AVR1016: AVR Training QTouch Xplained

Prerequisites

- **Knowledge level: Intermediate**
- **PC Platform: Windows® 2000, XP, Vista**
- **Hardware requirements:**
  - **JTAG ICE mkII**
  - **Xplain Kit**
  - **Atmel® QTouch® Xplained board**
- **Software requirements:**
  - **Atmel® AVR Studio® 4 (latest version)**
  - **(Atmel QTouch Studio, Optional)**
  - **Latest version of WinAVR**
  - **Atmel QTouch® Library (latest version)**
- **Time consumption: ~3 hours**

Introduction

The purpose of this training is to get familiar with the Atmel QTouch Library 4.0 and how to use it in your own applications.

This hands-on consist of a number of tasks. The first task shows how to connect everything and assure that your hardware is working properly. The second task explains how to set up a project and configure simple Atmel QTouch buttons. The next task show how to configure a more advanced sensor and how to use the debug interface in order to take advantage of the live debugging features of QTouch Studio.

Basics for Atmel AVR Studio, WinAVR or tools use are not covered.

8-bit AVR® Microcontrollers

Application Note

Rev. 8304A-AVR-04/10

Document Overview

The hands-on is split into several different assignments. Each assignment is further divided into individual sections to simplify understanding.

Throughout this document you will find some special icons. These icons are used to identify different sections of assignments and ease complexity.

 Information
Delivers contextual information about a specific topic

 Tip
Highlights useful tips and techniques

 **To do**
Highlights objectives to be completed

 **Result**
Highlights the expected result of an assignment step

 **Warning**
Indicates important information

Table of Contents

1 Requirements

1.1 Software

Make sure you install the latest versions of

- AVR Studio
- Atmel QTouch Studio 4.0.1 or newer (optional)
- WinAVR GCC
- Atmel QTouch Library

1.2 Hardware

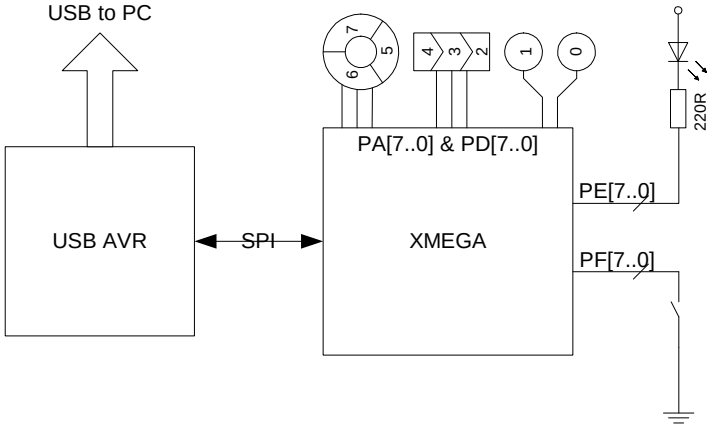
In this hands-on we are going to use:

- Xplain kit w/ USB cable
- QTouch Xplained top card
- JTAGICE mkII w/ USB cable

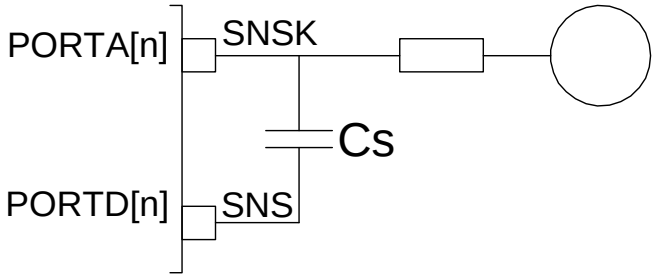
1.3 Xplain board

The Xplain kit is a small simple training kit that was designed to allow designers a simple way to test the ATxmega128A1.

1.3.1 Xplain board Schematics



Pin schematics:



2 Assignments

2.1 Task 1: Connecting tools and run a test project

2.1.1 Introduction

This task is designed to make sure all the tools are connected and working as intended. No code writing is needed for this task.

2.1.2 Tasks

Preparing software

1. Install the software required for this training
    a. Atmel AVR Studio (if you do not have the latest version)
    b. Atmel QTouch Studio (optional)
    c. Atmel QTouch Libraries
    d. WinAVR
2. Extract the file archive attached to this training document to a local folder.

Flashing the Xplain board

1. The bridge chip firmware needs to be programmed to the AT90USB1287 device on the Xplain board. This can be done in two ways:
    a. Program the AT90USB1287 with a JTAGICEmkII programmer. Connect the programmer to the port labeled `JTAG USB` on the Xplain board. Locate the file `USB_Bride_V3.3_646.a90` in the `Firmware` folder and write this file to the USB device. Warning: If you program the USB bridge firmware using the JTAG, it will override the bootloader. It advised that you make a backup of the original program.
    b. Program the AT90USB1287 using the Atmel Flip software and a bootloader. If your USB device has a bootloader present, which it should by default, you can program it without using a programmer. Instead you can use Atmel Flip which can be downloaded for free from the Atmel website. Before launching Flip, be sure to short out pins 1 and 2 on the `JTAG USB` header, this will enable the bootloader. Locate the file `USB_Bride_V3.3_646.a90` and use Flip to write this file to the USB device. This will preserve the bootloader.
1. Now the Atmel® AVR® XMEGA™ device on the Xplain needs to be flashed with the example application firmware. Connect your programmer to the ATxmega128A1 via the `JTAG&PDIXMEGA` port, then program it with the file `Example.hex`. Note that this step requires a programmer and can not be done with a bootloader.

dow

Connect QTouch Xplained board and PC

1. Connect the QTouch Xplained board to the Xplain. Make sure the orientation is correct, in which case the print on both board should be facing the same direction. The header on the underside of the top of the QTouch Xplained board is the JTAG header connected to the AVR XMEGA device.
2. Power the board by connecting the USB interface, and start QTouch Studio,
3. Try pushing the buttons, slider and wheel and verify that the lights on the Xplain board respond correctly.
4. Launch QTouch Studio and confirm that the graphical interface reports live debug data from the device.
5. If the data updates your kit is working and you can continue on to the next tasks.

2.2 Task 2: Simple Buttons

2.2.1 Introduction

In this task we will take a closer look at QTouch Buttons and configuring a project from scratch. We will start with opening the Task2 project (in the task2 folder). This project contains a very simple program that simply sets system frequency and launches into an infinite empty loop. There are no QTouch references anywhere in this project, but we are going to set it up to use two touch keys to blink the Xplain lights.

Check if the QTouch Library files are present

Before you start this task, copy the following files from the QTouch Library installation to the project folder:

`libavrxmega7g1-8qt-k-0rs.a`
`touch_api.h`
`touch_qt_config.h`
`qt_asm_xmega.s`

2.2.2 Setting up the code

Load the project file

First, we start AVR Studio and load the project associated with task 2. Take a minute to look at the code and familiarize yourself with the simple features. Notice how there is no code directly related to touch sensing yet.

Include QTouch Library header, binary and source

1. To use the touch keys, first we need to add `#include “touch_api.h”` to our list of include files. This will make the QTouch Library API available.
2. Now that we have the API, we need a reference to the library file containing the precompiled QTouch code. Open the **Project Options**, and under **Libraries** select the library file *libavrxmega7g1-8qt-k-0rs.a* and click **Add Library**.
3. Lastly we are going to need to add the file `qt_asm_xmega.s` to the project. This is a special assembly source file that contains optimization routines that need to be compiled with our project. In the AVR GCC tab in AVR Studio, right click the project and select *Add Existing File*, then select the assembly source file. Alternatively, you can drag and drop the source file onto the project name.

Configure the QTouch Library

The QTouch library needs a few pre-processor defines declared in order to function correctly. These needs to be set for all files, so a good place to enter them is in the compiler options in the Project Options menu. For simplicity the correct settings have been entered into the project already, as a build configuration. To be sure they are set

correctly open the Project Options window and in the General tab, assure Active Configuration is set to "*avrxmega7g1_8qt_k_0rs*".

Then switch to the Custom Options tab and have a look at the global compiler flags. The following flags should be present:

```
-D_QTOUCH_
-DQT_NUM_CHANNELS=8
-DSNSK1=A
-DSNS1=D
-DQT_DELAY_CYCLES=1
```

This tells the library a few things: We are going to use QTouch measurement technology (as opposed to QMatrix®). We are going to have a maximum of 8 channels. There will only be one delay cycle, and measurements should be done on port A and D. The exact meaning of some of these parameters is not too important right now, but if you want further information, please see the online QTouch User Guide.

These settings need to be entered into any project using the QTouch Library.

Adding more QTouch Library configuration

Next we need to set up some runtime global QTouch parameters. We have provided a function in which this should be done, `init_qt_globals( )`. Find this function and add the following code to it:

```
qt_config_data.qt_di              = DEF_QT_DI;
qt_config_data.qt_neg_drift_rate  = DEF_QT_NEG_DRIFT_RATE;
qt_config_data.qt_pos_drift_rate  = DEF_QT_POS_DRIFT_RATE;
qt_config_data.qt_max_on_duration = DEF_QT_MAX_ON_DURATION;
qt_config_data.qt_drift_hold_time = DEF_QT_DRIFT_HOLD_TIME;
qt_config_data.qt_recal_threshold = DEF_QT_RECAL_THRESHOLD;
```

Exactly what these parameters specify isn't important right now; we will go into further detail later. A tip here would of course be to copy paste the code from the block above into your project.

Now it's time to configure our first button.

We use the following API call to configure a button:

```
qt_enable_key( CHANNEL_0, AKS_GROUP_1, 10u, HYST_6_25 );
```

Add this after the `init_system` call, where it says "enable sensor 0". This tells QTouch that channel 0 should be treated as a key. We will look at the last three parameters a little later.

Initialize the library

After this initial configuration, we need to initialize the QTouch library. This is done by simply calling `qt_init_sensing()`. Add a call to this function under the label "initialize touch sensing".

Somewhere to store time

We are now almost ready to start taking measurements. A single measurement is taken by calling `qt_measure_sensors(uint16_t current_time_ms)`. But as you can see, it expects an argument. This argument is used to keep track of time to allow compensating for over-time drifting in the measured values. *If you are not concerned with drifting, you can pass this argument as zero*.

For our application, we are going to add a simple counter to keep track of time, so declare the following variable under *static variables*:

```
static volatile uint16_t current_time_ms_touch = 0u;
```

We are now ready to perform a touch measurement

Inside the main loop, under “measure touch sensors”, add a call to:

```
qt_measure_sensors(current_time_ms_touch);
```

The way it’s currently implemented, the counter will always remain at zero, but that’s okay for now.

Slow the main loop down

We have now successfully polled the touch sensors! However, the loop will keep polling over and over at full speed. This is not necessary, and so we can slow things down considerably. Inside the main loop, let’s add an idle delay:

```
_delay_ms( 10 );
current_time_ms_touch += 10;
```

As you can see, this is also a convenient point to increment the timer counter as well.

We poll the touch keys every 10ms (100Hz), which is pretty fast. 25ms is more common.

Note: In more complex applications, touch measurements are most likely triggered by a timer. We use a busy loop here for simplicity.

Read key states from touch data

We should now have valid touch data available, updated every 10ms. The on/off pressed state of our button can be found by reading this structure:

```
char key_states =
     qt_measure_data.qt_touch_status.sensor_states[0];
```

Add this to the main loop under “read key states”, and `key_states` will get populated with the binary states of the eight first sensors. We only enabled one sensor, the key, so its state should be bit 0 in `key_states`.

Adding a simple application

Let’s add some code to blink the lights when you press the key. Add this to the main loop after `key_states` is read:

```
/*Turn all lights off*/
PORTE.OUT = 0xFF;
if(key_states & 1){
      /*Turn Left Side lights on*/
      PORTE.OUT = 0xF0;
}
```

This should toggle the left row of lights on when you press the key, which by the way is key 0 and is the *upper most* of the two keys on the QTouch Xplained board.

Optional: Add a second key

Try and see if you can add a second key, on channel one. Here is some code to turn on the right side lights, it might be useful:

```
/*Turn Right Side lights on*/
PORTE.OUT = 0x0F;
```

2.2.3 Further explanation

Most of the function calls used in this task are quite simple and self explaining. The most complex line is the qt_enable_key() function, so lets look closer at this:

```
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.

And AKS group, or Adjacent Key Suppression group, is a group of keys or sensors that can not be in detect at the same time. So if two keys are close to each other on your PCB, and you don't want accidental presses, it might be a good idea to put these two keys in the same AKS group. The first key pressed in a group will remain pressed.

2.3 Task 3: Adding Slider and Debug Interface

2.3.1 Introduction

This task will show you how to set up a slider and a debug interface to output touch data and statistics to AVR Touch Studio. If you are not interested in a debug interface, you can safely skip that portion of this task.

In order to properly connect with AVR Touch Studio, the Xplain AT90USB1287 device needs to be flashed with the USB Bridge Chip firmware. This firmware should be supplied with the training files.

Check if the QTouch Library files are present

Before you start this task, make sure the following two files are present in the project folder:

```
libavrxmega7g1-8qt-k-2rs.a
touch_api.h
touch_qt_config.h
qt_asm_xmega.s
```

If they are not there, copy them from your QTouch Library installation.

2.3.2 Adding a slider

Sliders are a more sophisticated form of sensors than simple keys. They often take the form of a linear track on the layout, where the user can move his or her finger up and down to change the slider *value*. The slider value can have varying resolution. Our example slider will be 8 bit, and so have 256 different values.

A slider uses several channels. The slider on the QTouch Xplained board uses three; channel 2, channel 3 and channel 4.

To start this task, load up the task3 project file.

Once loaded, open then Project Options and find the Libraries tag. Note that, unlike what we did in task 2, we now loaded the *k-2rs* version of the QTouch library file. This means that the library supports Keys, and two Rotors or Sliders, not just keys like in task 2. The k-2rs library is slightly larger than the k-0rs library.

Take a minute to go over the new code till you feel familiar with it.

This project is very similar to what we set up in task 2, but there are some noteworthy differences. For instance, now we use a 25ms interval timer and a flag to figure out when to execute measurements.

There is also a lot of code referring to the debug interface, some of it is commented out. Don't mind this too much yet, we will look at that later.

Compile and test the code

The two keys should be active; the upper key should light the four topmost lights. And the lower key controls the four lower lights.

Configuring a slider is pretty easy; we just need to add one line.

Under the label “enable sensor 2”, add the following code:

```
qt_enable_slider(  CHANNEL_2,  CHANNEL_4,  AKS_GROUP_1,  16u,
HYST_6_25, RES_8_BIT, 0u  );
```

This will enable a slider using channels 2, 3 and 4. Notice the `RES_8_BIT` argument, this is the slider resolution. The other arguments are not too important right now.

That’s it. The slider will now be polled and calculated at the same time as the keys.

Using the slider value in an application

Unlike the keys, the slider also has a scalar *value*, in addition to its boolean pressed/not pressed state. Our slider can have any value from 0 to 255. In applications this can be used for almost anything, and it’s easy to suggest features like volume control. If you want to get more advanced, you can record the value when the user pressed down, and do relative scaling up or down from that point.

We are going to do something much simpler for this example: We are going to move a light up and down.

Add the following code after the two IF-statements that toggle the lights:

```
if(s_states & (1<<2)){
      uint16_t sliderval = qt_measure_data.qt_touch_status.
                           rotor_slider_values[0];
      val = 1<<(sliderval/32);
}
```

As you can see, we first check the sensor state for sensor number 2 (the keys are sensor 0 and 1 respectively) to see if it is detect, which means there is a touch on the slider.

If so, we read the current slider value from the `qt_measure_data`. The slider is the first rotor or slider, so it has index 0 in the `rotor_slider_values` array.

We then do some bit-shifting magic to light up the correct light on the Xplain board.

Optional: Fix the direction of the lights

When sliding the slider, the lights go the opposite way of what would be intuitive. This is because zero on the slider is on the top.

Can you fix this with some simple code?

Optional: Add a rotor

A Rotor is basically a slider, only wrapped around in a circle. Channels 5 through 7 on the QTouch Xplained board are mapped to the rotor. Can you add some code to enable the rotor too? It is almost identical to how we initialized the slider; we just substitute “slider” for “rotor”. If you wonder how it’s done, you can always look at the example project.

2.3.3 Using the Debug Interface

The debug interface is used to communicate various touch sensor information to a computer running AVR Touch Studio. The debug interface as seen from the XMEGA is implemented completely in firmware, and the code can be found near the end of the main file. If you investigate the code you will see it is a very simple serial transmission.

Creating holders for the debug information

The first step in implementing the debug interface is to create some variables to hold the debug data before transmission and to configure our layout, so we can tell AVR Touch Studio what keys, rotors, or sliders we are using.

Under *global variables*, we add these two declarations:

```
uint8_t sensor_config[QT_NUM_CHANNELS];
board_info_t board_info;
```

`sensor_config` will, not surprisingly, hold information on how the sensors are configured, and `board_info` will hold general information about the board.

Tell the debug interface how the sensors are configured

AVR Touch Studio needs to know how our channels are configured: Which are in use, and which make up what sensors? We’re going to start by configuring the data for sensor 0, which in our case is the upper key, which is channel 0.

Under the label “configure the debug data”, we add the following line:

```
sensor_config[0] =
  SENSOR_CONFIG( CHANNEL_0, CHANNEL_0, SENSOR_TYPE_KEY );
```

We use the macro `SENSOR_CONFIG` to assign a value to index 0 of the `sensor_config`, this corresponds to sensor 0.

The parameters above mean that we have a *key* type sensor, using the channels starting at channel 0 and ending on channel 0.

Add a line for configuring the second key. Remember to now use `CHANNEL_1`, and store the result in `sensor_config[1]`.

Sensors are numbered in the order they are configured, so be mindful about this. The order of your `qt_enable_key` and `qt_enable_slider` calls matter.

The last sensor we want to configure is sensor number 2: the slider. We configure it by adding this line:

```
sensor_config[2] =
  SENSOR_CONFIG( CHANNEL_2, CHANNEL_4, SENSOR_TYPE_SLIDER );
```

Notice how the type now changed, and that the start and end channels implicate all three channels the slider uses.

It is good practice to explicitly tell the debug interface about the sensors not in use. Our board has eight channels, so a max of eight sensors (if all were keys). Add these lines to configure the remaining sensors as inactive:

```
sensor_config[3] =
 SENSOR_CONFIG( CHANNEL_0, CHANNEL_0, SENSOR_TYPE_UNASSIGNED );
sensor_config[4] =
 SENSOR_CONFIG( CHANNEL_0, CHANNEL_0, SENSOR_TYPE_UNASSIGNED );
sensor_config[5] =
 SENSOR_CONFIG( CHANNEL_0, CHANNEL_0, SENSOR_TYPE_UNASSIGNED );
sensor_config[6] =
 SENSOR_CONFIG( CHANNEL_0, CHANNEL_0, SENSOR_TYPE_UNASSIGNED );
sensor_config[7] =
 SENSOR_CONFIG( CHANNEL_0, CHANNEL_0, SENSOR_TYPE_UNASSIGNED );
```

Hint: Copy & paste this code.

We have only been configuring the debug interface now; this has *nothing* to do with how the sensors actually work.

Configure the board information

We need to add some more information about our board to the debug system. This is done so AVR Touch Studio can recognize our board and display the appropriate layout for our QTouch Xplained board.

Right after the sensor configurations we add:

```
board_info.qt_max_num_rotors_sliders_board_id =
    ( ( QT_MAX_NUM_ROTORS_SLIDERS << 4 ) | BOARD_ID );
board_info.qt_num_channels = QT_NUM_CHANNELS;
```

Start sending debug reports

Now that we've completed the debug configuration, we are ready to start sending reports. One report is sent by calling `report_debug_data( )`, so add a call to this right after we've measured the sensors, where it says "report debug data back to host".

Sending a debug report can take a little time, so it should only be done *once* per touch measurement. Be therefore absolutely sure that your `report_debug_data` call is inside the same IF-block as your `qt_measure_sensors`, and that it's after the measurement call itself.

Before you compile, make sure to uncomment the `report_debug_data` function.

This is done by removing the `/* comment block */` surrounding it. If this is not done correctly you will get the error “undefined reference to `report_debug_data'”.

That’s it; we’re done configuring the debug interface. Compile and run your code.

Start AVR Touch Studio, and confirm the received data

If everything was set up correctly, QTouch Studio should now be reporting live data from the device.

Congratulations! You are now ready to start developing and debugging your own QTouch applications.

3 Summary

The main goal with this hands-on is to learn how to use set up and use the QTouch Library.

- How to initialize channels
- How to use a precompiled library
- How to use and “tune” sliders and wheels

In addition to all of this you have learned how to use the AVR Studio GUI to write C code, debug your code and program the device.

4 Atmel Technical Support Center

Atmel has several support channels available:

o	Web portal:	http://support.atmel.no/	All Atmel microcontrollers
o	Email:	avr@atmel.com	All AVR products
o	Email:	avr32@atmel.com	All AVR32 products

Please register on the web portal to gain access to the following services:

- o Access to a rich FAQ database
- o Easy submission of technical support requests
- o History of all your past support requests
- o Register to receive Atmel microcontrollers’ newsletters
- o Get information about available trainings and training material

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature