AVR252: TV Control Touch Keyboard

Features

• ATtiny48/88 centric QTouch® design
  - QTouch hardware & associated circuitry
  - Customized firmware using Atmel QTouch Library
• 6 Capacitive Touch Keys
• Panel thickness: upto 50mm glass or 20mm plastic
  (Electrode size & Cs dependent)
• Analog Output for all Keys
• Buzzer annunciation on Key touch
• Wake-up Functionality
• ISP Programming
• Debug support
• Operation with 3.3V & 4.2V
• Patents & Trademarks :
  - QTouch (patented charge-transfer method)
  - QMatrix™ (patented charge-transfer method)
  - AKS™ (patented Adjacent Key Suppression®) technology

1 Introduction

TV Control Touch Keyboard is designed to be the drop-in replacement for an existing electromechanical (resistive ladder) type TV control keyboard. It is based on the latest Capacitive Touch Technology for man-machine/user interface. Though designed for TV control application, can be used for any other application with 6~7 Touch Keys. The Touch Keyboard outputs respective analog (DC) output upon touching any key, based on the supply (VCC) voltage supplied externally.

The Figures 1-1 and 1-2 shows the AVR252 board assembled with the components, from the component side and the touch keypad side respectively. As shown, it has 7 Keys controlling various TV parameters such as ON/OFF, TV/AV, PROG +, PROG -, VOL +, VOL - & MENU (not a physical touch key, but simultaneous touching VOL+ & VOL- keys together will operate as a Menu Key

Figure 1-1. PCBA Component-Side View

Figure 1-2. PCBA Touch Keypad-Side View
2 Touch Keyboard - Hardware

The System Block Diagram is as illustrated in Figure 1-3 below. The control keyboard has an on-board buzzer for indicating a key touch, wake up functionality for recovering from the power down mode when no key is operated for a predefined duration. The analog output (ANO) is supplied to ADC of TV Controller or any Host Controller thereby to make a decision on respective key touch. The keyboard also has programming & debug interface available on the 8-pin connector and test points. The Figure 1-4 shows the placement of the critical components and the keys on the touch control board.

**Figure 2-3. System Block Diagram**

![System Block Diagram](image)

**Figure 2-4. Keyboard Layout and Critical Component Placement**

![Keyboard Layout](image)

Figure 5-1 contained in Appendix at the end of this application note shows the circuit schematic for the TV control touch keyboard. The keyboard must be externally powered with a well regulated supply voltage of 3.3 V or 4.2 V.

**CAVEAT:** During normal operation the bottom layer is to be insulated using a dielectric material and the exposed electrodes and vias must not be touched or made electrical contact with. Without the above pre-requisite the normal operation is not guaranteed.
2.1 QTouch

The QTouch devices are charging a sense electrode of unknown capacitance to a known potential. The electrode is typically a copper area on a printed circuit board. The resulting charge is transferred into a measurement circuit. By measuring the charge after one or more charge-and-transfer cycles, the capacitance of the sense plate can be determined. Placing a finger on the touch surface introduces external capacitance that affects the flow of charge at that point. This registers as a touch.

Signal processing in the decision logic makes QTouch robust and reliable. False triggering due to electrostatic spikes or momentary unintentional touch or proximity is eliminated. QTouch sensors can drive single or multiple keys. Where multiple keys are used, each key can be set for an individual sensitivity level. Keys of different sizes and shapes can be used to meet both functional and aesthetic requirements.

For excellent electromagnetic compatibility, QTouch sensors use spread-spectrum modulation and sparse, randomized charging pulses with long delays between bursts. Individual pulses can be as short as 5% or less of the intra-burst pulse spacing. The benefits of this approach include lower cross-sensor interference, reduced RF emissions and susceptibility, and low power consumption.

QTouch devices feature automatic drift compensation to account for slow changes due to ageing or changing environmental conditions. They have a dynamic range of several decades and do not require coils, oscillators, RF components, special cable, RC networks, or a lot of discrete parts. As an engineering solution QTouch is simple, robust, elegant, and affordable.

Where several touch keys are close together, an approaching finger causes a change in capacitance around more than one key. Atmel’s patented adjacent key suppression – AKS - uses an iterative technique to repeatedly measure the capacitance change on each key, compare the results and determine which key the user intended to touch. AKS then suppresses or ignores signals from all other keys, providing that the signal from the selected key remains above the threshold value. This prevents false touch detections on adjacent keys. The AKS is selectable by the system designer. For details on this patented AKS Adjacent Key Suppression, refer to an Application Note QTAN0031B : Avoiding False Touch Inputs on http://www.atmel.com/dyn/products/app_notes.asp?family_id=697

The general concept of capacitive touch sensing using QTouch is illustrated below.

Figure 2-1. QTouch Circuit Diagram
2.2 Sensitivity

Higher touch sensitivity on the QTouch technology can be achieved on the multiplexed channels using either larger touch keys or larger sense capacitances which increase the sensitivity on these channels. The power consumption is increased if touch keys with larger area used or the sense capacitances are of higher denomination. It is recommended to use the touch keys with areas not larger than the surface contact area of the finger on the front panel or dielectric surface. Increasing the sensitivity using either of these techniques can also lead to adverse effects such as susceptibility to noise. The existing sense capacitances of 22 nF can be increased up to 47nF for the above mentioned reasons for the multiplexed channels in order to have a normalized value of sensitivity across all channels. The thresholds can also be adjusted in the board firmware in order to take into account the sensitivity tuning and hardware changes. These changes can only come into affect during compile time and reflected in hardware after firmware upgrade.

2.3 Buzzer Driver

The pizo-electric Buzzer on-board is used for annunciation on any key touch. The Buzzer rings only for 30ms (4KHz, 2.2V) on any key touch. The Figure 2-4 below shows the circuit schematic of the active buzzer driver on-board AVR252. The buzzer driver circuit is driven by the capacitive MCU ATtiny48/88 on pin 32 of the MLF/QFN package which is pin number 2 on PORT D. Resistor R11 is used a current limiting resistor, while D1 as a free-wheeling diode.

![Buzzer Drive Circuit](image)

**Figure 2-4. Buzzer Drive Circuit**
2.4 Power Supply

2.4.1 Power Supply Considerations

The board must be powered externally from a well regulated D.C. supply of voltage 3.3V or from 4.2V. The VCC pin of the MCU ATtiny48/88 is decoupled using two external capacitances of value 0.1uF and one of 4.7uF. The schematic in Figure 5-1 shows the Ferrite Bead in series with the supply which can provide protection from high frequency current spikes in the supply. It is recommended that the supply ripple and noise be not more than ± 25 mV & load regulation not to exceed 1% for normal operation of the board. The supply voltage is sampled ten times at regular intervals during startup to measure the internal bandgap voltage of 1.1V with the reference pin AVCC tied externally to VCC. After these ten samples the supply is not monitored and changes in supply henceforth may lead to unpredictable results. The result of this sampling operation will determine the set of values for analog output.

2.4.2 Power Consumption

The firmware utilizes the power management features of capacitive MCU ATtiny48/88 to provide the wake-up functionality. During normal operation or PWM generation the current drawn from the supply is a function of the key touched as the PWM frequency is maintained constant at 117KHz to match the RC passive LP filter response. High frequency allows physically small sized external components (coils, capacitors), hence reduces total system cost. During idle mode or when no touch keys are touched after each set of capacitive measurements the MCU enters the power down sleep mode for 75ms and resumes the capacitive function on wakeup. During the power down sleep mode only the asynchronous modules are operational and WDT which runs on a separate oscillator of frequency 128KHz at standard operating conditions is running. The WDT interrupts are enabled and after the pre-defined period the WDT subsystem generates an interrupt to wakeup the MCU for continuing the capacitive functions. The tracked time is updated on wakeup. Similarly larger values of sense capacitance will also affect the currents drawn during burst cycles.

Figure 2-5. Steady State Power Consumption (Idle State) for VCC=3.3V
2.5 Board Connector

The Figure 2-2 shows the 8-pin connector details of J1 available on the board for interfacing with the TV and for programming.

- **GND**  Ground
- **ANO**  Analog output
- **GND**  Ground
- **VCC**  Supply
- **SCK**  SPI clock
- **MISO**  Master-In Slave-Out
- **MOSI**  Master-Out Slave-In
- **RESET** Reset

**NOTE:** Since MOSI, MISO and SCK pins are multiplexed with channels 3, 4 and 5 respectively, it is recommended to remove the 8-pin connector with 8 wires after the SPI programming for firmware update and preferably use an 8-pin connector with only 3 wires viz. ANO, GND and VCC (i.e. pins 2, 3 and 4) of connector J1 for normal operation. The wires can induce stray capacitive effects on the channels and add to noise as well as affect the capacitive measurement results and ultimately decrease touch sensitivity on the enabled channels using these pins.
2.6 Programming and Debugging

2.6.1 ISP Programming

The firmware upgrade on the board can be done using any standard programming tool with provision of SPI. The Figure 2-3 shows the connection details required to connect the board with any programming tool with SPI port using one-one signal mapping.

![Figure 2-3. ISP6PIN Connection Details](image)

2.6.2 debugWIRE

The debugWIRE On-chip debug system uses a one-wire, bi-directional interface to control the program flow, execute AVR instructions in the CPU and to program the different non-volatile memories. When debugWIRE is enabled, the RESET port pin is configured as a wire-AND (open-drain) bi-directional I/O pin with pull-up enabled and becomes the communication gateway between target and emulator. Please ensure the guidelines for this usage as mentioned in the device datasheet.
3 Analog Output

The board communicates the touch status to the host controller on pin-2 of J1 (8-Pin Connector), by outputting analog (DC) voltage for respective key. The analog output is RC filtered PWM signal. Table 3-1 lists the various analog output voltages for both the values of supply voltage (VCC).

Table 3-1. Analog Output Values

<table>
<thead>
<tr>
<th>Keys</th>
<th>Analog Output (VCC=3.3V)</th>
<th>Analog Output (VCC=4.2V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle State</td>
<td>3.3 V</td>
<td>4.2 V</td>
</tr>
<tr>
<td>ON/OFF</td>
<td>2.9 V</td>
<td>0.6 V</td>
</tr>
<tr>
<td>TV/AV</td>
<td>2.2 V</td>
<td>3.0 V</td>
</tr>
<tr>
<td>PROG +</td>
<td>1.3 V</td>
<td>1.5 V</td>
</tr>
<tr>
<td>PROG -</td>
<td>2.5 V</td>
<td>3.5 V</td>
</tr>
<tr>
<td>VOL +</td>
<td>1.9 V</td>
<td>2.5 V</td>
</tr>
<tr>
<td>VOL -</td>
<td>0.3 V</td>
<td>2.0 V</td>
</tr>
<tr>
<td>MENU *</td>
<td>0.8 V</td>
<td>1.1 V</td>
</tr>
</tbody>
</table>

* Touch keys VOL + and VOL - need to be touched together to activate the MENU function.

The Figure 3-1 and Figure 3-2 shows the PWM waveform outputs at pin 2 of J1 with timing details for both supply voltages (3.3V and 4.2V).

The waveforms which are observed at the Analog Output Signal pin will be heralded by two untimely events of user interaction i.e.

- A Touch Control Key being touched
- A Touch Control Key being released (not touched)

At steady state i.e. when none of the touch control keys are touched the PWM output will be held high (continuous). The buzzer will be active for the duration of 30ms from the time any key is touched. The PWM output will be held low for the buzzer active duration i.e. 30ms from the time any key is touched. The PWM output will be active as per the specifications for the respective keys i.e. ON/OFF, TV/AV, PROG +, PROG -, VOL +, VOL- and MENU generating a D.C. voltage level using a RC Low Pass passive filter from the time 30ms after any key is touched until the key is not touched (infinite max-on duration). The PWM output will again settle at the steady state value of in-active high from the time the key is not touched.
Figure 3-1. PWM Waveforms for VCC = 3.3 V

Figure 3-2. PWM Waveforms for VCC = 4.2 V
3.1 Generation of PWM signal

The PWM signal is generated using 16-bit Timer/Counter1 with PWM on chip ATtiny48/88. Figure 3-3 shows the values of different registers, modes, and constants used for the generation of PWM signal of a specified frequency (117kHz). Waveform Generation Mode15 is used for the PWM signal generation (16-bit Fast PWM mode) with compare output mode3 (set on compare match, clear on TOP). Depending upon the supply voltage applied one of the two look-up tables is selected for loading OCR1B register of Timer/Counter1. Each lookup table contains constants for generating PWM waveform where a position within the table is indexed using a pointer which denotes one of the seven key functions.

![PWM Signal Generation Diagram](image)

**Figure 3-3.** Generation of PWM signal using Timer/Counter1

3.2 PWM Lookup Tables

The system clock is an on-chip 8MHz calibrated RC oscillator with Startup Time (Power Down / Reset) as 6 CK / 14 CK + 65 ns set by the ATtiny48/88 fuses.

PWM Cycles in terms of System Clock Ticks

\[
\text{TOP value used for OCR1A for waveform generation mode 15 (16-bit Fast PWM)}
\]

\[
\text{PWM Cycles} = \frac{1}{\text{PWM frequency}} / \text{(System Clock Time Period)}
\]

\[
= \frac{1}{117 \text{ KHz}} / (1 / 8 \text{ MHz})
\]

\[
= 68
\]
Constant to be used as the compare value for generation of the desired analog (D.C.) voltage using PWM channel B in OCR1B.

\[ \text{Const} = 68 - \frac{V \times 68}{VCC} \]

Using the above method the lookup tables were populated for both the operating voltage levels viz. VCC = 3.3 V (Low) and VCC = 4.2 V (High).

**Table 3-2. Lookup Tables for PWM Generation**

<table>
<thead>
<tr>
<th>VCC</th>
<th>DC Voltage Low</th>
<th>DC Voltage High</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.3 V</td>
<td>4.2 V</td>
</tr>
<tr>
<td>Touch Control Parameter</td>
<td>ANO</td>
<td>Constant</td>
</tr>
<tr>
<td>Idle State</td>
<td>3.3 V</td>
<td>-</td>
</tr>
<tr>
<td>ON/OFF</td>
<td>2.9 V</td>
<td>8</td>
</tr>
<tr>
<td>TV/AV</td>
<td>2.2 V</td>
<td>23</td>
</tr>
<tr>
<td>PROG +</td>
<td>1.3 V</td>
<td>41</td>
</tr>
<tr>
<td>PROG -</td>
<td>2.5 V</td>
<td>16</td>
</tr>
<tr>
<td>VOL +</td>
<td>1.9 V</td>
<td>29</td>
</tr>
<tr>
<td>VOL -</td>
<td>0.3 V</td>
<td>62</td>
</tr>
<tr>
<td>MENU</td>
<td>0.8 V</td>
<td>52</td>
</tr>
</tbody>
</table>

During the idle state the compare output mode is switched to non PWM mode 0 in which OC1A/OC1B is disconnected for normal port operation which yields a steady high or low level as required at the PWM output pin without any drop when the port is configured as output and set accordingly.

### 3.3 PWM Resolution for analog (D.C.) output voltage

The PWM resolution is dependent on many factors such as the system clock frequency, waveform generation mode used, clock prescaler and TOP value. Higher clock frequency will allow for larger number of clock ticks within the given time period required for a specified frequency in this case 117 KHz. The waveform generation mode defines whether 8, 9, 10 or 16-bit PWM resolution can be achieved, the clock prescaler can divide the timer clock by a predetermined scale, and the TOP value determines the end of each counting cycle. The resolution of the PWM output can hence be calculated as follows:

\[ \text{Resolution} = \frac{1}{68} \times VCC \]

**Resolution**

- \( 0.0485 \text{ V} \) (For VCC = 3.3 V)
- \( 0.0618 \text{ V} \) (For VCC = 4.2 V)
3.4 Noise Margin for PWM output

The noise margin for the analog output can be calculated considering the worst case scenario. Leaving a margin of 33% of the minimum delta available at the output signal the noise margin of 0.1 V is obtained for the board design. Figure 3-4 shows the graphic representation of the calculation of noise margin under worst case considerations.

*Figure 3-4. Noise Margin for PWM analog output*
4 Firmware

The firmware of the Touch board consists of two parts, Atmel QTouch Library and User Application code. The touch functions can be invoked from the user application code. The Atmel QTouch Library does only the capacitive sensing and the post processing, the rest of the functions like driving GPIO, sleeping, communication etc. has to be carried out by the User application code. The Atmel QTouch library incorporates the patented AKS (Adjacent Key Suppression) and the board is designed on the patented QTouch technology for capacitive touch sensing. This application note does not exhaustively describe the details of the Atmel QTouch library and must be used in conjunction with the library user guide.

4.1 Atmel QTouch Library

The Atmel QTouch Library 1.1 is available in the form of a pre-compiled archive which is a royalty free static library available for use with the IAR™ compiler along with the C header files necessary for linking. The user applications can be developed on top of the Atmel QTouch Library for many of the supported capacitive MCUs including ATtiny48/88. The library uses no timers, interrupts or other chip resources except RAM, ROM some register variables and GPIO. The library provides only touch sensing and the user application must provide any other functionality required. The Atmel QTouch Library used for the existing firmware is t88_8qt_BD_c5_k.r90.

Library file nomenclature is as follows

"Device"._"NoChannels+Technology"._"SNSKport+SNSport"._"cN"._"k or krs".r90

- Device: Device name (m is ATmega, t is ATtiny)
- NoChannels: The number of channels supported
- Technology: qt=QTouch, qm=QMatrix
- SNSKport The port used for SNSK electrodes of the touch sensor
- SNSport The port used for SNS electrodes of the touch sensor
- cN: N is the no:of charge cycles for QT and the dwell time for QM
- k or krs: k=library for keys only, krs=library for keys, rotors and sliders

The general flow for using the Atmel QTouch library can be described as follows.

- The user 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 user wants to dynamically reconfigure the library at runtime.
- The user application calls “qt_enable_key()”, “qt_enable_rotor()” and/or “qt_enable_slider()” as required to configure the touch sensors.
- The user application calls “qt_init_sensing()” to initialize the library.
- Thereafter, the user 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.
4.2 Application Programming Interface

4.2.1 Manifest Constants

The API defines the manifest constants listed in Table 4-1 that document the library. The library has been built using these values, and they should not be changed.

Table 4-1. Manifest Constants

<table>
<thead>
<tr>
<th>Manifest Constants</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QT_NUM_CHANNELS</td>
<td>The number of touch channels supported by the library.</td>
</tr>
<tr>
<td>QT_MAX_NUM_ROTORS_SLIDERS</td>
<td>The maximum number of rotors or sliders supported by the library.</td>
</tr>
</tbody>
</table>

4.2.2 Type Definitions

The API defines the typedefs listed in Table 4-2.

Table 4-2. Typedefs

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>An unsigned 8-bit number.</td>
</tr>
<tr>
<td>uint16_t</td>
<td>An unsigned 16-bit number.</td>
</tr>
<tr>
<td>int16_t</td>
<td>A signed 16-bit number.</td>
</tr>
<tr>
<td>threshold_t</td>
<td>An unsigned 8-bit number setting a sensor detection threshold.</td>
</tr>
</tbody>
</table>

4.2.3 Structs

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

Table 4-3. Struct

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

The API uses the enumerations listed in Table 4-4.

**Table 4-4. Enumerations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>aks_group_t</td>
<td>NO_AKS_GROUP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AKS_GROUP_7</td>
<td></td>
</tr>
<tr>
<td>channel_t</td>
<td>CHANNEL_0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHANNEL_7</td>
<td></td>
</tr>
<tr>
<td>hysterisis_t</td>
<td>HYST_50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HYST_25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HYST_12.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HYST_6.25</td>
<td></td>
</tr>
<tr>
<td>recal_threshold_t</td>
<td>RECAL_100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECAL_50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECAL_12.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RECAL_6.25</td>
<td></td>
</tr>
<tr>
<td>resolution_t</td>
<td>RES_1_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_2_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_3_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_4_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_5_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_6_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_7_BIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RES_8_BIT</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4-4 Notes:
- **aks_group_t**: 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.

- **hysterisis_t**: 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**: 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**: 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.
4.2.5 Global Touch Sensing Status

The global touch sensing status is available to the user application through the variable listed in Table 4-5.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_touch_status</td>
<td>qt_touch_status_t</td>
<td>The state of the library sensors.</td>
</tr>
</tbody>
</table>

4.2.6 Global Touch Sensing Configuration

Touch sensing is configured globally with the parameters listed in Table 4-6.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_di</td>
<td>uint8_t</td>
<td>Sensor detect integration (DI) limit. Default value: 4</td>
</tr>
<tr>
<td>qt_drift_hold_time</td>
<td>uint8_t</td>
<td>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</td>
</tr>
<tr>
<td>qt_max_on_duration</td>
<td>uint8_t</td>
<td>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)</td>
</tr>
<tr>
<td>qt_neg_drift_rate</td>
<td>uint8_t</td>
<td>Sensor negative drift rate in units of 200 ms. Default value: 20 (20 x 200 ms = 4s per LSB)</td>
</tr>
<tr>
<td>qt_pos_drift_rate</td>
<td>uint8_t</td>
<td>Sensor positive drift rate in units of 200 ms. Default value: 5 (5 x 200 ms = 1s per LSB)</td>
</tr>
<tr>
<td>qt_recal_threshold</td>
<td>recal_threshold_t</td>
<td>Sensor recalibration threshold. Default: RECAL_50 (recalibration threshold = 50 percent of detection threshold)</td>
</tr>
</tbody>
</table>

4.2.7 Touch Sensing Data

The data arrays listed in Table 4-8 are available within the API. These are useful during system development to check that touch sensing is operating as expected.

<table>
<thead>
<tr>
<th>Array</th>
<th>Element Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel_signals[]</td>
<td>uint16_t</td>
<td>The measured signal on each channel</td>
</tr>
<tr>
<td>channel_references[]</td>
<td>uint16_t</td>
<td>The reference signal for each channel</td>
</tr>
<tr>
<td>sensor_deltas[]</td>
<td>int16_t</td>
<td>The signal delta on each sensor (which may comprise multiple channels).</td>
</tr>
</tbody>
</table>

4.2.8 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.
4.2.9 Configuring Sensors

4.2.9.1 Configuration Functions

The functions listed in Table 4-9 are used to assign channels to sensors, and to configure the sensor parameters.

Table 4-9. Typedefs

<table>
<thead>
<tr>
<th>Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_enable_key()</td>
<td>Enable a key sensor.</td>
</tr>
<tr>
<td>qt_enable_rotor()</td>
<td>Enable a rotor sensor.</td>
</tr>
<tr>
<td>qt_enable_slider()</td>
<td>Enable a slider sensor.</td>
</tr>
</tbody>
</table>

4.2.9.2 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”.

4.2.9.3 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.

4.2.9.4 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.

4.2.10 Measuring and Checking The Touch Status

4.2.10.1 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()”. The user application can then perform a touch measurement by calling the function “qt_measure_sensors()”, 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 user application can check the state of the enabled sensors by reading the “qt_touch_status” variable. The user 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.

4.2.10.2 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 user application. These are the “qt_calibrate_sensing()” and “qt_reset_sensing()” functions.

4.2.10.3 qt_init_sensing()

This function initializes touch sensing.

```c
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()”.

4.2.10.4 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.

```c
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.
4.2.10.5 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.

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

4.2.10.6 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.

4.3 User Application Code

4.3.1 User Application Requirements

The library requires the user application to meet the following requirements.

1. 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.

2. The GPIO internal pull-ups must be disabled when calling the library. Setting the “PUD” bit in the “MCUCR” register does this.

3. 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 user application code is not running. There is thus a trade-off between the processor time available to the user application, the power usage of the system, and the system responsiveness.

4. A sufficient stack size for both itself and the library. The user 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 user application can use the unused sensing pins as GPIO. Sensors are numbered in the order in which they are enabled. The library disables interrupts for time-critical periods during touch sensing. These periods are generally only a few cycles long, and so user 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. When building a user application, library functions will only be linked in if they are actually called.
4.3.2 Application Code Flow

Figure 4-1 shows the flowchart for the board firmware.

1. The system initialization consists of initializing the MCU ports by setting appropriately the data direction registers, setting the initial PWM output to inactive high, initializing the global variables, disabling the pull ups, disabling and shutting off any unused resources and initializing the debug interface.

2. The Atmel QTouch library is initialized by enabling keys on the appropriate channels by calling "qt_enable_key()", setting the thresholds, hysteresis, and AKS groups. The keys V+ and V- are assigned to different AKS groups for the MENU function.

3. The ADC measures the internal bandgap voltage of 1.1 V using the reference AVCC which is tied externally to VCC. This is used for measuring the supply voltage for selecting one of the two lookup tables for PWM generation.

4. The watchdog timer is reset and the system reset mode is enabled for 500 ms for taking care of any unhandled exception that may be generated at runtime which causes a blockage or delay larger than 500 ms in the regular operation. After each measurement cycle the normal operation will reset the watchdog thereby continuing operating without reset in case of error free operation.

5. The user application calls “qt_measure_channels()” to measure all the enabled channels.

6. Each key is checked for touch and in case of touch the following state is entered for 30 ms. The PWM output is held at steady LOW and the buzzer is turned ON by generating a 4 KHz square wave.

7. After the 30 ms the current time is updated for the library, buzzer is turned OFF and PWM signal with respect to VCC is generated on the analog output pin corresponding to the key touched, as long as the key is touched.

8. When the key is released or not touched the PWM output is set to inactive state of steady HIGH. The current time is updated for the library.

9. If none of the keys are currently being touched then the watchdog timer is reset and power down sleep mode is entered for 75 ms.

10. On wakeup from power down sleep mode the current time is updated and the same sequences of steps are repeated from step 4.

The following tables 4-10, 4-11, 4-12 and 4-13 list and describe the macro variables, macro functions, external global variables, Atmel QTouch Library APIs and function calls used in the firmware.
Figure 4-1. Flowchart for the Board Firmware

START

System Initialization

Initialize Touch Library

Measure VCC and Select Appropriate Lookup Table for PWM Generation

Reset Watchdog and Enable System Reset Mode at 500 ms

Measure Channels

Are Both V+ & V- Keys being touched?

NO

YES

Is V+ Key being touched?

NO

YES

Is V- Key being touched?

NO

YES

Is ON/OFF key being touched?

NO

YES

Is TV/AV key being touched?

NO

YES

Is PROG+ key being touched?

NO

YES

Is PROG- key being touched?

NO

YES

Reset Watchdog and Enter Power down Sleep Mode for 75 ms

Update Time for Library

Set PWM output to LOW
Turn ON Buzzer
Wait For 30 ms

Update Time for Library

Turn OFF buzzer
Generate PWM output for the touched key with respect to VCC as long as the key is touched

Set PWM output to inactive HIGH

Update Time for Library
### Table 4-10. Macro variables used in the firmware

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNSK</td>
<td>B</td>
<td>The port to be assigned for SNSK electrodes of each channel</td>
</tr>
<tr>
<td>SNS</td>
<td>D</td>
<td>The port to be assigned for SNS electrodes of each channel</td>
</tr>
<tr>
<td>QT_DELAY_CYCLES</td>
<td>5</td>
<td>The dwell time for each burst cycle</td>
</tr>
<tr>
<td>QT_NUM_CHANNELS</td>
<td>8</td>
<td>The number of capacitive measurement channels</td>
</tr>
<tr>
<td>QT_MAX_NUM_ROTORS_SLIDERS</td>
<td>0</td>
<td>Number of rotors and sliders</td>
</tr>
<tr>
<td>POWER_DOWN_PERIOD_MS</td>
<td>75</td>
<td>The power down sleep mode time</td>
</tr>
<tr>
<td>POWER_ON_PERIOD_MS</td>
<td>20</td>
<td>The power ON time between measurements</td>
</tr>
<tr>
<td>QT_NUM_SENSOR_STATE_BYTES</td>
<td>1</td>
<td>Number of bytes required for sensor states</td>
</tr>
<tr>
<td>QT_SNS_PORT</td>
<td>SNS</td>
<td>SNS port for the Atmel QTouch library</td>
</tr>
<tr>
<td>QT_SNSK_PORT</td>
<td>SNSK</td>
<td>SNSK port for the Atmel QTouch library</td>
</tr>
<tr>
<td>DBG_CLK_PORT</td>
<td>C</td>
<td>The debug port to be used for clock</td>
</tr>
<tr>
<td>DBG_DATA_PORT</td>
<td>C</td>
<td>The debug port to be used for data</td>
</tr>
<tr>
<td>DBG_CLK_BIT</td>
<td>5</td>
<td>The bit to be used on the debug port for clock</td>
</tr>
<tr>
<td>DBG_DATA_BIT</td>
<td>4</td>
<td>The bit to be used on the debug port for data</td>
</tr>
</tbody>
</table>

### Table 4-11. Macro Functions used in the firmware

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOIN</td>
<td>Join two tokens to create a variable name</td>
</tr>
<tr>
<td>REG</td>
<td>Create a register name using tokens</td>
</tr>
</tbody>
</table>

### Table 4-12. External Global Variables used in the firmware

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qt_touch_status</td>
<td>The state of library sensors</td>
</tr>
<tr>
<td>channel_references</td>
<td>The reference signal for each channel</td>
</tr>
<tr>
<td>sensor_deltas</td>
<td>The signal delta on each channel (which may comprise multiple channels)</td>
</tr>
</tbody>
</table>

### Table 4-13. APIs and Function Calls used in the firmware

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_system</td>
<td>Initialize host app, pins, watchdog, etc.</td>
</tr>
<tr>
<td>init_debug_if</td>
<td>Initializes the debug interface</td>
</tr>
<tr>
<td>qt_init_globals</td>
<td>Initializes the global threshold parameters</td>
</tr>
<tr>
<td>qt_enable_key</td>
<td>Enable a key sensor</td>
</tr>
<tr>
<td>qt_init_sensing</td>
<td>Initializes touch sensing</td>
</tr>
<tr>
<td>qt_measure_sensors</td>
<td>Performs capacitive measurements on all enabled sensors</td>
</tr>
<tr>
<td>report_debug_data</td>
<td>Reports the debug data to the host</td>
</tr>
<tr>
<td>output_to_debugger</td>
<td>Transmits multiple bytes over the debug interface</td>
</tr>
<tr>
<td>send_debug_byte</td>
<td>Sends one byte over the debug interface</td>
</tr>
</tbody>
</table>
4.4 Adding the Library to Application Code

1. Open IAR® Embedded Workbench and open the “t88_8qt_example_chip” workspace.
2. Copy the library into the project root directory. The library object file has the extension “r90”.
3. Right click on the project and on the shortcut menu, select Add, and then select Files.
4. Select the “r90” file for the library.
5. Change the main file as per the host requirement and compile to create the hex file.
5 Appendix

On the next pages the following documents are shown

- Schematics
- Assembly drawing
- Board Layout
- Bill of material

5.1 Schematics

Figure 5-1. Schematics
5.2 Assembly Drawing

Figure 5-2. Component Assembly (Bottom Side)

5.3 Board Layout

Figure 5-3. Top Layer

Figure 5-4. Bottom Layer
## 5.4 Bill of Material

### Table 5-1. Bill of Material

<table>
<thead>
<tr>
<th>SL NO</th>
<th>Description</th>
<th>Vendor</th>
<th>Vendor Part No</th>
<th>Designator</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RES SIN 1K 1% 0603 50V</td>
<td></td>
<td></td>
<td>R1-R6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>RES SIN 330R 5% 0603 50V</td>
<td></td>
<td></td>
<td>R7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>RES SIN 10K 1% 0603 50V</td>
<td></td>
<td></td>
<td>R9</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>RES SIN 4K7.5% 0603 50V</td>
<td></td>
<td></td>
<td>R10</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>RES SIN 22R1 1% 0603 50V</td>
<td>NXP</td>
<td></td>
<td>R11</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>BJT BC847B NPN SOT23</td>
<td>NXP</td>
<td>BC847B</td>
<td>Q1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1 Amp General Purpose rectifier</td>
<td>FAIRCHILD</td>
<td>BAS19</td>
<td>D1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>CAP CER 22nF 10% 0603 X7R 50V</td>
<td></td>
<td></td>
<td>C1, C2, C6</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>CAP CER 47nF 10% 0603 X7R 50V</td>
<td></td>
<td></td>
<td>C3-C5</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>CAP CER 4u7F 10% 0805 X5R 16V</td>
<td></td>
<td></td>
<td>C7</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>CAP CER 0u1F 10% 0603 X7R 50V</td>
<td></td>
<td></td>
<td>C8, C9</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>CAP CER 1uF 10% 0603 X7R 16V</td>
<td></td>
<td></td>
<td>C10</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>CAP CER 0u1F 10% 0603 X7R 50V</td>
<td></td>
<td></td>
<td>C11</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>BUZ 4KHz +/- 0.5 3 Vp-p</td>
<td>Bosan Hitech</td>
<td>BST-5523SA</td>
<td>BZ1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>CON 8P 1.25 mm pitch</td>
<td>MOLEX Inc.</td>
<td>53398-0819</td>
<td>J1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>ATtiny48/88 32pin MLF</td>
<td>Atmel</td>
<td>ATtiny48/88</td>
<td>IC1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>Ferrite Bead 0E</td>
<td></td>
<td></td>
<td>L1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>27</strong></td>
<td></td>
</tr>
</tbody>
</table>
EVALUATION BOARD/KIT IMPORTANT NOTICE

This evaluation board/kit is intended for use for FURTHER ENGINEERING, DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY. It is not a finished product and may not (yet) comply with some or any technical or legal requirements that are applicable to finished products, including, without limitation, directives regarding electromagnetic compatibility, recycling (WEEE), FCC, CE or UL (except as may be otherwise noted on the board/kit). Atmel supplied this board/kit “AS IS,” without any warranties, with all faults, at the buyer’s and further users’ sole risk. The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies Atmel from all claims arising from the handling or use of the goods. Due to the open construction of the product, it is the user’s responsibility to take any and all appropriate precautions with regard to electrostatic discharge and any other technical or legal concerns.

EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER USER NOR ATMEL SHALL BE LIABLE TO EACH OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

No license is granted under any patent right or other intellectual property right of Atmel covering or relating to any machine, process, or combination in which such Atmel products or services might be or are used.

Mailing Address: Atmel Corporation, 2325 Orchard Parkway, San Jose, CA 95131

Copyright © 2008, Atmel Corporation