Atmel QTouch Library

User Guide

Supports QTouch® and QMatrix® acquisition for Keys, Sliders and Rotors

Rev. 8207L-AT42-05/12

Table of Contents

1 Preface

This manual contains information that enables customers to implement capacitive touch solutions on ATMEL AVR® microcontrollers and ARM®-based AT91SAM microcontrollers using ATMEL QTouch libraries. This guide is a functional description of the library software, its programming interface and it also describes its use on the supported reference systems.

Use of this software is bound by the Software License Agreement included with the Library. This user guide is applicable for Atmel QTouch® Library 5.0 .

Related documents from ATMEL

Documents related to QTouch capacitive sensing solutions from ATMEL are

- TS2080A/B data sheet.
- QT600 users guide
- Release Notes for ATMEL QTouch libraries.
- A library selection excel workbook that is used for the selection of the appropriate library variant from the package available under in the install directory. The default location is C:\Program Files\Atmel\Atmel_QTouch_Libraries_5.x\
- Capacitive touch sensor design guide http://www.atmel.com/dyn/resources/prod_documents/doc10620.pdf .

If you need Assistance

For assistance with QTouch capacitive sensing software libraries and related issues, contact your local ATMEL sales representative or send an email to *touch@atmel.com for AVR libraries and at91support@atmel.com for SAM libraries.*

2 Introduction

ATMEL QTouch Library is a royalty free software library (available for GCC and IAR compiler tool chains) for developing touch applications on standard AVR and SAM microcontrollers. Customers can link the library into their applications in order to provide touch sensing capability in their projects. The Library can be used to develop single chip solutions for control applications which have touch sensing capabilities, or to develop standalone touch sensing solutions which interface with other host or control devices.

Features of ATMEL QTouch Library include

- Capacitive touch sensing using patented charge-transfer signal acquisition for robust sensing.
- Support for a wide range of 8- and 32-bit AVRs.
- Support for 32-bit ARM microcontrollers.
- Support for 8-bit tiny AVRs having flash of 2K bytes.
- Support both QTouch and QMatrix acquisition methods and autonomous touch for UC3L.
- Support up to 64 touch sense channels for generic libraries and up to 136 channels for UC3L libraries.
- Flexible choice of touch sensing functionality (keys, sliders, wheels) in a variety of combinations.
- Includes Adjacent Key Suppression® (AKS®) technology for the unambiguous detection of key events.
- Support for both IAR and GCC compiler tool chains.
- A comparison of various features and parameters between QTouch Libraries for Generic 8-bit and 32-bit AVRs as well as Device Specific Libraries is provided in the table below.

Feature Comparison between Generic QTouch Libraries and Device Specific Libraries

Parameter/Functionality	**Generic Libraries, Tiny_Mega_Xmega**	**Tiny 2K Libraries**	**Tiny20 Libraries**	**Tiny40 Libraries**	**Generic Libraries, 32 Bit AVR**	**UC3L Libraries**	**ATSAM Libraries**
Technology	QTouch, QMatrix	QTouch	QTouch-ADC	QTouch-ADC	QTouch, QMatrix	QTouch, QMatrix	QTouch
Rotors/Sliders Support	Yes	No	No	No	Yes	Yes	Yes
Filter Callback	Yes	Yes	No	Yes	Yes	Yes	Yes
Library Status Flags	Yes	Yes	No	Yes(Only Burst Again Flag)	Yes	Yes	Yes
Library Signature	Yes	No	No	No	Yes	Yes	Yes
Calibrate Sensing	Yes	Yes (Only burst_again flag)	No	Yes	Yes	Yes	Yes
Reset Sensing	Yes	Yes	No	Yes	Yes	Yes	Yes
Sensor Deltas	Yes	Yes	No	Yes	Yes	Yes	Yes
Maximum AKS Groups	7	7	1	7	7	7	7

Maximum Channels, QT	16	4	5	12	32	17	32
Maximum Rotors/Sliders, QT	4	0	0	0	8		8
Maximum Channels, QM	64	0	0	0	64	64	0
Maximum Rotors/Sliders, QM	8	0	0	0	8		0
Autonomous Touch	No	No	No	No	No	Yes	No
Sensor Reconfiguration	Yes	Yes	No	No	Yes	Yes	Yes
Frequency Hopping SS Enabled	Always	If _POWER_OPTIMIZATION = 0	Never	Never	Always	Programmable	Always
Delay Cycles Parameter	QT_DELAY_CLCYES (QT Values: 1 to 255 QM Values: 1,2,3,4,5,10,25,50)	QT_DELAY_CLCYES (Value: 1 to 255)	DEF_CHARGE_SHARE_DELAY (Value: 1 to 255)	DEF_QT_CHARGE_SHARE_DELAY (Value: 1 to 255)	QT_DELAY_CYCLES (QT Values : 1 to 255 QM Values: 1,2,3,4,5,10,25,50)	xx_CHLEN, xx_SELEN (QT/QM Value: 3 to 255)	QT_DELAY_CLCYES (Value: 3 to 255)
Debug Interface Enable Macro	_DEBUG_INTERFACE_	None	NDEBUG	_DEBUG_QTOUCH_STUDIO_	_DEBUG_INTERFACE_	DEF_TOUCH_QDEBUG_ENABLE	_DEBUG_INTERFACE_

This user guide describes the content, design and use of the QTouch Libraries. This should be read in conjunction with all of the applicable documents listed below

- Device datasheet for the selected ATMEL device used for touch sensing.
- Data sheet for the selected evaluation board.
- A library selection guide that is used for the selection of the appropriate library from the released package. Default path: *C:\ Program Files\Atmel\Atmel_QTouch_Libaries_5.x\Library_Selection_Guide.xls*

The intended readers of this document are engineers, who use the QTouch Library on ATMEL microcontrollers to realize capacitive touch sensing solutions.

3 Overview

This chapter gives a brief introduction to each of the chapters that make up this document

1. **Preface**
2. **Introduction**: Provides an introduction to the scope and use of the QTouch Library.
3. **Overview**: This chapter
4. **Abbreviations and Definitions**: Provides a description of the abbreviations and definitions used in this document
5. **Generic QTouch Libraries**: Provides an overview of the QTouch libraries and the different acquisition methods for generic ATMEL devices.
6. **Device Specific Libraries:** Provides an overview of the QTouch libraries and the different acquisition methods for ATMEL devices specific for touch sensing.
7. **Revision History**: Provides a revision history of this document

4 Abbreviations and Definitions

4.1 Definitions

- **AVR:** refers to a device(s) in the tinyAVR®, megaAVR®, XMEGA™ and UC3 microcontroller family.
- **ARM: refers to a device in the ATSAM ARM® basedmicrocontroller family.**
- **ATMEL QTouch Library**: The combination of libraries for both touch sensing acquisition methods (**QTouch and QMatrix).**
- **QTouch Technology**: A type of capacitive touch sensing technology using self capacitance - each channel has only one electrode.
- **QMatrix Technology**: A type of capacitive touch sensing technology using mutual capacitance – each channel has an drive electrode (X) and an receive electrode (Y).
- **Sensor**: A channel or group of channels used to form a touch sensor. Sensors are of 3 types (keys, rotors or sliders).
- **KEY**: a single channel forms a single KEY type sensor, also known as a BUTTON
- **ROTOR**, also known as a WHEEL, a group of channels forms a ROTOR sensor to detect angular position of touch.
  - A Rotor is composed of 3 channels for a QTouch acquisition method.
  - A Rotor can be composed of 3 to 8 channels for QMatrix acquisition method.
- **SLIDER**, a group of channels forms a SLIDER sensor to detect the linear position of touch.
  - A Slider is composed of 3 channels for a QTouch acquisition method.
  - A Slider can be composed of 3 to 8 channels for QMatrix acquisition method.
- **AKS**: Adjacent Key Suppression. See Section 5.4.5
- **SNS PIN**: Sense line for capacitive measurement using the QTouch Technology - connected to Cs.
- **SNSK PIN**: Sense Key line for capacitive measurement using the QTouch Technology - connected to channel electrode through Rs.
- **X Line**: The drive electrode (or drive line) used for QMatrix Technology.
- **Y Line**: The receive electrode (or receive line) used for QMatrix Technology.
- **Port Pair**: A combination of SNS port and SNSK port to which sensors are connected for QTouch technology. The SNS and SNSK ports used in a port pair can be located in the same AVR Port (8 pins for 4 sensors), or they may be in different 2 different AVR Ports (8+8 pins for 8 sensors).
- **Charge Cycle Period**: It is the width of the charging pulse applied to the channel capacitor.
- **Dwell Cycle**: In a QMatrix acquisition method, the duration in which charge coupled from X to Y is captured.
- **Acquisition**: A single capacitive measurement process.
- **Electrode**: Electrodes are typically areas of copper on a printed circuit board but can also be areas of clear conductive indium tin oxide (ITO) on a glass or plastic touch screen.

- **Intra-port:** A configuration for QTouch acquisition method libraries, when the sensor SNS and SNSK pins are available on the same port.
- **Inter-port:** A configuration for QTouch acquisition method libraries, when the sensor SNS and SNSK pins are available on distinct ports.

5 Generic QTouch Libraries

5.1 Introduction

ATMEL QTouch provides a simple to use solution to realize touch sensing solutions on a range of supported ATMEL AVR Microcontrollers. The QTouch libraries provide support for both QTouch and QMatrix acquisition methods.

Touch sensing using QMatrix or QTouch acquisition methods can be added to an application by linking the appropriate ATMEL QTouch Library for the AVR Microcontroller and using a simple set of API to define the touch channels and sensors and then calling the touch sensing API's periodically (or based on application needs) to retrieve the channel information and determine touch sensor states.

Figure 5-1 shows a typical configuration of channels when using an AVR and using the ATMEL QTouch Library. The ATMEL QTouch Library has been added to a host application running on an AVR microcontroller. The sample configuration illustrates using the library that supports eight touch channels numbered 0 to 7. The sensors are configured in the following order,

- Sensor 0 on channels 0 to 2 have been configured as a rotor sensor.
- Sensor 1 on channels 3 to 5 have been configured as a slider sensor.
- Sensor 2 on channel 6 is configured as key sensor.
- Sensor 3 on channel 7 is configured as key sensor.

The host application uses the QTouch Library API's to configure these channels and sensors, and to initiate detection of a touch using capacitive measurements.

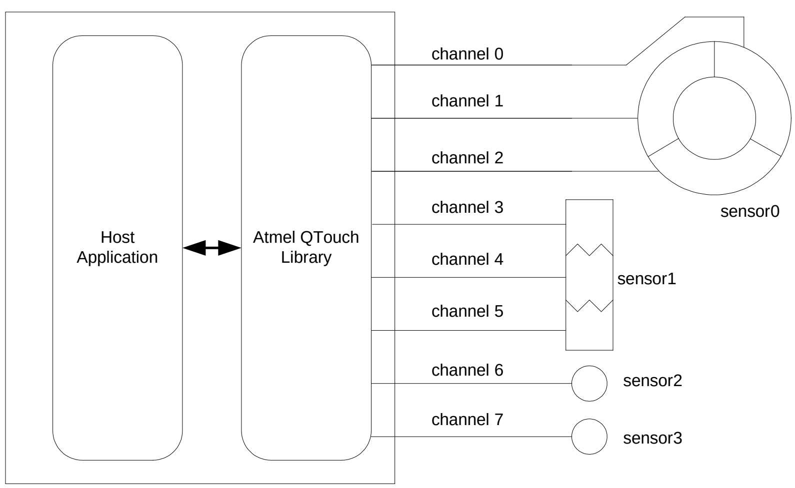


Figure 5-1 : Typical interface of the ATMEL QTouch library with the host application.

The QTouch libraries use minimal resources of the microcontroller. The sampling of the sensors is controlled by the QTouch library, while the sampling period is controlled by the application (possibly using timers, sleep periods, varying the CPU clock, external events like interrupts or communications, etc).

5.2 Acquisition Methods

There are two methods available for touch acquisition namely

1. QTouch acquisition method.
2. QMatrix acquisition method.

Libraries for AVR microcontrollers include both acquisition methods. Libraries for ATSAM microcontrollers include only QTouch acquisition method.

5.2.1 QTouch acquisition method

The QTouch acquisition method charges an electrode of unknown capacitance to a known potential. The resulting charge is transferred into a measurement capacitor (Cs). The cycle is repeated until the voltage across Cs reaches a voltage Vih. The signal level is the number of charge transfer cycles it took to reach that voltage. Placing a finger on the touch surface introduces external capacitance that increases the amount of charge transferred each cycle, reducing the total number of cycles required for Cs to reach the voltage. When the signal level (number of cycles) goes below the present threshold, then the sensor is reported to be in detected.

QTouch acquisition method sensors can drive single or multiple keys. Where multiple keys are used, each key can be set for an individual sensitivity level. Keys of different sizes and shapes can be used to meet both functional and aesthetic requirements.

NOTE: It is recommended to keep the size of the keys larger than 6mmx6mm to ensure reliable and robust measurements, although actual key design requirements also depend on panel thickness and material. Refer to the ATMEL Capacitive touch sensor design guide for details.

QTouch acquisition method can be used in two ways

- normal touch contact (i.e. when pressing buttons on a panel), and
- high sensitivity proximity mode (i.e. when a panel lights up before you actually contact it).

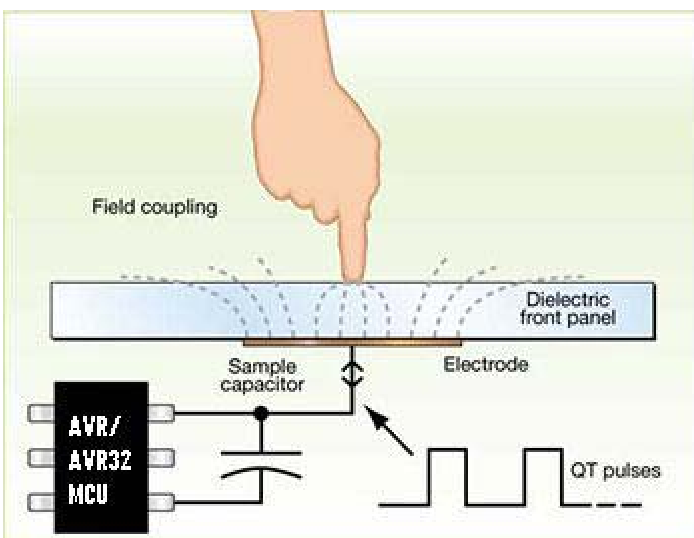
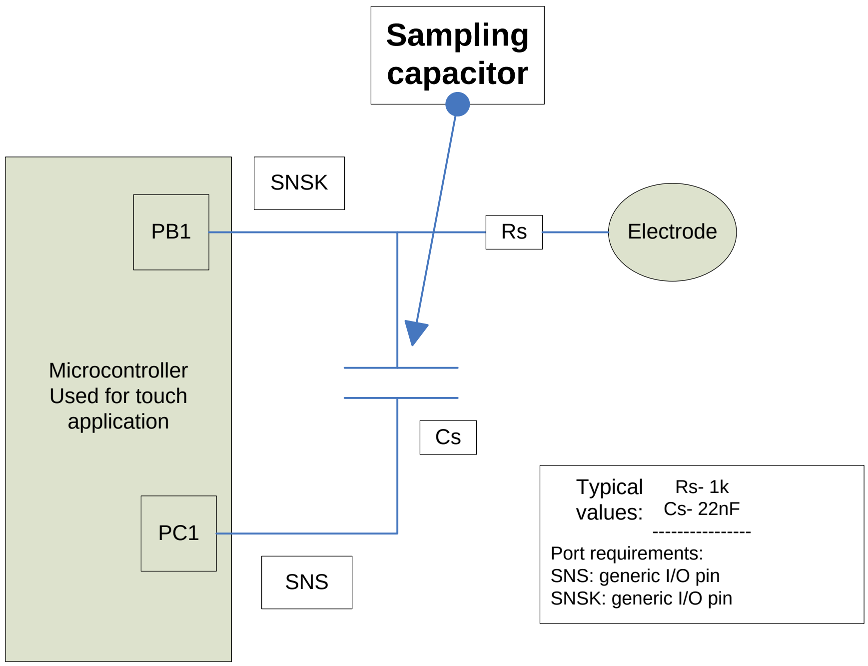


Figure 5-2 : QTouch Acquisition

QTouch circuits offers high signal-to-noise ratio, very good low power performance, and the easiest sensor layout.

5.2.1.1 Sensor schematics for a QTouch acquisition method design



Rs- Series resistor, Cs – Sample capacitor, PB1- PortB bit1, and PC1- PortC bit1

Figure 5-3 : Schematics for a QTouch acquisition method design

5.2.2 QMatrix acquisition method

QMatrix devices detect touch using a scanned passive matrix of electrode sets. A single QMatrix device can drive a large number of keys, enabling a very low cost-per-key to be achieved.

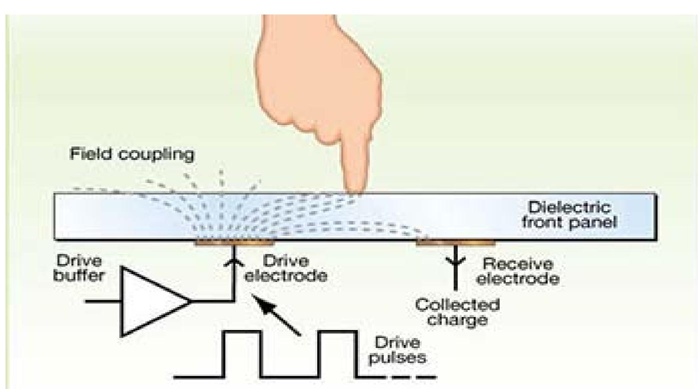


Figure 5-4 : QMatrix Acquisition method

QMatrix uses a pair of sensing electrodes for each channel. One is an emitting electrode into which a charge consisting of logic pulses is driven in burst mode. The other is a receive electrode that couples to the emitter via the overlying panel dielectric. When a finger touches the panel the field coupling is changed, and touch is detected. The drive electrode (or drive line) used for QMatrix charge transfer is labeled as the X line. The receiver electrode (or receive line) used for QMatrix charge transfer is labeled as the Y line.

QMatrix circuits offer good immunity to moisture films, extreme levels of temperature stability, superb low power characteristics, and small IC package sizes for a given key count.

5.2.3 Sensor schematics for a QMatrix acquisition method design

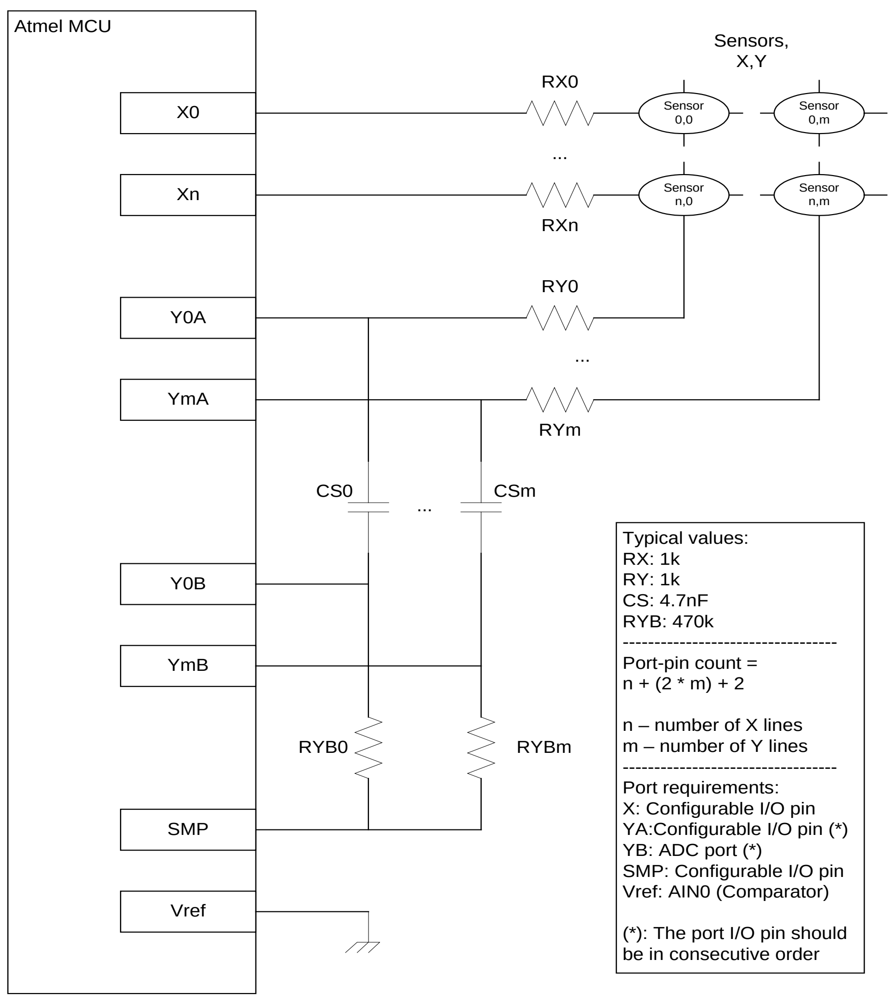


Figure 5-5 : Schematics for a QMatrix acquisition method design

5.3 Global settings common to all sensors of a specific acquisition method

The touch sensing using QTouch library could be fine tuned by using a number of configurable settings. This section explains the settings that are common to all sensors of a specific acquisition method like QMatrix or QTouch.

For example, if recalibration threshold (one of the global settings) of QMatrix acquisition method is set as 1, all QMatrix sensors will have recalibration threshold of 1.

5.3.1 Recalibration Threshold

Recalibration threshold is the level above 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 say, 40 and the Recalibration threshold value is set to 4. This implies an absolute value of 2 (40 * 6.25% = 2.5), but this is hard limited to 4.

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Recalibration threshold	qt_recal_threshold	uint8_t	Enum	4	Detect threshold	1

5.3.2 Detect Integration

The QTouch Library features a detect integration mechanism, which acts to confirm detection in a robust fashion. The detect integrator (DI) acts as a simple signal filter to suppress false detections caused by spurious events like 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 not seen to exceed 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, then the device has to exceed its threshold and stay there for 10 acquisitions in succession without going below the threshold level, before the sensor is declared to be touched.

Setting	Variable name	Data Type	Unit	Min	Max	Typical
DI	qt_di	uint8_t	Cycles	0	255	4

5.3.3 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	Variable name	Data Type	Unit	Min	Max	Typical
Drift hold time	qt_drift_hold_time	uint8_t	200 ms	1	255	20 (4s)

5.3.4 Maximum 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, perhaps after a time delay of some 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).

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Maximum ON Duration	qt_max_on_duration	uint8_t	200 ms	0	255	30 (6s)

5.3.5 Positive / Negative Drift

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 decrease when an object (touch) is approaching it or a characteristic change of sensor over time and temperature. Decreasing signals should not be compensated for quickly, as an approaching finger could be compensated for partially or entirely before even touching the channel (negative 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 (positive drift).

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Negative Drift	qt_neg_drift_rate	uint8_t	200 ms	1	127	20 (4s)
Positive Drift	qt_pos_drift_rate	uint8_t	200 ms	1	127	5 (1s)

5.3.6 Positive Recalibration Delay

If any key is found to have a significant drop in signal delta, (on the negative side), it is deemed to be an error condition. If this condition persists for more than the positive 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 positive 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 positive recalibration threshold level, the counter is cleared and the positive drifting is performed.

For example, if the PRD value 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.

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Positive Recalibration Delay	qt_pos_recal_delay	uint8_t	cycles	1	255	3

5.4 Sensor specific settings

Apart from global settings as mentioned in the section above, touch sensing using QTouch library could also be fine tuned by more number of configurable settings.

This section explains the settings that are specific to each sensor. For example, sensor 0 can have a detect threshold (one of the sensor specific setting) that is different from sensor 1.

5.4.1 Detect threshold

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

Threshold setting depends on the amount of signal swing that occurs when a sensor is touched. Thicker front panels or smaller electrodes usually have smaller signal swing on touch, thus require lower threshold levels.

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Threshold	threshold	uint8_t	counts	3	255	10 – 20

5.4.2 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)

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Hysteresis	detect_hysteresis	uint8_t (2 bits)	Enum	HYST_6_25	HYST_50	HYST_6_25

5.4.3 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 2bits to 8 bits.

Setting	Variable name	Data Type	Unit	Min	Reported position	Max	Reported position	Typical
Position Resolution	position_ resolution	uint8_t (3 bits)	-	2 bits	0 – 3	8 bits	0-255	8

5.4.4 Position Hysteresis

In case of QMatrix, 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 the touch down.

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 ..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.
Ex: do not have a hysteresis value of 5 positions with a resolution of 2 bits (4 positions).

Setting	Variable name	Data Type	Unit	Min	Max	Typical
Position Hysteresis	position_hysteresis	uint8_t (3 bits)	-	0	7	3

NOTE:
Position hysteresis is not valid (unused) in case of QTouch acquisition method libraries.

5.4.5 Adjacent Key Suppression (AKS)

In designs where the sensors are close together or set for high sensitivity, multiple sensors might report detect simultaneously if touch is near them. 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, then 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 in detect until its delta falls below its detection threshold, and then if any more sensors in the AKS group are still in detect then the strongest will report detection. So at any given time only one sensor from each AKS group will be reported to be in detect.

The library provides the ability to configure any sensor to be included in any one of the Adjacent Key Suppression Groups (AKS Group).

Setting	Variable name	Data Type	Unit	Min	Max	Typical
AKS Group	aks_group	uint8_t (3 bits)	Enum	0 (off)	7	0 (off)

5.5 Using the Sensors

5.5.1 Avoiding Cross-talk

In ATMEL QTouch library variants that use QTouch acquisition technology, adjacent sensors are not measured at the same time. This prevents interference due to cross-talk between adjacent channels, but at the same time some sensor configurations take longer to measure than others. For example, if an 8-channel device is configured to support 8 keys, then the library will measure the keys on channels 0, 2, 4, and 6 parallely, followed by keys on channels 1, 3, 5, and 7. If the same device is configured, say, to support 4 keys, putting them either on all the odd channels or on all the even channels means that they can all be measured simultaneously.

This means the library calls are faster, and the device can use less power. So, it is recommended that the appropriate channel numbers are used when using less than the maximum number of channels available for the device to ensure optimum performance. In a similar sense for faster execution and reduced power consumption, it is also advisable to use intra-port sensor configuration instead of inter-port sensor configuration while using 4 channels on the same port.

5.5.2 Multiple measurements

The library will not automatically perform multiple measurements on a sensor (Ex: To resolve for instance Detect Integration or recalibration.). The user is given the option to perform the measurement multiple times if certain conditions are met. This will enable the user to implement the time critical code thereby making the qt_measure_sensors() a non-blocking API .The host application has to perform multiple measurements, based on the need. The global flag QTLIB_BURST_AGAIN indicating that multiple measurements are needed is passed to the user. This is BIT8 of the return value from the qt_measure_sensors( ) API. The main_<devicename>.c has the example usage to perform multiple measurements.

If QTLIB_BURST_AGAIN = 1, multiple measurements are needed to

- To compensate for drift
- Resolve re-calibration
- Resolve calibration.
- Resolve detect integration.

If QTLIB_BURST_AGAIN = 0, multiple measurements are not needed and the user can execute the host application code. Apart from QTLIB_BURST_AGAIN, various flags are provided to the user to perform the multiple measurements based on the need of the host application to act to specific situation. Description of the these flags can be found in the section5.6.5.6

Note: To maintain robustness and timing of the touch sensing measurement, it is recommended that the user calls the qt_measure_sensors() immediately if the flag QT_BURST_AGAIN=1. However, the user is allowed to run time- critical section (not more than few instructions) of the host application comprising on the touch sensing timing.

5.5.3 Guard Channel

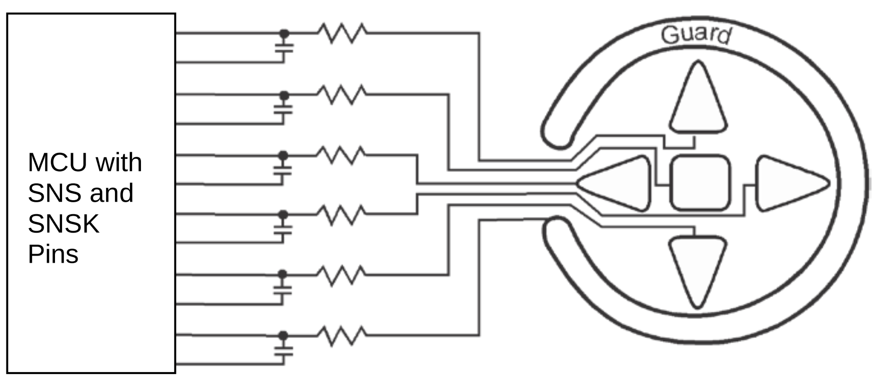
Guard channel in Qtouch Acquisition Method allows one key to be configured as a guard channel to help prevent false detection. Guard channel keys should be more sensitive than the other keys (physically bigger or larger Cs).To enable key as guard channel, the designated key is connected to a sensor pad which detects the presence of touch and overrides any output from the other keys using the AKS feature.

The key can be configured to have a guard channel function by adjusting a number of independent settings. The Guard channel is designed so that it is likely to be activated unless a key is accurately touched.

The guard channel sensor must be set up so that it is slightly more sensitive than the keys that it surrounds. The exact amount of increase depends on the application and is best determined by experimentation.
There are three methods of increasing the sensor sensitivity that can be used in combination:

1. Increasing the size of the sensor.
2. Increasing the value of the Sample Capacitor (Cs).
3. Adjust the detection threshold for the sensor.

The sensor size and capacitor values should be altered to establish the base sensitivity for the sensor. Once these values have been established, the detection threshold can be used to fine tune the sensor.



The Above figure illustrates how a Guard sensor/key is to be visualized.It has six keys and five keys are surrounded by a Guard Channel.

Please refer QTAN0031 for further information on Guard Channel.
http://www.atmel.com/dyn/resources/prod_documents/QTAN0031(2).pdf

5.6 QTouch API and Usage

The Atmel QTouch library provides support for many devices. This chapter explains the touch library for such devices without any hardware support.

5.6.1 QTouch Library API

This section describes the QTouch library Application Programming Interface (API) for touch sensing using QTouch and QMatrix acquisition methods.

Using the API, Touch sensors and the associated channels can be defined. Once touch sensing has been initiated by the user, the host application can use the API to make touch measurements and determine the status of the sensors.

5.6.2 touch_api.h - public header file

The *touch_api.h* header file is the public header file which needs to be included in users application and it has the type definitions and function prototypes of the API's listed in sections 5.6.3 , 5.6.4 and 5.6.5

The touch_api.h header file is located in the library distribution in the following directory.

- *..\Atmel_QTouch_Libraries_5.x\Generic_QTouch_Libraries\include*

5.6.3 Type Definitions and enumerations used in the library

5.6.3.1 Typedefs

This section lists the type definitions used in the library.

Typedef	Notes
uint8_t	unsigned 8-bit integer
int8_t	signed 8-bit integer
uint16_t	unsigned 16-bit integer
int16_t	signed 16-bit integer
uint32_t	unsigned 32-bit integer
threshold_t	unsigned 8-bit integer used for setting a sensor detection threshold

5.6.3.2 Enumerations

This section lists the enumerations used in the QTouch Library.

5.6.3.2.1 sensor_type_t

Enumeration sensor_type_t
Use Define the type of the sensor

Values	Comment
SENSOR_TYPE_UNASSIGNED	Channel is not assigned to any sensor
SENSOR_TYPE_KEY	Sensor is of type KEY
SENSOR_TYPE_ROTOR	Sensor is of type ROTOR

SENSOR_TYPE_SLIDER	Sensor is of type SLIDER

5.6.3.2.2 aks_group_t

Enumeration aks_group_t

Use Defines the Adjacent Key Suppression (AKS) groups each sensor may be associated with (see section 5.3.4 Maximum ON Duration)

AKS is selectable by the system designer
7 AKS groups are supported by the library

Values	**Comment**
NO_AKS_GROUP	NO AKS group selected for the sensor
AKS_GROUP_1	AKS Group number 1
AKS_GROUP_2	AKS Group number 2
AKS_GROUP_3	AKS Group number 3
AKS_GROUP_4	AKS Group number 4
AKS_GROUP_5	AKS Group number 5
AKS_GROUP_6	AKS Group number 6
AKS_GROUP_7	AKS Group number 7

5.6.3.2.3 channel_t

Enumeration channel_t

Use The channel numbers used in the library.

When using the QTouch acquisition method, the channel numbers have a one to one mapping to the pin numbers of the port being used.

When using the QMatrix acquisition method, the channel numbers are ordered in a matrix sequence

Values	**Comment**
CHANNEL_0	Channel number : 0
CHANNEL_1	Channel number : 1
CHANNEL_2	Channel number : 2
CHANNEL_3	Channel number : 3
.....	Channel number: ..
Upto CHANNEL (N-1)	Channel number N-1 : for an N Channel library

The maximum number of channels supported is dependent on the library variant. Possible values of N are as listed below

Acquisition method	**Device type**	**Possible values of N (Maximum number of channels)**
QTouch acquisition	8-bit	4,8,16
	32-bit	8, 16, 32
QMatrix Acquisition	8-bit	8,16,32,64

5.6.3.2.4 hysteresis_t

Enumeration Hysteresis_t

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

This is configurable per sensor.

HYST_x = hysteresis value is x percent of detection threshold value (rounded

down).
Note that a minimum value of 2 is used as a hard limit. Example: if detection threshold = 20, then:
HYST_50 = 10 (50 percent of 20)
HYST_25 = 5 (25 percent of 20)
HYST_12_5 = 2 (12.5 percent of 20)
HYST_6_25 = 2 (6.25 percent of 20 = 1, but set to the hard limit of 2)

Values	Comment
HYST_50	50% Hysteresis
HYST_25	25% Hysteresis
HYST_12_5	12.5% Hysteresis
HYST_6_25	6.25% Hysteresis

5.6.3.2.5 resolution_t

Enumeration resolution_t
Use 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.

Values	Comment
RES_1_BIT	1 bit resolution : reported positions range 0 – 1
RES_2_BIT	2 bit resolution : reported positions range 0 – 3
RES_3_BIT	3 bit resolution : reported positions range 0 – 7
RES_4_BIT	4 bit resolution : reported positions range 0 – 15
RES_5_BIT	5 bit resolution : reported positions range 0 – 31
RES_6_BIT	6 bit resolution : reported positions range 0 – 63
RES_7_BIT	7 bit resolution : reported positions range 0 – 127
RES_8_BIT	8 bit resolution : reported positions range 0 – 255

5.6.3.2.6 recal_threshold_t

Enumeration recal_threshold_t
Use A sensor recalibration threshold. This is expressed as a percentage of the sensor detection threshold.

This is for automatic recovery from false conditions, such as a calibration while sensors were touched, or a significant step change in power supply voltage.
If the false condition persists the library will recalibrate according to the settings of the recalibration threshold.

This setting is applicable to all the configured sensors.

Usage :
RECAL_x = recalibration threshold is x percent of detection threshold value (rounded down).
Note: a minimum value of 4 is used.
Example: if detection threshold = 40, then:
RECAL_100 = 40 (100 percent of 40)
RECAL_50 = 20 (50 percent of 40)
RECAL_25 = 10 (25 percent of 40)
RECAL_12_5 = 5 (12.5 percent of 40)
RECAL_6_25 = 4 (6.25 percent of 40 = 2, but value is limited to 4)

Values	Comment

RECAL_100	100% recalibration threshold
RECAL_50	50% recalibration threshold
RECAL_25	25% recalibration threshold
RECAL_12_5	12.5% recalibration threshold
RECAL_6_25	6.25% recalibration threshold

5.6.4 Data structures

This section lists the data structures that hold sensor status, settings, and diagnostics information

5.6.4.1 qt_touch_status_t

Structure qt_touch_status_t
Input / Output Output from the Library
Use Holds the status (On/ Off) of the sensors and the linear and angular positions of sliders and rotors respectively

Fields	Comment
sensor_states[]	For Sensor, the sensor_states. Bit "n" = state of nth sensor : Bit Value 0 - indicates the sensor is not in detect Bit Value 1 - indicates the sensor is in detect
rotor_slider_values[]	Rotors angles or slider positions if rotors and sliders are used. These values are valid when sensor states shows that the corresponding rotor or slider is in detect

The macro that can get the sensor state when the sensor number is provided can be something as below:

```
#define      GET_SENSOR_STATE(SENSOR_NUMBER)
      qt_measure_data.qt_touch_status.sensor_states[(SENSOR_NUMBER/8)] &
(1 << (SENSOR_NUMBER % 8))
```

The host application can use this macro to act accordingly, the following example shows how to toggle a IO pin (PD2) based on the sensor0 state.(Set PD2 if sensor0 is in detect, and clear PD2 if sensor0 is not in detect)

```
Ex:   /*Set pin PD2 direction as output*/

      DDRD |= (1u << PORTD2);

      if (GET_SENSOR_STATE(0) !=0)

      {     PORTD |= (1u << PORTD2); /* Set PORTD2 */

      }

      else {

            PORTD &= ~(1u << PORTD2); /* Clear PORTD2 */

      }
```

5.6.4.2 qt_touch_lib_config_data_t

Structure qt_touch_lib_config_data_t
Input / Output Input to the library

Use Global Configuration data settings for the library.

Fields	Type	Comment
qt_recal_threshold	recal_threshold_t	Sensor recalibration threshold. Default: RECAL_50 (recalibration threshold = 50 percent of detection threshold. Refer to section 5.3.1 Recalibration Threshold more details
qt_di	uint8_t	Sensor detect integration (DI) limit. Default value: 4. Refer to section 5.3.2 Detect Integration for more details
qt_drift_hold_time	uint8_t	Sensor drift hold time in units of 200 ms. Default value: 20 (20 x 200 ms = 4s), that is hold off drifting for 4 seconds after leaving detect. Refer to section 5.3.3 Drift Hold Time for more details
qt_max_on_duration	uint8_t	Sensor maximum on duration in units of 200 ms. For example: 150 = recalibrate after 30s (150 x 200 ms). 0 = recalibration disabled Default value: 0 (recalibration disabled). Refer to section 5.3.4 Maximum ON Duration for more details.
qt_neg_drift_rate	uint8_t	Sensor negative drift rate in units of 200 ms. Default value: 20 (20 x 200 ms = 4s per LSB). Refer to section 5.3.5 Positive / Negative Drift for more details
qt_pos_drift_rate	uint8_t	Sensor positive drift rate in units of 200 ms. Default value: 5 (5 x 200 ms = 1s per LSB). Refer to section 5.3.5 Positive / Negative Drift for more details
qt_pos_recal_delay	uint8_t	Sensor positive recalibration delay. Default: 3. Refer to section 5.3.6 for more details.

The measurement limit for touch sensing using QTouch acquisition method is hard coded as 8192.

The QTouch library exports a variable of this type so that the user can specify the threshold parameters for the library. The API *qt_set_parameters() should be called to apply the parameters specified.*

```
extern  qt_touch_lib_config_data_t qt_config_data;
```

5.6.4.3 qt_touch_lib_measure_data_t

Structure qt_touch_lib_measure_data_t
Input / Output Output from the library
Use Data structure which holds the sensor and channel states and values.

Fields	Type	Comment
channel_signals	uint16_t	The measured signal on each channel.
channel_references	uint16_t	The reference signal for each channel.
qt_touch_status	qt_touch_status_t	The state and position of the configured sensors

The QTouch library exports a variable of this type which can be accessed to retrieve the touch status of all the sensors.

```
extern qt_touch_lib_measure_data_t  qt_measure_data;
```

5.6.4.4 qt_burst_lengths

Structure qt_burst_lengths
Input / Output Input to the library
Use **NOTE: Applicable only to the QMatrix acquisition method libraries**

This data structure is used to specify the burst lengths for each of the QMatrix channels

Fields	Type	Comment
qt_burst_lengths[]	uint8_t	The burst length for each of the QMatrix channel in units of pulses. Default value: 64 pulses. These values can be configured for each channel individually.

The signal gain for each sensor is controlled by circuit parameters as well as the burst length. The burst length is simply the number of times the charge-transfer (‘QT’) process is performed on a given sensor. Each QT process is simply the pulsing of an X line once, with a corresponding Y line enabled to capture the resulting charge passed through the sensor’s capacitance Cx.

The QMatrix acquisition method library exports a variable of this type which can be accessed to set the burst length for each of the QMatrix channels

```
extern uint8_t qt_burst_lengths[QT_NUM_CHANNELS];
```

5.6.4.5 tag_sensor_t

Structure tag_sensor_t
Input / Output Output from the library
Use Data structure which holds the internal sensor state variables used by the library.

<table>
<tr><th>Fields</th><th>Type</th><th colspan="2">Comment</th></tr>
<tr><td>State</td><td>uint8_t</td><td colspan="2">internal sensor state</td></tr>
<tr><td>general_counter</td><td>uint8_t</td><td colspan="2">general purpose counter: used for calibration, drifting, etc</td></tr>
<tr><td>ndil_counter</td><td>uint8_t</td><td colspan="2">drift Integration counter</td></tr>
<tr><td>Threshold</td><td>uint8_t</td><td colspan="2">sensor detection threshold. Refer to section 5.4.1 Detect threshold for more details</td></tr>
<tr><td rowspan="6">type_aks_pos_hyst</td><td rowspan="6">uint8_t</td><td colspan="2">holds information for sensor type, AKS group, positive recalibration flag, and hysteresis value</td></tr>
<tr><td>Bit fields</td><td>Use</td></tr>
<tr><td>B1 : B0</td><td>Hysteresis. Refer to section 5.4.2 Hysteresis for more details</td></tr>
<tr><td>B2</td><td>positive recalibration flag</td></tr>
<tr><td>B5:B3</td><td>AKS group. Refer to section 5.4.5 for more details</td></tr>
<tr><td>B7:B6</td><td>sensor type</td></tr>
<tr><td>from_channel</td><td>uint8_t</td><td colspan="2">starting channel number for sensor</td></tr>
<tr><td>to_channel</td><td>uint8_t</td><td colspan="2">ending channel number for sensor</td></tr>
<tr><td>Index</td><td>uint8_t</td><td colspan="2">index for array of rotor/slider values</td></tr>
</table>

5.6.4.6 qt_lib_siginfo_t

Structure qt_lib_siginfo_t
Input / Output Output from the library
Use Data structure which holds the information about the library variant and its version information.

<table>
<tr><th colspan="4">qt_lib_siginfo_t structure definition for a QTouch acquisition method library variant</th></tr>
<tr><td>Fields</td><td>Type</td><td colspan="2">Comment</td></tr>
<tr><td rowspan="5">library_version</td><td rowspan="5">uint16_t</td><td colspan="2">Holds the library version information.</td></tr>
<tr><td>Bit fields</td><td>Use</td></tr>
<tr><td>B3 : B0</td><td>Patch version of the library</td></tr>
<tr><td>B7 : B4</td><td>Minor version of the library</td></tr>
<tr><td>B15:B8</td><td>Major version of the library</td></tr>
</table>

lib_sig_lword	uint16_t	Holds the general information about the library	
		Bit fields	**Use**
		B1 : B0	Library Type : 00 : QTouch acquisition method 01 : QMatrix acquisition method
		B2	Compiler tool chain used 0 – GCC 1 – IAR
		B9 : B3	Maximum number of channels supported by the library
		B10	0 – Library supports only keys 1 – Library supports keys and rotors
		B15 : B11	Maximum number of rotors and sliders supported by the library
lib_sig_hword	uint16_t	Reserved	

qt_lib_siginfo_t structure definitions for a QMatrix acquisition method library variant			
Fields	Type	Comment	
library_version	uint16_t	Holds the library version information.	
		Bit fields	**Use**
		B3 : B0	Patch version of the library
		B7 : B4	Minor version of the library
		B15:B8	Major version of the library
lib_sig_lword	uint16_t	Holds the general information about the library	
		Bit fields	**Use**
		B1 : B0	Library Type : 00 : QTouch acquisition method 01 : QMatrix acquisition method
		B2	Compiler tool chain used 0 – GCC 1 – IAR
		B9 : B3	Maximum number of channels supported by the library
		B10	0 – Library supports only keys 1 – Library supports keys and rotors
		B15 : B11	Maximum number of rotors and sliders supported by the library
lib_sig_hword	uint16_t	Holds information about the X and Y lines for a QMatrix library variant	
		Bit fields	Use
		B4 : B0	Number of X Lines
		B8 : B5	Number of Y Lines
		B9	0

5.6.5 Public Functions

This section lists the public functions available in the QTouch libraries and its usage.

5.6.5.1 qt_set_parameters

This function is used to initialize the global configuration settings in the variable **`qt_config_data`** of the QTouch and QMatrix acquisition method libraries.

```
void qt_set_parameters ( void )
```

Arguments	Type	Comment
Void	-	This function will initialize the parameters required by the library to default values .But the default values can be changed by the user by modifying the global threshold values as defined in *qt_touch_lib_config_data_t* . See section 0 for details.

NOTE:

- This function can be called any time to apply the threshold parameters of the library as specified by modifying the global data structure *qt_config_data* exported by the library.

5.6.5.2 qt_enable_key

This function is used to configure a channel as a key.

```
void qt_enable_key (
            channel_t         channel ,
            aks_group_t       aks_group ,
            threshold_t       detect_threshold ,
            hysteresis_t      detect_hysteresis
            )
```

Arguments	Type	Comment
Channel	channel_t	Specifies the channel number to be configured for use as a “key”
aks_group	aks_group	Specifies the aks group associated with the sensor being configured as “key”
detect_threshold	threshold_t	Specifies the detect threshold for the sensor
detect_hysteresis	hysteresis_t	Specifies the detection hysteresis for the sensor

5.6.5.3 qt_enable_rotor

This function is used to configure a set of channels as a rotor.

```
void qt_enable_rotor (
            channel_t         from_channel ,
            channel_t         to_channel ,
            aks_group_t       aks_group ,
            threshold_t       detect_threshold ,
            hysterisis_t      detect_hysterisis ,
            resoulution_t     angle_resolution ,
            uint8_t           angle_hysterisis
            )
```

Arguments	Type	Comment
from_channel	Channel_t	Specifies the starting channel number to be configured for use as a “Rotor”
to_channel	Channel_t	Specifies the end channel number to be configured for use as a “Rotor”
aks_group	aks_group	Specifies the aks group associated with the sensor being configured as “ROTOR”
detect_threshold	threshold_t	Specifies the detect threshold for the sensor
detect_hysterisis	hysterisis_t	Specifies the detection hysteresis for the sensor
angle_resolution	resolution_t	Specifies the resolution of the reported angle value
angle_hysterisis	uint8_t	Specifies the hysteresis of the reported angle value

NOTE:

- A “Rotor” sensor requires contiguous channel numbers.
- The rotor / slider number depends on the order in which the rotor or sliders are enabled. The first rotor or slider enabled will use “rotor_slider_values[0]”, the second will use “rotor_slider_values[1]”, and so on. The reported rotor value is valid when the rotor is reported as being in detect.

- In case of QMatrix acquisition method library, the *from_channel and to_channel* can be between 3 to 8 channel numbers apart (i.e. it can support 3 to 8 channel rotors).
- In case of QTouch acquisition method library, the *from_channel and to_channel* can be 3 channels apart (i.e. can support only 3 channel rotors).

5.6.5.4 qt_enable_slider

This function is used to configure a set of channels as a rotor.

```
void qt_enable_slider (
            channel_t          from_channel ,
            channel_t          to_channel ,
            aks_group_t        aks_group ,
            threshold_t        detect_threshold ,
            hysterisis_t       detect_hysterisis ,
            resolution_t       position_resolution ,
            uint8_t            position_hysteresis
            )
```

Arguments	Type	Comment
from_channel	Channel_t	Specifies the starting channel number to be configured for use as a "Slider"
to_channel	Channel_t	Specifies the end channel number to be configured for use as a "Slider"
aks_group	aks_group	Specifies the aks group associated with the sensor being configured as "Slider"
detect_threshold	threshold_t	Specifies the detect threshold for the sensor
detect_hysterisis	hysterisis_t	Specifies the detection hysteresis for the sensor
position_resolution	resolution_t	Specifies the resolution of the reported position value
position_hysterisis	uint8_t	Specifies the hysteresis of the reported position value

NOTE:

- A "Slider" sensor requires a contiguous numbers of channels.
- The rotor / slider number depends on the order in which the rotor or sliders are enabled. The first rotor or slider enabled will use "rotor_slider_values[0]", the second will use. "rotor_slider_values[1]", and so on. The reported rotor value is valid when the slider is reported as being in detect.
- In case of QMatrix acquisition method library, the *from_channel and to_channel* can be between 3 to 8 channels apart (i.e. it can support 3 to 8 channel sliders).
- In case of QTouch acquisition method library, the *from_channel and to_channel* can be 3 channels apart (i.e. can support only 3 channel sliders).

5.6.5.5 qt_init_sensing

This function is used to initialize the touch sensing for all enabled channels. All required sensors should be configured before calling this function.

```
void qt_init_sensing ( void )
```

Arguments	Type	Comment
Void	-	-

NOTE:

- All sensors must be configured (using *qt_enable_key, qt_enable_rotor or qt_enable_slider)* before calling this function.
- This functions initializes all the configured sensors, performs calibration.

5.6.5.6 qt_measure_sensors

This function performs a capacitive measurement on all enabled sensors. The measured signals for each sensor are then processed to check for user touches, releases, changes in rotor angle and changes in slider position.

```
unit16_t qt_measure_sensors( uint16_t current_time_ms  )
```

Arguments	Type	Comment
current_time_ms	uint16	The current time in milliseconds

<table>
<tr><th>Return Value</th><th colspan="3">Comment</th></tr>
<tr><td rowspan="11">uint16_t</td><td colspan="3">Returns the status of the Library as a combination of the following bit fields.</td></tr>
<tr><td>Return value</td><td>Bit definition</td><td>Comments</td></tr>
<tr><td>QTLIB_NO_ACTIVITY</td><td>0x0000</td><td>No activity detected on any of the sensors</td></tr>
<tr><td>QTLIB_IN_DETECT</td><td>0x0001</td><td>At least one sensor is in detect</td></tr>
<tr><td>QTLIB_STATUS_CHANGE</td><td>0x0002</td><td>At least one sensor has changed ON/OFF state since the last call to qt_measure_sensor()</td></tr>
<tr><td>QTLIB_ROTOR_SLIDER_POS_CHANGE</td><td>0x0004</td><td>At least one rotor/slider has changed position since the last call to qt_measure_sensors()</td></tr>
<tr><td>QTLIB_CHANNEL_REF_CHANGE</td><td>0x0008</td><td>At least one reference value has changed since last call to qt_measure_sensors()</td></tr>
<tr><td>QTLIB_BURST_AGAIN</td><td>0x0100</td><td>Flag to indicate Multiple measurements needed.</td></tr>
<tr><td>QTLIB_RESOLVE_CAL</td><td>0x0200</td><td>Multiple measurements needed to resolve calibration. Call qt_measure_sensors( ) once again.</td></tr>
<tr><td>QTLIB_RESOLVE_FILTERIN</td><td>0x0400</td><td>Multiple measurements needed to resolve filtering. Call qt_measure_sensors( ) once again.</td></tr>
<tr><td>QTLIB_RESOLVE_DI</td><td>0x0800</td><td>Multiple measurements needed to resolve detect integration. Call qt_measure_sensors( ) once again.</td></tr>
<tr><td></td><td>QTLIB_RESOLVE_POS_RECAL</td><td>0x1000</td><td>Multiple measurements needed to resolve positive recalibration. Call qt_measure_sensors( ) once again.</td></tr>
</table>

NOTE:

- All sensors must be configured (using *qt_enable_key or qt_enable_rotor or qt_enable_slider)* and initialized by calling *qt_init_sensing* before calling this function.

5.6.5.7 qt_calibrate_sensing

This function forces a recalibration of all enabled sensors.

```
void qt_calibrate_sensing( void )
```

Arguments	Type	Comment
Void	-	-

NOTE:

- Recalibration may be useful if, for example, it is desired to globally recalibrate all sensors on a change in application operating mode.
- This function must be called only when the sensors have been configured and initialized.

5.6.5.8 qt_reset_sensing

This function disables all sensors and resets all configuration settings (for example, “qt_di”) to their default values.

```
void qt_reset_sensing( void )
```

Arguments	Type	Comment
Void	-	-

NOTE:

- This may be useful if it is desired to dynamically reconfigure sensing. After calling this function, any required sensors must be re-enabled, filter callback needs to be re-initialized, and “qt_init_sensing()” must be called before “qt_measure_sensors()” is called again.
- In case of QMatrix, the burst lengths for all channels are set to zero.

5.6.5.9 qt_get_sensor_delta

This function returns the delta value for a given channel.

```
int16_t qt_get_sensor_delta( uint8_t sensor_number )
```

Arguments	Type	Comment
sensor_number	unit8_t	sensor id for which the delta is required

Return type	Comment
int16_t	The delta value of the sensor specified

NOTE:

- All sensors must be configured (using *qt_enable_key or qt_enable_rotor or qt_enable_slider)* and initialized by calling *qt_init_sensing* before calling this function.

5.6.5.10 qt_get_library_sig

This function is used to retrieve the library version and signature from the library.

```
void qt_get_library_sig( qt_lib_siginfo_t  *lib_sig_ptr )
```

Arguments	Type	Comment
lib_sig_ptr	qt_lib_siginfo_t *	Pointer to the structure which needs to be updated with the library signature information

NOTE:

- The function *qt_measure_sensors()* should have been called at least once prior to calling this function.

5.6.6 Sequence of Operations and Using the API

Figure 6 illustrates the sequence of operations required to be performed to add touch to an end application. By using the simple API's as illustrated in the sequence flowchart, the user can add touch sensing in his design.

5.6.6.1 Channel Numbering

5.6.6.1.1 Channel numbering when using QTouch acquisition method

QTouch acquisition method libraries require 2 GPIO pins per channel. QTouch libraries can be configured to use 1 to 16 channels requiring 2 to 32 pins respectively. There are two options provided for connecting the SNS and SNSK pins.

1. The SNS and SNSK pins are connected to separate ports. (i.e. Interport)
2. The SNS and SNSK pins are connected to the same port. (i.e. Intraport)

The following list provides a look at various combinations supported by various **8bit AVR libraries** released for each device.

When pin configurability is not used:

- 4-channel library – supports up to 4 channels using 4 consecutive pins on different SNS and SNSK ports (or) supports up to 4 channels using 8 consecutive pins on the same port used for both SNS and SNSK lines. This library requires 1 or 2 ports.
- 8-channel library – supports up to 8 channels using 8 consecutive pins on different SNS and SNSK ports (or) supports up to 8 channels using 16 pins spread over two ports (SNS and SNSK are on alternate pins) with SNS1 and SNSK1 pins on the first port and SNS2 and SNSK2 pins on the second port. This library requires 2 ports.
- 12-channel library (available only for 8bit AVR devices) – supports up to 12 channels out of which, 8 channels with 8 consecutive pins for SNS1 and SNSK1 are available on different ports and the other 4 channels with 8 consecutive pins available on the same port for both SNS and SNSK lines. This library requires a total of 3 ports.
- 16-channel library – supports up to 16 channels out of which, 8 channels with 8 consecutive pins for SNS1 and SNSK1 are available on different ports and the other 8 channels with 8 consecutive pins are available on a different pair of SNS2 and SNSK2 ports. This library requires a total of 4 ports.

When pin configurability is used:

- 4-channel library – supports up to 4 channels using any 4 pins on different SNS and SNSK ports (or) supports up to 4 channels using pins on the same port used for both SNS and SNSK lines. This library requires 1 or 2 ports.
- 8-channel library – supports up to 8 channels using 8 pins on different SNS and SNSK ports (or) supports up to 8 channels using pins spread over two ports (SNS and SNSK are on alternate pins) with SNS1 and SNSK1 pins on the first port and SNS2 and SNSK2 pins on the second port. This library requires 2 ports.
- 12-channel library (available only for 8bit AVR devices) – supports up to 12 channels out of which, 8 channels with 8 pins for SNS1 and SNSK1 are available on different ports and the other 4 channels with 8 pins available on the same port for both SNS and SNSK lines. This library requires a total of 3 ports.
- 16-channel library – supports up to 16 channels out of which, 8 channels with 8 pins for SNS1 and SNSK1 are available on different ports and the other 8 channels with 8 pins are available on a different pair of SNS2 and SNSK2 ports. This library requires a total of 4 ports.

Note:

- When a library supports 4 channels using 8 consecutive pins on the same port, the SNS and SNSK pins are allocated alternately. This is valid for all the libraries mentioned above.
- Usage of intraport configuration requires more code memory than the interport configuration. The values mentioned in the Library_selection_Guide.xls are for interport configurations. The memory consumption for intra-port will be higher to the values mentioned in the Library_selection_Guide.xls
- The configurations on pin configurability should be used in conjunction with the rules for assigning the pins that are described in section 5.8.2

For **UC3 and ATSAM libraries**, an n- channel library supports up to n channels using n consecutive pins on different SNS and SNSK ports (or) supports up to n/2 channels using (n) consecutive pins on the same port used for both SNS and SNSK lines. This library requires 1 or 2 UC3 or ATSAM ports. In addition to this, for the ATSAM libraries the pins can be configured on 3 ports based on the configuration selected.

NOTE:

Some of the devices in UC3 family has ports having more than 32 pins or less than 32 pins.In those devices, the mapping is given as below:

GPIO Port0 -> A
GPIO Port1 -> B
GPIO Port2 -> C
GPIO Port3 -> X

Example SNS=A and SNSK=X, So channel 0 will be (SNS0 = GPIO0_Pin0 and SNSK0 = GPIO3_Pin0).
Similarly,Example SNS=X and SNSK=X, So channel 0 will be (SNS0 = GPIO3_Pin0 and SNSK0 = GPIO3_Pin1).

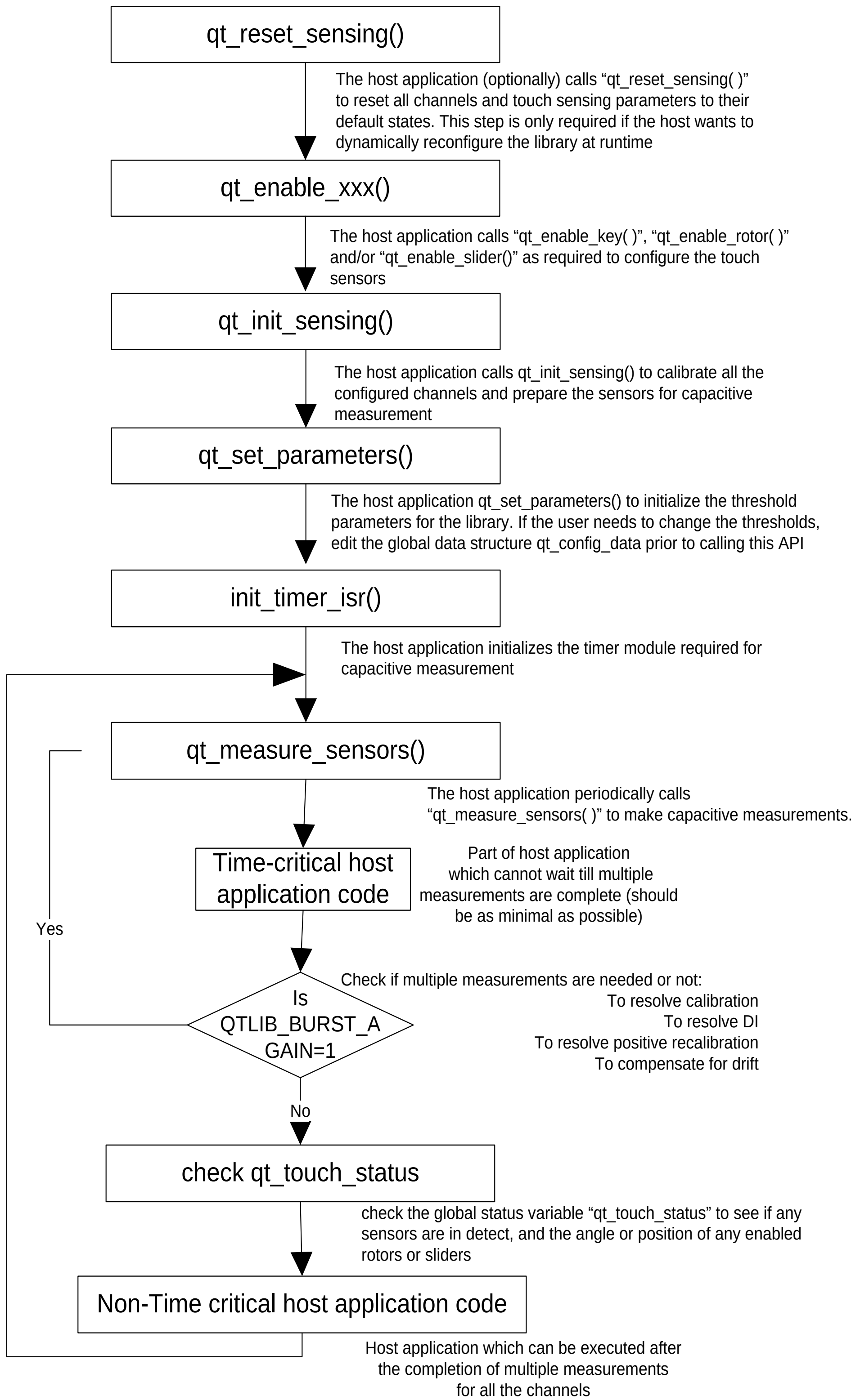


Figure 5-6: Sequence of operations to add Touch capability

5.6.6.1.1.1 Channel numbering when routing SNS and SNSK pins to different ports

Figure 5-7 illustrates a sample QTouch capacitive sensing solution which uses four ports (two port pairs) on a device for routing the SNS and SNSK lines required.

When SNS and SNSK pins are available on different ports, the channel numbering follows the pin numbering in the ports selected, when pin configurability is not used.

- The channel numbers follow the pin numbers starting with the LSB (pin 0 is channel 0 and pin 7 is channel 7).
- When a library on corresponding device is configured to use more than two ports for SNS and SNSK pins, the channel numbers in the second set of SNS/SNSK port pair continue from the preceding pair as illustrated in Figure 5-7(pin 0 of next port pair is channel 8 and pin 7 of the next port pair is channel 15).
- Support for more than one pair of SNS and SNSK ports are not available for UC3™ devices.
- SNS pins within a single port and SNSK pins within another single port can only be used as channels for slider/rotor. Slider/Rotor channels cannot share SNS/SNSK pins on different ports.
- Since the channel numbers are fixed to the pins of the SNS and SNSK ports, if the design calls for use of a subset of the pins available in the SNS and SNSK ports, the user has to skip the channel numbers of the unused SNS and SNSK pins.
    - For example, on a 8 channel configuration using a single pair of SNS and SNSK ports, if pin 2 is not used for touch sensing (on both SNS and SNSK ports), channel number 2 is unavailable and care should be taken while configuring the channels and sensors to avoid using this channel.

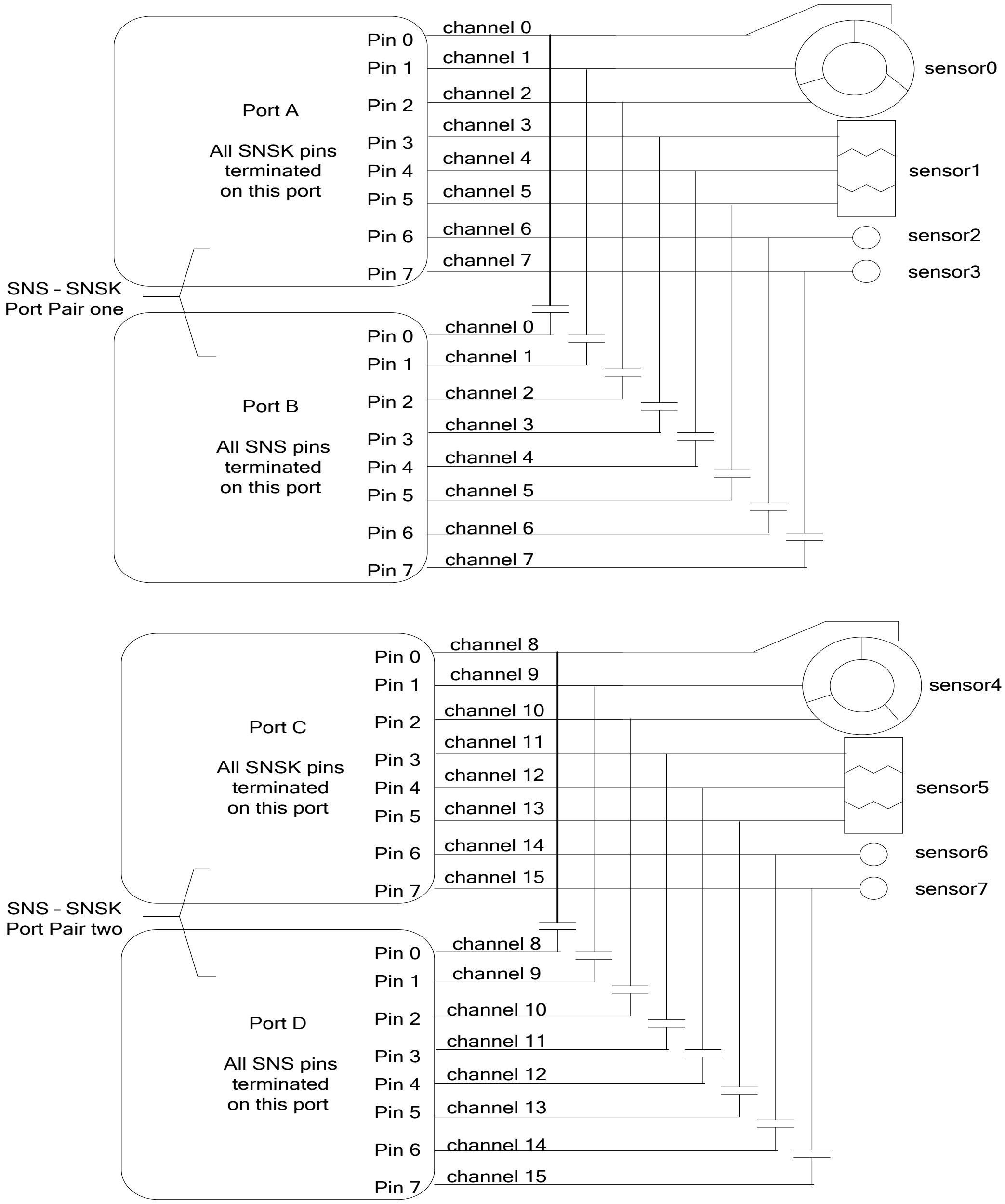


Figure 5-7 : channel numbering for QTouch acquisition method when the SNS and SNSK pins are connected to different ports.

5.6.6.1.1.2 Channel numbering when routing SNS and SNSK pins to different ports with pin configurability

When SNS and SNSK pins are available on different ports, the channel numbering follows the pin numbering in the ports selected based on SNS_array and SNSK_array bits enabled.The pins which needs to be used for touch should be provided in the Pin Configurator Wizard in QTouch Studio and the pin configurator Wizard tool will generate the SNS_array and SNSK_array masks and channel numbering will be based on which pins are enabled for touch in consecutive way.Below is an example to illustrate the same:

Example:

SNS and SNSK pins are configured with few rules keeping in mind as illustrated in section
Pins A0 ,A1,A4 and A6 of PORT A are SNS pins and pins B2,B3,B5,B7 are SNSK pins of PORT B.
Channel 0 will be forming a SNS-SNSK pair as A0B2.
Channel 1 will be forming a SNS-SNSK pair as A1B3
Channel 2 will be forming a SNS-SNSK pair as A4B5
Channel 3 will be forming a SNS-SNSK pair as A6B7.

The channel numbering is not dependent on the pin numbering.

5.6.6.1.1.3 Channel numbering when routing SNS and SNSK pins to the same port

When SNS and SNSK pins are connected to the same port, the even pin numbers will be used as SNS pins and the odd pins will be used as the SNSK pins.

- The number of channels supported will be limited 4 channels for an 8-bit device and 16 channels for a 32-bit device (e.g. UC3).
- For e.g., for a 4 channel configuration where the SNS and SNSK pins are connected to Port B, the port pins 0&1 are used for channel 0.
- The channel number is derived from the position of the pins used for SNS and SNSK lines for any channel.

  channel number = floor([SNS(or SNSK) pin number] / 2)

  - For e.g., pins 4 and 5 are connected to a SNS/SNSK pair and the channel number associated with the SNS/SNSK pin is 2.

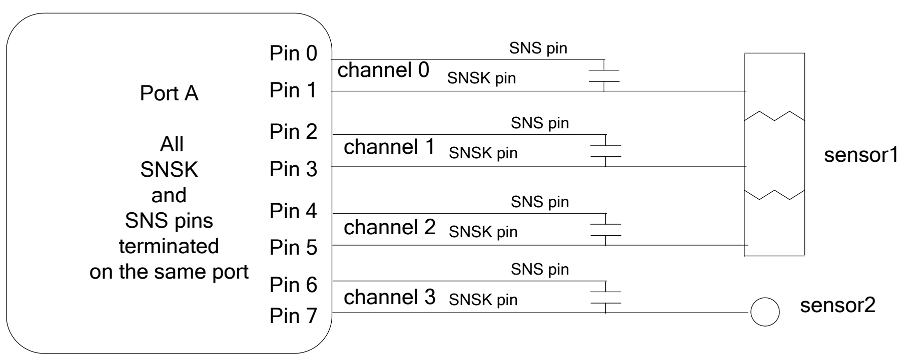


Figure 5-8 : Channel numbering for QTouch acquisition method when the SNS and SNSK pins are connected to the same port

5.6.6.1.1.4 Channel numbering when routing SNS and SNSK pins to the same port with pin configurability

When SNS and SNSK pins are connected to the same port, different pins can be used as SNS and SNSK pins.But SNS and SNSK pins are configured with few rules keeping in mind as illustrated in section

Example:
Pins A0 ,A3 and A5 of PORT A are SNS pins and pins A2,A4,A7 are SNSK pins of PORT A.
Channel 0 will be forming a SNS-SNSK pair as A0A2.

Channel 1 will be forming a SNS-SNSK pair as A3A4
Channel 2 will be forming a SNS-SNSK pair as A5A7.

The channel numbering is not dependent on the pin numbering.

5.6.6.1.2 Channel numbering when using QMatrix acquisition method

Figure 5-9 illustrates a QMatrix capacitive sensing solution which uses 4 X lines and 4 Y lines thereby providing a 16 channel solution.

Note:

1. All channels selected for a specific rotor or slider should be on a single Y line.
2. The choice of ports for X and Y lines is left to the user to based on the availability of the pins available in the particular device selected. Please refer to the section 5.8.2 for more details configuring of touch sensing pins for QMatrix.

The channel numbering for QMatrix configuration follows a matrix pattern with the channel numbers starting from 0 for the matrix intersection (X0Y0) and increasing along the X lines for a given Y line (Channel 1 is X1Y0) and then moving on to the row number 0 for the next column. Table 1 lists the possible channel numbers and the associated X/Y line associations for the different configurations of QMatrix library variants.

A group of channels form a sensor and the sensor numbering is determined by the order in which the user defines the association of channels and uses them as a sensor.

The channel numbering is fixed for a specific library variant based on the number of X and Y lines used whereas the sensor numbering is determined at the time of usage based on the order in which the user defines the association of the channels to create a sensor.

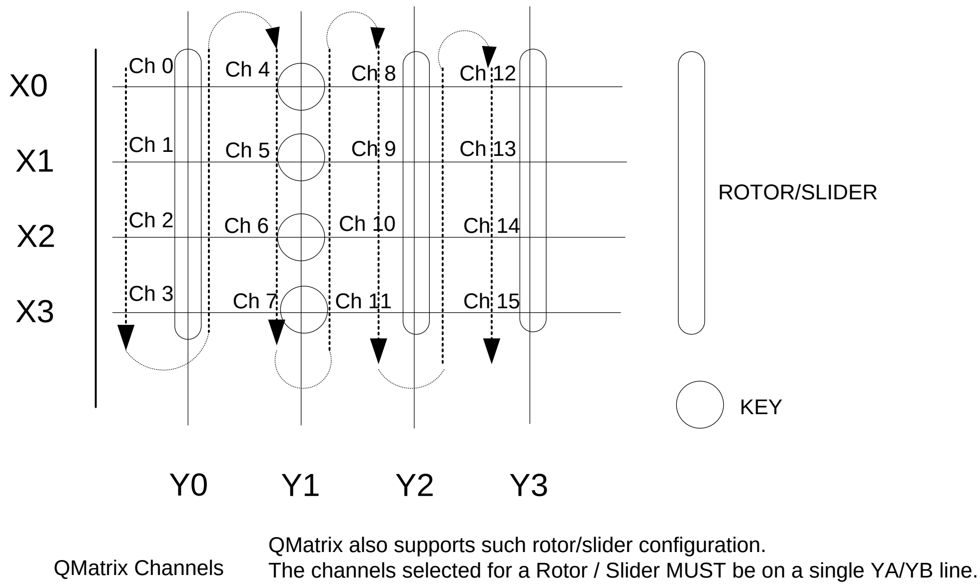


Figure 5-9: Channel Numbering for QMatrix acquisition method libraries

Table 1 : Channel numbers for QMatrix configurations

Line label	4 channel configuration (4 x 1)	8 channel configuration (4 x 2)	16 channel Configuration (8 x 2)	16 channel Configuration (4 x 4)	32 channel configuration (8 x 4)	56 channel configuration (8 x 7)	64 channel configuration (8 x 8)
Channel 0	X0Y0	X0Y0	X0Y0	X0Y0	X0Y0	X0Y0	X0Y0
Channel 1	X1Y0	X1Y0	X1Y0	X1Y0	X1Y0	X1Y0	X1Y0
Channel 2	X2Y0	X2Y0	X2Y0	X2Y0	X2Y0	X2Y0	X2Y0
Channel 3	X3Y0	X3Y0	X3Y0	X3Y0	X3Y0	X3Y0	X3Y0
Channel 4	N/A	X0Y1	X4Y0	X0Y1	X4Y0	X4Y0	X4Y0
Channel 5	N/A	X1Y1	X5Y0	X1Y1	X5Y0	X5Y0	X5Y0
Channel 6	N/A	X2Y1	X6Y0	X2Y1	X6Y0	X6Y0	X6Y0
Channel 7	N/A	X3Y1	X7Y0	X3Y1	X7Y0	X7Y0	X7Y0
Channel 8	N/A	N/A	X0Y1	X0Y2	X0Y1	X0Y1	X0Y1
Channel 9	N/A	N/A	X1Y1	X1Y2	X1Y1	X1Y1	X1Y1
Channel 10	N/A	N/A	X2Y1	X2Y2	X2Y1	X2Y1	X2Y1
Channel 11	N/A	N/A	X3Y1	X3Y2	X3Y1	X3Y1	X3Y1
Channel 12	N/A	N/A	X4Y1	X0Y3	X4Y1	X4Y1	X4Y1
Channel 13	N/A	N/A	X5Y1	X1Y3	X5Y1	X5Y1	X5Y1
Channel 14	N/A	N/A	X6Y1	X2Y3	X6Y1	X6Y1	X6Y1
Channel 15	N/A	N/A	X7Y1	X3Y3	X7Y1	X7Y1	X7Y1
Channel 16	N/A	N/A	N/A	N/A	X0Y2	X0Y2	X0Y2
Channel 17	N/A	N/A	N/A	N/A	X1Y2	X1Y2	X1Y2
Channel 18	N/A	N/A	N/A	N/A	X2Y2	X2Y2	X2Y2
Channel 19	N/A	N/A	N/A	N/A	X3Y2	X3Y2	X3Y2
Channel 20	N/A	N/A	N/A	N/A	X4Y2	X4Y2	X4Y2
Channel 21	N/A	N/A	N/A	N/A	X5Y2	X5Y2	X5Y2
Channel 22	N/A	N/A	N/A	N/A	X6Y2	X6Y2	X6Y2
Channel 23	N/A	N/A	N/A	N/A	X7Y2	X7Y2	X7Y2
Channel 24	N/A	N/A	N/A	N/A	X0Y3	X0Y3	X0Y3
Channel 25	N/A	N/A	N/A	N/A	X1Y3	X1Y3	X1Y3
Channel 26	N/A	N/A	N/A	N/A	X2Y3	X2Y3	X2Y3
Channel 27	N/A	N/A	N/A	N/A	X3Y3	X3Y3	X3Y3
Channel 28	N/A	N/A	N/A	N/A	X4Y3	X4Y3	X4Y3
Channel 29	N/A	N/A	N/A	N/A	X5Y3	X5Y3	X5Y3
Channel 30	N/A	N/A	N/A	N/A	X6Y3	X6Y3	X6Y3
Channel 31	N/A	N/A	N/A	N/A	X7Y3	X7Y3	X7Y3
Channel 32	N/A	N/A	N/A	N/A	N/A	X0Y4	X0Y4
Channel 33	N/A	N/A	N/A	N/A	N/A	X1Y4	X1Y4
Channel 34	N/A	N/A	N/A	N/A	N/A	X2Y4	X2Y4
Channel 35	N/A	N/A	N/A	N/A	N/A	X3Y4	X3Y4
Channel 36	N/A	N/A	N/A	N/A	N/A	X4Y4	X4Y4
Channel 37	N/A	N/A	N/A	N/A	N/A	X5Y4	X5Y4
Channel 38	N/A	N/A	N/A	N/A	N/A	X6Y4	X6Y4
Channel 39	N/A	N/A	N/A	N/A	N/A	X7Y4	X7Y4
Channel 40	N/A	N/A	N/A	N/A	N/A	X0Y5	X0Y5
Channel 41	N/A	N/A	N/A	N/A	N/A	X1Y5	X1Y5
Channel 42	N/A	N/A	N/A	N/A	N/A	X2Y5	X2Y5
Channel 43	N/A	N/A	N/A	N/A	N/A	X3Y5	X3Y5
Channel 44	N/A	N/A	N/A	N/A	N/A	X4Y5	X4Y5
Channel 45	N/A	N/A	N/A	N/A	N/A	X5Y5	X5Y5
Channel 46	N/A	N/A	N/A	N/A	N/A	X6Y5	X6Y5
Channel 47	N/A	N/A	N/A	N/A	N/A	X7Y5	X7Y5
Channel 48	N/A	N/A	N/A	N/A	N/A	X0Y6	X0Y6
Channel 49	N/A	N/A	N/A	N/A	N/A	X1Y6	X1Y6
Channel 50	N/A	N/A	N/A	N/A	N/A	X2Y6	X2Y6
Channel 51	N/A	N/A	N/A	N/A	N/A	X3Y6	X3Y6

Channel 52	N/A	N/A	N/A	N/A	N/A	X4Y6	X4Y6
Channel 53	N/A	N/A	N/A	N/A	N/A	X5Y6	X5Y6
Channel 54	N/A	N/A	N/A	N/A	N/A	X6Y6	X6Y6
Channel 55	N/A	N/A	N/A	N/A	N/A	X7Y6	X7Y6
Channel 56	N/A	N/A	N/A	N/A	N/A	N/A	X0Y7
Channel 57	N/A	N/A	N/A	N/A	N/A	N/A	X1Y7
Channel 58	N/A	N/A	N/A	N/A	N/A	N/A	X2Y7
Channel 59	N/A	N/A	N/A	N/A	N/A	N/A	X3Y7
Channel 60	N/A	N/A	N/A	N/A	N/A	N/A	X4Y7
Channel 61	N/A	N/A	N/A	N/A	N/A	N/A	X5Y7
Channel 62	N/A	N/A	N/A	N/A	N/A	N/A	X6Y7
Channel 63	N/A	N/A	N/A	N/A	N/A	N/A	X7Y7

5.6.6.2 Sensor Numbering

The ordering and numbering of sensors is related to the order in which the sensors are enabled. This is independent of the acquisition method (QMatrix or QTouch acquisition method libraries).

For example, consider this code snippet:

….

```
/* enable slider */
qt_enable_slider (CHANNEL_0, CHANNEL_2, AKS_GROUP_1, 16,
HYST_6_25, RES_8_BIT, 0);

/* enable rotor */
qt_enable_rotor (CHANNEL_3, CHANNEL_5, AKS_GROUP_1, 16, HYST_6_25,
RES_8_BIT, 0);

/* enable keys */
qt_enable_key (CHANNEL_6, AKS_GROUP_2, 10, HYST_6_25);
qt_enable_key (CHANNEL_7, AKS_GROUP_2, 10, HYST_6_25);
```

In the case above, the slider on channels 0 to 2 will be sensor 0, the rotor on channels 3-to-5 is sensor 1 and the keys on channels 6 and 7 are sensor numbers 3 and 4 respectively.

When the touch status is reported or queried, the corresponding sensor positions and status indicate the touch status. For example, the slider is in detect if "*qt_measure_data. qt_touch_status.sensor_states*" bit position 0 is set. Similarly, the rotor on channels 3 to 5 is sensor 1, and the keys on channels 6 and 7 are sensors 2 and 3 respectively.
However, the code could be re-arranged as follows to give a different sensor numbering.

```
/* enable rotor */
qt_enable_rotor (CHANNEL_3, CHANNEL_5, NO_AKS_GROUP, 16,
HYST_6_25, RES_8_BIT, 0);

/* enable keys */
qt_enable_key (CHANNEL_6, AKS_GROUP_2, 10, HYST_6_25);
qt_enable_key (CHANNEL_7, AKS_GROUP_2, 10, HYST_6_25);

 /* enable slider */
qt_enable_slider (CHANNEL_0, CHANNEL_2, NO_AKS_GROUP, 16,
HYST_6_25, RES_8_BIT, 0);
```

Now, the rotor is sensor 0, the keys are sensors 1 and 2, and the slider is sensor 3.

So, the order in which the user enables the sensors is the order in which the sensors are numbered. Depending on the user requirements, the sensors can be configured in the preferred order.
NOTE: In case of QMatrix, the channels on the Unused X lines (or) unused Y lines should be ignored and not to be used as arguments in this API.

Ex: If the host application needs only 24 channels , there are two possible options.

1. In 32 (8x4 configuration), if X6 and X7 are unused, channel6, channel7, channel14, channel15, channel 22, channel23, channel30, channel 31 cannot be used
2. In 32 (8x4 configuration), if Y3 is unused, channe24, channel25, channel26, channel27, channel 28, channel29, channel30, channel 31 cannot be used

5.6.6.3 Filtering Signal Measurements

The ATMEL QTouch Library API provides a function pointer called “qt_filter_callback”. The user can use this hook to apply filter functions to the measured signal values.

If the pointer is non-NULL, the library calls the function after library has made capacitive channel measurements, but before the library has processed the channel information and determining the sensor states.

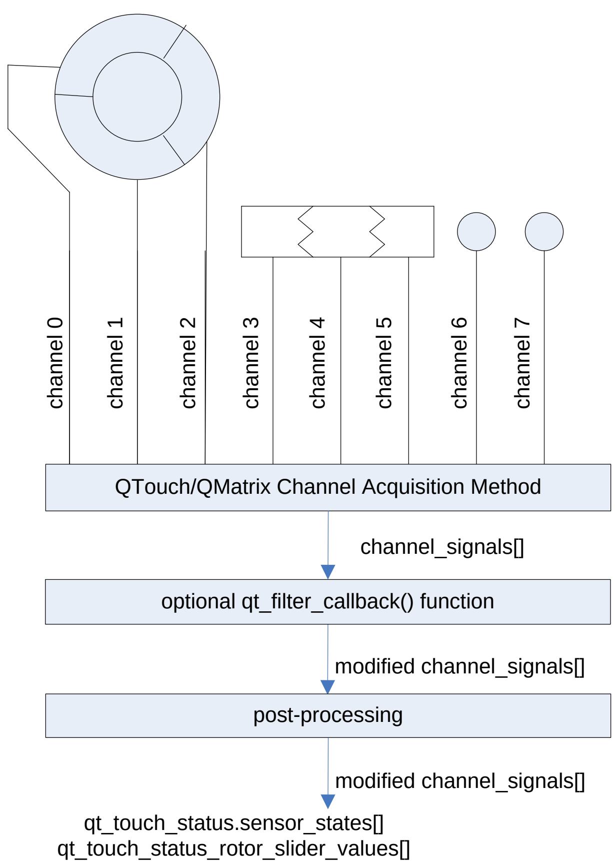


Figure 5-10 : Block diagram to represent usage of filter callback function

Example: Averaging the Last Four Signal Values
1. Add a static variable in the main module:

```
/* filter for channel signals */
static uint16_t filter[QT_NUM_CHANNELS][4];
```

2. Add a filter function prototype to the main module:

```
/* example signal filtering function */
static void filter_data_mean_4( void );
```

3. When configuring the ATMEL QTouch library, set the callback function pointer:

```
/* set callback function */
qt_filter_callback = filter_data_mean_4;
```

4. Add the filter function:

```
void filter_data_mean_4( void )
{
uint8_t i;
/*
* Shift previously stored channel signal data.
* Store new channel signal data.
* Set library channel signal data = mean of last 4 values.
*/

for( i = 0u; i < QT_NUM_CHANNELS; i++ )
      {
      filter[i][0] = filter[i][1];
      filter[i][1] = filter[i][2];
      filter[i][2] = filter[i][3];
      filter[i][3] = qt_measure_data.channel_signals[i];
      qt_measure_data.channel_signals[i] = ( (  filter[i][0] +
      filter[i][1] +
      filter[i][2] +
      filter[i][3] ) / 4u );
      }
}
```

The signal values processed by the ATMEL QTouch Library are now the mean of the last four actual signal values.

5.6.6.4 Allocating unused Port Pins for User Application

The GPIO pins within a port that are not used for QTouch or QMatrix acquisition methods can be used for user application. The usage of pins for QTouch is based on the channels that are being configured while enabling the sensors (keys/rotors/sliders).

The example below configuring 4 keys, a rotor and a slider shows how the pin configurability is achieved by configuring the sensor channels. The code snippet configures a specific 10 channels of a 16 channel library based on the GPIO port pins available for QTouch™.

Port Configuration:

```
#define SNSK1     C
#define SNS1      D
#define SNSK2     A
#define SNS2      B
```

Channel/Pin Configuration:

```
/* enable a key on channel 0    */
qt_enable_key( CHANNEL_0, AKS_GROUP_2, 10u, HYST_6_25 );
```

```
    /* enable a slider on channels 2 to 4  */
    qt_enable_slider( CHANNEL_2, CHANNEL_4, AKS_GROUP_1, 16u, HYST_6_25,
RES_8_BIT, 0u  );

    /* enable a key on channel 6 */
    qt_enable_key( CHANNEL_6, AKS_GROUP_2, 10u, HYST_6_25 );

    /* enable a key on channel 7 */
    qt_enable_key( CHANNEL_7, AKS_GROUP_2, 10u, HYST_6_25 );

    /* enable a rotor on channels 12 to14   */
    qt_enable_rotor(   CHANNEL_12,   CHANNEL_14,   AKS_GROUP_1,   16u,
HYST_6_25, RES_8_BIT, 0u );

    /* enable a key on channel 15 */
    qt_enable_key( CHANNEL_15, AKS_GROUP_2, 10u, HYST_6_25 );
```

The channel numbers 0,2,3,4,6,7 are allocated to pins 0,2,3,4,6,7 of (D,C) port pair respectively. Pins 1 and 5 of ports C and D can be used for user application. Similarly the channel numbers 12,13,14,15 are allocated to pins 4,5,6,7 of (B,A) port pair respectively. Pins 1, 2, 3 and 4 of ports B and A are again unused by the QTouch library and can be used for user application.

5.6.6.5 Disabling and Enabling of Pull-up for AVR devices

The Pull-up circuit available (in AVR devices) for each GPIO pin has to be disabled before QTouch acquisition is performed. For tinyAVR and megaAVR devices the Pull-up circuit for all GPIO port pins are enabled and disabled together. When user needs to configure the pins that are not used by QTouch library for his application, he may enable the Pull-up circuit after QTouch measurements are performed and disable them before the touch acquisition starts once again (as shown in the code snippet below).

```
  /* Disable pull-ups for all pins */
  MCUCR     |=    (1u << PUD);       //MCUCR_PUD = 1u;

  /* perform QTouch measurements */
  qt_measure_sensors ( current_time_ms_touch );

  /* Enable pull-ups for all pins */
    MCUCR   &=    ~ (1u << PUD);      //MCUCR_PUD = 0u;
```

For XMEGA devices the Pull-up circuit for each individual GPIO port pins can be configured individually, by writing to the PINnCTRL register of the ports being used.

5.6.7 Constraints

5.6.7.1 QTouch acquisition method constraints

QTouch acquisition method libraries are available for different port combinations.

Some of the key constraints while configuring the sensors are

- Rotors/sliders have to be connected on three adjacent channels. (e.g. (1,2,3) or (3,4,5) …) within the same port. Possible combinations are (0,1,2), (1,2,3) for a configuration which supports 4 channels. Possible combinations (0,1,2), (1,2,3), (2,3,4), (3,4,5), (4,5,6), (5,6,7) for a configuration which supports 8 channels.

- If two port pairs are used for the design, all the channels for a sensor have to be connected on a single port pair. Combining channels from multiple ports is not possible when designing sensors. e.g. It is not possible to have a rotor with channel numbers (7,8,9) on a 16 channel library variant which uses two port-pairs.

Note: The above constraints are explained with respect to 8bit AVR. The same could be extended to 32bit AVR and ATSAM for 32 channel libraries where each port has 32 pins.

5.6.7.2 QMatrix acquisition method constraints

QMatrix acquisition method libraries are available for a set of AVRs The library variants can be configured to have port and pin assignments for X, Ya, Yb and SMP. Please refer to section 5.8.2 for port-pin configurability.

Some of the key constraints are

- The QMatrix acquisition method libraries internally use TIMER1 for the operation, TIMER1 will not be available for critical sections of the code where the library is called. But resources are available to the host application when the normal user's application is running.
- In case of XMEGA™ devices, the resources are used internal to the library and hence cannot be used by the host application
  - Timer/Counter 1 on PORTC (TCC1)
  - Analog Comparator on PORTA (ACA)
  - Event System Channel0 (EVSYS_CH0)
- The sensor channel number and the relation with X and Y lines strictly follows from the table provided in the section Table 1.
- A rotor /slider sensor can be configured with 3 to 8 channels per rotor or slider depending on the requirement of the application subject to the total number of channels available in the library variant selected as listed below.

Number of channels	X x Y	Maximum Channels per ROTOR_SLIDER
4	4 x 1	4
8	4 x 2	4
16	4 x 4	4
16	8 x 2	8
32	8 x 4	8
56	8 x 7	8
64	8 x 8	8

- For example, 16 channel libraries with 4X and 4Y lines supports maximum of 4 channels per Rotor/Slider. But, a 16 channel with 8X and 2Y lines supports maximum of 8 channels per Rotor/Slider.
- If the lines of the Drive and Receive electrode (X lines or the Y lines) share the same lines with the JTAG, JTAG needs to be disabled. Please check the data sheet to ensure that there are no conflicts between the X/Y lines and JTAG lines used for the device.
- YB line for a particular device cannot be changed and it has to be the configured to be the ADC port of the selected device.
- The AIN0 pin of the device needs to be connected to the GND.
- In case of XMEGA devices, the reference pin for input to analog comparator is Pin7 of PORTA with all the combinations of libraries supported. Hence, this needs to be connected to GND
- Proper grounding should be taken care when the controller board and touch sensing board are different.

- The channels used for an individual rotor or slider should all be on the same Y line.
- The maximum number of Rotors / Sliders supported by the QMatrix acquisition method depends on the configuration. Refer to the Library_Selection_Guide.xls for details.
- Vcc should be kept at 4.5V or lower for reliable operation

5.6.7.3 Design Guidelines for QMatrix acquisition method systems

AVR Microcontrollers can use a number of clock sources, ranging from high precision external crystals to less accurate resonators down to simple external RC circuits. Most AVR devices also come with integrated RC oscillators. This provides a system clock source without additional cost or board space. When using internal RC oscillators some considerations need to be taken. The accuracy i.e. frequency of CMOS RC oscillators will vary slightly from device to device due to process variance.

QMatrix acquisition method uses an internal timer to measure the discharge time of a capacitor, and any frequency variation or fluctuation in the RC Oscillator will thus show up as a variance in the measurement data. The application should for this reason be designed and tuned to allow for such variance in the internal RC oscillator frequency. For most AVR microcontrollers, the rated accuracy of the internal RC oscillator is 2%, and to have some headroom and guarantee a robust and stable system, the designer should aim to follow these design rules:

- Reference Value should be in the 150-300 range
- Typical delta when touched should be at least 10% of the Reference Value
- Recommended threshold should be at least 5% of the reference value and at least 50% of the typical delta (Higher value gives better robustness)
- Hysteresis should be as high as possible in noisy systems (50%)
- DI should be set to at least 4

If the design of the system does not comply with the rules above, special attention should be taken when testing it to make sure that the design meets the desired performance. In systems with big signal values and small deltas (i.e. less than 10%) it is recommended to either change component values to conform to the 10% delta rule, or change to a higher precision clock source.

QTouch Studio is the preferred tool when checking and validating any QTouch Designs.

5.6.8 Frequency of operation (Vs) Charge cycle/dwell cycle times:

The library needs different charge / dwell cycles based on the operation and design. The charge/dwell cycles are determined by the QT_DELAY_CYCLES parameter defined by the user. The recommended range of charge/dwell cycle times that the user must select based on the operating clock frequency of the Microcontroller is provided in the table below.

Fine tuning of the QT_DELAY_CYCLES to match the sensor design may be done by monitoring the reference levels, and finding the charge/dwell time where the reference level has reached >99% of maximum reference value seen. For QTouch acquisition method, the reference value will decrease as the QT_DELAY_CYCLES is increased. For QMatrix acquisition method, the reference value will increase with increase in QT_DELAY_CYCLES. If the cycle time is not optimum, the design may experience temperature sensitivity.

Possible values:
The following table lists the possible values of QT_DELAY_CYCLES for both QTouch and QMatrix acquisition method libraries.

Acquisition method	Possible values

QTouch	Any value from 1- 255 for 8bit AVR 3,4,5,10,25,50 for UC3 and ATSAM libraries
QMatrix	1,2,3,4,5,10,25,50

Example:
When operating at 4 MHz, 1~10 cycle charge times are recommended (0.125us to 1.25us).

Table 2 : Frequency of operation

Frequency of Microcontroller (MHz))	microcontroller Cycle time (us)	Suitable Charge Cycle times (or) Suitable Dwell Cycle times (us)
1	1	1 to 2 cycles (1us to 2us)
2	0.5	1 to 5 cycles (0.5us to 2.5us)
4	0.25	1 to 10 cycles (0.25us to 2.5us)
8	0.125	1 to 10 cycles (0.125us to 1.25us)
10	0.1	2 to 25 cycles (0.2us to 2.5us)
16	0.0625	2 to 25 cycles (0.125us to 1.5625us)
20	0.05	3 to 50 cycles (0.15us to 2.5us)
48	0.02083	5~50 cycles (0.104us to 1.04us)
>48	<0.02083	5 to < 50 (up to 255 cycles for 8bit AVR)

Note:

- For UC3 and ATSAM devices, 1 & 2 charge cycle delay times are not supported.

If the microcontroller is only used for Touch detection then running at the lowest frequency possible for the desired touch response may provide the best power and EMC performance. If it is also used for other functions then running at a higher frequency may be necessary. In some power critical applications it may be worth switching the frequency on the fly, such as lowering the frequency during touch detect API instead of using long cycle times, and then switching to a higher frequency for non-touch code. It is necessary to carefully design timer operation when change frequencies.

5.6.9 Interrupts

This section illustrates the usage of interrupts during qt_measure_sensors call.

The library disables interrupts for time-critical periods during touch sensing. These periods are generally only a few cycles long, and so host application interrupts should remain responsive during touch sensing. However, any interrupt service routines (ISRs) during touch sensing should be as short as possible to avoid affecting the touch measurements or the application responsiveness.

Interrupts are disabled once for each signal count/burst pulse and this is typically 65 instruction cycles when Delay cycles (QT_DELAY_CYCLE=1).
The number of times interrupts are disabled during one measurement will depend on signal count of a channel as well as the number of channels and port configuration like interport (SNS and SNSK on different port)/intraport (SNS and SNSK on same port).

Example:

4 channel intraport case:
Channel0 is formed by PA0 and PA1 pin with signal count 300.
Channel1 is formed by PA2 and PA3 pin with signal count 200.
Channel2 is formed by PA4 and PA5 pin with signal count 250

Channel3 is formed by PA6 and PA7 pin with signal count 150
In the above case, the no of times interrupts disabled will be 300 (maximum signal count) as all four channels burst together in case of intraport.

4 channel interport case:
Channel0 is formed by PA0 and PB0 pin with signal count 300.
Channel1 is formed by PA1 and PB1 pin with signal count 200.
Channel2 is formed by PA2 and PB2 pin with signal count 250
Channel3 is formed by PA3 and PB3 pin with signal count 150

In the above case, as bursting happens in odd and even pairs, so maximum signal counts in case of both even and odd channels will be taken. Maximum signal count out of even channels channel0 and channel2 is 300 and maximum signal count out of odd channels channel1 and channel3 is 250.So total number of times interrupts disabled will be (300 + 250 = 550).

The recommended maximum ISR execution time is 1msec.If ISR time exceeds 1 ms then it may result in charge leakage which will further lead to rise in signal value. Noise observed in the signal value will increase. The 1msec ISR limit applies to the total time spent in an ISR during one acquisition period, and applies only to QTouch and QMatrix. One acquisition period is the time to complete one burst sequence. This will be the time required to do the 300 pulses situation in the first example above. In the second example above the 1msec limitation applies to the 300 and then the 250 pulses; 1msec of total ISR can occur during the 300 and another 1msec ISR can occur during the 250. This is all about self discharge happening while doing a burst. In the example below it would be perfectly OK to have a 1msec ISR happening every 5 msec.
If the ISR load is constant and synced with the acquisition (meaning the ISR takes a constant amount of time and executes in the same amount of time during each burst sequence), the signal will not suffer at all since the self discharge will be the same every time the acquisition is run. If there is a strong variation in the total ISR execution time during acquisition this will appear as noise in the signal due to the variable self discharge of the sample cap.

The time to execute one measurement will depend on various parameters like sampling capacitor, operating voltage, and different software parameters like QT_DELAY_CYCLES, CPU frequency.
For single button with below parameters
Sampling Capacitor = 10 nf
CPU Freq= 4 MHZ
VCC= 5V
QT_DELAY_CYCLES=1
Qtouch takes around 2.6 msec for one channel.

Please note that none of the API functions should be called from a user interrupt.

5.6.10 Integrating QTouch libraries in your application

This section illustrates the key steps required in integrating the QTouch library in your application.

5.6.10.1 Directory structure of the library files

The QTouch library directory structure is as listed below

What	Where	Comments
Root installation	Default directory is C:\Program Files\Atmel\Atmel_QTouch_Libraries_5.x\Generic_QTouch_Libraries	This is the default directory path but the user can install the directory in desired location.
Header file	..\include	touch_api.h is located in this

				directory.touch_api_2kdevic.h for 2K devices support is also added in this directory
Configuration and assembler routines for acquisition	QTouch acquisition method libraries	8-bit devices	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QTouch\common_files	qt_asm_avr.h qt_asm_tiny_mega.S qt_asm_xmega.S qt_asm_avr_config_2kdevice.h qt_asm_tiny_mega_2kdevice.S
		UC3	Not needed for UC3 devices	
		ATSAM	Not needed for ATSAM devices	
	QMatrix acquisition method libraries	8-bit devices	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QMatrix\common_files	qm_asm_avr.h qm_asm_tiny_mega.S qm_asm_m8535_m16.S qm_asm_xmega.S qm_asm_tiny_mega_m64_v3g4_avr51g1.S
		UC3	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries\32bit_AVR\UC3\QMatrix\common_files	touch_config.h touch_qm_config32_assembler.h qm_asm_uc3c_gcc.x qm_asm_uc3c_iar.s82
Library files	QTouch acquisition method libraries	8-bit devices	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QTouch\library_files	All libraries (.r90 for IAR and .a for GCC) for the supported 8 bit devices are in this location. Also r82 libraries for AVR 32 bit devices are also here
		UC3	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \32bit_AVR\UC3\QTouch\library_files	
		ATSAM	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AT91SAM\SAM3\QTouch\library_files ..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AT91SAM\SAM4\QTouch\library_files	
	QMatrix acquisition method libraries	8-bit devices	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QMatrix\library_files	
		UC3	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries\32bit_AVR\UC3\QMatrix\library_files	
Example Projects	QTouch acquisition method libraries	8-bit devices	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QTouch\example_projects	All example projects using the libraries above (IAR and GCC) for the supported devices are in

		UC3	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \32bit_AVR\UC3\QTouch\example_projects	this location
		ATSAM	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries\AT91SAM\SAM3\QTouch\example_projects ..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries\AT91SAM\SAM4\QTouch\example_projects\SAM4S_XPLAINED_DEMO_APPLICATION1	
	QMatrix acquisition method libraries	8-bit devices	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QMatrix\example_projects	
		UC3	..\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries\32bit_AVR\UC3\QMatrix\example_projects	

5.6.10.2 Integrating QTouch acquisition method libraries in your application

The following steps illustrate how to add QTouch acquisition method support in your application.

1) QTouch acquisition method library variants are offered for IAR and AVR Studio/GCC tool chains. First step is to select the compiler tool chain to be used based on the code and data memory requirements. The list of supported compiler tool chains can be found in 5.7.1.2.
   Use the library selection guide (C:\ Program Files\Atmel\Atmel_QTouch_Libaries_5.x\ Library_Selection_Guide.xls) to select the QTouch acquisition method library variant required for the device.
   a. There are specific library variants distributed for each microcontroller. You would need the following parameters to identify the right library variant to be used in your application
      i. The microcontroller to be used for the application.
      ii. The acquisition method to be used for the application.
      iii. The number of channels you need for the application.
      iv. Whether Rotor and/or Slider support required in the application.
      v. The number of rotors and/or slider needed for the application.
   b. There are specific variants of the library which is pre-built with a specific configuration set supported. Use the library selection guide (C:\ Program Files\Atmel\ Atmel_QTouch_Libaries_5.x\Library_Selection_Guide.xls) to find the sample project using the QTouch acquisition method library variant.
2) Define the constants and symbol names required
   a. The next step is to define certain constants and symbols required in the host application files where the touch API is going to be used.
   b. The constant/symbol names are as listed in the table below.
   c. The constant/symbol definitions can be placed in the touch_config.h file. The user may modify these defined values based on the requirements.

Table 3 : Constant and symbol name definitions required to use the QTouch acquisition method libraries

Symbol / Constant name	Range of values	Comments
QTOUCH	This macro has to be defined in order to use QTouch libraries.	

SNS1 & SNSK1	Refer to library selection guide.	To be used if only single port pair is needed for the design.
SNS1 – SNSK1 & SNS2 – SNSK2	Refer to library selection guide.	To be used if two port pairs are needed for the design.
_SNS1_SNSK1_SAME_PORT_	Comment/uncomment define	To be enabled if the same port is used for SNSK1 and SNS1 pins for QTouch. If SNSK1 and SNS1 pins are on different ports then this definition is not required.
_SNS2_SNSK2_SAME_PORT_	Comment/uncomment define	To be enabled if the same port is used for SNSK2 and SNS2 pins for QTouch. If SNSK1 and SNS1 pins are on different ports then this definition is not required.
QT_NUM_CHANNELS	4, 8, 12, 16 for tinyAVR, megaAVR and XMEGA device libraries and 8, 16, 32 for UC3 device libraries.	
_ROTOR_SLIDER_	Rotor / slider can be added to the design, if this macro is enabled.	A library with rotor / slider functionality already available needs to be selected if this macro is to be enabled.
QT_DELAY_CYCLES	1 to 255	Please refer to section 5.6.8.
_POWER_OPTIMIZATION_ (Required only for ATtiny and ATmega libraries. ATxmega and UC3 libraries by default optimized for power without any limitations)	0 or 1	Used to reduce the power consumed by the library. When power optimization is enabled the unused pins, within a port used for QTouch, may not be usable for interrupt driven applications. Spread spectrum noise reduction is also disabled when power optimization is enabled.
_TOUCH_ARM_	To be defined when using ATSAM libraries	For ATSAM libraries only.
QTOUCH_STUDIO_MASKS	This macro needs to be defined if QTouch Studio Pin Configurator Wizard.is used to generate the SNS and SNSK masks.	Please refer to section 5.8.1
_STATIC_PORT_PIN_CONF_	This macro needs to be defined only in case of 4 and 8 channel libraries with interport configuration and pin configurability.	Please refer to section 5.8.1

4) Using QTouch API’s in your application to add touch functionality

a. The clock, host application and other peripherals needed by the host application needs to be initialized.
b. In your application, create, initialize and configure the sensors.
   i. The APIs of interest are qt_enable_key/rotor/slider().see sections 5.6.5.2, 5.6.5.3 and 5.6.5.4.
c. The channel configuration parameters need to be set by calling the *qt_set_parameters()* (see section 5.6.5.1.
d. Once the sensors are configured, qt_init_sensing() has to be called to trigger the initialization of the sensors with the configuration defined in steps above.
d. Provide timing for the QTouch libraries to operate. i,e the QTouch libraries do not use any timer resources of the microcontroller. The Host application has to provide the required timing and also call the API's at the appropriate intervals to perform touch sense detect operations.

NOTE: The ATSAM example applications provided with the libraries illustrate the usage for the evaluation kits supported by the library. Please refer to the main.c files for reference.

5) Adding the necessary source files
The following files are to be added along with the touch library and user application before compilation:
   - ATtiny, ATmega devices - touch_api.h, qt_asm_avr.h, touch_config.h and qt_asm_tiny_mega.S
   - ATxmega devices - touch_api.h, qt_asm_avr.h, touch_config.h and qt_asm_xmega.S
   - UC3 devices – touch_api.h
   - ATSAM devices - touch_api.h and touch_qt_config.h

6) General application notes
   - The clock, host application and other peripherals needed by the host application needs to be initialized.
   - Ensure that there are no conflicts between the resources used by the touch library and the host application.
   - Ensure that the stack size for your application is adjusted to factor in the stack depth required for the operation of the touch libraries.

5.6.10.2.1 Example for 8bit AVR

The example below will explain in detail the steps to follow for library selection.

Criteria	**Selection**	**Notes**
Microcontroller	ATMega1280	
IDE and compiler tool chain used	AVR STUDIO® IDE and GNU compiler	The GCC compiled variant of the libraries for the device selected needs to be used.
Number of Keys required for the application	3	Each key requires 1 QTouch acquisition channel
Rotors and sliders required	Yes	
Number of Rotors and Sliders required	3	Each rotor / slider will require 3 channels.
Number of Channels required for the application (should be the sum of all channels required for all the keys ,rotors and sliders used in the design)	12	3 Keys + (3 rotors x 3 channels per rotor/slider) → 12 channels
Charge cycle time required for the design	1 cycle	Assuming the device is configured with a clock frequency of 4Mhz
Number of ports needed	3 ports	This is determined based on the number of channels required and the routing required for the channels SNS and SNSK pins to the ports For this design, 24 pins are required and we need

			3 ports to support the sensors.
Choice of ports available for the design	SNS/ SNSK Pair1 ports	SNS1 Port : A	The choice of ports for the port pairs is limited and can be found in the section 5.7.1.5
		SNSK1 Port : A	
	SNS/ SNSK Pair 2 ports	SNS2 Port : B	
		SNSK2 Port : C	
Is there a need for reduced power consumption (and reduced execution time)?	_POWER_ OPTIMIZATION_ = 1		Enabling _POWER_OPTIMIZATION_ will lead to a 40% reduction in power consumed by the library, but at the expense of reduced external noise immunity. When power optimization is enabled, the unused pins within a port used for QTouch, may not be usable for interrupt driven applications. This option is available only for ATtiny and ATmega devices.
SNS1 and SNSK1 pins use the same port.	_SNS1_SNSK1_SAME _PORT_		The _SNS1_SNSK1_SAME_PORT_ symbol needs to be defined as port A is used for both SNS1 and SNSK1 pins.

Given the above requirements for the applications, the first step is to select the right library variant required.

Step 1: Selecting the right library variant

Referring to the library selection guide, we see that there are a few variants of libraries supported for ATmega1280. Since the application requires 12 channels and rotor slider support, one has to select a library variant which supports at least 12 channels or more along with 3 Rotors/Sliders. Hence we select the 12 channel library variant for GCC complier which supports the required number of sensors/channels. This works out to be `libavr51g1_12qt_k_3rs.a`

Step 2: Defining the constants / symbols in the project space
In the host application file (say main.c), define the following constants and symbols

*#define QTOUCH_*
#define QT_NUM_CHANNELS 12
#define SNSK1 A
#define SNS1 A
#define SNSK2 B
#define SNS2 C
#define QT_DELAY_CYCLES 1
#define _POWER_OPTIMIZATION_ 1
*#define _SNS1_SNSK1_SAME_PORT_*

NOTE: The above definitions are available in touch_config.h *file. Alternatively, you can define these in your IDE's project options or have them defined in a separate header file.*

Step 3: Usage of library API's

Now, you can use the touch API's to create, initialize and perform touch sensing. Please refer to the sample applications in section 5.6.11.2 for reference. These sample applications illustrate the usage of the API's and the sequence of operation.

Step 4: Adding necessary source files for compilation

The source files needed for compiling your application along with the touch library are touch_api.h, touch_config.h and qt_asm_tiny_mega.S.

5.6.10.2.2 Example for ATSAM

The example below will explain in detail the steps to follow for library selection.

Criteria	Selection		Notes
Microcontroller	AT91SAM3S		
IDE and compiler tool chain used	IAR Workbench and GNU compiler		The GCC compiled variant of the libraries for the device selected needs to be used.
Number of Keys required for the application	3		Each key requires 1 QTouch acquisition channel
Rotors and sliders required	Yes		
Number of Rotors and Sliders required	3		Each rotor / slider will require 3 channels.
Number of Channels required for the application (should be the sum of all channels required for all the keys ,rotors and sliders used in the design)	12		3 Keys + (3 rotors x 3 channels per rotor/slider) → 12 channels
Charge cycle time required for the design	5 cycles		Assuming the device is configured with a clock frequency of 48Mhz
Number of SNS/SNSK port pairs needed	2 pairs		This is determined based on the free PIO of the board
Choice of ports available for the design	SNS/SNSK Pair1 port	SNS1 Port: A	The choice of ports for the port pairs is limited and can be found in the section 5.7.1.5
		SNSK1 Port: A	
	SNS/SNSK Pair 2 port	SNS2 Port: B	
		SNSK2 Port: B	

Given the above requirements for the applications, the first step is to select the right library variant required.

Step 1: Selecting the right library variant

Referring to the library selection guide, we see that there are a few variants of libraries supported for AT91SAM3S. One library is for IAR and the other is for GNU. If we want to use IAR Workbench, we use the library name: libsam3s-32qt-k-8rs-iar.a.

Step 2: Defining the constants / symbols in the project space
In IAR, change preprocessor options by adding the good defines:

```
_TOUCH_ARM_
_QTOUCH_
SNS1=B
SNSK1=B
SNS2=A
SNSK2=A
QT_NUM_CHANNELS=32
_ROTOR_SLIDER_
QT_DELAY_CYCLES=10
```

_SNS1_SNSK1_SAME_PORT_
_SNS2_SNSK2_SAME_PORT_

Step3: Usage of library API's

Now, you can use the touch API's to create, initialize and perform touch sensing.

5.6.10.2.3 Checklist of items for integrating QTouch acquisition method libraries

The following is a checklist of items which needs to be ensured when integrating QTouch acquisition method libraries

- The clock prescaler register (e.g. CLKPR, XDIV) needs to be configured correctly based on the device selected. Some devices have clock frequency selection based on fuses. It has to be ensured the fuses are set correctly in such cases.
- It is recommended to disable PULL-UP resistor on all port pins used for touch sensing on the device selected (e.g. PUD bit in MCUCR, SFIOR for a few of the tinyAVR and megaAVR devices Please refer to the Data sheet of the selected device).
- The 16 bit timer in each device has been used for performing touch measurements periodically. The datasheet for all the devices have to be checked to ensure that the correct timer peripheral and its registers are used (file: main.c).
- The interrupt vector macro may also change from device to device and this needs to be verified in the datasheet for the device used.
- Check if the timer is configured correctly to support the measurement period needed (e.g. 25msec or 50 msec).
- The sample applications for the evaluation kits and supported devices illustrate the proper initialization sequence and usage of the timer resources (file: main.c). Please use this as a reference for your application design.

The host application must provide the current time to the library. This information is passed to the library as an argument to the function qt_measure_sensors()". This is used for time-based library operations such as drift compensation.

5.6.10.3 Integrating QMatrix acquisition method libraries in your application

5.6.10.3.1 Example for 8bit AVR

Based on the application design needs, the user needs to select the right library variant and the configuration to be used along with the variant. This section illustrates the steps required to select the right QMatrix acquisition method library variant and configuration for your application. QMatrix acquisition method library Variants are offered for IAR and AVR-GCC tool chains. First step is to select the compiler tool chain for which the libraries are required. The list of supported compiler tool chains can be found in section 5.7.2.2

There are specific library variants distributed for each microcontroller. For your design, you would need the following information to select the correct library variant

a. Device to be used for the design
b. The number of touch sensing channels needed by the application – Then identify the Maximum number of channels required for the design that are supported by the library.
c. Number of X lines to be used in the design
    a. The ports on which your design permits to have the X lines
    b. The X lines can be spread on a maximum of three ports, the more ports used the more is the code memory requirement by the library.
d. Number of Y lines to be used in the design
    a. The port-pins ports on which your design permits to have the Y lines

e. Do you need support for Rotors and/or Sliders in your design
    a. If yes, how many rotors/sliders would be needed?
    b. Based on a) above, identify the maximum number of rotors sliders that the library supports
f. Which compiler platform you intend to use to integrate the libraries – IAR or AVR -GCC

Follow the steps listed below to arrive at the right library variant
1) Select the device from the list of supported devices listed in 5.7.2.4.1
2) Select the right library variant for the device selected from the selection guide available in
   C:\ Program Files\Atmel\Atmel_QTouch_Libaries_5.x\Library_Selection_Guide.xls
   Each variant supports
    a. a specific number of channels,
    b. Supports a specific configuration of X x Y matrix pins (eg 4 x 2 for 4 - X pins & 2 - Y pins)
    c. has support for Rotor / Slider (either supported or not)
    d. support is available for IAR and/or GCC compiler tool chain
    e. support for specific number of rotors sliders.
3) Define the constants and symbol names required
    a. The next step is to define certain constants and symbols required in the host application files where the touch API is going to be used. These values are derived from the parameters defined in step 2 for your application
    b. The constant/symbol names are as listed in the table below
    c. The constant/symbol definitions can be placed in any of the following
        i. In the user's 'C' file prior to include touch_api.h in the file
        ii. Modifying the defines in a touch_config.h available in the project folder

Table 4 :List of configurable parameters for touch library usage.

Symbol / Constant name	Range of values	Comments
QMATRIX	Symbol defined to indicate QMatrix acquisition method is required	Define this symbol to indicate QMatrix acquisition method is required
QT_NUM_CHANNELS	The number of channels the library supports.(Possible values:4,8,16,32,56,64). Note: 56 channel for only ATxmega Devices.	Value should be same as the number of channels that the library supports
NUM_X_LINES	The number of X lines the library supports.(Possible values:4,8)	Value should be same as the number of X lines that the library supports. Refer to library selection guide
NUM_Y_LINES	The number of Y lines the library supports.(Possible values:1,2,4,7,8) Note: 7 Y-lines for only ATxmega Devices)	Value should be same as the number of Y lines that the library supports. Refer to library selection guide
_ROTOR_SLIDER_	Symbol defined if Rotor and/or slider is required	Needs to be added in case user needs to configure ROTOR/SLIDER Needs to be removed for ALL KEYS configuration
QT_MAX_NUM_ROTORS_SLIDERS	Maximum number of rotors/sliders the library supports(possible values:0,2,4,8)	Subject to support for rotors/sliders in the library selected.
QT_DELAY_CYCLES	Possible values :1,2,3,4,5,10,25,50	Please refer to section 5.6.8
NUM_X_PORTS	Number of ports on which the X lines needs to be spread. (Possible values 1,2,3)	Maximum number of ports that the X lines can spread is 3. Note: Code memory required increases with the increase in NUM_X_PORTS
PORT_X_1	First IO port for configuring the X lines.Any IO port available with the device.	Drive electrode for touch sensing using QMatrix acquisition Valid when NUM_X_PORTS =1,2,3
PORT_NUM_1	1	Please donot edit this macro Valid when NUM_X_PORTS =1,2,3
PORT_X_2	Second IO port for configuring the X lines. Any IO port available with the device.	Drive electrode for touch sensing using QMatrix acquisition Valid when NUM_X_PORTS =2,3
PORT_NUM_2	2	Please donot edit this macro Valid when NUM_X_PORTS =2,3
PORT_X_3	Third IO port for configuring the X lines. Any IO port available with the device.	Drive electrode for touch sensing using QMatrix acquisition Valid when NUM_X_PORTS =3
PORT_NUM_3	3	Please donot edit this macro Valid when NUM_X_PORTS =3
PORT_YA	Any IO port available with the device.	Receive electrode for touch sensing using QMatrix acquisition
PORT_YB	ADC port available for the device.	Receive electrode for touch sensing using QMatrix acquisition
PORT_SMP	Any IO port available with the device.	Port of the Sampling pin for touch sensing using QMatrix acquisition
SMP_PIN	Any IO port available with the device.	Sampling pin for touch sensing using QMatrix acquisition
ATXMEGA	Symbol defined if an ATxmega Device is used for QMatrix sensing technology	Needs to be added if the device to be supported is ATxmegaxxxx

SHARED_YAYB	Possible values: 0 or 1.	#define SHARED_YAYB 1 in case YA and YB are on same port else 0.

Once you have selected the right library variant and configuration parameters for the application, follow the steps outlined below to integrate the library variant in your application.

4) Fill in the arrays x_line_info_t x_line_info[NUM_X_LINES] y_line_info_t ya_line_info[NUM_Y_LINES] and y_line_info_t yb_line_info[NUM_Y_LINES] using the pin configuration wizard provided by the QTouch Studio.
5) Copy the library variant that was selected in step one to your project's working directory or update your project to point to the library selected.
   Include the "touch_api.h" header file and assembler source file from the QTouch library in your application. The touch_api.h can be found in the release package at C:\Program Files\Atmel\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QMatrix\common_files. The assembler files mentioned below could be found at the location C:\Program Files\Atmel\Atmel_QTouch_Libraries_5.x\ Generic_QTouch_Libraries \AVR_Tiny_Mega_XMega\QMatrix\common_ files
   a. qm_asm_tiny_mega.S in case of ATtiny and ATmega devices.
   b. qm_asm_xmega.S in case of ATxmega devices.
   c. qm_asm_m8535_m16.S in case of ATmega8535 and ATmega16 devices.

6) Initialize/create and use the touch api's in your application
   a. In your application, create, initialize and configure the sensors.
      a. The APIs of interest are qt_enable_key/rotor/slider().see sections 5.6.5.2, 5.6.5.3 and 5.6.5.4
   b. configure the global configuration parameters valid for all the sensors in the library
   c. Provide timing for the QTouch libraries to operate. i,e. the QTouch libraries do not use any timer resources of the microcontroller. The Host application has to provide the required timing and also call the API's at the appropriate intervals to perform touch sense detect operations
7) General application notes
   a. The clock, host application and other peripherals needed by the host application needs to be initialized.
   b. The QMatrix acquisition method libraries internally use TIMER1 for their operation.
   c. Ensure that there are no conflicts between the resources used by the touch library and the host application
   d. Ensure that the stack size is adjusted to factor in the stack depth required for the operation of the touch libraries.

5.6.10.3.1.1 Example

The example below will explain in detail the steps to follow for library selection.

Criteria	Selection	Notes
Microcontroller	ATTiny88	List of supported devices can be found at Library_Selection_Guide.xls
Number of channels required for the application	6	number channels available for a Tiny88 is listed in Library_Selection_Guide.xls
Number of X lines	Based on the number of channels,	Since 3 X- lines (6 channels) are

needed	since 8 channels is needed, 4 X lines are supported. NUM_X_LINES is 4	used, Do not initialize 4th element in x_line_info[NUM_X_LINES]. Hence channel6, channel7 need not be used.
Number of Y lines needed	Based on the number of channels, since 8 channels is needed, 2 Y lines are supported	NUM_Y_LINES is 2.
Rotors and sliders required and Number of ROTOR/SLIDERS	Yes 2	Library variants supported for ATTiny88 is listed in the Library_Selection_Guide.xls

X_LINES on pins as below(4-X lines) X0- B0, X1- D2,X3 – B7, X4 – B5	`FILL_OUT_X_LINE_INFO(1,0),`
	`FILL_OUT_X_LINE_INFO(2,2),`
	`FILL_OUT_X_LINE_INFO(1,7),`
		Main file has to be edited based on the configuration. Refer to section 5.8.2.1 . Refer channel numbering from the section5.6.6.1.2
		Or
		This can be filled from the output of the pin configurator tool in QTouch Studio. Please refer to section 5.8.2

Y_LINES on pins as below (2 Y-Lines)
Y0A- D0, Y0B- C1, Y1A-D5, Y1B-C4,
	`FILL_OUT_YA_LINE_INFO(0),`
	`FILL_OUT_YA_LINE_INFO(5),`
	
	`FILL_OUT_YB_LINE_INFO(1),`
	`FILL_OUT_YB_LINE_INFO(4),`
		Main file has to be edited based on the configuration. Refer to section 5.8.2.1
		Or
		This can be filled from the output of the pin configurator tool in QTouch Studio. Please refer to section 5.8.2

NUM_X_PORTS	2	Since X lines are spread on a multiple(2) ports: PORTB, PORTD
Compiler tool chain	IAR	Supported compiler tool chains listed in 5.7.2.2
Choice of ports available for the design	PORT_X_1 = B PORT_X_2 = D YA Line on PORTD YB Line on PORTC	Any pins that are not conflicting with the host application and follow the configuration supported by library can be used.

	SMP Pin on PORTD pin 7 QT_DELAY_CYCLES of 4	Or This can be filled from the output of the pin configurator tool in QTouch Studio. Please refer to section 5.8.2
Choice of Shared Ya and Yb on same port	SHARED_YAYB	This should be defined as 0 if YA and YB not shared on same port else 1 if shared on same port.

Given the above requirements for the applications, the first step is to select the right library variant required.

Step 1:
Select the Device that suits the requirements based on the touch sensing channels needed from the library selection guide available at C:\ Program Files\Atmel\ Atmel_QTouch_Libaries_5.x\ Library_Selection_Guide.xls

Step 2:
From the Library_selection_Guide.xls list,, we see that there are a few variants of libraries supported for AT Tiny device. Since the application requires 6 channels and rotor slider support, one has to select a library variant which supports at least 6 channels or more. Hence we select the 8 channel library which supports the required Port combination and the delay cycle preferred which works out to be the variant

- libv1g1s1_8qm_4x_2y_krs_2rs.r90

Step 3:
Defining the constants / symbols in the project space or modifying in touch_config.h
In the host application file (say main.c), define the following constants and symbols

```
#define _QMATRIX_
#define QT_NUM_CHANNELS          8
#define NUM_X_LINES              4
#define NUM_Y_LINES              2
#define NUM_X_PORTS              2
#define PORT_X_1                 B
#define PORT_NUM_1               1
#define PORT_X_2                 D
#define PORT_NUM_2               2
#define PORT_YA                  D
#define PORT_YB                  C
#define PORT_SMP                 D
#define SMP_PIN                  7
#define QT_DELAY_CYCLES          4
#define ROTOR_SLIDER_
#define QT_MAX_NUM_ROTORS_SLIDERS     2
#define SHARED_YAYB              0
```

NOTE: The above definitions are available in touch_config.h *file. Alternatively, you can define these in your IDE's project options or have them defined in a separate header file.*

Note:

1. Some of these macro's can be taken from the output of the Pin configurator tool from QTouch Studio. Refer to section 5.8.2

2. These can also be modified in the touch_config.h, after defining the _QMATRIX_ in the project space.
3. In case XMEGA device is used for QMatrix the symbol __ATXMEGA_ has to be included in the Project space along with the symbols mentioned above.

Step 4:
Filling Arrays in the main.c file
According to the pin availability for the touch sensing, initialize the arrays in the main.c file as below:

```
x_line_info_t x_line_info[NUM_X_LINES]= {
    FILL_OUT_X_LINE_INFO(  1,0u ),
    FILL_OUT_X_LINE_INFO(  2,2u ),
    FILL_OUT_X_LINE_INFO(  1,7u ),
};

y_line_info_t ya_line_info[NUM_Y_LINES]= {
    FILL_OUT_YA_LINE_INFO(  0u ),
    FILL_OUT_YA_LINE_INFO(  5u ),
};
y_line_info_t yb_line_info[NUM_Y_LINES]= {
    FILL_OUT_YB_LINE_INFO(  0u ),
    FILL_OUT_YB_LINE_INFO(  5u ),
};
```

Note:

1. This part of the snippet can be taken from the output of the Pin configurator tool from QTouch Studio.

Step 5:
Usage of libraries
Now, you can use the touch API's to create, initialize and perform touch sensing. Please refer to the sample applications in section 5.6.11.3 for reference. These sample applications illustrate the usage of the API's and the sequence of operation

5.6.10.3.1.2 Resources used by QMatrix acquisition method libraries

The following additional resources are used by the QMatrix acquisition method libraries.

- One Analog Comparator
- One internal Timer (Usually Timer1 depending on the availability on particular microcontroller)
- One ADC Multiplexer(The critical section of the touch sensing library disables the use of ADC as conversion unit and enables the same ADC as a multiplexer, but the user can use the ADC for conversion in rest of his application code)
- The ADCMUX is used by the library during the touch sensing acquisition, however it is restored with the value from host application before exiting the qt_measure_sensors(). Such that the ADC is available to the host application for conversion.

In case of XMega devices, the resources are used internal to the library and hence cannot be used by the host application

- o Analog Comparator0 on PORTA (AC0 on PORTA)
- o Timer/Counter1 on PORTC (TCC1)
- o Event System Channel0 (EVSYS_CH0)

5.6.10.3.2 Example for 32bit AVR

Based on the application design, the user needs to select the right library variant and the configuration to be used along with the variant. This section illustrates the steps required to select the right QMatrix acquisition method library variant and configuration for your application.

For your design, you would need the following information to select the correct library variant

a. Device to be used for the design(only AT32UC3C0512 supported)
b. The number of touch sensing channels needed by the application – Then identify the Maximum number of channels required for the design that are supported by the library.
c. Number of X lines to be used in the design
    a. The port on which your design permits to have the X lines
    b. The X lines can be spread on a single port.
d. Number of Y lines to be used in the design
    c. The port-pins ports on which your design permits to have the Y lines
e. Do you need support for Rotors and/or Sliders in your design
    d. If yes, how many rotors/sliders would be needed?
    e. Based on a) above, identify the maximum number of rotors sliders that the library supports
f. Which compiler platform you intend to use to integrate the libraries – IAR or AVR -GCC

After selecting the right library variant, following steps are to be performed

1) Define the constants and symbol names required
    a. The next step is to define certain constants and symbols required in the host application files where the touch API is going to be used. These values are derived from the parameters defined in step 2 for your application
    b. The constant/symbol names are as listed in the table below
    c. The constant/symbol definitions can be placed in any of the following
        iii. In the user's 'C' file prior to include touch_api.h in the file
        iv. Defined user's project options.
        v. Modify the defines in a touch_config.h

Symbol / Constant name	**Range of values**	**Comments**
QMATRIX	Symbol defined to indicate QMatrix acquisition method is required	Define this symbol to indicate QMatrix acquisition method is required
QT_NUM_CHANNELS	The number of channels the library supports.(Possible values:4,8,16,24,32,64).	Value should be same as the number of channels that the library supports
NUM_X_LINES	The number of X lines the library supports.(Possible values:4,8)	Value should be same as the number of X lines that the library supports. Refer to library selection guide
NUM_Y_LINES	The number of Y lines the library supports.(Possible values:1,2,3,4,8)	Value should be same as the number of Y lines that the library supports. Refer to library selection guide
_ROTOR_SLIDER_	Symbol defined if Rotor	Needs to be added in case user

	and/or slider is required	needs to configure ROTOR/SLIDER Needs to be removed for ALL KEYS configuration
QT_MAX_NUM_ROTORS_SLIDERS	Maximum number of rotors/sliders the library supports(possible values:0,2,3,4,8)	Subject to support for rotors/sliders in the library selected.
QT_DELAY_CYCLES	Possible values :1,2,3,4,5,10,25,50	Please refer to section 5.6.8
PORT_X_1	First IO port for configuring the X lines.Any IO port available with the device.	Drive electrode for touch sensing using QMatrix acquisition
PORT_YA	Any IO port available with the device.	Receive electrode for touch sensing using QMatrix acquisition
PORT_YB	Analog Comparator port available for the device.	Receive electrode for touch sensing using QMatrix acquisition
PORT_SMP	Any IO port available with the device.	Port of the Sampling pin for touch sensing using QMatrix acquisition
SMP_PIN	Any IO port available with the device.	Sampling pin for touch sensing using QMatrix acquisition

Once you have selected the right library variant and configuration parameters for the application, follow the steps outlined below to integrate the library variant in your application.

1) Fill in the arrays x_line_info_t x_line_info[NUM_X_LINES] y_line_info_t ya_line_info[NUM_Y_LINES] and y_line_info_t yb_line_info[NUM_Y_LINES] as given in main.c file.

   Filling Arrays in the main.c file
   According to the pin availability for the touch sensing, initialize the arrays in the main.c file as below:

```
x_line_info_t x_line_info[NUM_X_LINES]= {
   FILL_OUT_X_LINE_INFO(  1,0u ),
   FILL_OUT_X_LINE_INFO(  1,2u ),
   FILL_OUT_X_LINE_INFO(  1,7u ),
   FILL_OUT_X_LINE_INFO(  1,15u ),
};
```

First argument of FILL_OUT_X_LINE_INFO should always be 1 as X port is only on one port.Second arguments denotes the pins on that particular port.

```
y_line_info_t ya_line_info[NUM_Y_LINES]= {
   FILL_OUT_YA_LINE_INFO(  0u ),
   FILL_OUT_YA_LINE_INFO(  5u ),
};

y_line_info_t yb_line_info[NUM_Y_LINES]= {
   FILL_OUT_YB_LINE_INFO(  7u ),
   FILL_OUT_YB_LINE_INFO(  22u ),
};
```

Yb lines are one of the inputs of the Analog Comparators.

2) Copy the library variant that was selected in step 1 to your project's working directory or update your project to point to the library selected.
   Include the "touch_api.h" header file and assembler source file from the QTouch library in your application. The touch_api.h can be found in the release package at C:\Program Files\Atmel\Atmel_QTouch_Libraries_5.x\Generic_QTouch_Libraries\include. The assembler files mentioned below could be found at the location C:\Program Files\Atmel\Atmel_QTouch_Libraries_5.x\Generic_QTouch_Libraries\32bit_AVR\UC3\Q Matrix\common_files
   e. qm_asm_uc3c_gcc.x in case of GCC compiler
   f. qm_asm_uc3c_iar.s82 in case of IAR compiler.

3) Initialize/create and use the touch api's in your application
   d. In your application, create, initialize and configure the sensors.
      a. The APIs of interest are qt_enable_key/rotor/slider().see sections 5.6.5.2, 5.6.5.3 and 5.6.5.4
   e. configure the global configuration parameters valid for all the sensors in the library
   f. Provide timing for the QTouch libraries to operate. i,e. the QTouch libraries do not use any timer resources of the microcontroller. The Host application has to provide the required timing and also call the API's at the appropriate intervals to perform touch sense detect operations
4) General application notes
   g. The clock, host application and other peripherals needed by the host application needs to be initialized.
   h. The QMatrix acquisition method libraries for 32 Bit devices internally use TIMER0 with channel0 for their operation.
   i. Ensure that there are no conflicts between the resources used by the touch library and the host application

5.6.10.3.2.1 Resources used by QMatrix acquisition method libraries for 32 Bit device

Devices supported by 32 Bit Qmatrix Acquisition libraries are:

1. AT32UC3C0512

The following additional resources are used by the QMatrix acquisition method libraries.

- Four Analog Comparator
- One internal Timer (Timer0 with channel0)
- Two Analog Comparator Interface ACIFA0/1.
- Event System Channel 16 is used.

The device has two Analog comparator interfaces ACIFA0 and ACIFA1 .Each interface provides the flexibility to configure two analog comparators ACA and ACB comparators..UC3C has Four Comparators (AC0A , AC1A , AC0B , AC1B), So there are 10 Possible Yb lines as given in table below.

User has flexibility to configure maximum 8 Yb lines for maximum 64 channel libraries.Below table states the Yblines which can be configured

Yb LInes of the Four	Port A

Analog Comparators	Pins
AC0AN0(AC0A Comparator)	PA22
AC0AN1(AC0A Comparator)	PA27
AC0BP0(AC0A Comparator)	PA23
AC1AN0(AC1A Comparator)	PA13
AC1AN1(AC1A Comparator)	PA07
AC1BP0(AC1A Comparator)	PA14
AC0BN0(AC0B Comparator)	PA21
AC0BN1(AC0B Comparator)	PA29
AC1BN0(AC1B Comparator)	PA15
AC1BN1(AC1B Comparator)	PA09

These Yb lines are the negative input pins of the analog comparator.To use the library, the below table shows the lines which are externally grounded for Proper Qmatrix operation

Positive Input Pins of the Four Analog Comparators	Port A Pins
AC0AP0(AC0A Comparator)	PA20
AC0BP1(AC0B Comparator)	PA28
AC1AP0(AC1A Comparator)	PA12
AC1BP1(AC1B Comparator)	PA08

If only one comparator is used , then the corresponding pin of that comparator is properly externally grounded.Above table shows the Analog comparator usage and the corresponding lines which needs to be grounded.

The Port pin configurability is provided in the library to select any port for Ya or X lines.The Port for Yb lines is fixed which is PORTA as these lines are inputs of the Analog comparators which are fixed pins of PORTA.

X,YA,YB,SMP Configurations	Ports on UC3C			
	PORTA	PORTB	PORTC	PORTD
X	Yes	Yes	Yes	Yes
YA	Yes	Yes	Yes	Yes
YB	Yes(only few pins of PortA can be configured as Yb Lines)	No	No	No
SMP	Yes	Yes	Yes	Yes

The configurability is provided to select X,SMP and YA lines on Same Port ,X,SMP and YB lines on same port.
Number of X ports should be 1 means the X lines should be connected to a single port.

Note: YA and YB cannot be on the same port.
SMP Pin should be less than equal to 19^{th} pin of any Port.

5.6.10.3.3 Checklist of items for integrating QMatrix Capacitive sensing libraries

When integrating QMatrix acquisition method libraries, ensure the following

- Check that the CLKPR register is available for the selected device. If not remove the CLKPR statements.
- Ensure that the configuration for the QMatrix is done in touch_config.h and the arrays of the x_line_info and y_line info are filled as indicated section 5.8.2
- MCUCR register is available and if so disable pullups
- Check if the timer registers and bit fields used are correct and change them if necessary.
- The above settings can be modified by the user by changing the API's that are available to the user. The API's include
  - *qt_set_parameters ( )*
- The host application must provide the current time.
  This information is passed to the library as an argument to the function *qt_measure_sensors()".* This is used for time-based library operations such as drift compensation.
- The GPIO internal pull-ups must be disabled for all port pins used for touch sensing when calling the library.
  For 8-bit AVR devices, this can be done by
    b. Setting the "PUD" bit in the "MCUCR" register or
    c. Setting the "PUD" bit in the "SFIOR" register.
- Setting the JTD bit in the "MCUCR" register to disable JTAG Interface in MCU (if available). This can be done only when the JTAG lines are in conflict with the desired touch sensing lines.
- The library must be called often enough to provide a reasonable response time to user touches. The typical time to call the library is from 25 ms to 50 ms.
- Care should be taken while using the ADC conversion logic and QMatrix library such that the host application waits for approximately 1msec before actually calling the qt_measure_sensors() API depending upon the ADC clock.

5.6.10.4 Common checklist items

5.6.10.4.1 Configuring the stack size for the application

The stack requirements for the QTouch library should be accounted for and the stack size adjusted in the user's project for proper operation of the software when using the IAR IDE. This section lists the stack usage for the different variants of the QTouch and QMatrix acquisition method libraries applicable to the IAR compiler tool chain.

Note: When using the IAR IDE / compiler tool chain, the map file generated for the application will list total CSTACK & RSTACK requirements. Please adjust the total CSTACK and RSTACK values in the IAR project options to be greater than the values listed in the map file. Refer to section 5.6.11.4 which illustrates how to change the settings in IAR IDE.

Table 5 : Stack requirements of the QTouch capacitive sensing libraries when using IAR IDE projects

QTouch Acquisition method Libraries : Stack usage for IAR compiler tool chain		
Configuration	**CSTACK size**	**RSTACK size**
Single port pair - only keys (4 / 8 channels)	0x30	0x28
Single port pair – keys/ rotors/ sliders (4/8 channel)	0x40	0x2C
Two port pairs - only keys keys (16 channel)	0x50	0x28
Two port pairs – keys/ rotors/ sliders (16 channel)	0x60	0x2C

Table 6 : Stack requirements of the QMatrix capacitive sensing libraries when using IAR IDE projects

QMatrix Acquisition method Libraries : Stack usage for IAR compiler tool chain			
Number of channels	**Configuration**	**CSTACK size**	**RSTACK size**
4	ONLY KEYS	0x20	0x20
4	KEYS/ROTOR/SLIDER	0x30	0x20
8	ONLY KEYS	0x25	0x20
8	KEYS/ROTOR/SLIDER	0x35	0x20
16	ONLY KEYS	0x30	0x20
16	KEYS/ROTOR/SLIDER	0x40	0x20
32	ONLY KEYS	0x35	0x25
32	KEYS/ROTOR/SLIDER	0x45	0x25
56	ONLY KEYS	0x45	0x25
56	KEYS/ROTOR/SLIDER	0x55	0x25
64	ONLY KEYS	0x45	0x25
64	KEYS/ROTOR/SLIDER	0x55	0x25

5.6.11 Example project files

The QTouch library is shipped with various example projects to illustrate the usage of the touch API's to add touch sensing to an application across various devices

Sample applications are also provided for the following kits

1 TS2080A, QT600_ATtiny88_QT8, QT600_ATxmega128a1_QT16 : QTouch Technology evaluation Kits
2 TS2080B, QT600_ATmega324_QM64 : QMatrix Technology evaluation Kits

Note: Example projects must be built in the installed folder, and if moved/copied elsewhere then paths must be edited appropriately.

5.6.11.1 Using the Sample projects

The sample applications are shipped with the complete set of files required to configure, build and download the application for both IAR-workbench and AVR Studio IDE.

Since more than one device may use the same library (applicable for QTouch acquisition method libraries), example project files and applications have been provided only for select devices which use these libraries.

5.6.11.2 Example applications for QTouch acquisition method libraries

5.6.11.2.1 Selecting the right configuration

Each example project for a device can support multiple configurations (i.e a. keys only, b. with rotors and sliders c.16 channel etc…). The configuration sets determine the configuration options for using the library and also the right library variant to link with the project.

The configuration sets for IAR IDE are named according to the convention listed below

Configuration set for IAR IDE **Naming convention** : <**vP**>g<**Q**>_<**CH**>qt_k_<**RS**>rs		
Field Name	**Values**	**Comments**
vP	v1, v3, xmega, uc3a, uc3b, uc3c	VersionP of the core AVR device supported by this library variant
Q	1 to 6	GroupQ of the core AVR device supported by this library variant
CH	4, 8, 12, 16, 32	Total number of channels supported by each library.
RS	1, 2, 3, 4, 8	Total number of rotors / sliders supported for the respective channel counts mentioned in previous row.

The configuration sets for AVR Studio IDE are named according to the convention listed below

Configuration set for AVR Studio IDE <**avrP**>g<**Q**>_<**CH**>qt_k_<**RS**>rs		
Field Name	**Values**	**Comments**
avrP	avr25, avr4, avr 51, avr5, xmega, uc3a, uc3b, uc3c	VersionP of the core AVR device supported by this library variant
Q	1 to 6	GroupQ of the core AVR device supported by this library variant
CH	4, 8, 12, 16, 32	Total number of channels supported by each library.
RS	1, 2, 3, 4, 8	Total number of rotors / sliders supported for the respective channel counts mentioned in previous row.

Depending on your need, you need to select the right configuration required and build the project.

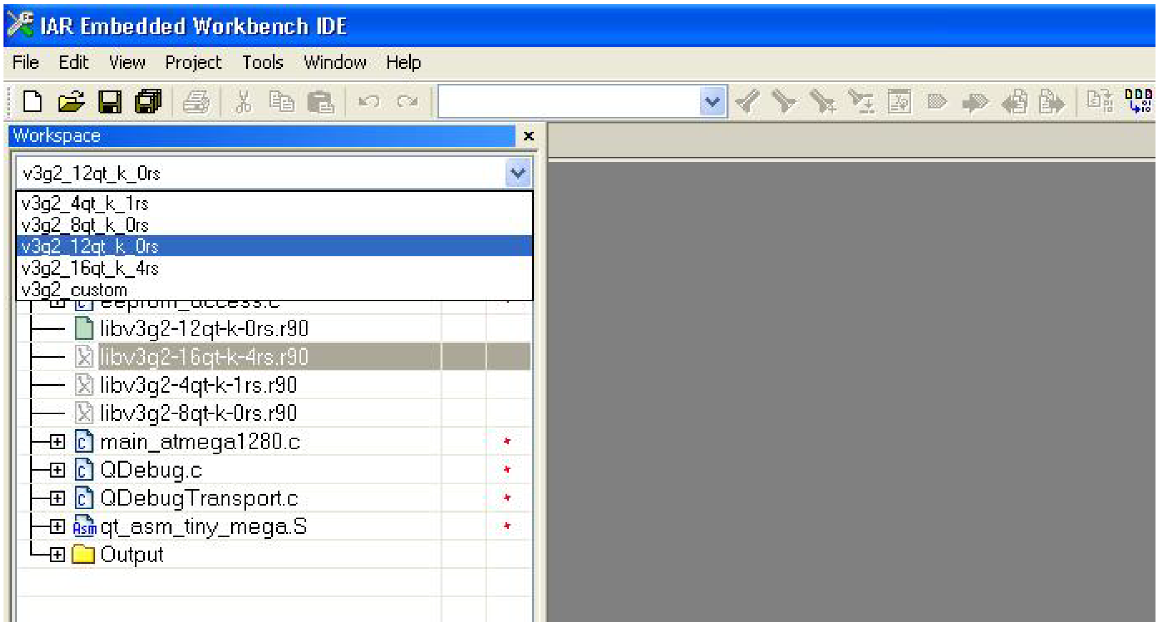


Figure 5-11: Selecting the right configuration in the QTouch acquisition method example applications in IAR –IDE

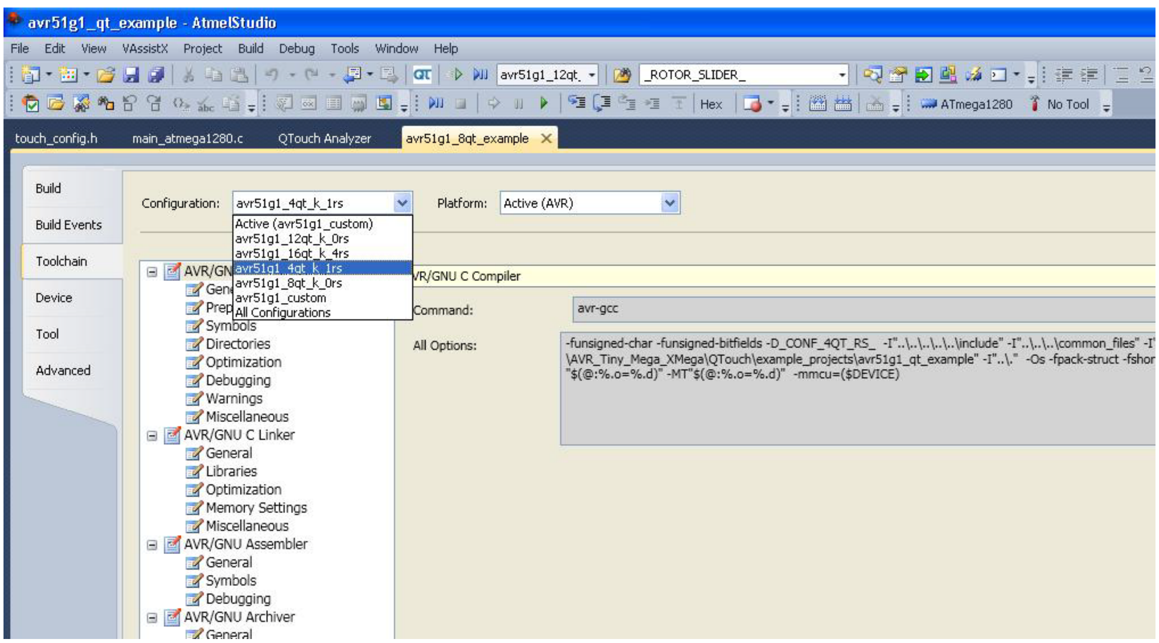


Figure 5-12 : Selecting the right configuration in QTouch acquisition method example applications in AVR-6 IDE

5.6.11.2.2 Changing the settings to match your device

5.6.11.2.2.1 Processor settings

Once you have selected the appropriate example project and the configuration, you need to ensure that the settings in the project are configured to reflect the correct device. The settings include

- Device type (CPU type) for the project

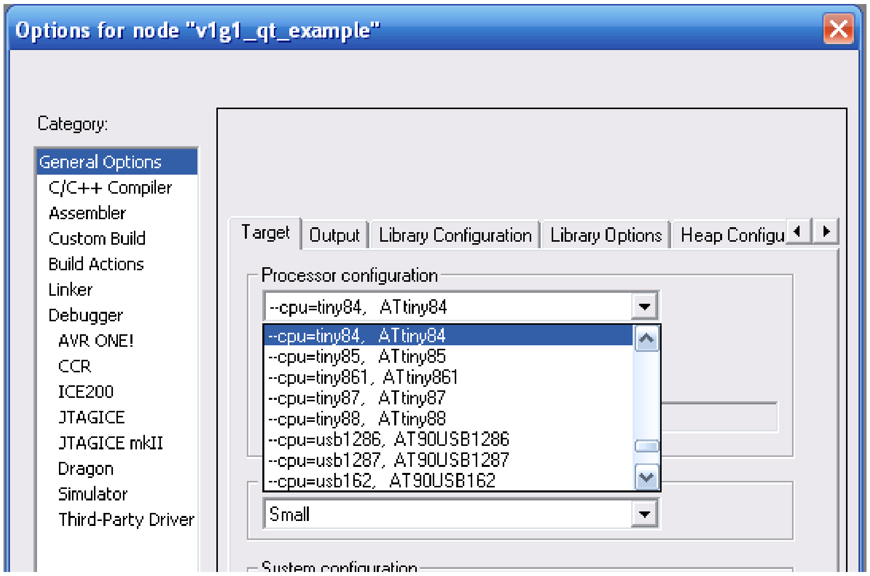


Figure 5-13 : Changing the processor settings for the examples in IAR IDE

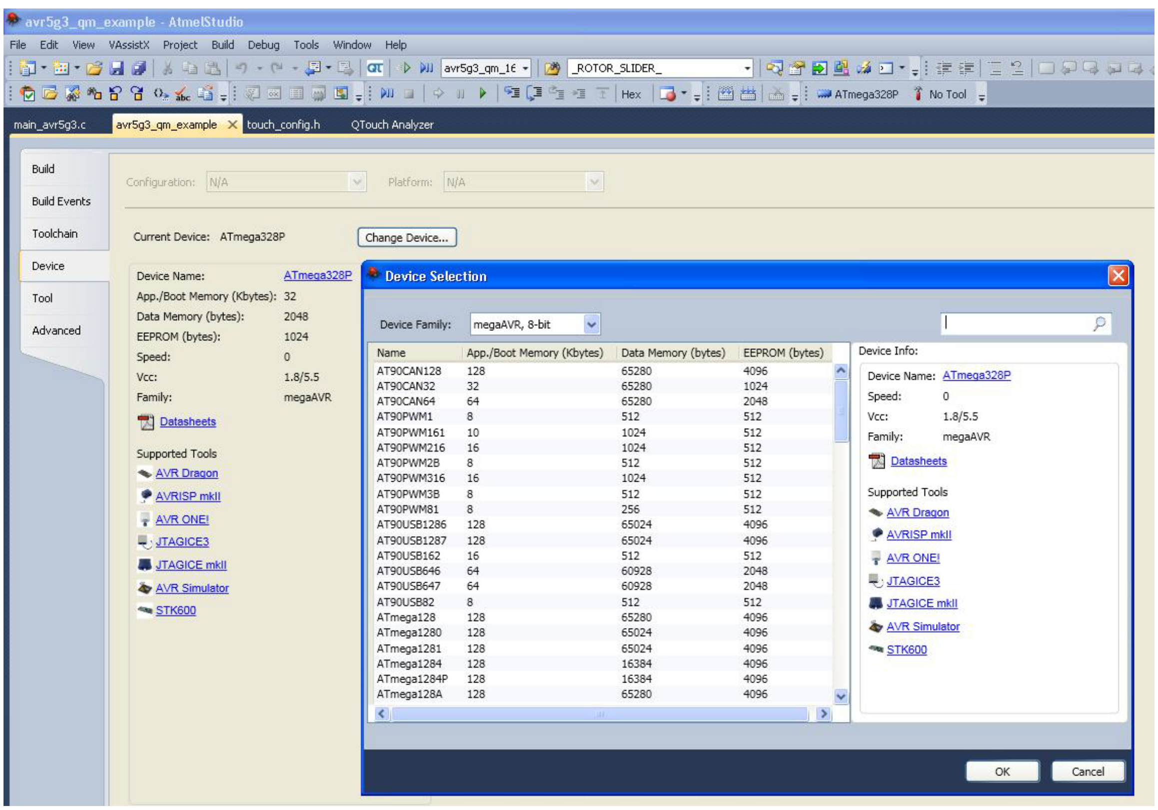


Figure 5-14 : Changing the processor settings for the examples in AVR-Studio 6

5.6.11.2.3 Changing the library configuration parameters

The configuration parameters required for the library are specified in the touch_config.h file of the examples under the custom user configuration section. Please refer to the example projects provided with the QTouch libraries release for more information. The mandatory constants to be defined are as listed below.

Symbol / Constant name	Range of values	Comments
QTOUCH	This macro has to be defined in order to use QTouch libraries.	
SNS & SNSK	Section 5.7.1.5 provides details on the range of values allowed.	To be used if only single port pair is needed for the design
SNS1 – SNSK1 & SNS2 – SNSK2	Section 5.7.1.5.2 has details on the range of values allowed	To be used if two port pairs are needed for the design
QT_NUM_CHANNELS	4, 8, 12, 16 for tinyAVR, megaAVR and XMEGA device libraries and 8, 16, 32 for UC3 device libraries.	
_ROTOR_SLIDER_	Rotor / slider can be added to the design, if this symbol is defined	A library with rotor / slider functionality already available needs to be selected if this macro is to be enabled
_DEBUG_INTERFACE_	The debug interface code in the example application will be enabled if this macro is enabled.	This will enable the application to output QTouch measurement values to GPIO pins, which can be used by a USB bridge to view the output on Hawkeye or QTouch Studio. This feature is currently supported by EVK/TS 2080A and QT600 boards.
QT_DELAY_CYCLES	1 to 255	Please refer to section
QTOUCH_STUDIO_MASKS	This macro needs to be defined if QTouch Studio Pin Configurator Wizard.is used to generate the SNS and SNSK masks.	Please refer to section 5.8.1
_STATIC_PORT_PIN_CONF_	This macro needs to be defined only in case of 4 and 8 channel libraries with interport configuration and pin configurability.	Please refer to section 5.8.1

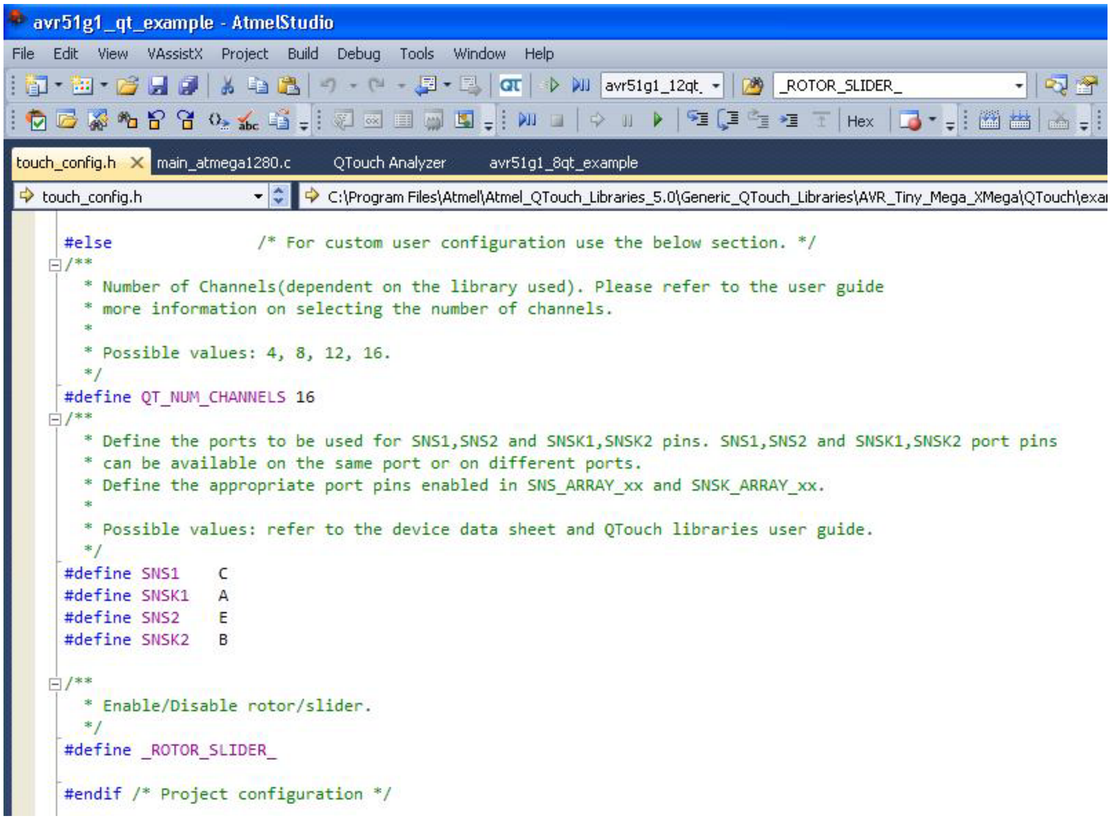


Figure 5-15 : Specifying the QTouch acquisition method library configuration parameters for QTouch example projects

5.6.11.2.4 Using the example projects

The sample applications are shipped with the complete set of files required to configure, build, execute and test the application for both IAR-workbench and AVR Studio IDEs.

The sample applications are provided for the evaluation kits and a few configurations for select devices. The user can use the sample applications as a reference or baseline to configure different configurations. Please ensure to change the configuration settings in the project options to match the device selected.

To change the configuration settings of the sample applications,

1. Select the configuration from the list of configurations available.
2. If the user wishes to have a new name for the configuration to be used, a new configuration can be added to the project.
3. If a different variant of the library needs to be used, remove the existing library in that particular configuration and add the library variant that you require. Please refer to 5.7.1.4 for details on the different library variants. Update the linker options to specify the library to be linked.
4. Specify the tunable configuration parameters for the project as illustrated in sections 5.6.11.2.2 and 5.6.11.2.3.

5.6.11.3 Example applications for QMatrix acquisition method libraries

The QMatrix acquisition method libraries include example projects for some of the supported devices. Example projects for both IAR IDE and AVR Studio IDE along with example applications are provided for select devices using the QMatrix acquisition libraries. These sample applications demonstrate the usage of the touch API's to add touch sensing to an application. Refer to the library selection guide for details on the example projects and sample applications supported for the release.

5.6.11.3.1 Selecting the right configuration

The sample applications are built to support a maximum channel support configuration available for that particular device for both IAR & AVR IDEs.

Internally there are two configurations for each device.

- ALL KEYS configuration : Supports only keys
- KEYS/ROTORS/SLIDERS configuration : Supports keys or rotors or sliders concurrently

These configurations enable a set of stored options and a specific library to be selected in order to build application using the specific library.

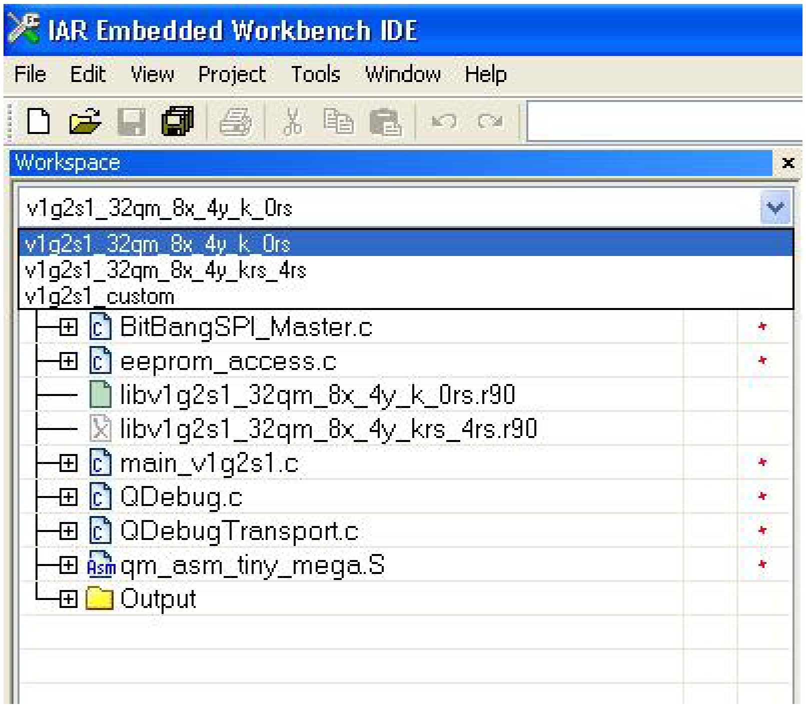


Figure 5-16 : Selecting the right configuration in the QMatrix acquisition method example applications in IAR –IDE

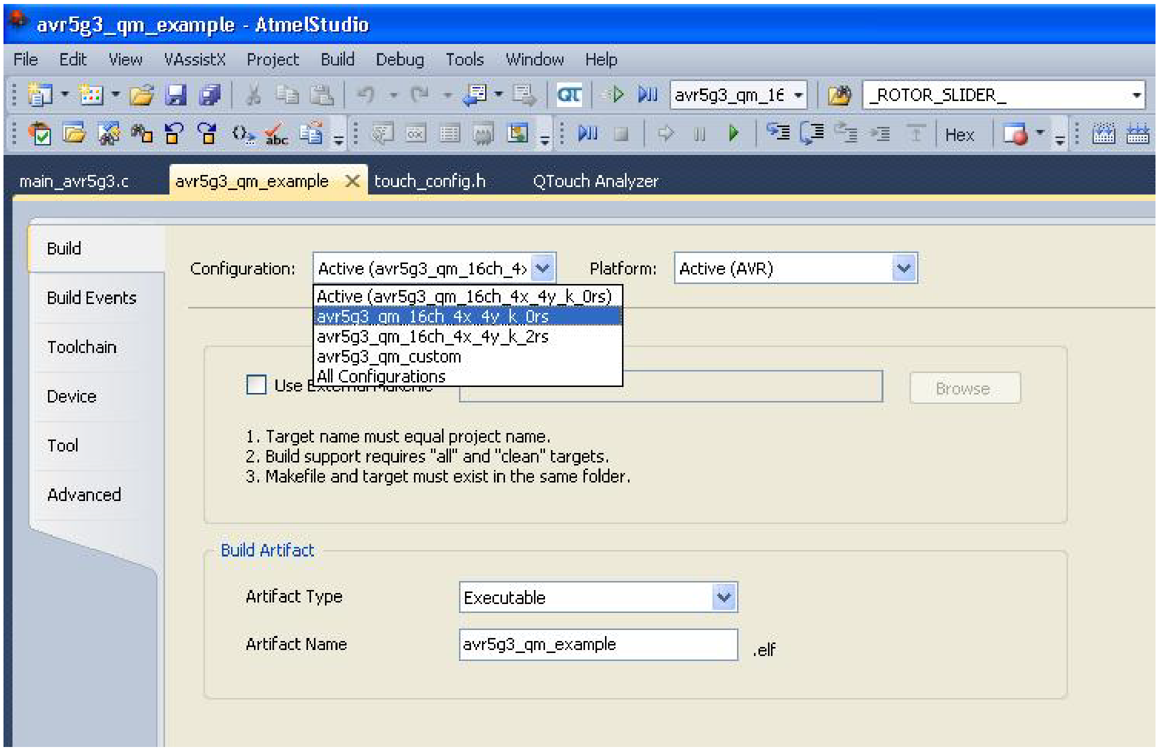


Figure 5-17 : Selecting the right configuration in the QMatrix acquisition method example applications in AVR Studio IDE

5.6.11.3.2 Changing the library configuration parameters

The configuration parameters required for the library are specified in the touch_config.h file of the examples. Please refer to the example projects provided with the QTouch libraries release for more information.

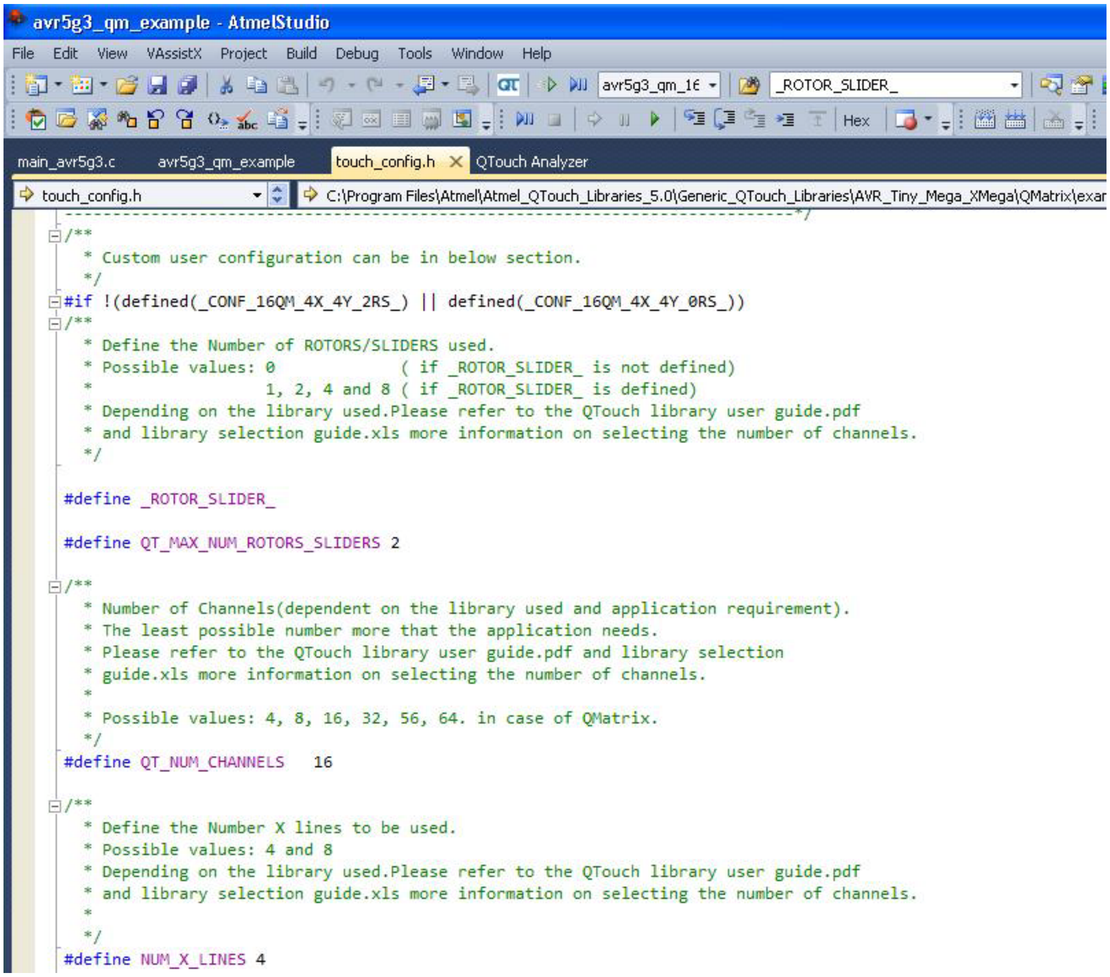


Figure 5-18 : Specifying QMatrix acquisition library parameters in touch_config.h for QMatrix projects

5.6.11.3.3 Using the example projects

The sample applications are shipped with the complete set of files required to configure, build, execute and test the application for both IAR-workbench and AVR Studio IDEs.

The sample applications are provided for the evaluation kits and a few configurations for select devices.

The user can use the sample applications as a reference or baseline to configure different configurations. Please ensure to change the configuration settings in the project options to match the device selected

To change the configuration settings of the sample applications,

1) Select the configuration from the list of configurations available as shown in section 5.6.11.3.1
2) If the user wishes to have a new name for the configuration to be used, a new configuration can be added to the project
3) If a different variant of the library needs to be used, remove the existing library in that particular configuration and add the library variant that you require. Please refer to library

selection guide for details on the different library variants. Update the linker options to specify the library to be linked

4) Specify the tunable configuration parameters for the project as illustrated in 5.6.11.3.2
5) For QMatrix on XMEGA devices, please check if the pre-processor symbol *_ATXMEGA_* is added in the project space or not.

5.6.11.4 Adjusting the Stack size when using IAR IDE

The example projects for IAR IDE, the CSTACK and RSTACK values are configured to account for the requirements of the QTouch libraries and the included main.c file which illustrates the usage of the touch API.

- Adjust the CSTACK and RSTACK values appropriately based on additional software integrated or added to the examples.

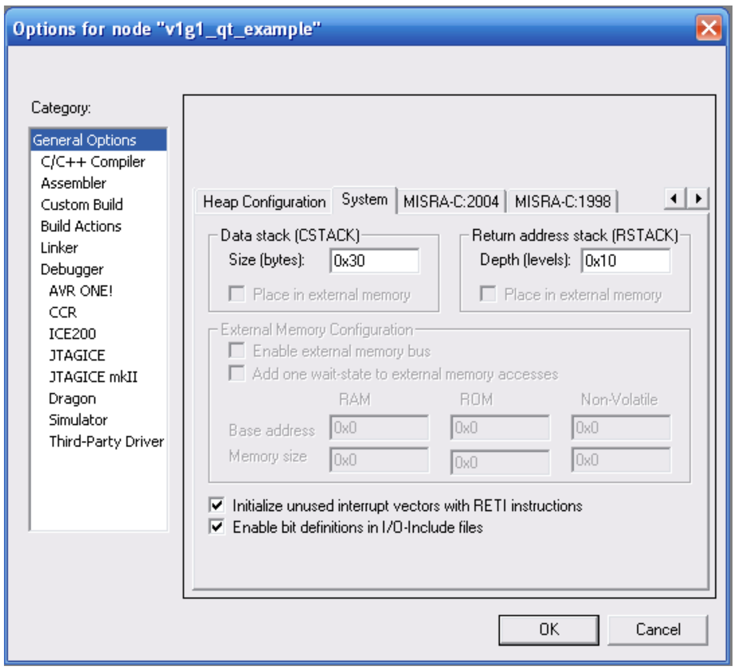


Figure 5-19 : Modifying the stack size in IAR IDE

5.6.11.5 Optimization levels

The default configuration settings in sample projects which ship with the library are set to the highest level of optimization for IAR and GCC variants of the libraries. The user might be required to change this setting for debugging purposes

- In case of IAR, The optimizations tab in project configuration options specifies High.
- In case of GCC, the libraries are compiled with the –Os which signifies that the Optimization for generating the library is maximum.

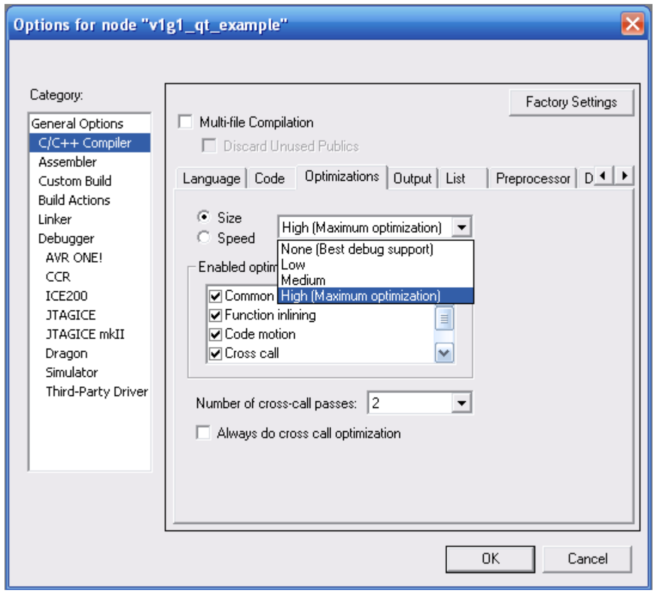


Figure 5-20 : Specifying the optimization level in IAR IDE

5.6.11.6 Debug Support in Example applications

The EVK2080 and QT600 applications provide output debug information on standard GPIO pins through the USB Bridge IC to PC software for display by AVR QTouch Studio. Similarly for ATMEL devices that are not supported through EVK or QT600 kits, the output measurement values can be viewed through AVR QTouch Studio using the same QDebug protocol and QT600 USB bridge.

If a QT600 bridge is not available, please refer to section 5.6.11.6.3 for more information on observing the output touch measurement data without the use of a USB bridge or AVR QTouch Studio.

5.6.11.6.1 Debug Support in the sample applications for EVK2080 and QT600 boards

The sample applications provided for the EVK2080 boards, QT600 boards and the other example projects output debug information which is captured by a USB bridge chip and then routed to the QTouch Studio for display.

Note:

The port and pins assigned for the QDebug protocol with the example projects are arbitrary and have to be changed based on the project configuration chosen and pin availability.

A separate App note is available on the Atmel website (in QTouch libraries webpage) explaining the QT600 debug protocol.

5.6.11.6.2 How to turn on the debug option

In the project options, the symbol definition _DEBUG_INTERFACE_ is used to enable reporting the debug data. You can enable the debug interface by enabling the debug macro in touch_config.h file.

```
/*----------------------------------------------------------------------------
                         Debug Interface settings
----------------------------------------------------------------------------*/
/**
  * Enable/Disable QDebug for touch debug information communication with
  * QTouch Studio PC Software.
  * QDebug is enabled by defining the below macro.
  */
#define _DEBUG_INTERFACE_

#ifdef _DEBUG_INTERFACE_
/**
  * Select the type of interface to use for the debug protocol.
  */

//! When 0, QDebug interface type is on-chip SPI peripheral
//! When 1, QDebug interface type is on-chip TWI peripheral
//! When 2, QDebug interface type is Bit bang SPI
#define DEF_QTOUCH_INTERFACE_TYPE     (2)

/**
  * Bit bang SPI port and pin configuration.
  * Note: Ensure that the port pins mentioned below are not used for SNS/SNSK
  */
#define SS_BB       4
#define SCK_BB      7
#define MOSI_BB     5
#define MISO_BB     6

#define SPI_BB_SS           B
#define SPI_BB_SCK          B
#define SPI_BB_MOSI         B
#define SPI_BB_MISO         B

#endif
```

Figure 5-21 : Enabling and configuring the Debug Support

5.6.11.6.3 Debug Interface if USB Bridge board is not available

For the sample applications using the devices that are not supported on EVK2080 and QT600 the debug interface code is not provided. This is because a separate USB bridge board is required to read the data and display it on QTouch studio. However in this case the output touch measurement data can still be viewed using the IAR or AVR Studio IDE when running the code in debug mode using debug wire or emulator.

```
extern qt_touch_lib_measure_data_t qt_measure_data;
```

The **qt_measure_data** global variable contains the output touch measurement data. Refer to section 5.6.4.3 for more information on the data type.

For GCC generated libraries the output touch measurement data can be observed on the watch window through the pointer **pqt_measure_data.**

```
qt_touch_lib_measure_data_t *pqt_measure_data = &qt_measure_data;
```

5.7 Library Variants

5.7.1 QTouch Acquisition method library variants

5.7.1.1 Introduction

Variants of the ATMEL QTouch Library based on QTouch Technology are available for a range of ATMEL Microcontrollers. This section lists the different variants available. By following a simple series of steps, the user can identify the right library variant to use in his application.

5.7.1.2 Support for different compiler tool chains

The QTouch acquisition method libraries are supported for the following compiler tool chains.

Table 7 Compiler tool chains supported for QTouch acquisition method libraries

Tool	Version
IAR Compiler for 8bit AVR	6.10.0
IAR Embedded Workbench for AVR	6.10
Atmel Studio	6.0.x
IAR Compiler 32bit AVR	4.10.1
GCC – GNU Toolchain for AVR 8bit	3.4.0.1028
GCC – GCC Toolchain AVR 32bit	3.4.0.1028
IAR Embedded Workbench for ARM	6.30
GCC for ARM	Sourcery G++ Lite for ARM EABI V 2011.3.0.42

5.7.1.3 QTouch Acquisition method library naming conventions

The libraries are named according the convention listed below

5.7.1.3.1 Naming convention for libraries to be used with GCC tool chain

lib<**coreP**>g1_<**CH**>qt_k_<**RS**>rs.a

Field name	Possible values	Comments
coreP	avr25 avr 35 avr 4 avr 51 avr 5 avrxmega2 avrxmega3 avrxmega4 avrxmega5 avrxmega6 avrxmega7 uc3a uc3b uc3c sam3s sam3u	VersionP of the core for AVR/ATSAM devices supported by this library variant for tinyAVR and megaAVR devices.

	sam3n sam4s	
CH	4, 8, 12, 16, 32	Total number of channels supported by each library.
RS	1, 2, 3, 4, 8	Total number of rotors / sliders supported for the respective channel counts mentioned in previous row.

For example, the library variant *"libavr25g1_8qt_k_2rs.a"* supports the following configuration

- Device : tinyAVR or megaAVR device belonging to core version avr25
- Belongs to a set of devices of group 1 supported by this library
- Support a maximum of 8 channels
- Supports a maximum of up to 2 rotors / sliders.

5.7.1.3.2 Naming convention for libraries to be used with IAR Embedded Workbench

The libraries are named according the naming convention listed below

lib<**coreP**>g<**Q**>_<**CH**>qt_k_<**RS**>rs.r90

Field name	**Possible values**	**Comments**
coreP	v1 v3 v3xmsf v3xm v4xm v5xm v6xm uc3a uc3b uc3c sam3s sam3u sam3n	VersionP of the for AVR/ATSAM devices supported by this library variant variant for tinyAVR and megaAVR devices.
Q	1 to 3	GroupQ of the core AVR device supported by this library variant
CH	4, 8, 12, 16, 32	Total number of channels supported by each library.
RS	1, 2, 3, 4, 8	Total number of rotors / sliders supported for the respective channel counts mentioned in previous row.

For example, the library variant *"libv3g2_4qt_k_1rs.r90"* supports the following configuration

- Device : tinyAVR or megaAVR device belonging to core version v3
- Belongs to a set of devices of group 2 supported by this library
- Supports a maximum of 4 channels
- Supports 1 rotor/slider

5.7.1.4 QTouch acquisition method library variants

lists the different QTouch acquisition method library variants supported for AVRs. Use this table to select the correct library variant to be used in your application. Each row in the table below indicates

- the corresponding Ports available for SNS and SNSK pins
- Compilers used for generating the libraries
- The library names to be selected for the requirements

Note: The libraries that are supported as listed in the table are only supported provided the device memory requirements are also satisfied.

Naming convention of the library

<ch>	Maximum channels supported by the library.	
	Device	Range
	tinyAVR, megaAVR, XMEGA	4,8,12,16
	UC3	8,16,32
	ATSAM	32
<RS>	Maximum number of rotor / sliders supported	

NOTE:

- For 8-bit devices, ports which have less than 8 pins cannot be used by the QTouch acquisition method libraries. Check the data sheet to determine the number of pins supported for each port

5.7.1.5 Port combinations supported for SNS and SNSK pin configurations

For the list of all ports supported for each device please refer to the library selection guide. There are no limitations for AVR devices (8bit and 32 bit) on the combination of SNS and SNSK port to be used from QTouch libraries 4.0 release onwards.

For ATSAM devices the one port pair combinations supported are given below in the table.

One port pair supported combinations for ATSAM	AA, BB, CC, AB, BA, AC, CA, BC, CB

5.7.1.5.1 Tips on pin assignments for the sensor design using one pair of SNS/SNSK ports

This section lists tips on selecting the pin assignments when using a single port pair for the SNS and SNSK Pins.

Design choice for the sensor	**Example Port configuration with pin assignments**
SNSK & SNS pins are on different ports, number of channels = 4	• If the SNS1(C) and SNSK1(B) pins are on two different ports, the user should mount the sensors onto the corresponding pins such as (PC0,PB0), (PC1,PB1), (PC2,PB2) and (PC3,PB3), when pin configurability is not used. • In case of pin configurability, sensors should be mounted on the pins as selected based on rules illustrated in section 5.8.1
SNSK & SNS pins are on different ports, number of channels = 8	• If the SNS1(C) and SNSK1(B) pins are on two different ports, the user should mount the sensors onto the corresponding pins such as (PC0,PB0), (PC1,PB1), (PC2,PB2) and so on, When pin configurability is not used. • When using pin configurability, sensors should be mounted on the pins as selected based on rules illustrated in section 5.8.1 • When pin configurability is not used, channel 0 will be on (PC0, PB0) pins, channel 1 will be on (PC1, PB1) pins and so on up to channel 7 will be on (PC7, PB7) pins. • When using pin configurability, channel should be assigned as given in section 5.6.6.1.1.2

	•
SNSK & SNS pins are on different ports, number of channels = 32 when using UC3 device	• If the SNS1(B) and SNSK1(A) pins are on two different ports, the user should mount the sensors onto the corresponding pins such as (PB0,PA0), (PB1,PA1), (PB2,PA2).. • In this case channel 0 will be on (PB0, PA0) pins, channel 1 will be on (PB1, PA1) pins and so on up to channel 31 will be on (PB31, PA31) pins.
SNSK & SNS pins are on the same port, number of channels = 2	• If the use of SNS1(A) and SNSK1(A) pins are on the same port, the user should always have the configuration (PA0, PA1) & (PA2, PA3). In this case channel 0 will be on (PA0, PA1) pins; channel 1 will be on (PA2, PA3) pins. The even pins of the port are used as SNS1 pins and odd pins of the port are used as SNSK1 pins • When pin configurability is used, sensors should be mounted on the pins as selected as per the rules illustrated in section 5.8.1 and channels should be assigned as given in section 5.6.6.1.1.4
SNSK & SNS pins are on the same port, number of channels = 4	• If the use of SNS1(A) and SNSK1(A) pins are on the same port, the user should always have the configuration (PA0, PA1), (PA2, PA3), (PA4, PA5) & (PA6, PA7). In this case channel 0 will be on (PA0, PA1) pins, channel 1 will be on (PA2, PA3) pins and so on up to channel 4 will be on (PA6, PA7) pins. The even pins of the port are used as SNS1 pins and odd pins of the port are used as SNSK1 pins, when pin configurability is not being used. • When using pin configurability, sensors should be mounted on the pins as selected as per the rules illustrated in section 5.8.1 and channels should be assigned as given in section 5.6.6.1.1.4
SNSK & SNS pins are on the same port, number of channels = 16 *(Available only for UC3 devices if more than 4 channels are to be used on a single port. For tinyAVR, megaAVR, XMEGA devices up to 8 channels with SNS and SNSK on same ports refer to section 5.7.1.5.2)*	• This configuration is available only for UC3 library variants. • In the use of SNS(A) and SNSK(A) pins are on the same port, the user should always have the configuration (PA0, PA1), (PA2, PA3), (PA4, PA5) & so on. In this case channel 0 will be on (PA0, PA1) pins, channel 1 will be on (PA2, PA3) pins and so on up to channel 15 will be on (PA30, PA31) pins. The even pins of the port are used as SNS pins and odd pins of the port are used as SNSK pins
SNSK & SNS pins are on the same port, number of channels = 16 *(Available only for SAM devices)*	• If the use of SNS(A) and SNSK(A) pins are on the same port, the user should always have the configuration (PA0, PA1), (PA2, PA3), (PA4, PA5), (PA6, PA7) and so on. • In this case channel 0 will be on (PA0, PA1) pins, channel 1 will be on (PA2, PA3) pins and so on up to channel 15 will be on (PA30, PA31) pins. • The even pins of the port are used as SNS pins and odd pins of the port are used as SNSK pins

5.7.1.5.2 Port combinations supported for two port pair SNS and SNSK pin configurations

For the list of all ports supported for each device please refer to the library selection guide. There are no limitations on the combination of SNS and SNSK port to be used from QTouch libraries 4.0 release onwards.

For ATSAM devices the total two port pairs supported combinations are given below in the table.

Two port pairs supported combinations for ATSAM	AA_BB, BB_AA, AA_CC, CC_AA, BB_CC, CC_BB, AA_BC, AA_CB, BB_AC, BB_CA, CC_BA, CC_AB

5.7.1.5.2.1 Tips on pin assignments for the sensor design using two pairs of SNS / SNSK ports

This section lists tips on selecting the pin assignments when using a single port pair for the SNS and SNSK Pins.

Design choice for the sensor	Example Port configuration with pin assignments
SNSK1-SNS1 & SNSK2-SNS2 pins are all on different ports, number of channels = 16 *(Use the 16channel library in this case. Ensure the port definitions for SNS1,SNSK1,SNS2,SNSK2 are all in place)*	• E. g. SNS1(D), SNSK1(B) & SNS2(C), SNSK2(A) • Recommended configuration: (PD0, PB0), (PD1, PB1),..(PD7, PB7), (PC0,PA0).. to (PC7, PA7). In this case channel 0 will be on (PD0, PB0) pins, channel 1 will be on (PD1, PB1) pins, channel 8 will be on (PC0, PA0), channel 9 will be on (PC1, PA1) and so on up to channel 15 will be on (PC7, PA7) pins. • However, the user can mount the sensors on pins as selected as per the rules illustrated in section 5.8.1 and channels should be assigned as given in section 5.6.6.1.1.2
SNSK1-SNS1 are on same port & SNSK2-SNS2 pins are on same port, number of channels = 8 *(Use the 8channel library in this case. Ensure the port definitions for SNS1,SNSK1,SNS2,SNSK2 are all in place)*	• E.g. SNS1(K), SNSK1(K) & SNS2(H), SNSK2(H) on same ports, • Recommended configuration: In case Pin configurability is not used, (PK0, PK1), (PK2, PK3),..(PK6, PK7), (PH0,PH1).. to (PH6, PH7).In this case channel 0 will be on (PK0, PK1) pins, channel 1 will be on (PK2, PK3) pins, channel 4 will be on (PH0, PH1), channel 5 will be on (PH2, PH3) and so on up to channel 7 will be on (PH6, PH7) pins. The even pins of the port are used as SNS pins and odd pins of the port are used as SNSK pins. • When pin configurability is used, sensors should be mounted on the pins as selected as per the rules illustrated in section 5.8.1 and channels should be assigned as given in section 5.6.6.1.1.2
SNSK1-SNS1 are on different ports & SNSK2-SNS2 pins are on same port, number of channels = 12 *(Use the 12channel library in this case. Ensure the port definitions for SNS1,SNSK1,SNS2,SNSK2 are all in place)*	• E.g. SNS1(H), SNSK1(F) on different ports & SNS2(E), SNSK2(E) on same ports. • Recommended configuration : In case Pin configurability is not used, (PH0, PF0), (PH1, PF1),..(PH7, PF7), (PE0,PE1).. to (PE6, PE7). In this case channel 0 will be on (PH0, PF0) pins, channel 1 will be on (PH1, PF1) pins... channel 8 will be on (PE0,PE1), channel 9 will be on (PE2,PE3) and so on up to channel 11 will be on (PH6, PH7) pins. The even pins of the port E are used as SNS pins and odd pins of the port E are used as SNSK pins. • When pin configurability is used, sensors should be mounted on the pins as selected as per rules illustrated in section 5.8.1 and channels should be assigned as given in section

	5.6.6.1.1.2 and section 5.6.6.1.1.4
SNSK1-SNS1 are on same port & SNSK2-SNS2 pins are on different ports, number of channels = 12 *(Use the 12channel library in this case. Ensure the port definitions for SNS1,SNSK1,SNS2,SNSK2 are all in place)*	• E.g. SNS1(G), SNSK1(G) on different ports & SNS2(B), SNSK2(D) on same ports • Recommended configuration: In case Pin configurability is not used, (PG0, PG1), (PG2, PG3),..(PG6, PG7), (PB0,PD0)... to (PB7, PD7). In this case channel 0 will be on (PG0, PG1) pins, channel 1 will be on (PG2, PG3) pins... channel 3 will be on (PG6, PG7), channel 4 will be on (PB0,PD0) and so on up to channel 11 will be on (PB7, PD7) pins. The even pins of the port G are used as SNS pins and odd pins of the port G are used as SNSK pins • When pin configurability is used, sensors should be mounted on the pins as selected as per rules illustrated in section 5.8.1 and channels should be assigned as given in section 5.6.6.1.1.2 and section 5.6.6.1.1.4

5.7.1.6 Sample applications and Memory requirements for QTouch acquisition method libraries

Refer to the library selection guide for memory requirements for each of the libraries supported in the release.

5.7.2 QMatrix acquisition method library variants

5.7.2.1 Introduction

Variants of the ATMEL QTouch Library based on Matrix™ acquisition technology are available for a range of ATMEL Microcontrollers. Refer to the library selection guide (C:\ Program Files\Atmel\ Atmel_QTouch_Libaries_5.x\Library_Selection_Guide.xls) for the list of devices currently supported for QMatrix.

5.7.2.2 Support for different compiler tool chains

The QMatrix acquisition method libraries are supported for the following compiler tool chains.

Tool	Version
IAR Compiler	6.10
IAR Embedded Workbench	6.10
Atmel Studio 6	6.0.x
GCC – GNU Toolchain for AVR	3.4.0.1028
IAR Compiler 32bit AVR	4.10.1
GCC – GNU Toolchain for AVR32	3.4.0.1028

5.7.2.3 QMatrix Acquisition method library naming conventions

The libraries are named according the naming convention listed below

Tool Chain	Naming convention
GCC Tool Chain	lib<***D***>_<***NC***>qm_<***NX***>x_<***NY***>y_<***CFG***>_<***NRS***>rs.a
IAR –EWAR	lib<***D***>_<***NC***>qm_<***NX***>x_<***NY***>y_<***CFG***>_<***NRS***>rs.r90

Field name	Possible values	Comments
D	**Common for IAR & GCC:** ATtiny167, ATmega128rfa1, ATmega8535 **Specific to IAR:** v1g1s0 (ATtiny44, ATtiny84) v1g1s1 (ATtiny48, ATtiny88) v1g1s2(ATtiny461, ATtiny861) ATmega16a v1g2s1 (ATmega48PA, ATmega88PA) v3xmsf (ATxmega16A4, ATxmega16D4, ATxmega32A4, ATxmega32D4) v3xm (ATxmega64A3) v4xm(ATxmega64A1) v5xm(ATxmega128A3, ATxmega192A3, ATxmega256A3, ATxmega256A3B) v6xm(ATxmega128A1) v3g3 (ATmega165P, ATmega325P, ATmega645, ATmega164p, ATmega324p, ATmega324pa, ATmega644p, ATmega168p, ATmega328p, AT90CAN32, AT90CAN64) v3g5 (AT90CAN128, AT90USB1286, AT90USB1287, ATmega1280, ATmega1281) v3g6 (AT90USB162) v3g7 (AT90USB646,	Indicates the device / core group name in short form. For XMEGA Devices, Core groups are taken which follows As below for both GCC and IAR Supported XMEGA Devices ATxmega16A4, ATxmega16D4, ATxmega32A4, ATxmega32D4, ATxmega64A1 ATxmega128A1 ATxmega64A3 ATxmega128A3, ATxmega192A3, ATxmega256A3, ATxmega256A3B)

	AT90USB647) **Specific to GCC:** avr25g1s0 (ATtiny44, ATtiny84) avr25g1s1 (ATtiny48, ATtiny88) avr25g1s2(ATtiny461, ATtiny861) ATmega16 avr4g1s1 (ATmega48P, ATmega88P) avr5g4 (AT90USB646, AT90USB647) avr5g6 (AT90USB162) avrxmega2 (ATxmega16A4, ATxmega16D4, ATxmega32D4) avrxmega3 (ATxmega32A4) avrxmega4 (ATxmega64A3) avrxmega5(ATxmega64A1) avrxmega6(ATxmega128A3, ATxmega192A3, ATxmega256A3, ATxmega256A3B) avrxmega7(ATxmega128A1) avr5g3 (ATmega165P, ATmega325P, ATmega645, ATmega164p, ATmega324p, ATmega324p, ATmega644p, ATmega168p, ATmega328p, AT90CAN32, AT90CAN64) avr51g2 (AT90CAN128, AT90USB1286, AT90USB1287, ATmega1280, ATmega1281) AT32UC3C0512	
NC	4,8,16,24,32,56,64	Indicates the maximum number of channels that the

		library supports 56 (8 x 7) support only for ATXmega Devices. 24((8 x 3) support only for 32 Bit Devices.
NX	4,8	Indicates the number of X-Lines that the library needs for supporting the listed number of channels. The X lines on a PORT always start with Least Significant Bit of the PORT. Ex: #define PORT_X_1 B in case of a 4x2 QMatrix library means X0,X1,X2,X3 are on PB0,PB1,PB2,PB3
NY	1,2,3,4,7,8	Indicates the number of Y-Lines that the library needs for supporting the listed number of channels NY=7 support only for ATXmega Devices NY=3 support only for 32Bit Devices
CFG	k krs	k – library variant supports only keys krs – library variant supports keys, Rotors and Sliders
NRS	0,1,2,3,4,8	Maximum number of rotor sliders that the library supports. NRS=3 support only for 32Bit Devices

The table below provides a few examples of the naming convention.

Example Library name	Configuration supported
libavr51g2_8qm_4x_2y_krs_2rs.a	• Compiler tool chain : GCC • Device : ATMega164P • 8 Channels • 4 X lines • 2 Y lines • Supports Keys, Rotors and Sliders (krs) • 2 Rotors and Sliders
libavr25g1s1_16qm_8x_2y_k_0rs.r90	• Compiler tool chain : IAR • Device : ATTiny88 • 16 Channels • 8 X lines • 2 Y lines • Supports only keys (k) • 0 Rotors and Sliders

5.7.2.4 QMatrix acquisition method library variants

5.7.2.4.1 Devices supported for QMatrix Acquisition

Refer to the Library_selection_guide.xls for the list of devices supported for QMatrix for this release.

5.8 PIN Configuration for QTouch Libraries

5.8.1 Pin Configuration for QTouch Acquisition Method

Pin configurability for QTouch acquisition method is provided for 8Bit AVR's. QTouch acquisition method libraries can be used to configure SNS and SNSK on any pins of the port. But few rules should be followed while assigning the SNS and SNSK on particular pins. These rules are internal to the library. But QTouch Studio –Pin Configuration Wizard can be used to assign SNS and SNSK on the pins and rules are internally taken care in the QTouch Studio Pin Configuration Wizard.

By default, for 4 and 8 channel QTouch acquisition libraries, the channel numbering follows the pin number of the port.

To use the pin configurability, enable the macro _STATIC_PORT_PIN_CONF_ in the project options or define the macro in the touch_config.h file.

To use the pin configurability feature, the SNS_array and SNSK_array masks are exported for the user, which needs to be initialized. These SNS_array and SNSK_array masks can be taken from the QTouch Studio Pin Configuration Wizard and can be copied at appropriate place in the main.c file as explained in the example projects provided.

QTOUCH_STUDIO_MASKS macro is used for providing pin configurability feature for QTouch Acquisition method libraries.

In case the macro QTOUCH_STUDIO_MASKS enabled in project space, SNS_array and SNSK_array takes values that are supplied by the user in main.c files. This will reduce the code memory foot print of the library.

In case the macro QTOUCH_STUDIO_MASKS is not enabled in project space, SNS_array and SNSK_array are calculated internal to the library according to the configured sensors.

Note:

1. Port pin configurability is enabled for the following configurations,

   4- channel intraport configuration

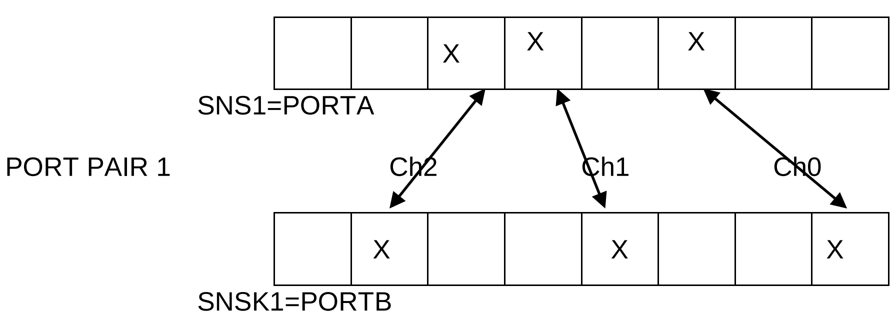
   8-channel intraport configuration

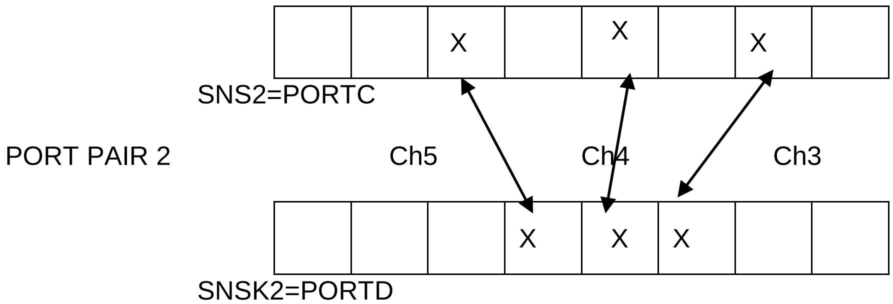
   12-channel configuration

   16- channel configuration

2. In case, the user wants to use the pin configurability for the other supported configurations, (4- channel interport and 8-channel interport), the user has to enable the macro _STATIC_PORT_PIN_CONF_ in his project space.

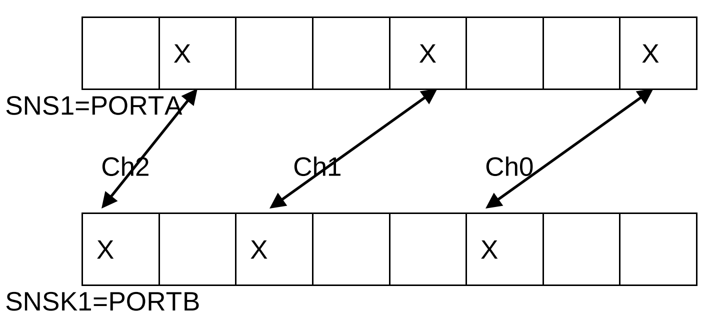
5.8.1.1 Rules for configurable SNS-SNSK Mask Generation





1. The channel numbers are allocated based on enabled SNS pins starting from LSBit of port 1(SNS1) and ending with MSBit of port 2(SNS2).
2. The number of SNS pins in a port pair should be equal to the SNSK pins in the same port pair so it can form a pair.
3. The first SNS port pin should always be mapped to the first SNSK port pin in any port pair. Similarly the second SNS port pin should always be mapped to second SNSK pin and so on.
4. Even sensors with in a port pair should be placed in one mask and odd sensors with-in a port pair should be placed in the second mask. In case of interport, first channel should always start with odd masks and then even masks is filled .
5. All the three channels for ROTORS and SLIDERS should be placed within the same mask. And should be in the same port pair.
6. Keys on adjacent channels should be placed on different masks.
7. For 8 channel case when 2 ports are enabled, the pins for the 8 channels can be spread on the 2 ports. The pin configuration is done based on the rules mentioned above.
8. For 16 channel case when 2 ports are enabled, all the pins for the 16 channels are allocated among the pins of the 2 ports.

5.8.1.1.1 Example for 8 channel interport mask Calculation with one port pair



This example is for interport 8 channel library with only one port pair used.
Channel0 is A0B2,Channel1 is A3B5 and Channel2 is A6B7 are enabled for the 8 channel library.
The SNS_array and SNSK_array masks are calculated by the Qtouch Studio with rules mentioned above.
In this case, the SNS_array and SNSK_array values will be as mentioned below:

SNS_array[0][0]=0x41; (SNS even mask for port pair 1)

SNS_array[0][1]=0x08; (SNS odd mask for port pair 1)

SNS_array[1][0]=0x00; (SNS even mask for port pair 2)

SNS_array[1][1]=0x00; (SNS odd mask for port pair 2)

SNSK_array[0][0]=0x84; (SNSK even mask for port pair 1)

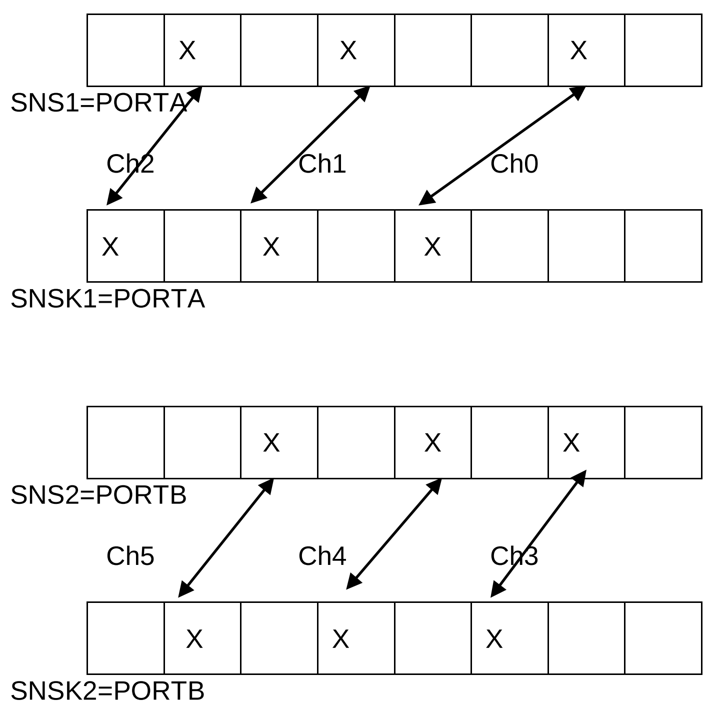
SNSK_array[0][1]=0x20; (SNSK odd mask for port pair 1)

SNSK_array[1][0]=0x00; (SNSK even mask for port pair 2)

SNSK_array[1][1]=0x00; (SNSK odd mask for port pair 2)

As there is no second port pair used for this, so that's why SNS_array[1][0], SNS_array[1][1], SNSK_array[0][1] and SNSK_array[1][1] are having value zero.

5.8.1.1.2 Example for 8 channel intraport mask Calculation with two port pairs



This example is for intraport 8 channel library with two port pair used.
Channel0 is A1A3,Channel1 is A4A5 and Channel2 is A6A7 are enabled in the first port pair. Channel3 is B1B2,Channel4 is B3B4 and Channel5 is B5B6 are enabled in the second port pair.

The SNS_array and SNSK_array masks are calculated by the Qtouch Studio with rules mentioned above.
In this case, the SNS_array and SNSK_array values will be as mentioned below:

SNS_array[0][0]=0x52; (SNS even mask for port pair 1)

SNS_array[0][1]=0x00; (SNS odd mask for port pair 1)

SNS_array[1][0]=0x2a; (SNS even mask for port pair 2)

SNS_array[1][1]=0x00; (SNS odd mask for port pair 2)

SNSK_array[0][0]=0xa8; (SNSK even mask for port pair 1)

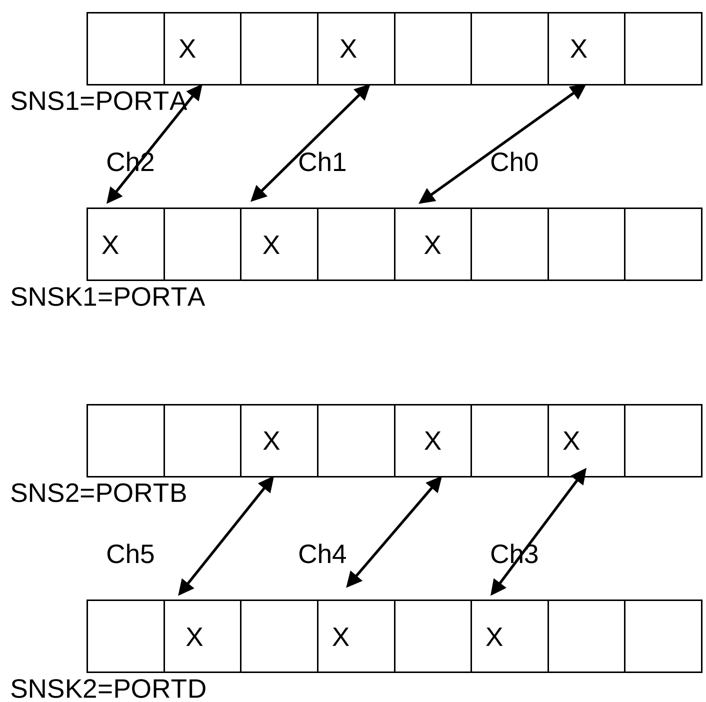
SNSK_array[0][1]=0x00; (SNSK odd mask for port pair 1)

SNSK_array[1][0]=0x54; (SNSK even mask for port pair 2)

SNSK_array[1][1]=0x00; (SNSK odd mask for port pair 2)

In case of Intraport, odd SNS_array and SNSK_array masks are always zero.So that's why SNS_array[0][1] ,SNS_array[1][1], SNSK_array[0][1] and SNSK_array[1][1] are zero for both the port pairs.

5.8.1.1.3 *Example for 12 channel intraport-interport mask Calculation with two port pairs*



This example is for intraport-interport 12 channel library with two port pair used. Channel0 is A1A3,Channel1 is A4A5 and Channel2 is A6A7 are enabled in the first port pair. Channel3 is B1D2,Channel4 is B3D4 and Channel5 is B5D6 are enabled in the second port pair.

The SNS_array and SNSK_array masks are calculated by the Qtouch Studio with rules mentioned above.
In this case, the SNS_array and SNSK_array values will be as mentioned below:

SNS_array[0][0]=0x52; (SNS even mask for port pair 1)

SNS_array[0][1]=0x00; (SNS odd mask for port pair 1)

SNS_array[1][0]=0x22; (SNS even mask for port pair 2)

SNS_array[1][1]=0x08; (SNS odd mask for port pair 2)

SNSK_array[0][0]=0xa8; (SNSK even mask for port pair 1)

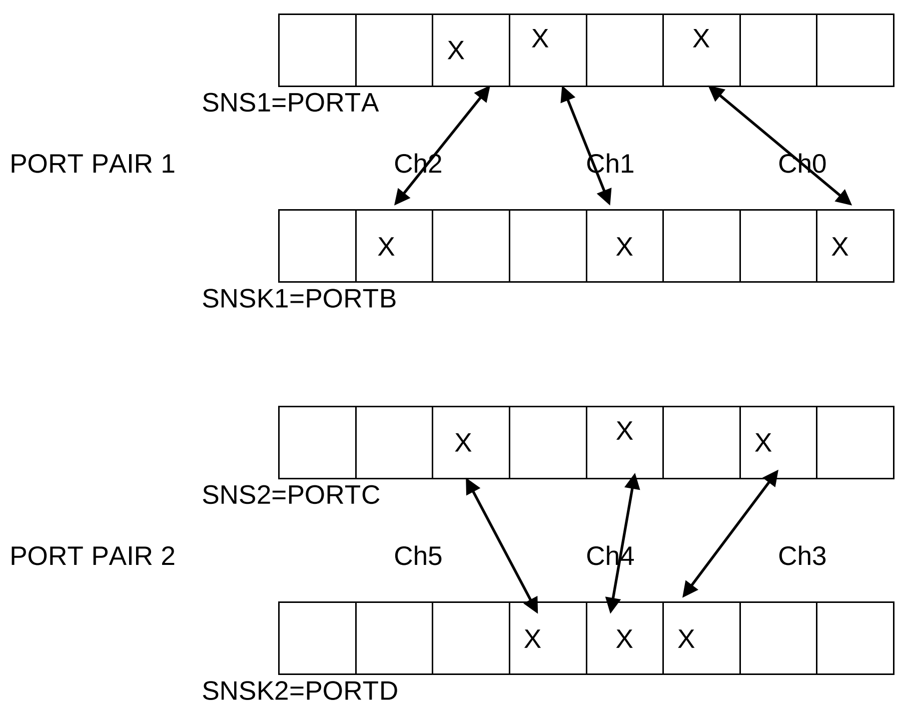
SNSK_array[0][1]=0x00; (SNSK odd mask for port pair 1)

SNSK_array[1][0]=0x44; (SNSK even mask for port pair 2)

SNSK_array[1][1]=0x10; (SNSK odd mask for port pair 2)

As the first port pair is intraport, so that's why SNS_array[0][1] and SNSK_array[0][1] are zero as odd masks are always zero in case of Intraport.

5.8.1.1.4 Example for 16 channel intreport-interport mask Calculation with two port pairs



This example is for interport-interport 16 channel library with two port pair used. Channel0 is A2B0,Channel1 is A4B3 and Channel2 is A5B6 are enabled in the first port pair. Channel3 is C1D2,Channel4 is C3D3 and Channel5 is C5D4 are enabled in the second port pair.

The SNS_array and SNSK_array masks are calculated by the Qtouch Studio with rules mentioned above.

In this case, the SNS_array and SNSK_array values will be as mentioned below:

SNS_array[0][0]=0x24; (SNS even mask for port pair 1)

SNS_array[0][1]=0x10; (SNS odd mask for port pair 1)

SNS_array[1][0]=0x22; (SNS even mask for port pair 2)

SNS_array[1][1]=0x08; (SNS odd mask for port pair 2)

SNSK_array[0][0]=0x41; (SNSK even mask for port pair 1)

SNSK_array[0][1]=0x08; (SNSK odd mask for port pair 1)

SNSK_array[1][0]=0x14; (SNSK even mask for port pair 2)

SNSK_array[1][1]=0x08; (SNSK odd mask for port pair 2)

5.8.1.2 How to Use QTouch Studio For Pin Configurability

Note: The Qtouch Composer is available for few select set of devices.But user can use Qtouch Studio 4.4 for the devices which are not supported in Qtouch Composer.

The following steps describe the details on how to use pin configurability for QTouch Acquisition method:

1. Open AVR QTouch Studio .Enable the Design Mode Radio button on the left hand side of the screen.

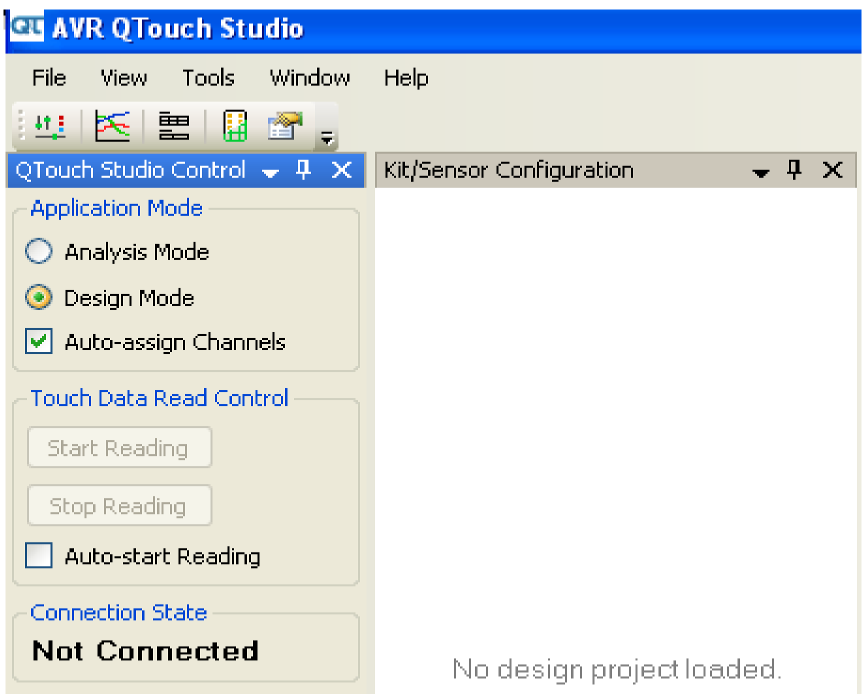


Figure 5-22 Selecting the Design mode in the AVR QTouch Studio

1. Go to File Menu option and click New Design.

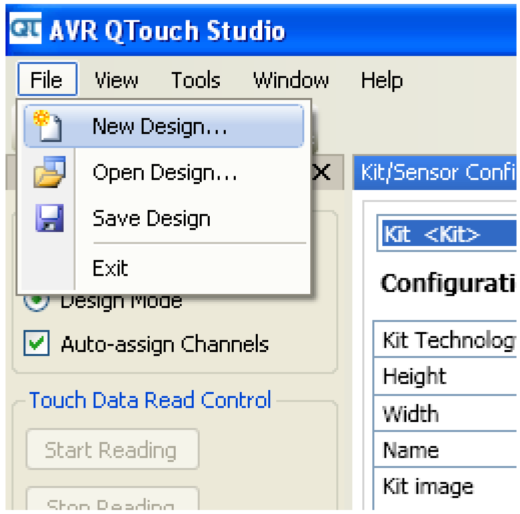


Figure 5-23 Selecting the New Design in the AVR QTouch Studio

2. In the Create New Design Window, give the Project name and Kit Technology (QTouch in this case) and and Number of sensors (Keys/Rotors/Sliders) and click Create Design.

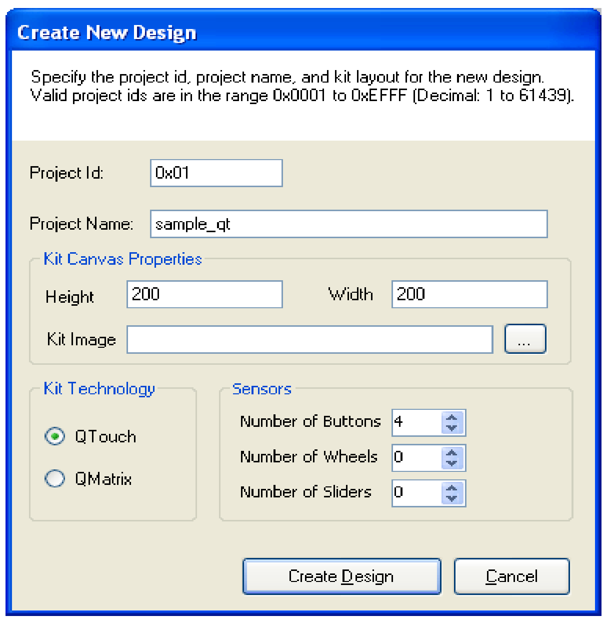


Figure 5-24: Creating New Design in the AVR QTouch Studio

3. After Creating Design, the new screen pops up which shows all the sensors which have been created.

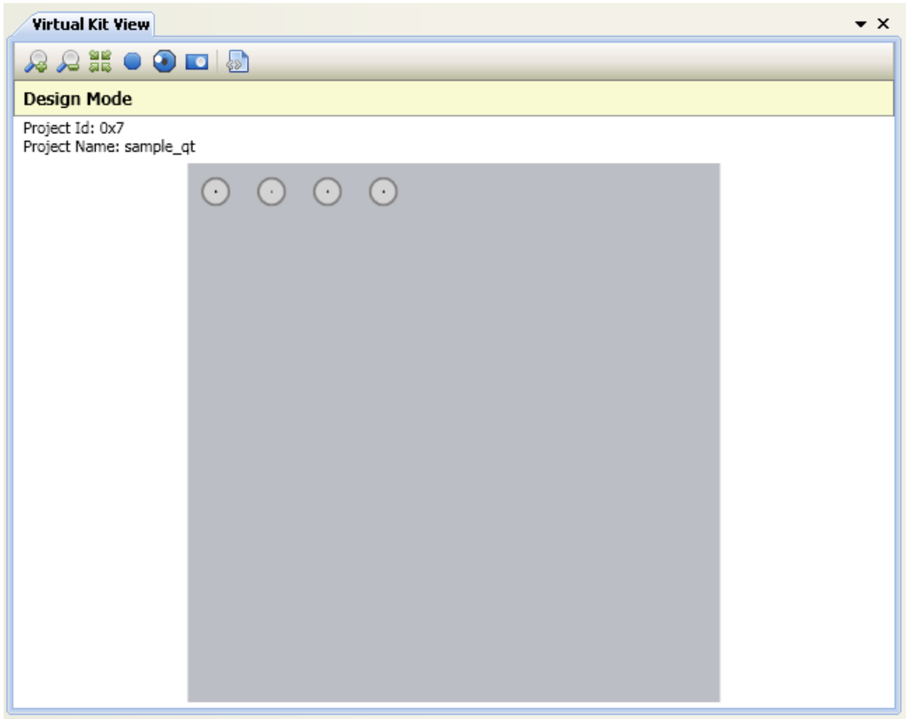


Figure 5-25: New Design Sensors in the AVR QTouch Studio virtual kit

4. Now Go to Tools->Pin Configuration Wizard.Pin configuration

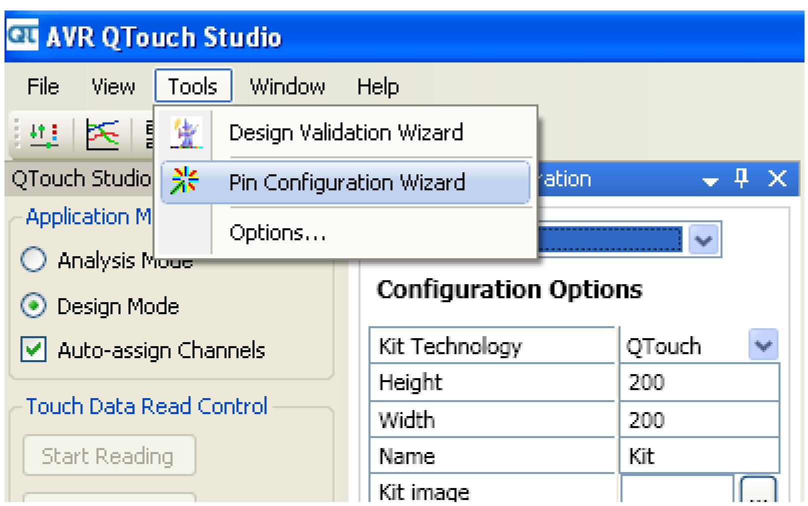


Figure 5-26: Selecting the pin configuration wizard for theDesign

5. Pin configuration Window will pop up with the information on the usage of the tool. Click Next to proceed to the configuration.

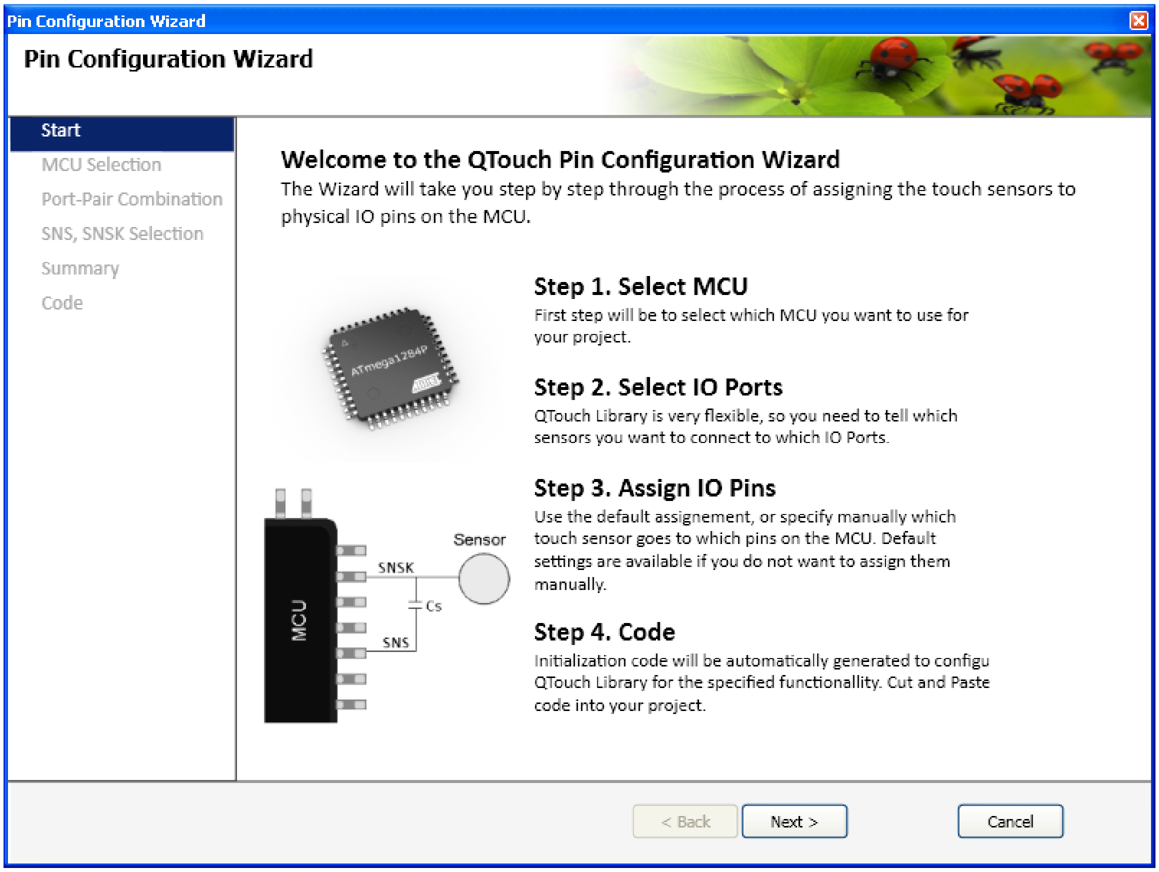


Figure 5-27: : Start page of the wizard

6. Select the MCU and click Next as shown below.

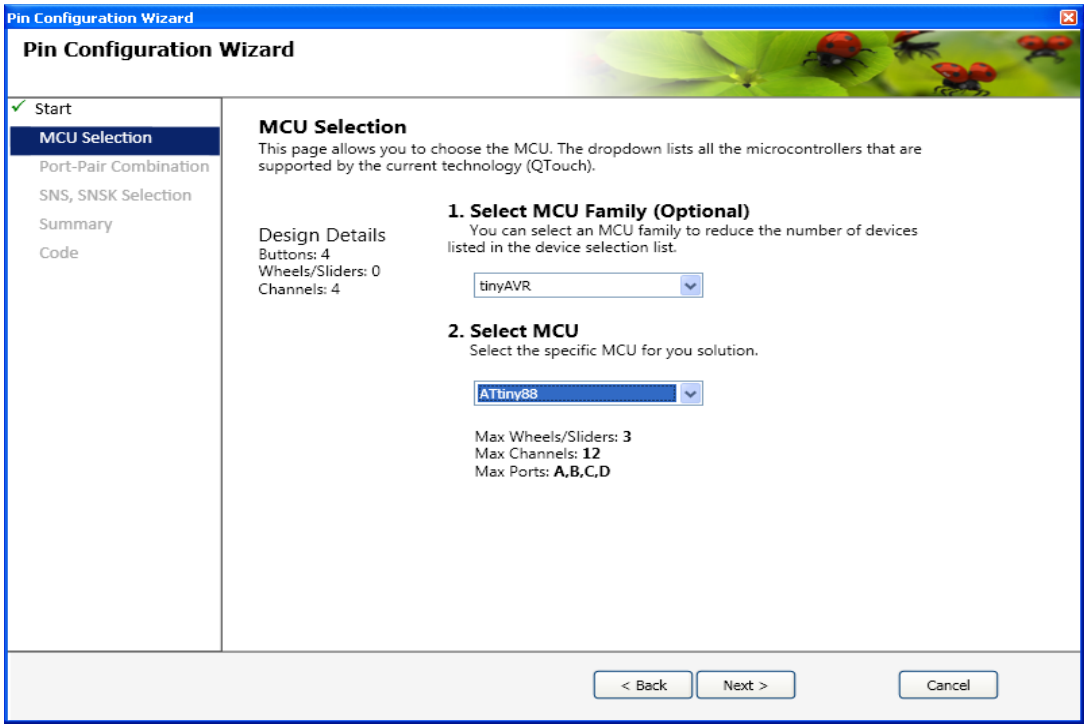


Figure 5-28: Selecting the MCU for the New Design

7. Select the SNS and SNSK ports needs for the design and click Next.

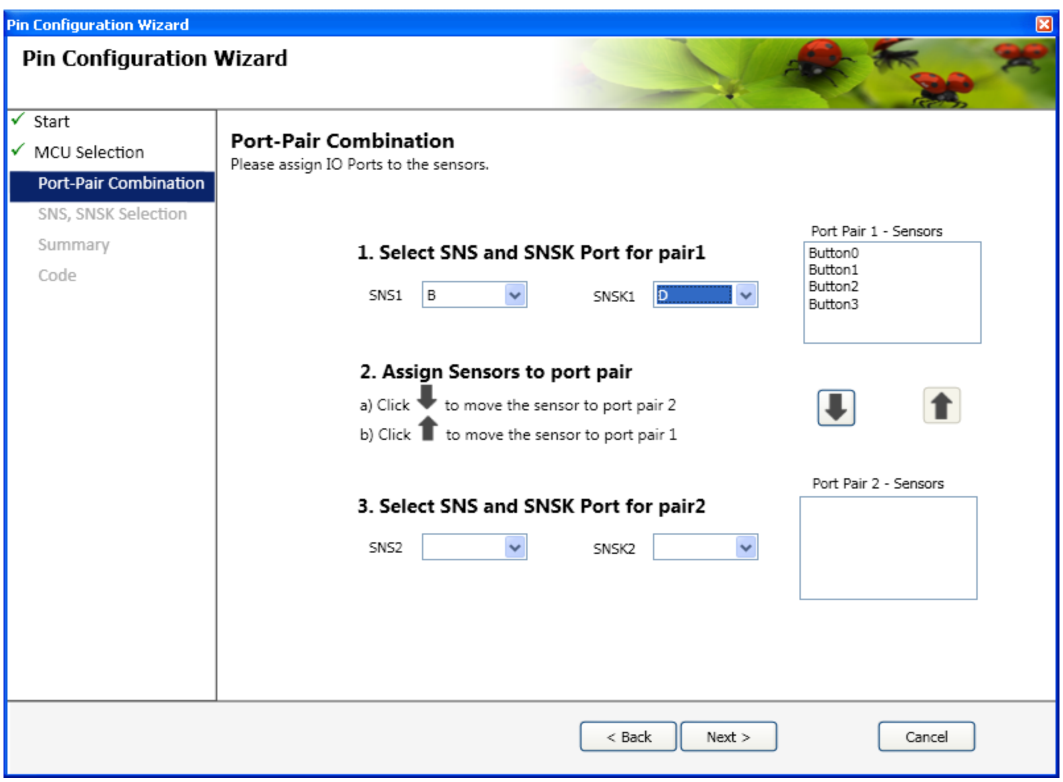


Figure 5-29: Selecting SNS and SNSK ports in the New Design

8. Select the pins used for the design and click Next

If there is error in the selection of the pins (Ex: conflictin pins used), a red marker will be appear and the user cannot proceed to next step in configuration until the user has done the correct pin selection.

Now once the selection is done without errors, Click Next

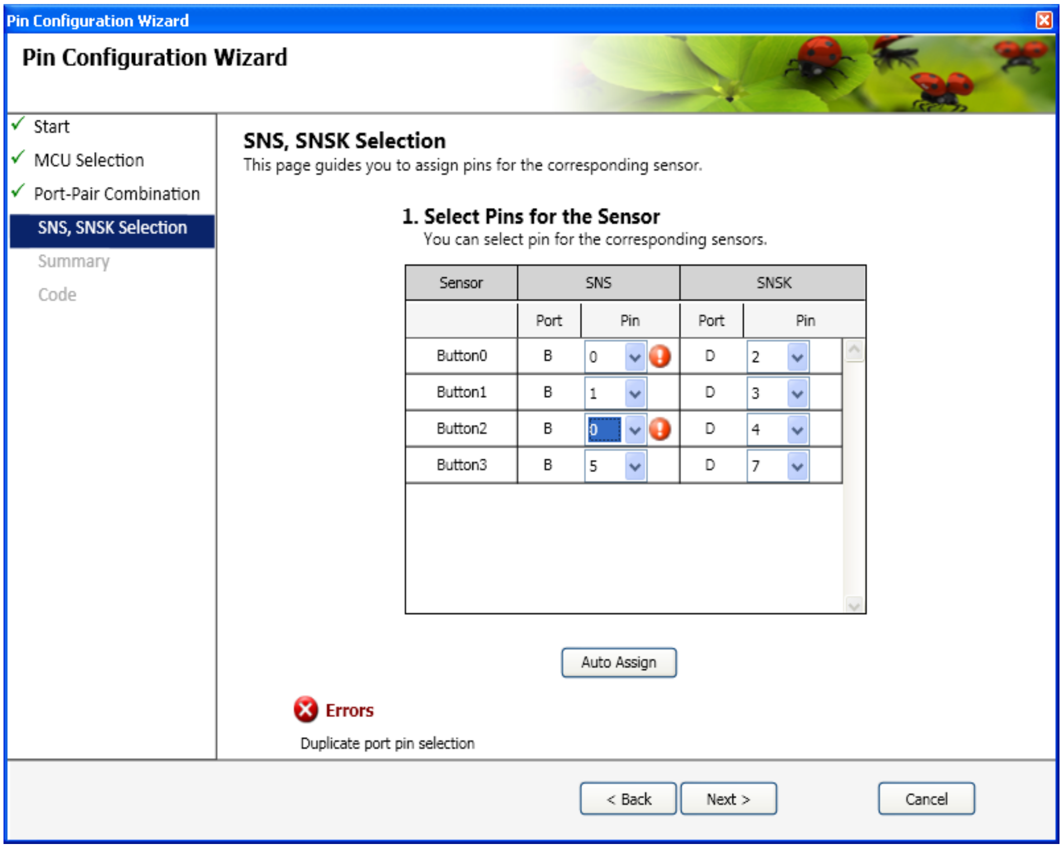


Figure 5-30: Selecting the SNS and SNSK Port Pins in the new Design(With Error)

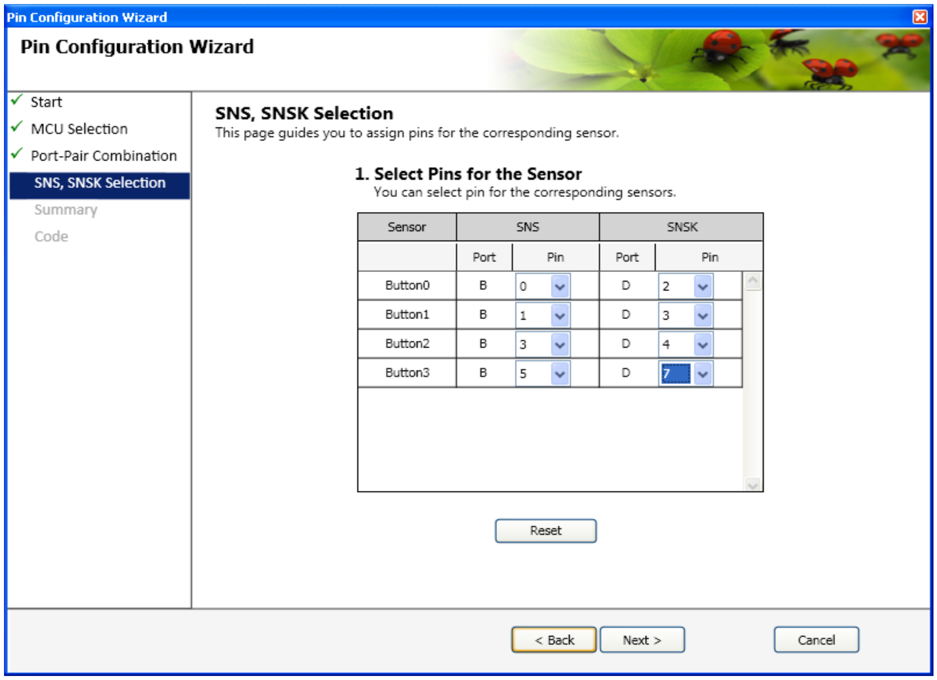


Figure 5-31: Selecting the SNS and SNSK Port Pins in the new Design(Without Error)

Once the pins are selected, Pin Wizard will provide the summary report .Check whether details are correct as specified.Click Next

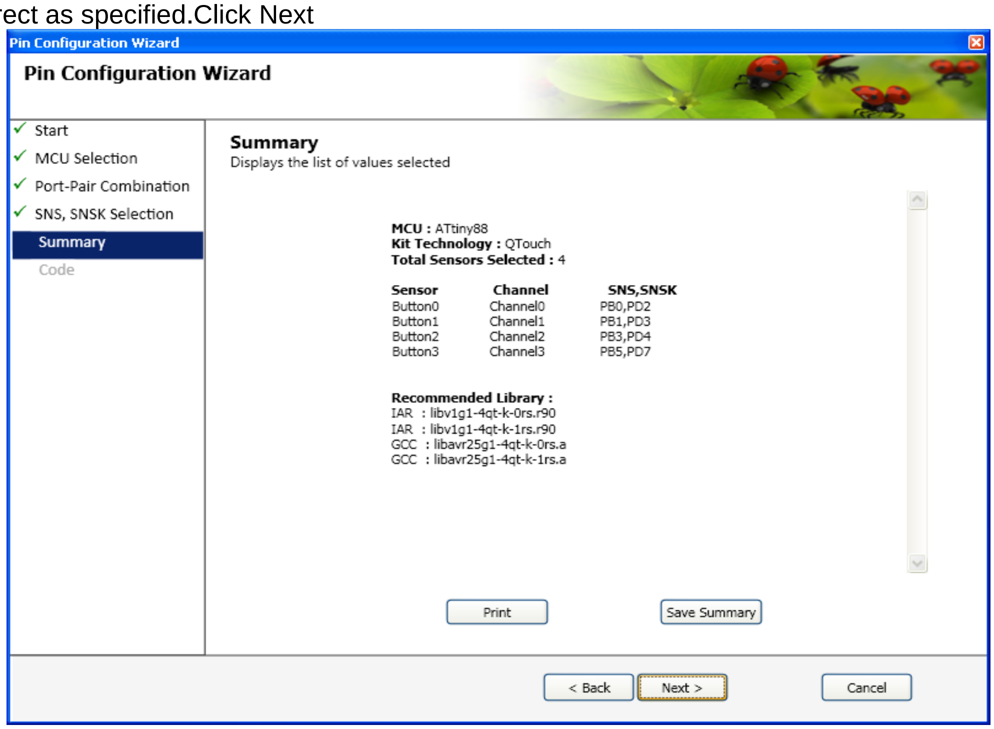


Figure 5-32: Summary report

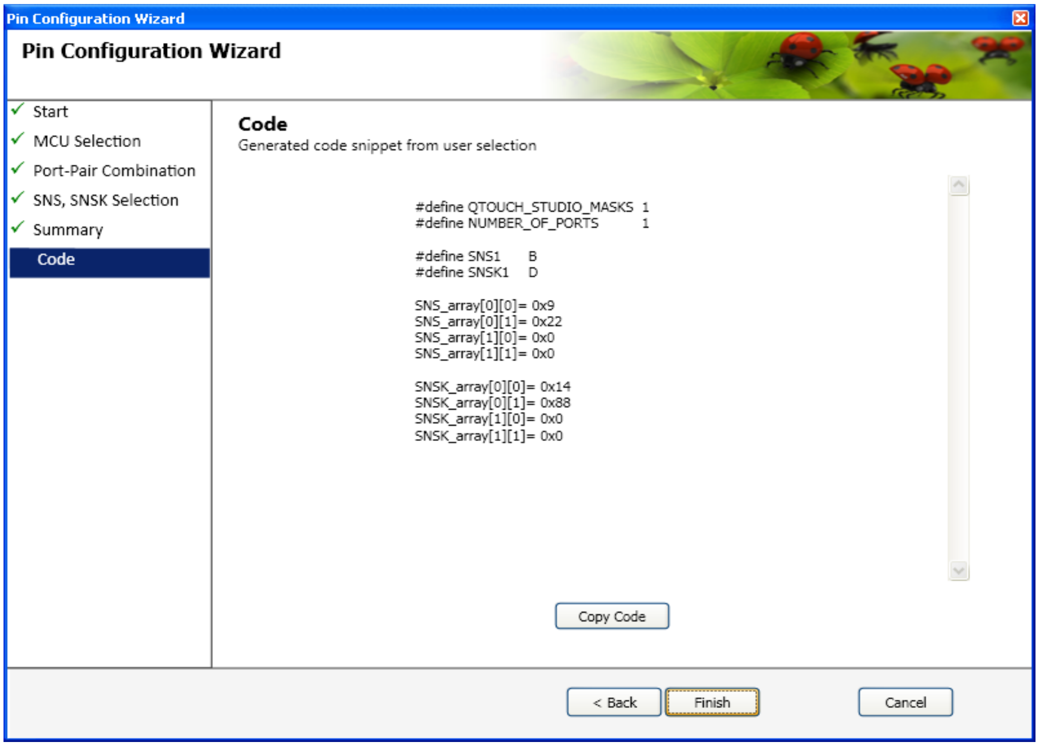


Figure 5-33: Code Generation tab in Pin Configuration wizard

9. In the New Window Screen, the code is shown on the screen.QTOUCH_STUDIO_MASKS needs to be enabled in the project option or in touch_ config.h file. And in the main.c file, this code SNS_array and SNSK_array needs to be copied from here and put under QTOUCH_STUDIO_MASKS macro as shown below in the main.c file:

```
#ifdef QTOUCH_STUDIO_MASKS
    SNS_array[0][0]=0x09;
    SNS_array[0][1]=0x22;
    SNS_array[1][0]=0x00;
    SNS_array[1][1]=0x00;
    SNSK_array[0][0]=0x14;
    SNSK_array[0][1]=0x88;
    SNSK_array[1][0]=0x00;
    SNSK_array[1][1]=0x00;
#endif
```

Note:

1. To use 4 and 8 channel libraries(interport case) for pin configurability , _STATIC_PORT_PIN_CONF_ macro needs to be enabled in the touch_config.h file.

2. QTOUCH_STUDIO_MASKS needs to be enabled if using pin configurability .If not enabled then, static pin mapping will work same as in the earlier versions of the libraries

5.8.2 Pin Configuration for QMatrix Acquisition Method

The QMatrix acquisition method libraries needs to be used after configuring X and YA and YB lines on IO pins of the port as described in the configuration rules described in the section below. The QTouch Studio Pin Configurator Wizard can be used to assign X, YA, YB, SMP lines on the pins and rules are internally taken care in the Qtouch Studio Pin Configurator Wizard.

The snippets can be taken from the QTouch Studio Pin Configurator Wizard and copied to appropriate places in the main.c and touch_ config.h files in the example projects provided.

5.8.2.1 Configuration Rules:

1. The X lines can be configured on different ports up to a maximum of 3 ports

   Ex: NUM_X_PORTS = 3 (maximum value supported). However the possible values are NUM_X_PORTS = 1 or NUM_X_PORTS = 2 or NUM_X_PORTS = 3

2. The X lines can be configured on the three different ports.
3. The X lines can be configured on any pins of the ports selected above

   Ex: X0 on PB2, X1 on PD5, X2 on PE0, X3 on PD1(when NUM_X_LINES= 4), Provided that these pins do not conflict with the other pins for touch sensing or with the host application usage.

4. The Y lines can be configured on the any of the pins of the ports selected

   Ex: Any pins on the PORT_YA and PORT_YB selected.

   Suppose, PORT_YA is D, and PORT_YB is C

   Since, pin 5 and pin 1 PORTD are already used for X lines(X1, X3), the user can select any of the remaining pins for Y0A lines. Suppose that Y0 is on pin2 and Y1 is on pin6

   Hence, Y0A – PD2, Y0B – PC2, Y1A – PD6, Y1B – PC6,

5. All the Qmatrix usage Pins X lines,YA lines, YB lines and SMP line can be on same port.Both YA and YB lines can share the same port. And the YA and YB need not be on same corresponding pins of the ports.The Macro SHARED_YAYB should be defined as 1 if YA and YB are on same port else should be defined as 0.
6. The PORT_YB is fixed for each device and should be same as the PORT on which the ADC input pins are available.
7. The SMP pin can be configured on any of the IO PORT pins available.

   Ex: PORT_SMP = D

   SMP_PIN = 7 as this pin is not being used by touch sensing.

Note:

- Please take care that the touch sensing pins do not conflict with other IO pins used by host application

5.8.2.2 How to use QTouch Studio for Pin Configurability:

Note: The Qtouch Composer is available for few select set of devices.But user can use Qtouch Studio 4.4 for the devices which are not supported in Qtouch Composer. Also YA,YB shared on same port feature is available in Qtouch Composer and not available in Qtouch Studio 4.4.Please refer section 5.6.10.3

The following steps describe the details on how to use pin configurability for QMatix Acquisition method:

1. Open AVR Qtouch Studio .Enable the Design Mode Radio button on the left hand side of the screen..

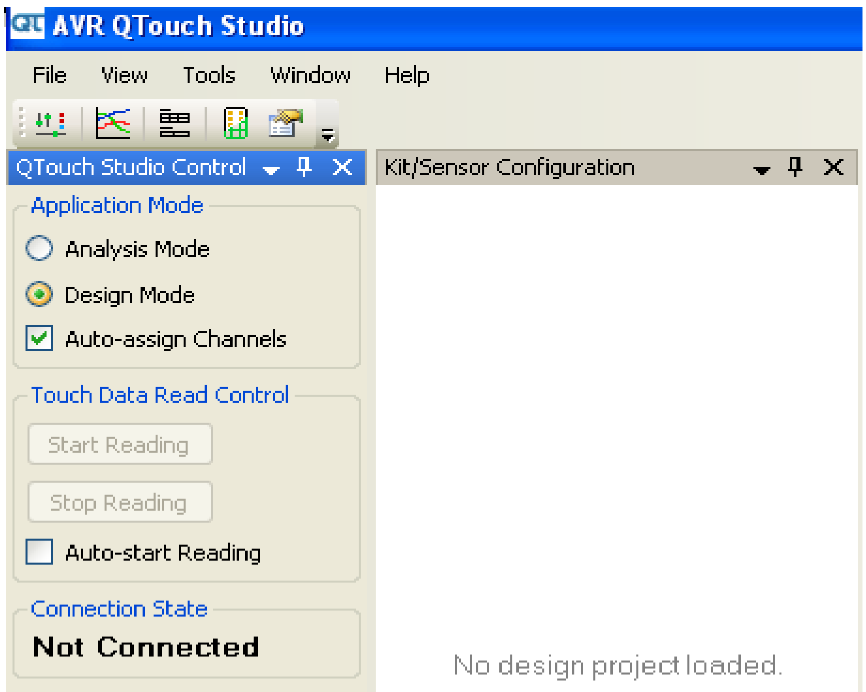


Figure 5-34: Selecting the Design mode in the AVR QTouch Studio

2. Go to File Menu option and click New Design

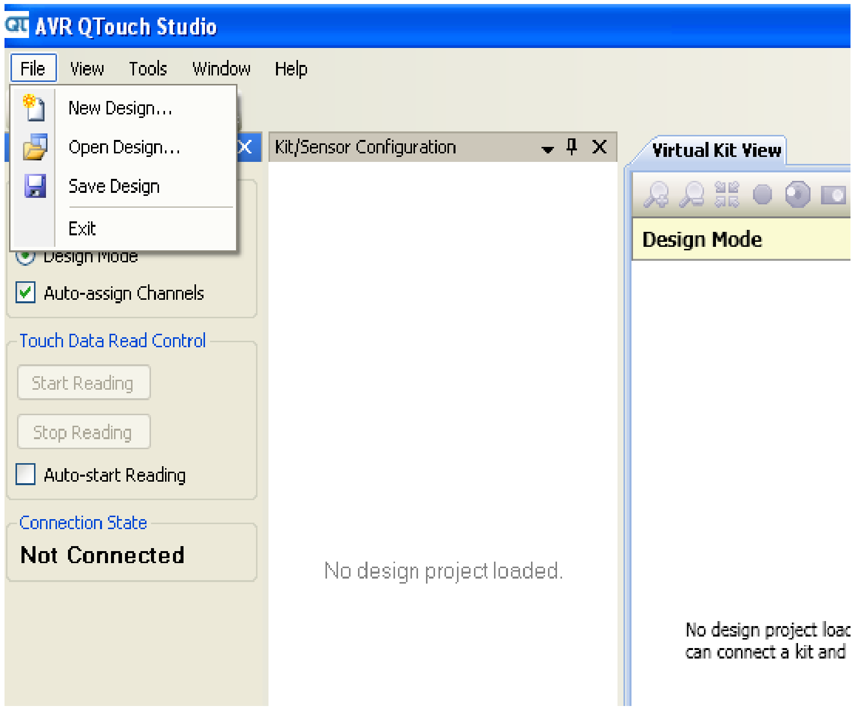


Figure 5-35: Selecting New Design

3. In the Create New Design Window, give the Project name and Kit Technology (QMatrix in this case) and Number of sensors (Keys/Rotors/Sliders) and click Create Design.

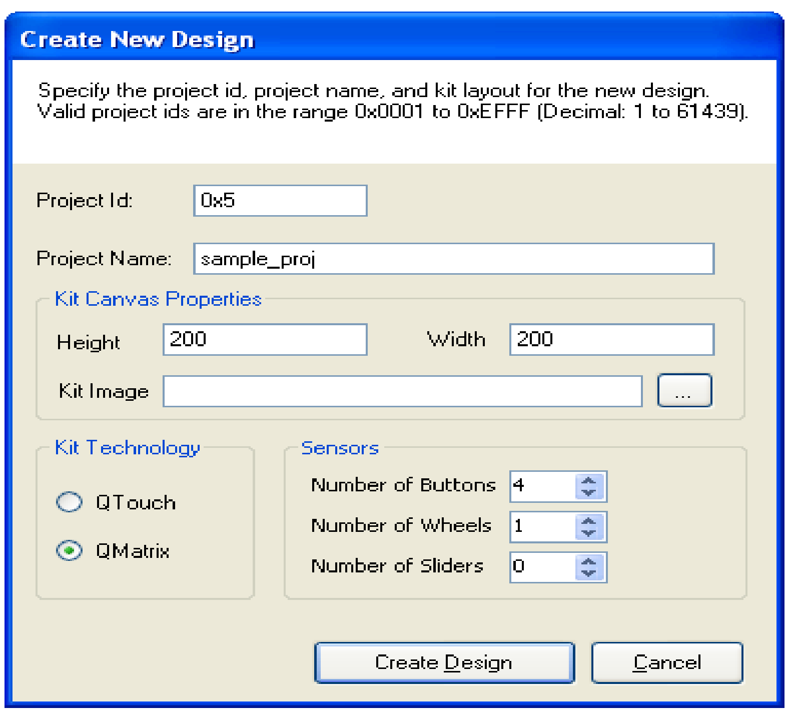


Figure 5-36 Creating New Design in the AVR QTouch Studio

After Creating Design , the new design mode shows the virtual kit view with sensors that have been created in some order.

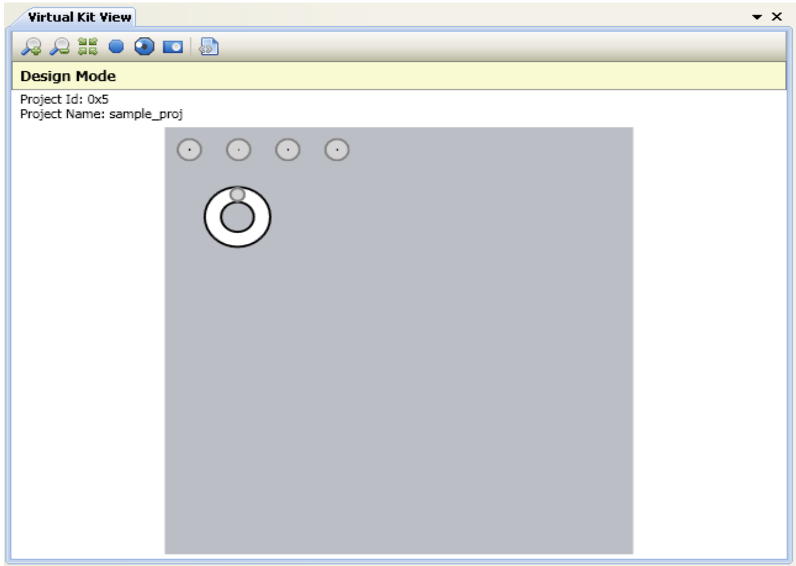


Figure 5-37: New Design Sensors in the AVR QTouch Studio

4. Now Go to Tools->Pin Configuration Wizard as shown below.

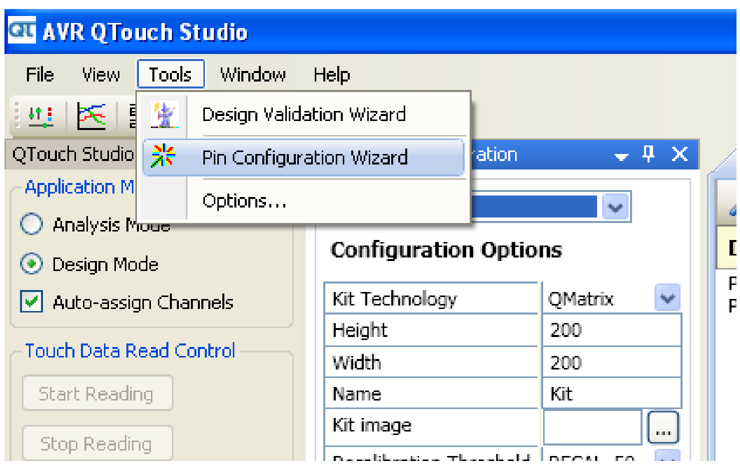


Figure 5-38: Selecting the pin configuration wizard

5. Pin configuration Window will pop up with the information on the usage of the tool.

Click Next to proceed.

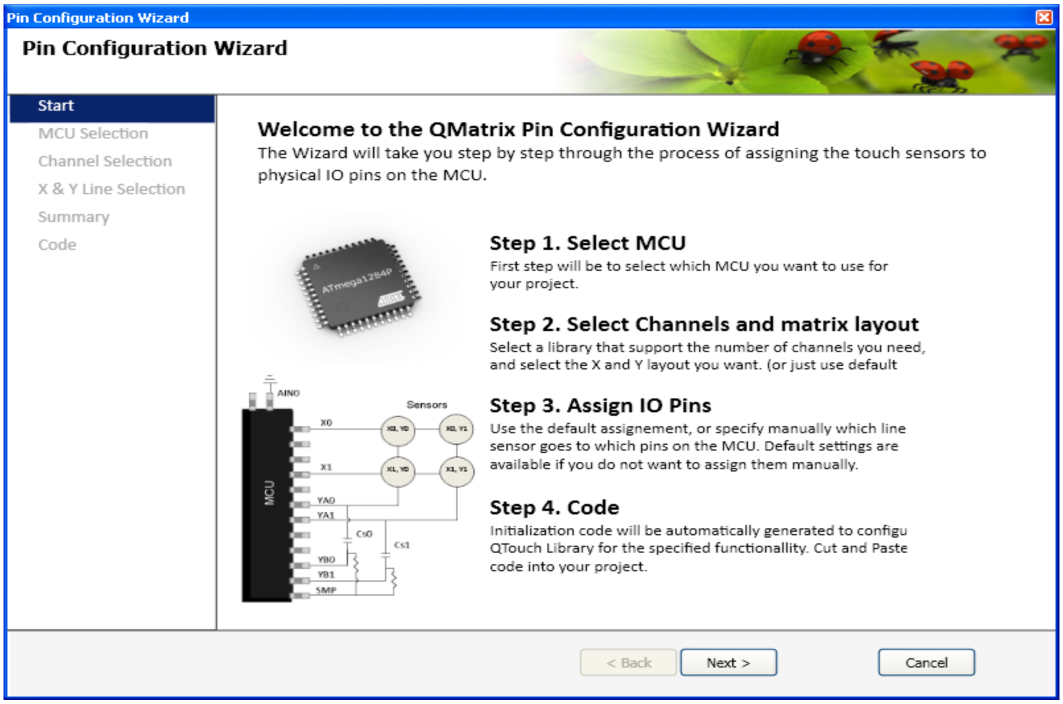


Figure 5-39: Start window of the configuration wizard

6. .Select the MCU and click Next as shown below.

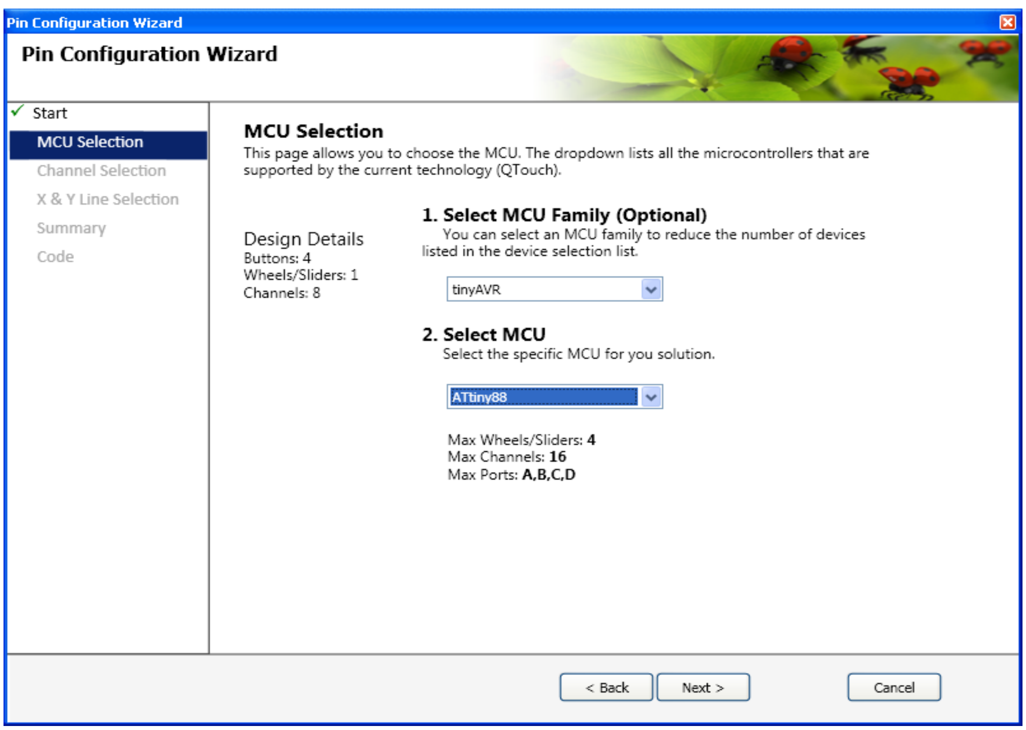


Figure 5-40: MCU selection window from the configuration wizard.

7. Select the Channels needed for the design from the list provided and click Next.

If 6 channels are needed, the next immediate value that suits the design needs to be selected. Ie., 8 channels (4 x 2) configuration.

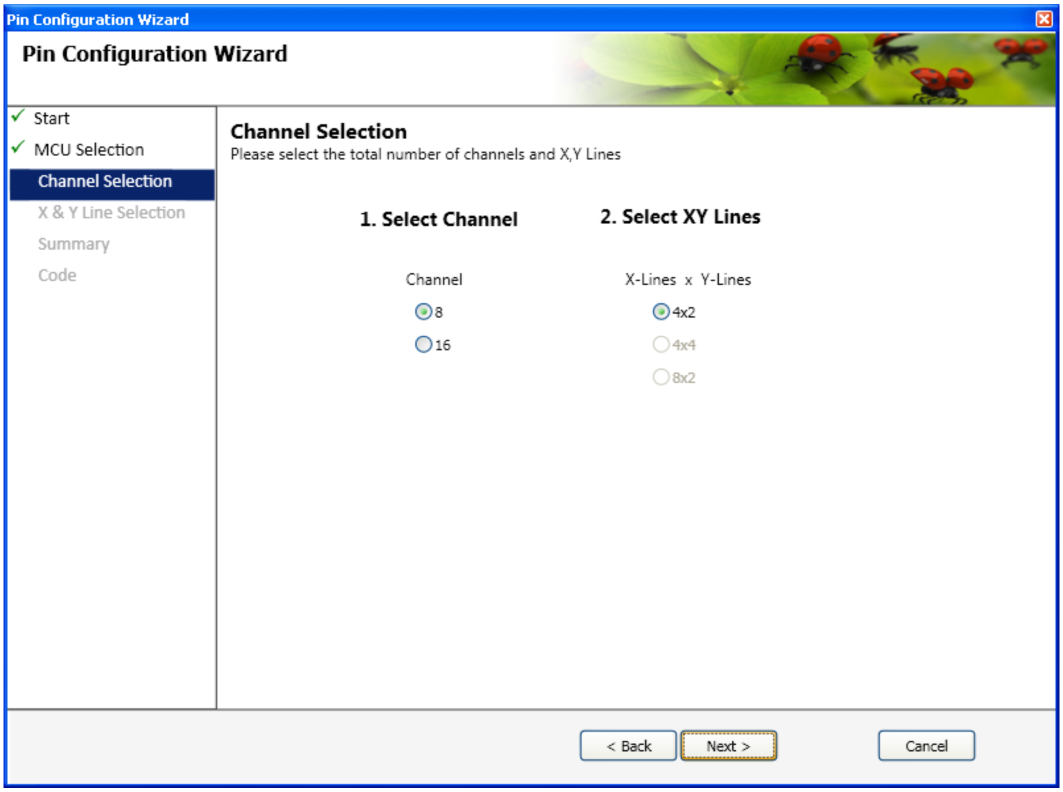


Figure 5-41: Selecting channels and configuration in the New Design

8. Select the pins used for the design and click Next.

If there is error in the selection of the pins (Ex: conflictin pins used), a red marker will be appear and the user cannot proceed to next step in configuration until the user has done the correct pin selection.

Now once the selection is done without errors, Click Next.

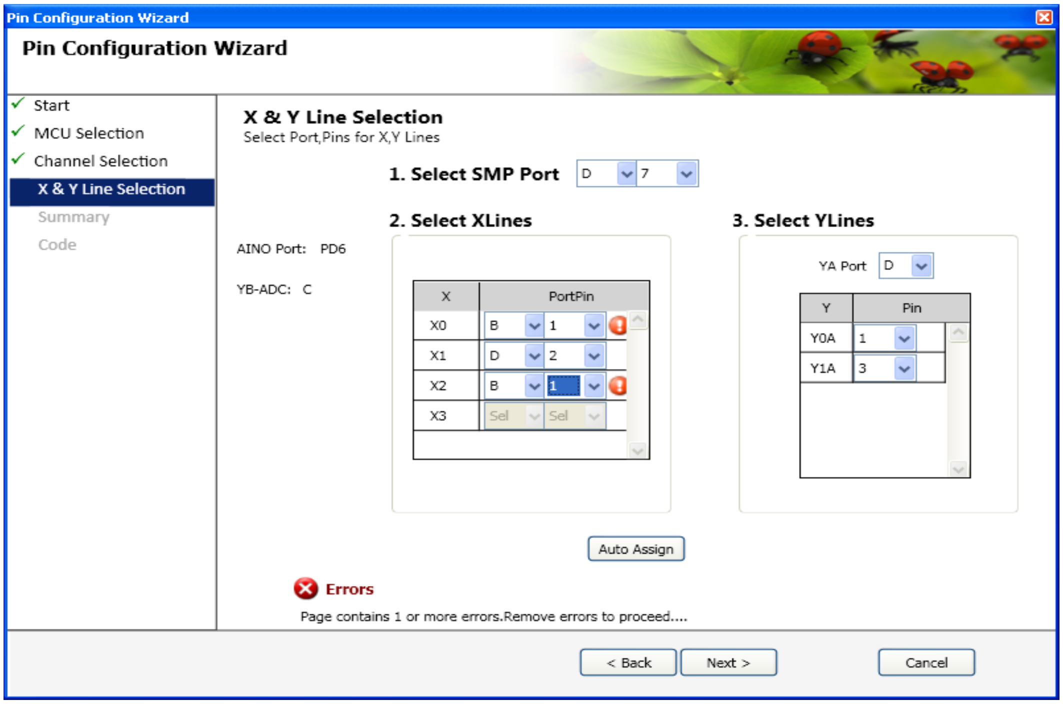


Figure 5-42: Selecting the X,YA,YB,SMP Pins in the new Design with errors.

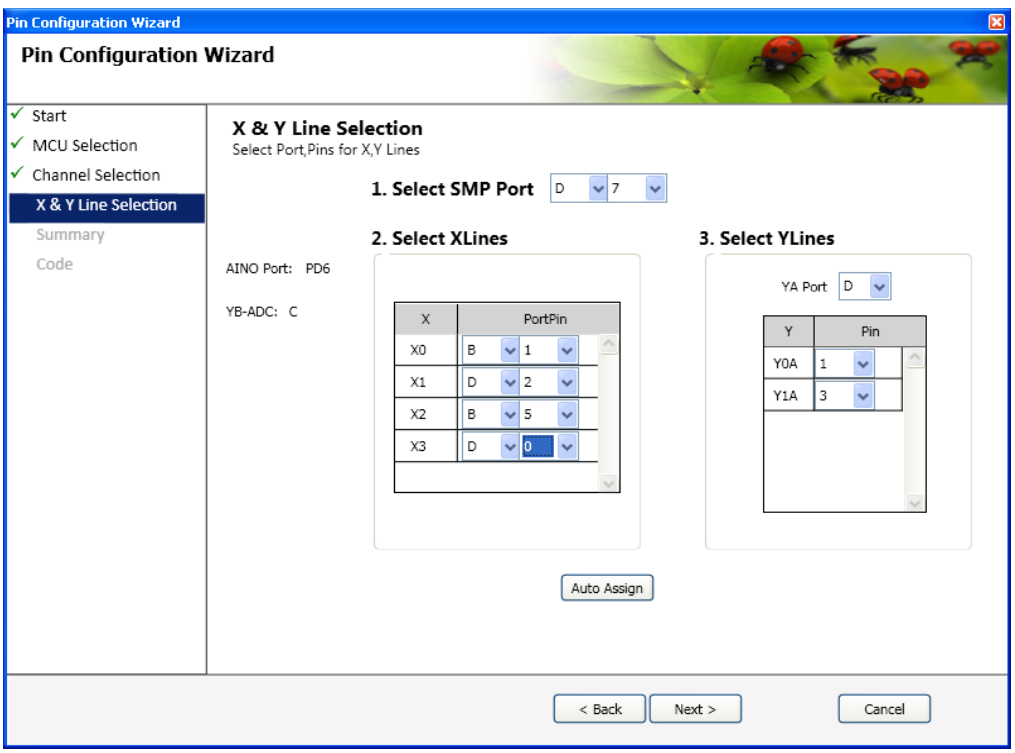


Figure 5-43: Selecting the X,YA,YB,SMP Pins in the new Design without errors.

Once the pins are selected, Pin Wizard will provide the summary report .Check whether details are correct as specified.Click Next.

If there are some errors that are found in the summary report, the user can click "back" button and modify the changes needed.

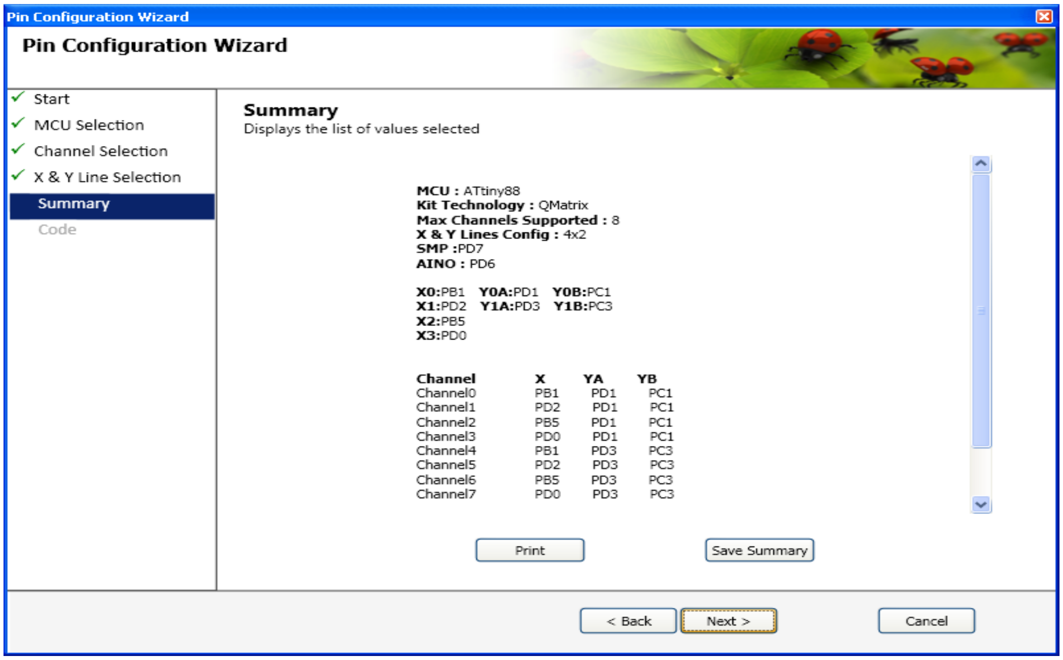


Figure 5-44: Summary report

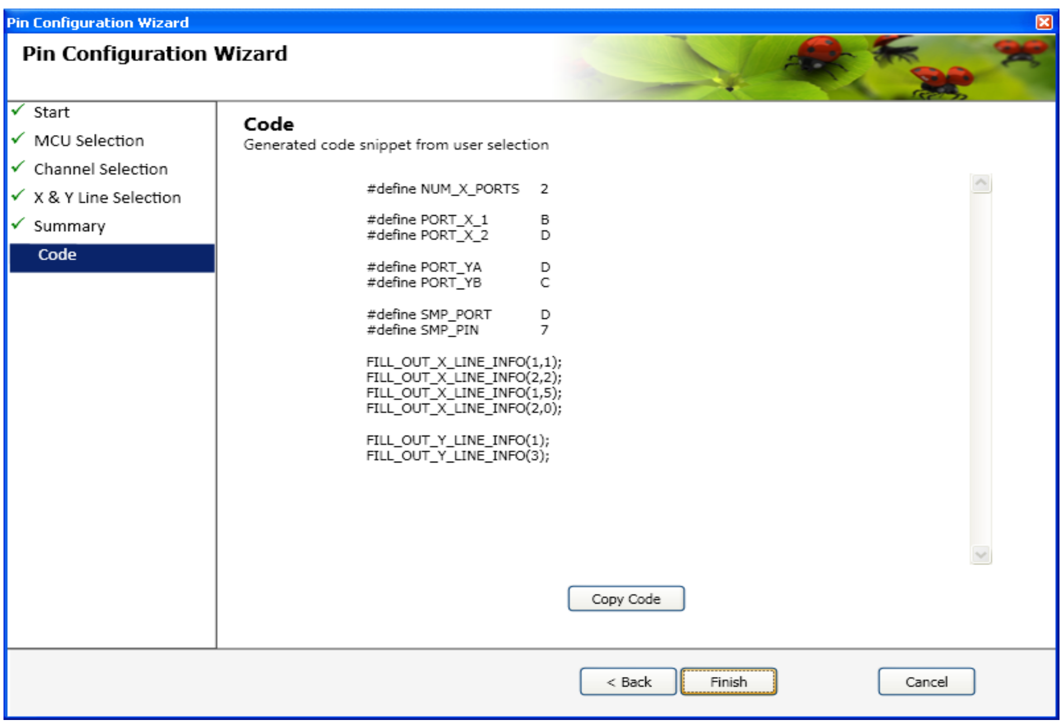


Figure 5-45: Code Generation tab in Pin Configuration wizard

9. The code is shown in the New Window Screen.

Note: Use FILL_OUT_YA_LINE_INFO and FILL_OUT_YB_LINE_INFO code instead of

FILL_OUT_Y_LINE_INFO as now YA and YB can be shared on same port.

The code can be copied using the “copy code” and pasted in the main.c and touch_config .h file,

a. In ***touch _config.h,***

Copy the following header definitions as part of the preprocessor directives in the project space or in the beginning of the file

```
#define NUM_X_PORTS   2
#define PORT_X_1      B
#define PORT_X_2      D
#define PORT_YA       D
#define PORT_YB       C
#define SMP_PORT      D
#define SMP_PIN       7
#define SHARED_YAYB   0
```

b. In ***main.c*** ,

Copy the code as below

```
x_line_info_t x_line_info[NUM_X_LINES]= {
    FILL_OUT_X_LINE_INFO(1,1);
    FILL_OUT_X_LINE_INFO(2,2);
    FILL_OUT_X_LINE_INFO(1,3);
    FILL_OUT_X_LINE_INFO(2,0);
};
y_line_info_t ya_line_info[NUM_Y_LINES]= {
    FILL_OUT_YA_LINE_INFO(1);
    FILL_OUT_YA_LINE_INFO(3);
};
y_line_info_t yb_line_info[NUM_Y_LINES]= {
    FILL_OUT_YB_LINE_INFO(1);
    FILL_OUT_YB_LINE_INFO(3);
};
```

Note: If YA,YB shared on same port feature is needed , then apart from the above Macros #define SHARED_YAYB needs to be enabled as 1.

5.9 MISRA Compliance Report

This section lists the compliance and deviations for MISRA standards of coding practice for the QTouch and QMatrix acquisition method libraries.

5.9.1 What is covered

The QTouch and QMatrix acquisition method libraries adhere to the MISRA standards. The additional reference code provided in the form of sample applications is not guaranteed to be MISRA compliant.

5.9.2 Target Environment

Development Environment	IAR Embedded Workbench
MISRA Checking software	The MISRA C Compliance has been performed for the library using MISRA C 2004 Rules in IAR Workbench development environment.
MISRA Rule set applied	MISRAC 2004 Rule Set

5.9.3 Deviations from MISRA C Standards

5.9.3.1 QTouch acquisition method libraries

The QTouch acquisition method libraries were subject to the above mentioned MISRA compliance rules. The following exceptions have not been fixed as they are required for the implementation of the library.

Applicable Release	QTouch libraries version 5.0	
Rule No	**Rule Description**	**Exception noted / How it is addressed**
1.1	Rule states that all code shall conform to ISO 9899 standard C, with no extensions permitted.	This Rule is not supported as the library implementation requires IAR extensions like __interrupt. These intrinsic functions relate to device hardware functionality, and cannot practically be avoided.
10.1	Rule states that implicit conversion from Underlying long to unsigned long	The library uses macros to combine symbol definitions to form a unique expanded symbol name and in this, the usage of unsigned qualifiers for numeric constants (e.g. 98u) causes name mangling. This is the only occurrence of this error in the library.
10.6	This Rule says that a 'U' suffix shall be applied to all constants of 'unsigned' type	The library uses macros to combine symbol definitions to form a unique expanded symbol name and in this, the usage of unsigned qualifiers for numeric constants (e.g. 98u) causes name mangling. This is the only occurrence of this error in the library.
14.4	Rule states that go-to statement should not be used.	The library uses conditional jump instructions to reduce the code footprint at a few locations and this is localized to small snippets of code. Hence this rule is not supported.
19.10	Rule states that In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis	There is one instance where the library breaks this rule where two macro definitions are combined to form a different symbol name. Usage of parenthesis cannot be used in this scenario.
19.12	Rule states that there shall be at most one occurrence of the # or ## preprocessor operator in a single macro definition	There is one instance in the library where this rule is violated where the library concatenates two macro definitions to arrive at a different definition.

5.9.3.2 QMatrix acquisition method libraries

The QMatrix acquisition method software was subject to the above mentioned MISRA compliance rules. The following exceptions have not been fixed as they are required for the implementation of the library.

Applicable release	**QTouch libraries ver 5.0**	
Rule No	**Rule Description**	**Exceptions Reason**
1.1	Rule states that all code shall conform to ISO 9899 standard C, with no extensions permitted.	This Rule is not supported as the library implementation requires IAR extensions like __interrupt. These intrinsic functions relates to device hardware functionality, and cannot practically be avoided
10.1	Rule states that Illegal implicit conversion from Underlying long to unsigned long	The library uses macros to combine symbol definitions to form a unique expanded symbol name and in this, the usage of unsigned qualifiers for numeric constants (e.g. 98u) causes name mangling. This is the only occurrence of this error in the library.
10.6	This Rule says that a 'U' suffix shall be applied to all constants of 'unsigned' type	The library uses macros to combine symbol definitions to form a unique expanded symbol name and in this, the usage of unsigned qualifiers for numeric constants (e.g. 98u) causes name mangling. This is the only occurrence of this error in the library.
19.10	Rule states that In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis	There is one instance where the library breaks this rule where two macro definitions are combined to form a different symbol name. Usage of parenthesis cannot be used in this scenario.
19.12	Rule states that there shall be at most one occurrence of the # or ## preprocessor operator in a single macro definition	There is one instance in the library where this rule is violated where the library concatenates two macro definitions to arrive at a different definition.

5.10 Known Issues

Issue	**Cause**	**Remedy / workaround**
Buiding QTouch Libraries Release 5.0 with WinAVR Compiler results in Linker Error. (Skipping Library libavrxxx.a, File not found).	QTouch Libraries Release 5.0 are build with Atmel Studio6 Native Toolchain Flavor.	Always use Native Toolchain Flavor (Advanced Tab in Project properties) setting in Atmel Studio6 for Building QTouch Library Release.
The GCC example projects for QMatrix does not compile the delay cycles (QT_DELAY_CYCLES) above a value of 5 because of the preprocessor expansions.		Recommended to remove UL from the preprocessor constants and in the chain of macros used for QT_DELAY_CYCLES. Valid for QT_DELAY_CYCLES = 5,10,25,50.
Compiling QT600 project files throws		These variables are available in

unused variable warning.		the debug protocol for future use.
When using IAR workbench for ATSAM to integrate the touch libraries, the linker would generate a warning indicating: Warning[Lp005]: placement includes a mix of sections with content (example "ro data section .data_init in xfiles.o(dl7M_tl_if.a)") and sections without content (example "rw data section .data in xfiles.o(dl7M_tl_if.a)") Warning[Lp006]: placement includes a mix of writable sections (example "rw data section .data in xfiles.o(dl7M_tl_if.a)") and non-writable sections (example "ro data section .data_init in xfiles.o(dl7M_tl_if.a)")	This is because we link the library in RW data section.	

5.11 Checklist

This section lists troubleshooting tips and common configuration tips.

Symptom	**Cause**	**Action**
Sensors do not go into detect or have unknown results	Multiplexing pins used by QTouch libraries in your design	Check the Pins used for QTouch or QMatrix acquisition methods do not overlap with the applications usage of the ports
Signal values report arbitrary values	Stray capacitance	Check the sensor design and minimize stray capacitance interference in your design
Waveforms of charging / discharging of channels do not show up properly in oscilloscopes	JTAG ICE connected to the board	Try disconnecting the JTAG ICE completely from the kit
When using the example applications, the debug values for some of the channels does not display appropriate values	JTAG Pins are enabled in the target.	JTAG Pins are explicitly needs to be disabled in the main.c file /* disable JTAG pins */ MCUCR |= (1u << JTD); MCUCR |= (1u << JTD)

6 Device Specific Libraries

6.1 Introduction

This section provides an overview of the usage of Device specific QTouch Libraries. Device Specific Libraries have been provided for special devices, which are not covered as part of Generic Libraries.

6.2 Devices supported

The following devices are covered by the Device Specific QTouch Libraries.

1. AT32UC3L0, AT32UCL3U/L4U family devices.
2. ATtiny10, ATtiny20 and ATtiny40 devices.

6.3 QTouch Library for AT32UC3L devices

ATMEL QTouch Library for UC3L can be used for embedding capacitive touch buttons, sliders and wheels functionality into UC3L application. The QTouch Library for UC3L uses the Capacitive Touch Module (CAT) that senses touch on external capacitive touch sensors.

This Section describes the QTouch Library Application Programming Interface (API) for QMatrix and QTouch method acquisition using the AT32UC3L devices.

6.3.1 Salient Features of QTouch Library for UC3L

6.3.1.1 QMatrix method sensor

- N Touch Channels formed by an X by Y matrix require (X+2Y+1) physical pins (when using internal discharge mode), N=X*Y. Please refer Figure 37 for pin requirements in different modes.
- 1 to 136 Touch Channels can be configured.
- Max X Lines = 17, Max Y Lines = 8.
- Button is formed using 1 Touch Channel.
- Slider is formed using 3 to 8 Touch Channels.
- Wheel is formed using 3 to 8 Touch Channels.

6.3.1.2 QTouch method sensor

- 2 Physical pins per Touch Channel.
- QTouch Sensors can be divided into two groups Group A and Group B.
- Each QTouch group can be configured with different properties.
- 1 to 17 Touch Channels can be configured.
- Button is formed using 1 Touch Channel.
- Slider is formed using 3 Touch Channels.

- Wheel is formed using 3 Touch Channels.

6.3.1.3 Autonomous QTouch sensor

- A Single QTouch sensor that is capable of detecting touch or proximity without CPU intervention.
- Allows proximity or activation detection in low-power sleep modes.

6.3.1.4 Additional Features

- Standalone QMatrix, QTouch Group A/B or Autonomous QTouch operation.
- Support for operation of two or more methods at the same time.
    - Scenario 1: QMatrix and Autonomous QTouch method at the same time.
    - Scenario 2: QTouch Group A, QTouch Group B & Autonomous QTouch at the same time.
    - Scenario 3: QMatrix, QTouch Group A/B and Autonomous QTouch at the same time.
- Disable/Re-enable Sensors at any given time for reduced power consumption.
- Raw data acquisition mode without any post-processing of data.
- External synchronization to reduce 50 or 60 Hz mains interference.
- Spread spectrum sensor drive capability.
- QTouch Studio-Touch Analyzer support to fine tune touch implementation.
- IAR and GCC Tool chain support.
- MISRA Compliant, MISRAC 2004 Rule Set.
- Single Library for QMatrix, QTouch Group A/B and Autonomous QTouch methods.

6.3.2 Device variants supported for UC3L

Below is the list of different devices in AT32UC3L family that is supported by the QTouch library.

1. AT32UC3L016, AT32UC3L032, AT32UC3L064
2. ATUC64L3U, ATUC128L3U, ATUC256L3U (Studio 6 Support only.)
3. ATUC64L4U, ATUC128L4U, ATUC256L4U (Studio 6 Support only.)

For capacitive touch sensing module related information Refer to, “Capacitive touch module (CAT)” of the datasheet.

6.3.3 Development tool support for UC3L

The QTouch libraries for AT32UC3L devices are supported for the following development tools.

Tool	Version
IAR Embedded Workbench for Atmel AVR32. IAR32 Compiler.	4.1
Atmel Studio 6. GCC Compiler	6.0

Table 8 Development tool support for UC3L QTouch Library

6.3.4 Overview of QTouch Library API for UC3L

The diagram below captures the high level arrangement of the QTouch Library for UC3L API.

The QTouch Library for UC3L API can be used for Sensor configuration, Sensor Acquisition parameter setting and Sensor Enable/Disable operations. Based on this input Sensor configuration, the QTouch Library takes care of the initialization, configuration and acquisition data capture operations using the CAT module. The UC3L CAT module interfaces with the external capacitive touch sensors and is capable of performing QTouch and QMatrix method acquisition. For an Overview of QMatrix and QTouch Capacitive Touch acquisition, refer Section 5.2.

The raw acquisition data from the CAT module is processed by the QTouch Library. The Adjacent Key Suppression (AKS), Detect Integration mechanism, Drift compensation and Automatic Recalibration components of the Touch Library aid in providing a robust Touch performance. Once the raw acquisition data is processed, the individual Sensor Status and Wheel/Slider position information is provided to the user by means of a measurement complete callback operation.

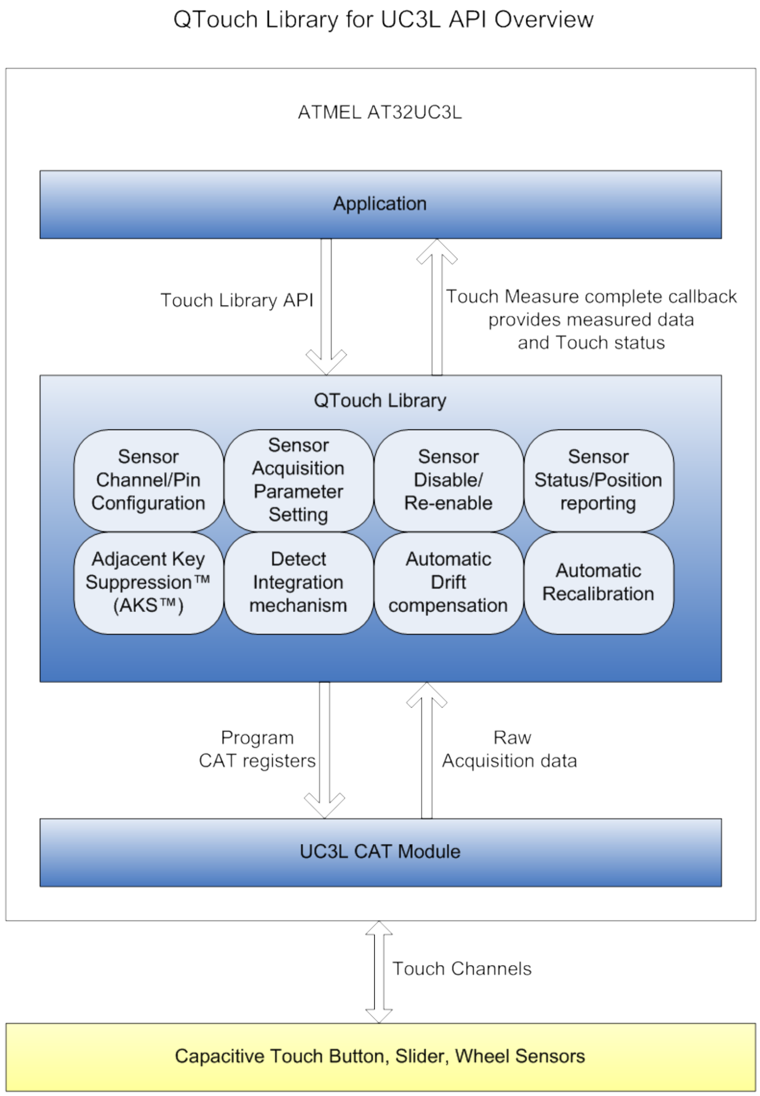


Figure 35 Overview diagram of QTouch Library for UC3L

6.3.5 Acquisition method support for UC3L

With the QTouch Library for UC3L, it is possible for a user to configure the following types of Sensors.

- QMatrix method sensors.
- QTouch Group A method sensors.
- QTouch Group B method sensors.
- Autonomous QTouch sensor.

The QTouch Library for UC3L API has been arranged such that it is possible for the user application either to use any of the above method Standalone or two or more methods combined together. The Table below captures the different API available under each method. For normal operation, it is only required to use the Regular API set for each method. By using only the Regular API set, it is possible to achieve reduced code memory usage when using the QTouch Library. The Helper API is provided for added flexibility to the user application.

Acquisition method	Regular API	Helper API
QMatrix method API	touch_qm_sensors_init touch_qm_sensor_config touch_qm_sensors_calibrate touch_qm_sensors_start_acquisition touch_event_dispatcher	touch_qm_sensor_update_config touch_qm_sensor_get_config touch_qm_channel_update_burstlen touch_qm_update_global_param touch_qm_get_global_param touch_qm_get_libinfo touch_qm_sensor_get_delta touch_deinit
QTouch Group A/B method API (The first parameter to the QTouch API, allows to distinguish between QTouch Group A and QTouch Group B.)	touch_qt_sensors_init touch_qt_sensor_config touch_qt_sensors_calibrate touch_qt_sensors_start_acquisition touch_event_dispatcher	touch_qt_sensor_update_config touch_qt_sensor_get_config touch_qt_update_global_param touch_qt_get_global_param touch_qt_get_libinfo touch_qt_sensor_get_delta touch_qt _sensor_disable touch_qt _sensor_reenable touch_deinit
Autonomous QTouch API	touch_at_sensor_init touch_at_sensor_enable touch_at_sensor_disable	touch_at_sensor_update_config touch_at_sensor_get_config touch_at_get_libinfo touch_deinit

Table 9 Acquisition method specific API

6.3.6 API State machine for UC3L

The QTouch Library State machine diagram captures the different library States, Events that are allowed in each State and Event transition from one State to the other. The QTouch Library maintains the States of QMatrix, QTouch Group A and QTouch Group B methods independently. This means that QMatrix can be in a state that is different from the state of QTouch Group A or B and vice versa.

For the case of Autonomous QTouch, only the TOUCH_STATE_NULL and TOUCH_STATE_INIT states apply in the State diagram.

- The touch_at_sensor_init event causes a transition from TOUCH_STATE_NULL to TOUCH_STATE_INIT.
- The touch_deinit event causes a transition from TOUCH_STATE_INIT to TOUCH_STATE_NULL.

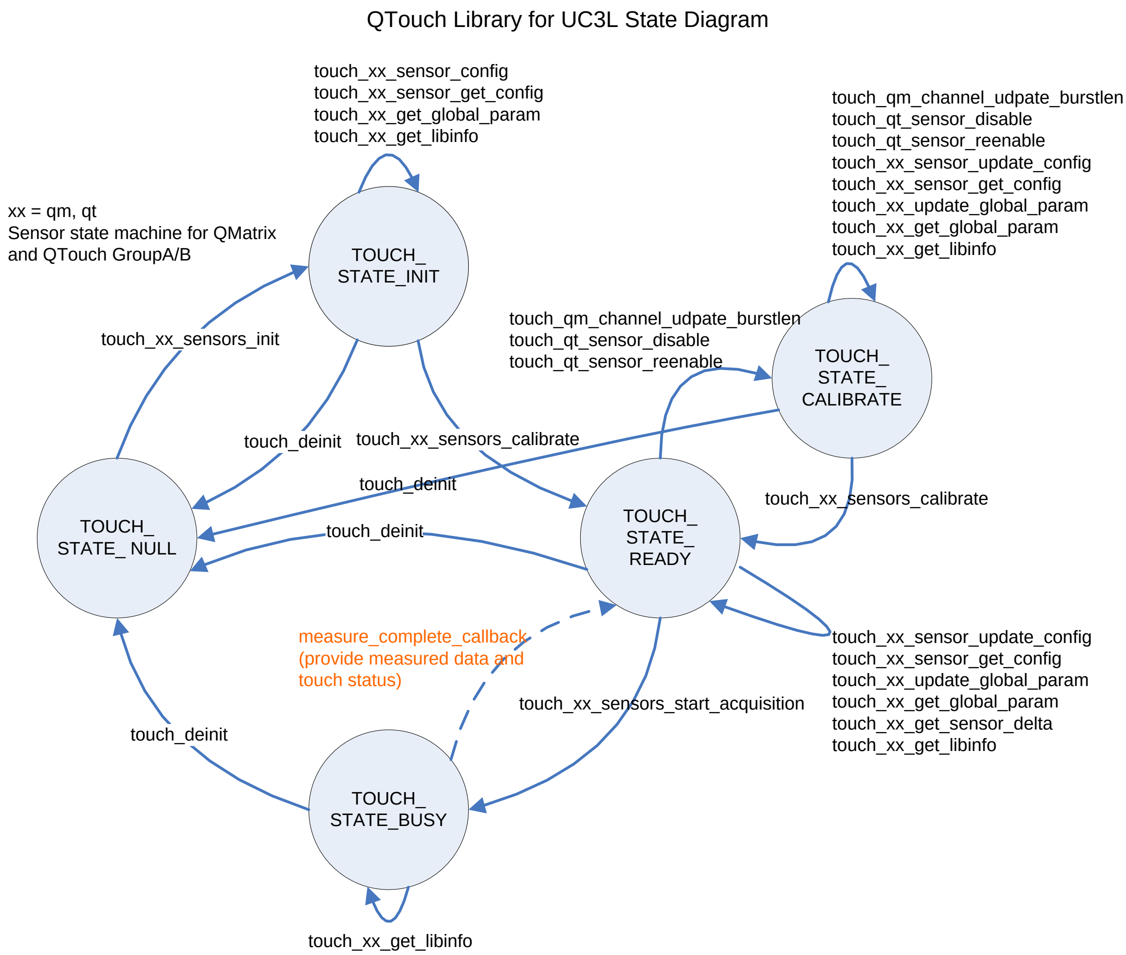


Figure 36 State Diagram of QTouch Library for UC3L

6.3.7 QMatrix method sensor operation for UC3L

6.3.7.1 QMatrix method pin selection for UC3L

Please refer AT32UC3L datasheet Table 28-2 Pin Selection Guide and Table 3-1 GPIO Controller Function multiplexing, for mapping between the QMatrix method pin name and the GPIO pin. It is possible to configure a maximum of 17 X Lines and 8 Y-Yk pairs. The X Line X8 (PA16) cannot be used for the QMatrix method as it is required to use this pin for the ACREFN function.

The CAT module provides an option to enable a nominal output resistance of 1kOhm on specific CAT module pins during the burst phase. The Table below captures the different QMatrix method pin wherein a Resistive Drive can be optionally enabled. The rows marked with Grey indicate that Resistive Drive option is not available on that pin. By carefully choosing the QMatrix method

X and Yk pins wherein Resistive Drive can be enabled, saving on external components is possible.

Section 6.3.1.1 provides detail on the number of Pin and Touch channels required for different QMatrix method sensor. The hardware arrangement for Wheel or Slider must be such that all Touch channels corresponding to the Wheel or Slider belong to the same Yk Line.

Also, Section 6.3.11 indicates the various Pin Configuration options for the QTouch Library that can be used to specify a user defined configuration.

CAT Module Pin Name	QMatrix method Pin Name
CSA0	X0
CSB0	X1
CSA1	Y0
CSB1	YK0
CSA2	X2
CSB2	X3
CSA3	Y1
CSB3	YK1
CSA4	X4
CSB4	X5
CSA5	Y2
CSB5	YK2
CSA6	X6
CSB6	X7
CSA7	Y3
CSB7	YK3
CSA8	X8
CSB8	X9
CSA9	Y4
CSB9	YK4
CSA10	X10
CSB10	X11
CSA11	Y5
CSB11	YK5
CSA12	X12
CSB12	X13
CSA13	Y6
CSB13	YK6
CSA14	X14
CSB14	X15
CSA15	Y7
CSB15	YK7
CSA16	X16
CSB16	X17

Table 10 QMatrix Resistive drive pin option

(The rows marked with Grey indicate that Resistive Drive option is not available on that pin)

6.3.7.2 QMatrix method Schematic for UC3L

6.3.7.2.1 Internal Discharge mode

The CAT module provides an internal discharge arrangement for QMatrix method. When this arrangement is used along with the Resistive drive capability, minimal external component is required as shown in the case A of Figure 27. When the Resistive drive is option is not enabled, it is recommended to use 1kOhm resistors on X and Yk Lines external to the UC3L device. This hardware arrangement is shown in case B.

6.3.7.2.2 External Discharge mode

When the External Discharge arrangement is used, a logic-level (DIS) pin is connected to an external resistor (Rdis) that can be used to control the discharge of the Capacitors. A typical value for Rdis is 100 kOhm. This value of Rdis will give a discharge current of approximately 1.1V/(100 kOhm) = 11 microAmp. The case C shows this arrangement. The Resistive drive option on the X and Yk lines can be optionally enabled or disabled with this arrangement. When the Resistive drive is option is not enabled, it is recommended to use 1kOhm resistors on X and Yk Lines external to the UC3L device.

6.3.7.2.3 SMP Discharge Mode

When the SMP Discharge mode arrangement is used, a logic-level (SMP) pin is connected to the capacitors through external high value resistors for the discharge of the capacitors. The case D shows this arrangement. The Resistive drive option on the X and Yk lines can be optionally enabled or disabled with this arrangement. When the Resistive drive is option is not enabled, it is recommended to use 1kOhm resistors on X and Yk Lines external to the UC3L device.

6.3.7.2.4 VDIVEN Voltage Divider Enable option

The VDIV pin provides an option to make ACREFN a small positive voltage if required. The VDIV pin is driven when the analog comparators are in use, and this signal can be used along with a voltage divider arrangement to create a small positive offset on the ACREFN pin. The VDIVEN option can be used optionally with any of the QMatrix modes discussed in the previous sections. Typical values for Ra and Rb are Ra=8200 ohm and Rb = 50 ohm. Assuming a 3.3V I/O supply, this will shift the comparator threshold by 3.3V*(Rb/(Ra+Rb)) which is 20 mV. The VDIVEN pin option usage in the Internal Discharge mode scenario is shown in case E.

6.3.7.2.5 SYNC pin option

In order to prevent interference from the 50 or 60 Hz mains line the CAT can optionally trigger QMatrix acquisition on the external SYNC input signal. The SYNC signal should be derived from the mains line and the acquisition will trigger on a falling edge of this signal. The SYNC pin option can be used with any of the QMatrix modes discussed in the previous sections. The SYNC pin usage in the Internal Discharge mode scenario is shown in case F.

For QMatrix method SMP, DIS, VDIV and SYNC pin options discussed in this Section, Refer to Section 6.3.15.2.13.

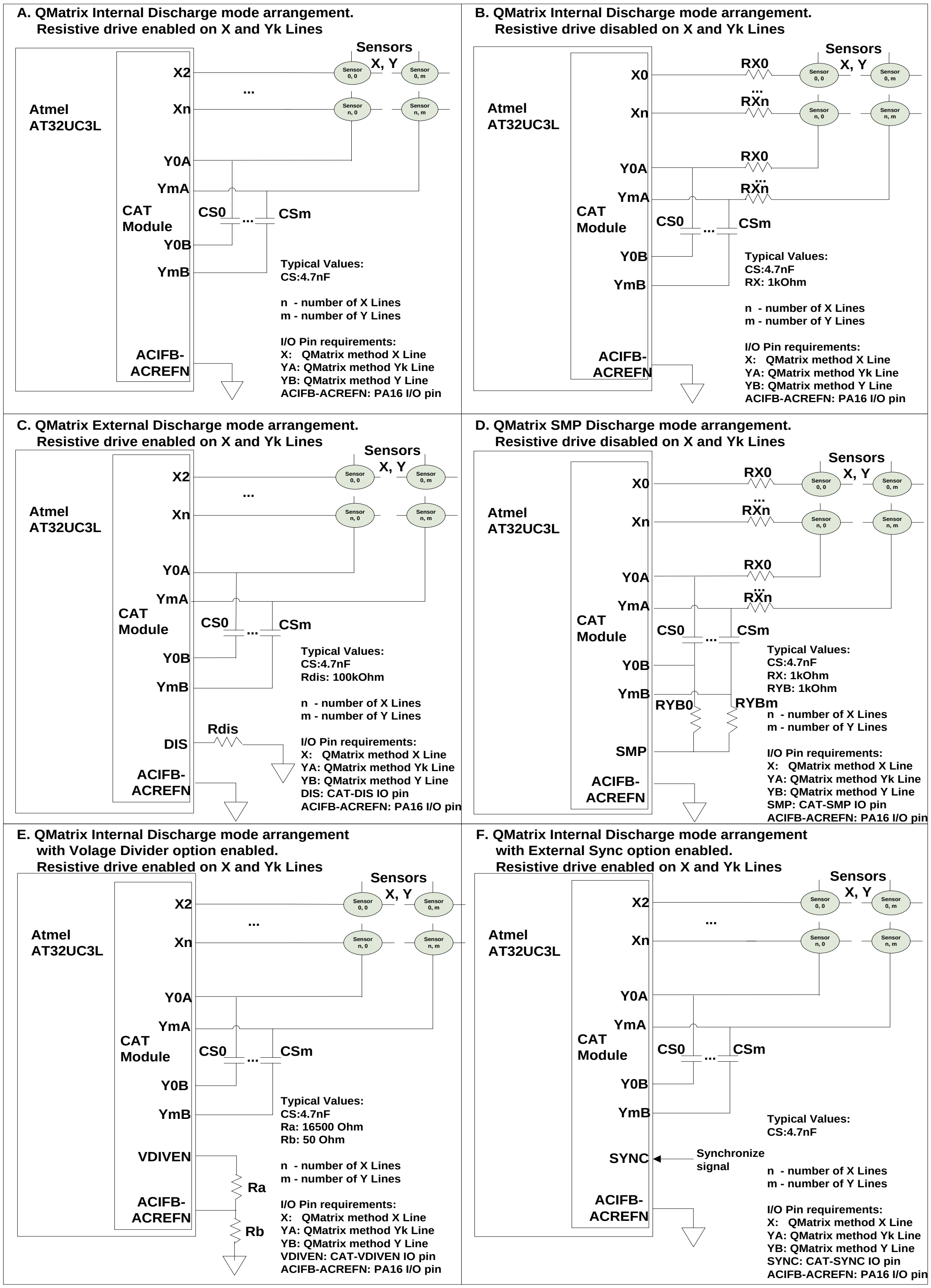


Figure 37 QMatrix method schematic

6.3.7.3 QMatrix method hardware resource requirement for UC3L

The clock for the CAT module, CLK_CAT, is generated by the Power Manager (PM). This clock is turned on by default, and can be enabled and disabled in the PM. The user must ensure that CLK_CAT is enabled before initializing the QTouch Library.

QMatrix operations also require the CAT generic clock, GCLK_CAT. This generic clock is generated by the System Control Interface (SCIF), and is shared between the CAT and the Analog Comparator Interface. The user must ensure that the GCLK_CAT is enabled in the SCIF before using QMatrix functionality. For proper QMatrix operation, the frequency of GCLK_CAT must be less than one fourth the frequency of CLK_CAT.

For QMatrix operation, the Analog comparators channels are used (using the ACIFB interface) depending on the Y Lines enabled. See Note 4 in Section 6.3.7.4.

The QMatrix method acquisition using the CAT module requires two Peripheral DMA channels that must be provided by the application.

6.3.7.4 QMatrix method Channel and Sensor numbering for UC3L

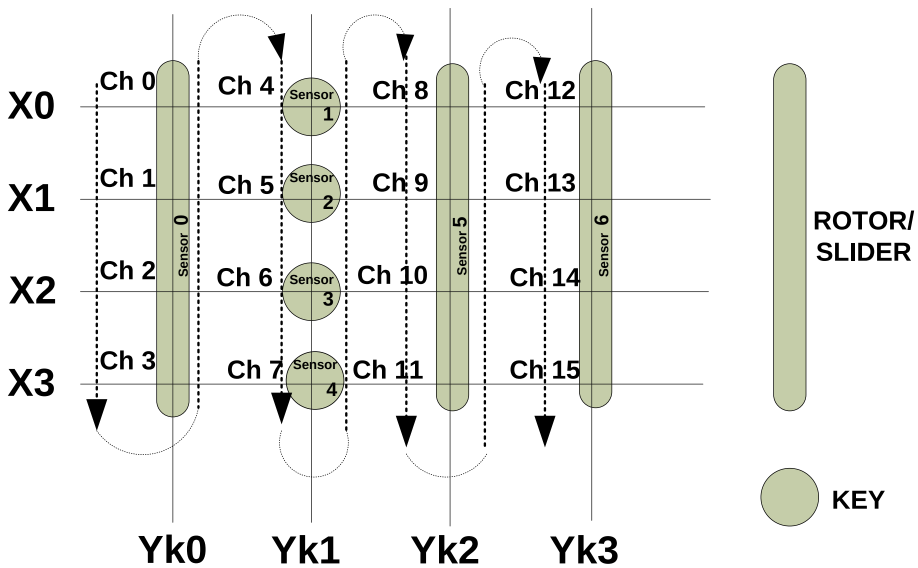


Figure 38 QMatrix channel numbering for UC3L

The above figure represents a typical 4 X 4 matrix of QMatrix sensor arrangement along with the channel numbers. The Channel numbering starts with Channel 0 (Ch0) and increase sequentially from Ch0 to Ch15. Similarly the Sensor numbering starts with Sensor 0. The Channel number signifies the order in which the QTouch Library stores the acquisition data in the memory.

Note: The touch_qm_sensor_config API must follow the above Channel and Sensor numbering when configuring the Sensors.

6.3.7.5 QMatrix method API Flow for UC3L

For the QMatrix operation, the CAT_CLK and GCLK_CAT clocks must be setup appropriately as a first step. The QMatrix and Common configuration parameters in the touch_config_at32uc3l.h configuration must then be set up.

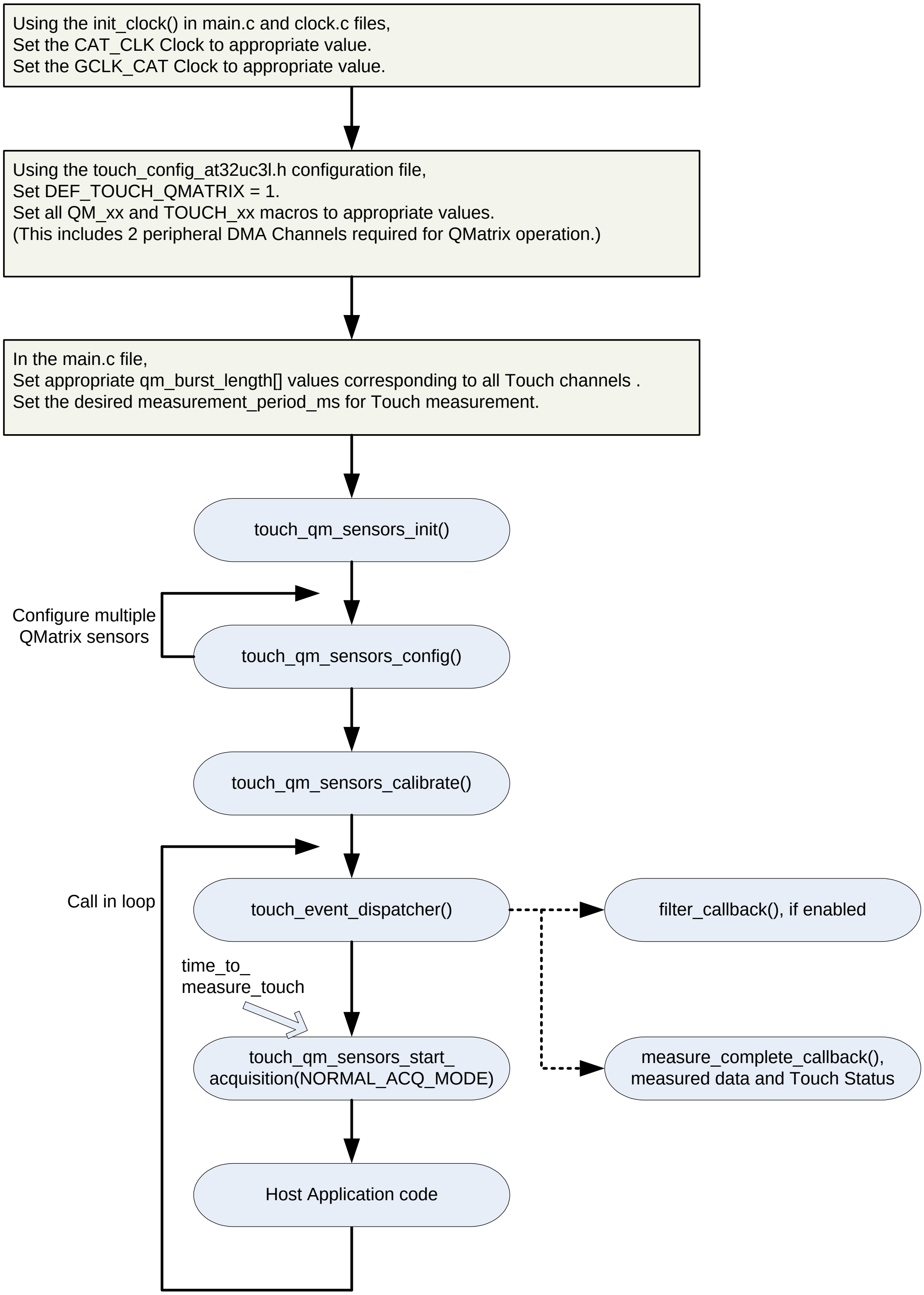


Figure 39 QMatrix API Flow diagram for UC3L

The burst length values of each Touch channel must be specified using the qm_burst_length[] array in the main.c file. The burst length must be specified in the same order of Touch Channel numbering.

The touch_qm_sensors_init API initializes the QTouch Library as well as the CAT module and does the QMatrix method specific pin, register and Global Sensor configuration. The touch_qm_sensor_config API is used to configure individual sensor. The Sensor specific configuration parameter can be provided as input to this API.

The touch_qm_sensors_calibrate API is used to calibrate all the configured sensors thereby preparing the sensors for acquisition. The touch_qm_sensors_start_acquisition API initiates a QMatrix method measurement on all the configured Sensors. This API takes the peripheral DMA channels as an input. When a filter_callback function is enabled, the touch_event_dispatcher function calls the filter_callback function as soon as the raw acquisition data from the Sensors is available. The user can now optionally apply any filtering routine on the raw acquisition data before the QTouch Library does any processing on this data. (For an overview of Filter callback usage, refer Section 5.6.6.4 Example code). Once the QTouch Library has finished processing the acquisition data from Sensors, the touch_event_dispatcher function calls the measure_complete_callback function indicating the end of a single Touch measurement operation. The measure_complete_callback provides the measured data and Touch status information. The measured data is available in the same order of Touch Channel numbering.

Note 1: The Host Application code can execute once a QMatrix acquisition is initiated with the touch_qm_sensors_start_acqusition API. Care must be taken in the Host Application such that the touch_event_dispatcher function is called frequently in order to process the acquired data. For a single Touch measurement operation (between a touch_qm_sensors_start_acquisition API call and the measure_complete_callback function being called), the touch_event_dispatcher function may execute multiple times in order to resolve the Touch status of Sensors. Failing to call the touch_event_dispatcher frequently can adversely impact the Touch Sensitivity.

Note 2: Once the Touch Library has been initialized for QMatrix method using the touch_qm_sensors_init API, a new qm_burst_length[x] value of a Touch channel must be updated only using the touch_qm_channel_update_burstlen API. It is recommended to have qm_burst_length array as global variable as the Touch Library updates this array when the touch_qm_channel_update_burstlen API is called.

Note 3: QMatrix burst length setting recommendation.

For a given X Line, the burst length value of ALL enabled Y Lines MUST be the same or set to 0x01(disabled). For example, the burst length value corresponding to (X0,Y1),(X0,Y2)...(X0,Yn) must be the same. In case of a scenario, wherein it is required to have a different a burst length, then the following option can be tried out - Enable the 1k ohm drive resistors on all the enabled Y lines by setting the corresponding bit in the CSARES register.

Note 4: For QMatrix operation, the Analog comparators channels are used (using the ACIFB interface) depending on the Y Lines enabled. For example, when Y lines Y2 and Y7 are enabled the Analog comparator channels 2 and 7 are used by the CAT module for QMatrix operation. The user can uses the rest of the Analog comparator channels in the main application. The QTouch Library enables the ACIFB using the Control register (if not already enabled by the main application) when the touch_qm_sensors_init API is called.

6.3.7.6 QMatrix method Disable and Re-enable Sensor for UC3L

The touch_qm_channel_update_burstlen API can be used for Disabling and Re-enabling of QMatrix Sensors. In order to Disable a sensor, the QMatrix burst length value of all the Touch Channels corresponding to the Sensor must be set to 1. For Example, when a Wheel or Slider is composed of 4 Touch Channels, the touch_qm_channel_update_burstlen API should be used to set the burst length of all the 4 Touch Channels to 1. For the case of a Button, touch_qm_channel_update_burstlen API should be used to set burst length of the corresponding single Touch Channel of the Button to 1. Similarly, when re-enabling a Sensor, appropriate burst length must be set to all the Touch channels corresponding to the Sensor.

When a QMatrix Sensor is Disabled or re-enabled, it is mandatory to force Calibration on all Sensors. The Calibration of all Sensors is done using the touch_qm_sensors_calibrate API.

Note: When disabling a Wheel or Slider, care must be taken to set the burst length of all the Touch channels corresponding to the Wheel or Slider to 1. If any of the Touch channels are missed out, it may result in undesired behavior of the Wheel or Slider. Similarly when re-enabling a Wheel or Slider, burst length of all the Touch channels corresponding to the Wheel or Slider must be set to an appropriate value. If any of the Touch Channels are left disabled with a burst length value 1, it may result in undesired behavior of Wheel or Slider.

6.3.8 QTouch Group A/B method sensor operation for UC3L

6.3.8.1 QTouch Group A/B method pin selection for UC3L

Please refer AT32UC3L datasheet Table 28-2 Pin Selection Guide and Table 3-1 GPIO Controller Function multiplexing, for mapping between the QTouch method pin name (SNS/SNSK) and the GPIO pin. The CAT module provides an option to enable a nominal output resistance of 1kOhm on specific CAT module pins during the burst phase. The Table below captures the different QTouch method pin wherein a Resistive Drive can be optionally enabled. The rows marked with Grey indicate that Resistive Drive option is not available on that pin. By carefully choosing the QTouch method SNSK pins wherein Resistive Drive can be enabled, saving on external components is possible. Section 6.3.1.2 provides detail on the number of Pin and Touch channels required for different QTouch method sensor. Also, Section 6.3.11 indicates the various Pin Configuration options for the QTouch Library that can be used to specify a user defined configuration.

CAT Module Pin Name	QTouch method Pin Name
CSA0	SNS0
CSB0	SNSK0
CSA1	SNS1
CSB1	SNSK1
CSA2	SNS2
CSB2	SNSK2
CSA3	SNS3
CSB3	SNSK3
CSA4	SNS4
CSB4	SNSK4
CSA5	SNS5
CSB5	SNSK5
CSA6	SNS6
CSB6	SNSK6
CSA7	SNS7
CSB7	SNSK7
CSA8	SNS8
CSB8	SNSK8
CSA9	SNS9
CSB9	SNSK9
CSA10	SNS10
CSB10	SNSK10
CSA11	SNS11
CSB11	SNSK11
CSA12	SNS12
CSB12	SNSK12
CSA13	SNS13
CSB13	SNSK13
CSA14	SNS14
CSB14	SNSK14
CSA15	SNS15
CSB15	SNSK15
CSA16	SNS16
CSB16	SNSK16

Table 11 QTouch Resistive drive pin option

(The rows marked with Grey indicate that Resistive Drive option is not available on that pin.)

6.3.8.2 QTouch Group A/B method Schematic for UC3L

6.3.8.2.1 Resistive Drive option

The cases A and B of the Figure provide the schematic arrangement of QTouch Group A/B and Autonomous QTouch Sensors. In option A, Resistive drive is enabled on SNSK line. In case B, Resistive drive is disabled on the SNSK line and in this case, it is recommended to use 1kOhm resistors on SNSK Line external to the UC3L device.

6.3.8.2.2 SYNC pin option

In order to prevent interference from the 50 or 60 Hz mains line the CAT can optionally trigger QTouch Group A/B and Autonomous QTouch acquisition on the external SYNC input signal. The SYNC signal should be derived from the mains line and the acquisition will trigger on a falling edge of this signal. The SYNC pin usage in the Internal Discharge mode scenario is shown in case C. For QTouch method SYNC pin options Refer to Section 6.3.15.2.13.

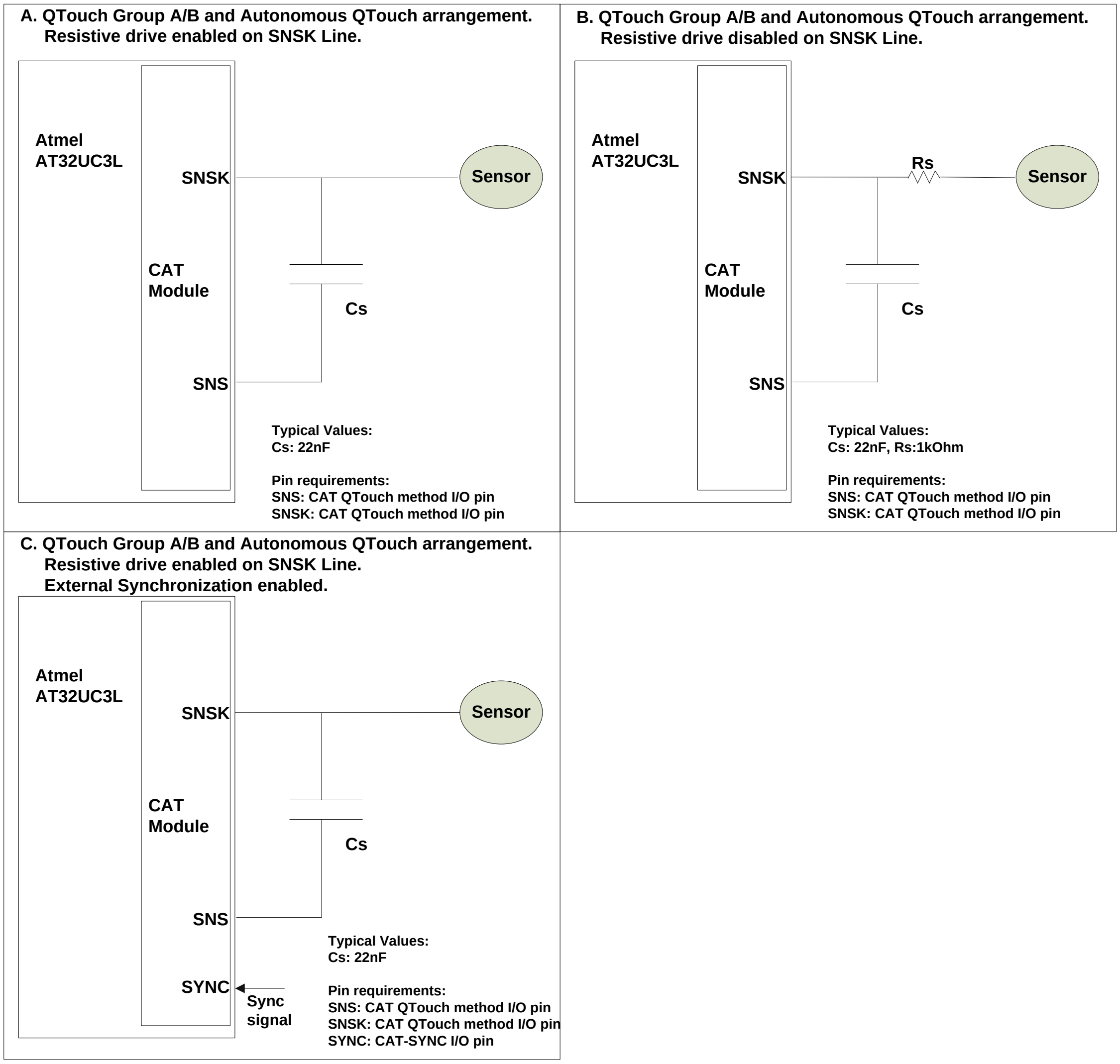


Figure 40 QTouch Group A/B and Autonomous QTouch schematic arrangement

6.3.8.3 QTouch Group A/B method hardware resource requirement for UC3L

The clock for the CAT module, CLK_CAT, is generated by the Power Manager (PM). This clock is turned on by default, and can be enabled and disabled in the PM. The user must ensure that CLK_CAT is enabled before initializing the QTouch Library.

The QTouch method acquisition using the CAT module requires one Peripheral DMA channel that must be provided by the application.

6.3.8.4 QTouch Group A/B method Channel and Sensor numbering for UC3L

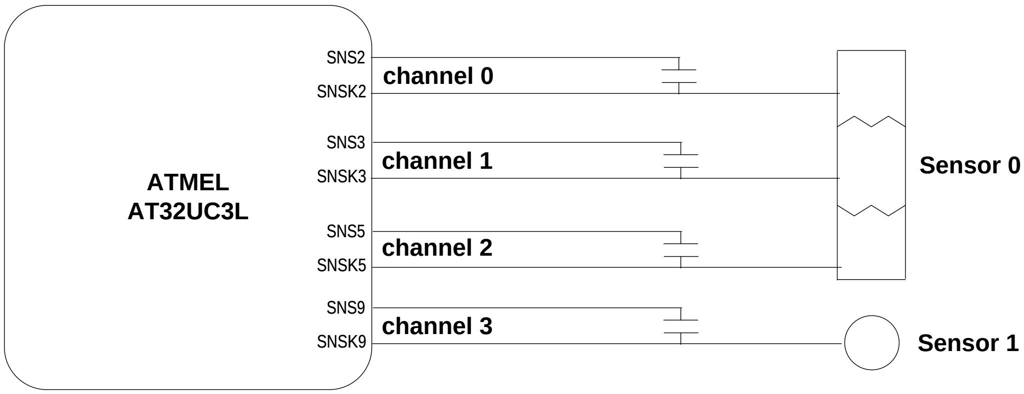


Figure 41 QTouch method Channel/Sensor numbering

The above Figure represents an example 4 Channel QTouch sensor arrangement along with the channel numbers. The Channel numbering starts with the lowest SNS-SNSK QTouch method pair number (SNS2-SNSK2 being the least in this case) and increases as the SNS-SNSK pair number increases. Similarly the Sensor numbering starts with Sensor 0. The Channel number signifies the order in which the QTouch Library stores the acquisition data in the memory.

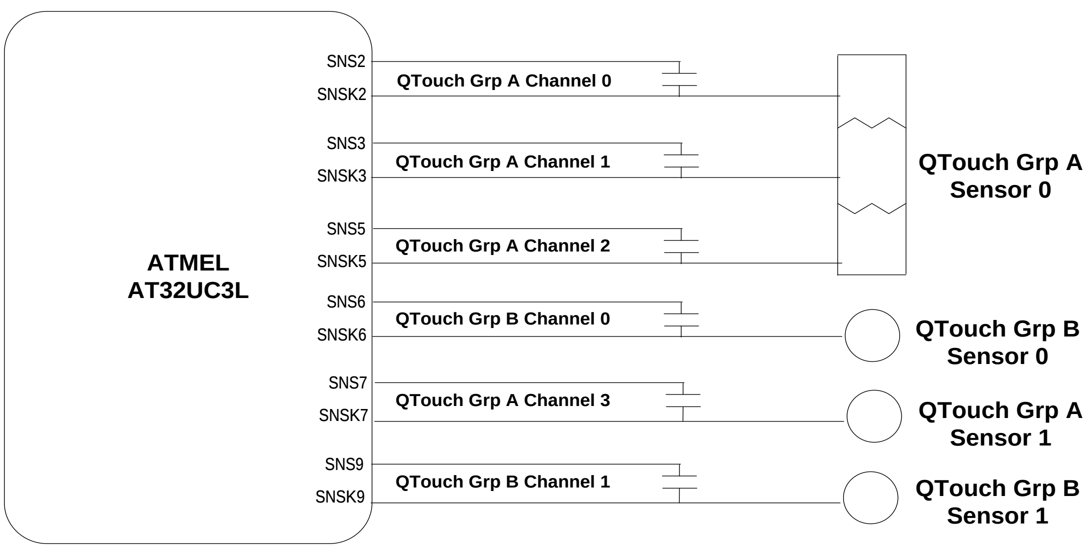


Figure 42 QTouch method Channel/Sensor numbering when Group A and B are used together

When both QTouch Group A and QTouch Group B method are used at the same time, the SNS-SNSK pairs associated with the individual group alone must be taken into consideration when determining the Channel number.

Note: The touch_qt_sensor_config API must follow the above Channel and Sensor numbering when configuring the Sensors.

6.3.8.5 QTouch Group A/B method API Flow for UC3L

For the QTouch operation, the CAT_CLK must be setup appropriately as a first step. Depending on QTouch Group that need to be used, the QTouch Group A, QTouch Group B and Common configuration parameters in the touch_config_at32uc3l.h configuration must then be set up.

The first input argument to the QTouch API, TOUCH_QT_GRP_A or TOUCH_QT_GRP_B indicates if the QTouch API must perform the necessary operation on Group A Sensors or Group B Sensors. The touch_qt_sensors_init API initializes the QTouch Library as well as the CAT module and does the QTouch method specific pin, register and Global Sensor configuration. The touch_qt_sensor_config API is used to configure individual sensor. The Sensor specific configuration parameter can be provided as input to this API.

The touch_qt_sensors_calibrate API is used to calibrate all the configured sensors thereby preparing the sensors for acquisition. The touch_qt_sensors_start_acquisition API initiates a QTouch method measurement on all the configured Sensors (corresponding to the input Touch Group A or B). This API takes the peripheral DMA channels as an input. When a filter_callback function is enabled, the touch_event_dispatcher function calls the filter_callback function as soon as the raw acquisition data from the Sensors is available. The user can now optionally apply any filtering routine on the raw acquisition data before the QTouch Library does any processing on this data. (For an overview of Filter callback usage, refer Section 5.6.6.4 Example code). Once the QTouch Library has finished processing the acquisition data from Sensors, the touch_event_dispatcher function calls the measure_complete_callback function indicating the end of a single Touch measurement operation. The measure_complete_callback provides the measured data and Touch status information. The measured data is available in the same order of Touch Channel numbering. Separate Filter and Measure complete callback functions must be provided for Group A and Group B Sensors.

Note: The Host Application code can execute once a QTouch acquisition is initiated with the touch_qt_sensors_start_acqusition API. Care must be taken in the Host Application such that the touch_event_dispatcher function is called frequently in order to process the acquired data. For a single Touch measurement operation (between a touch_qt_sensors_start_acquisition API call and the measure_complete_callback function being called), the touch_event_dispatcher function may execute multiple times in order to resolve the Touch status of Sensors. Failing to call the touch_event_dispatcher frequently can adversely impact the Touch Sensitivity.

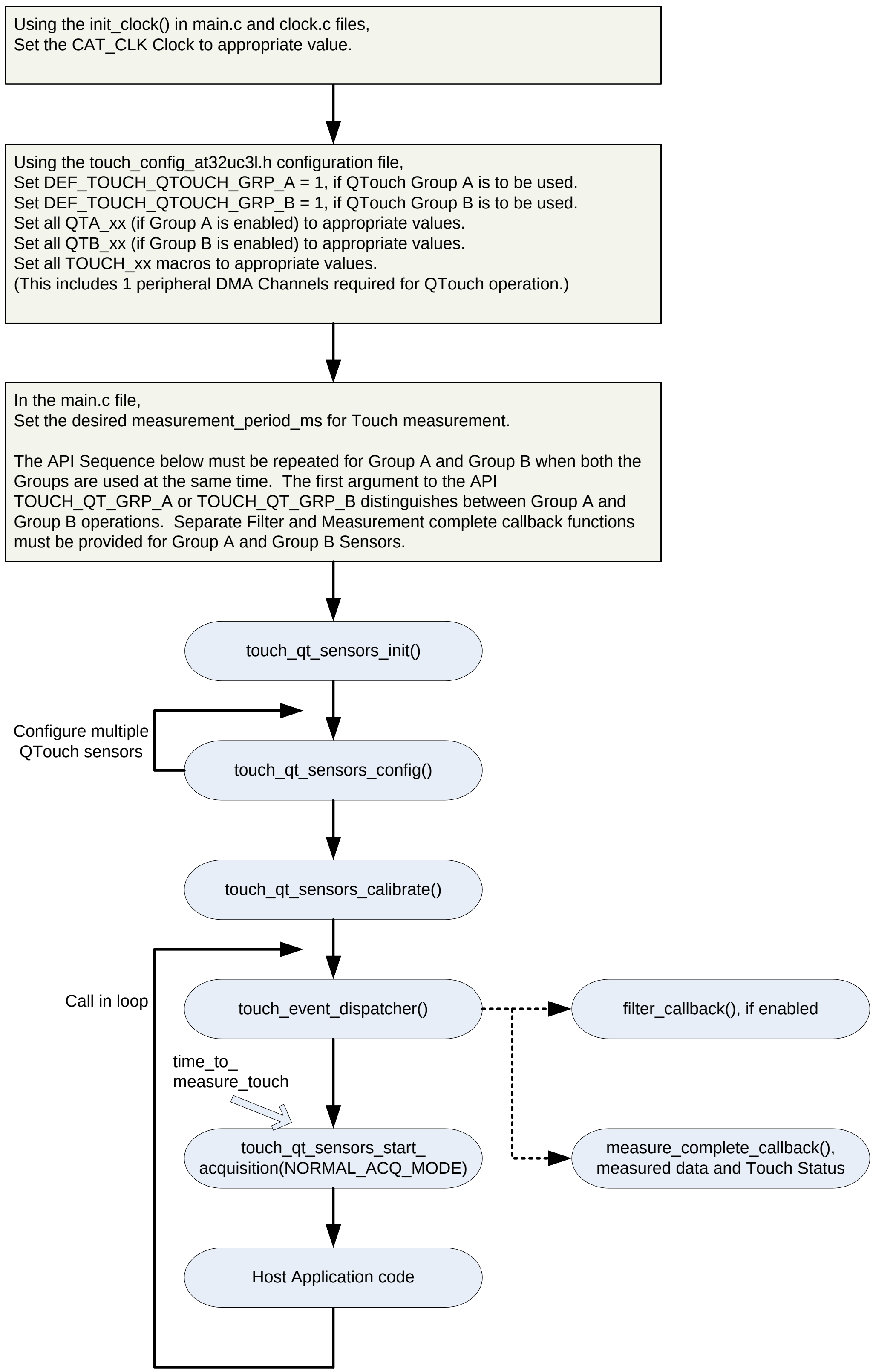


Figure 43 QTouch method API Flow diagram

6.3.8.6 QTouch Group A/B method Disable and Re-enable Sensor for UC3L

The touch_qt_sensor_disable and touch_qt_sensor_reenable API can be used for Disabling and Re-enabling of QTouch Group A and Group B Sensors. In order to Disable or re-enable a sensor, the API must be called with the corresponding sensor_id. Disabling a Sensor disables the measurement process on all the Touch Channels corresponding to Sensor.

When a QTouch Sensor is Disabled or re-enabled, it is mandatory to force Calibration on all Sensors. The Calibration of all Sensors is done using the touch_qt_sensors_calibrate API.

6.3.9 Autonomous QTouch sensor operation for UC3L

6.3.9.1 Autonomous QTouch Sensor pin selection for UC3L

The Autonomous QTouch Sensor pin selection is similar to selection of pin for QTouch Group A/B as indicated in Section 6.3.8.1. Any one SNS-SNSK pair between SNS0-SNSK0 and SNS16-SNSK16 can be chosen to function as an Autonomous QTouch sensor.

6.3.9.2 Autonomous QTouch sensor Schematic for UC3L

The Autonomous QTouch Sensor Sensor schematic is similar QTouch schematic as indicated in Section 6.3.8.2.

6.3.9.3 Autonomous QTouch method hardware resource requirement for UC3L

The clock for the CAT module, CLK_CAT, is generated by the Power Manager (PM). This clock is turned on by default, and can be enabled and disabled in the PM. The user must ensure that CLK_CAT is enabled before initializing the QTouch Library for Autonomous QTouch.

For the Autonomous QTouch Sensor, the complete detection algorithm is implemented within the CAT module. This allows detection of proximity or touch without CPU intervention. Since the Autonomous QTouch Sensor operates without software interaction, this Sensor can be used to wakeup from sleep modes when activated. The Autonomous QTouch Status change interrupt can be used to wakeup from any of the Sleep modes shown in the Table. The ‘Static’ Sleep mode being the deepest possible Sleep mode from which a wake up from Sleep is possible using the Autonomous QTouch. Both an IN_TOUCH status change and OUT_OF_TOUCH status change indication is available when using Autonomous QTouch.

The Autonomous QTouch method acquisition using the CAT module does not require any Peripheral DMA channel for operation.

Sleep Mode	CPU	HSB	PBA,B GCLK	Clock sources	Osc32	RCSYS	BOD & Bandgap	Voltage Regulator
Idle	Stop	Run	Run	Run	Run	Run	On	Full power
Frozen	Stop	Stop	Run	Run	Run	Run	On	Full power
Standby	Stop	Stop	Stop	Run	Run	Run	On	Full power
Stop	Stop	Stop	Stop	Stop	Run	Run	On	Low power
DeepStop	Stop	Stop	Stop	Stop	Run	Run	Off	Low power
Static	Stop	Stop	Stop	Stop	Run	Stop	Off	Low power

Table 12 Sleep mode support for Autonomous QTouch

6.3.9.4 Autonomous QTouch Sensor API Flow for UC3L

For the Autonomous QTouch operation, the CAT_CLK must be setup appropriately as a first step. The Autonomous QTouch and Common configuration parameters in the touch_config_at32uc3l.h configuration must then be set up.

The touch_at_sensors_init API initializes the QTouch Library as well as the CAT module for the Autonomous QTouch sensor related pin, register and Global Sensor configuration. The Autonomous QTouch Sensor can be enabled at any time by the Host Application. Once the Autonomous QTouch Sensor is enabled, the CAT module performs measurements on this sensor continuously to detect a Touch Status. When an IN_TOUCH or OUT_OF_TOUCH status is detected, the QTouch Library calls the touch_at_status_change_interrupt_callback function to indicate the status to the Host application. It is possible to enable and disable Autonomous QTouch sensor multiple times in the Host application by using the touch_at_sensor_enable and touch_at_sensor_disable API.

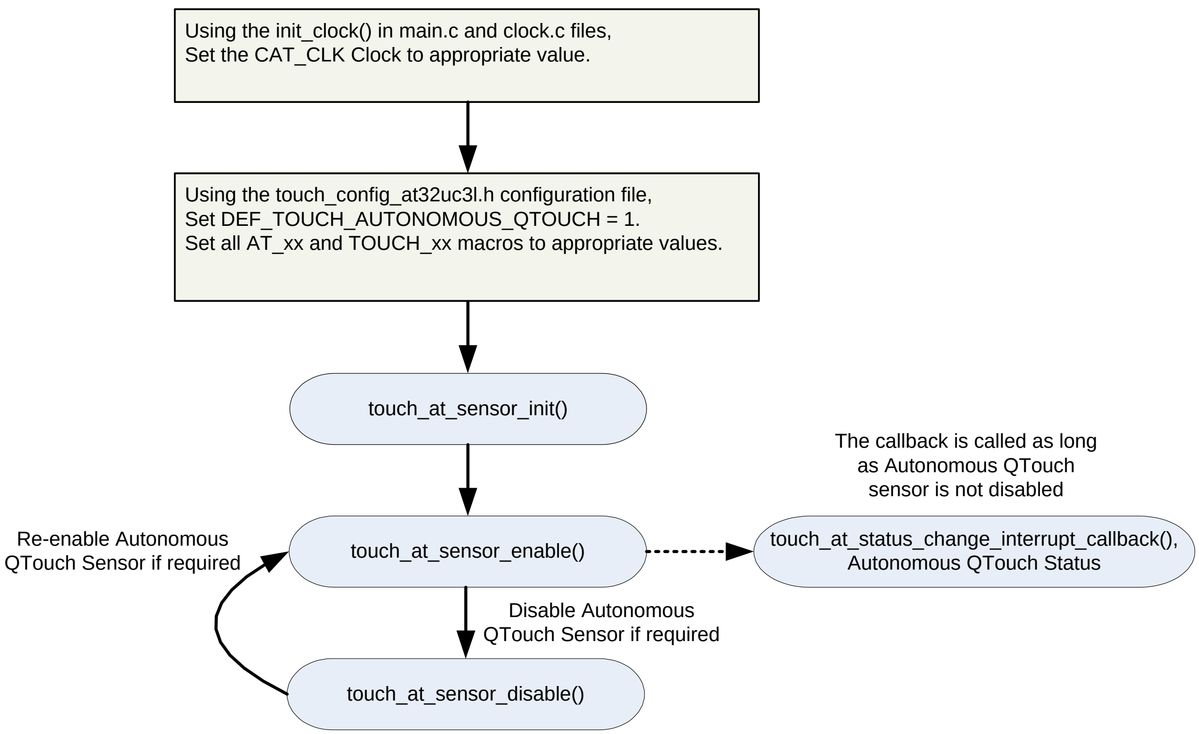


Figure 44 Autonomous QTouch API Flow diagram

6.3.9.5 Autonomous QTouch method Enable and Disable Sensor for UC3L

The touch_at_sensor_enable and touch_at_sensor_disable API can be used for Enabling and Disabling and the Autonomous QTouch Sensor. Once the Autonomous QTouch sensor is enabled, the CAT module performs continuous Touch Measurements on the Sensor in order to detect the Touch Status.

6.3.10 Raw acquisition mode support for UC3L

The QTouch Library Raw acquisition mode can be used with QMatrix, QTouch Group A and QTouch Group B methods. When raw data acquisition mode is used, once the raw acquisition data is available from the CAT module for all the sensors, the measure_complete_callback function is immediately called with acquisition data (channel_signals). The channel_references, sensor_states and rotor_slider_values data are not updated by the Touch Library in this mode.

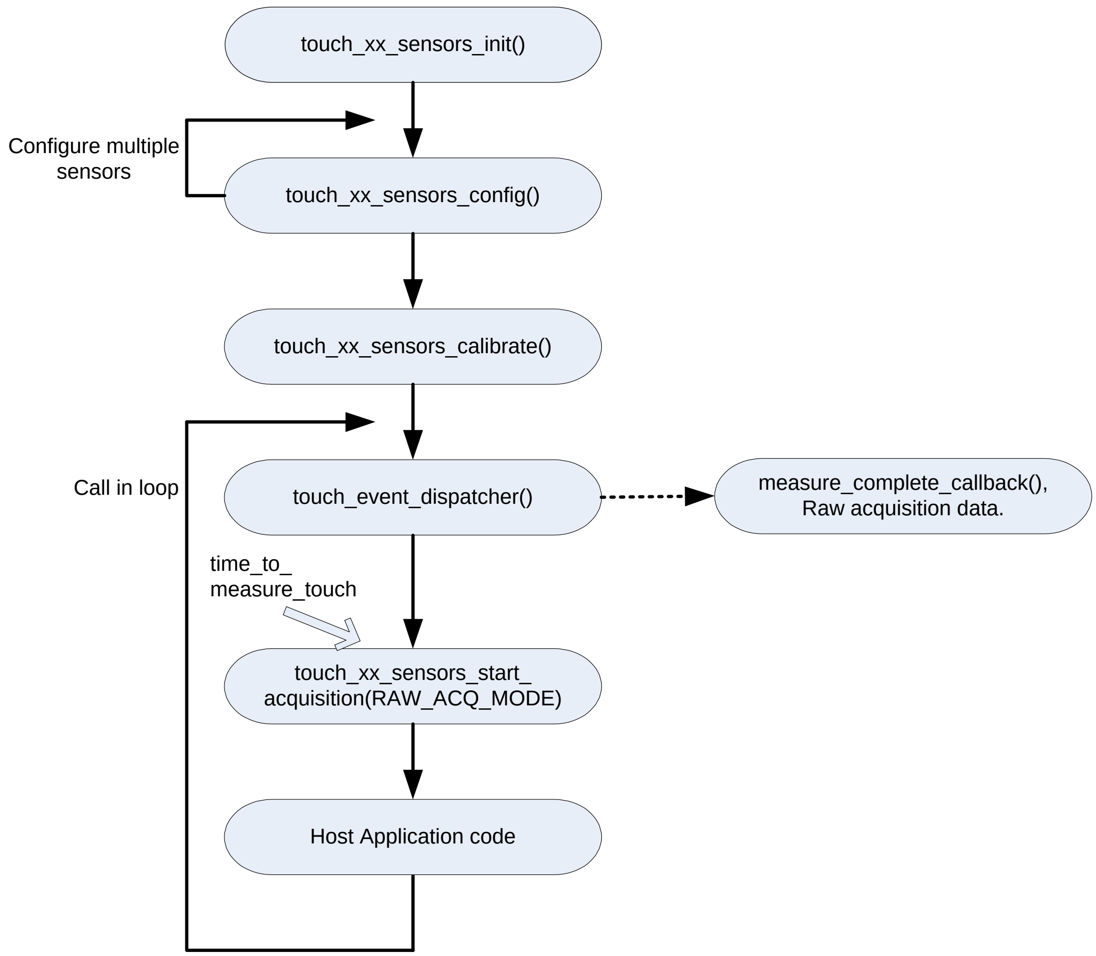


Figure 45 Raw acquisition mode API Flow diagram

6.3.11 Library Configuration parameters for UC3L

The QTouch Library for UC3L provides a single configuration header file touch_config_at32uc3l.h file for setting the various configuration parameters for each method. The different configuration parameters corresponding to QMatrix, QTouch Group A/B and Autononmous QTouch sensors are listed in the Table below.

Parameter	QMatrix	QTouch Group A/B	Autonomous QTouch
Sensor Configuration	QM_NUM_X_LINES QM_NUM_Y_LINES QM_NUM_SENSORS QM_NUM_ROTORS_SLIDERS	QTx_NUM_SENSORS QTx_NUM_ROTORS_SLIDERS	None
Pin Configuration	QM_X_PINS_SELECTED QM_Y_PAIRS_SELECTED QM_SMP_DIS_PIN_OPTION QM_VDIV_PIN_OPTION	QTx_SP_SELECTED	AT_SP_SELECTED
Clock and Register Configuration	QM_GCLK_CAT_DIV QM_CAT_CLK_DIV QM_CHLEN QM_SELEN QM_CXDILEN QM_DILEN QM_DISHIFT QM_MAX_ACQ_COUNT QM_CONSEN QM_INTREFSEL QM_INTVREFSEL QM_ENABLE_SPREAD_SPECTRUM QM_ENABLE_EXTERNAL_SYNC QM_SYNC_TIM	QTx_CAT_CLK_DIV QTx_CHLEN QTx_SELEN QTx_DILEN QTx_DISHIFT QTx_MAX_ACQ_COUNT QTx_ENABLE_SPREAD_SPECTRUM QTx_ENABLE_EXTERNAL_SYNC	AT_CAT_CLK_DIV AT_CHLEN AT_SELEN AT_DILEN AT_DISHIFT AT_MAX_ACQ_COUNT AT_ENABLE_SPREAD_ SPECTRUM AT_ENABLE_ EXTERNAL_SYNC AT_FILTER AT_OUTSENS AT_SENSE AT_PTHR AT_PDRIFT AT_NDRIFT
Peripheral DMA Channel Configuration	QM_DMA_CHANNEL_0 QM_DMA_CHANNEL_1	QTx_DMA_CHANNEL_0	None
Global acquisition parameter Configuration	QM_DI QM_NEG_DRIFT_RATE QM_POS_DRIFT_RATE QM_MAX_ON_DURATION QM_DRIFT_HOLD_TIME QM_POS_ RECAL_DELAY QM_RECAL_THRESHOLD	QTx_DI QTx_NEG_DRIFT_RATE QTx_POS_DRIFT_RATE QTx_MAX_ON_DURATION QTx_DRIFT_HOLD_TIME QTx_ POS_RECAL_DELAY QTx_RECAL_THRESHOLD	None
Callback Function Configuration	QM_FILTER_CALLBACK	QTx_FILTER_CALLBACK	None
Common Configuration Options	TOUCH_SYNC_PIN_OPTION, TOUCH_SPREAD_SPECTRUM_MAX_DEV, TOUCH_CSARES, TOUCH_CSBRES		

Table 13 QTouch Library for UC3L Configuration parameters

For an overview of the Global acquisition configuration parameters and Sensor specific parameters, refer Section 5.3 and Section 5.4. The detailed information on other parameters is available in the configuration header file. For QMatrix method Design guidelines regarding Sensor parameters refer Section 5.6.7.3.

6.3.12 Example projects for QTouch Library for UC3L

6.3.12.1 Example Project usage

The GNU Example projects can be used along with Atmel Studio 6.

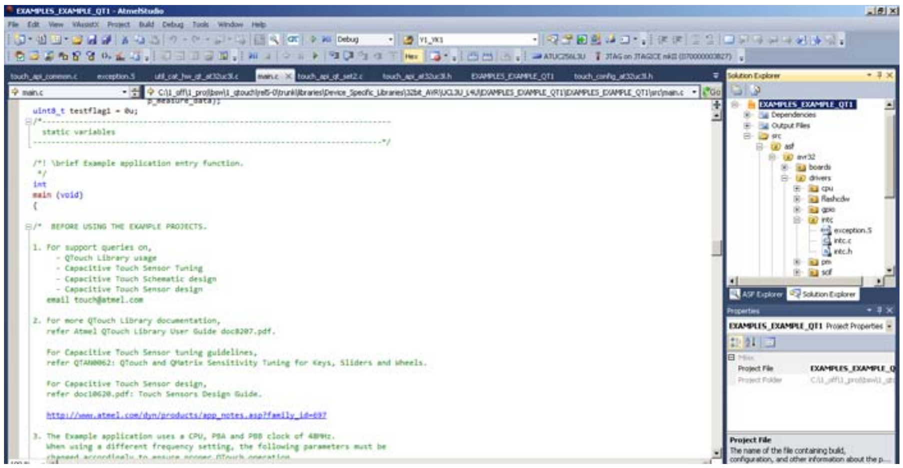


Figure 46 GNU Example project usage with AVR32 Studio

The IAR Example Projects can be used with IAR Embedded Workbench for AVR32 v4.1

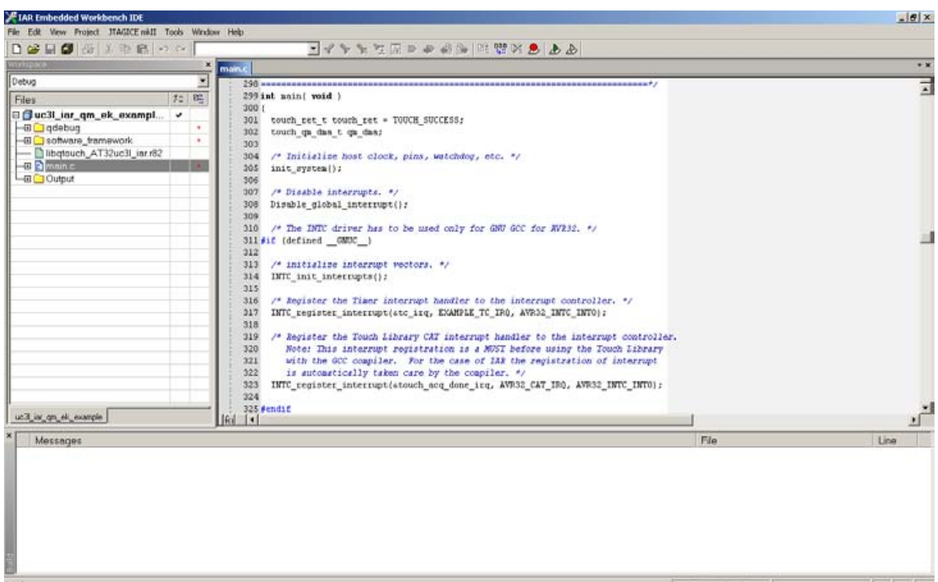


Figure 47 IAR Example project usage with IAR Embedded Workbench for AVR32

6.3.12.2 QMatrix Example Project

The QMatrix method GNU and IAR Example projects can be found in the following path.

\Device_Specific_Libraries\32bit_AVR\UCxx\example_projects\uc3l_gnu_qm_ek_example and

\Device_Specific_Libraries\32bit_AVR\UCxx\example_projects\uc3l_iar_qm_ek_example

The QMatrix Example projects demonstrate the QMatrix operation on the UC3L Evaluation Kit (Rev 2). QMatrix SMP discharge mode hardware arrangement is used for the UC3L Evaluation Kit with 6 X Lines and 2 Y Lines. Using the 12 Touch Channels (6x2), 6 Touch Sensors are formed that include a Rotor (that uses six Touch Channels) and 5 keys (each using one Touch channel).

The Example projects demonstrate the QMatrix measured data and Touch Status usage using the LED Demo application. The onboard LED0, LED1, LED2 and LED3 are set when the Touch Position of the Rotor position varies from 0 to 255. By Touching the up key (^), left key (<), play/pause key (>/||) and right key (>), the LED0, LED1, LED2 and LED3 can be individually cleared. When the down key (v) is touched, it clears all LEDs.

Additionally QMatrix Example projects are also available for QT600 and STK600 boards.

6.3.12.3 QTouch Group A Example Project

The QTouch Group A method GNU and IAR Example projects can be found in the following path.

\Device_Specific_Libraries\32bit_AVR\UCxx\example_projects\uc3l_gnu_qt_grp_a_example and

\Device_Specific_Libraries\32bit_AVR\UCxx\example_projects\uc3l_iar_qt_grp_a_example

The QTouch Group A Example projects demonstrate the QTouch method API usage with a Rotor, Slider and two keys Sensor configuration.

6.3.12.4 Autonomous QTouch Example Project

The Autonomous QTouch Sensor GNU and IAR Example projects can be found in the following path.

\Device_Specific_Libraries\32bit_AVR\UCxx\example_projects\uc3l_gnu_autonomous_qt_example and

\Device_Specific_Libraries\32bit_AVR\UCxx\example_projects\uc3l_iar_autonomous_qt_example

The Autonomous QTouch Example projects demonstrate the Autonomous QTouch Sensor API usage. The Example projects also demonstrate wake up from Sleep mode using the Asynchronous Timer peripheral event.

Note 1: The Example Projects also support relaying the Touch Sensor debug information to the "QTouch Studio – Touch Analyzer" PC Software. The QTouch Studio can also be used for setting the Sensor and Global configuration parameters of the QTouch Library at run-time.

The QTouch Studio can be downloaded from the following path.

http://www.atmel.com/products/touchsoftware/qtouchsuite.asp?family_id=702

The QDebug two-way debug protocol used by the Example project to communicate (transmit or receive touch debug data) with the QTouch Studio can be found in the following installation path.

\Device_Specific_Libraries\32bit_AVR\UCxx\qdebug

- For the UC3L Evaluation kit (*uc3l_xx_qm_ek_example Example project*) to connect with the QTouch Studio using the USB interface, the UC3B MCU on the UC3L Evaluation kit must be Flashed with ISP and Program binaries. The procedure to flash the binaries is available in the readme note in the following path.

  \Device_Specific_Libraries\32bit_AVR\UC3L\example_projects\uc3l_gnu_qm_ek_example\ uc3b\readme.txt or

  \Device_Specific_Libraries\32bit_AVR\UC3L\example_projects\uc3l_iar_qm_ek_example\ uc3b\readme.txt

- For the case of QTouch Group A and Autonomous QTouch Example projects, the 'QT600-USB Bridge' board can be use to capture the QDebug debug data in the QTouch Studio.

Note 2: In order to flash the generated elf binary file for GNU and IAR, the following command can be used from the Command Line.

avr32program --part UC3L064 program -finternal@0x80000000 -e --run -R -cint uc3l_gnu_qm_ek_example.elf

6.3.13 Code and Data Memory requirements for UC3L

6.3.13.1 QMatrix method memory requirement

The Table below captures the Typical Code & Data Memory requirement for the QTouch Library when QMatrix method is used standalone.

In addition to the Data memory captured in the Table, the QMatrix method requires additional Data Memory that must be provided to the Touch Library for storing the Signals, References, Sensor information and Touch status. This data memory is provided by the Host Application to the QTouch Library as QMatrix data block. The size of this Data memory block depends on the Number of Sensors and the Number of Wheel or Slider configured. The PRIV_QM_DATA_BLK_SIZE macro in touch_api_at32uc3l.h calculates the size of this data memory block. For example, for the UC3L Evaluation kit Rev2 that has 6 Sensors including 1 Wheel and 5 Buttons, the QMatrix data block memory size is 236 bytes.

Library	Typical Code with Keys Only	Typical Code when one or more Wheel/Sliders is used	Typical Data Memory
libuc3l-qtouch-iar.r82	5882	7296	278
libuc3l-qtouch-gnu.a	6228	8080	278

Table 14 Typical Code and Data memory for Standalone QMatrix operation

Note: *This Typical Code memory usage is achieved when only QMatrix Regular API is used in the application. Usage of QMatrix Helper API would consume additional Code memory. Also, the Code and Data memory indicated in the Table do not account for Example QMatrix application.*

6.3.13.2 QTouch Group A/B method memory requirement

The Table below captures the Typical Code & Data Memory requirement for the QTouch Library when QTouch Group A or QTouch Group B Sensor is used standalone. (Additional Data memory will be required when both Group A and Group B are used at the same time.)

In addition to the Data memory captured in the Table, the QTouch Group A/B method requires additional Data Memory that must be provided to the Touch Library for storing the Signals, References, Sensor information and Touch status. This data memory is provided by the Host Application to the QTouch Library as QTouch data block. The size of this Data memory block depends on the Number of Sensors and the Number of Wheel or Slider configured. Refer PRIV_QTx_DATA_BLK_SIZE macro in touch_api_at32uc3l.h. For example, when 6 Sensors are used that include 1 Wheel, 1 Slider and 2 Button, the QTouch GroupA/B data block memory size is 184 bytes.

Library	Typical Code with Keys Only	Typical Code when one or more Wheel/Sliders is used	Typical Data Memory
libuc3l-qtouch-iar.r82	5198	6450	358
libuc3l-qtouch-gnu.a	5290	6774	358

Table 15 Typical Code and Data memory for Standalone QTouch Group A/B operation

Note: *This Typical Code memory usage is achieved when only the QTouch Group A/B Regular API is used in the application. Usage of QTouch Group A/B Helper API would consume additional Code memory. Also, the Code and Data memory indicated in the Table do not account for Example QTouch application.*

6.3.13.3 Autonomous QTouch memory requirement

The Table below captures the Typical Code & Data Memory requirement for the QTouch Library when Autonomous Touch Sensor is used standalone.

Library	Typical Code with Keys Only	Typical Data Memory
libuc3l-qtouch-iar.r82	1184	22
libuc3l-qtouch-gnu.a	966	16

Table 16 Minimum Code and Data for Standalone Autonomous QTouch sensor

Note: *This Typical Code memory usage is achieved when only the Autonomous QTouch Regular API is used in the application. Usage of Autonomous QTouch Helper API would consume additional Code memory. Also, the Code and Data memory indicated in the Table do not account for Example Autonomous QTouch application.*

6.3.14 Public header files of QTouch Library for UC3L

Following are the public header files which need to be included in user's application and these have the type definitions and function prototypes of the APIs listed in the following sections

1. touch_api_at32uc3l.h - QTouch Library API and Data structures file.
2. touch_config_at32uc3l.h - QTouch Library configuration file.

6.3.15 Type Definitions and enumerations used in the library

6.3.15.1 Typedefs

This section lists the type definitions used in the library.

Typedef	Notes
uint8_t	unsigned 8-bit integer.
int8_t	signed 8 bit integer.
uint16_t	unsigned 16-bit integer.
int16_t	signed 16-bit integer.

uint32_t	unsigned 32 bit integer.
int32_t	signed 32 bit integer.
channel_t	unsigned 8 bit integer that represents the channel number, starts from 0.
threshold_t	unsigned 8 bit integer to set sensor detection threshold.
sensor_id_t	unsigned 8 bit integer that represents the sensor ID, starts from 0.
touch_time_t	unsigned 16 bit integer that represents current time maintained by the library.
touch_bl_t	unsigned 8 bit integer that represents the burst length of a QMatrix channel.
touch_delta_t	signed 16 bit integer that represents the delta value of a channel.
touch_acq_status_t	unsigned 16 bit Status of Touch measurement.
touch_qt_grp_t	unsigned 8 bit QTouch Group type.
touch_qt_dma_t	unsigned 8 bit QTouch Group A/ Group B DMA channel type..

6.3.15.1.1 touch_acq_status_t

uint16_t touch_acq_status_t
Use Indicates the result of the last acquisition & processing for a specific touch acquisition method.

Values	Bitmask	Comment
TOUCH_NO_ACTIVITY	0x0000u	No Touch activity.
TOUCH_IN_DETECT	0x0001u	At least one Touch channel is in detect.
TOUCH_STATUS_CHANGE	0x0002u	Status change in at least one channel.
TOUCH_ROTOR_SLIDER_POS_CHANGE	0x0004u	At least one rotor or slider has changed position.
TOUCH_CHANNEL_REF_CHANGE	0x0008u	Reference values of at least one of the channel has changed.
TOUCH_BURST_AGAIN	0x0100u	Indicates that reburst is required to resolve Filtering or Calibration state.
TOUCH_RESOLVE_CAL	0x0200u	Indicates that reburst is needed to resolve Calibration.
TOUCH_RESOLVE_FILTERIN	0x0400u	Indicates that reburst is needed to resolve Filtering.
TOUCH_RESOLVE_DI	0x0800u	Indicates that reburst is needed to resolve Detect Integration.
TOUCH_RESOLVE_POS_RECAL	0x1000u	Indicates that reburst is needed to resolve Recalibration.

6.3.15.1.2 touch_qt_grp_t

uint8_t touch_qt_grp_t
Use QTouch Group type.

Values	Value	Comment
TOUCH_QT_GRP_A	0u	QTouch Group A.
TOUCH_QT_GRP_B	1u	QTouch Group B.

6.3.15.2 Enumerations

This section lists the enumerations used in the QTouch Library.

6.3.15.2.1 touch_ret_t

Enumeration touch_ret_t
Use Indicates the Touch Library error code.

Values	Comment
TOUCH_SUCCESS	Successful completion of operation.
TOUCH_ACQ_INCOMPLETE	Touch 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_QM_CONFIG_PARAM	Invalid QMatrix config input parameter.
TOUCH_INVALID_AT_CONFIG_PARAM	Invalid Autonomous Touch config input parameter.
TOUCH_INVALID_QT_CONFIG_PARAM	Invalid QTouch config input parameter.
TOUCH_INVALID_GENERAL_CONFIG_PARAM	Invalid General config input parameter.
TOUCH_INVALID_QM_NUM_X_LINES	Mismatch between number of X lines specified as QM_NUM_X_LINES and number of X lines enabled in QMatrix pin configuration touch_qm_pin_t x_lines.
TOUCH_INVALID_QM_NUM_Y_LINES	Mismatch between number of Y lines specified as QM_NUM_Y_LINES and number of Y lines enabled in QMatrix pin configuration touch_qm_pin_t y_yk_lines.
TOUCH_INVALID_QM_NUM_SENSORS	Number of Sensors specified is greater than (Number of X Lines * Number of Y Lines).
TOUCH_INVALID_MAXDEV_VALUE	Spread spectrum MAXDEV value should not exceed (2*DIV + 1).
TOUCH_INVALID_RECAL_THRESHOLD	Invalid Recalibration threshold input value.
TOUCH_INVALID_CHANNEL_NUM	Channel number parameter exceeded total number of channels configured.
TOUCH_INVALID_SENSOR_TYPE	Invalid sensor type. Sensor type can NOT be SENSOR_TYPE_UNASSIGNED.
TOUCH_INVALID_SENSOR_ID	Invalid Sensor number parameter.
TOUCH_INVALID_DMA_PARAM	DMA Channel numbers are out of range.
TOUCH_FAILURE_ANALOG_COMP	Analog comparator configuration error.
TOUCH_INVALID_RS_NUM	Number of Rotor/Sliders set as 0, when trying to configure a rotor/slider.

6.3.15.2.2 touch_lib_state_t

Enumeration touch_lib_state_t
Use Indicates the current state of the library with respect to a specific acquisition method

Values	Comment
TOUCH_LIB_STATE_NULL	Library is not yet initialized for the specific acquisition method
TOUCH_LIB_STATE_INIT	Library is initialized, sensor configuration and calibration is not yet done.
TOUCH_LIB_STATE_READY	Library is ready for a new acquisition in the specific method
TOUCH_LIB_STATE_CALIBRATE	Library requires re-calibration before acquisition can be

	done for the specific acquisition method
TOUCH_LIB_STATE_BUSY	Library is busy with acquisition & processing for the specific acquisition method

6.3.15.2.3 touch_acq_mode_t

Enumeration touch_acq_mode_t
Use Touch library acquisition mode type.

Values	Comment
RAW_ACQ_MODE	When Raw acquisition mode is used, the measure_complete_callback function is called immediately once fresh values of Signals are available. In this mode, the Touch Library does not do any processing on the Signals. So, the references, Sensor states or Rotor/Slider position values are not updated in this mode.
NORMAL_ACQ_MODE	When Nomal 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.

6.3.15.2.4 sensor_type_t

Enumeration sensor_type_t
Use Define the type of the sensor

Values	Comment
SENSOR_TYPE_UNASSIGNED	Channel is not assigned to any sensor
SENSOR_TYPE_KEY	Sensor is a key
SENSOR_TYPE_ROTOR	Sensor is a rotor
SENSOR_TYPE_SLIDER	Sensor is a slider

6.3.15.2.5 aks_group_t

Enumeration aks_group_t
Use Defines the Adjacent Key Suppression (AKS) groups that each sensor may be associated with

AKS™ is selectable by the system designer
7 AKS groups are supported by the library

Values	Comment
NO_AKS_GROUP	No AKS group is selected for the sensor
AKS_GROUP_1	AKS Group number 1
AKS_GROUP_2	AKS Group number 2
AKS_GROUP_3	AKS Group number 3
AKS_GROUP_4	AKS Group number 4
AKS_GROUP_5	AKS Group number 5
AKS_GROUP_6	AKS Group number 6
AKS_GROUP_7	AKS Group number 7

6.3.15.2.6 hysteresis_t

Enumeration Hysteresis_t
Use Defines the sensor detection hysteresis value. This is expressed as a percentage of the sensor detection threshold.

This is configurable per sensor.

HYST_x = hysteresis value is x percent of detection threshold value (rounded down).
Note that a minimum value of 2 is used as a hard limit. Example: if detection threshold = 20, then:
HYST_50 = 10 (50 percent of 20)
HYST_25 = 5 (25 percent of 20)
HYST_12_5 = 2 (12.5 percent of 20)
HYST_6_25 = 2 (6.25 percent of 20 = 1, but set to the hard limit of 2)

Values	**Comment**
HYST_50	50% Hysteresis
HYST_25	25% Hysteresis
HYST_12_5	12.5% Hysteresis
HYST_6_25	6.25% Hysteresis

6.3.15.2.7 recal_threshold_t

Enumeration recal_threshold_t

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

This is for automatic recovery from false conditions, such as a calibration while sensors were touched, or a significant step change in power supply voltage.
If the false condition persists the library will recalibrate according to the settings of the recalibration threshold.

This setting is applicable to all the configured sensors.

Usage :
RECAL_x = recalibration threshold is x percent of detection threshold value (rounded down).
Note: a minimum value of 4 is used.
Example: if detection threshold = 40, then:
RECAL_100 = 40 (100 percent of 40)
RECAL_50 = 20 (50 percent of 40)
RECAL_25 = 10 (25 percent of 40)
RECAL_12_5 = 5 (12.5 percent of 40)
RECAL_6_25 = 4 (6.25 percent of 40 = 2, but value is limited to 4)

Values	**Comment**
RECAL_100	100% recalibration threshold
RECAL_50	50% recalibration threshold
RECAL_25	25% recalibration threshold
RECAL_12_5	12.5% recalibration threshold
RECAL_6_25	6.25% recalibration threshold

6.3.15.2.8 resolution_t

Enumeration resolution_t

Use 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.

Values	Comment
RES_1_BIT	1 bit resolution : reported positions range 0 – 1
RES_2_BIT	2 bit resolution : reported positions range 0 – 3
RES_3_BIT	3 bit resolution : reported positions range 0 – 7
RES_4_BIT	4 bit resolution : reported positions range 0 – 15
RES_5_BIT	5 bit resolution : reported positions range 0 – 31
RES_6_BIT	6 bit resolution : reported positions range 0 – 63
RES_7_BIT	7 bit resolution : reported positions range 0 – 127
RES_8_BIT	8 bit resolution : reported positions range 0 – 255

6.3.15.2.9 at_status_change_t

Enumeration at_status_change_t
Use Indicates the current status of autonomous QTouch sensor

Values	Comment
OUT_OF_TOUCH	Currently the autonomous QTouch channel is out of touch
IN_TOUCH	Currently the autonomous QTouch channel is in detect

6.3.15.2.10x_pin_options_t

Enumeration x_pin_options_t
Use Options for various pins to be assigned as X lines in QMatrix

Values	Comment
Xn	Use Pin Xn for QMatrix, n ranges from 0 to 17. **Note**: X8 pin must NOT be used as X Line and it is recommended to be used as ACREFN pin for QMatrix.

6.3.15.2.11y_pin_options_t

Enumeration y_pin_options_t
Use Options for various pins to be assigned as Y lines in QMatrix

Values	Comment
Yn_YKn	Use Pin Yn & YKn for QMatrix, n ranges from 0 to 7

6.3.15.2.12qt_pin_options_t

Enumeration qt_pin_options_t
Use Options for various pins to be assigned as Sense pair for Autonomous QTouch, QTouch Group A and QTouch Group B acquisition methods.

Values	Comment
SPn	Use Sense Pair ‘n’ , n ranges from 0 to 16.

6.3.15.2.13general_pin_options_t

Enumeration general_pin_options_t
Use Options of various pins to be used for SMP, Discharge, SYNC & VDIV.

Values	Comment
USE_NO_PIN	No Pin is to be assigned for this purpose

USE_PIN_PA12_AS_SMP	Use Pin PA12 as SMP for QMatrix
USE_PIN_PA13_AS_SMP	Use Pin PA13 as SMP for QMatrix
USE_PIN_PA14_AS_SMP	Use Pin PA14 as SMP for QMatrix
USE_PIN_PA17_AS_SMP	Use Pin PA17 as SMP for QMatrix
USE_PIN_PA21_AS_SMP	Use Pin PA21 as SMP for QMatrix
USE_PIN_PA22_AS_SMP	Use Pin PA22 as SMP for QMatrix
USE_PIN_PA17_AS_DIS	Use Pin PA17 as Discharge current control for QMatrix
USE_PIN_PB11_AS_VDIV	Use Pin PB11 as Voltage divider enable (VDIVEN) for QMatrix
USE_PIN_PA15_AS_SYNC	Use Pin PA15 as external synchronization input signal (SYNC)
USE_PIN_PA18_AS_SYNC	Use Pin PA18 as external synchronization input signal (SYNC)
USE_PIN_PA19_AS_SYNC	Use Pin PA19 as external synchronization input signal (SYNC)
USE_PIN_PB08_AS_SYNC	Use Pin PB08 as external synchronization input signal (SYNC)
USE_PIN_PB12_AS_SYNC	Use Pin PB12 as external synchronization input signal (SYNC)

6.3.16 Data structures

This section lists the data structures that hold sensor status, settings, and diagnostics information.

6.3.16.1 sensor_t

structure sensor_t
Input / Output Output from the library
Use Data structure which holds the sensor state variables used by the library.

<table>
<tr><th>Fields</th><th>Type</th><th colspan="2">Comment</th></tr>
<tr><td>state</td><td>uint8_t</td><td colspan="2">internal sensor state</td></tr>
<tr><td>general_counter</td><td>uint8_t</td><td colspan="2">general purpose counter: used for calibration, drifting, etc</td></tr>
<tr><td>ndil_counter</td><td>uint8_t</td><td colspan="2">drift Integration counter</td></tr>
<tr><td>threshold</td><td>uint8_t</td><td colspan="2">sensor detection threshold</td></tr>
<tr><td rowspan="6">type_aks_pos_hyst</td><td rowspan="6">uint8_t</td><td colspan="2">holds information for sensor type, AKS group, positive recalibration flag, and hysteresis value</td></tr>
<tr><td>Bit fields</td><td>Use</td></tr>
<tr><td>B1 : B0</td><td>Hysteresis</td></tr>
<tr><td>B2</td><td>positive recalibration flag</td></tr>
<tr><td>B5:B3</td><td>AKS group</td></tr>
<tr><td>B7:B6</td><td>sensor type</td></tr>
<tr><td>from_channel</td><td>uint8_t</td><td colspan="2">starting channel number for sensor</td></tr>
<tr><td>to_channel</td><td>uint8_t</td><td colspan="2">ending channel number for sensor</td></tr>
<tr><td>Index</td><td>uint8_t</td><td colspan="2">index for array of rotor/slider values</td></tr>
</table>

6.3.16.2 touch_global_param_t

structure touch_global_param_t
Input / Output Input to the Library
Use Holds the sensor acquisition parameters for a specific acquisition method

Fields	Type	Comment
di	uint8_t	Sensor detect integration (DI) limit.

neg_drift_rate	uint8_t	Sensor negative drift rate in units of 200 ms.
pos_drift_rate	uint8_t	Sensor positive drift rate in units of 200 ms.
max_on_duration	uint8_t	Sensor maximum on duration in units of 200ms.
drift_hold_time	uint8_t	Sensor drift hold time in units of 200 ms.
pos_recal_delay	uint8_t	Sensor Positive recalibration delay.
recal_threshold	recal_threshold_t	Sensor recalibration threshold.

Refer Section 5.3 for Overview of Global configuration parameters.

6.3.16.3 touch_filter_data_t

structure touch_filter_data_t
Input / Output Output from the Library
Use Touch Filter Callback data type.

Fields	Type	Comment
num_channel_signals	uint8_t	Length of the measured signal values list.
p_channel_signals	uint16_t*	Pointer to measured signal values for each channel.

6.3.16.4 touch_measure_data_t

structure touch_measure_data_t
Input / Output Output from the Library
Use This structure provides updated measure data values each time the measure complete callback function is called.

Fields	Type	Comment
p_acq_status	touch_acq_status_t	Acquisition status for the specific acquisition method.
num_channel_signals	uint8_t	Length of the measured signal values list
p_channel_signals	uint16_t*	Pointer to the sequential list of measured signal values of all channels
num_channel_references	uint8_t	Length of the measured reference values list.
p_channel_references	uint16_t*	Pointer to the sequential list of reference values of all channels
num_sensor_states	uint8_t	Number of sensor state bytes.
p_sensor_states	uint8_t*	Pointer to the sequential list of touch status of all sensors
num_rotor_slider_values	uint8_t	Length of the Rotor and Slider position values list.
p_rotor_slider_values	uint8_t*	Pointer to the sequential list of position of all rotors & sliders
num_sensors	uint8_t	Length of the sensors data list.
p_sensor	sensor_t*	Pointer to the sequential list of data of all sensors.

6.3.16.5 touch_qm_param_t

structure touch_qm_param_t
Input / Output Passed as input to touch_qm_sensor_update_config API & got as output from touch_qm_sensor_get_config API
Use Data structure which holds the configuration parameters for a specific QMatrix sensor

Fields	Type	Comment
aks_group	aks_group_t	AKS group to which the sensor belong.
detect_threshold	threshold_t	Detection threshold for the sensor
detect_hysteresis	hysteresis_t	Detect hysteresis for the sensor.
position_resolution	resolution_t	Resolution required for the sensor.
position_hysteresis	uint8_t	Position hysteresis for the sensor

6.3.16.6 touch_at_param_t

structure touch_at_param_t

Input / Output Passed as input to touch_at_sensor_update_config API & got as output from touch_at_sensor_get_config API

Use Data structure which holds the configuration parameters for the autonomous QTouch sensor

Structure field	Type	Corresponds to the device register	Register Field
filter	uint8_t	ATCFG2	FILTER
outsens	uint8_t	ATCFG2	OUTSENS
sense	uint8_t	ATCFG2	SENSE
pthr	uint8_t	ATCFG3	PTHR
pdrift	uint8_t	ATCFG3	PDRIFT
ndrift	uint8_t	ATCFG3	NDRIFT

Refer Section 5.3 for an overview of FILTER (Detect Integration), PTHR (Positive Recalibration threshold), PDRIFT (Positive Drift rate) and NDRIFT (Negative Drift rate).

OUTSENS - Autonomous Touch Out-of-Touch Sensitivity.

For the autonomous QTouch sensor, specifies how sensitive the out-of-touch detector should be. When the sensor is not touched, the Autonomous Touch Current count register is same as the Autonomous Touch Base count register. When the sensor is touched the Autonomous Touch Current count register decreases. When using the Autonomous QTouch in proximity mode, the Autonomous Touch Base count register decreases as we move towards proximity of the sensor. The OUTSENS value can be arrived at by watching the CAT Autonomous Touch Base Count Register(at memory location 0xFFFF686Cu) and Autonomous Touch Current Count Register(at memory location 0xFFFF6870u) during a sensor touch/proximity and not in touch/proximity. A smaller difference between the Autonomous Touch Base count and Autonomous Touch Current count register can be chosen as the OUTSENS value. Range: 0u to 255u.

SENSE - Autonomous Touch Sensitivity.

For the autonomous QTouch sensor, specifies how sensitive the touch detector should be. When the sensor is not touched, the Autonomous Touch Current count register is same as the Autonomous Touch Base count register. When the sensor is touched the Autonomous Touch Current count register decreases. When using the Autonomous QTouch in proximity mode, the Autonomous Touch Base count register decreases as we move towards proximity of the sensor. The SENSE value can be arrived at by watching the CAT Autonomous Touch Base Count Register(at memory location 0xFFFF686Cu) and Autonomous Touch Current Count Register(at memory location 0xFFFF6870u) during a sensor touch/proximity and not in touch/proximity. A larger difference between the Autonomous Touch Base count and Autonomous Touch Current count register can be chosen as the SENSE value. Range: 0u to 255u.

6.3.16.7 touch_qt_param_t

structure touch_qt_param_t
Input / Output Passed as input to touch_qt_sensor_update_config API & got as output from touch_qt_sensor_get_config API
Use Data structure which holds the status parameters for the QTouch Group A or Group B sensor.

Fields	Type	Comment
aks_group	aks_group_t	AKS group to which the sensor belong.
detect_threshold	threshold_t	Detection threshold for the sensor
detect_hysteresis	hysteresis_t	Detect hysteresis for the sensor.
position_resolution	resolution_t	Resolution required for the sensor.

6.3.16.8 touch_at_status

structure touch_at_status
Input / Output Output structure received as part of the Autonomous QTouch Interrupt callback function.
Use Data structure which holds the status parameters for the autonomous QTouch sensor.

Structure field	Type	Comment
status_change	at_status_change_t	Autonomous QTouch Status change.
base_count	uint16_t	The base count currently stored by the autonomous touch sensor. This is useful for autonomous touch debugging purposes.
current_count	uint16_t	The current count acquired by the autonomous touch sensor. This is useful for autonomous touch debugging purposes.

6.3.16.9 touch_qm_dma_t

structure touch_qm_dma_t
Input / Output Input to the touch_qm_sensors_start_acquisition() API.
Use Data structure which holds the DMA channel information for touch acquisition data transfer

Fields	Type	Comment
dma_ch1	uint8_t	Indicates the DMA channel 1. Can take values from 0 – 11, but should not be same as dma_ch2
dma_ch2	uint8_t	Indicates the DMA channel 2. Can take values from 0 – 11, but should not be same as dma_ch1

6.3.16.10 touch_qm_pin_t

structure touch_qm_pin_t
Input / Output Input to the library
Use Data structure which holds the Pin configuration information for QMatrix

<table>
<tr><th>Fields</th><th>Type</th><th>Comment</th></tr>
<tr><td>x_lines</td><td>uint32_t</td><td>Bitmask that indicates the selected X pins for QMatrix. If bit n is set, Xn is enabled for QMatrix; n can be 0 to 17. Any other bits set are ignored. Note: For QMatrix operation, X8 is not available as it must be used for ACREFN function.
<table>
<tr><td>Bit</td><td>18 - 31</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td></tr>
<tr><td>X Line</td><td>-</td><td>X 17</td><td>X 16</td><td>X 15</td><td>X 14</td><td>X 13</td><td>X 12</td><td>X 11</td><td>X 10</td><td>X 9</td><td>X 8</td><td>X 7</td><td>X 6</td><td>X 5</td><td>X 4</td><td>X 3</td><td>X 2</td></tr>
</table></td></tr>
<tr><td>y_yk_lines</td><td>uint8_t</td><td>Bitmask that indicates the selected Y pins for QMatrix. If bit n is set, Yn & Ykn is enabled for QMatrix; n can be 0 to 7.
<table>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Y Line</td><td>Y7 & YK7</td><td>Y6 & YK6</td><td>Y5 & YK5</td><td>Y4 & YK4</td><td>Y3 & YK3</td><td>Y2 & YK2</td><td>Y1 & YK1</td><td>Y0 & YK0</td></tr>
</table></td></tr>
<tr><td>smp_dis_pin</td><td>general_pin_options_t</td><td>Specify one of the following

USE_NO_PIN
USE_PIN_PA12_AS_SMP
USE_PIN_PA13_AS_SMP
USE_PIN_PA14_AS_SMP
USE_PIN_PA17_AS_SMP
USE_PIN_PA21_AS_SMP
USE_PIN_PA22_AS_SMP
USE_PIN_PA17_AS_DIS</td></tr>
<tr><td>vdiv_pin</td><td>general_pin_options_t</td><td>Specify either USE_NO_PIN or USE_PIN_PB11_AS_VDIV</td></tr>
</table>

6.3.16.11 touch_at_pin_t

structure touch_at_pin_t
Input / Output Input to the library
Use Data structure which holds the Pin configuration for Autonomous QTouch sensor

Fields	Type	Comment
atsp	uint8_t	Sense pair to be used for autonomous QTouch detection. Choose any one sense pair from SP0 to SP16 using the qt_pin_options_t enum. For example, if atsp is set as SP7, Sense pair 7 (CSA7, CSB7) will be assigned for autonomous QTouch detection

6.3.16.12 touch_qt_pin_t

structure touch_at_pin_t
Input / Output Input to the library
Use Data structure which holds the Pin configuration for QTouch sensor.

Fields	Type	Comment

<table>
<tr><td>sp</td><td>uint32_t</td><td>Bit n indicates Sense Pair SP[n] is selected. Choose sense pairs from SP0 to SP16.

<table>
<tr><th>Bit</th><td>17-31</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><th>SPn</th><td>-</td><td>X16</td><td>X15</td><td>X14</td><td>X13</td><td>X12</td><td>X11</td><td>X10</td><td>X9</td><td>X8</td><td>X7</td><td>X6</td><td>X5</td><td>X4</td><td>X3</td><td>X2</td><td>X1</td><td>X0</td></tr>
</table>
</td></tr>
</table>

6.3.16.13 touch_qm_reg_t

structure touch_qm_reg_t
Input / Output Input to the library
Use Data structure which holds the Register configuration information for QMatrix

This structure contains the data fields that correspond to specific fields in different registers. For a more detailed explanation of the register fields, refer to the device datasheet.

For example, CHLEN field of MCCFG0 is 8 bits wide (bit 8-15 of MGCFG0 register). The user needs to set values from 0 to 255 (0xFF) in the *chlen* field of this structure. The library will take care of writing this to the appropriate bit position of MCCFG0 register.

Fields	Type	Corresponds to Register	Register Field
div	uint16_t	MGCFG0	DIV
chlen	uint8_t	MGCFG0	CHLEN
selen	uint8_t	MGCFG0	SELEN
dishift	uint8_t	MGCFG1	DISHIFT
sync	uint8_t	MGCFG1	SYNC
spread	uint8_t	MGCFG1	SPREAD
dilen	uint8_t	MGCFG1	DILEN
max	uint16_t	MGCFG1	MAX
acctrl	uint8_t	MGCFG2	ACCTRL
consen	uint8_t	MGCFG2	CONSEN
cxdilen	uint8_t	MGCFG2	CXDILEN
synctim	uint16_t	MGCFG2	SYNCTIM
fsources	uint8_t	DICS	FSOURCES
glen	uint8_t	DICS	GLEN
intvrefsel	uint8_t	DICS	INTVREFSEL
Intrefsel	uint8_t	DICS	INTREFSEL
trim	uint8_t	DICS	TRIM
sources	uint8_t	DICS	SOURCES
shival0	uint16_t	ACSHI0	SHIVAL
shival1	uint16_t	ACSHI1	SHIVAL
shival2	uint16_t	ACSHI2	SHIVAL
shival3	uint16_t	ACSHI3	SHIVAL
shival4	uint16_t	ACSHI4	SHIVAL
shival5	uint16_t	ACSHI5	SHIVAL
shival6	uint16_t	ACSHI6	SHIVAL
shival7	uint16_t	ACSHI7	SHIVAL

6.3.16.14 touch_at_reg_t

structure touch_at_reg_t
Input / Output Input to the library
Use Data structure which holds the Register configuration information for Autonomous QTouch

This structure contains the data fields that correspond to specific fields in different registers. For a more detailed explanation of the register fields, refer to the device datasheet.

For example, DISHIFT field of ATCFG1 is 2 bits wide (bit 28-29 of ATCFG1 register). The user needs to set values from 0 to 3 in the *dishift* field of this structure. The library will take care of writing this to the appropriate bit position of ATCFG1 register.

Fields	Type	Corresponds to Register	Register Field
div	uint16_t	ATCFG0	DIV
chlen	uint8_t	ATCFG0	CHLEN
selen	uint8_t	ATCFG0	SELEN
dishift	uint8_t	ATCFG1	DISHIFT
sync	uint8_t	ATCFG1	SYNC
spread	uint8_t	ATCFG1	SPREAD
dilen	uint8_t	ATCFG1	DILEN
max	uint16_t	ATCFG1	MAX
at_param	touch_at_param_t	Autonomous Touch Sensor parameters corresponding to ATCFG2 and ATCFG3.	FILTER, OUTSENS, SENSE, PTHR, PDRIFT, NDRIFT

6.3.16.15 touch_qt_reg_t

structure touch_qt_reg_t
Input / Output Input to the library
Use Data structure which holds the Register configuration information for QTouch Group A/B.

Fields	Type	Corresponds to Register	Register Field
div	uint16_t	TGxCFG0	DIV
chlen	uint8_t	TGxCFG0	CHLEN
selen	uint8_t	TGxCFG0	SELEN
dishift	uint8_t	TGxCFG1	DISHIFT
sync	uint8_t	TGxCFG1	SYNC
spread	uint8_t	TGxCFG1	SPREAD
dilen	uint8_t	TGxCFG1	DILEN
max	uint16_t	TGxCFG1	MAX

6.3.16.16 touch_qm_config_t

structure touch_qm_config_t
Input / Output Input to the library
Use Data structure which holds all configuration information pertaining to QMatrix

Fields	Type	Comment
num_channels	uint8_t	Indicates the number of QMatrix channels required by the user

num_sensors	uint8_t	Indicates the number of QMatrix sensors required by the user.
num_rotors_and_sliders	uint8_t	Indicates the number of QMatrix rotors / sliders required by the user.
num_x_lines	uint8_t	Number of QMatrix X lines required by the user.
num_y_lines	uint8_t	Number of QMatrix Y lines required by the user.
num_x_sp	uint8_t	Number of X sense pairs used. This is a private variable to the Touch library. The user must provide PRIV_QM_NUM_X_SENSE_PAIRS for this input field.
bl_write_count	uint8_t	Burst length write count. This is a private variable to the Touch library. The user must provide the PRIV_QM_BURST_LENGTH_WRITE_COUNT macro for this input field.
pin	touch_qm_pin_t	Holds the QMatrix Pin configuration information as filled by the user.
reg	touch_qm_reg_t	Holds the QMatrix register configuration information as filled by the user.
global_param	touch_global_param_t	Holds the global parameters for QMatrix as filled by the user.
p_data_blk	uint8_t*	Pointer to the data block allocated by the user
buffer_size	uint16_t	Size of the data block pointed to by p_data_blk. The user must provide the PRIV_QM_DATA_BLK_SIZE macro for this input field.
p_burst_length	uint8_t*	Pointer to an array of 8-bit Burst lengths, where each 8-bit value correspond to the burst length of each channel starting from channel 0 to number of channels.
filter_callback	Pointer to a function	Pointer to callback function that will be called before processing the signals

6.3.16.17 touch_at_config_t

structure touch_at_config_t
Input / Output Input to the library
Use Data structure which holds the configuration parameters & register values for autonomous QTouch acquisition

Fields	Type	Comment
pin	touch_at_pin_t	Holds the autonomous QTouch configuration information as filled by the user.
reg	touch_at_reg_t	Holds the autonomous QTouch register configuration information as filled by the user.
touch_at_status_change_callback	Pointer to a function	Pointer to callback function that will be called by the library whenever there is a touch status change in the autonomous QTouch sensor

6.3.16.18 touch_qt_config_t

structure touch_qm_config_t
Input / Output Input to the library
Use Data structure which holds all configuration information pertaining to QMatrix

Fields	Type	Comment
num_channels	uint8_t	Indicates the number of QTouch Group A/B channels required by the user
num_sensors	uint8_t	Indicates the number of QTouch Group A/B sensors required by the user.
num_rotors_and_sliders	uint8_t	Indicates the number of QTouch Group A/B rotors / sliders required by the user.
pin	touch_qt_pin_t	Holds the QTouch Group A/B Pin configuration information as filled by the user.
reg	touch_qt_reg_t	Holds the QTouch Group A/B register configuration information as filled by the user.
global_param	touch_global_param_t	Holds the global parameters for QTouch Group A/B as filled by the user.
p_data_blk	uint8_t*	Pointer to the data block allocated by the user
buffer_size	uint16_t	Size of the data block pointed to by p_data_blk. The user must provide the PRIV_QTA_DATA_BLK_SIZE or PRIV_QTB_DATA_BLK_SIZE macro for this input field.
filter_callback	Pointer to a function	Pointer to callback function that will be called before processing the signals

6.3.16.19 touch_general_config_t

structure touch_general_config_t
Input / Output Input to the library
Use Data structure which holds the configuration parameters & register values common to all acquisition methods.

Fields	Type	Comment
sync_pin	general_pin_options_t	Specify one of the following values indicating the pin to be assigned as SYNC pin. Refer to the device datasheet for more details. USE_NO_PIN USE_PIN_PA15_AS_SYNC USE_PIN_PA18_AS_SYNC USE_PIN_PA19_AS_SYNC USE_PIN_PB08_AS_SYNC USE_PIN_PB12_AS_SYNC
maxdev	uint8_t	Corresponds to MAXDEV field of SSCFG register that indicates the maximum deviation when spread spectrum is enabled. Ensure that maxdev is always less than or equal to (2*div + 1). div represents div field in touch_qm_reg_t & touch_at_reg_t structures.

csares	uint32_t	Corresponds to RES field of CSARES register.
csbres	uint32_t	Corresponds to RES field of CSBRES register.

6.3.16.20 touch_config_t

structure touch_config_t
Input / Output Input to the library
Use Pointer to this structure is passed as input to touch_qm_sensors_init & touch_at_sensor_init APIs

Fields	Type	Comment
p_qm_config	touch_qm_config_t*	Pointer to the QMatrix configuration structure.
p_at_config	touch_at_config_t*	Pointer to the autonomous QTouch configuration structure.
p_qta_config	touch_qt_config_t*	Pointer to the QTouch Group A configuration structure.
p_qtb_config	touch_qt_config_t*	Pointer to the QTouch Group B configuration structure.
p_general_config	touch_general_config_t*	Pointer to the general configuration structure.

6.3.16.21 touch_info_t

structure touch_info_t
Input / Output Output from the library
Use Pointer to this structure is passed as input to touch_qm_get_libinfo & touch_at_get_libinfo APIs

Fields	Type	Comment
num_channels_in_use	uint8_t	Number of channels in use
num_sensors_in_use	uint8_t	Number of sensors in use
num_rotors_sliders_in_use	uint8_t	Number of rotor/sliders in use
max_channels_per_rotor_slider	uint8_t	Maximum number of channels per rotor/slider allowed by the library
hw_version	uint32_t	CAT module hardware revision as per VERSION register in CAT module.
fw_version	uint16_t	QTouch Library version with MSB indicating the major version & LSB indicating the minor version.

6.3.17 Public Functions of QTouch Library for UC3L

This section lists the public functions available in the QTouch™ libraries for AT32UC3L devices.

6.3.17.1 QMatrix API

This section lists the functions that are specific to QMatrix method of acquisition.

6.3.17.1.1 touch_qm_sensors_init

*touch_ret_t touch_qm_sensors_init (touch_config_t *p_touch_config)*

Arguments	Type	Comment
p_touch_config	touch_config_t*	Pointer to Touch Library input configuration structure. The touch_qm_config_t and touch_general_config_t members of the Structure should be non-NULL.

- This API initializes the Touch library for QMatrix method acquisition. This API has to be called before calling any other QMatrix API.

- Based on the input parameters, the CAT module is initialized with QMatrix method Pin and Register configuration.

- The Analog comparators necessary for QMatrix operation are initialized by this API.
- Both p_qm_config & p_general_config members of the input configuration structure must point to valid configuration data.
- The General configuration data provided by the p_general_config pointer is common to both QMatrix, QTouch Group A, QTouch Group B and Autonomous Touch sensors.

6.3.17.1.2 touch_qm_sensor_config

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

Arguments	**Type**	**Comment**
sensor_type	sensor_type_t	Specifies sensor type – SENSOR_TYPE_KEY or SENSOR_TYPE_ROTOR or SENSOR_TYPE_SLIDER. The SENSOR_TYPE_UNASSIGNED enum is not a valid input to this API.
from_channel	channel_t	Start channel of the Sensor (rotor, slider or key).
to_channel	channel_t	End channel of the Sensor (rotor, slider or key). For a key, the start and end channels must be the same.
aks_group	aks_group_t	AKS group of this sensor.
detect_threshold	threshold_t	Touch Detect threshold level for Sensor.
detect_hysteresis	hysteresis_t	Value for detection hysteresis.
position_resolution	resolution_t	Position resolution when configuring rotor / slider
position_hysteresis	uint8_t	Position hysteresis when configuring rotor / slider
p_sensor_id	sensor_id_t*	The Sensor ID is updated by the Touch Library upon successful sensor configuration. The Sensor ID starts with 0.

- This API configures a single QMatrix Key, Rotor or Slider.

- The user must provide all the sensor specific settings as input to this API.

- Rotor / Slider sensor will occupy contiguous channels from *from_channel* to *to_channel*.

- For QMatrix acquisition method, 3 to 8 Touch channels per rotor / slider are supported. Keys are always formed using 1 Touch channel.

6.3.17.1.3 touch_qm_sensor_update_config

touch_ret_t touch_qm_sensor_update_config(
sensor_id_t sensor_id,
*touch_qm_param_t *p_touch_sensor_param)*

Arguments	Type	Comment
sensor_id	sensor_id_t	Sensor ID for which the configuration needs to be updated.
p_touch_sensor_param	touch_qm_param_t*	Pointer to the user sensor configuration structure.

- This API updates the configuration of a QMatrix sensor with values different from the ones initialized by the touch_qm_sensor_config API. If the sensor was not configured already, the API will return error.

- The user must populate the structure pointed by p_touch_sensor_param with required settings before calling this API.

6.3.17.1.4 touch_qm_sensor_get_config

touch_ret_t touch_qm_sensor_get_config(
sensor_id_t sensor_id,
*touch_qm_param_t *p_touch_sensor_param)*

Arguments	Type	Comment
sensor_id	sensor_id_t	Sensor ID for which the configuration needs to be updated.
p_touch_sensor_param	touch_sensor_param_t*	Pointer to the user sensor configuration structure.

- This API copies the current configuration of a QMatrix sensor into the user configuration structure.

6.3.17.1.5 touch_qm_channel_udpate_burstlen

touch_ret_t touch_qm_channel_udpate _burstlen(
channel_t channel,
touch_bl_t qm_burst_length)

Arguments	Type	Comment
channel	uint8_t	Channel number for which the burst length is to be set.
qm_burst_length	touch_bl_t	QMatrix burst length. The burst length value can be 1 to 255. A value of 1 can be used to disable bursting on a given channel.

- This API updates the burst length of the specified QMatrix channel

- This API can also be used to disable Touch measurement on a Sensor.

- In order to disable a Sensor, the burst length value of all the channels corresponding to the Sensor must be set to 1. A Sensor can then be re-enabled by setting the appropriate burst length for all channels using this API.

  Note: When disabling a sensor care must be taken such that all channels of the Sensor are set to 1. If any of the channels are missed out, it will result in undesired behavior of the Sensor. Similarly when re-enabling a Sensor, if one or more channels are left disabled with a burst length value of 1, it will result in undesired behavior of the Sensor.

- The touch_qm_sensors_calibrate API needs to be called whenever burst length is updated for one or more channels before starting a new Touch measurement using the touch_qm_sensors_start_acquisition API.

6.3.17.1.6 touch_qm_update_global_param

*touch_ret_t touch_qm_update_global_param(touch_global_param_t *p_global_param)*

Arguments	Type	Comment
p_global_param	touch_global_param_t	Pointer to user global parameters structure for QMatrix.

- This API can be used to update the QMatrix global parameters, with values different from the ones initialized using touch_qm_sensors_init API.

6.3.17.1.7 touch_qm_get_global_param

*touch_ret_t touch_qm_get_global_param(touch_global_param_t *p_global_param)*

Arguments	Type	Comment
p_global_param	touch_global_param_t	Pointer to user global parameters structure for QMatrix.

- This API can be called to retrieve the QMatrix global parameters.

6.3.17.1.8 touch_qm_sensors_calibrate

touch_ret_t touch_qm_sensors_calibrate(void)

Arguments	Type	Comment
Void	-	

- This API can be used to calibrate all configured Sensors.
- Calibration of all Sensors must be performed when –
  - All the Sensors have been configured using touch_sensor_config API after initialization of the Touch Library.
  - A sensor or a group of Sensors have been disabled or re-enabled.

6.3.17.1.9 touch_qm_sensors_start_acquisition

touch_ret_t touch_qm_sensors_start_acquisition(
touch_time_t current_time_ms,
*touch_qm_dma_t *p_touch_dma,*
touch_acq_mode_t qm_acq_mode,
*void (*measure_complete_callback)(touch_measure_data_t *p_measure_data))*

Arguments	**Type**	**Comment**
current_time_ms	touch_time_t	Current time in ms
p_touch_dma	touch_qm_dma_t*	DMA channels to be used for transfer to burst length & acquisition count
qm_acq_mode	touch_acq_mode_t	Specify whether Normal acquisition mode or Raw acquisition mode should be done.
void (*measure_complete_callback)(void)	void (*measure_complete_callback)(touch_measure_data_t *p_measure_data)	QMatrix Measure complete callback function pointer

- This API initiates a capacitive measurement on all enabled QMatrix sensors.
- When normal acquisition mode is used, once the Touch measurement is completed on all the QMatrix sensors, before processing the raw acquisition data (channel_signals), a *filter_callback* function is optionally called by the Touch Library.
- Once the *filter_callback* is completed, the signal values will be processed by the Touch Library. The *measure_complete_callback* function is then called with touch data (channel_signals, channel_references, sensor_states, sensors structure) as well as the Touch Status (sensor_states) and Rotor/Slider position (rotor_slider_values).
- The touch_event_dispatcher API needs to be called as frequently as possible for the Touch Library to process the raw acquisition data.
- When raw data acquisition mode is used, once the raw acquisition data is available from the CAT module for all the sensors, the measure_complete_callback function is immediately called with acquisition data (channel_signals). The channel_references, sensor_states and rotor_slider_values data are not updated by the Touch Library in this mode.
- This API will return error if a Touch measurement is already in progress.
- Two peripheral DMA channels must be provided using p_touch_dma for QMatrix operation.

6.3.17.1.10 touch_qm_get_libinfo

*touch_ret_t touch_qm_get_libinfo(touch_info_t *p_touch_info)*

Arguments	**Type**	**Comment**
p_ touch_info	touch_info_t*	User passes the memory address at which the library information is to be stored by the library.

- The touch_info_t structure is filled by the library with information like number of QMatrix channels, number of QMatrix sensors, number of QMatrix rotors/slider, CAT hardware version, and library version.

- The QMatrix number of channels, sensors and rotors/slider indicate the total number of channels, sensors and rotor/slider in use irrespective of Touch measured being disabled or enabled. (Disabling and Re-enabling of a Sensor using the touch_qm_sensor_upate_burstlen API does not alter these values).

6.3.17.1.11 touch_qm_sensor_get_delta

touch_ret_t touch_qm_sensor_get_delta(
sensor_id_t sensor_id,
*touch_delta_t *p_delta)*

Arguments	Type	Comment
sensor_id	sensor_id_t	Sensor ID for which the delta needs to be retrieved.
p_delta	touch_delta_t*	Pointer to Delta variable, that will be update by the Touch Library

- This API retrieves the delta information associated with a specific QMatrix sensor. Delta is the difference between the current signal value and reference value.

- The user must provide the sensor ID whose delta is sought along with a valid pointer to a Delta variable.

- The API updates the delta variable associated with the requested sensor.

6.3.17.2 QTouch Group A and QTouch Group B API

This section lists the functions that are specific to QTouch Group A/B method of acquisition.

6.3.17.2.1 touch_qt_sensors_init

touch_ret_t touch_qt_sensors_init (touch_qt_grp_t touch_qt_grp,
*touch_config_t *p_touch_config)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
p_touch_config	touch_config_t*	Pointer to Touch Library input configuration structure. The p_qta_config/p_qtb_config (based on whether Group A is used or Group B is used) and p_general_config members of the Structure should be non-NULL.

- This API initializes the Touch library for QTouch Group A or QTouch Group B method acquisition. This API has to be called before calling any other QTouch API.

- Based on the input parameters, the CAT module is initialized with QTouch method Pin and Register configuration.

- The p_qta_config/p_qta_config (based on whether Group A is used or Group B is used) and p_general_config members of the input configuration structure must point to valid configuration data.

- The General configuration data provided by the p_general_config pointer is common to both QMatrix, QTouch Group A, QTouch Group B and Autonomous Touch sensors.

6.3.17.2.2 touch_qt_sensor_config

touch_ret_t touch_qt_sensor_config(touch_qt_grp_t touch_qt_grp,
sensor_type_t sensor_type,
channel_t from_channel,
channel_t to_channel,
aks_group_t aks_group,
threshold_t detect_threshold,
hysteresis_t detect_hysteresis,
resolution_t position_resolution
*sensor_id_t *p_sensor_id)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
sensor_type	sensor_type_t	Specifies sensor type – SENSOR_TYPE_KEY or SENSOR_TYPE_ROTOR or SENSOR_TYPE_SLIDER. The SENSOR_TYPE_UNASSIGNED enum is not a valid input to this API.
from_channel	channel_t	Start channel of the Sensor (rotor, slider or key).
to_channel	channel_t	End channel of the Sensor (rotor, slider or key). For a key, the start and end channels must be the same.
aks_group	aks_group_t	AKS group of this sensor.
detect_threshold	threshold_t	Touch Detect threshold level for Sensor.
detect_hysteresis	hysteresis_t	Value for detection hysteresis.
position_resolution	resolution_t	Position resolution when configuring rotor / slider
p_sensor_id	sensor_id_t*	The Sensor ID is updated by the Touch Library upon successful sensor configuration. The Sensor ID starts with 0.

- This API configures a single QTouch Key, Rotor or Slider.
- The user must provide all the sensor specific settings as input to this API.
- Rotor / Slider sensor will occupy contiguous channels from *from_channel* to *to_channel*.
- For QTouch acquisition method, 3 Touch channels per rotor / slider are supported. Keys are always formed using 1 Touch channel.

6.3.17.2.3 touch_qt_sensor_update_config

touch_ret_t touch_qt_sensor_update_config(touch_qt_grp_t touch_qt_grp,
sensor_id_t sensor_id,
*touch_qt_param_t *p_touch_sensor_param)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.

sensor_id	sensor_id_t	Sensor ID for which the configuration needs to be updated.
p_touch_sensor_param	touch_qt_param_t *	Pointer to the user sensor configuration structure.

- This API updates the configuration of a QTouch sensor with values different from the ones initialized by the touch_qt_sensor_config API. If the sensor was not configured already, the API will return error.

- The user must populate the structure pointed by p_touch_sensor_param with required settings before calling this API.

6.3.17.2.4 touch_qt_sensor_get_config

touch_ret_t touch_qt_sensor_get_config(touch_qt_grp_t touch_qt_grp,
sensor_id_t sensor_id,
*touch_qt_param_t *p_touch_sensor_param)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
sensor_id	sensor_id_t	Sensor ID for which the configuration needs to be updated.
p_touch_sensor_param	touch_qt_param_t*	Pointer to the user sensor configuration structure.

- This API copies the current configuration of a QTouch sensor into the user configuration structure.

6.3.17.2.5 touch_qt_update_global_param

touch_ret_t touch_qt_update_global_param(touch_qt_grp_t touch_qt_grp,
*touch_global_param_t *p_global_param)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
p_global_param	touch_global_param_t	Pointer to user global parameters structure for QTouch Group A/B.

- This API can be used to update the QTouch A or QTouch B global parameters, with values different from the ones initialized using touch_qt_sensors_init API.

6.3.17.2.6 touch_qt_get_global_param

touch_ret_t touch_qt_get_global_param(touch_qt_grp_t touch_qt_grp,
*touch_global_param_t *p_global_param)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.

p_global_param	touch_global_param_t	Pointer to user global parameters structure for QTouch Group A/B.

- This API can be called to retrieve the QTouch Group A or Group B global parameters.

6.3.17.2.7 touch_qt_sensors_calibrate

touch_ret_t touch_qt_sensors_calibrate(touch_qt_grp_t touch_qt_grp)

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.

- This API can be used to calibrate all configured Sensors.
- Calibration of all Sensors must be performed when –
  - All the Sensors have been configured using touch_sensor_config API after initialization of the Touch Library.
  - A sensor or a group of Sensors have been disabled or re-enabled.

6.3.17.2.8 touch_qt_sensors_start_acquisition

touch_ret_t touch_qt_sensors_start_acquisition(touch_qt_grp_t touch_qt_grp,
touch_time_t current_time_ms,
*touch_qt_dma_t *p_touch_dma,*
touch_acq_mode_t qt_acq_mode,
*void (*measure_complete_callback)(touch_measure_data_t *p_measure_data))*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
current_time_ms	touch_time_t	Current time in ms
p_touch_dma	touch_qt_dma_t*	DMA channels to be used for transfer to burst length & acquisition count
qt_acq_mode	touch_acq_mode_t	Specify whether Normal acquisition mode or Raw acquisition mode should be done.
void (*measure_complete_callback)(void)	void (*measure_complete_callback)(touch_measure_data_t *p_measure_data)	QTouch Group A or Group B Measure complete callback function pointer

- This API initiates a capacitive measurement on all enabled QTouch Group A or Group B sensors depending on the touch_qt_grp specified.

- When normal acquisition mode is used, once the Touch measurement is completed on all the QTouch sensors, before processing the raw acquisition data (channel_signals), a *filter_callback* function is optionally called by the Touch Library.

- Once the *filter_callback* is completed, the signal values will be processed by the Touch Library. The *measure_complete_callback* function is then called with touch data (channel_signals, channel_references, sensor_states, sensors structure) as well as the Touch Status (sensor_states) and Rotor/Slider position (rotor_slider_values).

- The touch_event_dispatcher API needs to be called as frequently as possible for the Touch Library to process the raw acquisition data.

- When raw data acquisition mode is used, once the raw acquisition data is available from the CAT module for all the sensors, the measure_complete_callback function is immediately called with acquisition data (channel_signals). The channel_references, sensor_states and rotor_slider_values data are not updated by the Touch Library in this mode.

- This API will return error if a Touch measurement is already in progress.

- One peripheral DMA channels must be provided using p_touch_dma for QTouch operation.

6.3.17.2.9 touch_qt _sensor_ disable

touch_ret_t touch_qt_ sensor_disable(touch_qt_grp_t touch_qt_grp,
sensor_id_t sensor_id)

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
sensor_id	sensor_id_t	Sensor ID of the Sensor to be disabled.

- This API can be used to disable Touch measurement on a QTouch Sensor.

- The touch_qt_sensors_calibrate API needs to be called whenever one or more Sensors are disabled before starting a new Touch measurement using the touch_qt_sensors_start_acquisition API.

- Note: Care must be taken such that a valid Sensor ID corresponding to a QTouch Group A sensor or QTouch Group B Sensor is provided.

6.3.17.2.10 touch_qt _sensor_ reenable

touch_ret_t touch_qt _sensor_reenable(touch_qt_grp_t touch_qt_grp,
sensor_id_t sensor_id)

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
sensor_id	sensor_id_t	Sensor ID of the Sensor to be disabled.

- This API can be used to reenable a disabled QTouch Sensor.

- The touch_qt_sensors_calibrate API needs to be called whenever one or more Sensors are reenabled before starting a new Touch measurement using the touch_qt_sensors_start_acquisition API.

- Note: Care must be taken such that a valid Sensor ID corresponding to a QTouch Group A sensor or QTouch Group B Sensor is provided.

6.3.17.2.11 touch_qt_get_libinfo

touch_ret_t touch_qt_get_libinfo(touch_qt_grp_t touch_qt_grp,
*touch_info_t *p_touch_info)*

Arguments	Type	Comment
touch_qt_grp	touch_qt_grp_t	Specify if the operation is to be performed on Group A Sensors or Group B Sensors.
p_ touch_info	touch_info_t*	User passes the memory address at which the library information is to be stored by the library.

- The touch_info_t structure is filled by the library with the Group specific (based on touch_qt_grp input) information like number of QTouch channels, number of QTouch sensors, number of QTouch rotors/slider, CAT hardware version, and library version.

- The QTouch number of channels, sensors and rotors/slider indicate the total number of channels, sensors and rotor/slider in use irrespective of Touch measured being disabled or enabled. (Disabling and Re-enabling of a Sensor using the touch_qt_sensor_disable and touch_qt_sensor_reenable API does not alter these values).

6.3.17.2.12touch_qt_sensor_get_delta

touch_ret_t touch_qt_sensor_get_delta(touch_qt_grp_t touch_qt_grp,
sensor_id_t sensor_id,
*touch_delta_t *p_delta)*

Arguments	Type	Comment
sensor_id	sensor_id_t	Sensor ID for which the delta needs to be retrieved.
p_delta	touch_delta_t*	Pointer to Delta variable, that will be update by the Touch Library

- This API retrieves the delta information associated with a specific QTouch sensor. Delta is the difference between the current signal value and reference value.

- The user must provide the sensor ID whose delta is sought along with a valid pointer to a Delta variable.

- The API updates the delta variable associated with the requested sensor.

6.3.18 Autonomous touch API

This section lists the functions that are specific to Autonomous QTouch sensor.

6.3.18.1.1 touch_at_sensor_init

*touch_ret_t touch_at_sensor_init(touch_config_t *p_touch_config)*

Arguments	Type	Comment
p_touch_config	touch_config_t*	Pointer to Touch Library input configuration structure. The p_at_config and p_general_config members of the input configuration structure must be non-NULL.

- This API initializes the touch library Autonomous touch sensor. This API has to be called before calling any other Autonomous touch API function.
- Based on the input parameters, the CAT module is initialized with Autonomous Sensor Pin and Register configuration.
- The General configuration data provided by the p_general_config pointer is common to both QMatrix, QTouch Group A, QTouch Group B and Autonomous Touch sensors.

6.3.18.1.2 touch_at_sensor_enable

touch_ret_t touch_at_sensor_enable(void)

Arguments	Type	Comment
void (*touch_at_status_change_interrupt_callback) (touch_at_status *p_at_status)	void (*touch_at_status_change_interrupt_callback) (touch_at_status *p_at_status)	Autonomous QTouch Callback function.

- This API enables the autonomous touch sensor and initiates continuous Touch measurement on the Autonomous QTouch sensor.
- When there is a change in the autonomous QTouch sensor status, the callback function as specified in touch_at_status_change_interrupt_callback will be called. The callback function lets the user know whether the autonomous QTouch sensor is currently in touch or out of touch.

  Note that this callback function will be called from an interrupt service routine. Hence it is recommended to have as minimal code as possible in the callback function.
- This API should be called only after touch_at_sensor_init API is called.

6.3.18.1.3 touch_at_sensor_disable

touch_ret_t touch_at_sensor_disable(void)

Arguments	Type	Comment
void	-	

- This API disables the Touch measurement on the Autonomous QTouch sensor. The status change callback function is not called when the Sensor is disabled.

6.3.18.1.4 touch_at_sensor_update_config

*touch_ret_t touch_at_sensor_update_config(touch_at_param_t *p_at_param)*

Arguments	Type	Comment

p_at_param	touch_at_param_t*	Pointer to autonomous QTouch sensor configuration structure.

- This API updates the configuration of autonomous QTouch sensor with a setting that is different from the one configured by calling touch_at_sensor_init API.

- The user must populate the structure pointed by p_at_param with required settings before calling this API.

6.3.18.1.5 touch_at_sensor_get_config

*touch_ret_t touch_at_sensor_get_config(touch_at_param_t *p_at_param)*

Arguments	Type	Comment
p_at_param	touch_at_param_t*	Pointer to autonomous QTouch sensor configuration structure.

- This API retrieves the current configuration of the autonomous QTouch sensor.

6.3.18.1.6 touch_at_get_libinfo

*touch_ret_t touch_at_get_libinfo(touch_info_t *p_touch_info)*

Arguments	Type	Comment
p_touch_info	touch_info_t*	User passes the memory address at which the library information is to be updated.

- The touch_info_t structure is filled by the library with information on the number of autonomous QTouch channels (Fixed value of 1), number of autonomous QTouch sensors (Fixed value of 1), number of autonomous QTouch rotors/slider (Fixed value of 0), CAT hardware version and library version.

6.3.18.2 Common API

This section lists the functions that are common to QMatrix, QTouch Group A/B and Autonomous QTouch acquisition methods.

6.3.18.2.1 touch_event_dispatcher

void touch_event_dispatcher (void)

Arguments	Type	Comment
Void	-	

- This API needs to be called by the user application to allow the library to process the raw acquisition data from the sensors.

- Once touch_qm_sensors_start_acquisition is called, touch_event_dispatcher API needs to be called as frequently as possible by the Host application.

- The *signals_callback* and *measure_complete_callback* functions are called from the *touch_event_dispatcher* API context.

6.3.18.2.2 touch_deinit

void touch_deinit (void)

Arguments	Type	Comment
void	-	

- This API can be used to de-initalize the Touch Library and disable the CAT module.
- Calling this API de-initializes the Touch Library for Sensors corresponding to all methods of acquisition (QMatrix, QTouch Group A, QTouch Group B and Autonomous QTouch).

6.3.19 Integrating QTouch libraries for AT32UC3L in your application

This section illustrates the key steps required in integrating the QTouch™ library in your application.

a. For your design, you would need the following information to select the correct library variant
    - Device to be used for the design – Current library supports AT32UC3L064, AT32UC3L032, AT32UC3L016 device variants.
    - Compiler platform you intend to use to integrate the libraries.

b. Copy the library variant that was selected in step one to your project's working directory or update your project to point to the library selected.

c. Include touch_api_at32uc3l.h & touch_config_at32uc3l.h header files of the QTouch™ library in your application. The header files can be found in the library installation folder.

d. Initialize/create and use the Touch APIs in your application
    - Set the various configuration options using the touch_config_at32uc3l.h file.
    - Initialize and configure the sensors in the Host application.
    - The Host application also has to provide the required timing so as to perform Touch measurement at regular intervals.

e. General application notes
    - Ensure that there are no conflicts between the resources used by the Touch library and the host application
    - Ensure that the stack size is adjusted to factor in the stack depth required for the operation of the touch libraries.

6.3.20 MISRA Compliance Report of QTouch Library for UC3L

This section lists the compliance and deviations for MISRA standards of coding practice for the UC3L QTouch libraries.

6.3.21 What is covered

The MISRA compliance covers the QTouch library for AT32UC3L devices. The Example projects and associated code provided is not guaranteed to be MISRA compliant.

6.3.22 Target Environment

Development Environment	IAR Embedded Workbench for Atmel AVR32
MISRA Checking software	The MISRA C Compliance has been performed for the library using MISRA C 2004 Rules in IAR Workbench for Atmel AVR32
MISRA Rule set applied	MISRAC 2004 Rule Set, All including advisory

6.3.23 Deviations from MISRA C Standards

The QTouch library was subjected to the above mentioned MISRA compliance rules. The following table lists the exceptions in the AT32UC3L QTouch library source code and also provides explanation for these exceptions.

Apart from these, there were many exceptions in the standard header files supplied by the tool chain and those are not captured here.

Rule	Rule Description	Advisory/ Required	Exception noted / How it is addressed
1.1	All code shall conform to ISO 9899 standard C, with no extensions permitted.	Required	This Rule is not supported as the library implementation requires IAR extensions like __interrupt. These intrinsic functions relate to device hardware functionality, and cannot practically be avoided.
5.4	A tag name shall be a unique identifier	Required	This is violated as for the reason that enumerated types are mixed with other types. This is caused by integers being assigned to enumerated types in some places to save code space
6.3	The basic types of char, int, short, long, float, and double should not be used, but specific-length equivalents should be typedef'd for the specific compiler, and these type names used in the code	Advisory	The type bool supported by the compiler violate this rule.
10.3	The value of a complex expression of integer type shall only be cast to a type that is not wider and of the same signedness as the underlying type of the expression	Required	This is required in the code to do align some pointers in the data block memory. Cannot be avoided.
11.3	A cast should not be done between a pointer type and an integral type	Advisory	This is required in the code to do align some pointers in the data block memory. Cannot be avoided.
17.4	Array indexing shall be the only allowed form of pointer arithmetic	Required	Pointer increment has been done in some places for sequential access of signals, references, etc.
19.13	The # and ## preprocessor operators should not be used	Advisory	This is required for implementation of a macro for ease of use & abstraction

6.3.24 Known Issues with QTouch Library for UC3L

- When the IAR Example Project is build, the IAR32 compiler reports the following Warning - Warning[Pe047]: incompatible redefinition of macro "AVR32_PM_PPCR_MASK" (declared at line 607 of "C:\Program Files\IAR Systems\Embedded C:\Program Files\IAR Systems\Embedded Workbench 5.6\avr32\INC\avr32\pm_400.h 467 Workbench 5.6\avr32\INC\avr32/uc3l064.h").

  In order to avoid this, this warning (Pe047) has been disabled using the Diagnostics option in the IAR32 Project.

6.4 QTouch Library for ATtiny20 device

ATMEL QTouch Library for ATtiny20 can be used for embedding capacitive touch buttons functionality into ATtiny20 device application.

This Section describes the QTouch Library Application Programming API and Configuration interface for QTouch method acquisition using the ATtiny20 devices.

6.4.1 Salient Features of QTouch Library for ATtiny20

6.4.1.1 QTouch method sensor

- 1 Physical pin per Touch Button.
- 1 to 5 Touch Buttons can be configured.
- Individual Sensor Threshold, Sensor Hysteresis and Sensor Global acquisition parameters can be configured.
- Adjacent Key Supression (AKS) support.
- QTouch Studio support for Touch data analysis.
- 'C' Programming interface for easy inclusion of User application.

6.4.2 Compiler tool chain support for ATtiny20

The QTouch libraries for ATtiny20 devices are supported for the following compiler tool chains.

Tool	Version
IAR Embedded Workbench for Atmel AVR. IAR Compiler.	5.5

Table 17 Compiler tool chains support for ATtiny20 QTouch Library

6.4.3 Overview of QTouch Library for ATtiny20

For an overview of QTouch method based capacitive touch acquisition, refer Section 5.2.1 QTouch Acquistion method.

The QTouch Library for ATtiny20 device allows for Sensor configuration and Sensor Acquisition parameter setting. Based on the input Sensor configuration, the QTouch Library takes care of the capacitive touch acquisition data capture operations on the external capacitive touch sensors. The captured Touch Data and Touch Button ON/OFF Status information is then available for user application.

The diagram below indicates a Typical Sensor arrangement using the Tiny20 device. The QTouch Library uses the ATtiny20 ADC Module to peform capacitive Touch measurements. The ADC module must be enabled by the Host Application and configured in Free running mode for QTouch Library to function correctly. The PA0 pin must be configured as Output pin and should be in HIGH state before the qt_measure_sensors API is called. Port pins PA1 to PA7 can be used to support upto 5 Touch Buttons. The Touch Buttons must be connected to sequential Port pins. However, it is not necessary to start the first Touch Button on Port pin PA1. For Example, when 3 Touch Buttons are required, they can by connected to pins PA5, PA6 and PA7.

The Sensor numbering is always in the increasing order of Port pin.

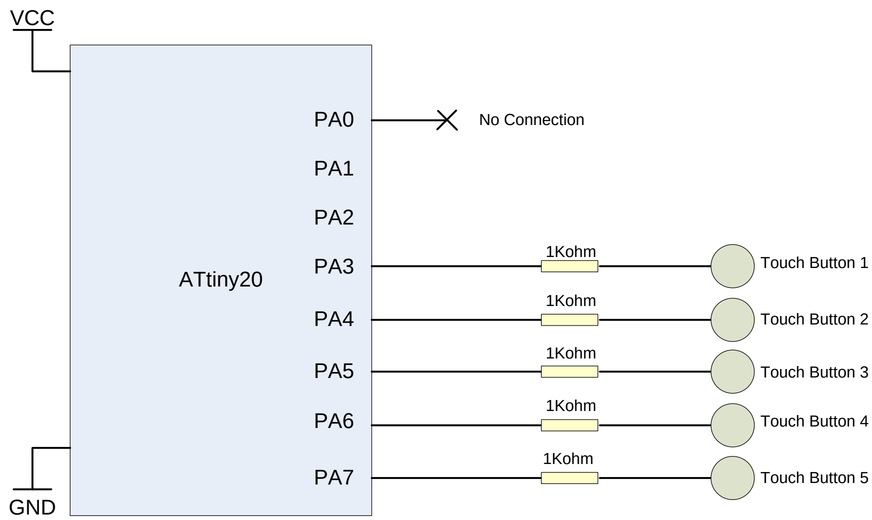


Figure 48 Schematic overview of QTouch on Tiny20

6.4.4 API Flow diagram for ATtiny20

For the QTouch Libraries, the timing information is provided by the Host Application by updating the 'time_current_ms' variable in the Timer ISR. The QTouch Library uses this variable to calculate the necessary timing for Max ON Duration, Drift and Recalibration functionality. Before using the QTouch Libraries, the Timer ISR must be configured appropriately. Also, the Timer Interrupt is used to update the 'time_to_measure_touch' variable inorder to start a capacitive touch measurement. It is recommended to call qt_measure_sensors within 100ms each time to avoid error in QTouch Library timing.

The touch_config_tiny20.h configuration header file can be used to set the desired number of Touch Sensors (Buttons) as well as individual sensor Threshold, Hysteresis and Recalibration parameters. The Sensor Global Configuration parameters must be specified using the IAR Linker define options.

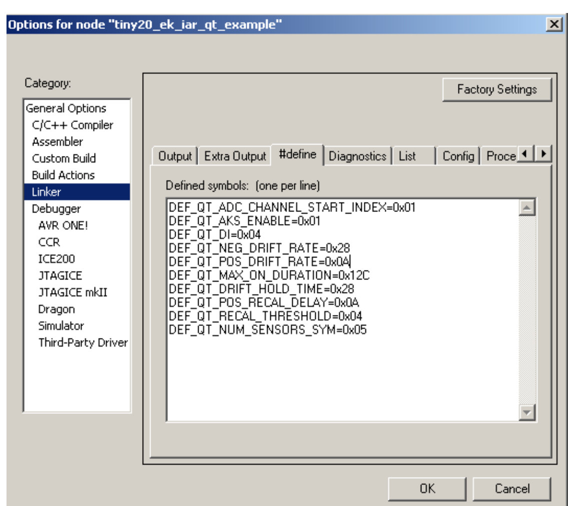


Figure 49 Linker configuration options for Tiny20

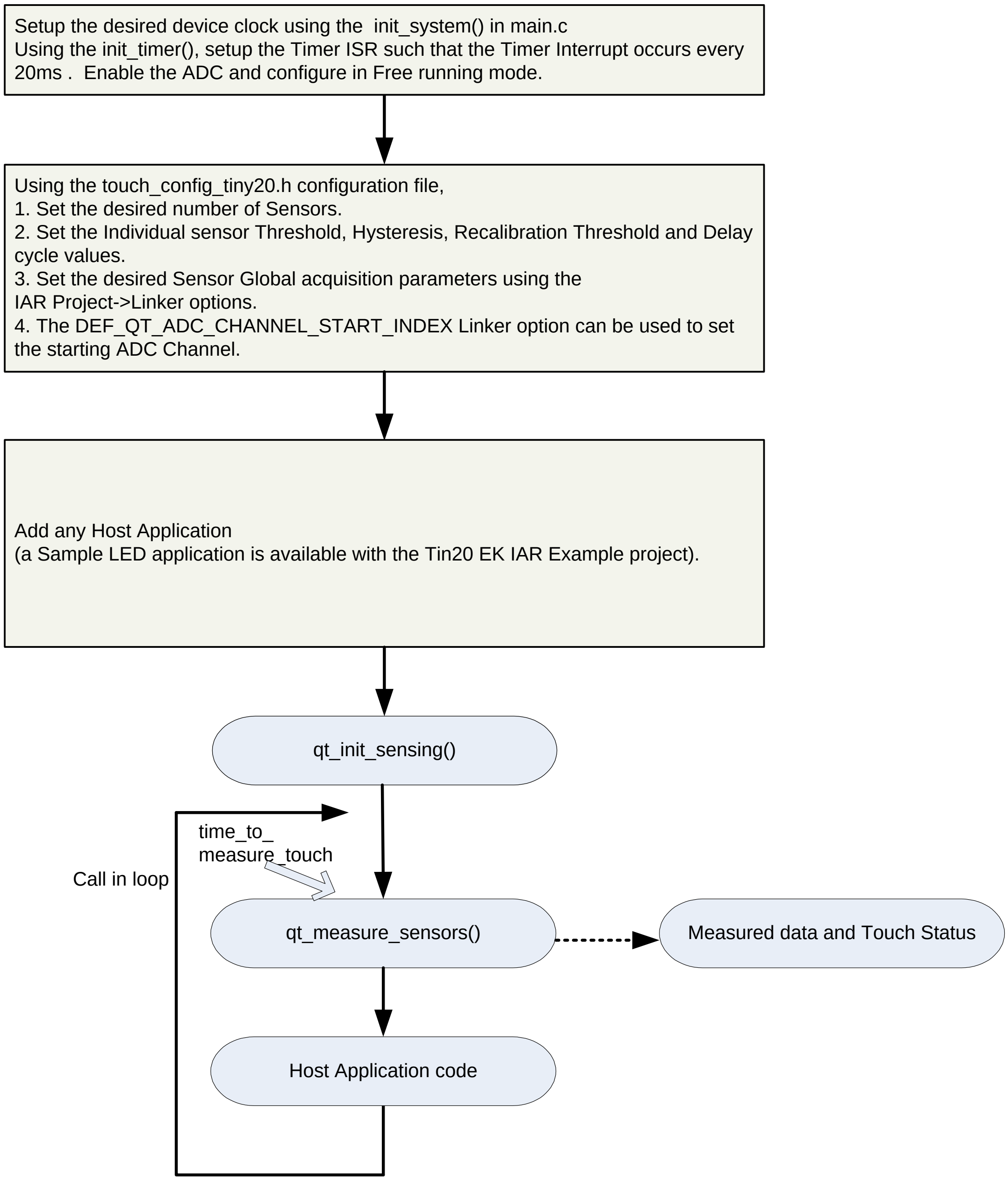


Figure 50 QTouch method for Tiny20 API Flow diagram

6.4.5 QTouch Library configuration parameters for ATtiny20

The Table below describes the various configuration parameters corresponding to the ATtiny20.

Parameter	Description
DEF_QT_QDEBUG_ENABLE	Enable/Disable QDebug debug data communication to QTouch Studio.
DEF_QT_NUM_SENSORS	QTouch number of Sensors. Range: 1u to 5u.
DEF_QT_SENSOR_0_THRESHOLD, DEF_QT_SENSOR_1_THRESHOLD, DEF_QT_SENSOR_2_THRESHOLD, DEF_QT_SENSOR_3_THRESHOLD, DEF_QT_SENSOR_4_THRESHOLD	Sensor detection threshold value. Range: 1u to 255u.
DEF_QT_SENSOR_0_HYSTERESIS, DEF_QT_SENSOR_1_HYSTERESIS, DEF_QT_SENSOR_2_HYSTERESIS, DEF_QT_SENSOR_3_HYSTERESIS, DEF_QT_SENSOR_4_HYSTERESIS	Sensor detection hysteresis value. Refer hysteresis_t in touch_api_tiny20.h HYST_50 = (50% of Sensor detection threshold value) HYST_25 = (25% of Sensor detection threshold value) HYST_12_5 = (12.5% of Sensor detection threshold value) HYST_6_25 = (6.25%, but value is hardlimited to 2)
DEF_QT_SENSOR_0_RECAL_THRESHOLD, DEF_QT_SENSOR_1_RECAL_THRESHOLD, DEF_QT_SENSOR_2_RECAL_THRESHOLD, DEF_QT_SENSOR_3_RECAL_THRESHOLD, DEF_QT_SENSOR_4_RECAL_THRESHOLD	Sensor recalibration threshold value. Refer recal_threshold_t in touch_api_tiny20.h RECAL_100 = (100% of Sensor detection threshold value) RECAL_50 = (50% of Sensor detection threshold value) RECAL_25 = (25% of Sensor detection threshold value) RECAL_12_5 = (12.5% of Sensor detection threshold value) RECAL_6_25 = (6.25%, but value is hardlimited to 4)
DEF_QT_DELAY_CYCLES	Delay cycles that determine the capacitance charge transfer time. Range: 1, 2, 4, 8 or 10 internal System Clock cycles.
DEF_QT_ADC_CHANNEL_START_INDEX	ADC Channel starting index. Range: 1u to 7u.
DEF_QT_AKS_ENABLE	Enable/Disable Adjacent Key suppression (AKS) on all channels.
DEF_QT_DI	Sensor detect integration (DI) limit.Range: 0u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_NEG_DRIFT_RATE*(See Note 1)	Sensor negative drift rate. Units: 100ms, Range: 1u to 127u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_POS_DRIFT_RATE*(See Note 1)	Sensor positive drift rate. Units: 100ms, Range: 1u to 127u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_MAX_ON_DURATION	Sensor maximum on duration. Units: 100ms, Range: 0u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_DRIFT_HOLD_TIME	Sensor drift hold time. Units: 100ms, Range: 1u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_POS_RECAL_DELAY	Positive Recalibration delay. Range: 1u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_NUM_SENSORS_SYM	QTouch number of Sensors Symbol for QTouch Library. MUST be the same as DEF_QT_NUM_SENSORS.
DEF_QT_BURST_LENGTH	Specifies the no:of burst sequence required for a sensor. Multiple burst for adjusting sensitivity by increasing the resolution of the Signal measured.Use higher value for increasing sensitivity. Values: 1u, 4u and 16u, 32u. With a setting of 32u, the Touch Response time would be sluggish.
DEF_CHARGE_SHARE_DELAY	Defines the Charging Share Delay time as an additional Number of CPU Cycles delay to be introduced during the Charge transfer. Values:0 to 255.

Table 18 QTouch Library for ATtiny20 Configuration parameters

Note1:

For the case of ATtiny20 devices, a ‘touch’ causes the Signal value measured on the Sensor to increase above the Sensor Reference value (In the case of Generic Library devices, a ‘touch’ causes the Signal value to decrease below the Reference value).

However, the Negative drift rate and Positive drift rate functionality for the case of Tiny20 devices shall be consistent with the Generic Library case.

So, it is recommended to have a ‘Slower’ Negative Drift rate (4 seconds is the default setting) and a ‘Faster’ Positive Drift rate (1 second is the default setting) for the Tiny20 device.

6.4.6 QTouch Library ATtiny20 Example projects

The QTouch method IAR Example project for the Tiny20 Evaluation Kit can be found in the following path.

\Device_Specific_Libraries\8bit_AVR\AVR_Tiny_Mega_XMega\ATtiny20\ tiny20_ek_iar_qt_example

The Example projects demonstrate the 5 button sensor configuration with a Sample LED application. The Example projects also support QDebug data transfer to QTouch Studio – Touch Analyzer PC Application.

It is possible to configure the number of Sensors in the Example project from 1 to 5 for testing on the ATtiny20 Evaluation kit.

6.4.7 QTouch Library ATtiny20 code and data memory requirements

The code and data memory requirements for QTouch Library for ATtiny20 devices is captured in the Table below. The Table indicates these values for the standalone library and not for the entire Example Project application.

Library	Number of Sensors	Code Memory	Data memory	CStack/RStack
libtiny20-5qt-k-0rs	5	1231 + 15 bytes Const data	70	CStack= 0x1C bytes RStack= 10 (16 bytes) is the Recommended setting.
libtiny20-5qt-k-0rs	4	1231 + 12 bytes Const data	60	CStack= 0x1C bytes RStack= 10 (16 bytes) is the Recommended setting.
libtiny20-5qt-k-0rs	3	1231 + 9 bytes Const data	50	CStack= 0x1C bytes RStack= 10 (16 bytes) is the Recommended setting.
libtiny20-5qt-k-0rs	2	1231+ 6 bytes Const data	40	CStack= 0x1C bytes RStack= 10 (16 bytes) is the Recommended setting.
libtiny20-5qt-k-0rs	1	1231 + 3 bytes Const data	30	CStack= 0x1C bytes RStack= 10 (16 bytes) is the Recommended setting.

Table 19 QTouch Library for ATtiny20 Memory requirements

Data memory for ATtiny20 QTouch Library include the following.

10. QTouch Library data memory – 19 bytes, allocated inside the Library.
11. channel_signals – 2 bytes per Sensor, allocated in main.c
12. channel_references – 2 bytes per Sensor, allocated in main.c
13. sensor_delta – 2 bytes per Sensor, allocated in main.c
14. sensor_general_counter – 2 bytes per Sensor, allocated in main.c
15. sensor_state – 1 byte per Sensor, allocated in main.c
16. sensor_ndil_counter – 1 byte per Sensor, allocated in main.c
17. sensor_states – 1 byte, allocated in main.c

Const Data memory for ATtiny20 QTouch Library include the following.

1. sensor_threshold, 1 byte per Sensor, allocated in main.c
2. sensor_hyst_threshold, 1 byte per Sensor, allocated in main.c
3. sensor_recal_threshold, 1 byte per Sensor, allocated in main.c

6.5 QTouch Library for ATtiny40 device

ATMEL QTouch Library for ATtiny40 can be used for embedding capacitive touch buttons functionality into ATtiny40 device application.

This Section describes the QTouch Library Application Programming API and Configuration interface for QTouch method acquisition using the ATtiny40 devices.

6.5.1 Salient Features of QTouch Library for ATtiny40

6.5.1.1 QTouch method sensor

- One Physical pin per Touch Button.
- 1 to 12 Touch Buttons can be configured.
- Individual Sensor Threshold, Sensor Hysteresis and Sensor Global acquisition parameters can be configured.
- Signal resolution can be configured.
- Charge Share Delay can be configured.
- Adjacent Key Supression (AKS) support.
- QTouch Studio support for Touch data analysis.
- ‘C’ Programming interface for easy inclusion of User application.

6.5.2 Compiler tool chain support for ATtiny40

The QTouch libraries for ATtiny40 devices are supported for the following compiler tool chains.

Tool	Version
IAR Embedded Workbench for Atmel AVR. IAR Compiler.	6.10

Table 20 Compiler tool chains support for ATtiny40 QTouch Library

6.5.3 Overview of QTouch Library for ATtiny40

For an overview of QTouch method based capacitive touch acquisition, refer Section 5.2.1 QTouch Acquistion method.

The QTouch Library for ATtiny40 device allows for Sensor configuration and Sensor Acquisition parameter setting. Based on the input Sensor configuration, the QTouch Library takes care of the capacitive touch acquisition data capture operations on the external capacitive touch sensors. The captured Touch Data and Touch Button ON/OFF Status information is then available for user application.

The diagram below indicates a Typical Sensor arrangement using the Tiny40 device. For one channel configuration, two ADC pins are used for acquisition. For number of touch buttons greater than one, no extra ADC pins are used. Port pins PA0 to PA7 and PB0 to PB3 can be used to support upto 12 Touch Buttons. The Touch Buttons may be connected anywhere on the said port pins.

The Sensor numbering is always in the increasing order of Port pin.

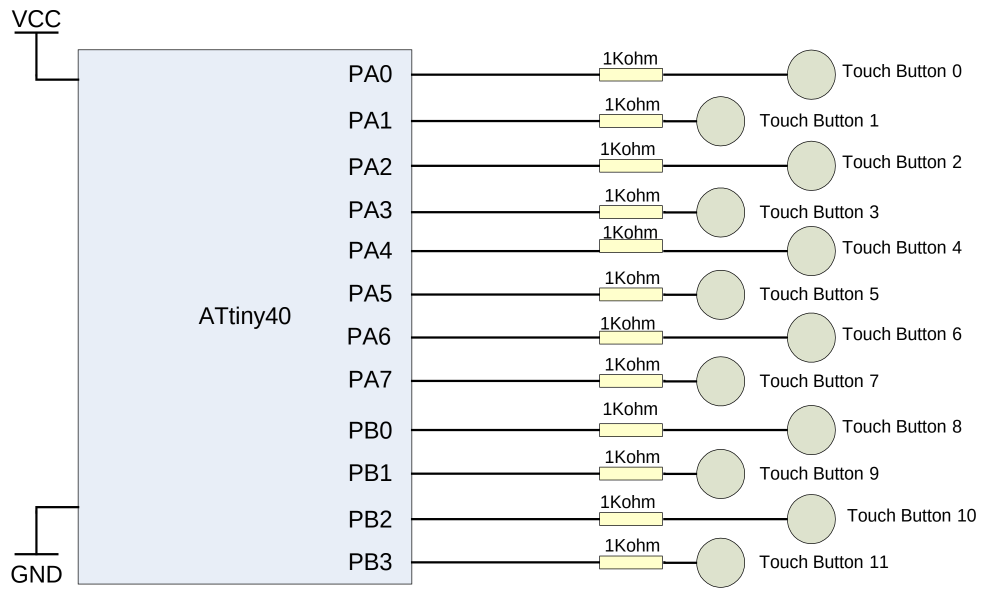


Figure 51 Schematic overview of QTouch on Tiny40

6.5.4 API Flow diagram for ATtiny40

For the QTouch Libraries, the timing information is provided by the Host Application by updating the 'time_current_ms' variable in the Timer ISR. The QTouch Library uses this variable to calculate the necessary timing for Max ON Duration, Drift and Recalibration functionality. Before using the QTouch Libraries, the Timer ISR must be configured appropriately. Also, the Timer Interrupt is used to update the 'time_to_measure_touch' variable inorder to start a capacitive touch measurement.

The touch_config_dp.h configuration header file must be used to set the number of channels based on the library used. For example, if the library used is a 12 channel library then QT_NUM_CHANNELS must be specified as 12 in the touch_config_dp.h. This information must be provided irrespective of the number of channels actually used.

The desired number of touch buttons used can be enabled using the qt_enable_key() routine. The channel numbers are sequential from Port A through Port B. Also, individual sensor Threshold, Hysteresis, AKS group and Recalibration parameters can be set using this function call. The Sensor Global Configuration parameters can also be set by the user by directly accessing the global configuration data structure.

When developing a Host application for ATtiny40 device, ensure that the ADC prescalar is set in such a way that it is in the range of 50 KHz to 250 KHz. For example, if the main clock is running at 8MHz then set the ADC prescalar to 32 or more. This must be done to ensure proper touch sensing acquisition.

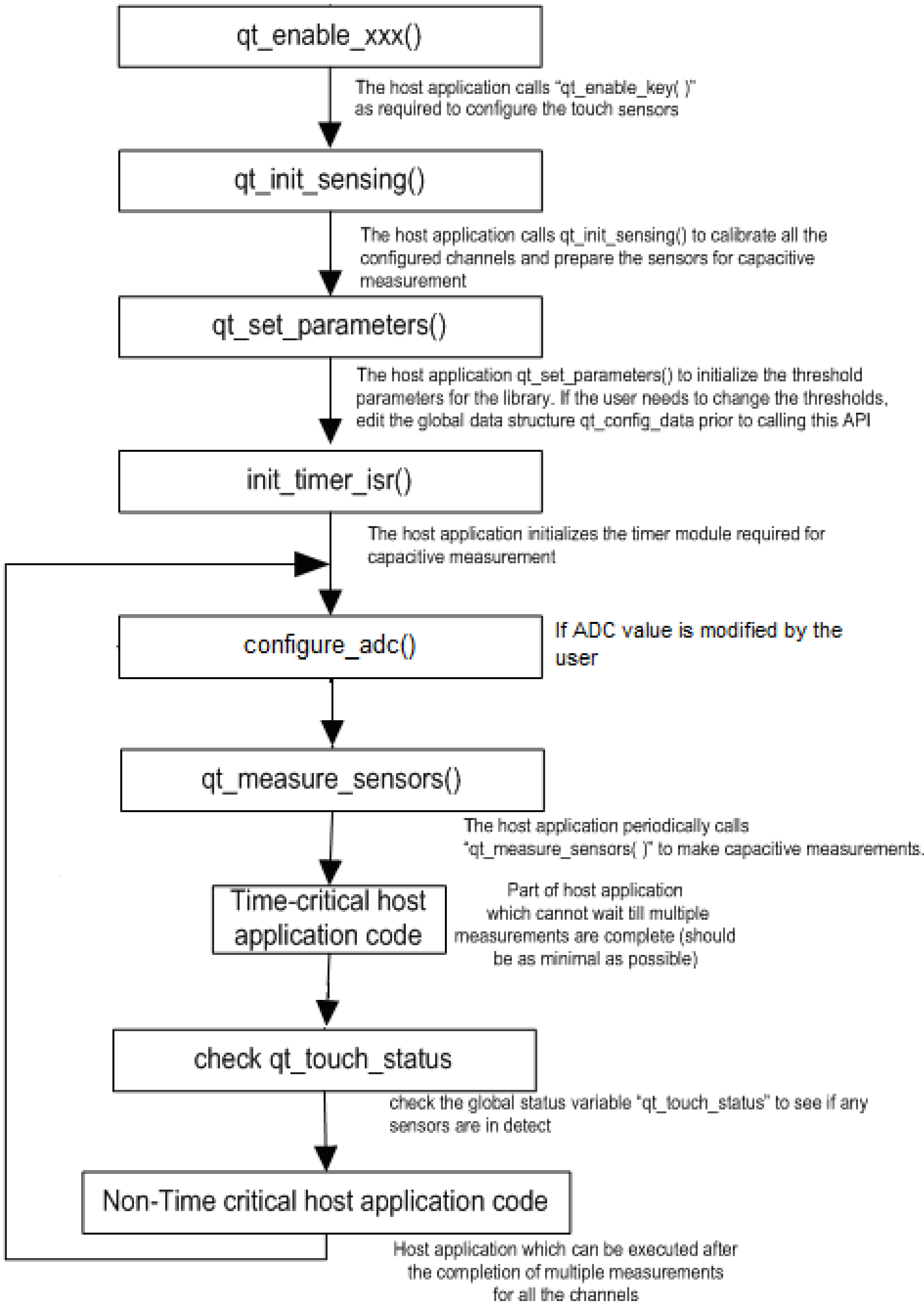


Figure 52 QTouch method for Tiny40 API Flow diagram

6.5.5 QTouch Library configuration parameters for ATtiny40

The Table below describes the various configuration parameters corresponding to the ATtiny40 QTouch Library.

Parameter	Description
DEF_QT_DI	Sensor detect integration (DI) limit.Range: 0u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_NEG_DRIFT_RATE*(See Note 1)	Sensor negative drift rate. Units: 100ms, Range: 1u to 127u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_POS_DRIFT_RATE*(See Note 1)	Sensor positive drift rate. Units: 100ms, Range: 1u to 127u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_MAX_ON_DURATION	Sensor maximum on duration. Units: 100ms, Range: 0u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_DRIFT_HOLD_TIME	Sensor drift hold time. Units: 100ms, Range: 1u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_RECAL_THRESHOLD	• Setting of 0 = 100% of detect threshold • Setting of 1 = 50% of detect threshold • Setting of 2 = 25% of detect threshold • Setting of 3 = 12.5% of detect threshold • Setting of 4 = 6.25% of detect threshold Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_POS_RECAL_DELAY	Positive Recalibration delay. Range: 1u to 255u. Refer Section 5.3 and Section 5.4 for more info.
DEF_QT_PULSE_SCALE	Default value is 0x0 means 1 oversamples and scaling/averaging factor 1. Refer Note on using Pulse Scale Value:
DEF_QT_CHARGE_SHARE_DELAY	Charge Share Delay. Range: 1u – 255u. This value needs to be increased if we use high value of series resistor on sensor pin to ensure proper charge time.

Table 21 QTouch Library for ATtiny40 Configuration parameters

Note1:

For the case of ATtiny40 devices, a ‘touch’ causes the Signal value measured on the Sensor to increase above the Sensor Reference value (In the case of Generic Library devices, a ‘touch’ causes the Signal value to decrease below the Reference value).

However, the Negative drift rate and Positive drift rate functionality for the case of Tiny40 devices shall be consistent with the Generic Library case.

So, it is recommended to have a ‘Slower’ Negative Drift rate (4 seconds is the default setting) and a ‘Faster’ Positive Drift rate (1 second is the default setting) for the Tiny40 device.

Note on using Pulse Scale Value:

This variable is used to increase the resolution of the signal. qt_pulse_scale variable is available to change the oversampling and scaling factor.This is available for all the channels and now with this one should increase the gain for each channel individually.

Function set_qt_pulse_scale is provided in main.c file for reference which will change the values of pulse scale variable.

This variable is divided into higher(pulse/oversamples) and lower(scaling/averaging) nibble.The higher nibble corresponds to number of oversamples and lower nibble corresponds to scaling/averaging. The Pulse value is the power of 2 number of pulses which will accumulate their results.The Scale value is the power of 2 number which will divide the accumulated result from the pulse measurements.

The maximum value of higher nibble possible is 0xA and maximum value of lower nibble is 0xA.

Consideration should be taken on the overall effect on timing when setting Pulse values.

A high gain setting will add considerably time taken to acquire all channels.

6.5.6 QTouch Library ATtiny40 Example projects

The QTouch method IAR Example project for the Tiny40 Evaluation Kit can be found in the following path.

\Device_Specific_Libraries\8bit_AVR\AVR_Tiny_Mega_XMega\ATtiny40\tiny40_qt_example_iar

The Example projects demonstrate the 12 button sensor configuration. The Example projects also support QDebug data transfer to QTouch Studio – Touch Analyzer PC Application.

It is possible to configure the number of Sensors in the Example project from 1 to 12 for testing on the ATtiny40 Evaluation kit.

6.5.7 QTouch Library ATtiny40 code and data memory requirements

The code and data memory requirements for QTouch Library for ATtiny40 devices is captured in the Table below. The Table indicates these values for the standalone library and not for the entire Example Project application.

Library	Number of Sensors	Code Memory	Data memory	CStack/RStack
libtiny40_4qt_k_0rs	4	2400	149	CStack= 40 bytes RStack= 24 bytes
libtiny40_8qt_k_0rs	8	2400	193	CStack= 40 bytes RStack= 24 bytes
libtiny40_12qt_k_0rs	12	2500	235	CStack= 40 bytes RStack= 24 bytes

Table 22 QTouch Library for ATtiny40 Memory requirements

6.5.8 Interrupt Handling in QTouch ADC

In case of ATtiny40, the Interrupts are disabled for each acquisition sample means once per charge transfers cycle and typically this is 1184 Instruction cycles.
Depending on number of oversamples and channels, the total number of times the interrupts are disabled will vary.
Total number of times interrupts disabled = No of oversamples * No of channels.
The Touch measurement will not be affected by longer ISR execution, only the response time will get affected.
The acquisition time is a lot more predictable since a fixed amount of pulses/oversamples are used. The time to execute one measurement will depend on various software parameters like no of oversamples, Charge share delay and CPU Frequency.

For one button to execute with following below parameters, QTouch ADC takes around 0.7 msec.
CPU Freq=4 MHZ, Number of oversamples=1, Charge Share Delay =1.

7 Generic QTouch Libraries for 2K Devices

7.1 Introduction

This section provides information about the QTouch library Acquisition Support for Tiny devices with 2K Flash memory. These libraries have the same API's as Generic libraries, except for a few which are not supported.Information about the API's are provided in touch_api_2kdevice.h file which is placed at location mentioned in section 5.6.10.1

7.2 Devices supported

The list of different devices that are supported by the QTouch library for 2K devices is given below:

1. ATtiny2313A
2. ATtiny261A
3. ATtiny24A
4. ATtiny25A

Complete information is available in Library_Selection_Guide.xls.

7.3 Salient Features of QTouch Library for 2K Devices

- 1 to 4 Touch Buttons can be configured. Supports maximum of 4 Buttons.
- Libraries in variants of 1, 2 and 4 channels are provided.
- 2K device libraries are supported only for IAR.
- Library API's are same as Generic QTouch libraries.
- Support for more than one pair of SNS and SNSK ports are not available for 2K tiny devices.

NOTE:

No AKS, no Power Optimization and no pin configurability support in case of 2K device libraries.

The change information like library status flags which reflects if there is any change in Reference values, rotor slider position change status flags etc are not part of the 2K device libraries except burst again flag.

7.4 Library Variants

For Different library variants available for 2K Devices, please refer Library_Selection_Guide.xls

7.5 QTouch API for 2K Devices and Usage

This section describes the different API's used during touch sensing. Using the API, Touch sensors and the associated channels can be defined. Once touch sensing has been initiated by the user, the host application can use the API to make touch measurements and determine the status of the sensors.Refer section 5.6.6 and Figure 5.6 for API usage

7.5.1 touch_api_2kdevice.h - public header file

The touch_api_2kdevice.h header file is the public header file which needs to be included in user's application. The type definitions and function prototypes of the API's listed in sections 5.6.3 , 5.6.4 and 5.6.5

The touch_api_2kdevice.h header file is located in the library distribution in the following directory.

- ..\Atmel_QTouch_Libraries_5.x\Generic_QTouch_Libraries\include

The constant/symbol definitions can be placed in any of the following.

- Defined user's project options. All the constants/symbols must be defined for both the compiler and assembler preprocessing definitions.
- As an alternative, it is also declared in the touch_qt_config_2kdevice.h file. The user may modify these defined values based on the requirements.

Global settings common to all sensors and sensor specific settings are listed in sections 5.3 and 5.4 respectively

7.5.2 Sequence of Operations and Using the API

Figure 5-6 illustrates the sequence of operations required to be performed to add touch to an end application. By using the simple API's as illustrated in the sequence flowchart, the user can add touch sensing in his design.

7.5.2.1 Channel Numbering

- 1-channel library – supports 1 channel using 1 consecutive pins on different SNS and SNSK ports (or) supports 1 channel using 2 consecutive pins on the same port used for both SNS and SNSK lines. This library requires 1 or 2 port.
- 2-channel library – supports up to 2 channels using 2 consecutive pins on different SNS and SNSK ports (or) supports up to 2 channels using 4 consecutive pins on the same port used for both SNS and SNSK lines. This library requires 1 or 2 ports.
- 4-channel library – supports up to 4 channels using 4 consecutive pins on different SNS and SNSK ports (or) supports up to 4 channels using 8 consecutive pins on the same port used for both SNS and SNSK lines. This library requires 1 or 2 ports.

7.5.2.1.1 Channel numbering when routing SNS and SNSK pins to different ports

When SNS and SNSK pins are available on different ports, the channel numbering follows the pin numbering in the ports selected.

- The channel numbers follow the pin numbers starting with the LSB (pin 0 is channel 0 and pin 3 is channel 3).
- Since the channel numbers are fixed to the pins of the SNS and SNSK ports, if the design calls for use of a subset of the pins available in the SNS and SNSK ports, the user has to skip the channel numbers of the unused SNS and SNSK pins.

    For example, on a 4 channel configuration using SNS and SNSK ports, if pin 2 is not used for touch sensing (on both SNS and SNSK ports), channel number 2 is unavailable and care should be taken while configuring the channels and sensors to avoid using this channel. Also, the SNS and SNSK masks are assigned properly as explained in section 7.5.2.2

7.5.2.1.2 Channel numbering when routing SNS and SNSK pins to the same port

When SNS and SNSK pins are connected to the same port, the even pin numbers will be used as SNS pins and the odd pins will be used as the SNSK pins.

- The number of channels supported will be limited 4 channels
- For e.g., for a 4 channel configuration where the SNS and SNSK pins are connected to Port B, the port pins 0&1 are used for channel 0.
- The channel number is derived from the position of the pins used for SNS and SNSK lines for any channel.

  channel number = floor([SNS(or SNSK) pin number] / 2)

  - For e.g., pins 4 and 5 are connected to a SNS/SNSK pair and the channel number associated with the SNS/SNSK pin is 2.

7.5.2.2 Rules For Configuring SNS and SNSK masks for 2K Devices

The libraries internally need SNS_array and SNSK_array masks. These masks need to be defined under Macro QTOUCH_STUDIO_MASKS as per the following rules given below:

1. In case of Interport, SNS_array[0] and SNSK_array[0] mask is used for configuring the Channel0 and Channel2.And SNS_array[1] and SNSK_array[1] mask is used for configuring the Channel1 and Channel3.And In case of Intraport SNS_array[0] and SNSK-array[0] are used for all the four channels configured based on enabled bits in SNS_array[0] and SNSK-array[0].

2. The channel numbers are allocated based on enabled SNS pins starting from LSBBit.

In case of Interport, Keys on adjacent channels should be placed on different masks. Channel0 and Channel1 should be on different SNS/SNSK masks ie channel0 on SNS_array[0]/SNSK_array[0] and channel1 on SNS_array[1]/ SNSK_array[1].

But in case of Intraport, Keys on adjacent channels should be placed on same masks. Channel0 and Channel2 should be on same mask ie SNS_array[0]/SNSK_array[0] and Channel1 and Channel3 on SNS_array[1]/ SNSK_array[1].

7.5.2.2.1 Configuring SNS and SNSK masks in case of Interport:

1. Enable the Bit0 in SNS_array[0] and Bit0 in SNSK_array[0] mask when enabling Channel0.
2. Enable the Bit1 in SNS_array[1] and Bit1 in SNSK_array[1] mask when enabling Channel1.
3. Enable the Bit2 in SNS_array[0] and Bit2 in SNSK_array[0] mask when enabling Channel2.
4. Enable the Bit3 in SNS_array[1] and Bit3 in SNSK_array[1] mask when enabling Channel3.

Example 1:
In a 4 channel library, two keys on channel 0 and 3 are enabled.SNS on Port A and SNSK on Port B .Channel0 will A0B0 and Channel3 will be A3B3.
The SNS and SNSK masks will be
SNS_array[0]=0x01;
SNS_array[1]=0x08;
SNSK_array[0]=0x01;
SNSK_array[1]=0x08;

7.5.2.2.2 Configuring SNS and SNSK masks in case of Intraport:

1. Enable the Bit0 in SNS_array[0] and Bit1 in SNSK_array[0] mask when enabling Channel0.
2. Enable the Bit2 in SNS_array[0] and Bit3 in SNSK_array[0] mask when enabling Channel1.
3. Enable the Bit4 in SNS_array[0] and Bit5 in SNSK_array[0] mask when enabling Channel2.
4. Enable the Bit6 in SNS_array[0] and Bit7 in SNSK_array[0] mask when enabling Channel3.

Example 1:
In a 4 channel library, two keys on channel 0 and 3 are enabled.SNS and SNSK on Port B .Channel0 will B0B1 and Channel3 will be B6B7.
The SNS and SNSK masks will be
SNS_array[0]=0x41;
SNS_array[1]=0x00;
SNSK_array[0]=0x82;
SNSK_array[1]=0x00;

7.5.3 Integrating QTouch libraries for 2K Devices in your application

In order to Integrate QTouch libraries for 2K devices, the constants and symbol names listed in Table 1 below need to be defined in the user application. These can be defined in either the compiler/assembler preprocessing definitions or in the touch_t_config_2kdegice.h file. Example projects are provided for all the four devices supported.Refer 5.6.10.1 for directory structure of all the files.

Table 1: Constant and symbol name definitions required to use the QTouch acquisition method libraries for 2K device libraries

Symbol / Constant name	Range of values	Comments
QTOUCH	This macro has to be defined in order to use QTouch libraries.	
SNS & SNSK	Refer to library selection guide.	
_SNS_SNSK_SAME_PORT_	Comment/uncomment define	To be enabled if the same port is used for SNSK and SNS pins for QTouch. If SNSK and SNS pins are on different ports then this definition is not required.
QT_NUM_CHANNELS	1, 2 and 4 for 2K device libraries.	
QT_DELAY_CYCLES	1 to 255	Please refer to section 5.6.8.
QTOUCH_STUDIO_MASKS	This macro has to be defined in order to use QTouch libraries for 2K devices.	SNS_array and SNSK_array masks variablesare initialized under this Macro in main file.Refer section 7.5.2.2

The following files are to be added along with the touch library and user application before compilation:

- For ATtiny 2K devices - touch_api_2kdevice.h, touch_qt_config_2kdevice.h and qt_asm_tiny_mega_2kdevice.S

7.6 MISRA Compliance Report

This section lists the compliance and deviations for MISRA standards of coding practice for the QTouch acquisition method libraries for 2K devices

7.6.1 What is covered

The QTouch acquisition method libraries for 2K devices adhere to the MISRA standards. The additional reference code provided in the form of sample applications is not guaranteed to be MISRA compliant.

7.6.2 Target Environment

Development Environment	**IAR Embedded Workbench**
MISRA Checking software	The MISRA C Compliance has been performed for the library using MISRA C 2004 Rules in IAR Workbench development environment.
MISRA Rule set applied	MISRAC 2004 Rule Set

7.6.3 Deviations from MISRA C Standards

7.6.3.1 QTouch acquisition method libraries for 2K devices

The QTouch acquisition method libraries were subject to the above mentioned MISRA compliance rules. The following exceptions have not been fixed as they are required for the implementation of the library.

Applicable Release	**QTouch libraries**	
Rule No	**Rule Description**	**Exception noted / How it is addressed**
1.1	Rule states that all code shall conform to ISO 9899 standard C, with no extensions permitted.	This Rule is not supported as the library implementation requires IAR extensions like __interrupt. These intrinsic functions relate to device hardware functionality, and cannot practically be avoided.
10.1	Rule states that implicit conversion from Underlying long to unsigned long	The library uses macros to combine symbol definitions to form a unique expanded symbol name and in this, the usage of unsigned qualifiers for numeric constants (e.g. 98u) causes name mangling. This is the only occurrence of this error in the library.
10.6	This Rule says that a 'U' suffix shall be applied to all constants of 'unsigned' type	The library uses macros to combine symbol definitions to form a unique expanded symbol name and in this, the usage of unsigned qualifiers for numeric constants (e.g. 98u) causes name mangling. This is the only occurrence of this error in the library.
14.4	Rule states that go-to statement should not be	The library uses conditional jump instructions to reduce the code footprint at a few locations and

	used.	this is localized to small snippets of code. Hence this rule is not supported.
19.10	Rule states that In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis	There is one instance where the library breaks this rule where two macro definitions are combined to form a different symbol name. Usage of parenthesis cannot be used in this scenario.
19.12	Rule states that there shall be at most one occurrence of the # or ## preprocessor operator in a single macro definition	There is one instance in the library where this rule is violated where the library concatenates two macro definitions to arrive at a different definition.

8 Revision History

The table below lists the revision history for chapters in the user guide.

QTouch Library User guide Revision History		
Date/Version	**Chapter**	**Change notes**
May 2009 Ver2.0	All	2nd release of QTouch library users guide
Sep 2009 Ver. 3.0	All	Re-structured user guide with new and expanded sections
Nov 2009 Ver. 3.1	6.3, 6.9, 6.10, 7, 10	• Updated API changes • Updated new libraries and device support information • Updated debug interface information supported by the QTouch libraries • Updated known issues table
Dec 2009 Ver. 3.2	6.10.4, 7.1.2, 7.1.5, 7.1.6, 10, 7.2.4.2.2, 7.2.4.3.7, 7.2.4.3.2, 7.2.4.3.5, 7.2.4.3.7,	• Added section about configuring unused pins in user application • Added more information to some sections to clear ambiguity • Updated Memory footprint information for IAR and GCC compiled QTouch libraries. • Updated known issues table • Added the device support, port combinations, memory requirements • QMatrix IAR and GCC libraries to support ATmega325P, ATmega645, and ATtiny167. • Modified port combinations for the 165P for QMatrix libraries. • Few Port combinations added in case of ATmega88 libraries.
Feb 2010 Ver 4.0	All chapters changed	• A separate library selection guide is provided external to the user guide. All sections included in the library selection guide have been removed from the user guide. • All sections have been updated to account for the improved configurability of the libraries.
Apr 2010 Ver 4.1		• Added sections related to Positive Recalibration Delay, Position Hysteresis, and Position Resolution.

		• Device support extended for QMatrix for the release has been added in section 5.7.2.4.1 and 5.7.2.3 • In case of QMatrix, 4 (4x1) channel has been added wherever needed and in case of ATxmega devices 56 (8x7) channel has been added according to the changes • QTouch Library for UC3L API Device Specific Libraries Section has been added.
May 2010 Ver 4.2		• Qtouch acquisition libraries support will be available for ATSAM3U and ATSAM3S devices. • Qdebug protocol support will be extended for all example projects. • Analog comparator usage and burst length setting recommendation Note added for UC3L QMatrix method. • QMatrix device support added for AT90USB82 / 162 / 646 / 647 / 1286
July 2010 Ver 4.3	Section 5.8, Section 5.7.2.4	• Device support added for Tiny44/84/461/861 • Added the details on Pin configuration support for both QTouch and QMatrix libraries. • Added section related to the usage of the pin configurator tool on QTouch Studio.(section 5.8) • Added sections for Tiny20 and Tiny40 Devices.
Jan 2011 Ver 4.3.1	Chapter 7, Section 5.6.11.2.1, Section 5.7.11.2.1	• Device support added for QTouch 2K devices ATtiny2313A/261A/24A/25A. • Added Chapter 7 on 2K Device libraries. • QTouch Support added for UC3C family devices. • QTouch Support added for ATtiny87 device • Tiny20 code memory requirement section updated.
Aug 2011 Ver 4.4	Chapter 2 Section 5.6.10.3 Section 6.5 Section 5.5.3	• Added Feature Comparison Table • Section 5.6.10 changed and updated for Support for QMatrix AT32UC3C0512 Device • Section 6.5 changed and updated for ATtiny40 libraries • Section 5.5.3 added for Guard Channel
April 2012 Ver 5.0	Section 6.5.5 Section 6.5.8	• Section 6.5.5 changed and updated for ATtiny40 libraries • Section 6.5.8 changed and updated for ATtiny40 libraries • Section 5.6.10.3.1 changed and updated for 8-bit Qmatrix libraries related to Shared YA/YB. • Sections 5.6.10.1, 5.6.10.2, 5.7.1.2, 5.7.1.3.1 updated for sam4s addition

Headquarters

ATMEL Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

ATMEL Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

ATMEL Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

ATMEL Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
http://www.atmel.com/

Literature Request
www.atmel.com/literature

Technical Support
AVR Libraries:
touch@atmel.com
SAM Libraries:
at91support@atmel.com

Sales Contact
www.atmel.com/contacts