level beyond which automatic recalibration occurs. Recalibration threshold is expressed as a percentage of the detection threshold setting.

This setting is an enumerated value and its settings are as follows:

- Setting of 0 = 100% of detect threshold (RECAL_100)
- Setting of 1 = 50% of detect threshold (RECAL_50)
- Setting of 2 = 25% of detect threshold (RECAL_25)
- Setting of 3 = 12.5% of detect threshold (RECAL_12_5)
- Setting of 4 = 6.25% of detect threshold (RECAL_6_25)
However, an absolute value of 4 is the hard limit for this setting. For example, if the detection threshold is 40 and the Recalibration threshold value is set to 4.

Although this implies an absolute value of 2 \( (40 \times 6.25\% = 2.5) \), it is hard limited to 4.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recalibration threshold</td>
<td>DEF_MUTLCAP_ATCH_RECAL_THRESHOLD,</td>
<td>uint8_t</td>
<td>Enum</td>
<td>RECAL_6_25</td>
<td>Detect</td>
<td>RECAL_100</td>
</tr>
<tr>
<td></td>
<td>DEF_SELFCAP_ATCH_RECAL_THRESHOLD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Away From Touch Recalibration Delay**

If any key is found to have a significant negative delta, it is deemed to be an error condition. If this condition persists for more than the away from touch recalibration delay, i.e., \( qt\_pos\_recal\_delay \) period, then an automatic recalibration is carried out.

A counter is incremented each time the sensor delta is equal to the away from touch recalibration threshold and stayed there for a specific number of acquisitions. When this counter reaches a preset limit (the PRD value) the sensor is finally recalibrated. If on any acquisition the delta is seen to be greater than the away from touch recalibration threshold level, the counter is cleared and the away from touch drifting is performed.

For example, if the away from touch recalibration delay setting is 10, then the delta has to drop below the recalibration threshold and stay there for 10 acquisitions in succession without going below the threshold level, before the sensor is declared to be recalibrated. Away from touch recalibration can be disabled with a setting of 0.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Away from touch</td>
<td>DEF_MUTLCAP_ATCH_RECAL_DELAY,</td>
<td>uint8_t</td>
<td>Cycles</td>
<td>0</td>
<td>255</td>
<td>10</td>
</tr>
<tr>
<td>Recalibration Delay</td>
<td>DEF_SELFCAP_ATCH_RECAL_DELAY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sensor Post-Processing Mode**

When **TOUCH_LIBRARY_DRIVEN** mode is selected, the library self-initiates repeated touch measurements to resolve touch press, release and calibration. This mode is suited for best response time.

When **TOUCH_APPLN_DRIVEN** mode is selected, the library does not initiate repeated touch measurement to resolve touch press, release and calibration. This mode suits deterministic PTC interrupt execution time for applications requiring stringent CPU time requirements. As repeated touch measurements are delayed due to other critical application code being executed. This mode can potentially affect the touch response time.

In order to improve the touch response time with the **TOUCH_APPLN_DRIVEN** mode, the **touch_xxxxcap_sensors_measure** API call should be modified as below to initiate touch measurements periodically or when the burst again acquisition status flag has been set.

```c
if (((touch_time.time_to_measure_touch == 1u) || (p_mutlcap_measure_data->acq_status & TOUCH_BURST_AGAIN))
{
    /* Start a touch sensors measurement process. */
    touch_ret = touch_mutlcap_sensors_measure(touch_time.current_time_ms, NORMAL_ACQ_MODE, touch_mutlcap_measure
```
### Charge Share Delay

Charge share delay indicates the number of additional charge cycles that are inserted within a capacitance measurement cycle to ensure that the touch sensor is fully charged. The CSD value is dependent on the sensor capacitance along with the series resistor on the Y line.

**Note:** Any increase in the charge share delay also increases the measurement time for a specific configuration.

When manual tuning is performed, the CSD value for the sensor with largest combination of capacitance along with series resistance should be considered.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CSD (Charge Share Delay)</strong></td>
<td>DEF_MUTL_CAP_CSD_VALUE, DEF_SELF_CAP_CSD_VALUE</td>
<td>uint8_t</td>
<td>PTC cycles</td>
<td>0</td>
<td>250</td>
<td>0</td>
</tr>
</tbody>
</table>

### How to tune the CSD setting manually?

1. Initially, use an arbitrarily large value such as 64 and note the signal value. A large value ensures that the charge time is enough for full charge transfer.
2. Reduce the CSD and verify the signal value drop, until signal is approximately 97-98% of the value used initially. This ensures a good charge transfer without any major loss in the signal.
3. Continue the same procedure [Step 1 and 2] for all the sensors available in the system. Use the largest value of the CSD used in the system for the global setting.

**Note:** For the same CSD setting, Mutual capacitance has a lower burst time than self-capacitance. A unit increase in mutual capacitance CSD consumes around 12 PTC cycles. Whereas for the self-capacitance an increase in CSD consumes approximately twice the mutual capacitance CSD time with the same setting.

### Auto-OS Signal Stability Limit

The parameter `DEF_XXXXCAP_AUTO_OS_SIGNAL_STABILITY_LIMIT` defines the stability limit of the signals for performing over-samples. Stability limit is the variance in sensor signal value under noisy environment. A high level of stability limit is set to auto trigger oversamples on large noise presence. It is recommended to keep this setting close to the lowest sensor detect threshold of the system and tune it further based on the noise.

Range: 1 to 1000

### 5.2.4. Sensor Acquisition Parameters

#### Filter Level

The filter level setting controls the number of samples taken 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 resulting in increased power consumption and response time. This setting is available on per channel basis, allowing easy tuning.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter level</td>
<td>DEF_MUTLCAP_FILTER_LEVEL_PER_NODE, DEF_SELFCAP_FILTER_LEVEL_PER_NODE</td>
<td>filter_level_t</td>
<td>Number of samples</td>
<td>1</td>
<td>64</td>
<td>16</td>
</tr>
</tbody>
</table>

**Auto Oversamples**

Auto oversample controls the automatic oversampling of sensor channels when unstable signals are detected with the default setting of 'Filter level'. Enabling Auto oversample results in 'Filter level' x 'Auto Oversample' number of samples taken 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 taken with stable signal values and 16 oversamples are taken when unstable signal is detected. This setting is available on per channel basis, allowing easy tuning.

A higher filter level 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.

**Figure 5-7. Auto oversamples**

![Diagram showing auto oversamples](image)

Auto oversamples can be disabled to obtain best power consumption.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto Oversamples</td>
<td>DEF_MUTLCAP_AUTO_OS_PER_NODE, DEF_SELFCAP_AUTO_OS_PER_NODE</td>
<td>auto_os_t</td>
<td>Sample multiplier</td>
<td>2</td>
<td>128</td>
<td>AUTO_OS_NONE</td>
</tr>
</tbody>
</table>

**Gain Setting**

Gain setting is applied on a per-channel basis to allow a scaling-up of the touch delta upon contact. Gain setting depends on the sensor design and touch panel thickness.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gain</td>
<td>DEF_MUTLCAP_GAIN_PER_NODE, DEF_SELFCAP_GAIN_PER_NODE</td>
<td>gain_t</td>
<td>Gain multiplier</td>
<td>1</td>
<td>32</td>
<td>1 (For self-capacitance) 4 (For mutual capacitance)</td>
</tr>
</tbody>
</table>
The figure shows the expected signal value for a given combination of gain setting and filter level setting. The values provided are only indicative and the actual sensor signal values might be close to the suggested levels.

**Figure 5-8. Average Settling Signal Value for FILTER LEVEL and GAIN Combination**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prescalar</td>
<td>DEF_MUTLCAP_CLK_PRESCALE_PER_NODE, DEF_SELFCAP_CLK_PRESCALE_PER_NODE</td>
<td>prsc_div_sel_t</td>
<td>PRSC_DIV_SEL_1, PRSC_DIV_SEL_2, PRSC_DIV_SEL_4, PRSC_DIV_SEL_8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Prescalar Setting**

The prescaler parameter denotes the clock divider for the particular channel. It can be set on per channel basis and is independent to each sensor node/channel. This parameter is auto tuned based on the auto tune settings. Tuning this parameter allows for improved noise performance.

**Series Resistor Setting**

The series resistor denotes the resistor used on the particular channel for the acquisition. The value is tunable and allows both auto and manual tuning options. Tuning this parameter allows for improved noise performance.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Series Resistor</td>
<td>DEF_MUTLCAP_SENSE_RESI_STOR_PER_NODE, DEF_SELFCAP_SENSE_RESI_STOR_PER_NODE</td>
<td>rsel_val_t</td>
<td>RSEL_VAL_0, RSEL_VAL_20, RSEL_VAL_50, RSEL_VAL_100</td>
<td>RSEL_VAL_0</td>
<td>RSEL_VAL_100</td>
<td>RSEL_VAL_100</td>
</tr>
</tbody>
</table>

**Boot Prescalar Setting**

The boot prescaler parameter denotes the clock divider for the particular channel. It can be set on per channel basis and is independent to each sensor node/channel. This setting is used for calibrating the sensors after a power-on. This parameter must be configured as the auto tune is not available.
### Boot Prescaler

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Prescaler</td>
<td>DEF_MUTLCAP_CC_CAL_CLK_PRESCALE_PER_NODE, DEF_SELF_CAP_CC_CAL_CLK_PRESCALE_PER_NODE</td>
<td>prscDivSel_t</td>
<td>PRSC_DIV_SEL_1, PRSC_DIV_SEL_2, PRSC_DIV_SEL_4, PRSC_DIV_SEL_8</td>
<td>PRSC_DIV_SEL_1</td>
<td>PRSC_DIV_SEL_8</td>
<td>PRSC_DIV_SEL_1</td>
</tr>
</tbody>
</table>

### Boot Series Resistor Setting

The boot series resistor denotes the resistor used on the particular channel on device power-on calibration. This parameter must be configured as the auto tune is not available.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Series Resistor</td>
<td>DEF_MUTLCAP_CC_CAL_SENSE_RESISTOR_PER_NODE, DEF_SELF_CAP_CC_CAL_SENSE_RESISTOR_PER_NODE</td>
<td>rselVal_t</td>
<td>RSEL_VAL_0, RSEL_VAL_20, RSEL_VAL_50, RSEL_VAL_100</td>
<td>RSEL_VAL_0</td>
<td>RSEL_VAL_100</td>
<td>RSEL_VAL_100</td>
</tr>
</tbody>
</table>

### Frequency Mode

Frequency mode setting allows users to tune the PTC touch acquisition frequency characteristics to counter environment noise.

**FREQ_MODE_HOP**

When frequency mode hopping option is selected, the PTC runs a frequency hopping cycle with subsequent measurements done using the three PTC acquisition frequency delay settings as specified in DEF_SELF_CAP_HOP_FREQS. In this case, an additional software median filter is applied to the measured signal values.

**FREQ_MODE_SPREAD**

When frequency mode spread spectrum option is selected, the PTC runs with spread spectrum enabled for jittered delay based acquisition.

**FREQ_MODE_SPREAD_MEDIAN**

When frequency mode spread spectrum median option is selected, the PTC runs with spread spectrum enabled. In this case, an additional software median filter is applied to the measured signal values.

**FREQ_MODE_NONE**

When frequency mode none option is selected, the PTC runs at constant speed. This mode is suited for best power consumption.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Options</th>
<th>Min</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency mode</td>
<td>DEF_MUTLCAP_FREQ_MODE, DEF_SELF_CAP_FREQ_MODE</td>
<td>freqModeSel_t</td>
<td>FREQ_MODE_NONE, FREQ_MODE_HOP, FREQ_MODE_SPREAD, FREQ_MODE_SPREAD_MEDIAN</td>
<td>FREQ_MODE_NONE</td>
<td>FREQ_MODE_SPREAD_MEDIAN</td>
<td>FREQ_MODE_NONE</td>
</tr>
</tbody>
</table>

### Frequency Hop Delay

The frequency hop delay setting is used when the Frequency mode is set to FREQ_MODE_HOP. A set of three frequency hop delay settings should be specified. 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 taken for capacitance measurement on a given sensor as compared to a lower delay setting. A desired setting can be used to avoid noise around the same frequency as the acquisition frequency.
5.2.5. **Sensor Calibration Auto Tune Setting**

Auto tune parameter setting is passed to the `touch_xx_sensors_calibrate` API in order to allow users to tune the PTC module for power consumption or noise performance.

**AUTO_TUNE_PRSC**

When Auto tuning of pre-scaler is selected, the PTC uses the user defined internal series resistor setting (DEF_XXXXCAP_SENSE_RESISTOR_PER_NODE) and the pre-scaler is adjusted to slow down the PTC operation to ensure full charge transfer. Auto tuning of pre-scaler with RSEL_VAL_100 as the series resistor results in least power consumption while resulting in increased power consumption and touch response time.

**AUTO_TUNE_RSEL**

When Auto tuning of the series resistor is selected, the PTC runs at user defined pre-scaler setting speed (DEF_XXXXCAP_CLK_PRESCALE_PER_NODE) and the internal series resistor is tuned automatically to the optimum value to allow for full charge transfer. Auto tuning of series resistor with PRSC_DIV_SEL_1 as the PTC pre-scale results in best case power consumption.

**AUTO_TUNE_NONE**

When manual tuning option is selected, the user defined values of PTC pre-scaler and series resistor is used for PTC operation as given in DEF_XXXXCAP_CLK_PRESCALE_PER_NODE and DEF_XXXXCAP_SENSE_RESISTOR_PER_NODE

5.2.6. **Sensor Noise Measurement and Lockout Parameters**

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:

```c
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_MUTLCAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT limit during buffer window period.

When the measured noise exceeds DEF_MUTLCAP_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_xxxxxcap_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_xxxxxcap_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.

Figure 5-9. Noise Calculation

Noise Measurement Signal Stability Limit
The parameter **DEF_XXXXAP_NOISE_MEAS_SIGNAL_STABILITY_LIMIT** is the variance in sensor signal value under noisy environment. Any noise level over and above the noise signal stability limit contributes to the Noise limit.

It is recommended to keep this setting close to the lowest sensor detect threshold of the system and tune it further based on the noise.

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 Limit**

The **DEF_XXXXCAP_NOISE_LIMIT** specifies the limit to the total noise accumulated over the noise buffer count. If the accumulated noise exceeds the noise limit, then lockout is triggered. There are two purposes for this parameter:

- If the noise level calculated during a running window exceeds **DEF_XXXXCAP_NOISE_LIMIT**, then the corresponding sensor are declared noisy and sensor global noisy bit is set as '1'.

- If 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** as 5u.

**Sensor Lockout Selection**

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 report false touch drift or recalibrate incorrectly.

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_LOCKOUT**.

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

### 5.2.7. Sensor Acquisition Frequency Auto Tuning Parameters

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_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_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.
Frequency Auto Tune Signal Stability

The `DEF_XXXXCAP_FREQ_AUTO_SIGNAL_STABILITY_LIMIT` is the variance in sensor signal value under noisy environment. A signal stability limit level is set to auto tune acquisition frequency on noise presence. It is recommended to keep this setting close to the lowest sensor detect threshold of the system and tune it further based on the noise.

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.
5.2.8. Quick Re-burst Parameter

Quick Reburst

This macro is used to enable or disable quick re-burst feature. When Quick re-burst is enabled, upon user touch and release, only that touched sensor or channel is subsequently measured to resolve detect integration (or debounce). Enabling this feature results in best touch response time.

When Quick re-burst is disabled, upon user touch and release, all sensors or channels are measured to resolve detect integration (or debounce). This feature should only be disabled when developing any special application involving all sensor measurements during user activity.

Within an AKS (Adjacent Key suppression) group, all the sensors within that group are measured during user touch independent of this feature being enabled or disabled.

5.2.9. Common Parameters

Measurement Period

The measurement period setting is used to set the periodic interval for touch sensor measurement. The minimum measurement period setting should be greater than the time taken to complete measurement on all sensors. This can be simply determined by calling the `touch_xx_sensors_measure` API in a while loop and then toggling a GPIO pin in the measurement complete callback.

```c
main()
{
  while(1)
  {
    touch_ret =
    touch_mutlcap_sensors_measure(touch_time.current_time_ms,NORMAL_ACQ_MODE,touch_mutlcap_measure_complete_callback);
  }
}

void touch_mutlcap_measure_complete_callback( void )
{
  if (!(p_mutlcap_measure_data->acq_status & TOUCH_BURST_AGAIN))
  {
    /* Set the Mutual Cap measurement done flag. */
    p_mutlcap_measure_data->measurement_done_touch = 1u;
    port_pin_toggle_output_level(PIN_PB00);
  }
}
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Configuration Name</th>
<th>Data Type</th>
<th>Unit</th>
<th>Values</th>
<th>Max</th>
<th>Typical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor measurement interval</td>
<td>DEF_TOUCH_MEASUREMENT_PERIOD_MS</td>
<td>uint16_t</td>
<td>millisecond</td>
<td>Should be found through GPIO pin toggle procedure.</td>
<td>65535</td>
<td>20</td>
</tr>
</tbody>
</table>

PTC Interrupt Priority Level

The Nested Vectored Interrupt Controller (NVIC) in the SAM 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. Setting the PTC interrupt priority level to lowest can have an impact on the touch response time, depending on the execution time taken by other higher priority interrupts.
To avoid stack overflow, ensure that adequate stack size has been set in the user application. This configuration is applicable only for SAM devices.

**touch_suspend_app_cb**

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.

### Low power Sensor Event Periodicity

When the CPU returns to standby mode from active, the sensor configured as the low power sensor is scanned at this interval. A high value for this parameter will reduce power consumption but increase response time for a low power sensor.

The following macros are used for configuring the low power sensor event periodicity:

- The macro `LOWPOWER_PER0_SCAN_3_P_9_MS` sets the scan rate at 3.9ms
- The macro `LOWPOWER_PER1_SCAN_7_P_8_MS` sets the scan rate at 7.8ms
- The macro `LOWPOWER_PER2_SCAN_15_P_625_MS` sets the scan rate at 15.625ms
- The macro `LOWPOWER_PER3_SCAN_31_P_25_MS` sets the scan rate at 31.25ms
- The macro `LOWPOWER_PER4_SCAN_62_P_5_MS` sets the scan rate at 62.5ms
- The macro `LOWPOWER_PER5_SCAN_125_MS` sets the scan rate at 125ms
- The macro `LOWPOWER_PER6_SCAN_250_MS` sets the scan rate at 250ms
- The macro `LOWPOWER_PER7_SCAN_500_MS` sets the scan rate at 500ms

### Low power Sensor Drift Periodicity

This parameter configures the scan interval for a single active measurement during low power mode. This active measurement is required for reference tracking of low power sensor.
The macro `DEF_LOWPOWER_SENSOR_ID` is used to configure a sensor as low power sensor. Only one sensor can be configured as low power sensor. Low power sensor can be a normal sensor or a lumped sensor.

5.2.10. Moisture Parameters

Moisture Tolerance Enable

The macro `DEF_XXXXCAP_MOIS_TOLERANCE_ENABLE` is used to Enable or disable Moisture detection feature.

Moisture Quick Reburst

The macro `DEF_XXXXCAP_MOIS_QUICK_REBURST_ENABLE` is used to enable or disable quick re-burst feature within a given moisture group. When enabled, if within a given moisture group, when any sensor is touched, repeated measurements are done only that sensor to resolve detect integration or de-bounce. When disabled, if within a given moisture group, when any sensor is touched, repeated measurements are done on all sensors within the moisture group to resolve detect integration or de-bounce. It is recommended to enable this feature for best touch response time.

Moisture groups

The macro `DEF_XXXXCAP_NUM_MOIS_GROUPS` specifies the total number of individual moisture group present the system.

5.2.11. PTC Lines Ground Feature

PTC GPIO State

The macro `DEF_XXXXCAP_PTC_GPIO_STATE` is used to set the unmeasured self/mutual capacitance PTC lines to Ground / Vcc in between PTC measurement cycle. Setting the PTC lines to `GND_WHEN_NOT_MEASURED` will set the state of the pin to low whenever the pin is unmeasured. Setting the PTC lines to `PULLHIGH_WHEN_NOT_MEASURED` will make the PTC lines to float in between sensor measurement in a measurement cycle. It is recommended to set `GND_WHEN_NOT_MEASURED` configuration to get low power.

5.3. 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 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.

Note: Lumped sensor and the sensor which is part of the specific lump should not be assigned to same moisture group.
5.3.1. Moisture Tolerance Group

This feature enables the customer application to group a set of sensors into 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.

5.3.2. 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.

5.4. Reading Sensor States
When noise immunity and moisture tolerance features are enabled the validity of the sensor state is based on the moisture status and noise status. Refer Noise Counter Measures and Moisture Parameters 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.

```c
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 */
}
```

5.5. Application Flow

5.5.1. Application Flow SAM

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 touch status and rotor/slider position. An interrupt callback function is 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.

Before using the PTC, the generic clock generator for the PTC peripheral should be set up by the Application. It is recommended to set the PTC generic clock to 4MHz.
5.5.2. **Application Flow - megaAVR**

The application periodically initiates a touch measurement on either mutual capacitance or self-capacitance sensors either in polled or interrupt mode. In polling mode, touch API's are blocking API's and will consume more CPU time. In ISR mode, touch API's are non blocking and will generates an end of conversion (EOC) interrupt at the end of each sensor measurement. Touch measurement is initiated on first sensor by calling `touch_xxxxCap_sensors_measure()` API. The touch measurement is initiated sequentially and additional post-processing is performed on the measured sensor data to determine touch status and rotor/slider position by calling `touch_ptc_process()` API in application context instead of interrupt context. A callback function is triggered to indicate completion of measurement. The ISR mode sequence of operation facilitates the CPU to either sleep or perform other functions during touch sensor measurement.

It is recommended to set the PTC clock to 4MHz.
5.6. **API Sequence**

The `touch_xx_sensors_init` API initializes the QTouch Library as well as the PTC module. It also initializes the mutual or self-capacitance method specific pin, register, and global sensor configuration.

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

The `touch_xx_sensors_calibrate` API calibrates all the configured sensors and prepares the sensors for normal operation. The `touch_xx_sensors_measure` API initiates a sensor measurement on all the configured sensors.
5.7. **State Machine**

The PTC QTouch Library state machine that presents the various library States and Event transitions can be found in the figure below. The state machine is maintained separately for each of the touch acquisition method, which means the state of mutual capacitance sensor operation can be different from the state of self-capacitance allowing them to co-exist.
The `touch_xx_sensors_init` API initializes the QTouch Library as well as the PTC module. It also initializes the mutual or self-capacitance method specific pin, register, and global sensor configuration.

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

The `touch_xx_sensors_calibrate` API calibrates all the configured sensors and prepares the sensors for normal operation.

The `touch_xx_sensors_measure` API initiates a sensor measurement on all the configured sensors.

The `touch_xx_sensors_deinit` function is used to clear the initialized library state. Used for clearing the internal library data and states. When called will modify the library state to TOUCH_STATE_NULL.

The `touch_xxxx_lowpower_sensor_enable_event_measure` API is used to start a event trigger based low power sensor measurement.

**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.

**Note:** If it is not registered, then the suspend call will return TOUCH_INVALID_INPUT_PARAM.

The application then should disable corresponding clock to reduce the power consumption. The following flowchart depicts the suspend sequence.
Figure 5-16. Suspend Sequence

If the touch state is not TOUCH_STATE_BUSY the user can disable the clock and proceed to complete the suspend routine.

To resume the operations, perform the following sequence:

Figure 5-17. Resume Sequence

The SAM controllers may be configured to operate PTC touch sensing autonomously using the Event System. In this mode, a single sensor channel is designated as the 'Low Power' key and may be periodically measured for touch detection without any CPU action. The CPU may be held in STANDBY throughout the operation, minimizing power consumption.

The low power key may be a discrete electrode with one Y (Sense) line for self-capacitance or one X (Drive) plus one Y (Sense) for mutual capacitance, or it may be a combination of multiple Drive and/or Sense lines as a lumped mode sensor as described.

With this method, a fast response may be achieved even in large key-count applications while operating at an extremely low power level, drawing less than 10uA at 3.3V.
5.8. Operation Modes
The QTouch Library can operate in the following sensor measurement modes.
- Periodic measurement
- Continuous measurement

5.8.1. Periodic Measurement
In the periodic measurement mode, sensor measurement is initiated by the application through a periodic event such as timer interrupt. The periodic measurement mode scenario is when none of the sensors are touched. While a long measurement period can be used to achieve lower device power consumption, a short measurement period is required for better touch response time. Hence, the measurement period should be tuned to suit a given application. Typical measurement period can range between 20 millisecond to 250 millisecond.

Figure 5-18. Periodic Measurement Mode
5.8.2. Continuous Measurement

In the continuous measurement mode, back to back sensor measurement can be initiated from the touch library. This mode can be triggered to resolve user presence or resolve calibration under the following scenario.

- Resolve user presence, when sensor is touched or released
- Resolve calibration, when
  - Sensor is calibrated using the `touch_xx_sensors_calibrate` API
  - Sensor is in Away from touch re-calibration condition
  - Sensor is in Max-on duration condition

The `TOUCH_BURST_AGAIN` acquisition status data bit field in the measure data structure is set to indicate continuous measurement mode.

```c
void touch_mutlcap_measure_complete_callback(void)
{
    if (!(p_mutlcap_measure_data->acq_status & TOUCH_BURST_AGAIN))
    {
        /* Set the Mutual Cap measurement done flag. */
        p_mutlcap_measure_data->measurement_done_touch = 1u;
    }
}
```

**Touch Library Acquisition Status Flags**

The touch library acquisition status information during continuous measurement mode is available using the `touch_acq_status_t acq_status` element of the `touch_measure_data_t` touch measure data structure.

**Table 5-1. Touch Acquisition Status Bit Fields**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Bitfield</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOUCH_NO_ACTIVITY</td>
<td>0x0000u</td>
<td>No Touch activity</td>
</tr>
<tr>
<td>TOUCH_IN_DETECT</td>
<td>0x0001u</td>
<td>Atleast one Touch channel is in detect</td>
</tr>
<tr>
<td>TOUCH_STATUS_CHANGE</td>
<td>0x0002u</td>
<td>Change in Touch status of atleast one Touch channel</td>
</tr>
<tr>
<td>TOUCH_ROTOR_SLIDER_POS_CHANGE</td>
<td>0x0004u</td>
<td>Change in Rotor or Slider position of atleast one rotor or slider</td>
</tr>
<tr>
<td>TOUCH_CHANNEL_REF_CHANGE</td>
<td>0x0008u</td>
<td>Change in Reference value of atleast one Touch channel</td>
</tr>
<tr>
<td>TOUCH_BURST_AGAIN</td>
<td>0x0008u</td>
<td>Indicates that reburst is required to resolve Filtering or Calibration state</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_CAL</td>
<td>0x0200u</td>
<td>Indicates that reburst is needed to resolve Calibration</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_FILTERIN</td>
<td>0x0200u</td>
<td>Indicates that reburst is needed to resolve Filtering</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_DI</td>
<td>0x0800u</td>
<td>Indicates that reburst is needed to resolve Detect Integration</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_POS_RECAL</td>
<td>0x1000u</td>
<td>Indicates that reburst is needed to resolve Recalibration</td>
</tr>
<tr>
<td>Macro</td>
<td>Bitfield</td>
<td>Comment</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>TOUCH_CC_CALIB_ERROR</td>
<td>0x2000u</td>
<td>Indicates that CC calibration failed on at least one channel</td>
</tr>
<tr>
<td>TOUCH_AUTO OS_IN_PROGRESS</td>
<td>0x4000u</td>
<td>Indicates that Auto OS in progress to get stable channel signal</td>
</tr>
</tbody>
</table>

The acquisition status flags can be monitored within the measure complete callback as shown.

```c
void touch_mutlicap_measure_complete_callback(void)
{
    if ((p_mutlicap_measure_data->acq_status & TOUCH_BURST_AGAIN))
    {
        //Denotes acquisition is incomplete.
    }
    if ((p_mutlicap_measure_data->acq_status & TOUCH_RESOLVE_CAL))
    {
        //Denotes sensor calibration is on-going.
    }
    if (!(p_mutlicap_measure_data->acq_status & TOUCH_BURST_AGAIN))
    {
        //Denotes acquisition is completed.
        /* Set the Mutual Cap measurement done flag. */
        p_mutlicap_measure_data->measurement_done_touch = 1u;
    }
}
```

**Continuous Measurement Post Processing Mode**

The sensor data post-processing mode for QTtouch library can be selected using the DEF_xxxxCAP TOUCH_POSTPROCESS_MODE configuration item available as part of touch.h file.

When TOUCH_LIBRARY_DRIVEN mode is selected, the library self-initiates repeated touch measurements to resolve touch press, release and calibration. This mode is suited for best response time.

When TOUCH_APPLN_DRIVEN mode is selected, the library does not initiate repeated touch measurement to resolve touch press, release and calibration. This mode suits deterministic PTC interrupt execution time for applications requiring stringent CPU time requirements. As repeated touch measurements are delayed due to other critical application code being executed. This mode can potentially affect the touch response time.

In order to improve the response time with the TOUCH_APPLN_DRIVEN mode, the following condition should be applied to initiate sensor measurement, so as to cater for additional measurements without any delay. The same condition can also be applied to other application scenario such as sleep to check for pending acquisitions to be completed before the system can go to sleep.

```c
if ((touch_time.time_to_measure_touch == 1u)||(p_mutlicap_measure_data->acq_status & TOUCH_BURST_AGAIN))
{
    /* Start a touch sensors measurement process periodically, or if there is a pending measurement. */
    touch_ret =
    touch_mutlicap_sensors_measure(touch_time.current_time_ms,NORMAL_ACQ_MODE,touch_mutlicap_measure_complete_callback);
}
```

### 5.9. Touch Library API Error

The following table provides the touch library API error code information. The API error code type is touch_ret_t enum.
<table>
<thead>
<tr>
<th>ErrorCode Enumeration</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOUCH_SUCCESS</td>
<td>Successful completion of operation</td>
</tr>
<tr>
<td>TOUCH_ACQ_INCOMPLETE</td>
<td>Touch Library is busy with pending previous touch measurement</td>
</tr>
<tr>
<td>TOUCH_INVALID_INPUT_PARAM</td>
<td>Invalid input parameter</td>
</tr>
<tr>
<td>TOUCH_INVALID_LIB_STATE</td>
<td>Operation not allowed in the current Touch Library state</td>
</tr>
<tr>
<td>TOUCH_INVALID_SELF_CAP_CONFIG_PARAM</td>
<td>Invalid self-capacitance configuration input parameter</td>
</tr>
<tr>
<td>TOUCH_INVALID_MUTCAP_CONFIG_PARAM</td>
<td>Invalid mutual capacitance configuration input parameter</td>
</tr>
<tr>
<td>TOUCH_INVALID_RECAL_THRESHOLD</td>
<td>Invalid Recalibration threshold input value</td>
</tr>
<tr>
<td>TOUCH_INVALID_CHANNEL_NUM</td>
<td>Channel number parameter exceeded total number of channels configured</td>
</tr>
<tr>
<td>TOUCH_INVALID_SENSOR_TYPE</td>
<td>Invalid sensor type. Sensor type can not be SENSOR_TYPE_UNASSIGNED</td>
</tr>
<tr>
<td>TOUCH_INVALID_SENSOR_ID</td>
<td>Invalid sensor number parameter</td>
</tr>
<tr>
<td>TOUCH_INVALID_RS_NUM</td>
<td>Number of Rotor/Sliders set as 0, when trying to configure a rotor/slider</td>
</tr>
</tbody>
</table>

The application error codes in touch projects can be enabled or disabled using a macro **DEF TOUCH_APP_ERR_HANDLER**. By default, the value of macro **DEF TOUCH_APP_ERR_HANDLER** is set to 0 in order to disable the application error handler. To enable the application error handler, set the macro **DEF TOUCH_APP_ERR_HANDLER** as 1. When it is enabled, **while(1)** is used to trap errors.

Refer Application Error Code (tag_touch_app_err_t) for further information.
6. Tuning for Noise Performance

The PTC has been designed with great care making it easy to design a capacitive touch solution, while at the same time maintaining high quality of touch and performance. Nevertheless in any touch sensing application, the system designer must consider how electrical interference in the target environment may affect the performance of the sensors.

Touch sensors with insufficient tuning can show failures in tests of either radiated or conducted noise, which can occur in the environment or power domain of the appliance or may be generated by the appliance itself during normal operation. In many applications there are quality standards which must be met where EMC performance criteria are clearly defined. However meeting the standards cannot be considered as proof that the system will never show EMC problems, as the standards include only the most commonly occurring types and sources of noise.

Noise immunity comes at a cost of increased touch response time and power consumption. The system designer must carry out proper tuning of the touch sensors in order to ensure least power consumption. The PTC QTouch library has a number of user configurable features which can be tuned to give the best balance between touch response time, noise immunity and power consumption.

6.1. Noise Sources

Noise sources that affect touch sensor performance can be classified as follows.

Self-generated
- Motors
- Piezo buzzers
- PWM controls Radiated
- Fluorescent lamp
- Radio transmission
- Inductive cook top Conducted
- Power supply / charger
- Mains supply

Applicable EMC standards
- Conducted Immunity EN61000-4-6

6.2. Noise Counter Measures

The effects of noise are highly dependent on the amplitude of the noise signal induced or injected onto the sensors, and the frequency profile of that noise signal.

Generally, this noise can be classified as -
- Broadband noise
- Narrow band noise

6.2.1. Broadband Noise Counter Measures

Broadband noise refers to noise signals whose frequency components are not harmonically related to the capacitance measurement acquisition frequencies of the PTC.

Provided that the maximum and minimum voltage levels of the acquisition signal combined with noise signals are within the input range of the PTC and a sufficiently large number of samples are taken,
broadband noise interference can be averaged out by setting a high value of Filter level 
(DEF_MUTLCAP_FILTER_LEVEL_PER_NODE, DEF_SELFCAP_FILTER_LEVEL_PER_NODE) and Auto 
oversample (DEF_MUTLCAP_AUTO_OS_PER_NODE, DEF_SELFCAP_AUTO_OS_PER_NODE) settings.

If the noise amplitude is excessive, then PTC components experience saturation of measurement. In this 
case the acquisition signals combined with the noise signals are outside the input range of the PTC, 
which results in clipping of the measurements.

Often the clipping is not observable in the resolved measurement, as it occurs only on a portion of the 
measurement samples, but the presence of clipped samples prevents effective averaging of the sample 
points.

In this case, averaging of samples will not result in a noise-free measurement even with large rates of 
oversampling. The resolved signal will show a shift from its correct level due to asymmetry of signal 
clipping.

<table>
<thead>
<tr>
<th>Configuration Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEF_MUTLCAP_FILTER_LEVEL_PER_NODE, DEF_SELFCAP_FILTER_LEVEL_PER_NODE</td>
<td>FILTER_LEVEL_64</td>
</tr>
<tr>
<td>DEF_MUTLCAP_AUTO_OS_PER_NODE, DEF_SELFCAP_AUTO_OS_PER_NODE</td>
<td>AUTO_OS_DISABLE</td>
</tr>
<tr>
<td>DEF_MUTLCAP_FREQ_MODE, DEF_SELFCAP_FREQ_MODE</td>
<td>FREQ_MODE_NONE</td>
</tr>
<tr>
<td>DEF_MUTLCAP_CLK_PRESCALE_PER_NODE, DEF_SELFCAP_CLK_PRESCALE_PER_NODE</td>
<td>PRSC_DIV_SEL_1</td>
</tr>
<tr>
<td>DEF_MUTLCAPSENSE_RESISTOR_PER_NODE, DEF_SELFCAPSENSE_RESISTOR_PER_NODE</td>
<td>RSEL_VAL_100</td>
</tr>
<tr>
<td>Auto-tune input to touch_mutlcap_sensors_calibrate(), touch_selfcap_sensors_calibrate API</td>
<td>AUTO_TUNE_PRSC</td>
</tr>
</tbody>
</table>

**STEP 1: PREVENT CLIPPING**

This requires the implementation of a hardware low pass filter in order to reduce the scale of the noise 
combined with acquisition signal. The sensor capacitance is combined with a series resistor on the Y 
(Sense) line, either the PTC internal resistor or externally mounted on the PCB. The external series 
resistor should be mounted between the Y line of the device to the Sensor, closest to the device pin.

**Note:** Always use an external series resistor for self-capacitance applications in order to prevent 
clipping. The internal series resistor of the PTC is limited to 100K. Depending on the noise levels, external 
series resistors up to 1 megaohms can be evaluated.

**STEP 2: CHARGE TRANSFER TEST**

As an effect of adding a series resistor to form a low pass filter, the time constant for charging the sensors 
is increased. It is essential to ensure that the sensor capacitance is fully charged and discharged during 
each measurement sampling.

Insufficient charging can be observed as a reduced touch delta or it may be seen on an oscilloscope by 
connecting to the sense electrode.

However, this problem may not be apparent in the touch sensor operation; the application may behave 
perfectly well even in the presence of low-level noise, but show much worse performance during noise 
tests with the addition of the resistor compared to a configuration which excludes the resistor.
Charge transfer through Auto tuning setting:

The QTouch library Auto tune setting provides a mechanism which carries out a charge transfer test on each enabled key and sets the prescalar to the fastest available setting ensuring full charge transfer.

The following combination of setting should be used.

- DEF_MUTLCAP_SENSE_RESISTOR_PER_NODE and DEF_SELFCAP_SENSE_RESISTOR_PER_NODE should be set to RSEL_VAL_100.
- Auto tune pre-scaler AUTO_TUNE_PRSC should be provided as input parameter to touch_mutlcap_sensors_calibrate(AUTO_TUNE_PRSC) and touch_selfcap_sensors_calibrate(AUTO_TUNE_PRSC).

Testing for Charge transfer by Manual tuning:

- If the AUTO_TUNE_NONE setting is provided as an input to the touch_mutlcap_sensors_calibrate(AUTO_TUNE_NONE) and touch_selfcap_sensors_calibrate(AUTO_TUNE_NONE) calibration API, then the PTC uses the user defined settings of the PTC Clock pre-scaler (DEF_MUTLCAP_CLK_PRESCALE, DEF_SELFCAP_CLK_PRESCALE_PER_NODE) and internal series resistor (DEF_MUTLCAP_SENSE_RESISTOR_PER_NODE, DEF_SELFCAP_SENSE_RESISTOR_PER_NODE).
- Reference measurement: An acquisition measurement (Signal value) is taken with the prescalar set to maximum, i.e. PRSC_DIV_SEL_8.

Test measurement: A second measurement (Signal value) is taken with reduced prescalar: PRSC_DIV_SEL_4.

If the difference between the two measurements is less than ~3% (1/32) of the first value, the conclusion is that full charge transfer is achieved with PRSC_DIV_SEL_4.

This measurement is repeated for PRSC_DIV_SEL_2 and PRSC_DIV_SEL_1 to find the fastest PTC operating speed for which full charge transfer is achieved.

STEP 3: ADJUST OVERSAMPLING

Once clipping is prevented by hardware filtering and full charge transfer is ensured the next step is to find the best settings for Filter level (DEF_MUTLCAP_FILTER_LEVEL_PER_NODE, DEF_SELFCAP_FILTER_LEVEL_PER_NODE) and Auto over samples (DEF_MUTLCAP_AUTO_OS_PER_NODE, DEF_SELFCAP_AUTO_OS_PER_NODE).

Auto over samples feature provides the advantage that additional samples are only taken on a sensor which has shown significant change. In the absence of such a change, the measurement cycle can be much shorter compared to applying (* AUTO_OS) as the oversampling rate on every measurement.

Care should be taken when using AUTO_OS to ensure that it does not occur too frequently.

The measurement time for FILTER_LEVEL samples can be represented as:

A + (B * FILTER_LEVEL)

Where A is the total time for PTC configuration and post-processing, and B is the oversampling period (the per sample measurement time).

When AUTO_OS is applied, this time is increased to:

A + (B * FILTER_LEVEL * (1 + AUTO_OS))

FILTER_LEVEL should be sufficiently large to ensure that AUTO_OS is only applied during the worst-case noise circumstances.
6.2.2. **Narrowband Noise Counter Measures**

If the noise includes a frequency component which is related to the PTC capacitance measurement acquisition frequency, then no amount of oversampling will average out the noise effects. Any batch of measurement samples taken with the same sampling frequency will result in a measurement offset. The actual offset resulting from each measurement depends on the relative phase of the noise component and the sampling frequency.

This effect is illustrated in the following diagram, where the noise is represented by a sine wave.

![Diagram showing narrowband noise counter measures](image)

**STEP 4: SELECT FREQUENCY MODE**

**Note:** Step1, Step2 and Step3 provided in the previous section should be used in combination with this step in a system which has both broadband noise and narrow band noise. Default settings provided before STEP1 should be used as a starting point before starting noise tuning.

With **FREQUENCY_MODE_NONE**, a single acquisition frequency is used and samples are taken at the fastest rate possible with the given pre-scalar setting. This gives the best response time, and with sufficient oversampling excellent noise immunity at all noise frequencies which are not related to the sampling frequency.

However in the case where the noise is at (or close to) a frequency which is harmonically related to the sampling frequency then the noise issue becomes severe, as illustrated above.

This is particularly important in applications where a frequency sweep test is required, such as EN61000-4-6.

**FREQUENCY_MODE_SPREAD** applies a modification to the sampling rate, such that the period between successive samples is modified in a saw-tooth fashion to apply a wider spectrum to the sampling frequency. The sampling frequency $F_0$ is thus spread across the range $(F_0/2, F_0)$. With relatively low noise amplitude, this can be effective at improving performance with minimal cost in response time.

**FREQUENCY_MODE_HOP** utilizes 3 base frequencies and a median filter to avoid using measurements taken at an affected frequency. The frequencies should be selected to minimize the set of crossover harmonics within the problem frequency band.

Each of the 3 frequencies is used in sequence for each measurement cycle, i.e.

- Cycle 1: All sensors measured with Frequency 0
- Cycle 2: All sensors measured with Frequency 1
- Cycle 3: All sensors measured with Frequency 2
- Cycle 4: All sensors measured with Frequency 0
- Cycle 5: All sensors measured with Frequency 1
If Frequency 0 is related to the noise frequency, then the measurements taken with F0 will show high variation. Using a median filter, this ensures that the outlying measurements will be rejected.

In some applications, self-generated noise may be present which affects one or more of the default HOP frequencies. In such a case, the HOP frequencies should be changed to avoid this frequency.

Some noise frequencies can occur which are close to harmonics of two of the HOP frequencies, in which case the system must be tuned with higher settings of FILTER_LEVEL or AUTO_OS to provide enough samples to average the noise out of the measurement.

**Determining PTC Acquisition Frequency**

The PTC acquisition frequency is given by the following formula,

\[
\text{PTC Acquisition Frequency} = \frac{1}{\text{PTC Acquisition Time}}
\]

The PTC acquisition time is given by the following formula,

\[
\text{PTC Acquisition Time} = (\text{Cycles per Acquisition} + \text{Hop Freq}) \times \text{PTC IO Clock Period}
\]

Where, Cycles per Acquisition = Number of PTC clock cycles required for each acquisition. This is a fixed value of 15. Hop Freq = PTC acquisition frequency delay setting.

This parameter is represented in the `touch.h` file by the symbols `DEF_MUTLCAP_HOP_FREQS` and `DEF_SELFCAP_HOP_FREQS`.

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 cycles between consecutive measurements.
- `FREQ_HOP_SEL_16` setting inserts 15 PTC clock cycles.

Hence, higher delay setting will increase the total time taken for capacitance measurement on a given sensor as compared to a lower delay setting. A desired setting can be used to avoid noise around the same frequency as the acquisition frequency.

**Range:** `FREQ_HOP_SEL_1` to `FREQ_HOP_SEL_16`

Three frequency hop delay settings need to be specified when assigning values to this parameter. Duration of each PTC clock period is given by the following formula,

Where,

\[
\text{CLKPTC} = \text{Generic clock input to the PTC}
\]

Refer `touch_configure_ptc_clock()` API in `touch.c` file for clock configuration.

Prescaler = PTC clock prescaler setting.

This parameter is represented in the `touch.h` file by the symbols `DEF_MUTLCAP_CLK_PRESCALE_PER_NODE` and `DEF_SELFCAP_CLK_PRESCALE_PER_NODE`.

**Example:** If Generic clock input to PTC = 4MHz, then:

- `PRSC_DIV_SEL_1` sets PTC Clock to 4MHz
- `PRSC_DIV_SEL_2` sets PTC Clock to 2MHz
- `PRSC_DIV_SEL_4` sets PTC Clock to 1MHz
- `PRSC_DIV_SEL_8` sets PTC Clock to 500KHz
When CLKPTC = 4MHz, Prescaler = `PRSC_DIV_SEL_1`, the PTC Acquisition Frequencies obtained are as follows,

<table>
<thead>
<tr>
<th>Hop Freq</th>
<th>PTC Acquisition Frequency(kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQ_HOP_SEL_1</td>
<td>66.67</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_2</td>
<td>62.50</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_3</td>
<td>58.82</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_4</td>
<td>55.56</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_5</td>
<td>52.63</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_6</td>
<td>50.00</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_7</td>
<td>47.62</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_8</td>
<td>45.45</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_9</td>
<td>43.48</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_10</td>
<td>41.67</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_11</td>
<td>40.00</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_12</td>
<td>38.46</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_13</td>
<td>37.04</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_14</td>
<td>35.71</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_15</td>
<td>34.48</td>
</tr>
<tr>
<td>FREQ_HOP_SEL_16</td>
<td>33.33</td>
</tr>
</tbody>
</table>

**Note:** The acquisition frequencies may vary based on the tolerance of the clock source.
7. Application Design

7.1. Touch Library and Associated Files

The table below provides the mandatory files required to use the QTouch library. In order to add QTouch functionality into an existing user example project, these files and associated library based on the compiler should be added to the user project.

Table 7-1. Touch Library Files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>touch_api_ptc.h</td>
<td>QTouch Library API header file, contains API and Data structure used to interface with the library</td>
</tr>
<tr>
<td>touch.h</td>
<td>QTouch library configuration header file</td>
</tr>
<tr>
<td>touch.c</td>
<td>A helper file to demonstrate QTouch library initialization and sensor configuration</td>
</tr>
<tr>
<td>libsamxxx_qtouch_iar.a or</td>
<td>QTouch library compiled for IAR or GCC compiler that supports both self-capacitance and mutual capacitance sensors.</td>
</tr>
<tr>
<td>libsamxxx_qtouch_gcc.a</td>
<td></td>
</tr>
</tbody>
</table>

7.2. Code and Data Memory Considerations

The table below captures the typical code and data memory required for QTouch library. The typical memory requirements provided in the table are arrived considering only Regular API usage in the application. Usage of Helper API would consume additional code memory.

Each measurement method requires additional data memory from the application for storing 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 sensors configured. The PRIV_xx_DATA_BLK_SIZE macro in touch_api_ptc.h calculates the size of this data memory block.

Table 7-2. Mutual Capacitance Method

<table>
<thead>
<tr>
<th>Series</th>
<th>Code Memory Keys Only</th>
<th>Data Memory Keys Only</th>
<th>Code Memory Keys with Rotor or Slider</th>
<th>Data Memory Keys with Rotor or Slider</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsamd1x-qtouch-gcc.a</td>
<td>9602</td>
<td>845</td>
<td>11114</td>
<td>861</td>
</tr>
<tr>
<td>libsamd1x-qtouch-iar.a</td>
<td>9005</td>
<td>497</td>
<td>10377</td>
<td>513</td>
</tr>
<tr>
<td>libsamd2x-qtouch-gcc.a</td>
<td>9222</td>
<td>841</td>
<td>10734</td>
<td>857</td>
</tr>
<tr>
<td>libsamd2x-qtouch-iar.a</td>
<td>8881</td>
<td>497</td>
<td>10254</td>
<td>513</td>
</tr>
<tr>
<td>libsaml21-qtouch-gcc.a</td>
<td>9282</td>
<td>841</td>
<td>10794</td>
<td>857</td>
</tr>
<tr>
<td>libsaml21-qtouch-iar.a</td>
<td>9744</td>
<td>497</td>
<td>11115</td>
<td>513</td>
</tr>
<tr>
<td>libsamdal-qtouch-gcc.a</td>
<td>9222</td>
<td>841</td>
<td>10734</td>
<td>857</td>
</tr>
</tbody>
</table>
In case of ATmega328PB, for a single touch channel (mutual capacitance mode) without noise, moisture, auto-tune and qdebug features, RAM usage is 503 bytes. RAM usage gets increased by 36 bytes for each additional channel.

Table 7-3. Self-capacitance Method

<table>
<thead>
<tr>
<th>Series</th>
<th>Code Memory Keys Only</th>
<th>Data Memory Keys Only</th>
<th>Code Memory Keys with Rotor or Slider</th>
<th>Data Memory Keys with Rotor or Slider</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsamdal-qtouch-gcc.a</td>
<td>9576</td>
<td>841</td>
<td>10884</td>
<td>849</td>
</tr>
<tr>
<td>libsamdal-qtouch-iar.a</td>
<td>8952</td>
<td>497</td>
<td>10216</td>
<td>505</td>
</tr>
<tr>
<td>libsamd2x-qtouch-gcc.a</td>
<td>9198</td>
<td>845</td>
<td>10506</td>
<td>845</td>
</tr>
<tr>
<td>libsamd2x-qtouch-iar.a</td>
<td>8841</td>
<td>497</td>
<td>10101</td>
<td>505</td>
</tr>
<tr>
<td>libatmega328pb_qtouch_gcc.a</td>
<td>9258</td>
<td>841</td>
<td>10566</td>
<td>845</td>
</tr>
<tr>
<td>libatmega328pb_qtouch.r90</td>
<td>9806</td>
<td>497</td>
<td>11070</td>
<td>505</td>
</tr>
<tr>
<td>libatmega324pb_qtouch_r90</td>
<td>10646</td>
<td>441</td>
<td>12379</td>
<td>562</td>
</tr>
</tbody>
</table>

Atmel QTouch Library Peripheral Touch Controller [USER GUIDE]
<table>
<thead>
<tr>
<th>Series</th>
<th>Code Memory Keys Only</th>
<th>Data Memory Keys Only</th>
<th>Code Memory Keys with Rotor or Slider</th>
<th>Data Memory Keys with Rotor or Slider</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsamr21-qtouch iar.a</td>
<td>8851</td>
<td>497</td>
<td>10115</td>
<td>505</td>
</tr>
<tr>
<td>libsaml22-qtouch gcc.a</td>
<td>9530</td>
<td>841</td>
<td>11158</td>
<td>845</td>
</tr>
<tr>
<td>libsaml22-qtouch iar.a</td>
<td>9410</td>
<td>501</td>
<td>10733</td>
<td>509</td>
</tr>
<tr>
<td>libatmega328pb_qtouch gcc.a</td>
<td>13274</td>
<td>500</td>
<td>15260</td>
<td>519</td>
</tr>
<tr>
<td>libMega328PB_qtouch.r90</td>
<td>10705</td>
<td>594</td>
<td>12041</td>
<td>613</td>
</tr>
<tr>
<td>libatmega324pb_qtouch gcc.a</td>
<td>14026</td>
<td>478</td>
<td>16024</td>
<td>593</td>
</tr>
<tr>
<td>libMega328PB_qtouch.r90</td>
<td>10596</td>
<td>437</td>
<td>12329</td>
<td>558</td>
</tr>
</tbody>
</table>

In case of ATmega328PB, for a single touch channel (self-capacitance mode) without noise, moisture, auto-tune and qdebug features, RAM usage is 500 bytes. RAM usage gets increased by 32 bytes for each additional channel.

**Note:**

1. The total number of sensors supported by a specific device variant is limited by the number of XY-lines as well as code, data, and stack memory requirements.
2. To save the memory utilized for code and data, new lib-nano C library has been used for GCC example projects.
8. Example Applications

8.1. Atmel Board Example Projects

The GCC Xplained Pro example projects can be accessed through File>New Example Project menu option in Atmel Studio.

The IAR Xplained Pro example projects can be accessed through Atmel QTouch Library PTC Partpack.

The following example projects are available for Xplained Pro kits:

- SAM D20 Xplained Pro and QT1 Xplained Pro Mutual Capacitance example application
- SAM D20 Xplained Pro and QT1 Xplained Pro Self Capacitance example application
- SAM D21 Xplained Pro and QT1 Xplained Pro Mutual Capacitance example application
- SAM D21 Xplained Pro and QT1 Xplained Pro Self Capacitance example application
- SAM D20 Xplained Pro and QT1 Xplained Pro Mutual Capacitance example application with Lump-Low Power configuration
- SAM D20 Xplained Pro and QT1 Xplained Pro Self Capacitance example application with Lump-Low Power configuration
- SAM D11 Xplained Pro Self Capacitance example application
- SAM D10 Xplained Mini Self Capacitance example application
- SAM D20 QTouch Robustness Demo Moisture Example Application (self + mutual)
- SAM C20 QTouch Robustness Demo Moisture Example Application
- SAM D20 Xplained Pro and QT3 Xplained Pro Mutual Capacitance example application with Lump-Low Power configuration
- SAM L21 Xplained Pro and QT3 Xplained Pro Mutual Capacitance example application with Lump-Low Power configuration
- SAM DA1 Xplained Pro and QT4 Xplained Pro Self Capacitance example application
- SAM C21 Xplained Pro and QT1 Xplained Pro Mutual Capacitance example application
- SAM C21 Xplained Pro and QT1 Xplained Pro Self Capacitance example application
- SAM C21 Xplained Pro Self Capacitance example application(on-board sensor)
- SAM C21 Xplained Pro and QT5 Xplained Pro Mutual Capacitance example application
- SAM L22 Xplained Pro and Touch Segment LCD Xplained Pro Mutual Capacitance example application
- ATmega328PB Xplained Mini Self Capacitance example application
- ATmega324PB Xplained Pro and QT5 Xplained Pro Mutual Capacitance example application

Note: For SAM L22, it is recommended to set the PTC Clock to 8MHz.

Clock Configuration Changes in Projects:

- For SAM C20/C21 RevB devices, DPLL is used as the main clock and OSC32K is used as reference clock for the DPLL clock source. For SAM C20/C21 RevC devices, OSC48MHz is used as the main clock. This is demonstrated in the example projects by using the same project for both SAM C20/C21 RevB and SAM C20/C21 RevC devices.
- The example projects which have DFLL as main clock source use scaled OSC8MHz/OSC16MHz clock as the reference input clock.
- SAM L22 example project configures DFLL for 16MHz (performance level - PL2) and utilizes it as the main clock. This clock setting offers high performance.
• SAM L22 low power user board project configures DFLL for 8MHZ (performance level - PL0) and utilizes it as the main clock. This clock setting offers low power consumption.

• SAM L21/L22 low power example projects are configured in PL0 (Low power oriented mode) with Buck regulator as the main regulator in standby sleep mode. This is the best suitable configuration to achieve low power numbers.

The example projects make use of Xplained Pro boards and the extension kits for showcasing touch. Those extension kits are explained in the following sections.

**QT1 Xplained Pro kit:**

The QT1 Xplained Pro self-capacitance and mutual capacitance extension boards are supported by SAM D20, SAM D21, SAM DA1, SAM C21, and SAM L22 Xplained Pro Evaluation kits.

![Figure 8-1. QT1 Xplained Pro Mutual Capacitance and Self-capacitance](image)

**Note:** SAM C21 Xplained Pro can operate at 3.3V and 5V Vcc, while the QT1 Xplained Pro can operate at a maximum voltage of 3.6V. Please make sure to put the Vcc selection header on the SAM C21 Xplained Pro in the 3.3V position.

The QT1 Xplained Pro boards demonstrate the following combinations of buttons, slider, and wheels.

- 2 buttons + 2 yellow LED
- 1 slider + 8 yellow LED
- 1 wheel + 1 RGB LED

**QT3 Xplained Pro kit:**

The QT3 Xplained Pro extension board has 12 mutual capacitance buttons on it and it is supported by SAM D20, SAM D21, SAM DA1, SAM L21, SAM L22 and SAM C21 Xplained Pro Evaluation kits.
QT4 Xplained Pro kit:
The QT4 Xplained Pro boards demonstrate the following arrangement.
• Two self-capacitance buttons
• One unshielded proximity sensor
• One proximity sensor with driven shield with external op-amp driver
• One LED indicator for each self-capacitive button
• One LED indicator for each proximity sensor

QT5 Xplained Pro kit:
The QT5 Xplained Pro board demonstrates the following arrangement.
• One 4-channel (4X x 1Y) mutual capacitance curved slider
• Two mutual capacitance buttons
• 16 LEDs arranged as two 7-segment digits separated with a colon
• IS31FL3728 I2C LED matrix controller from ISSI
8.2. User Board Example Projects

Atmel Studio QTouch Composer can be used to create GCC projects based on the sensor and pin configuration defined by the requirements of a user board. The generated example projects also allow for QDebug data streaming to QT Touch Analyzer.

The User board example project can be generated by accessing the QT Touch Composer using the following menu options in the Atmel Studio.

File > New Project > GCC C QT Touch Executable Project > Create QT Touch Library Project

The QT Touch Project Builder wizard appears as shown in the screenshot. Selection of sensors, devices, pins, debug interface and tuning of parameters can be done according to user preferences and project can be generated. The figure shows one of the user board generated projects.

Figure 8-5. QT Touch Project Builder
8.3. Using Atmel Software Framework (ASF) with the Example Projects
The example projects are based on Atmel Software Framework (ASF). For more information on ASF refer to Atmel Software Framework User Guide http://www.atmel.com/.

The Atmel® Software Framework (ASF) is a MCU software library providing a large collection of embedded software for Atmel flash MCUs: mega AVR, AVR XMEGA, AVR UC3 and SAM devices.

- It simplifies the usage of microcontrollers, providing an abstraction to the hardware and high-value middleware
- ASF is designed to be used for evaluation, prototyping, design and production phases
- ASF is integrated in the Atmel Studio IDE with a graphical user interface or available as standalone for GCC, IAR compilers
- ASF can be downloaded for free

8.4. Using Xplained Pro Kit to Program User Board
The SAM D20 Xplained Pro features a Cortex® Debug Connector (10-pin) for programming and debugging an external target. The connector is limited to the SWD interface and is intended for in-system programming and debugging of SAM D20 devices in the final product developed by the users. For more information refer SAM D20 Xplained Pro User Guide (www.atmel.com).

8.5. Using QDebug Touch Data Debug Communication Interface
When using IAR and GCC example projects, QDebug touch data debug communication interface can be enabled. This allows the communication between the touch device and QTouch Analyzer.

To enable or disable QDebug, configure `DEF TOUCH_QDEBUG_ENABLE` in the `touch.h` file.

When QDebug is enabled and touch debug data is being updated in the QTouch Analyzer, touch response time will be slower due to the debug communication data transfer which increases the delay in the response time.
After tuning the touch sensors using QTouch Analyzer, disable the QDebug for optimized touch performance.

**Figure 8-7. Atmel DGI Interface for QDebug Data**

![Atmel DGI Interface for QDebug Data](image)

**Figure 8-8. QTouch Analyzer view**

![QTouch Analyzer view](image)

### 8.6. Using Xplained Pro Kit for QDebug Data Streaming from User Board

SAM D20 Xplained Pro contains Embedded Debugger (EDBG) that features an Atmel Data Gateway Interface (DGI) over SPI and TWI. The DGI can be used to transmit a variety of data from the Xplained Pro kit to the host PC. This arrangement can be used to send QDebug data from a user board to Atmel Studio QTouch Analyzer for touch sensor data analysis and tuning.
The example project generated using QTouch composer makes use of SPI for data transfer. To stream QDebug data from user board, a relay firmware should be flashed onto the SAM D20/D21 microcontroller on the Xplained Pro kit. After connecting the SAM D20/D21 Xplained Pro to the PC, the device name appears in the connected kits of QTouch Start Page. Right click the device name and choose Enable User Board Analysis to flash the relay firmware.

The following table indicates the SPI connection between SAM D20 Xplained Pro Kit and User Board:

### Table 8-1. SPI Connection Information

<table>
<thead>
<tr>
<th>Pin on EXT3</th>
<th>Function</th>
<th>UserBoard Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>SPIMOSI (PB22)</td>
<td>MOSI</td>
</tr>
<tr>
<td>17</td>
<td>SPIMISO (PB16)</td>
<td>MISO</td>
</tr>
<tr>
<td>18</td>
<td>SPISCK(PB23)</td>
<td>SCK</td>
</tr>
<tr>
<td>-</td>
<td>SS-Connect to GND</td>
<td>GND</td>
</tr>
<tr>
<td>19</td>
<td>GND</td>
<td>GND</td>
</tr>
</tbody>
</table>
8.7. Using Atmel ICE for QDebug Data Streaming from User Board

Atmel ICE can be used to stream data from the user board. Refer the following table and connect the mini squid cable from AVR header of Atmel ICE to user board.

<table>
<thead>
<tr>
<th>Atmel-ICE AVR port pins</th>
<th>Target pins</th>
<th>Mini-squid pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin 1 (TCK)</td>
<td>SCK</td>
<td>1</td>
</tr>
<tr>
<td>Pin 2 (GND)</td>
<td>GND</td>
<td>2</td>
</tr>
<tr>
<td>Pin 3 (TDO)</td>
<td>MISO</td>
<td>3</td>
</tr>
<tr>
<td>Pin 4 (VTG)</td>
<td>VCC</td>
<td>4</td>
</tr>
<tr>
<td>Pin 5 (TMS)</td>
<td>SS</td>
<td>5</td>
</tr>
<tr>
<td>Pin 6 (nSRST)</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>Pin 7 (Not Connected)</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td>Pin 8 (nTRST)</td>
<td>-</td>
<td>8</td>
</tr>
<tr>
<td>Pin 9 (TDI)</td>
<td>MOSI</td>
<td>9</td>
</tr>
<tr>
<td>Pin 10 (GND)</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

While creating the project using QTouch composer project builder wizard, the pins SCK, MISO, SS and MOSI can be chosen from the debug interface setup pane as shown in the figure.

Figure 8-11. Debug Interface Setup Pane

When the connections are made correctly and debug interface setup is also done in the project, flash the project in the user board. Data can be streamed and visualized via QTouch Analyzer.

Note: Atmel ICE would be listed in QTouch Analyzer as QDEBUG_DGI.
9. **Known Issues**

1. **PTC in Self-capacitance Mode**

The following errata is applicable for SAM D20 (Revision B)

**Description:**

The two lowest gain settings are not selectable and an attempt by the QTouch Library to set enable of these may result in a higher sensitivity than optimal for the sensor. The PTC will not detect all touches. This errata does not affect mutual capacitance mode which operates as specified.

**Fix/workaround:**

Use SAM D20 revision C or later for self-capacitance capacitive touch sensing.

2. **Touch acquisition may fail and stop working**

The following errata is applicable for QTouch Library versions up to 5.0.7. This issue has been fixed in QTouch Library version 5.0.8 or later.

**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).

In an application with single touch channel and filter level 16, the total measurement time is ~350µs. The total measurement time doubles for two touch channels, and triples for 3 touch channels. It increases up to 10 times or 3.5ms with 10 touch channels.

**Fix/workaround:**

- **Recommended workaround:**
  - Use QTouch Library version 5.0.8 or later.
- **Other alternatives:**
  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.

```c
Disable_global_interrupt();
touch_ret = touch_xxxxcap_sensors_measure( touch_time.current_time_ms, 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.

<table>
<thead>
<tr>
<th>CPU Frequency (in MHz)</th>
<th>Interrupt Blocking Time (in µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>~77</td>
</tr>
<tr>
<td>24</td>
<td>~124</td>
</tr>
<tr>
<td>16</td>
<td>~176</td>
</tr>
<tr>
<td>12</td>
<td>~223</td>
</tr>
</tbody>
</table>
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 the `touch_xxxxxcap_sensors_measure` function.

When IAR compiler is used, utilize the `system_interrupt_enable_global()` and `system_interrupt_disable_global()` functions to enable and disable the global interrupts, respectively. In case of AVR, use `cli()` and `sei()` instructions to disable and enable the global interrupts.
# FAQ on PTC Qtouch

## Table 10-1. Frequently Asked Questions

<table>
<thead>
<tr>
<th>Query</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>When can we change an acquisition, sensor configuration or global sensor parameter? After changing an acquisition parameter do we need to recalibrate or reinitialize the sensors and PTC?</td>
<td>Its best to call the helper APIs to update these parameter when the measurement_done_touch flag (part of touch_measure_data_t structure) is true, which means the library is not in the middle of an (previously started) incomplete acquisition. Changing Gain and Filter level settings can affect the Signal value, so recalibration is mandatory by invoking the touch_sensors_calibrate() API.</td>
</tr>
<tr>
<td>Can sensors be disabled and re-enabled run time? For example, scan 2 sensors while sleeping and then scan all sensors when the system wakes up.</td>
<td>Yes, this is possible using the touch_xxxcap_sensor_disable() and touch_xxxcap_sensor_reenable() API.</td>
</tr>
<tr>
<td>There is a low amplitude pulse prior to the 16 acquisition samples and a large amplitude pulse after the 16 acquisition samples.</td>
<td>These pulses are part of setting up the sense line's initial conditions.</td>
</tr>
<tr>
<td>Is Detect integration calculated inside the PTC or by QTouch library?</td>
<td>This is done by QTouch library.</td>
</tr>
<tr>
<td>When Auto Oversampling is enabled how can one determine touch timing?</td>
<td>The absolute maximum cycle, is the case that auto oversamples is applied to all channels: (Normal acquisition time) x (1 + auto_os). This can only happen with a poorly tuned system, as FILTER_LEVEL should be sufficient to prevent AUTO_OS happening except on a touched key under noisy conditions.</td>
</tr>
<tr>
<td>Can sensor signal lines (Y or X lines) be used to drive LEDs, etc., when not being used for sensor acquisitions?</td>
<td>No. This is not recommended</td>
</tr>
</tbody>
</table>
11. Appendix

11.1. Macros

11.1.1. Touch Library Acquisition Status Bit Fields

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOUCH_NO_ACTIVITY</td>
<td>0x0000u</td>
<td>No touch activity.</td>
</tr>
<tr>
<td>TOUCH_IN_DETECT</td>
<td>0x0001u</td>
<td>Atleast one touch channel is in detect.</td>
</tr>
<tr>
<td>TOUCH_STATUS_CHANGE</td>
<td>0x0002u</td>
<td>Change in touch status of at least one Touch channel.</td>
</tr>
<tr>
<td>TOUCH_ROTOR_SLIDER_POS_CHANGE</td>
<td>0x0004u</td>
<td>Change in the position of at least one rotor or slider.</td>
</tr>
<tr>
<td>TOUCH_CHANNEL_REF_CHANGE</td>
<td>0x0008u</td>
<td>Change in the reference value of at least one touch channel.</td>
</tr>
<tr>
<td>TOUCH_BURST_AGAIN</td>
<td>0x0100u</td>
<td>Indicates that re-burst is required to resolve filtering or calibration state.</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_CAL</td>
<td>0x0200u</td>
<td>Indicates that re-burst is required to resolve calibration.</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_FILTERIN</td>
<td>0x0400u</td>
<td>Indicates that re-burst is required to resolve calibration.</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_DI</td>
<td>0x0800u</td>
<td>Indicates that re-burst is needed to resolve Detect Integration.</td>
</tr>
<tr>
<td>TOUCH_RESOLVE_POS_RECAL</td>
<td>0x1000u</td>
<td>Indicates that re-burst is needed to resolve recalibration.</td>
</tr>
<tr>
<td>TOUCH_CC_CALIB_ERROR</td>
<td>0x2000u</td>
<td>Indicates that CC calibration failed on at least one channel.</td>
</tr>
<tr>
<td>TOUCH_AUTO_OS_IN_PROGRESS</td>
<td>0x4000u</td>
<td>Indicates that Auto-os in progress to get stable channel signal.</td>
</tr>
</tbody>
</table>

11.1.2. Sensor State Configurations

GET_SENSOR_STATE (SENSOR_NUMBER)

To get the sensor state (whether detect or not) 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_XXXXCAP_SENSOR_MOIS_STATUS (SNSR_NUM)
To get the moisture status of a particular sensor. The return value is 1 in case of sensor is affected by moisture and returns 0 if sensor is affected by moisture.

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

**GET_XXXXCAP_MOIS_GRP_SUM_DELTA (GRP_ID)**

To get the xxxxcap moisture group sum delta.
The return value is 32 bit integer indicating the sum delta of moisture group.

```c
#define GET_XXXXCAP_MOIS_GRP_SUM_DELTA(GRP_ID)(mois_XXXX_grp_delta_arr[(GRP_ID)-1])
```

**GET_XXXXCAP_MOIS_GRP_ADJ_DELTA (GRP_ID)**

To get the xxxxcap moisture group Adjacent delta. The return value is 32 bit integer indicating the adjacent delta of moisture group.

```c
#define GET_MUTLCAP_MOIS_GRP_ADJ_DELTA(GRP_ID)(mois_mutl_grp_adj_delta_arr[(GRP_ID)-1])
```

**GET_MOIS_XXXX_GLOB_LOCK_STATE**

To get the moisture lock status of xxxxcap moisture groups. The return value is 1 if any moisture group is in moisture global lockout and 0 if no moisture group is in moisture global lockout.

```c
#define GET_MOIS_MUT_GLOB_LOCK_STATE(mois_lock_global_mutl)
```

**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.

```c
#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_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.

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

DEF_TOUCH_MUTLCAP must be set to 1 in the application to enable the Mutual capacitance touch acquisition method.

DEF_TOUCH_SELF_CAP must be set to 1 in the application to enable the Self-capacitance touch acquisition method.
### 11.2. Typedef

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threshold_t</td>
<td>uint8_t</td>
<td>An unsigned 8-bit number setting a sensor detection threshold.</td>
</tr>
<tr>
<td>sensor_id_t</td>
<td>uint8_t</td>
<td>Sensor number type.</td>
</tr>
<tr>
<td>touch_current_time_t</td>
<td>uint16_t</td>
<td>Current time type.</td>
</tr>
<tr>
<td>touch_delta_t</td>
<td>int16_t</td>
<td>Touch sensor delta value type.</td>
</tr>
<tr>
<td>touch_acq_status_t</td>
<td>uint16_t</td>
<td>Status of touch measurement.</td>
</tr>
<tr>
<td>mois_snsr_threshold_t</td>
<td>int32_t</td>
<td>Moisture threshold for individual sensor.</td>
</tr>
<tr>
<td>mois_system_threshold_t</td>
<td>int32_t</td>
<td>Moisture threshold for the entire system.</td>
</tr>
</tbody>
</table>

### 11.3. Enumeration

#### 11.3.1. Gain Setting (tag_gain_t)

Gain per touch channel.

Gain is applied on a per-channel basis to allow a scaling-up of the touch sensitivity on contact.

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

#### 11.3.2. Filter Level Setting (tag_filter_level_t)

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. Refer filter_level_t in touch_api_ptc.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
11.3.3. **Auto Oversample Setting** (tag_auto_os_t)

Auto oversample controls the automatic oversampling of sensor channels when unstable signals are detected with the default setting of 'Filter level'. Enabling Auto oversample results in 'Filter level' x 'Auto Oversample' number of samples taken 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 taken with stable signal values and 4+16 oversamples are taken when unstable signal is detected.

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

11.3.4. **Low Power Sensor Scan Rate** (tag_lowpower_scan_int_t)

When the CPU returns to standby mode from active, the sensor configured as the low power sensor is scanned at this interval. A high value for this parameter will reduce power consumption but increase response time for a low power sensor.

*Note:* This enum is applicable only for ATmega devices.

Range: LOWPOWER_PER0_SCAN_3_P_9_MS to LOWPOWER_PER7_SCAN_250_MS

**Data Fields**
- LOWPOWER_PER0_SCAN_3_P_9_MS
- LOWPOWER_PER1_SCAN_7_P_8_MS
- LOWPOWER_PER2_SCAN_15_P_625_MS
- LOWPOWER_PER3_SCAN_31_P_25_MS
- LOWPOWER_PER4_SCAN_62_P_5_MS
- LOWPOWER_PER5_SCAN_125_MS
- LOWPOWER_PER6_SCAN_250_MS

11.3.5. **Library Error Code** (tag_touch_ret_t)

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_SELF_CAP_CONFIG_PARAM Invalid self-capacitance configuration input parameter.
11.3.6. Application Error Code (tag_touch_app_err_t)
The application error codes are listed below.

Data Fields
- TOUCH_INIT_CONFIG_ERR The `touch_xxxxxcap_sensors_init` is fed with an incompatible / incomplete parameter.
- TOUCH_SENSOR_CONFIG_ERR The `touch_xxxxxcap_sensor_config` is fed with an incompatible parameter / Touch Library state is not in TOUCH_STATE_INIT.
- TOUCH_INIT_CALIB_ERR The `touch_xxxxxcap_sensors_calibrate` is fed with an invalid parameter / Touch Library state is TOUCH_STATE_NULL/ TOUCH_STATE_BUSY.
- TOUCH_MEASURE_INCOMPLETE The `touch_measure api` has error due to an invalid input param / it was on an invalid Touch Library state.
- TOUCH_MEASURE_CC_CAL_FAILED Hardware calibration error; check the hardware and ensure it is proper. If the error persists, check the user manual for sensor design guidelines.

11.3.7. Touch Channel (tag_channel_t)
Sensor start and end channel type of a Sensor. Channel number starts with value 0.

Data Fields
CHANNEL_0 to CHANNEL_255

11.3.8. Touch Library State (tag_touch_lib_state_t)
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.

11.3.9. Sensor Type (tag_touch_lib_state_t)
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_LUMP Sensor type lump
• SENSOR_TYPE_SLIDER Sensor type slider
• MAX_SENSOR_TYPE Max value of enum type for testing

11.3.10. Touch Sensing Type (tag_touch_acq_t)
Based on the two types of charge transfer technology, the capacitive touch sensing may be either mutual capacitance sensing or self-capacitance sensing.

Data Fields
• TOUCH_MUTUAL Mutual capacitance sensing
• TOUCH_SELF Self-capacitance sensing
• MAX_TOUCH_ACQ Max value of enum

11.3.11. Touch Library Acquisition Mode (tag_touch_acq_mode_t)
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.

11.3.12. Calibration Auto tune Setting (tag_auto_tune_type_t)
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.

11.3.13. PTC Acquisition Frequency Mode Setting (tag_freq_mode_sel_t)
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

11.3.14. PTC Clock Pre-scaler Setting (tag_prsc_div_sel_t)

Refer touch_configure_ptc_clock() API in touch.c

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

11.3.15. PTC Series Resistor Setting (tag_rsel_val_t)

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.

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

11.3.16. PTC Acquisition Frequency Delay Setting (tag_rsel_val_t)

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.

A desired setting avoids noise in the same frequency as the acquisition frequency.

Data Fields
11.3.17. AKS Group (tag_aks_group_t)

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.

11.3.18. Sensor Hysteresis Setting (tag_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).**

**Note:** 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)

**Data Fields**

- **HYST_50**
11.3.19. Sensor Recalibration Threshold (*tag_recal_threshold_t*)
This is expressed as a percentage of the sensor detection threshold. \( \text{RECAL}_x \) = recalibration threshold is \( x \)% of detection threshold value (rounded down).

**Note:** A minimum value of 4 is used.

**Example:** If detection threshold = 40,
- \( \text{RECAL}_100 \) = 40 (100% of 40)
- \( \text{RECAL}_50 \) = 20 (50% of 40)
- \( \text{RECAL}_25 \) = 10 (25% of 40)
- \( \text{RECAL}_12.5 \) = 5 (12.5% of 40)
- \( \text{RECAL}_6.25 \) = 4 (6.25% of 40 = 2, but value is hard limited to 4)

**Data Fields**
- \( \text{RECAL}_100 \)
- \( \text{RECAL}_50 \)
- \( \text{RECAL}_25 \)
- \( \text{RECAL}_12.5 \)
- \( \text{RECAL}_6.25 \)
- \( \text{MAX\_RECAL} \) Max value of enum type for testing.

11.3.20. Rotor Slider Resolution (*tag_resolution_t*)
For rotors and sliders, the resolution of the reported angle or position.

- \( \text{RES}\_x\_BIT \) = rotor/slider reports \( x \)-bit values.

**Example:** If slider resolution is \( \text{RES}_7\_BIT \), then reported positions are in the range 0..127.

**Data Fields**
- \( \text{RES}_1\_BIT \)
- \( \text{RES}_2\_BIT \)
- \( \text{RES}_3\_BIT \)
- \( \text{RES}_4\_BIT \)
- \( \text{RES}_5\_BIT \)
- \( \text{RES}_6\_BIT \)
- \( \text{RES}_7\_BIT \)
- \( \text{RES}_8\_BIT \)
- \( \text{MAX\_RES} \) Max value of enum type for testing.

11.3.21. PTC Sensor Noise Lockout setting (*nm_sensor_lockout_t*)
The sensor lockout setting option allows the system to be configured in the following modes.

- \( \text{SINGLE\_SENSOR\_LOCKOUT} \) Single sensor can be locked out.
- \( \text{GLOBAL\_SENSOR\_LOCKOUT} \) All the sensors are locked out for touch detection.
- \( \text{NO\_LOCK\_OUT} \) 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

11.3.22. 11_3_21_PTC_GPIO_STATE(ptc_gpio_state_t)

Detailed Description
PTC lines state in unmeasured condition can be set using this enum
- PULLHIGH_WHEN_NOT_MEASURED Indicates that default state of PTC lines are at vcc.
- GND_WHEN_NOT_MEASURED Indicates that default state PTC lines are grounded.

Range: PULLHIGH_WHEN_NOT_MEASURED=0 and GND_WHEN_NOT_MEASURED.

Data Fields
- PULLHIGH_WHEN_NOT_MEASURED
- GND_WHEN_NOT_MEASURED

11.3.23. 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

11.3.24. 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
11.3.25. **Touch Mode Configuration** (tag_tch_mode)

Touch mode can be configured.

**Note:** This is applicable only for ATmega devices.

**Data Fields**

- TCH_MODE_POLLED Polled mode
- TCH_MODE_ISR Interrupt mode
- TCH_MODE_NONE Touch mode is null.

11.3.26. **Trigger Mode** (tag_trigger_mode)

Trigger source for continuous hardware PTC acquisition. It is n clock cycles of internal 128Khz clock.

**Note:** This is applicable only for ATmega devices.

**Data Fields**

- TCH_TRIGGER_128KHZ_4MS
- TCH_TRIGGER_128KHZ_8MS
- TCH_TRIGGER_128KHZ_16MS
- TCH_TRIGGER_128KHZ_32MS
- TCH_TRIGGER_128KHZ_64MS
- TCH_TRIGGER_128KHZ_128MS
- TCH_TRIGGER_128KHZ_256MS

11.4. **Datastructures**

11.4.1. **Touch Library Timing Info** (tag_touch_time_t)

Touch library time parameter.

**Data Fields**
### Field Info

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>measurement_period_ms</td>
<td>uint16_t</td>
<td>Touch measurement period in milliseconds. This variable determines how often a new touch measurement must be done.</td>
</tr>
<tr>
<td>current_time_ms</td>
<td>volatile</td>
<td>Current time set by timer ISR.</td>
</tr>
<tr>
<td>time_to_measure_touch</td>
<td>volatile</td>
<td>Flag set by timer ISR when it is time to measure touch.</td>
</tr>
</tbody>
</table>

#### 11.4.2. Sensor Info (tag_sensor_t)

Sensor structure for storing sensor related information.

**Data Fields**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>uint8_t</td>
<td>Sensor state (calibrate, on, off, filter-in, filter-out, disable, pos-recal)</td>
</tr>
<tr>
<td>general_counter</td>
<td>uint8_t</td>
<td>General purpose counter used for calibrating, drifting, etc</td>
</tr>
<tr>
<td>ndil_counter</td>
<td>uint8_t</td>
<td>Counter used for detect integration</td>
</tr>
<tr>
<td>type_aks_pos_hyst</td>
<td>uint8_t</td>
<td>bits 7..6: sensor type: {00: key, 01: rotor, 10: slider, 11: reserved} bits 5..3: AKS group (0..7): 0 = no AKS group bit 2 : positive recal flag bits 1..0: hysteresis</td>
</tr>
<tr>
<td>threshold</td>
<td>uint8_t</td>
<td>Sensor detection threshold</td>
</tr>
<tr>
<td>from_channel</td>
<td>uint8_t</td>
<td>Sensor from channel for keys: from channel = to channel. Rotors: Top channel. Sliders : Left most channel</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> We need to_channel for rotors/sliders only</td>
</tr>
<tr>
<td>to_channel</td>
<td>uint8_t</td>
<td>For keys, this is unused. For rotors: Bottom left channel. For sliders: Middle channel</td>
</tr>
<tr>
<td>index</td>
<td>uint8_t</td>
<td>Index into array of rotor/slider values</td>
</tr>
</tbody>
</table>

#### 11.4.3. Global Sensor Configuration Info (tag_touch_global_param_t)

Touch library global parameter.

**Data Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>di</td>
<td>uint8_t</td>
<td>Detect Integration (DI) limit.</td>
</tr>
<tr>
<td>atch_drift_rate</td>
<td>uint8_t</td>
<td>Sensor away from touch drift rate.</td>
</tr>
<tr>
<td>tch_drift_rate</td>
<td>uint8_t</td>
<td>Sensor towards touch drift rate.</td>
</tr>
<tr>
<td>max_on_duration</td>
<td>uint8_t</td>
<td>Maximum ON time duration.</td>
</tr>
<tr>
<td>drift_hold_time</td>
<td>uint8_t</td>
<td>Sensor drift hold time.</td>
</tr>
<tr>
<td>atch_recal_delay</td>
<td>uint8_t</td>
<td>Sensor away from touch recalibration delay.</td>
</tr>
<tr>
<td>Field</td>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>cal_seq_1_count</td>
<td>uint8_t</td>
<td>Sensor calibration dummy burst count.</td>
</tr>
<tr>
<td>cal_seq_2_count</td>
<td>uint8_t</td>
<td>Sensor calibration settling burst count.</td>
</tr>
<tr>
<td>recal_threshold</td>
<td>recal_threshold_t</td>
<td>Sensor away from touch recalibration threshold.</td>
</tr>
<tr>
<td>touch_postprocess_mode</td>
<td>Uint16_t</td>
<td>Sensor post-processing mode.</td>
</tr>
<tr>
<td>auto_os_sig_stability_limit</td>
<td>uint8_t</td>
<td>Stability limit for Auto Oversample feature.</td>
</tr>
<tr>
<td>auto_tune_sig_stability_limit</td>
<td>uint16_t</td>
<td>Stability limit for frequency auto tune feature.</td>
</tr>
<tr>
<td>auto_freq_tune_in_cnt</td>
<td>uint8_t</td>
<td>Frequency auto tune In counter.</td>
</tr>
<tr>
<td>nm_sig_stability_limit</td>
<td>uint16_t</td>
<td>Stability limit for noise counter.</td>
</tr>
<tr>
<td>nm_noise_limit</td>
<td>uint8_t</td>
<td>Noise limit.</td>
</tr>
<tr>
<td>nm_enable_sensor_lock_out</td>
<td>nm_sensor_lockout_t</td>
<td>Sensor lockout feature variable.</td>
</tr>
<tr>
<td>nm_lockout_countdown</td>
<td>uint8_t</td>
<td>Lockout countdown for noise measurement.</td>
</tr>
<tr>
<td>Charge_share_delay</td>
<td>uint8_t</td>
<td>Charge share delay value; applicable only for SAM C20, SAM C21, SAM L22 and ATmega devices.</td>
</tr>
</tbody>
</table>

### 11.4.4. Filter Callback Data Type (tag_touch_filter_data_t)

Touch library filter callback data type.

**Data Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_channel_signals</td>
<td>uint16_t</td>
<td>Length of the measured signal values list.</td>
</tr>
<tr>
<td>p_channel_signals</td>
<td>uint16_t</td>
<td>Pointer to measured signal values for each channel.</td>
</tr>
</tbody>
</table>

### 11.4.5. Measure Data Type (tag_touch_measure_data_t)

Touch library measure data type.

**Data Fields**
<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>measurement_done_touch</td>
<td>volatile uint8_t</td>
<td>Flag set by touch_xxxxcap_measure_complete_callback() function when a latest Touch status is available.</td>
</tr>
<tr>
<td>acq_status</td>
<td>touch_acq_status_t</td>
<td>Status of touch measurement.</td>
</tr>
<tr>
<td>num_channel_signals</td>
<td>uint16_t</td>
<td>Length of the measured signal values list.</td>
</tr>
<tr>
<td>*p_channel_signals</td>
<td>uint16_t</td>
<td>Pointer to measured signal values for each channel.</td>
</tr>
<tr>
<td>num_channel_references</td>
<td>uint16_t</td>
<td>Length of the measured reference values list.</td>
</tr>
<tr>
<td>*p_channel_references</td>
<td>uint16_t</td>
<td>Pointer to measured reference values for each channel.</td>
</tr>
<tr>
<td>num_sensor_states</td>
<td>uint8_t</td>
<td>Number of sensor state bytes.</td>
</tr>
<tr>
<td>*p_sensor_states</td>
<td>uint8_t</td>
<td>Pointer to touch status of each sensor.</td>
</tr>
<tr>
<td>num_rotor_slider_values</td>
<td>uint8_t</td>
<td>Length of the rotor and slider position values list.</td>
</tr>
<tr>
<td>*p_rotor_slider_values</td>
<td>uint8_t</td>
<td>Pointer to rotor and slider position values.</td>
</tr>
<tr>
<td>num_sensors</td>
<td>uint16_t</td>
<td>Length of the sensors data list.</td>
</tr>
<tr>
<td>*p_cc_calibration_vals</td>
<td>uint16_t</td>
<td>Pointer to calibrated compensation values for a given sensor channel.</td>
</tr>
<tr>
<td>*p_sensors</td>
<td>sensor_t</td>
<td>Pointer to sensor data.</td>
</tr>
<tr>
<td>*p_sensor_noise_status</td>
<td>uint8_t</td>
<td>Pointer to noise status of the sensors.</td>
</tr>
<tr>
<td>*p_nm_ch_noise_val</td>
<td>uint16_t</td>
<td>Pointer to noise level value of each channel.</td>
</tr>
<tr>
<td>*p_sensor_moistatus</td>
<td>uint8_t</td>
<td>Pointer to moisture status</td>
</tr>
<tr>
<td>*p_auto_os_status</td>
<td>uint8_t</td>
<td>Pointer to auto-oversamples status</td>
</tr>
<tr>
<td>cc_calib_status_flag</td>
<td>uint8_t</td>
<td>Flag is set when CC-calibration is ongoing.</td>
</tr>
</tbody>
</table>

### 11.4.6. Sensor Configuration Parameter

(tag_touch_selfcap_param_t,tag_touch_mutlcap_param_t)

Touch library self-capacitance and mutual capacitance sensor parameter.

**Data Fields**
<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aks_group</td>
<td>aks_group_t</td>
<td>Which AKS group, the sensor belongs to.</td>
</tr>
<tr>
<td>detect_threshold</td>
<td>threshold_t</td>
<td>An unsigned 8-bit number setting a sensor detection threshold.</td>
</tr>
<tr>
<td>detect_hysteresis</td>
<td>hysteresis_t</td>
<td>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)</td>
</tr>
<tr>
<td>position_resolution</td>
<td>resolution_t</td>
<td>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</td>
</tr>
<tr>
<td>position_hysteresis</td>
<td>uint8_t</td>
<td>Sensor position hysteresis. This is valid only for a rotor or slider. bits 1..0: hysteresis. Note: This parameter is valid only for mutual capacitance method.</td>
</tr>
</tbody>
</table>

11.4.7. Sensor Acquisition Parameter
(tag_touch_selfcap_acq_param_t, _tag_touch_mutlcap_acq_param_t)

Sensor acquisition parameter.

Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*p_xxxxxcap_gain_per_node</td>
<td>gain_t</td>
<td>Pointer to gain per node.</td>
</tr>
<tr>
<td>touch_xxxxxcap_freq_mode</td>
<td>Freq_mode_sel_t</td>
<td>Set-up acquisition frequency mode.</td>
</tr>
<tr>
<td>*xxxxxcap_ptc_prsc</td>
<td>prsc_div_sel_t</td>
<td>Pointer to PTC clock pre-scaler value.</td>
</tr>
<tr>
<td>*xxxxxcap_resistor_value</td>
<td>rsel_val_t</td>
<td>Pointer to PTC series resistor value.</td>
</tr>
<tr>
<td>p_xxxxxcap_hop_freqs</td>
<td>*freq_hop_sel_t</td>
<td>Pointer to acquisition frequency settings.</td>
</tr>
<tr>
<td>*p_xxxxxcap_filter_level</td>
<td>filter_level_t</td>
<td>Pointer to filter level.</td>
</tr>
<tr>
<td>*p_xxxxxcap_auto_os</td>
<td>auto_os_t</td>
<td>Pointer to auto oversampling.</td>
</tr>
</tbody>
</table>
11.4.8. **Self-capacitance Sensor Configuration** *(touch_selfcap_config_t)*

Touch Library self-capacitance configuration input type.

### Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*xxxxxcap_ptc_prsc_cc_cal</td>
<td>prsc_div_sel_t</td>
<td>Pointer to PTC clock prescale value during CC calibration.</td>
</tr>
<tr>
<td>*xxxxxcap_resistor_value_cc_cal</td>
<td>rsel_val_t</td>
<td>Pointer to PTC sense resistor value during CC calibration.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_channels</td>
<td>uint16_t</td>
<td>Number of channels.</td>
</tr>
<tr>
<td>num_sensors</td>
<td>uint16_t</td>
<td>Number of sensors.</td>
</tr>
<tr>
<td>num_rotors_and_sliders</td>
<td>uint8_t</td>
<td>Number of rotors/sliders.</td>
</tr>
<tr>
<td>global_param</td>
<td>touch_global_param_t</td>
<td>Noise measurement enable/disable.</td>
</tr>
<tr>
<td>touch_xxxxcap_acq_param</td>
<td>touch_xxxxcap_acq_param_t</td>
<td>Sensor acquisition parameter info.</td>
</tr>
<tr>
<td>*p_data_blk</td>
<td>uint8_t</td>
<td>Pointer to data block buffer.</td>
</tr>
<tr>
<td>*buffer_size</td>
<td>uint16_t</td>
<td>Size of data block buffer.</td>
</tr>
<tr>
<td>*p_mutlcap_xy_nodes</td>
<td>uint16_t</td>
<td>Pointer to xy-nodes.</td>
</tr>
<tr>
<td>mutl_quick_reburst_enable</td>
<td>uint8_t</td>
<td>Quick re-burst enable.</td>
</tr>
<tr>
<td>(touch_filter_data_t *p_filter_data)</td>
<td>void(* filter_callback )</td>
<td>Mutual capacitance filter callback.</td>
</tr>
<tr>
<td>enable_freq_auto_tune</td>
<td>uint8_t</td>
<td>Frequency auto tune enable.</td>
</tr>
<tr>
<td>enable_noise_measurement</td>
<td>uint8_t</td>
<td>Noise measurement enable.</td>
</tr>
<tr>
<td>nm_buffer_cnt</td>
<td>uint8_t</td>
<td>Memory allocation buffer.</td>
</tr>
<tr>
<td>mutl_mois_tlrnce_enable</td>
<td>uint8_t</td>
<td>Mutual capacitance moisture tolerance enable flag.</td>
</tr>
<tr>
<td>mutl_mois_groups</td>
<td>uint8_t</td>
<td>Number of mutual capacitance moisture groups.</td>
</tr>
</tbody>
</table>
### 11.4.10. Moisture Structure (tag_snsr_mois_t)
Structure for storing moisture and multi-touch group information.

**Data Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mois_grp</td>
<td>uint8_t</td>
<td>Moisture group member</td>
</tr>
<tr>
<td>multch_grp</td>
<td>uint8_t</td>
<td>Multi-touch group member</td>
</tr>
</tbody>
</table>

### 11.4.11. Touch Library Input Configuration (touch_config_t)
Touch Library Input Configuration Structure.

**Data Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_mutlcap_config</td>
<td>touch_mutlcap_config_t</td>
<td>Pointer to mutual capacitance configuration structure.</td>
</tr>
<tr>
<td>p_selfcap_config</td>
<td>touch_selfcap_config_t</td>
<td>Pointer to self-capacitance configuration structure.</td>
</tr>
<tr>
<td>ptc_isr_lvl</td>
<td>uint8_t</td>
<td>PTC ISR priority level. Note: This is applicable only for SAM devices.</td>
</tr>
<tr>
<td>tch_mode</td>
<td>tch_mode_t</td>
<td>Touch mode configuration. Note: This is applicable only for ATmega devices.</td>
</tr>
</tbody>
</table>

### 11.4.12. Library Function List (tag_tlib_init_fn_ptr_t)
Touch Library support functions initializer.

**Data Fields**

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_tune_init</td>
<td>void(*auto_tune_init)</td>
<td>Auto-tune function initializer</td>
</tr>
<tr>
<td>auto_os_init</td>
<td>uint32_t (*auto_os_init)</td>
<td>Auto-OS function initializer</td>
</tr>
<tr>
<td>lk_chk</td>
<td>void(*lk_chk)</td>
<td>Sensor lock-out function initializer</td>
</tr>
<tr>
<td>enable_aks</td>
<td>void enable_aks(void)</td>
<td>AKS function initializer</td>
</tr>
</tbody>
</table>

### 11.4.13. Touch Library Information (tag_touch_info_t)
Touch Library information structure.
Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tlib_state</td>
<td>touch_tlib_state_t</td>
<td>Touch library state is specified</td>
</tr>
<tr>
<td>num_channels_in_use</td>
<td>unit16_t</td>
<td>Number of channels in use; irrespective of the corresponding sensor being disabled or enabled</td>
</tr>
<tr>
<td>num_sensors_in_use</td>
<td>uint16_t</td>
<td>Number of sensors in use; irrespective of the sensor being disabled or enabled</td>
</tr>
<tr>
<td>num_rotors_sliders_in_use</td>
<td>uint8_t</td>
<td>Number of rotor sliders in use; irrespective of the Rotor/Slider being disabled or enabled</td>
</tr>
<tr>
<td>max_channels_per_rotor_slider</td>
<td>uint8_t</td>
<td>Max possible number of channels per rotor or slider</td>
</tr>
</tbody>
</table>

11.4.14. Touch Library Version Information (touch_libver_info_t)

Touch Library version information structure.

Data Fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chip_id</td>
<td>unit32_t</td>
<td>Chip ID</td>
</tr>
<tr>
<td>product_id</td>
<td>unit16_t</td>
<td>Product ID</td>
</tr>
<tr>
<td>fw_version</td>
<td>uint16_t</td>
<td>Touch Library Version</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits[12:15] Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits[8:11] TLIB_MAJOR_VERSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits[4:7] TLIB_MINOR_VERSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits[0:3] TLIB_PATCH_VERSION</td>
</tr>
</tbody>
</table>

11.5. Global Variables

<table>
<thead>
<tr>
<th>Field</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>touch_time</td>
<td>touch_time_t</td>
<td>This holds the library timing info</td>
</tr>
<tr>
<td>touch_acq_status</td>
<td>touch_acq_status_t</td>
<td>This holds the Touch Library acquisition status</td>
</tr>
<tr>
<td>cc_cal_max_signal_limit</td>
<td>uint16_t</td>
<td>CC calibration maximum signal limit variable</td>
</tr>
<tr>
<td>cc_cal_min_signal_limit</td>
<td>uint16_t</td>
<td>CC calibration minimum signal limit variable</td>
</tr>
</tbody>
</table>
### API

#### 11.6.1. Sensor Init and De-init

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>touch_ret_t touch_mutlcap_sensors_init (touch_config_t * p_touch_config)</code></td>
<td>This API is used to initialize the Touch Library with Mutual cap or Self cap method pin, register and sensor configuration provided by the user. <strong>Parameters:</strong> <code>p_touch_config</code> Pointer to Touch configuration structure. <strong>Returns:</strong> <code>touch_ret_t</code>: Touch Library Error status.</td>
</tr>
<tr>
<td><code>touch_ret_t touch_selfcap_sensors_init (touch_config_t * p_touch_config)</code></td>
<td></td>
</tr>
<tr>
<td><code>touch_ret_t touch_mutlcap_sensors_deinit(void)</code></td>
<td>This API can be used to de-initialize the sensor for specific sensing group. <strong>Parameters:</strong> void. <strong>Returns:</strong> <code>touch_ret_t</code>: Touch Library Error status.</td>
</tr>
<tr>
<td><code>touch_ret_t touch_selfcap_sensors_deinit(void);</code></td>
<td></td>
</tr>
</tbody>
</table>

#### 11.6.2. Sensor Setup and Configuration

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>touch_ret_t touch_mutlcap_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_tposition_resolution, uint8_t position_hysteresis, sensor_id_t * p_sensor_id)</code></td>
<td></td>
</tr>
<tr>
<td><code>touch_ret_t touch_selfcap_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_tposition_resolution, sensor_id_t * p_sensor_id)</code></td>
<td></td>
</tr>
</tbody>
</table>
This API can be used to configure a sensor of type key, rotor or slider.

### Data Fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sensor_type</td>
<td>can be of type key, lump, rotor, or slider.</td>
</tr>
<tr>
<td>from_channel</td>
<td>the first channel in the slider sensor.</td>
</tr>
<tr>
<td>to_channel</td>
<td>the last channel in the slider sensor.</td>
</tr>
<tr>
<td>aks_group</td>
<td>which AKS group (if any) the sensor is in.</td>
</tr>
<tr>
<td>detect_threshold</td>
<td>the sensor detection threshold.</td>
</tr>
<tr>
<td>detect_hysteresis</td>
<td>the sensor detection hysteresis value.</td>
</tr>
<tr>
<td>position_resolution</td>
<td>the resolution of the reported position value.</td>
</tr>
<tr>
<td>position_hysteresis</td>
<td>the hysteresis for position value (available only for mutual capacitance mode).</td>
</tr>
<tr>
<td>p_sensor_id</td>
<td>the sensor id value of the configured sensor is updated by the Touch Library.</td>
</tr>
</tbody>
</table>

**Returns:** `touch_ret_t`: Touch Library Error status.

11.6.3. **Sensor Calibration**

```c
touch_ret_t touch_mutlcap_sensors_calibrate (auto_tune_type_t )
```

```c
touch_ret_t touch_selfcap_sensors_calibrate (auto_tune_type_t )
```

This API is used to calibrate the sensors for the first time before starting a Touch measurement. This API can also be used to force calibration of sensors when any of the Touch sensor parameters are changed during runtime.

**Returns:** `touch_ret_t`: Touch Library Error status.

11.6.4. **Sensor Measure**

```c
touch_ret_t touch_mutlcap_sensors_measure (touch_current_time_t current_time_ms, touch_acq_mode_t mutlcap_acq_mode, void(*)(void) measure_complete_callback)
```

```c
touch_ret_t touch_selfcap_sensors_measure (touch_current_time_t current_time_ms, touch_acq_mode_t selfcap_acq_mode, void(*)(void) measure_complete_callback)
```

This API can be used to start a Touch measurement.

**Parameters:**

- `current_time_ms` Current time in millisecond.
- `measure_complete_callback` Interrupt callback to indicate measurement completion.

**Returns:**

`touch_ret_t`: Touch Library Error status.
11.6.5. **Sensor Suspend and Resume**

```c
touch_ret_t touch_suspend_ptc(void)
```

```c
touch_ret_t touch_resume_ptc(void)
```

The `touch_suspend_ptc` 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 Sensor Global Parameters.

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

**Parameters:**

- `void`.

**Returns:**

- `touch_ret_t`: Touch Library Error status.

11.6.6. **Sensor Disable and Re-enable**

```c
touch_ret_t touch_mutlcap_sensor_disable (sensor_id_t sensor_id)
```

```c
touch_ret_t touch_selfcap_sensor_disable (sensor_id_t sensor_id)
```

This API can be used to disable any sensor.

**Parameters:**

- `sensor_id` Sensor number which needs to be disabled

**Returns:**

- `touch_ret_t`: Touch Library Error status.

```c
touch_ret_t touch_mutlcap_sensor_reenable (sensor_id_t sensor_id, uint8_t no_calib)
```

```c
touch_ret_t touch_selfcap_sensor_reenable (sensor_id_t sensor_id, uint8_t no_calib)
```

This API can be used to re-enable a disabled sensor.

**Parameters:**

- `sensor_id` Sensor number which needs to be reenabled
- `no_calib` When value is set to 1, force calibration is not applicable. When value is set to 0, force calibration is applied

**Returns:**

- `touch_ret_t`: Touch Library Error status.
11.6.7. **Read-back Sensor Configuration**

- `touch_ret_t touch_mutlcap_sensor_get_acq_config (touch_mutlcap_acq_param_t * p_touch_mutlcap_acq_param)`
- `touch_ret_t touch_selfcap_sensor_get_acq_config (touch_selfcap_acq_param_t * p_touch_selfcap_acq_param)`

This API can be used to read back the sensor acquisition parameters.

**Parameters:**

- `p_touch_mutlcap_acq_param` The acquisition parameters for the mutual capacitance.
- `p_touch_selfcap_acq_param` The acquisition parameters for the self-capacitance.

**Returns:**

- `touch_ret_t`: Touch Library Error status.

- `touch_ret_t touch_mutlcap_sensor_get_config (sensor_id_t sensor_id, touch_mutlcap_param_t * p_touch_sensor_param)`
- `touch_ret_t touch_selfcap_sensor_get_config (sensor_id_t sensor_id, touch_selfcap_param_t * p_touch_sensor_param)`

This API can be used to read back the sensor configuration parameters.

**Parameters:**

- `sensor_id` The sensor id for which the parameters has to be read-back.
- `p_touch_sensor_param` The sensor parameters for the mutual or self-capacitance.

**Returns:**

- `touch_ret_t`: Touch Library Error status.

- `touch_ret_t touch_mutlcap_sensor_get_delta (sensor_id_t sensor_id, touch_delta_t * p_delta)`
- `touch_ret_t touch_selfcap_sensor_get_delta (sensor_id_t sensor_id, touch_delta_t * p_delta)`

This API can be used to retrieve the delta value corresponding to a given sensor.

**Parameters:**

- `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 Error status.

- `touch_ret_t touch_mutlcap_get_global_param (touch_global_param_t * p_global_param)`
- `touch_ret_t touch_selfcap_get_global_param (touch_global_param_t * p_global_param)`

This API can be used to read back the global parameter.

**Parameters:**

- `p_global_param` The pointer to global sensor configuration.
Returns:

`touch_ret_t`: Touch Library Error status.

### 11.6.8. Update Sensor Configuration

```c
#include "touchlib.h"

// Update sensor acquisition parameters

// Mutual capacitance sensor
touch_ret_t touch_mutlcap_sensor_update_acq_config (touch_mutlcap_acq_param_t *p_touch_mutlcap_acq_param)

// Self-capacitance sensor
touch_ret_t touch_selfcap_sensor_update_acq_config (touch_selfcap_acq_param_t * p_touch_selfcap_acq_param)
```

This API can be used to update the sensor acquisition parameters.

**Parameters:**

- `p_touch_mutlcap_acq_param` The acquisition parameters for the mutual capacitance.
- `p_touch_selfcap_acq_param` The acquisition parameters for the self-capacitance.

**Returns:**

- `touch_ret_t`: Touch Library Error status.

### 11.6.9. Get Library Information and Version

```c
#include "touchlib.h"

// Get library information and version

// Mutual capacitance sensor
touch_ret_t touch_mutlcap_get_libinfo (touch_info_t *p_touch_info)

// Self-capacitance sensor
touch_ret_t touch_selfcap_get_libinfo (touch_info_t *p_touch_info)
```

This API can be used to get the library information and version.

**Parameters:**

- `p_touch_info` The pointer to library information.

**Returns:**

- `touch_ret_t`: Touch Library Error status.
This API can be used to get the Touch Library configuration.

**Parameters:**

p_touch_info Pointer to the Touch info data structure that will be updated by the Touch Library.

**Returns:**

touch_ret_t: Touch Library Error status.

```
touch_ret_t touch_library_get_version_info (touch_libver_info_t * p_touch_libver_info)
```

This API can be used to get the Touch Library version information.

**Parameters:**

p_touch_libver_info Pointer to the Touch Library Version info data structure that will be updated by the Touch Library.

---

### 11.6.10. Moisture Tolerance API

```
touch_ret_t touch_mutlcap_cnfg_mois_mltchgrp(sensor_id_t snsr_id, moisture_grp_t mois_grpid, mltch_grp_t mltch_grpid)
```

```
touch_ret_t touch_selfcap_cnfg_mois_mltchgrp(sensor_id_t snsr_id, moisture_grp_t mois_grpid, mltch_grp_t mltch_grpid)
```

This API can be used to assign moisture group and multi touch group for a sensor.

**Parameters:**

snsr_id - sensor ID
mois_grpid - moisture group ID
mltch_grp_t - multi-touch group

**Returns:**

touch_ret_t: Touch Library Error status.

```
touch_ret_t touch_mutlcap_cnfg_mois_threshold(moisture_grp_t mois_grpid, mois_snsr_threshold_t snsr_threshold, mois_system_threshold_t system_threshold)
```

```
touch_ret_t touch_selfcap_cnfg_mois_threshold(moisture_grp_t mois_grpid, mois_snsr_threshold_t snsr_threshold, mois_system_threshold_t system_threshold)
```

This API is used to assign moisture sensor threshold and moisture system threshold to a moisture group ID.

**Parameters:**

mois_grpid - moisture group ID
snsr_threshold - moisture sensor threshold
system_threshold - moisture system threshold

**Returns:**
touch_ret_t: Touch Library Error status.

touch_ret_t touch_mutlcap_mois_tolrnce_enable(void)
touch_ret_t touch_selfcap_mois_tolrnce_enable(void)

This API is used to enable moisture tolerance check during run time.

Returns:
touch_ret_t: Touch Library Error status.

ntouch_ret_t touch_mutlcap_mois_tolrnce_quick_reburst_enable(void)
touch_ret_t touch_selfcap_mois_tolrnce_quick_reburst_enable(void)

This API is used to enable moisture tolerance quick re-burst feature during run time.

Returns:
touch_ret_t: Touch Library Error status.

touch_ret_t touch_mutlcap_mois_tolrnce_disable(void)
touch_ret_t touch_selfcap_mois_tolrnce_disable(void)

This API is used to disable moisture tolerance check during run time.

Returns:
touch_ret_t: Touch Library Error status.

touch_ret_t touch_mutlcap_mois_tolrnce_quick_reburst_disable(void)
touch_ret_t touch_selfcap_mois_tolrnce_quick_reburst_disable(void)

This API is used to disable moisture tolerance quick re-burst feature during run time.

Returns:
touch_ret_t: Touch Library Error status.
## 12. Revision History

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Rev.M     | 07/2016 | 1. Updated the latest software version numbers in Section 1  
2. Added a new errata in Section 9 |
| Rev.L     | 04/2016 | Updated Sections 1, 5, and 8 with reference to the latest extension release |
| Rev.K     | 02/2016 | Added ATmega324PB support. Updated Sections 1, 5 and 8 with reference to the latest extension release |
| Rev.J     | 01/2016 | Included the following new sections:  
1. Compensation Circuit  
2. Using Atmel ICE for Qdebug Data Streaming  
3. Application flow for megaAVR  
Updated Sections 5 and 8 with reference to the latest extension release |
| Rev.I     | 09/2015 | Included Charge share delay  
Updated Section 5.2.8 and 5.2.10 - Library parameters for quick re-burst and moisture parameters added  
Updated Section 11.6.8 - Moisture API's Added  
Updated section 8 - Example projects updated |
| Rev.H     | 06/2015 | Revised Section 2 - Device Variants Supported and included information on device multiplexing option  
Updated Section 7.2 - Code and data memory considerations  
Updated Section 5.2.1 - Pin, Channel, and Sensor Parameters |
| Rev.G     | 04/2015 | Updated Section 2 - Device Variants Supported and included information on device multiplexing option |
| Rev.F     | 02/2015 | Included relevant information regarding low-power and lumped mode support |
| Rev.E     | 11/2014 | Included Section 5.2.6 and 5.2.7 regarding noise counter measures. Included Section 3 regarding overview of capacitive touch technology. |
| Rev.D     | 02/2014 | Global updates across the document related to QTouch Library and QTouch Composer 5.3 |
| Rev.C     | 10/2013 | Included Section 3.3.4, Using QDebug Touch Data Debug Communication  
Included a note on interrupt handler for IAR example project in Section 3.3.3 |
<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rev.B</td>
<td>10/2013</td>
<td>Updated errata in Section 4, Known Issues</td>
</tr>
<tr>
<td>Rev.A</td>
<td>09/2013</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>