# Atmel QTouch Library

## User Guide

Rev. 8207C-AT42-06/09

# 1 Introduction

The Atmel's QTouch™ Library provides support for both IAR™ and GCC compilers for capacitive keys, rotors, and sliders in Atmel® firmware applications.

It is provided as a library file and C header file for linking with user code. This guide explains the library operation, and how it is used in firmware applications.

This guide should be read in conjunction with the following documents:

- Atmel's QTouch Library Overview
- Atmel's QTouch Library Datasheet

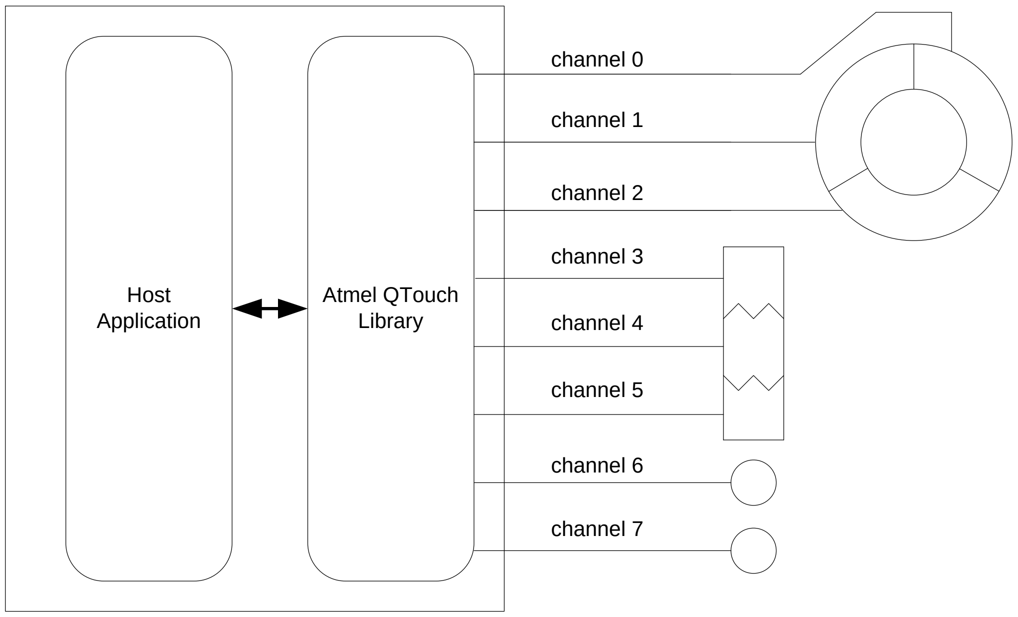
# 2 Adding Touch Sensing to an Application

## 2.1 Introduction

The Atmel QTouch 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 how the channels are configured to the microcontroller based on Atmel's touch technology (QTouch and QMatrix™), in which the Atmel® QTouch Library has been added to a host application running on an an Atmel AVR microcontroller. 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 2-1** Typical of interfacing the Atmel's Touch library with the host application.



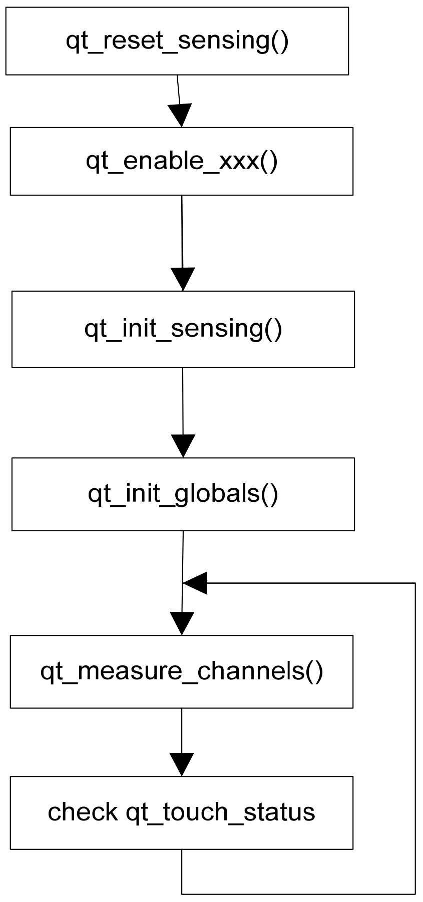
The Atmel QTouch Library only runs when called from the host application. It uses no timers, interrupts, or other chip resources apart from ROM, RAM, some register variables, and GPIO. It only supports touch sensing; the host application must provide any other functionality required, such as reading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

## 2.2 Host Application Program Flow

Adding Touch Sensing to an Application
The general flowchart for use of the Atmel QTouch Library is shown in Figure 1-2. The steps are as follows:

- 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.
- The host application calls “qt_enable_key( )”, “qt_enable_rotor( )” and/or “qt_enable_slider()” as required to configure the touch sensors.
- The host application calls “qt_init_globals()” to intialize the global threshold parameters used by the library.If the user wishes to change these threshold parameters he can edit the qt_init_globals() and supply the parameters.
- The host application calls “qt_init_sensing( )” to initialize the library.
- 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 2-2**. Flowchart for Using the Atmel QTouch Library in a Host Application


qt_reset_sensing()
qt_enable_xxx()
qt_init_sensing()
qt_init_globals()
qt_measure_channels()
check qt_touch_status

## 2.3 Host Application Requirements

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

- One has to have the data sheet downloaded and ready before starting to use the main.c and touch_api.h file
    - Check if the debug ports and pins used are valid for the selected device.
    - Check that the CLKPR register is available for the selected device. If not remove the CLKPR statements.
    - MCUCR register is available and if so disable pullups
    - Check if the timer registers and bitfields used are correct and change them if necessary.
- It must track the current time.  
  This information is passed to the code library as an argument to the function qt_measure_sensors()". 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 host application 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.  
  The library stack requirements for different configurations are shown in Table 1-1:

**Table 2-1** Library stack requirements

| **Configuration** | **CSTACK size** | **RSTACK size** |
|---|---|---|
| ONLY KEYS | 0x18 | 0x10 |
| KEYS/ROTOR/SLIDER | 0x2A | 0x18 |

### 2.3.1 Configuration selection:

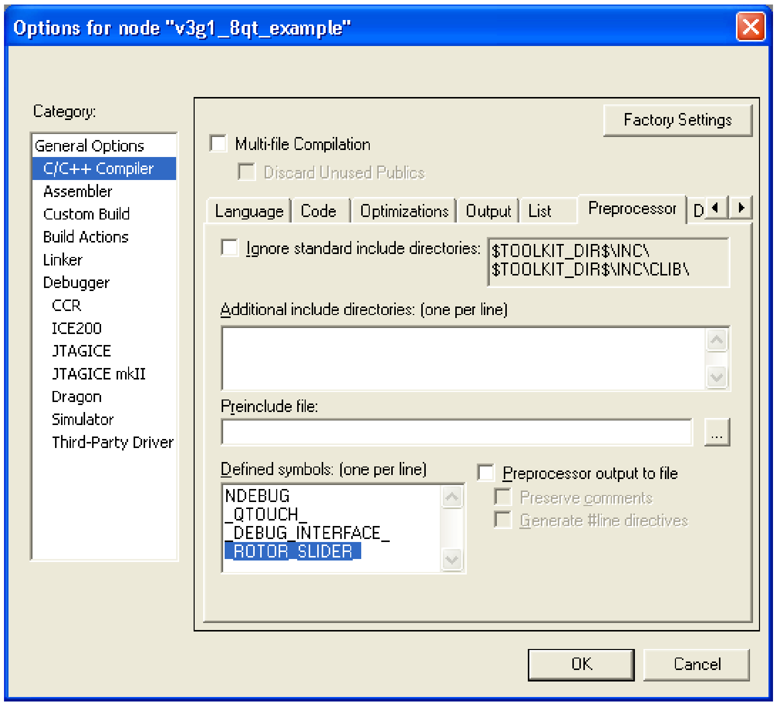
Based on the requirement, the user can select the configuration by editing the project options and thereby adding the pre-processor directive _ROTOR_SLIDER_

#### *2.3.1.1 For KEYS/ROTORS/SLIDERS configuration:*

If the user wishes to use KEYS/ROTORS/SLIDERS configuration, then the user has to follow these steps:

- ***IAR-EWAVR:***  
  Go to project options in IAR work bench -> C/C++ compiler -> Preprocessor Tab

Then Add the Directive _ROTOR_SLIDER_ in the space provided for the directives before building the project.

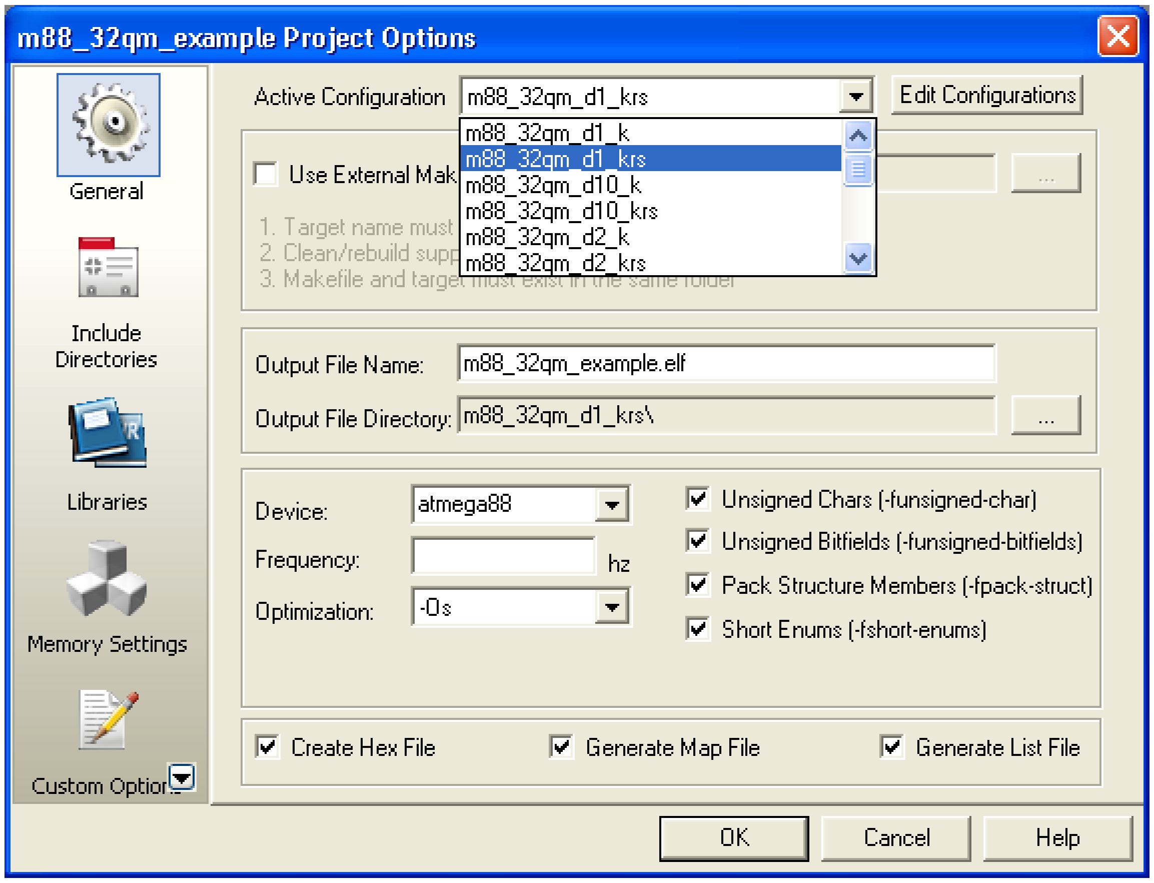


```
$TOOLKIT_DIR$\INC\
$TOOLKIT_DIR$\INC\CLIB\
```

Defined symbols:

```
NDEBUG
_QTOUCH_
_DEBUG_INTERFACE_
_ROTOR_SLIDER_
```

- ***WINAVR- GCC:***
  Go to Project configuration Options -> General Options->Active Configuration
  ***QMatrix:***
  Select the type of configuration that the user wishes to use from the Drop down of Active configuration Tab.



m88_32qm_example Project Options

Active Configuration: m88_32qm_d1_krs

- m88_32qm_d1_k
- m88_32qm_d1_krs
- m88_32qm_d10_k
- m88_32qm_d10_krs
- m88_32qm_d2_k
- m88_32qm_d2_krs

Output File Name: m88_32qm_example.elf

Output File Directory: m88_32qm_d1_krs\

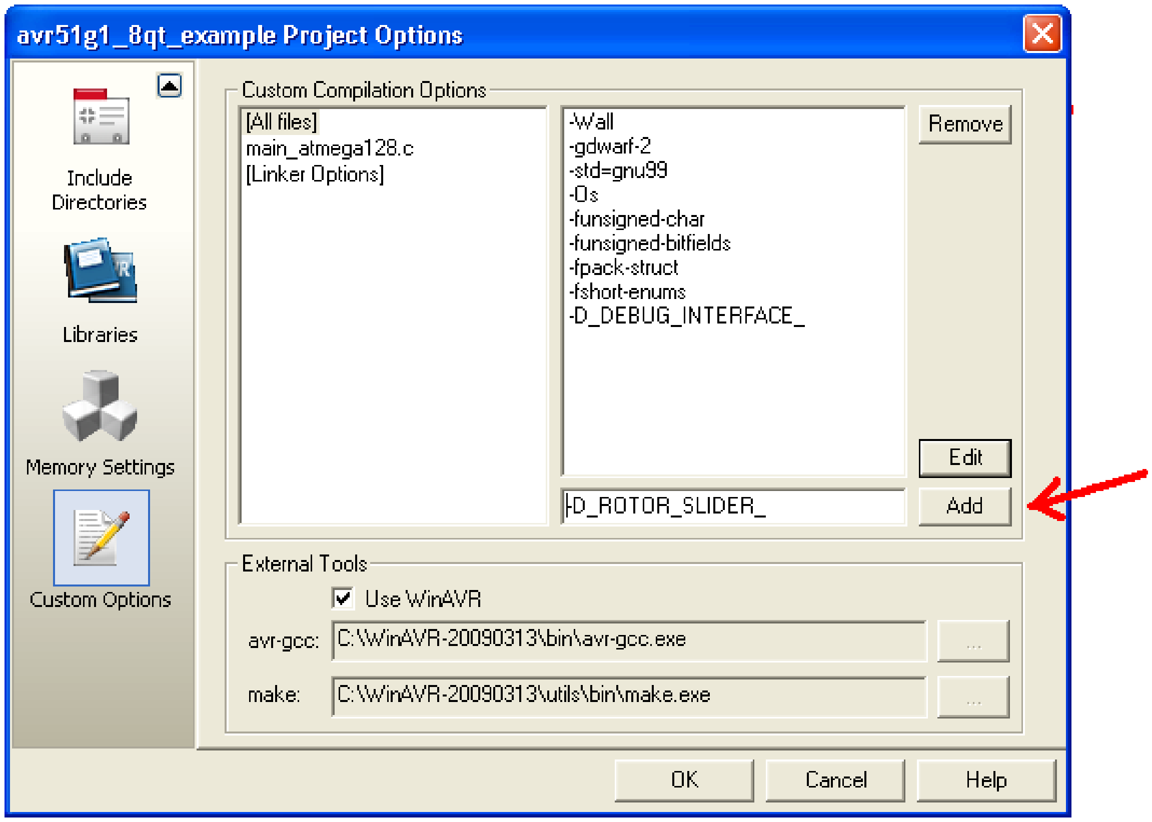
Device: atmega88

Frequency: hz

Optimization: -Os

- [x] Unsigned Chars (-funsigned-char)
- [x] Unsigned Bitfields (-funsigned-bitfields)
- [x] Pack Structure Members (-fpack-struct)
- [x] Short Enums (-fshort-enums)
- [x] Create Hex File
- [x] Generate Map File
- [x] Generate List File

***QTouch:***
Then Add the Directive –D _ROTOR_SLIDER_ in the space provided for the directives before building the project.


avr51g1_8qt_example Project Options

Custom Compilation Options

[All files]
main_atmega128.c
[Linker Options]

-Wall
-gdwarf-2
-std=gnu99
-Os
-funsigned-char
-funsigned-bitfields
-fpack-struct
-fshort-enums
-D_DEBUG_INTERFACE_

-D_ROTOR_SLIDER_

External Tools

☑ Use WinAVR

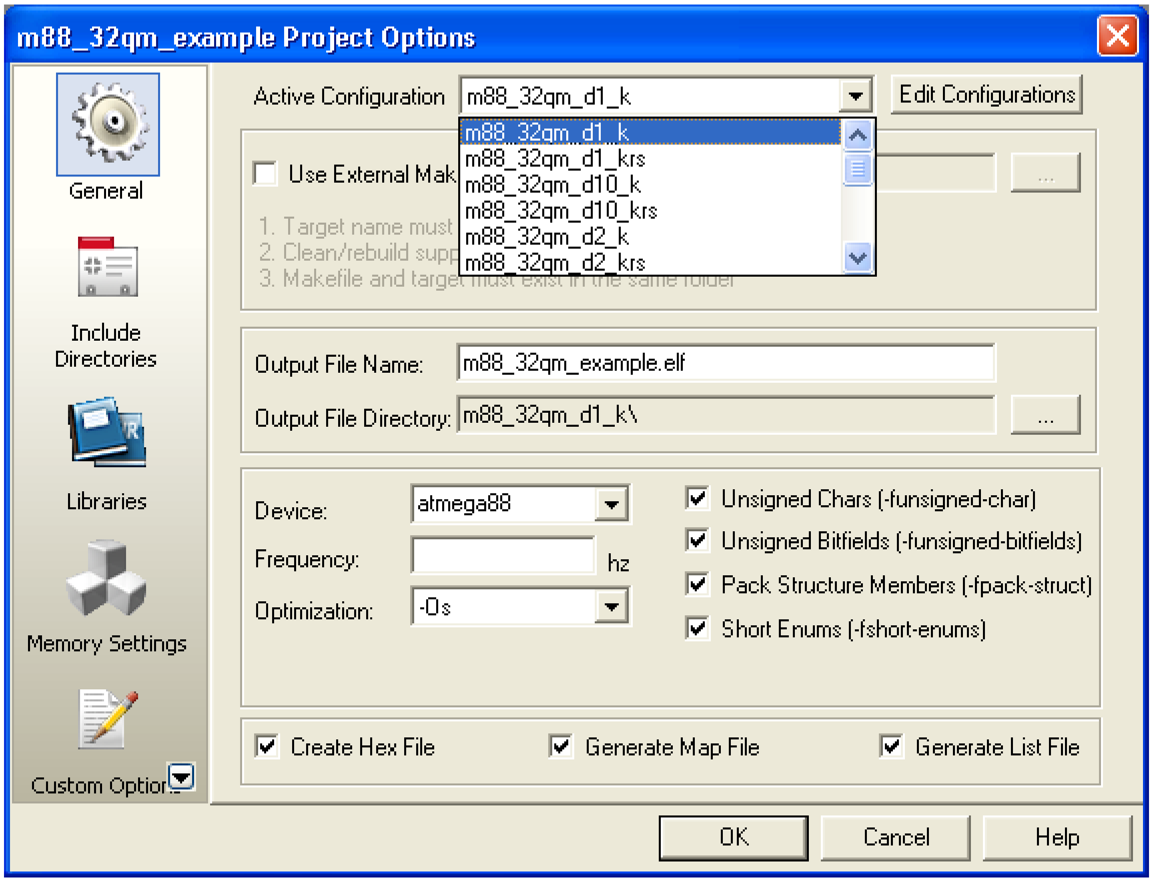
avr-gcc: C:\WinAVR-20090313\bin\avr-gcc.exe

make: C:\WinAVR-20090313\utils\bin\make.exe

### 2.3.1.2 *For ONLY KEYS configuration:*

If the user wishes to use ONLY KEYS configuration, then the user has to follow these steps:

- ***IAR-EWAVR:***
  Go to project options in IAR work bench -> C/C++ compiler -> Preprocessor Tab
  Then remove the Directive _ROTOR_SLIDER_ (if exists) in the space provided for the directives before building the project.
- ***WINAVR- GCC:***
  ***QMatrix:***
  Go to Project configuration Options -> General Options->
  Select the type of configuration that the user wishes to use from the Drop down of Active configuration Tab.



m88_32qm_example Project Options

Active Configuration: m88_32qm_d1_k

m88_32qm_d1_k
m88_32qm_d1_krs
m88_32qm_d10_k
m88_32qm_d10_krs
m88_32qm_d2_k
m88_32qm_d2_krs

Edit Configurations

☐ Use External Mak

1. Target name must
2. Clean/rebuild supp
3. Makefile and target must exist in the same folder

Output File Name: m88_32qm_example.elf

Output File Directory: m88_32qm_d1_k\

Device: atmega88

Frequency: hz

Optimization: -Os

☑ Unsigned Chars (-funsigned-char)

☑ Unsigned Bitfields (-funsigned-bitfields)

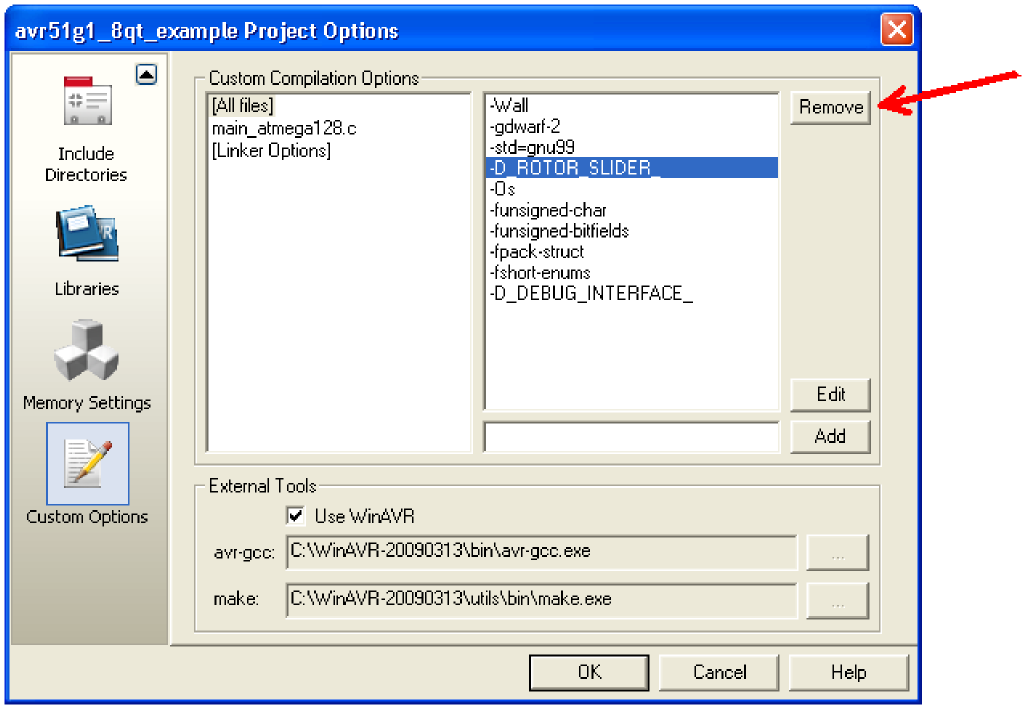
☑ Pack Structure Members (-fpack-struct)

☑ Short Enums (-fshort-enums)

☑ Create Hex File ☑ Generate Map File ☑ Generate List File

***QTouch:***

Go to Project configuration Options -> Custom Options->
Then remove the Directive –D_ROTOR_SLIDER_ in the space provided for the directives before building the project.


avr51g1_8qt_example Project Options

Custom Compilation Options

[All files]
main_atmega128.c
[Linker Options]

```
-Wall
-gdwarf-2
-std=gnu99
-D_ROTOR_SLIDER_
-Os
-funsigned-char
-funsigned-bitfields
-fpack-struct
-fshort-enums
-D_DEBUG_INTERFACE_
```

External Tools

☑ Use WinAVR

avr-gcc: C:\WinAVR-20090313\bin\avr-gcc.exe

make: C:\WinAVR-20090313\utils\bin\make.exe

## 2.4 Example Host Application

The Atmel QTouch Library evaluation kit includes a complete example host application. This comprises an IAR project or AVR Studio® GCC 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.

Each version of the library is provided with the example host application project file in binary to help the user start up with the touch library. These example project files can be found from the library variant section described later in this document.

# 3 Using the Atmel's QTouch Library

## 3.1 Sensing Channels

### 3.1.1 Disabled Channels

All sensing channels are disabled by default.
The host application can use unused sensing pins as GPIO.

### 3.1.2 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.

So, the order in which the user enables the sensors is the order in which  the sensors are numbered. Depending on his requirement, the user can enable the sensors configured to corresponding channels.

In the  "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.

## 3.2 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.
Please note that none of the API functions should be called from a user interrupt.

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

**Table 3-1**. Frequency of operatoion

| Frequency of Microcontroller (MHz)) | MCU Cycle time (us) | Suitable Charge Cycle times (or) Suitable Dwell Cycle times (us) |
|---|---|---|
| 1 | 1 | 1~2 cycles (1us to 2us) |
| 2 | 0.5 | 1~5 cycles (0.5us to 2.5us) |
| 4 | 0.25 | 1~10 cycles (0.25us to 2.5us) |
| 8 | 0.125 | 1~10 cycles (0.125us to 1.25us) |
| 10 | 0.1 | 2~25 cycles (0.2us to 2.5us) |
| 16 | 0.0625 | 2~25 cycles (0.125us to 1.5625us) |
| 20 | 0.05 | 3~50 cycles (0.15us to 2.5us) |

## 3.4 Sensor Measurements

### 3.4.1 Avoiding Cross-talk

In Atmel QTouch 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.
So, it is recommended that the channels are configured accordingly.

### 3.4.2 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.

### 3.4.3 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.

### 3.4.4 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.

### 3.4.5 Filtering signal measurements

The Atmel QTouch 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.

#### *3.4.5.1 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.

### *3.4.5.2 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 QTouch Library are now the mean of the last four actual signal values.

#### 3.4.5.3 *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 QTouch Library are now the median of the last three actual signal values.

# 4 Application Programming Interface

## 4.1 Introduction

This section defines the API that the Atmel QTouch 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).

## 4.2 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 4-1**. 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. |

## 4.3 Type Definitions

### 4.3.1 Typedefs

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

**Table 4-2** 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 |

### 4.3.2 Enumerations

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

**Table 4-3**. Enumerations

| Name | Values | Notes |
|---|---|---|
| Sensor_t | SENSOR_TYPE_UNASSIGNED<br>SENSOR_TYPE_KEY,<br>SENSOR_TYPE_ROTOR,<br>SENSOR_TYPE_SLIDER | Type of sensor that needs to be configured |
| aks_group_t | NO_AKS_GROUP<br>AKS_GROUP_1<br>AKS_GROUP_2<br>AKS_GROUP_3<br>AKS_GROUP_4<br>AKS_GROUP_5<br>AKS_GROUP_6<br>AKS_GROUP_7 | Which AKS™ group, if any, a sensor is in.<br>NO_AKS_GROUP = sensor is not in an AKS group, and cannot be suppressed.<br>AKS_GROUP_x = sensor is in AKS group x. |
| channel_t | CHANNEL_0<br>CHANNEL_1<br>CHANNEL_2<br>CHANNEL_3<br>CHANNEL_4<br>CHANNEL_5<br>CHANNEL_6<br>CHANNEL_7<br>…<br>(Number_of channels – 1) | The channel(s) in a sensor. |
| Hysteresis_t | HYST_50<br>HYST_25<br>HYST_12_5<br>HYST_6_25 | A sensor detection hysteresis value. This is expressed as a percentage of the sensor detection threshold.<br>HYST_x = hysteresis value is x percent of detection threshold value (rounded down).<br>Note that a minimum value of 2 is used as a hard limit. Example: if detection threshold = 20, then:<br>HYST_50 = 10 (50 percent of 20)<br>HYST_25 = 5 (25 percent of 20)<br>HYST_12_5 = 2 (12.5 percent of 20)<br>HYST_6_25 = 2 (6.25 percent of 20 = 1, but set to the hard limit of 2) |
| recal_threshold_t | RECAL_100<br>RECAL_50<br>RECAL_25<br>RECAL_12_5<br>RECAL_6_25 | A sensor recalibration threshold. This is expressed as a percentage of the sensor detection threshold.<br>RECAL_x = recalibration threshold is x percent of detection threshold value (rounded down).<br>Note: a minimum value of 4 is used.<br>Example: if detection threshold = 40, then:<br>RECAL_100 = 40 (100 percent of 40)<br>RECAL_50 = 20 (50 percent of 40)<br>RECAL_25 = 10 (25 percent of 40)<br>RECAL_12_5 = 5 (12.5 percent of 40)<br>RECAL_6_25 = 4 (6.25 percent of 40 = 2, but value is limited to 4) |
| resolution_t | RES_1_BIT<br>RES_2_BIT<br>RES_3_BIT<br>RES_4_BIT<br>RES_5_BIT<br>RES_6_BIT<br>RES_7_BIT<br>RES_8_BIT | For rotors and sliders, the resolution of the reported angle or position.<br>RES_x_BIT = rotor/slider reports x-bit values.<br>Example: if slider resolution is RES_7_BIT, then reported positions are in the range 0..127. |

### 4.3.3 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 4-4**. Struct

| Struct | Field | Type | Notes |
|---|---|---|---|
| qt_touch_status_t | sensor_states | uint8_t | The state (on/off) of the library sensors. Bit “n” = state of sensor “n”: 0 = not in detect, 1 = in detect. |
| | Rotor_slider_values[] | uint16_t | Rotor angles or slider positions. These values are valid when “sensor_states” shows that the corresponding rotor or slider sensor is in detect. |
| Board_info_t | qt_max_num_rotors_sliders_board_id | uint8_t | Maximum number of rotors and sliders and board id sent as debug information. |
| | Qt_num_channels | uint8_t | Number of channels that the library supports sent as debug information |
| qt_touch_lib_config_data_t | qt_recal_threshold | uint8_t | Sensor detect integration (DI) limit. Default value: 4 |
| | qt_di | 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_drift_hold_time | 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_max_on_duration | uint8_t | Sensor negative drift rate in units of 200 ms. Default value: 20 (20 x 200 ms = 4s per LSB) |
| | qt_neg_drift_rate | uint8_t | Sensor positive drift rate in units of 200 ms. Default value: 5 (5 x 200 ms = 1s per LSB) |
| | qt_pos_drift_rate | recal_threshold_t | Sensor recalibration threshold. Default: RECAL_50 (recalibration threshold = 50 percent of detection threshold |
| qt_touch_lib_measure_data_t | 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 sensor states as described above in this table. |

## 4.4 Per-channel Touch Sensing Configuration

In library variants based on the QMatrix technology, the data array listed in Table 4-5 is available to the host application.

**Table 4-5**. Struct

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

## 4.5 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 4-6**. 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. |

The delta value for a given sensor may be obtaing by calling `qt_get_sensor_delta( sensor_number)`

## 4.6 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 filter the measured signal values according to his needs.

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

## 4.7 Configuring Sensors

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

### 4.7.1 Configuration Functions

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

**Table 4-7**. Functions

| Function | Notes |
|---|---|
| qt_enable_key() | Enable a key sensor. |
| Qt_enable_rotor() | Enable a rotor sensor. |
| Qt_enable_slider() | Enable a slider sensor. |

### 4.7.2 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".

### 4.7.3 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.

### 4.7.4 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:

i) from_channel = the first channel in the slider sensor.
ii) to_channel = the last channel in the slider sensor
iii) aks_group = which AKS group (if any) the sensor is in
iv) detect_threshold = the sensor detection threshold
v) detect_hysteresis = the sensor detection hysteresis value
vi) position_resolution = the resolution of the reported position value
vii) 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.

## 4.8 Measuring and Checking the Touch Status

### 4.8.1 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.

### 4.8.2 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.

### 4.8.3 qt_init_globals()

This function initializes the global threshold parameters
```
void qt_init_globals(void);
```

The threshold parameters are globally available and to the user to configure them. The user is provided with this function where he can change the values of the parameters according to the needs.

### 4.8.4 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()".

### 4.8.5 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.

### 4.8.6 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 );

### 4.8.7 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.

### 4.8.8 qt_get_sensor_delta (sensor_number)

This function returns the delta value for a given channel

```
int16_t qt_get_sensor_delta( uint8_t sensor )
```

## 4.9 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;
}
```

# 5 Touch Data Debug interface

The user is provided with a main.c file and a header file (touch_lib_api.h). In main.c file the user is provided with a default debug protocol which can be used for communication with the PC through the USB bridge.

The user can call the function report_debug_data(void) for using the debug protocol to send the debug data from the microcontroller to the AVR QTouch Studio.

**Void report_debug_data(void);**

This function bit bangs the data available from the library through the USB bridge to tool called Hawkeye which can be analyzed.

The data send by this debug function include

board_info_t – which captures the information related to the board used.

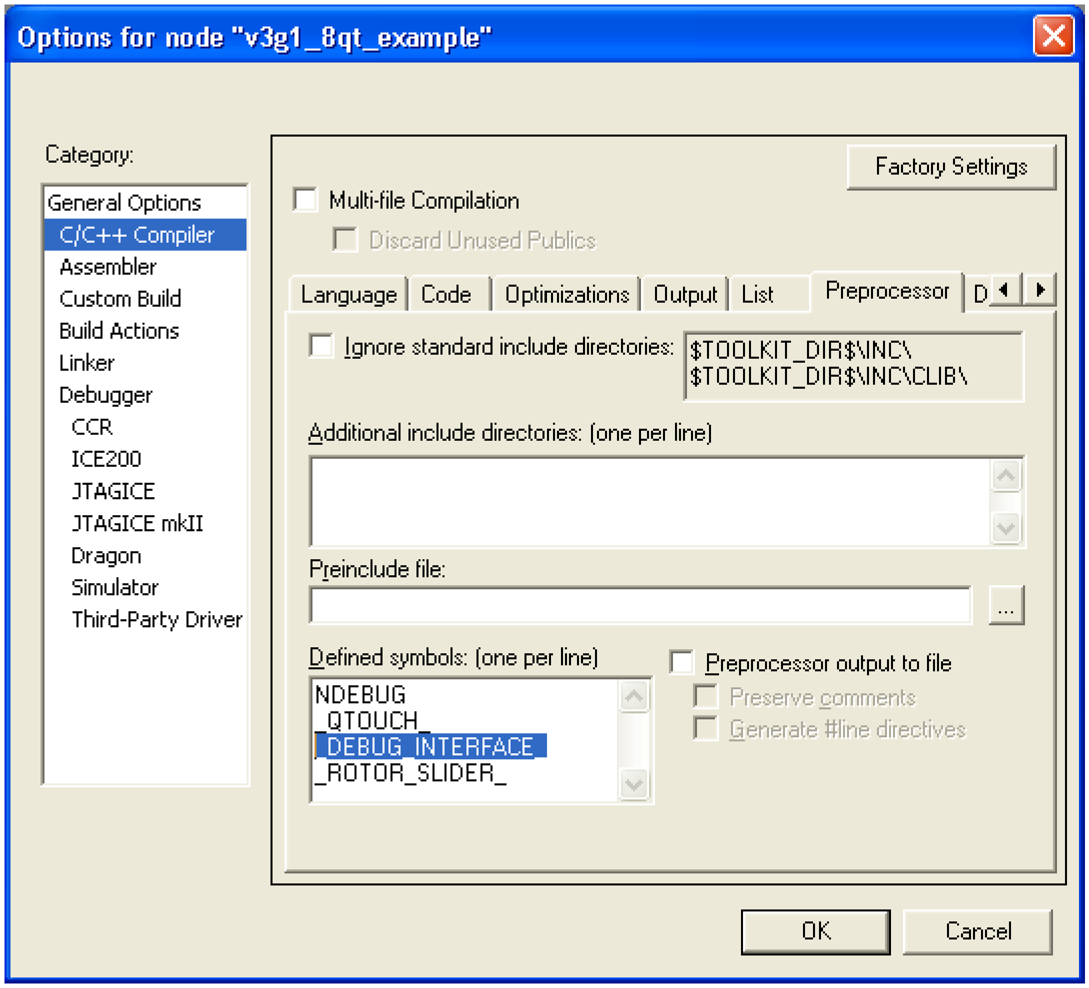
- Type of board used (0 if TS2080A and 1 if TS2080B, list will be expanded when new kits are released)
- Maximum number of rotors and sliders that the library supports

The user can change the id for the board according to the need by changing the values in the structure.
Channel_signals – The signal data related to each channel
Channel_reference – The reference data related to each channel
Sensor_deltas – The difference when the sensor is detected.
Qt_touch_status - The states of each sensor configured.
Sensor_config – debug information related to each sensor.

However based on the user's requirement the same data can be transferred through the debug protocol that the user wishes to use.

For using the default touch data debug interface the user has to enable the directive _DEBUG_INTERFACE_ in the project configuration options as below:

- ***IAR-EWAVR:***
  Go to project options in IAR work bench -> C/C++ compiler -> Preprocessor Tab
  Then Add the Directive _DEBUG_INTERFACE_ in the space provided for the directives before building the project.


Options for node "v3g1_8qt_example"

Category: C/C++ Compiler

Ignore standard include directories: $TOOLKIT_DIR$\INC\ $TOOLKIT_DIR$\INC\CLIB\

Additional include directories: (one per line)

Preinclude file:

Defined symbols: (one per line)

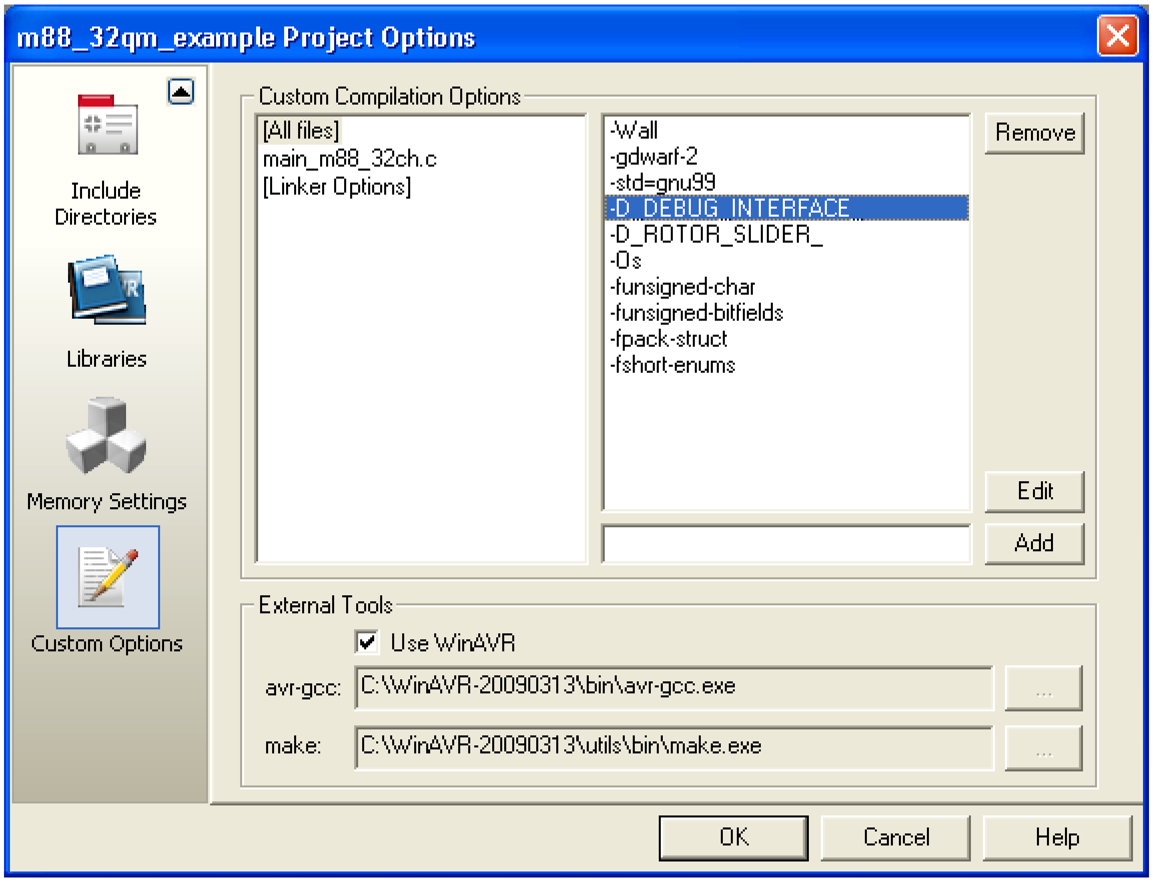
```
NDEBUG
_QTOUCH_
_DEBUG_INTERFACE_
_ROTOR_SLIDER_
```

Preprocessor output to file

Preserve comments

Generate #line directives

- ***WINAVR- GCC:***
  Go to Project configuration Options -> Custom Options->
  Then Add the Directive –D_DEBUG_INTERFACE_ in the space provided for the directives before building the project.



Custom Compilation Options

[All files]
main_m88_32ch.c
[Linker Options]

-Wall
-gdwarf-2
-std=gnu99
-D_DEBUG_INTERFACE_
-D_ROTOR_SLIDER_
-Os
-funsigned-char
-funsigned-bitfields
-fpack-struct
-fshort-enums

External Tools

☑ Use WinAVR

avr-gcc: C:\WinAVR-20090313\bin\avr-gcc.exe

make: C:\WinAVR-20090313\utils\bin\make.exe

# 6 Library variants

## 6.1 QTouch Acquisition:

### 6.1.1 Introduction

Variants of the Atmel QTouch 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.

### 6.1.2 QTouch acquisition method library

**List of libraries included:**

```
1  libavr25g1_8qt_k.a
2  libavr25g1_8qt_krs.a
3  libavr25g2_8qt_k.a
4  libavr25g2_8qt_krs.a
5  libavr4g1_8qt_k.a
6  libavr4g1_8qt_krs.a
7  libavr4g2_8qt_k.a
8  libavr4g2_8qt_krs.a
9  libavr51g1_8qt_k.a
10 libavr51g1_8qt_krs.a
11 libavr51g2_8qt_k.a
12 libavr51g2_8qt_krs.a
13 libavr5g1_8qt_k.a
14 libavr5g1_8qt_krs.a
15 libavr5g2_8qt_k.a
16 libavr5g2_8qt_krs.a
17 libavr5g3_8qt_k.a
18 libavr5g3_8qt_krs.a
19 libv1g1_8qt_k.r90
20 libv1g1_8qt_krs.r90
21 libv1g2_8qt_k.r90
22 libv1g2_8qt_krs.r90
23 libv1g3_8qt_k.r90
24 libv1g3_8qt_krs.r90
25 libv1g4_8qt_k.r90
26 libv1g4_8qt_krs.r90
27 libv3g1_8qt_k.r90
28 libv3g1_8qt_krs.r90
29 libv3g2_8qt_k.r90
30 libv3g2_8qt_krs.r90
31 libv3g3_8qt_k.r90
32 libv3g3_8qt_krs.r90
33 libv3g4_8qt_k.r90
34 libv3g4_8qt_krs.r90
35 libv3g5_8qt_k.r90
36 libv3g5_8qt_krs.r90
37 libuc3a0128_16qt_k.a
38 libuc3a0128_16qt_krs.a
39 libuc3a0256_16qt_k.a
40 libuc3a0256_16qt_krs.a
41 libuc3a0512_16qt_k.a
42 libuc3a0512_16qt_krs.a
43 libxm128a1_gnu_8qt_k.a
44 libxm128a1_gnu_8qt_krs.a
45 libxm128a1_iar_8qt_k.r90
46 libxm128a1_iar_8qt_krs.r90
```

The library follows specific naming convention for all the configurations and devices supported. The general naming convention would be as follows:

#### *6.1.2.1 For the devices supported by the AVR GCC:*

**libvPgQ_8qt_R.a**

Which specifies the library is generated for corevP (Core number of Device), group gQ of the AVR device for the 8 channel QTouch with the configuration R for sensors.

Here , vP stands for versionP of core of AVR device to be used
(P can take values of 25,4,51,5).
gQ stands for groupQ of core of AVR device to be used
(Q can take values of 1,2 and
3 only if P is 5)
R stands for configuration based on users need
( R can take k – if configuration is only keys
krs – if configuration is keys/rotors/sliders
Alternativly specific devices are listed like UC3A0512 and xm128a1

#### *6.1.2.2 For devices supported by IAR EWAR:*

**libvPgQ_8qt_R.r90**

Which specifies the library is generated for corevP (Core number of Device), group gQ(group of core number for AVR) of the AVR device for the 8 channel QTouch with the configuration R for sensors.

Here , vP stands for versionP of core of AVR device to be used
(P can take values of 1 and 3).
gQ stands for groupQ of core of AVR device to be used
(Q can take values of 1,2,3,4 and
5 only if P is 3)
R stands for configuration based on users need
( R can take k – if configuration is only keys
krs – if configuration is keys/rotors/sliders

Alternativly specific devices are listed like xm128a1

Atmel QTouch Library 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.

### 6.1.3 Compatibility with Compilers:

The QTouch Library is supported by following versions of compilers. The user of the libraries is recommended to use the same versions for compatibility issues:

**Table 6-1** Compilers supporting QTouch Library

| **Tool** | **Version** |
|---|---|
| IAR Compiler | 5.20.3 |

| | |
|---|---|
| Embedded Workbench | 5.20 |
| IAR library builder | 1.03R |
| GCC – AVRStudio | 4.16 build 638 |
| WinAVR | 20090313 |

### 6.1.4 Appendix for Libraries:

**Table 6-2** QTouch acquisition method library

| Device | Ports Supported | Compiler | Library to be used (K) | Library to be used (Krs) |
|---|---|---|---|---|
| Attiny43u | A,B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny44 | A,B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny45 | B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny461 | A,B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny84 | A,B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny85 | B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny861 | A,B | IAR | `libv1g1_8qt_k.r90` | `libv1g1_8qt_krs.r90` |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Attiny48 | A,B,C,D | IAR | `libv1g2_8qt_k.r90` | `libv1g2_8qt_krs.r90` |
| | | GCC | `libv25g2_8qt_k.a` | `libv25g2_8qt_krs.a` |
| Attiny88 | A,B,C,D | IAR | `libv1g2_8qt_k.r90` | `libv1g2_8qt_krs.r90` |
| | | GCC | `libv25g2_8qt_k.a` | `libv25g2_8qt_krs.a` |
| Atmega8515 | A,B,C,D,E | IAR | `libv1g3_8qt_k.r90` | `libv1g3_8qt_krs.r90` |
| | | GCC | `libv4g1_8qt_k.a` | `libv4g1_8qt_krs.a` |
| Atmega8535 | A,B,C,D | IAR | `libv1g3_8qt_k.r90` | `libv1g3_8qt_krs.r90` |
| | | GCC | `libv4g1_8qt_k.a` | `libv4g1_8qt_krs.a` |
| Atmega8A | B,C,D | IAR | `libv1g3_8qt_k.r90` | `libv1g3_8qt_krs.r90` |
| | | GCC | `libv4g1_8qt_k.a` (use Atmega8 when compiling) | `libv4g1_8qt_krs.a` (use Atmega8 when compiling) |
| Atmega48P | B,C,D | IAR | `libv1g4_8qt_k.r90` | `libv1g4_8qt_krs.r90` |
| | | GCC | `libv4g2_8qt_k.a` | `libv4g2_8qt_krs.a` |
| Atmega88PA | B,C,D | IAR | `libv1g4_8qt_k.r90` | `libv1g4_8qt_krs.r90` |
| | | GCC | `libv4g2_8qt_k.a` (use Atmega88P when compiling) | `libv4g2_8qt_krs.a` (use Atmega88P when compiling) |
| Atmega8HVA | A,B,C | IAR | `libv1g4_8qt_k.r90` | `libv1g4_8qt_krs.r90` |
| | | GCC | `libv4g2_8qt_k.a` | `libv4g2_8qt_krs.a` |
| ATPWM2 | B,C,D,E | IAR | `libv1g4_8qt_k.r90` | `libv1g4_8qt_krs.r90` |
| | | GCC | `libv4g2_8qt_k.a` | `libv4g2_8qt_krs.a` |
| ATPWM2B | B,C,D,E | IAR | `libv1g4_8qt_k.r90` | `libv1g4_8qt_krs.r90` |
| | | GCC | `libv4g2_8qt_k.a` | `libv4g2_8qt_krs.a` |

| | | | | |
|---|---|---|---|---|
| ATPWM3 | B,C,D,E | IAR | libv1g4_8qt_k.r90 | libv1g4_8qt_krs.r90 |
| | | GCC | libv4g2_8qt_k.a | libv4g2_8qt_krs.a |
| ATPWM3B | B,C,D,E | IAR | libv1g4_8qt_k.r90 | libv1g4_8qt_krs.r90 |
| | | GCC | libv4g2_8qt_k.a | libv4g2_8qt_krs.a |
| Atmega64A | A,B,C,D,E, F,G | IAR | libv3g1_8qt_k.r90 | libv3g1_8qt_krs.r90 |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Atmega161 | A,B,C,D,E | IAR | libv3g2_8qt_k.r90 | libv3g2_8qt_krs.r90 |
| | | GCC | NOT SUPPORTED | NOT SUPPORTED |
| Atmega162 | A,B,C,D,E | IAR | libv3g2_8qt_k.r90 | libv3g2_8qt_krs.r90 |
| | | GCC | libv5g2_8qt_k.a | libv5g2_8qt_krs.a |
| Atmega163 | A,B,C,D | IAR | libv3g2_8qt_k.r90 | libv3g2_8qt_krs.r90 |
| | | GCC | libv5g2_8qt_k.a | libv5g2_8qt_krs.a |
| Atmega16A | A,B,C,D | IAR | libv3g2_8qt_k.r90 | libv3g2_8qt_krs.r90 |
| | | GCC | libv5g2_8qt_k.a (use Atmega16 when compileing) | libv5g2_8qt_krs.a (use Atmega16 when compileing) |
| Atmega323 | A,B,C,D | IAR | libv3g2_8qt_k.r90 | libv3g2_8qt_krs.r90 |
| | | GCC | libv5g2_8qt_k.a | libv5g2_8qt_krs.a |
| Atmega32A | A,B,C,D | IAR | libv3g2_8qt_k.r90 | libv3g2_8qt_krs.r90 |
| | | GCC | libv5g2_8qt_k.a (use Atmega32 when compiling) | libv5g2_8qt_krs.a (use Atmega32 when compiling)) |
| ATCAN32 | A,B,C,D,E, F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| ATCAN64 | A,B,C,D,E, F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega164P | A,B,C,D | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega165P | A,B,C,D,E, F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega168PA | B,C,D | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a (use Atmega168P when compiling) | libv5g3_8qt_krs.a (use Atmega168P when compiling) |
| Atmega169P | A,B,C,D,E, F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega16HVA | A,B,C | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega16U4 | B,C,D,E, F | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega324PA | A,B,C,D | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a (use Atmega324P when compiling) | libv5g3_8qt_krs.a (use Atmega324P when compiling) |
| Atmega325 | A,B,C,D,E,F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega3250P | A,B,C,D,E,F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega325P | A,B,C,D,E,F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega328P | B,C,D | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |
| | | GCC | libv5g3_8qt_k.a | libv5g3_8qt_krs.a |
| Atmega329 | A,B,C,D,E,F,G | IAR | libv3g3_8qt_k.r90 | libv3g3_8qt_krs.r90 |

| | | | | |
|---|---|---|---|---|
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega3290P | A,B,C,D,E,F,G | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega329P | A,B,C,D,E,F,G | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega32C1 | B,C,D,E | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega32HVB | A,B,C | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega32M1 | B,C,D,E | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega32U4 | B,C,D,E,F | IAR | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega32U6 | A,B,C,D,E,F | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega406 | A,B,C,D | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega640 | A,B,C,D,E,F,G | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega644P | A,B,C,D | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega645 | A,B,C,D,E,F,G, | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega6450 | A,B,C,D,E,F,G | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega649 | A,B,C,D,E,F,G, | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega6490 | A,B,C,D,E,F,G | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega64C1 | B,C,D,E | IAR | `NOT SUPPORTED` | `NOT SUPPORTED` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega64M1 | B,C,D,E | IAR | `NOT SUPPORTED` | `NOT SUPPORTED` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| ATPWM216 | B,C,D,E | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| ATPWM316 | B,C,D,E | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| ATUSB646 | A,B,C,D,E,F, | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| ATUSB647 | A,B,C,D,E,F, | IAR | `libv3g3_8qt_k.r90` | `libv3g3_8qt_krs.r90` |
| | | GCC | `libv5g3_8qt_k.a` | `libv5g3_8qt_krs.a` |
| Atmega128A | A,B,C,D,E,F,G | IAR | `libv3g4_8qt_k.r90` | `libv3g4_8qt_krs.r90` |
| | | GCC | `libv51g1_8qt_k.a (use Atmega128 when compiling)` | `libv51g1_8qt_krs.a (use Atmega128 when compiling)` |
| Atmega1280 | A,B,C,D,E,F,G | IAR | `libv3g5_8qt_k.r90` | `libv3g5_8qt_krs.r90` |
| | | GCC | `libv51g2_8qt_k.a` | `libv51g2_8qt_krs.a` |
| Atmega1281 | A,B,C,D,E,F,G | IAR | `libv3g5_8qt_k.r90` | `libv3g5_8qt_krs.r90` |
| | | GCC | `libv51g2_8qt_k.a` | `libv51g2_8qt_krs.a` |
| Atmega1284P | A,B,C,D | IAR | `libv3g5_8qt_k.r90` | `libv3g5_8qt_krs.r90` |
| | | GCC | `libv51g2_8qt_k.a` | `libv51g2_8qt_krs.a` |
| ATCAN128 | A,B,C,D,E,F,G | IAR | `libv3g5_8qt_k.r90` | `libv3g5_8qt_krs.r90` |

| | | | | |
|---|---|---|---|---|
| | | GCC | libv51g2_8qt_k.a | libv51g2_8qt_krs.a |
| ATUSB1287 | A,B,C,D,E,F | IAR | libv3g5_8qt_k.r90 | libv3g5_8qt_krs.r90 |
| | | GCC | libv51g2_8qt_k.a | libv51g2_8qt_krs.a |
| ATUSB1286 | A,B,C,D,E,F | IAR | libv3g5_8qt_k.r90 | libv3g5_8qt_krs.r90 |
| | | GCC | libv51g2_8qt_k.a | libv51g2_8qt_krs.a |
| | | | | |
| AT32UC3A0128 | A,B | IAR | NOT SUPPORTED | NOT SUPPORTED |
| | | GCC | libuc3a0128_16qt_k.a | libuc3a0128_16qt_krs.a |
| AT32UC3A0256 | A,B | IAR | NOT SUPPORTED | NOT SUPPORTED |
| | | GCC | libuc3a0256_16qt_k.a | libuc3a0256_16qt_krs.a |
| AT32UC3A0512 | A,B | IAR | NOT SUPPORTED | NOT SUPPORTED |
| | | GCC | libuc3a0512_16qt_k.a | libuc3a0512_16qt_krs.a |
| ATxmega128A1 | A,B,C,D,E,F | IAR | libxm128a1_iar_8qt_k.r90 | libxm128a1_iar_8qt_krs.r90 |
| | | GCC | libxm128a1_gnu_8qt_k.a | libxm128a1_gnu_8qt_krs.a |

The libraries that are supported as listed in the table is only supported provided the device memory requirements are also satisfied.
For AT32UC3A devices, IAR version 1 Charge Delay cycle is not supported. (2, 3, 4, 5, 10, 25, 50 are supported. For the GCC version 1 and 2 Charge Delay Cycles are not supported. 3, 4, 5, 10, 25, 50 are supported.
For ATxmega128A1 devices, GCC version, 1, 2, 3 Charge Delay Cycles are not supported. 4, 5, 10, 25, 50 are supported – For the IAR version 1 Charge Delay cycle is not supported. 2, 3, 4, 5, 10, 25, 50 are supported

### 6.1.5 Table Usage

Table 6-2 gives the various devices along with the libraries supported by both the IAR and GCC compilers along with the usage of different combinations of ports.

The user can refer the table as below:
1. Select the device that needs to have the QTouch library on.
2. Check the port availability for the selected device from appendix of libraries and select the combination of the ports based on the user application requirement from the table below. The user should be aware that the table below lists the complete combinations available, but shrinks based on the device that should be supported.

**Table 6-3** Port availability for touch sensing

| SNSK\SNS | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | AA | BA | CA | DA | EA | FA | GA | HA |
| B | AB | BB | CB | DB | EB | FB | GB | HB |
| C | AC | BC | CC | DC | EC | FC | GC | HC |
| D | AD | BD | CD | DD | ED | FD | GD | HD |
| E | AD | BE | CE | DE | EE | FE | GE | HE |
| F | AF | BF | CF | DF | EF | FF | GF | HF |
| G | AG | BG | CG | DG | EG | FG | GG | HG |
| H | AH | BH | CH | DH | EH | FH | GH | HH |

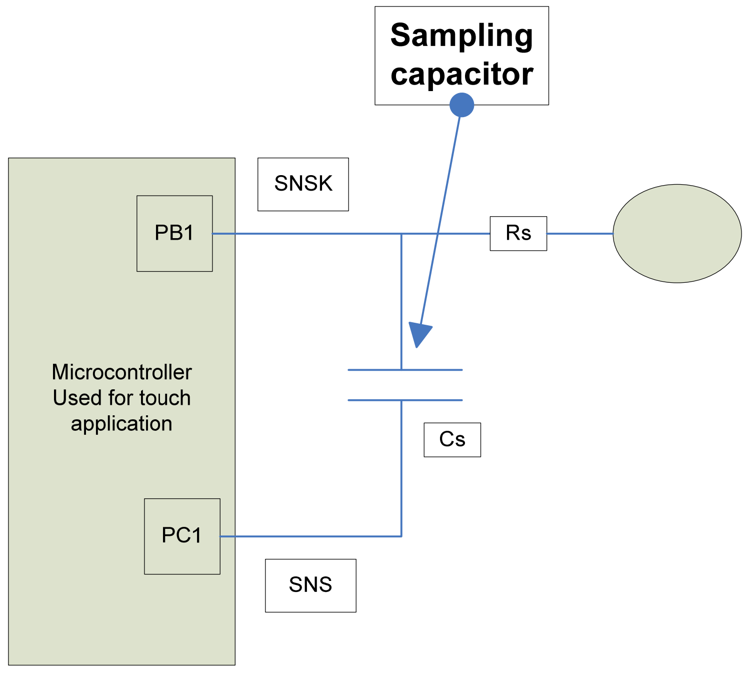
| | |
|---|---|
| 🟥 | = Not supported |
| ⬜ | = SNSK PORT |
| 🟨 | = SNS PORT |
| 🟦 | = Supported combination of ports |

In case of the SNS(C) and SNK(B) on two different ports, the user should mount the sensors onto the corresponding pins such as (PC0,PB0), (PC1,PB1), (PC2,PB2)..so on.
In this case 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 (UC3 up to pin 15)

In case of the SNS(A) and SNSK(A) on the same port, the user should always have the configuration as (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 . For AVR and XMEGA devices only 4 channels are supported when using SNS and SNSK ports on the same port, while 16 channels are supported on UC3

The sample schematic for using the port pins for SNS and SNSK looks similar to the one in figure 6-1:

**Figure 6-1** Schematic of mounting a channel using QTouch technology.
**Rs**- Series Resistor, **Cs** – Sample capacitor, **PB1**- PortB bit1, and **PC1**- PortC bit1.



3. Select the Compiler (after checking with the version of the compiler on the top of the table).
4. Select the library file from the last two columns of the table based on the configuration that the user wants to use

4.1. From last before column for ONLY KEYS configuration and
4.2. From last column for KEYS/ROTORS/SLIDERS configuration.

5. In the touch_lib_api.h, provide values to the PORTS and DELAY_CYLES depending on the need:

Ex:
```
#define QT_DELAY_CYCLES 1
#define QT_SNS_D
#define QT_SNSK_B
```

Meaning that the user should provide the desired combination of the ports taken from step No:2( D and B in this case) and measurement at 1 charge cycle time.
DELAY_CYCLES can have values of 1,2,3,4,5,10,25,50

After selecting the library file the user links the library file with the main.c application program file (including the header file touch_api.h) after editing it to include the code for the host application program.

### 6.1.6 Memory requirements and Example Projects for each of the library:

When the user uses the channels across ports (SNS and SNSK), the memory requirements for the library are as follows:

**Table 6-4** Memory requirements

| Library | Maximum Data memory required | Maximum Code memory required | Example projects |
|---|---|---|---|
| | | **GCC** | |
| `libavr25g1_8qt_k.a` | 112 | 3582 | avr25g1_8qt_example |
| `libavr25g1_8qt_krs.a` | 128 | 4042 | |
| `libavr25g2_8qt_k.a` | 112 | 3590 | avr25g2_8qt_example |
| `libavr25g2_8qt_krs.a` | 128 | 4050 | |
| `libavr4g1_8qt_k.a` | 112 | 2902 | avr4g1_8qt_example |
| `libavr4g1_8qt_krs.a` | 128 | 4026 | |
| `libavr4g2_8qt_k.a` | 112 | 2910 | avr4g2_8qt_example |
| `libavr4g2_8qt_krs.a` | 128 | 4034 | |
| `libavr51g1_8qt_k.a` | 112 | 2938 | avr51g1_8qt_example |
| `libavr51g1_8qt_krs.a` | 128 | 4106 | |
| `libavr51g2_8qt_k.a` | 112 | 2938 | avr51g2_8qt_example |
| `libavr51g2_8qt_krs.a` | 128 | 4106 | |
| `libavr5g1_8qt_k.a` | 112 | 2938 | avr5g1_8qt_example |

| | | | |
|---|---|---|---|
| libavr5g1_8qt_krs.a | 128 | 4106 | |
| libavr5g2_8qt_k.a | 112 | 2930 | avr5g2_8qt_example |
| libavr5g2_8qt_krs.a | 128 | 4098 | |
| libavr5g3_8qt_k.a | 112 | 2938 | avr5g3_8qt_example |
| libavr5g3_8qt_krs.a | 128 | 4106 | |
| libuc3a0128_16qt_k.a | 213 | 6258 | libuc3a0128_16qt_k_example |
| libuc3a0128_16qt_krs.a | 270 | 9194 | libuc3a0128_16qt_krs_example |
| libuc3a0256_16qt_k.a | 213 | 6258 | libuc3a0256_16qt_k_example |
| libuc3a0256_16qt_krs.a | 270 | 9194 | libuc3a0256_16qt_krs_example |
| libuc3a0512_16qt_k.a | 213 | 6258 | libuc3a0512_16qt_k_example |
| libuc3a0512_16qt_krs.a | 270 | 9194 | libuc3a0512_16qt_krs_example |
| libxm128a1_gnu_8qt_k.a | 112 | 2826 | libxm128a1_gnu_8qt_k_example |
| libxm128a1_gnu_8qt_krs.a | 128 | 3994 | libxm128a1_gnu_8qt_krs_example |
| **IAR** | | | |
| libv1g1_8qt_k.r90 | 112 | 1792 | v1g1_8qt_example |
| libv1g1_8qt_krs.r90 | 141 | 2922 | |
| libv1g2_8qt_k.r90 | 112 | 1800 | v1g2_8qt_example |
| libv1g2_8qt_krs.r90 | 141 | 2930 | |
| libv1g3_8qt_k.r90 | 112 | 1770 | v1g3_8qt_example |
| libv1g3_8qt_krs.r90 | 141 | 2894 | |
| libv1g4_8qt_k.r90 | 112 | 1778 | v1g4_8qt_example |
| libv1g4_8qt_krs.r90 | 141 | 2902 | |
| libv3g1_8qt_k.r90 | 112 | 1830 | v3g1_8qt_example |
| libv3g1_8qt_krs.r90 | 141 | 2988 | |
| libv3g2_8qt_k.r90 | 112 | 1822 | v3g2_8qt_example |
| libv3g2_8qt_krs.r90 | 141 | 2980 | |
| libv3g3_8qt_k.r90 | 112 | 1830 | v3g3_8qt_example |
| libv3g3_8qt_krs.r90 | 141 | 2988 | |
| libv3g4_8qt_k.r90 | 112 | 1830 | v3g4_8qt_example |
| libv3g4_8qt_krs.r90 | 141 | 2988 | |
| libv3g5_8qt_k.r90 | 112 | 1830 | v3g5_8qt_example |

| | | | |
|---|---|---|---|
| libv3g5_8qt_krs.r90 | 141 | 2988 | |
| libxm128a1_iar_8qt_k.r90 | 142 | 1812 | libxm128a1_iar__8qt_k_example |
| libxm128a1_iar_8qt_krs.r90 | 168 | 3066 | libxm128a1_iar__8qt_krs_example |

## 6.2 QMatrix acquisition

### 6.2.1 Introduction

Variants of the Atmel QTouch 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.

### 6.2.2 Compatibility with Compilers:

The libraries are supported based on the following versions of compilers, The user of the libraries is recommended to use the same versions for compatibility issues:

**Table 6-5 Compilers supporting QTouch Library**

| Tool | Version |
|---|---|
| IAR Compiler | 5.20.3 |
| Embedded Workbench | 5.20 |
| IAR library builder | 1.03R |
| GCC – AVR Studio | 4.16 build 638 |
| WinAVR | 20090313 |

**Table 6-6** QMatrix acquisition method library

| Device | Channels | Number of X x Y lines | Comp-iler | Library to be used (K) | Library to be used (Krs) |
|---|---|---|---|---|---|
| ATtiny48 | 8 | 4 x 2 | IAR | `libv1_8qm_dXX_k.r90` | `libv1_8qm_dXX_krs.r90` |
| | | | GCC | `NOT SUPPORTED` | `NOT SUPPORTED` |
| ATtiny88 | 8 | 4 x 2 | IAR | `libv1_8qm_dXX_k.r90` | `libv1_8qm_dXX_krs.r90` |
| | | | GCC | `libt88_8qm_dXX _k.a` | `libt88_8qm_dXX _krs.a` |
| ATmega48 | 8 | 4 x 2 | IAR | `libv1_8qm_dXX _k.r90` | `libv1_8qm_dXX _krs.r90` |
| | | | GCC | `NOT SUPPORTED` | `NOT SUPPORTED` |
| ATmega88 | 8 | 4 x 2 | IAR | `libv1_8qm_dXX _k.r90` | `libv1_8qm_dXX _krs.r90` |
| | | | GCC | `libm88_8qm_dXX _k.a` | `libm88_8qm_dXX _krs.a` |
| ATmega88 | 32 | 8 x 4 | IAR | `libv1_32qm_dXX _k.r90` | `libv1_32qm_dXX _krs.r90` |
| | | | GCC | `libm88_32qm_dXX _k.a` | `libm88_32qm_dXX _krs.a` |

XX- stands for the delays cycles (XX – 1, 2, 3, 4, 5, 10, 25, 50)

### 6.2.3 Table Usage

The Table gives the various devices along with the libraries that were supported by both the IAR and GCC compilers along with the usage of different combinations of ports.

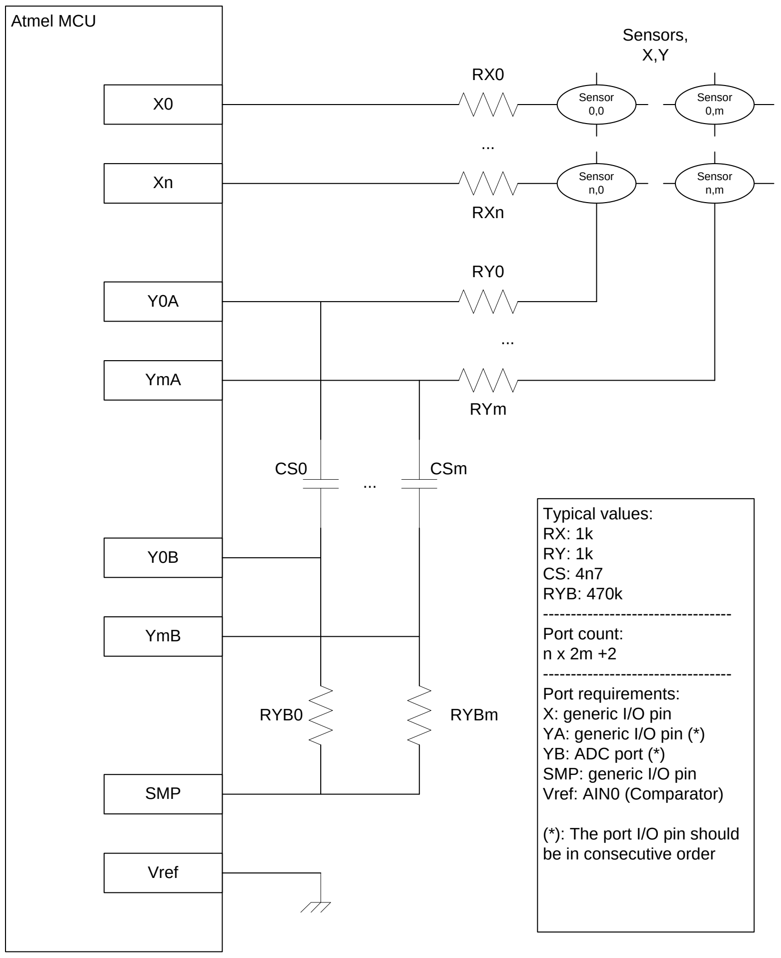
The user can refer the table as below:
1. Select the device that needs to have the QMatrix touch library on
2. Select the library file from the last two columns of the table based on the configuration that the user wants to use
   2.1. From last before column for ONLY KEYS configuration and
   2.2. From last column for KEYS/ROTORS/SLIDERS configuration.

3. Select the Compiler (after checking with the version of the compiler on the top of the table).
4. Select the library file from the last two columns of the table based on the configuration that the user wants to use
   4.1. From last before column for ONLY KEYS configuration and
   4.2. From last column for KEYS/ROTORS/SLIDERS configuration.

After selecting the library file the user links the library file with the main.c application program file (including the header file touch_api.h) after editing it to include the code for the host application program.

### 6.2.4 Schematics:

**Figure 6-2** Schematics showing the configuration of QMatrix circuit with ATMEL's AVR microcontroller



Typical values:
RX: 1k
RY: 1k
CS: 4n7
RYB: 470k
----------------------------------
Port count:
n x 2m +2
----------------------------------
Port requirements:
X: generic I/O pin
YA: generic I/O pin (*)
YB: ADC port (*)
SMP: generic I/O pin
Vref: AIN0 (Comparator)

(*): The port I/O pin should be in consecutive order

**Table 6-7** Library I/O configurations

| **Line label** | **8-channel configuration** | **32 channel configuration** |
|---|---|---|
| X0 | PORTB0 | PORTB0 |
| X1 | PORTB1 | PORTB1 |
| X2 | PORTB2 | PORTB2 |
| X3 | PORTB3 | PORTB3 |
| X4 | (not used in this configuration) | PORTB4 |
| X5 | (not used in this configuration) | PORTB5 |

| | | |
|---|---|---|
| X6 | (not used in this configuration) | PORTB6 |
| X7 | (not used in this configuration) | PORTB7 |
| Y0A | PORTD0 | PORTD0 |
| Y1A | PORTD1 | PORTD1 |
| Y2A | (not used in this configuration) | PORTD2 |
| Y3A | (not used in this configuration) | PORTD3 |
| Y0B | PORTC0 | PORTC0 |
| Y1B | PORTC1 | PORTC1 |
| Y2B | (not used in this configuration) | PORTC2 |
| Y3B | (not used in this configuration) | PORTC3 |
| SMP | PORTD7 | PORTD7 |
| VREF | PORTD6 | PORTD6 |

**Table 6-7** Channel configuration

| Line label | 8-channel configuration | 32 channel configuration |
|---|---|---|
| Channel 0 | X0Y0 | X0Y |
| Channel 1 | X1Y0 | X1Y |
| Channel 2 | X2Y0 | X2Y |
| Channel 3 | X3Y0 | X3Y |
| Channel 4 | X0Y1 | X4Y |
| Channel 5 | X1Y1 | X5Y |
| Channel 6 | X2Y1 | X6Y |
| Channel 7 | X3Y1 | X7Y |
| Channel 8 | N/A | X0Y1 |
| Channel 9 | N/A | X1Y1 |
| Channel 10 | N/A | X2Y1 |
| Channel 11 | N/A | X3Y1 |
| Channel 12 | N/A | X4Y1 |
| Channel 13 | N/A | X5Y1 |
| Channel 14 | N/A | X6Y1 |

| | | |
|---|---|---|
| Channel 15 | N/A | X7Y1 |
| Channel 16 | N/A | X0Y2 |
| Channel 17 | N/A | X1Y2 |
| Channel 18 | N/A | X2Y2 |
| Channel 19 | N/A | X3Y2 |
| Channel 20 | N/A | X4Y2 |
| Channel 21 | N/A | X5Y2 |
| Channel 22 | N/A | X6Y2 |
| Channel 23 | N/A | X7Y2 |
| Channel 24 | N/A | X0Y3 |
| Channel 25 | N/A | X1Y3 |
| Channel 26 | N/A | X2Y3 |
| Channel 27 | N/A | X3Y3 |
| Channel 28 | N/A | X4Y3 |
| Channel 29 | N/A | X5Y3 |
| Channel 30 | N/A | X6Y3 |
| Channel 31 | N/A | X7Y3 |

### 6.2.5 Memory requirements and Example Projects for each of by the library:

**Table 6-8** Memory requirements

| Library | Maximum Data Memory required | Maximum Code memory required | Example projects |
|---|---|---|---|
| | GCC | | |
| `libt88_8qm_dXX_k.a` | 130 | 3824 | t88_8qm_example |
| `libt88_8qm_dXX_krs.a` | 169 | 4599 | t88_8qm_example |
| `libm88_8qm_dXX_k.a` | 130 | 3188 | m88_8qm_example |
| `libm88_8qm_dXX_krs.a` | 169 | 4607 | m88_8qm_example |
| `libm88_32qm_dXX_k.a` | 419 | 3609 | m88_32qm_example |

| | | | |
|---|---|---|---|
| libm88_32qm_dXX_krs.a | 529 | 5040 | m88_32qm_example |
| **IAR** | | | |
| libv1_8qm_dXX_k.r90 | 204 | 2310 | v1_8qm_example |
| libv1_8qm_dXX_krs.r90 | 249 | 3548 | v1_8qm_example |
| libv1_32qm_dXX_k.r90 | 505 | 2357 | v1_32qm_example |
| libv1_32qm_dXX_krs.r90 | 626 | 3599 | v1_32qm_example |

XX- stands for the delays cycles (XX – 1, 2, 3, 4, 5, 10, 25, 50)

# 7 Known Issues:

## 7.1 Linker warning:

### 7.1.1 SFRB Warning:

The Library generates a linker warning with SFR register when some of the devices are used. These warnings need to be ignored.

```
      Warning[w6]: Type conflict for external/entry "_A_DDRC", in
module burst_10_BC against external/entry in module main;
class/struct/union field/base
types do not match for field/base ''; class/struct/union field names
do not match: DDRC_DDC7 vs DDRC_Dummy7
/* In module burst_10_BC: */
union /* Elements: 3, Bytes: 1 */
/* First seen in burst_10_BC */
{
unsigned char DDRC;
struct /* Elements: 8, Bytes: 1 */
/* First seen in main */
{
unsigned char DDRC_Bit0 : 1 /* disp: 0 */;
unsigned char DDRC_Bit1 : 1 /* disp: 1 */;
unsigned char DDRC_Bit2 : 1 /* disp: 2 */;
unsigned char DDRC_Bit3 : 1 /* disp: 3 */;
unsigned char DDRC_Bit4 : 1 /* disp: 4 */;
unsigned char DDRC_Bit5 : 1 /* disp: 5 */;
unsigned char DDRC_Bit6 : 1 /* disp: 6 */;
unsigned char DDRC_Bit7 : 1 /* disp: 7 */;
} ;
struct /* Elements: 8, Bytes: 1 */
/* First seen in burst_10_BC */
{
unsigned char DDRC_DDC0 : 1 /* disp: 0 */;
unsigned char DDRC_DDC1 : 1 /* disp: 1 */;
unsigned char DDRC_DDC2 : 1 /* disp: 2 */;
unsigned char DDRC_DDC3 : 1 /* disp: 3 */;
unsigned char DDRC_DDC4 : 1 /* disp: 4 */;
unsigned char DDRC_DDC5 : 1 /* disp: 5 */;
unsigned char DDRC_DDC6 : 1 /* disp: 6 */;
unsigned char DDRC_DDC7 : 1 /* disp: 7 */;
} ;
} __io volatile _A_DDRC;
/* In module main: */
union /* Elements: 3, Bytes: 1 */
/* First seen in main */
{
unsigned char DDRC;
struct /* Elements: 8, Bytes: 1 */
/* First seen in main */
{
unsigned char DDRC_Bit0 : 1 /* disp: 0 */;
unsigned char DDRC_Bit1 : 1 /* disp: 1 */;
unsigned char DDRC_Bit2 : 1 /* disp: 2 */;
unsigned char DDRC_Bit3 : 1 /* disp: 3 */;
unsigned char DDRC_Bit4 : 1 /* disp: 4 */;
unsigned char DDRC_Bit5 : 1 /* disp: 5 */;
unsigned char DDRC_Bit6 : 1 /* disp: 6 */;
unsigned char DDRC_Bit7 : 1 /* disp: 7 */;
} ;
struct /* Elements: 8, Bytes: 1 */
/* First seen in main */
{
unsigned char DDRC_DDC0 : 1 /* disp: 0 */;
unsigned char DDRC_DDC1 : 1 /* disp: 1 */;
unsigned char DDRC_DDC2 : 1 /* disp: 2 */;
unsigned char DDRC_DDC3 : 1 /* disp: 3 */;
unsigned char DDRC_DDC4 : 1 /* disp: 4 */;
unsigned char DDRC_DDC5 : 1 /* disp: 5 */;
unsigned char DDRC_DDC6 : 1 /* disp: 6 */;
unsigned char DDRC_Dummy7 : 1 /* disp: 7 */;
} ;
} __io volatile _A_DDRC;
Warning[w6]: Type conflict for external/entry "_A_PORTC", in module
burst_10_BC against external/entry in module main;
class/struct/union field/base
types do not match for field/base ''; class/struct/union field names
do not match: PORTC_PORTC7 vs PORTC_Dummy7
/* In module burst_10_BC: */
union /* Elements: 3, Bytes: 1 */
/* First seen in burst_10_BC */
{
unsigned char PORTC;
struct /* Elements: 8, Bytes: 1 */
/* First seen in main */
{
unsigned char PORTC_Bit0 : 1 /* disp: 0 */;
unsigned char PORTC_Bit1 : 1 /* disp: 1 */;
unsigned char PORTC_Bit2 : 1 /* disp: 2 */;
unsigned char PORTC_Bit3 : 1 /* disp: 3 */;
unsigned char PORTC_Bit4 : 1 /* disp: 4 */;
unsigned char PORTC_Bit5 : 1 /* disp: 5 */;
unsigned char PORTC_Bit6 : 1 /* disp: 6 */;
unsigned char PORTC_Bit7 : 1 /* disp: 7 */;
} ;
struct /* Elements: 8, Bytes: 1 */
/* First seen in burst_10_BC */
{
unsigned char PORTC_PORTC0 : 1 /* disp: 0 */;
unsigned char PORTC_PORTC1 : 1 /* disp: 1 */;
unsigned char PORTC_PORTC2 : 1 /* disp: 2 */;
unsigned char PORTC_PORTC3 : 1 /* disp: 3 */;
unsigned char PORTC_PORTC4 : 1 /* disp: 4 */;
unsigned char PORTC_PORTC5 : 1 /* disp: 5 */;
unsigned char PORTC_PORTC6 : 1 /* disp: 6 */;
unsigned char PORTC_PORTC7 : 1 /* disp: 7 */;
} ;
} __io volatile _A_PORTC;
/* In module main: */
union /* Elements: 3, Bytes: 1 */
/* First seen in main */
{
unsigned char PORTC;
struct /* Elements: 8, Bytes: 1 */
/* First seen in main */
{
unsigned char PORTC_Bit0 : 1 /* disp: 0 */;
unsigned char PORTC_Bit1 : 1 /* disp: 1 */;
unsigned char PORTC_Bit2 : 1 /* disp: 2 */;
unsigned char PORTC_Bit3 : 1 /* disp: 3 */;
unsigned char PORTC_Bit4 : 1 /* disp: 4 */;
unsigned char PORTC_Bit5 : 1 /* disp: 5 */;
unsigned char PORTC_Bit6 : 1 /* disp: 6 */;
unsigned char PORTC_Bit7 : 1 /* disp: 7 */;
} ;
struct /* Elements: 8, Bytes: 1 */
/* First seen in main */
{
unsigned char PORTC_PORTC0 : 1 /* disp: 0 */;
unsigned char PORTC_PORTC1 : 1 /* disp: 1 */;
unsigned char PORTC_PORTC2 : 1 /* disp: 2 */;
unsigned char PORTC_PORTC3 : 1 /* disp: 3 */;
unsigned char PORTC_PORTC4 : 1 /* disp: 4 */;
unsigned char PORTC_PORTC5 : 1 /* disp: 5 */;
unsigned char PORTC_PORTC6 : 1 /* disp: 6 */;
unsigned char PORTC_Dummy7 : 1 /* disp: 7 */;
} ;
} __io volatile _A_PORTC;
```

## 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/

**_Technical Support_**
avr@atmel.com

**_Sales Contact_**
www.atmel.com/contacts

**_Literature Request_**
www.atmel.com/literature

Rev. 8207C-AT42-06/09