Atmel Touch Library

User Guide

8207B–AT42–01/09

Table of Contents

Section 4

Section 1

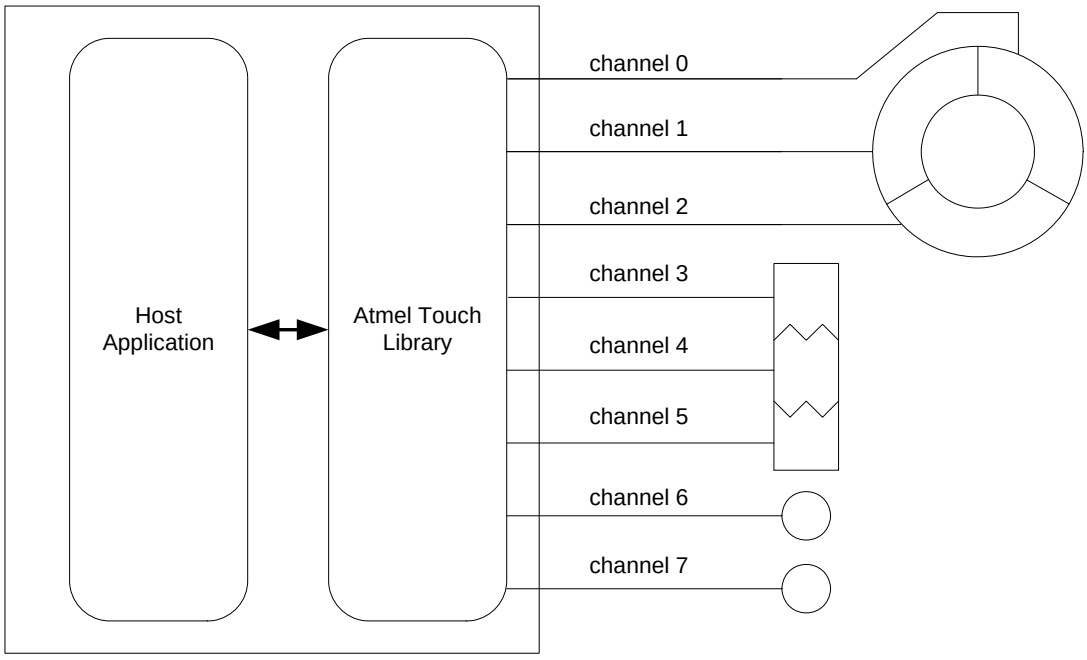
Adding Touch Sensing to an Application

1.1 Introduction

The Atmel Touch Library is added to an application by linking in the library file, including the library header file, and calling the functions defined in the application programming interface (API).

Figure 1-1 shows a typical configuration, in which the Atmel Touch Library has been added to a host application running on an ATmega88. The library supports eight touch channels numbered 0 to 7. Channels 0 to 2 have been configured as a rotor sensor, channels 3 to 5 have been configured as a slider sensor, and channels 6 and 7 are configured as key sensors. The host application calls the library to configure these sensors, and to make and process capacitive measurements.

Figure 1-1. **Atmel Touch Library running on an ATmega88**



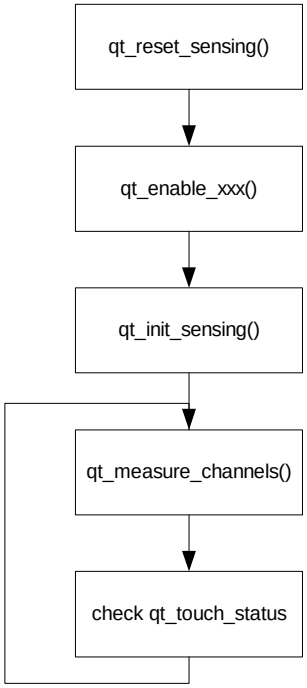
The Atmel Touch Library only runs when called from the host application. It uses no timers, interrupts, or other chip resources apart from ROM, RAM, some register variables, and GPIO. It only supports touch sensing; the host application must provide any other functionality required, such as reading switches, driving LEDs, communicating with other processors, etc.

1.2 Host Application Program Flow

The general flowchart for use of the Atmel Touch Library is shown in Figure 1-2 on page 1-2. The steps are as follows:

1. The host application (optionally) calls “qt_reset_sensing()” to reset all channels and touch sensing parameters to their default states. This step is only required if the host wants to dynamically reconfigure the library at runtime.
2. The host application calls “qt_enable_key()”, “qt_enable_rotor()” and/or “qt_enable_slider()” as required to configure the touch sensors.
3. The host application calls “qt_init_sensing()” to initialize the library.
4. Thereafter, the host application regularly calls “qt_measure_channels()” to make capacitive measurements. After each call, it can check the global variable “qt_touch_status” to see if any sensors are in detect, and the angle or position of any enabled rotors or sliders.

Figure 1-2. Flowchart for Using the Atmel Touch Library in a Host Application.



1.3 Host Application Requirements

The host application must meet certain criteria for touch sensing to work correctly:

- It must track the current time.

  This information is passed to the code library as an argument to the function “qt_measure_channels()”. This is used for time-based library operations such as drifting.
- The GPIO internal pull-ups must be disabled when calling the library.

  Setting the “PUD” bit in the “MCUCR” register does this.
- The library must be called often enough to provide a reasonable response time to user touches.

  During a call to the library functions the main host application code is not running. There is thus a trade-off between the processor time available to the host application, the power usage of the system, and the system responsiveness.
- A sufficient stack size for both itself and the library.

  The host application stack must be large enough for the library, plus its own operation when calling library functions, plus any enabled interrupts that may be serviced during a library function call.

1.4 Example Host Application

The Atmel Touch Library evaluation kit includes a complete example host application. This comprises an IAR™ project in which a supplied library and header file are linked into an example application. The kit evaluation board is supplied pre-programmed with this application.

Section 2

Using the Atmel Touch Library

2.1 Sensing Channels

2.1.1 Disabled Channels

All sensing channels are disabled by default.

The host application can use unused sensing pins as GPIO.

2.1.2 Sensor Order

Sensors are numbered in the order in which they are enabled.

For example, consider this code fragment:

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

/* 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 );
```

In this code, the slider on channels 0 to 2 will be sensor 0, as it is the first enabled sensor. The slider is in detect if “qt_touch_status. sensor_states” bit 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:

```
/* 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.

In the API “sensor_deltas” array, the values reported are for each sensor, not for each channel. The order of the values reported depends on the order in which the sensors are enabled. In the example above, “sensor_deltas[0”] would report the overall delta for the rotor on channels 3 to 5, “sensor_deltas[1]” and “sensor_deltas[2]” would report the delta values for the two key sensors on channels 6 and 7 respectively, and “sensor_deltas[3]” would report the overall delta for the slider on channels 0 to 2.

2.2 Interrupts

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. As a rule of thumb, the combined durations of any ISRs during a capacitive measurement should be less than 1 ms. This can be tested during system development by checking the burst duration on the touch channels on an oscilloscope. If the burst duration varies by more than 1 ms when the user is not touching any sensors, then ISRs could adversely affect the measurements.

2.3 Sensor Measurements

2.3.1 Avoiding Cross-talk

In Atmel Touch library variants that use QTouch™, adjacent sensors are not measured at the same time. This prevents interference due to cross-talk between adjacent channels, but means that some sensor configurations take longer to measure than others.

For example, if an 8-channel chip is configured to support 8 keys, then the library will measure the keys on channels 0, 2, 4, and 6 simultaneously, and then the 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.

2.3.2 Multiple Measurements

The library will automatically perform multiple measurements on a sensor in certain conditions, typically when sensors are calibrating, filtering into detect, or filtering out of detect. Multiple measurements are performed to resolve these situations before returning to the host application. This means that some calls to measure the sensors take longer than others.

2.3.3 Measurement Limit

In Atmel Touch library variants that use QTouch™, measurements on a channel are automatically stopped when they reach a value of 8192 pulses. In this case a signal level of 1 will be reported for the channel. This limit is long enough for practical uses of QTouch sensing, and traps hardware fault conditions such as shorted-out sampling capacitors.

2.3.4 Linking Library Functions

When building a host application, library functions will only be linked in if they are actually called. This means that code space can be saved in the host application if it does not call the optional functions “qt_reset_sensing()” and “qt_calibrate_sensing()”. This may not always be possible.

2.3.5 Filtering Signal Measurements

The Atmel Touch Library API contains a function pointer called “qt_filter_callback”. You 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 it has made capacitive measurements, but before it has processed them.

2.3.5.1 Example 1: Averaging the Last Two Signal Values

1. Add a static variable in the main module:

```
/* filter for channel signals */
static uint16_t filter[QT_NUM_CHANNELS][2];
```

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

```
/* example signal filtering function */
static void filter_data_mean_2( void );
```

3. When configuring the Atmel Touch library, set the callback function pointer:

```
/* set callback function */
qt_filter_callback = filter_data_mean_2;
```

4. Add the filter function:

```
void filter_data_mean_2( void )
{
   uint8_t i;

   /*
   * Shift previously stored channel signal data.
   * Store new channel signal data.
   * Set library channel signal data = mean of last 2 values.
   */
   for( i = 0u; i < QT_NUM_CHANNELS; i++ )
   {
      filter[i][0] = filter[i][1];
      filter[i][1] = channel_signals[i];
      channel_signals[i] = ( ( filter[i][0] + filter[i][1] ) / 2u );
   }
}
```

The signal values processed by the Atmel Touch code library are now the mean of the last two actual signal values.

2.3.5.2 Example 2: 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 Touch 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] = channel_signals[i];
      channel_signals[i] = ( ( filter[i][0] +
                               filter[i][1] +
                               filter[i][2] +
                               filter[i][3] ) / 4u );
   }
}
```

The signal values processed by the Atmel Touch code library are now the mean of the last four actual signal values.

2.3.5.3 Example 3: Using the Median of the Last Three Signal Values

1. Add a static variable in the main module:

```
/* filter for channel signals */
static uint16_t filter[QT_NUM_CHANNELS][3];
```

2. Add filter function prototypes to the main module:

```
/* example signal filtering function */
static void filter_data_median_3( void );
static uint16_t median_3( uint16_t d0, uint16_t d1, uint16_t d2 );
```

3. When configuring the Atmel Touch library, set the callback function pointer:

```
/* set callback function */
qt_filter_callback = filter_data_median_3;
```

4. Add the filter functions:

```
void filter_data_median_3( void )
{
   uint8_t i;

   /*
   * Shift previously stored channel signal data.
   * Store new channel signal data.
   * Set library channel signal data = median of last 3 values.
   */
   for( i = 0u; i < QT_NUM_CHANNELS; i++ )
   {
      filter[i][0] = filter[i][1];
      filter[i][1] = filter[i][2];
      filter[i][2] = channel_signals[i];

      channel_signals[i] = median_3( filter[i][0], filter[i][1], filter[i][2] );
   }
}

uint16_t median_3( uint16_t d0, uint16_t d1, uint16_t d2 )
{
   uint16_t rtnval;

   if( d0 > d1 )
   {
      if( d1 > d2 )
      {
         rtnval = d1;
      }
      else if( d0 > d2 )
      {
         rtnval = d2;
      }
      else
      {
         rtnval = d0;
      }
   }
   else
   {
      if( d1 < d2 )
      {
         rtnval = d1;
      }
      else if( d0 < d2 )
      {
         rtnval = d2;
      }
```

```
        else
        {
            rtnval = d0;
        }
    }

    return rtnval;
}
```

The signal values processed by the Atmel Touch Library are now the median of the last three actual signal values.

Section 3

Application Programming Interface

3.1 Introduction

This section defines the API that the Atmel Touch Library offers to host applications. This API is generic, and the facilities offered by a particular version of the library are related to the capabilities of the device concerned.

Touch sensing can be configured globally to determine, for example, how quickly environmental changes are tracked.

Individual sensors can be configured to assign channels to them, and set their touch sensing parameters.

Once touch sensing has started, the host application can call the library to make touch measurements. It can then read the touch status (for example, which keys are touched).

3.2 Manifest Constants

The API defines the manifest constants listed in Table 3-1 that document the library. The library has been built using these values, and they should not be changed.

Table 3-1. Manifest Constants

Manifest Constant	Notes
QT_NUM_CHANNELS	The number of touch channels supported by the library.
QT_MAX_NUM_ROTORS_SLIDERS	The maximum number of rotors or sliders supported by the library.

3.3 Type Definitions

3.3.1 Typedefs

The API defines the typedefs listed in Table 3-2.

Table 3-2. Typedefs

Typedef	Notes
uint8_t	An unsigned 8-bit number.
uint16_t	An unsigned 16-bit number.
int16_t	A signed 16-bit number.
threshold_t	An unsigned 8-bit number setting a sensor detection threshold.

3.3.2 Enumerations

The API uses the enumerations listed in Table 3-3.

Table 3-3. Enumerations

Name	Values	Notes
aks_group_t	NO_AKS_GROUP AKS_GROUP_1 AKS_GROUP_2 AKS_GROUP_3 AKS_GROUP_4 AKS_GROUP_5 AKS_GROUP_6 AKS_GROUP_7	Which AKS™ group, if any, a sensor is in. NO_AKS_GROUP = sensor is not in an AKS group, and cannot be suppressed. AKS_GROUP_x = sensor is in AKS group x.
channel_t	CHANNEL_0 CHANNEL_1 CHANNEL_2 CHANNEL_3 CHANNEL_4 CHANNEL_5 CHANNEL_6 CHANNEL_7	The channel(s) in a sensor.
hysteresis_t	HYST_50 HYST_25 HYST_12_5 HYST_6_25	A sensor detection hysteresis value. This is expressed as a percentage of the sensor detection threshold. 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)
recal_threshold_t	RECAL_100 RECAL_50 RECAL_25 RECAL_12_5 RECAL_6_25	A sensor recalibration threshold. This is expressed as a percentage of the sensor detection threshold. 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)
resolution_t	RES_1_BIT RES_2_BIT RES_3_BIT RES_4_BIT RES_5_BIT RES_6_BIT RES_7_BIT RES_8_BIT	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.

3.3.3 Structs

The API uses the struct listed in Table 3-4. The global variable “qt_touch_status” of this type is declared, and shows the current state of all enabled sensors (Section 3.4).

Table 3-4. Struct

Struct	Field	Notes
qt_touch_status_t	sensor_states	The state (on/off) of the library sensors. Bit “n” = state of sensor “n”: 0 = not in detect, 1 = in detect.
	rotor_slider_values[]	Rotor angles or slider positions. These values are valid when “sensor_states” shows that the corresponding rotor or slider sensor is in detect.

3.4 Global Touch Sensing Status

The global touch sensing status is available to the host application through the variable listed in Table 3-5.

Table 3-5. Global Touch Sensing Status

Variable	type	Notes
qt_touch_status	qt_touch_status_t	The state of the library sensors.

3.5 Global Touch Sensing Configuration

Touch sensing is configured globally with the parameters listed in Table 3-6.

Table 3-6. Touch Sensing Configuration

Variable	type	Notes
qt_di	uint8_t	Sensor detect integration (DI) limit. Default value: 4
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
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)
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)
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)
qt_recal_threshold	recal_threshold_t	Sensor recalibration threshold. Default: RECAL_50 (recalibration threshold = 50 percent of detection threshold)

3.6 Per-channel Touch Sensing Configuration

In library variants based on the QMatrix™ technology, the data array listed in Table 3-7 is available to the host application.

Table 3-7. Per-channel Touch Sensing Configuration

Variable	type	Notes
qt_burst_length[]	uint16_t	The burst length on each QMatrix channel in units of pulses. Default value: 64 pulses

3.7 Touch Sensing Data

The data arrays listed in Table 3-8 are available within the API. These are useful during system development to check that touch sensing is operating as expected.

Table 3-8. Touch Sensing Data Arrays

Array	Element Type	Notes
channel_signals[]	uint16_t	The measured signal on each channel.
channel_references[]	uint16_t	The reference signal for each channel.
sensor_deltas[]	int16_t	The signal delta on each sensor (which may comprise multiple channels).

3.8 Hook For User Functions

The function pointer “qt_filter_callback” is provided as a hook for user-supplied filter functions. This function is called after the library has made capacitive measurements, but before it has processed them. The user can use this hook to apply filter functions to the measured signal values.

By default the pointer is NULL, and no function is called.

3.9 Configuring Sensors

3.9.1 Configuration Functions

The functions listed in Table 3-9 are used to assign channels to sensors, and to configure the sensor parameters.

Table 3-9. Functions

Function	Notes
qt_enable_key()	Enable a key sensor.
qt_enable_rotor()	Enable a rotor sensor.
qt_enable_slider()	Enable a slider sensor.

3.9.2 qt_enable_key()

This function enables a key sensor.

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

The parameters are as follows:

- channel = which touch channel the key sensor uses
- aks_group = which AKS group (if any) the sensor is in
- detect_threshold = the sensor detection threshold
- detect_hysteresis = the sensor detection hysteresis value

The sensor number corresponding to the key depends on the order in which sensors are enabled. The first sensor enabled is sensor 0, the second is sensor 1, and so on.

The current state of the key (on or off) can be checked in “qt_touch_status.sensor_states”.

3.9.3 qt_enable_rotor()

This function enables a rotor sensor.

```
void qt_enable_rotor(
     channel_t from_channel,
     channel_t to_channel,
     aks_group_t aks_group,
     threshold_t detect_threshold,
     hysteresis_t detect_hysteresis,
     resolution_t angle_resolution,
     uint8_t angle_hysteresis );
```

The parameters are as follows:

- from_channel = the first channel in the rotor sensor
- to_channel = the last channel in the rotor sensor
- aks_group = which AKS group (if any) the sensor is in
- detect_threshold = the sensor detection threshold
- detect_hysteresis = the sensor detection hysteresis value
- angle_resolution = the resolution of the reported angle value
- angle_hysteresis = the hysteresis of the reported angle value

The sensor number corresponding to the rotor depends on the order in which sensors are enabled. The first sensor enabled is sensor 0, the second is sensor 1, and so on.

The current state of the rotor (on or off) can be checked in “qt_touch_status.sensor_states”.

The rotor value is in “qt_touch_status.rotor_slider_values[]”. Which array element is used depends on the order in which sensors 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 on.

3.9.4 qt_enable_slider()

This function enables a slider sensor.

```
void qt_enable_slider(
     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 );
```

The parameters are as follows:

- from_channel = the first channel in the slider sensor
- to_channel = the last channel in the slider sensor
- aks_group = which AKS group (if any) the sensor is in
- detect_threshold = the sensor detection threshold
- detect_hysteresis = the sensor detection hysteresis value
- position_resolution = the resolution of the reported position value
- position_hysteresis = the hysteresis of the reported position value

The sensor number corresponding to the slider depends on the order in which sensors are enabled. The first sensor enabled is sensor 0, the second is sensor 1, and so on.

The current state of the slider (on or off) can be checked in “qt_touch_status.sensor_states”.

The slider value is in “qt_touch_status.rotor_slider_values[]”. Which array element is used depends on the order in which sensors 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 slider value is valid when the slider is on.

3.10 Measuring and Checking the Touch Status

3.10.1 Touch Status Functions

Once all required channels have been configured as keys, rotors, or sliders, touch sensing is initialized by calling the function “qt_init_sensing()” (see Section 3.10.3).

The host application can then perform a touch measurement by calling the function “qt_measure_sensors()” (see Section 3.10.4), passing in as a parameter the current time in milliseconds. The library uses this information for timed events such as calculating how long a sensor has been in detect.

After calling “qt_measure_sensors()”, the host application can check the state of the enabled sensors by reading the “qt_touch_status” variable (see Section 3.4 on page 3-3).

The host application should call “qt_measure_sensors()” on a regular basis so that any user touches are promptly detected, and any environmental changes are drifted out.

3.10.2 Additional Sensing Commands

In addition to the “qt_init_sensing()” and “qt_measure_sensors()” functions, there are two additional touch sensing commands available to the host application. These are the “qt_calibrate_sensing()” and “qt_reset_sensing()” functions.

3.10.3 qt_init_sensing()

This function initializes touch sensing.

```
void qt_init_sensing( void );
```

Any sensors required must be enabled (using the appropriate “qt_enable_xxx()” function) before calling this function.

This function calculates internal library variables and configures the touch channels, and must be called before calling “qt_measure_sensors()”.

3.10.4 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, changes in slider position, etc.

```
void qt_measure_sensors( uint16_t current_time_ms );
```

The parameter is as follows:

- current_time_ms = the current time, in ms

The current state of all enabled sensors is reported in the “qt_touch_status” struct.

Before calling this function, one or more sensors must have been enabled (using the appropriate “qt_enable_xxx()” function), and “qt_init_sensing()” must have been called.

3.10.5 qt_calibrate_sensing()

This function forces a recalibration of all enabled sensors. This may be useful if, for example, it is desired to globally recalibrate all sensors on a change in application operating mode.

```
void qt_calibrate_sensing( void );
```

3.10.6 qt_reset_sensing()

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

This may be useful if it is desired to dynamically reconfigure sensing. After calling this function, any required sensors must be re-enabled, and “qt_init_sensing()” must be called before “qt_measure_sensors()” is called again.

3.11 Example Host Application

The following code sample shows how a host application could configure and use the Atmel Touch Library.

```
/* flag set by timer ISR when it's time to measure touch */
static uint8_t time_to_measure_touch = 0u;

/* current time, set by timer ISR */
uint16_t current_time_ms = 0;

void main( void )
{
   /* initialise host app, pins, watchdog, etc */
   init_system();

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

   /* 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 );

   /* initialise touch sensing */
   qt_init_sensing();

   /* configure timer ISR to fire regularly */
   init_timer_isr();

   /* enable interrupts */
   __enable_interrupt();

   /* loop forever */
   for( ; ; )
   {
      /* test flag: is it time to measure touch? */
      if( time_to_measure_touch )
      {
         /* clear flag: it's time to measure touch */
         time_to_measure_touch = 0;
```

```
            /* measure touch sensors */
            qt_measure_sensors( current_time_ms );
        }

        /* host application code goes here */
    }
}

/* timer ISR: fires every MEASUREMENT_PERIOD_MS */
void timer_isr( void )
{
    /* set flag: it's time to measure touch */
    time_to_measure_touch = 1u;

    /* update the current time */
    current_time_ms += MEASUREMENT_PERIOD_MS;
}
```

Section 4

Library Variants

4.1 Introduction

Variants of the Atmel Touch Library run on a range of Atmel chips. This section lists the variants available, along with their resource usage and a note of any limitations on the chip operating conditions.

The library uses chip resources. This has implications for the resources available to the host application. The actual resource usage depends on the chip, the library facilities, the sensing technology, and the number of sense channels. In general a library needs some GPIO pins, some code space in the chip flash, some RAM for storing channel and sensor data, some register variables, and will have a minimum stack size requirement.

The library is linked into host applications running on Atmel chips. The host application is subject to the chip operating conditions (voltage, temperature, and so on) listed in the datasheet for the device. The following sections list any known restrictions on the operating conditions for touch sensing to work correctly.

4.2 ATmega88 Microcontroller

4.2.1 Atmel Touch Library m88_8qt

Atmel Touch Library m88_8qt runs on an ATmega88. It is based on QTouch™ technology, and provides eight sensing channels. These channels can be configured as keys, rotors, or sliders, depending on the library variant. Versions of the library are also available providing between one and fity cycle charge pulses.

4.2.2 ATmega88 Library Variants Supporting Keys Only

Table 4-1. Information on ATmega88 Library Supporting Keys Only

Specification	Details
Library File	M88_8qt_c1/2/3/4/5/10/25/50_k.r90
Version	1.2
Header File	touchstone_k.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys
Sensor Configurations	Keys can be on any channel

Table 4-2. Resource Usage for ATmega88 Library Supporting Keys Only

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	2071 bytes
RAM	122 bytes
Register Variables	2 (R14 and R15)
CSTACK	16 bytes
RSTACK	10 bytes

4.2.3 ATmega88 Library Variants Supporting Keys, Rotors and Sliders

Table 4-3. Information on ATmega88 Library Supporting Keys, Rotors and Sliders

Specification	Details
Library File	M88_8qt_c1/2/3/4/5/10/25/50_krs.r90
Version	1.2
Header File	touchstone_krs.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys, rotors and sliders
Sensor Configurations	Keys can be on any channel Rotors can be on channels 0 to 2, or 3 to 5 Sliders can be on channels 0 to 2, or 3 to 5

Table 4-4. Resource Usage for ATmega88 Library Supporting Keys, Rotors and Sliders

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	3263 bytes
RAM	148 bytes
Register Variables	2 (R14 and R15)
CSTACK	28 bytes
RSTACK	10 bytes

4.3 ATtiny88 Microcontroller

4.3.1 Atmel Touch Library t88_8qm

Atmel Touch Library t88_8qm runs on an ATtiny88. It is based on QMatrix™ technology, and provides eight sensing channels. These channels can be configured as keys, rotors, or sliders, depending on the library variant. Versions of the library are also available providing between one and fifty cycle dwell times.

4.3.2 ATtiny88 Library Variants Supporting Keys Only

Table 4-5. Information on ATtiny88 Library Supporting Keys Only

Specification	Details
Library File	t88_8qm_d1/2/3/4/5/10/25/50_k.r90
Version	1.2
Header File	touchstone_k.h
Technology	QMatrix (1/2/3/4/5/10/25/50 cycle dwell time)
Number of Channels	8
Sensor Types	Keys
Sensor Configurations	Keys can be on any channel

Table 4-6. Resource Usage for ATtiny88 Library Supporting Keys Only

Specification	Details
GPIO Pins	Each channel is specified as the intersection of an X/Y line. Channels 0 to 3 are on X0Y0, X1Y0, X2Y0, and X3Y0 respectively. Channels 4 to 7 are on X0Y1, X1Y1, X2Y1, and X3Y1 respectively. All channels require the use of pin PD7. X0 to X3 use pins PB0 to PB3. Y0 uses pins PC0 and PD0, and Y1 uses pins PC1 and PD1. Pins for unused channels can be used by the host application.
ROM	2205 bytes
RAM	131 bytes
Register Variables	6 (R10 to R15)
CSTACK	16 bytes
RSTACK	10 bytes

4.3.3 ATtiny88 Library Variants Supporting Keys, Rotors and Sliders

Table 4-7. Information on ATtiny88 Library Supporting Keys, Rotors and Sliders

Specification	Details
Library File	t88_8qm_c1/2/3/4/5/10/25/50_krs.r90
Version	1.2
Header File	touchstone_krs.h
Technology	QMatrix (1/2/3/4/5/10/25/50 cycle dwell time)

Table 4-7. Information on ATtiny88 Library Supporting Keys, Rotors and Sliders

Specification	Details
Number of Channels	8
Sensor Types	Keys, rotors and sliders
Sensor Configurations	Keys can be on any channel. Rotors can be on channels 0 to 2, 0 to 3, 4 to 6, 4 to 7, or 5 to 7. Sliders can be on channels 0 to 2, 0 to 3, 4 to 6, 4 to 7, or 5 to 7.

Table 4-8. Resource Usage for ATtiny88 Library Supporting Keys, Rotors and Sliders

Specification	Details
GPIO Pins	Each channel is specified as the intersection of an X/Y line. Channels 0 to 3 are on X0Y0, X1Y0, X2Y0, and X3Y0 respectively. Channels 4 to 7 are on X0Y1, X1Y1, X2Y1, and X3Y1 respectively. All channels require the use of pin PD7. X0 to X3 use pins PB0 to PB3. Y0 uses pins PC0 and PD0, and Y1 uses pins PC1 and PD1. Pins for unused channels can be used by the host application.
ROM	3549 bytes
RAM	175 bytes
Register Variables	6 (R10 to R15)
CSTACK	31 bytes
RSTACK	14 bytes

4.4 ATmega88PA Microcontroller

4.4.1 Atmel Touch Library m88PA_8qt

Atmel Touch Library m88PA_8qt runs on an ATmega88PA. It is based on QTouch™ technology, and provides eight sensing channels. These channels can be configured as keys, rotors, or sliders, depending on the library variant. Versions of the library are also available providing between one and fity cycle charge pulses.

4.4.2 ATmega88PA Library Variants Supporting Keys Only

Table 4-9. Information on ATmega88PA Library Supporting Keys Only

Specification	Details
Library File	M88PA_8qt_c1/2/3/4/5/10/25/50_k.r90
Version	1.2
Header File	touchstone_k.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys
Sensor Configurations	Keys can be on any channel

Table 4-10. Resource Usage for ATmega88PA Library Supporting Keys Only

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	2071 bytes
RAM	122 bytes
Register Variables	2 (R14 and R15)
CSTACK	16 bytes
RSTACK	10 bytes

4.4.3 ATmega88PA Library Variants Supporting Keys, Rotors and Sliders

Table 4-11. Information on ATmega88PA Library Supporting Keys, Rotors and Sliders

Specification	Details
Library File	M88PA_8qt_c1/2/3/4/5/10/25/50_krs.r90
Version	1.2
Header File	touchstone_krs.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys, rotors and sliders
Sensor Configurations	Keys can be on any channel Rotors can be on channels 0 to 2, or 3 to 5 Sliders can be on channels 0 to 2, or 3 to 5

Table 4-12. Resource Usage for ATmega88PA Library Supporting Keys, Rotors and Sliders

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	3263 bytes
RAM	148 bytes
Register Variables	2 (R14 and R15)
CSTACK	28 bytes
RSTACK	10 bytes

4.5 ATmega168P Microcontroller

4.5.1 Atmel Touch Library m168P_8qt

Atmel Touch Library m168_8qt runs on an ATmega168P. It is based on QTouch™ technology, and provides eight sensing channels. These channels can be configured as keys, rotors, or sliders, depending on the library variant. Versions of the library are also available providing between one and fity cycle charge pulses.

4.5.2 ATmega168P Library Variants Supporting Keys Only

Table 4-13. Information on ATmega168P Library Supporting Keys Only

Specification	Details
Library File	M168P_8qt_c1/2/3/4/5/10/25/50_k.r90
Version	1.2
Header File	touchstone_k.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys
Sensor Configurations	Keys can be on any channel

Table 4-14. Resource Usage for ATmega168P Library Supporting Keys Only

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	2693 bytes
RAM	257 bytes
Register Variables	2 (R14 and R15)
CSTACK	16 bytes
RSTACK	10 bytes

4.5.3 ATmega168P Library Variants Supporting Keys, Rotors and Sliders

Table 4-15. Information on ATmega168P Library Supporting Keys, Rotors and Sliders

Specification	Details
Library File	M168P_8qt_c1/2/3/4/5/10/25/50_krs.r90
Version	1.2
Header File	touchstone_krs.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)

Table 4-15. Information on ATmega168P Library Supporting Keys, Rotors and Sliders

Specification	Details
Number of Channels	8
Sensor Types	Keys, rotors and sliders
Sensor Configurations	Keys can be on any channel Rotors can be on channels 0 to 2, or 3 to 5 Sliders can be on channels 0 to 2, or 3 to 5

Table 4-16. Resource Usage for ATmega168P Library Supporting Keys, Rotors and Sliders

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	3959 bytes
RAM	283 bytes
Register Variables	2 (R14 and R15)
CSTACK	28 bytes
RSTACK	10 bytes

4.6 ATmega328P Microcontroller

4.6.1 Atmel Touch Library m328P_8qt

Atmel Touch Library m328P_8qt runs on an ATmega328P. It is based on QTouch™ technology, and provides eight sensing channels. These channels can be configured as keys, rotors, or sliders, depending on the library variant. Versions of the library are also available providing between one and fity cycle charge pulses.

4.6.2 ATmega328P Library Variants Supporting Keys Only

Table 4-17. Information on ATmega328P Library Supporting Keys Only

Specification	Details
Library File	M328P_8qt_c1/2/3/4/5/10/25/50_k.r90
Version	1.2
Header File	touchstone_k.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys
Sensor Configurations	Keys can be on any channel

Table 4-18. Resource Usage for ATmega328P Library Supporting Keys Only

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	2693 bytes
RAM	257 bytes
Register Variables	2 (R14 and R15)
CSTACK	16 bytes
RSTACK	10 bytes

4.6.3 ATmega328P Library Variants Supporting Keys, Rotors and Sliders

Table 4-19. Information on ATmega328P Library Supporting Keys, Rotors and Sliders

Specification	Details
Library File	M328P_8qt_c1/2/3/4/5/10/25/50_krs.r90
Version	1.2
Header File	touchstone_krs.h
Technology	QTouch (1/2/3/4/5/10/25/50 cycle charge time)
Number of Channels	8
Sensor Types	Keys, rotors and sliders
Sensor Configurations	Keys can be on any channel Rotors can be on channels 0 to 2, or 3 to 5 Sliders can be on channels 0 to 2, or 3 to 5

Table 4-20. Resource Usage for ATmega328P Library Supporting Keys, Rotors and Sliders

Specification	Details
GPIO Pins	Each channel uses two pins. Channel “x” is on pins PBx and PDx (for example, channel 3 is on pins PB3 and PD3). Pins for unused channels can be used by the host application.
ROM	3959 bytes
RAM	283 bytes
Register Variables	2 (R14 and R15)
CSTACK	28 bytes
RSTACK	10 bytes

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
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature