
Atmel AVR8015: QT600 - AT32UC3L QMatrix Training Guide



Features

- Knowledge level: Intermediate
- PC platform: Windows® 2000, Windows XP, Windows Vista®, Win7
- Hardware requirements:
 - Atmel® QT™600 (entire kit)
 - USB cables
- Software requirements:
 - Atmel AVR® QTouch® Studio 4.3.1 or later
 - Atmel QTouch Library 4.4 or later
 - Atmel AVR Studio® 5 (latest version)
- Estimated time to complete all tasks in this guide: 1 day

1 Introduction

The purpose of this training is to get familiar with the [Atmel AVR QTouch Suite](#) for developing and debugging any Atmel Touch application. It includes four core solutions: [Atmel QTouch Studio](#), [Atmel QT™600 Development Kit](#), [Atmel QTouch Library](#), and [Atmel AVR Studio 5](#).

Atmel QTouch Studio - Touch Analyzer

QTouch Studio is the front-end software used to display and evaluate the data reported by the QT600 development kit.

Atmel QTouch Library - Touch SW library for AVR

The QTouch Library is a software library for developing touch applications on standard Atmel AVR microcontrollers. Customers can link the library into their firmware in order to integrate touch-sensing capability into their projects.

Atmel AVR Studio 5 - Debugger and Programmer

AVR Studio 5 is a professional Integrated Development Environment (IDE) for writing, simulating and debugging applications for AVR microcontrollers. It also comprises the programming interface for all AVR tools. It has the following features:

- Supports all 8- and 32-bit AVR
- Integrated C compiler
- New project wizard and Intelligent Editor

Atmel QT600 - Touch Hardware Kit and example Code

The QT600 is a complete touch development kit for buttons, sliders and wheels. This advanced development platform allows designers to experiment with Atmel touch technology, and provides the easiest way to analyze and validate touch products. It supports both Atmel QTouch and Atmel QMatrix acquisition methods. It comes with one USB-powered interface board, MCU boards representing the Atmel tinyAVR®, Atmel megaAVR®, Atmel AVR XMEGA® and 32-bit Atmel AVR UC3 families of microcontrollers, and three touch sensor boards supporting up to 64-channels.

32-bit Atmel Microcontrollers

Application Note

Rev. 32185A-AVR-01/12





The Atmel QT600 MCU boards can be connected to the Atmel STK[®]600 for easy access to unused I/O-pins. STK600, however, is not needed for the QT600 kit to function properly. QT600 is fully supported by [Atmel AVR QTouch Studio](#), [Atmel QTouch Library](#), and [Atmel AVR Studio 5](#), and together these tools form the [Atmel QTouch Suite](#).

This hands-on training consists of a number of tasks.

1. Connect everything and assure that your hardware is working properly.
2. Set up a project from scratch, select and include the library, add the touch files and ASF driver files.
3. Create a virtual kit in design mode and use the pin configuration wizard.
4. Configure the global options and set/enable the sensors.
5. Add debug files and code to use the debug interface in order to take advantage of the live debugging features of QTouch Studio.
6. Use the Analysis mode and a review brief description of the various parameters.
7. Use the design validation wizard.
8. Log data and use it for further analysis and debugging purposes.

Basics of [AVR Studio 5](#), or tools use are not covered.

2 Document overview

The hands-on training 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.

3 Requirements

3.1 Software

Make sure you install the latest versions of:

- Atmel QTouch Studio 4.3.x (www.atmel.com/avrqtouchstudio)
- Atmel QTouch Library 4.4 (www.atmel.com/qtouchlib)
 - QTouch Library User Guide
 - QTouch Library Selection Guide
- Atmel AVR Studio 5 – Latest (www.atmel.com/avrstudio)

3.2 Hardware

In this hands-on training we are going to use:

- Atmel QT600 Interface Board
- Atmel QT600 – AT32UC3L-QM64
- Atmel QT600 – QMatrix 8x8 Touch Sensor board
- Debugger – Atmel AVR JTAGICE mkII with squid cable

NOTE

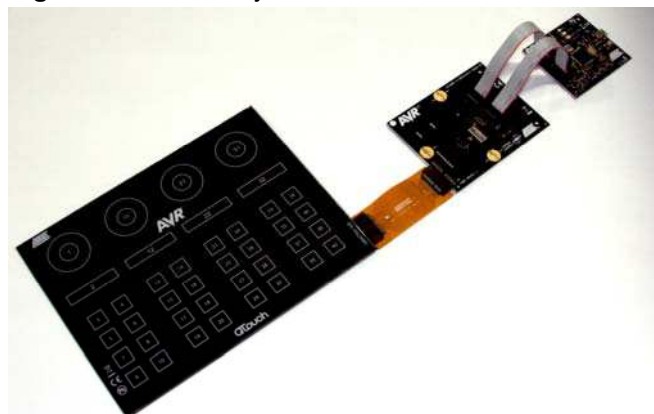
AVR JTAGICE mkII with hardware revision 0 does NOT have aWire capabilities. Hence AVR JTAGICE mkII other than hardware revision 0 should only be used. Any of the following debuggers maybe also used Atmel JTAGICE 3, Atmel STK600 and Atmel AVR ONE!.

3.3 Atmel QT600 system description

The QT600 system is based on three boards connected together.

- QT600 Interface Board
- QT600 MCU Board
- Touch Sensor Board

Figure 3-1. QT600 system.



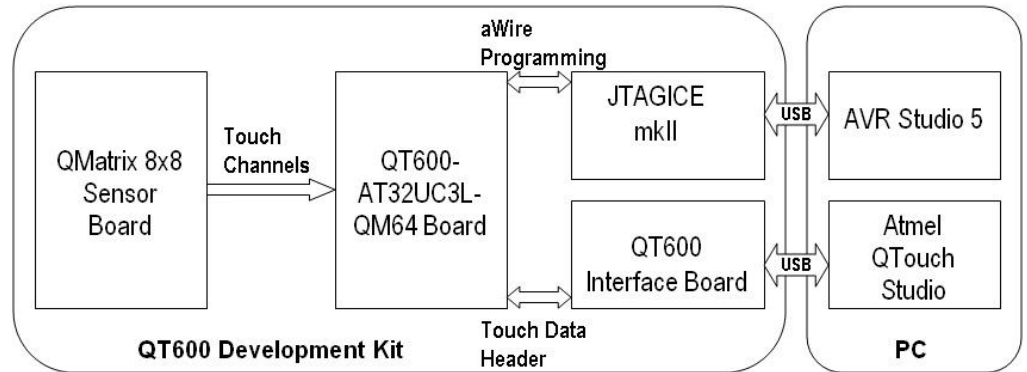
The MCU board and touch sensor boards together comprise the user touch system. The QT600 interface board is used to stream live touch data from the Atmel AVR



MCU mounted on the MCU board. Atmel QTouch Studio is used as the PC front end to visualize the touch data.

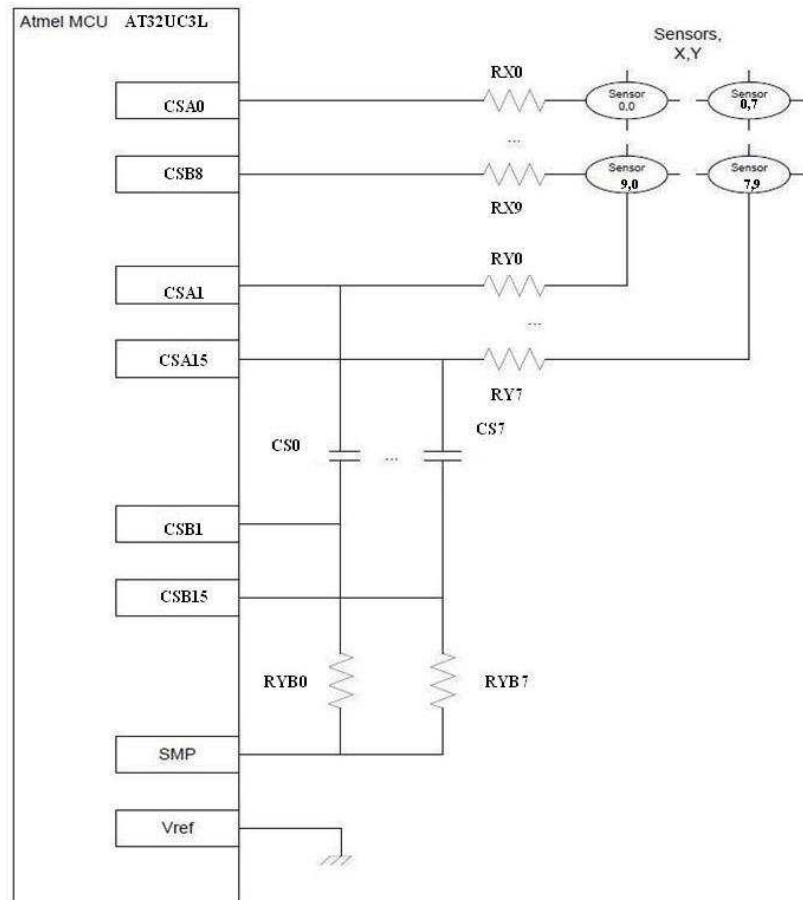
3.3.1 Block diagram

Figure 3-2. Block diagram.



3.3.2 Pin schematic

Figure 3-3. Pin schematic.



4 Assignment

For the following tasks, please note:

- Start with all the cables disconnected (PC to Atmel QT600, QT600 to MCU boards...)
- Ensure you have installed the latest versions of software listed in Chapter 1
- From the same webpage where you found this application note download and expand the associated zip file (disk icon). Project folders for each task will be relative to where you expanded the zip file {AVR8015}. File paths will be relative to the expanded task folder where each project should be created
- The zip file consists of Completed task and Exercise folders. Completed task folder is for user reference, and it consists of the step by step solution of the training. If the user intends to work on only a particular task, then the project files inside Exercise folder can be used
- User can also create his own project by following the steps described in the assignment section
- All the library files and the necessary support files required for these tasks are available inside Exercise folder

4.1 Task 1: Connect the tools and run a test application

4.1.1 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. You can load the Hex file `uc31_gnu_qm_qt600.hex` present at the location:

```
..\..\Exercises\QMatrix\Tasks\Task1\
```

This can be programmed directly into the flash of the MCU. The Atmel QT600-AT32UC3L-QM64 MCU board is configured for aWire programming. The QT600 interface board does not support aWire programming. It supports JTAG, ISP, TPI and PDI modes only. Hence we need to use a programmer like the Atmel AVR JTAGICE mkII (other than revision 0), Atmel JTAGICE 3, Atmel STK600, or Atmel AVR ONE!, which supports aWire programming. In this hands-on training we are using AVR JTAGICE mkII.

To use the AVR JTAGICE mkII for programming the QT600-AT32UC3L-QM64 MCU board: Make the following connections between the squid cable and the aWire header on the MCU board.

Table 4-1. JTAGICE mkII aWire connections.

JTAGICE mkII squid cable	MCU board aWire header
GND (white)	Pin 2
VTREF (purple)	Pin 4
TDI (red)	Pin 6

Connect the QT600 TOUCH DATA header to the MCU board TOUCH DATA header. This is required to power the device (supply VTG), since the AVR JTAGICE mkII does not power the device. Keep the QT600 interface board in the default data streaming mode.

The Touch Data LED on the QT600 indicates mode:



- Touch Data LED green: Touch Debug mode (default mode after power-up)
- Touch Data LED off: Programmer mode

Launch the AVR Programming tool on Atmel AVR Studio 5 and program the device. Use AVR Studio 5.0 or later.

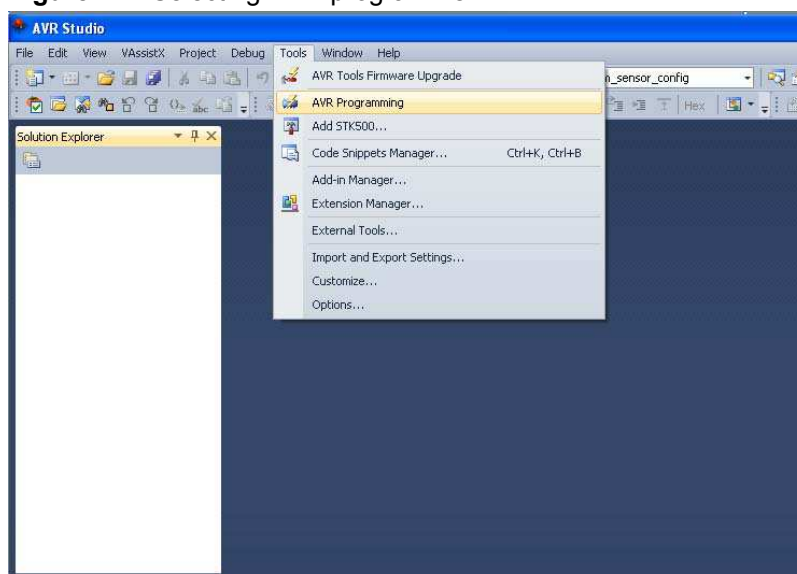


Programming output file using AVR Studio 5.

.Hex file or .Elf file can be loaded into the target board by following the below steps.

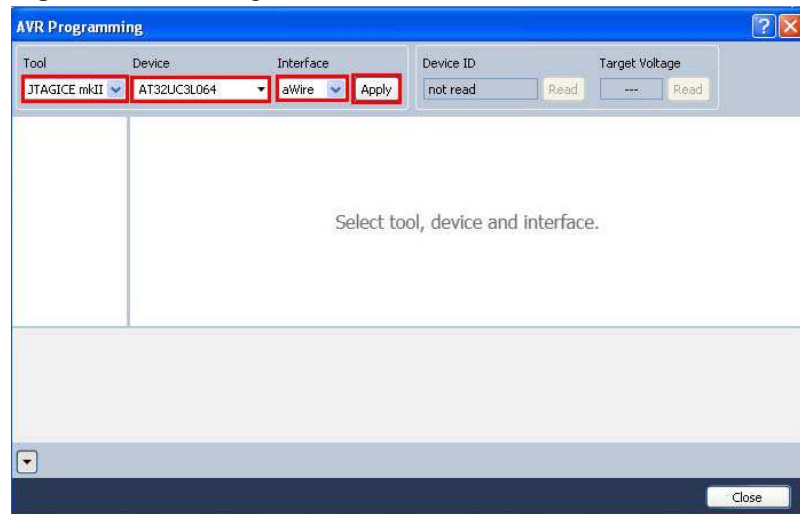
1. In studio5 IDE, select and click on Tools → AVR Programming option as shown in [Figure 4-1](#).
2. Switch the device to Programmer mode as described in [Section 4.1.1](#).

Figure 4-1. Selecting AVR programmer.



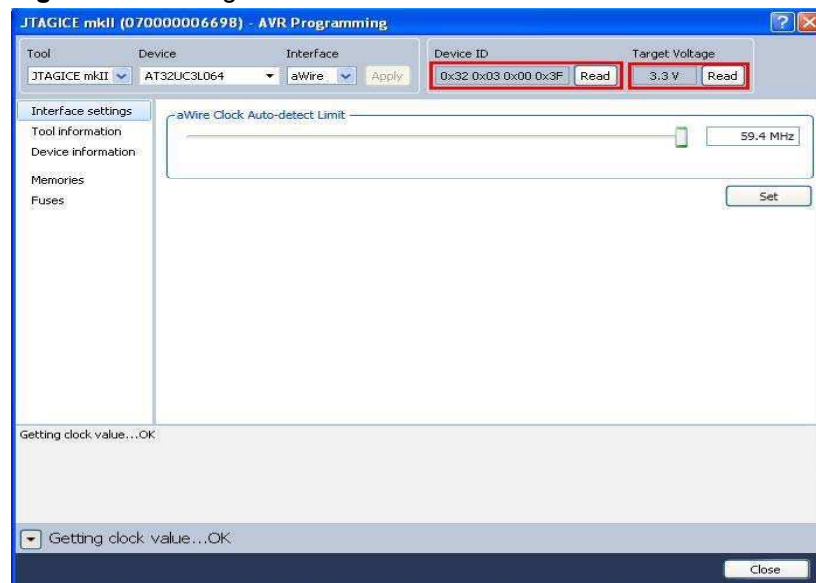
3. Set the target hardware under tools drop down menu, proper device under device icon and interface type under interface icon, and click on apply button as shown in [Figure 4-2](#).

Figure 4-2. Selecting the device.



4. Click on Device ID Read option and also on Target Voltage Read option. Ensure that the target voltage as per the device datasheet. Minimum voltage is 1.8V. Also ensure that Device ID and target voltage are read by the IDE as shown in Figure 4-3.

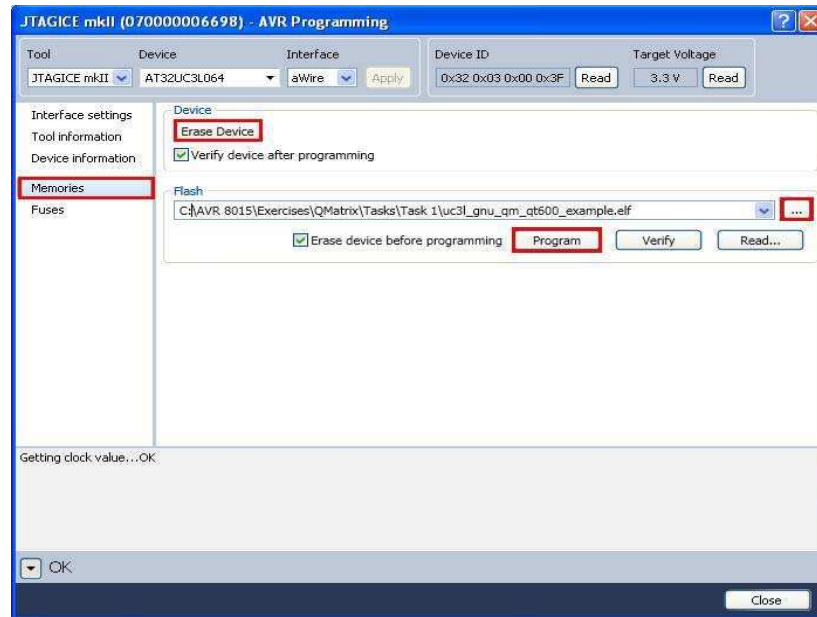
Figure 4-3. Setting device ID.



5. Select Memories option, click on Erase Device.
6. Set the appropriate path for the output file and click on program option as shown in Figure 4-4.



Figure 4-4. Writing output file into the target device.



4.1.2 Connecting the hardware

1. Connect the USB cable to the Atmel QT600 interface board.
2. Make sure the VTG header is mounted on the QT600 interface board.
3. Connect the TOUCH DATA header of the QT600 interface board and Atmel QT600-AT32UC3L-QM64 board.
4. Connect the JTAGICE mkII and the QT600-AT32UC3L-QM64 board through the aWire interface as described in Section 4.1.1.

Figure 4-5. Squid connection.



5. Make sure that the QT600 interface board is in the default data streaming mode.
6. Ensure the target voltage is set as per the device datasheet. Minimum voltage is 1.8V.
7. Program the application code (`uc3l_gnu_qm_qt600.hex`). For detailed instructions on how to do this, see the help section in [Atmel AVR Studio 5](#).
8. Disconnect the USB plug from the Atmel QT600 interface board.

9. Remove the squid cable between the programming headers.

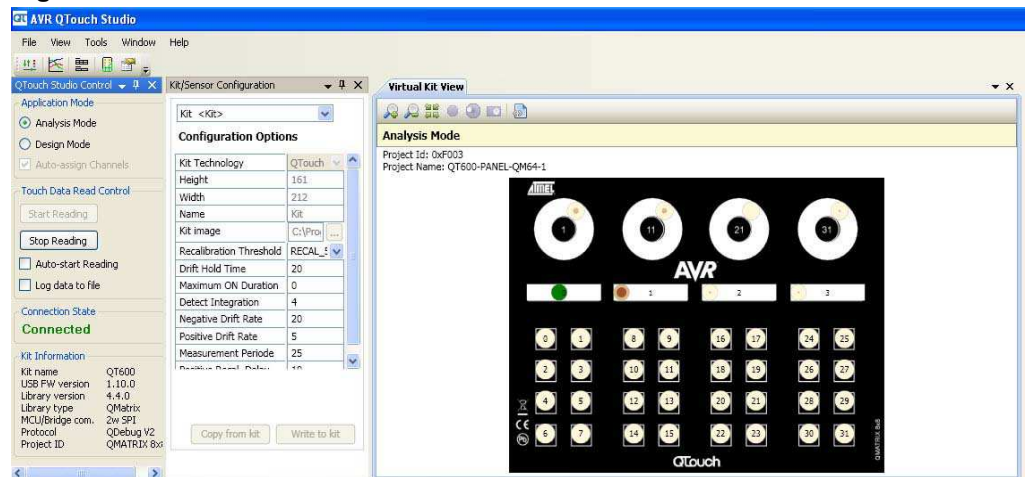
Figure 4-6. Sensor board connections.





10. Connect the touch panel.
11. For Atmel AT32UC3L: Use the QMatrix 8x8 Panel.
12. Launch Atmel QTouch Studio.
13. Plug the USB cable in to the Atmel QT600 interface board. Atmel QTouch Studio should now automatically connect to the kit. Touch data LED on USB interface board now starts blinking. This shows that data is being sent by the sensor board.
14. Select Analysis mode, and then press the Start Reading button.

You should now be able to view the touch data signals and the state of each sensor. Refer the [Figure 4-7](#).

Figure 4-7. Virtual kit view.



-  In Analysis mode, sensors displayed in a light grey color are inactive, and the kit will not respond to touches to these sensors. Sensors displayed in a light brown color are active sensors, and the kit will respond to touches to these sensors.
-  Whenever a touch to an active sensor is detected, a green filled circle can be seen indicating where the user touches the sensor.


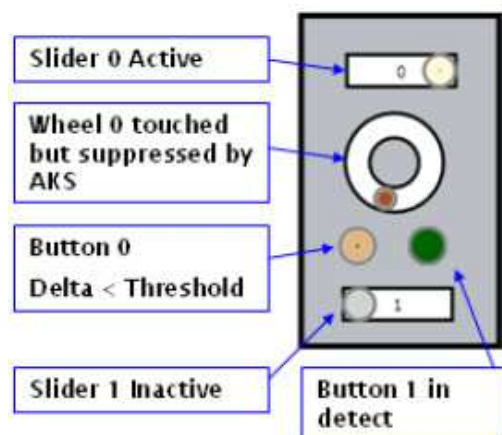
-  When a touch to the active sensor is no longer detected, the color of the sensor will go back to light brown again.

Figure 4-8. Sensor states.



4.2 Task 2: Set up a new project, select and include library, add touch files and ASF driver files

4.2.1 Introduction

In this task, we will create a new project. We will add the necessary libraries, touch files and assembler files.

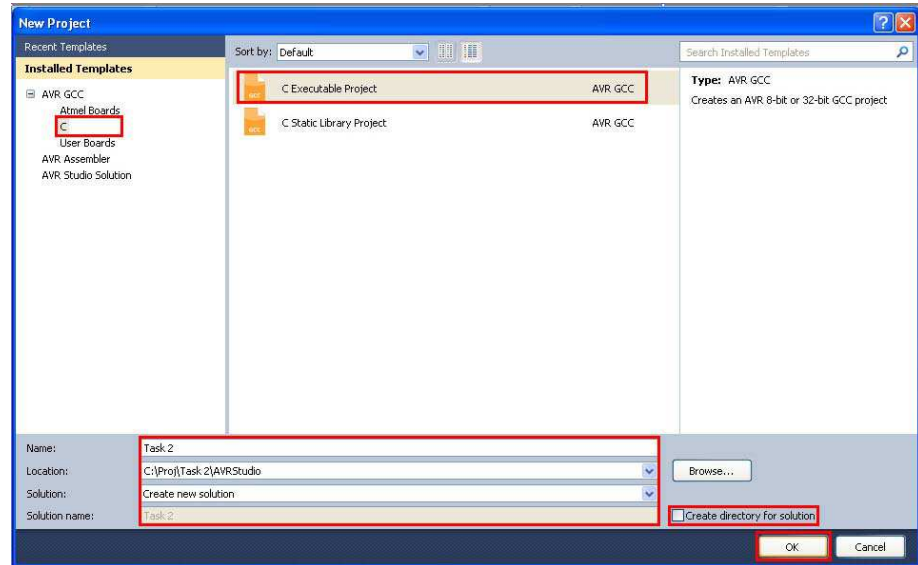
4.2.2 Setting up the code



Create a new project.

1. Open Atmel AVR Studio and create a new project. If AVR Studio is already running, then use File → New → Project.
2. Select project type AVR GCC, select C Executable project from installed templates and set project name to Task 2. Uncheck create directory, set Location, and then click Finish, as shown in [Figure 4-9](#).

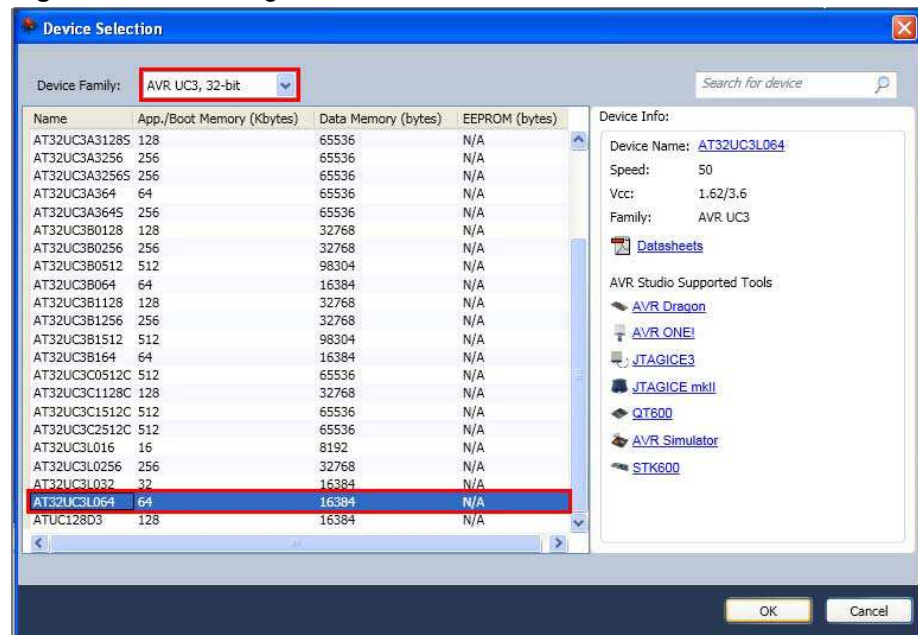
Figure 4-9. New project.



Set the MCU type.

1. Select Device family as AVR UC3, 32-bit, part number as AT32UC3L064 (or AT32UC3L0256) and then click OK, as shown in Figure 4-10.

Figure 4-10. Selecting the device.



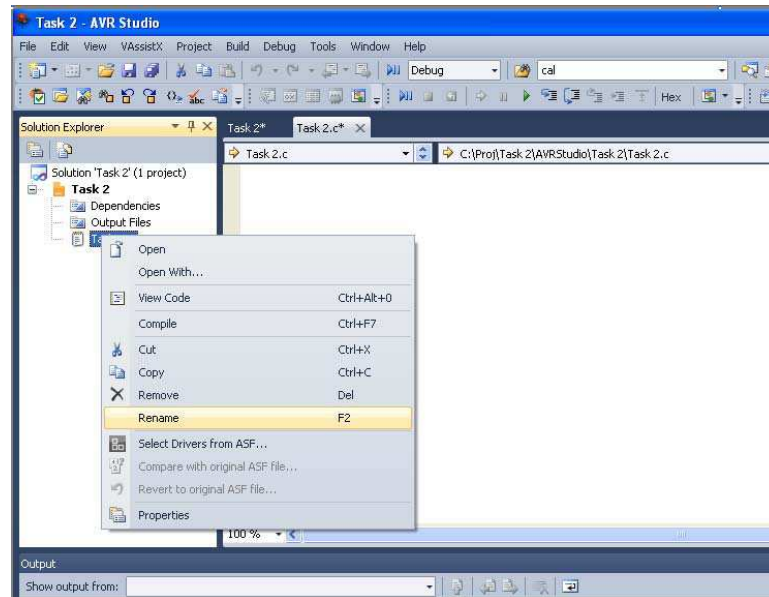
Add the main.c file.

In the solution explorer window,

1. Select and right click on Task 2.c, and click on remove option, as shown in Figure 4-11. Remove Task 2.c file from the project.



Figure 4-11. Removing the Task 2.c file.

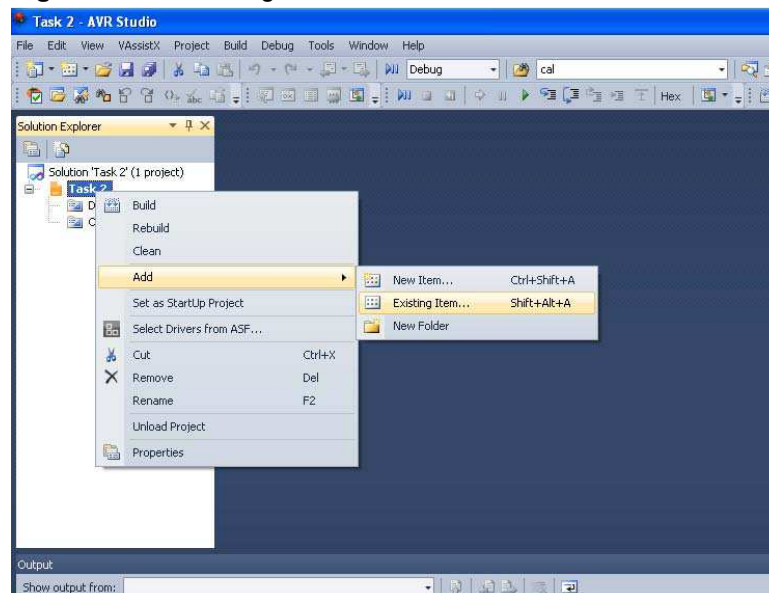


2. Right click on Project file (Task 2 in this case) and add the existing `main.c` source file to the project, as shown in Figure 4-12. `main.c` location is :
 `..\..\Exercises\QMatrix\Tasks\Task 2`



User can ignore the above steps in Section 4.2.2, if he intends to use the Task 2 project available inside Exercises folder.

Figure 4-12. Including the main.c.



Test compile to ensure all tools are installed OK.

1. Select Build → Rebuild solution, and ensure that all tools are installed. Check for error messages related to tools installation.
2. Any error messages related to tools installation; uninstall and install tools properly. Please ignore the error messages related to code and header files, as the necessary code and the files are not included at this stage.



Select the library file.

1. Open the Library_Selection_Guide.xls file in the Task2 folder and click on the QMatrix Tab (we are using Atmel QMatrix). Similarly, for Atmel QTouch, click on the tab for QTouch.
2. Select the technology as QMatrix, MCU family as Atmel AT32UC3, MCU type as 32-bit, MCU as Atmel AT32UC3L, the Tool chain as GCC, Ports available for QMatrix as CAT CSA-CSB GPIO function Maximum number of channels as 136, Maximum number of rotors or sliders as 34. As you can see, it shows the appropriate library file to be included and example project to be used.

NOTE

As we are building a new project, note down which library file is to be included.



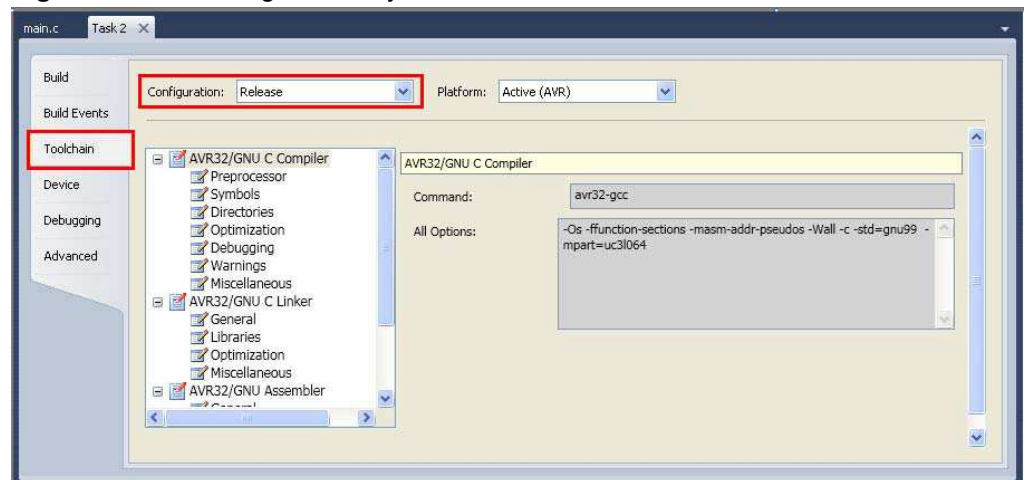
Here, the library file to be included is `libuc31-qtouch-gnu.a`.



Include the library file.

1. Select Project menu and click on properties.
2. Select Tool chain tab from the left most window and select release profile from the configuration window as shown in [Figure 4-13](#).

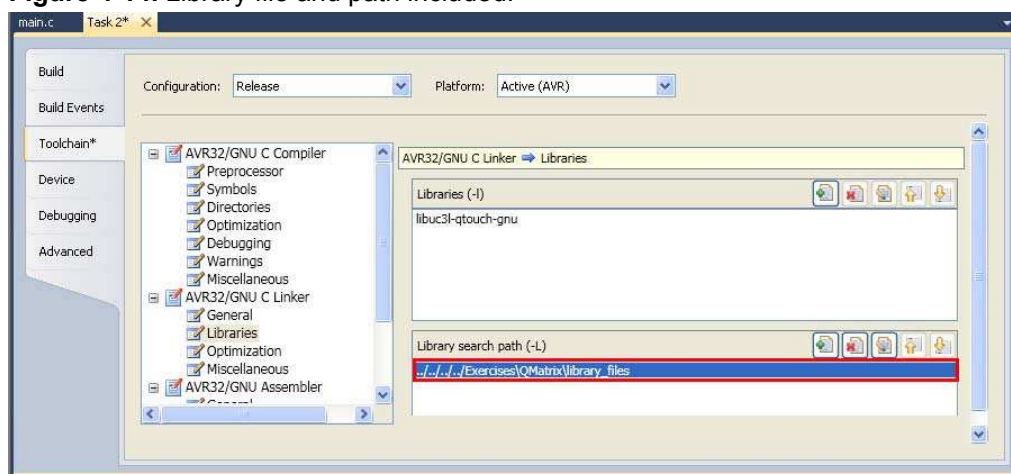
Figure 4-13. Including the library file.



3. Click on Libraries option under AVR/GNU C Linker, select add library option and add the library file `libuc31-qtouch-gnu.a` as shown in [Figure 4-14](#).
4. Click on add library path to set the appropriate path for the library as shown in [Figure 4-14](#).

The library file can be found at
`..\..\library_files\`

Figure 4-14. Library file and path included.

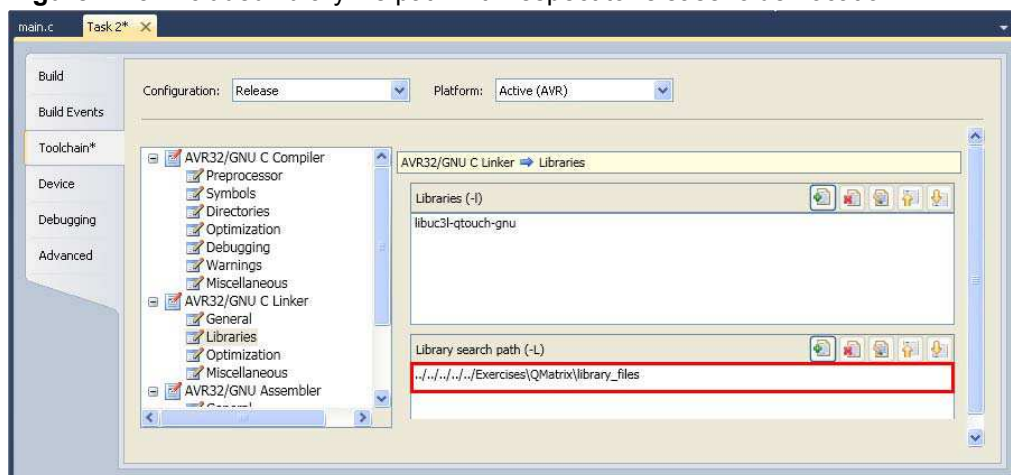


Here the path included is not relative to the output file. Output file is located inside release folder (\AVR 8015\Completed_Tasks\QMatrix\Tasks\Task2\Release). The library path has to come out of one location with respect to release folder, so that the library file path included properly. By adding one “..” in the path leads to control come out of one location. Figure 4-15 represents the same.



Missing the above step causes build error. Build error shows that the corresponding file is not found.

Figure 4-15. Included library file path with respect to release folder location.



5. Similarly include library path and library file in Debug profile also.



Include touch header files.

1. To use the touch keys, first we need to add `#include "touch_api_at32uc31.h"` to our list of include files. This will make the Atmel QTouch Library API available.
2. Add it under the following comment in the `main.c` file
`/* now include touch_api_at32uc31.h with the localization defined above */`
3. Open Project properties and select tool chain tab.

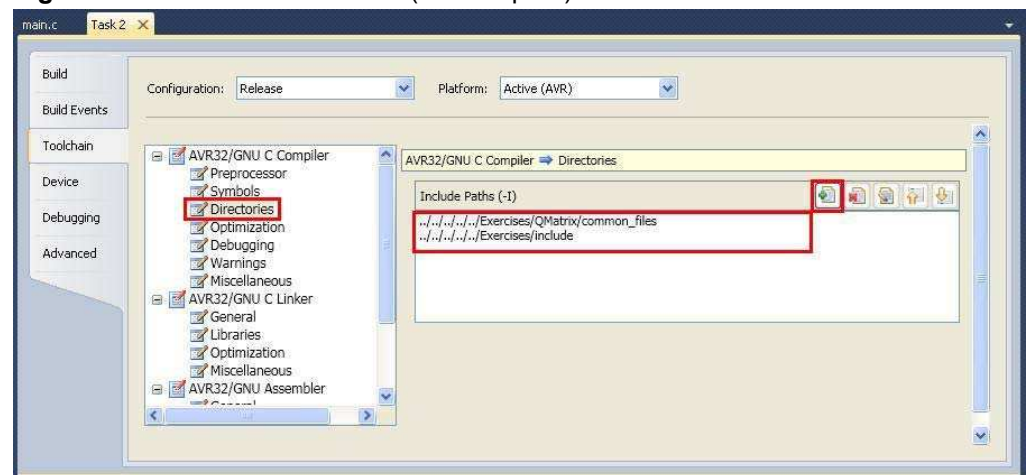
4. Click on directories option under AVR/GNU C compiler and enter the path where the `touch_api_at32uc31` file is found. This file is found in the include folder under Exercises.
5. Similarly, click on Directories and add the path for `touch_config_at32uc31.h` file. This file is found in the `common_files` folder.
6. The `touch_config_at32uc31.h` file is where all the technology options are configured.



Enter the relative path as shown in the Figure 4-16 (similar to how you added the library).

- IDE can use either relative or absolute file path depends upon the location of the file
- If the file to be included and project folder (Task 2 in this case) are in the same drive, relative path (is a path relative to the output file of the application) or absolute path can be used. Here output file is located in Task 2 → Release folder
- In the case of relative path, please ensure path of the file is included with respect to project output file location (Task2 → Release folder) as shown in Figure 4-16

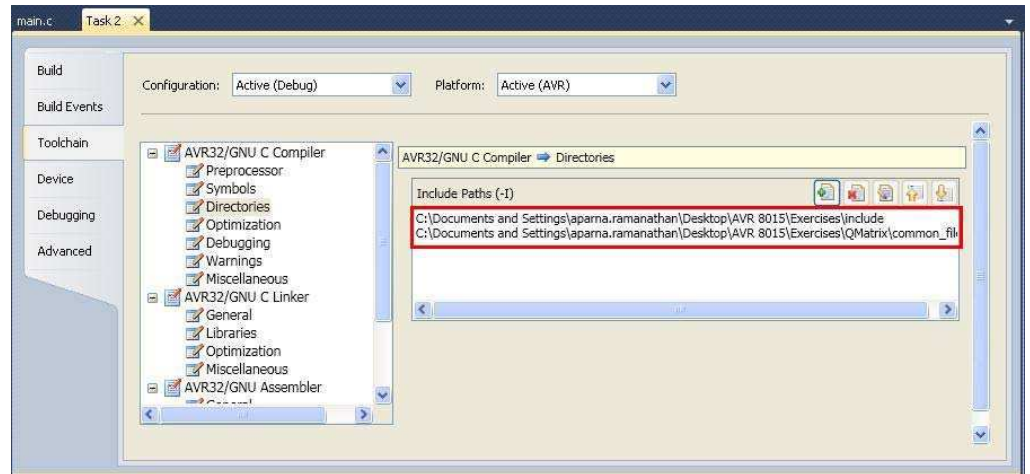
Figure 4-16. Included directories (relative path).



Please ensure that the file path is added relative to output file location. Missing this step causes build error. Build error shows that the corresponding file is not found.

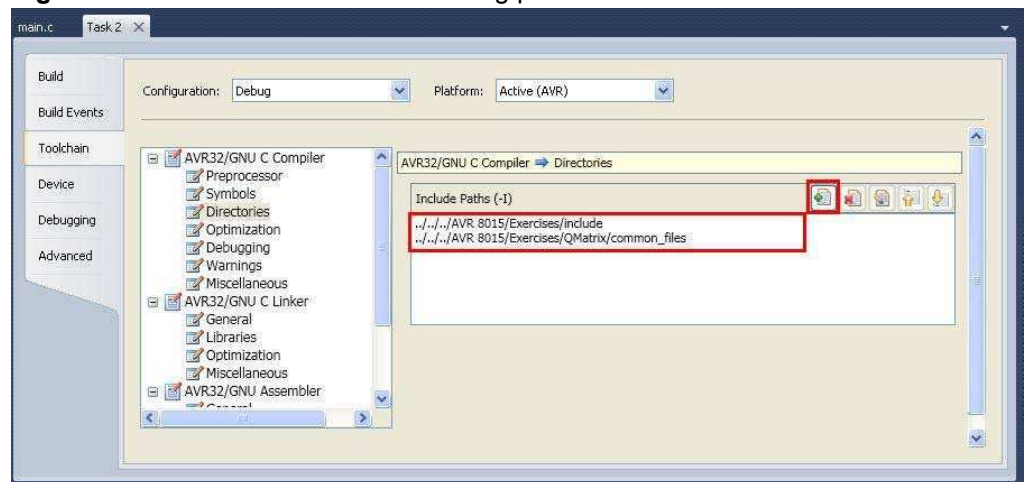
- If both are in different drives, absolute path gets selected by IDE, as shown in Figure 4-17

Figure 4-17. Included directories (absolute path).



7. Similarly include paths for both `touch_api_at32uc31.h` and `touch_config_at32uc31.h` files in Debug profile also, as shown in [Figure 4-18](#).

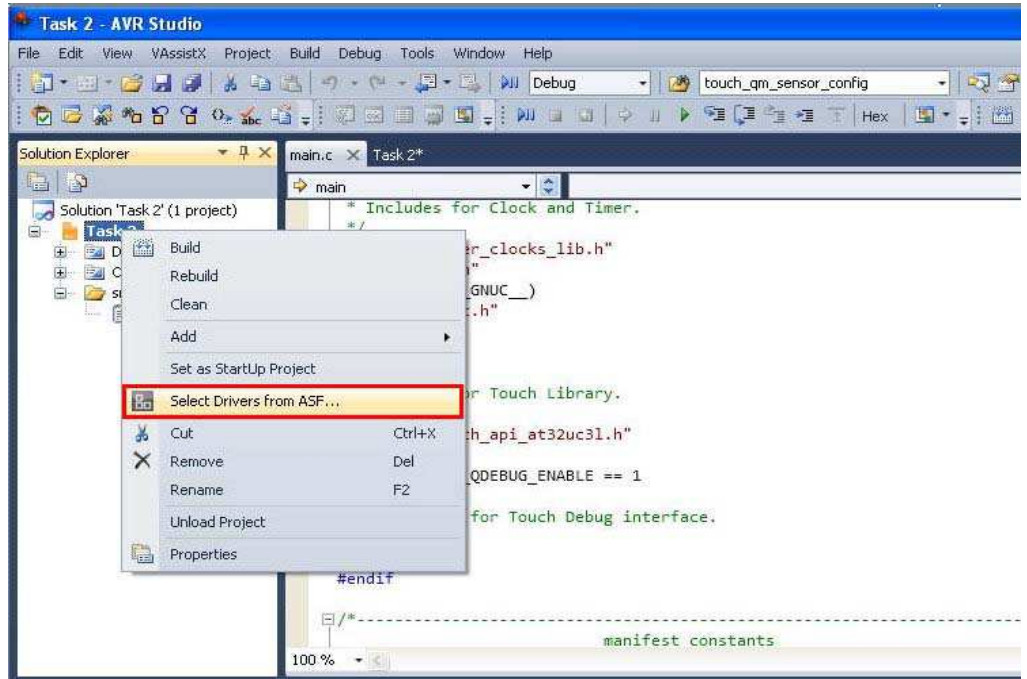
Figure 4-18. Included directories in debug profile.



Adding drivers from ASF.

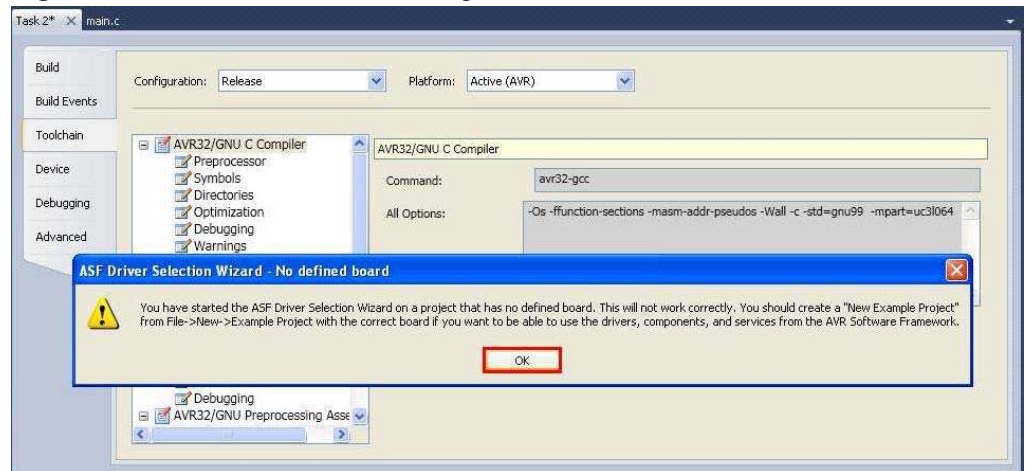
1. We need to add the necessary driver files to the project. This can be done by adding the drivers from the AVR Software Framework using Atmel AVR Studio 5.
2. Go to Project properties → Toolchain, in the Configuration tab select All configurations from the drop down list.
3. Right click on the project name and select the option “Select drivers from ASF”. Refer to [Figure 4-19](#).

Figure 4-19. Selecting drivers from ASF.



4. ASF Driver Selection-No defined board warning occurs. Press OK to continue. This opens the driver selection wizard. See [Figure 4-20](#).

Figure 4-20. No board defined warning.

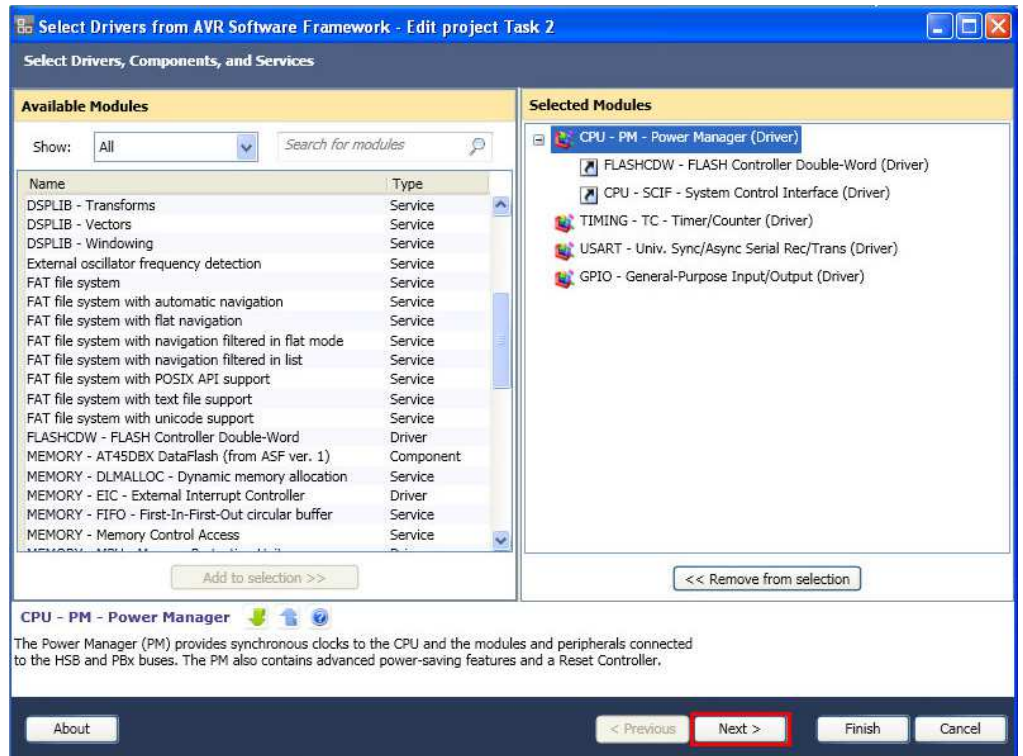


5. Add the following modules and press Next to continue.
 - CPU – PM - Power Manager (Driver)
 - GPIO - General Purpose Input/Output (Driver)
 - Timing – Timer/Counter (Driver)
 - USART-Univ. Sync/Async Serial Rec/Trans (Driver)

See [Figure 4-21](#) for selecting the modules to be added.

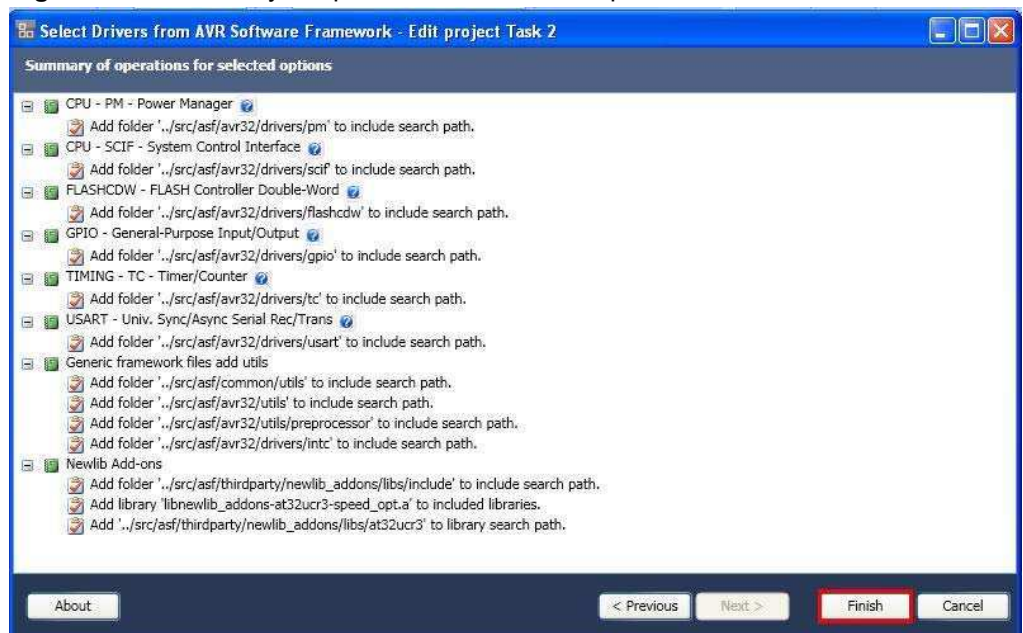


Figure 4-21. ASF driver selection wizard.



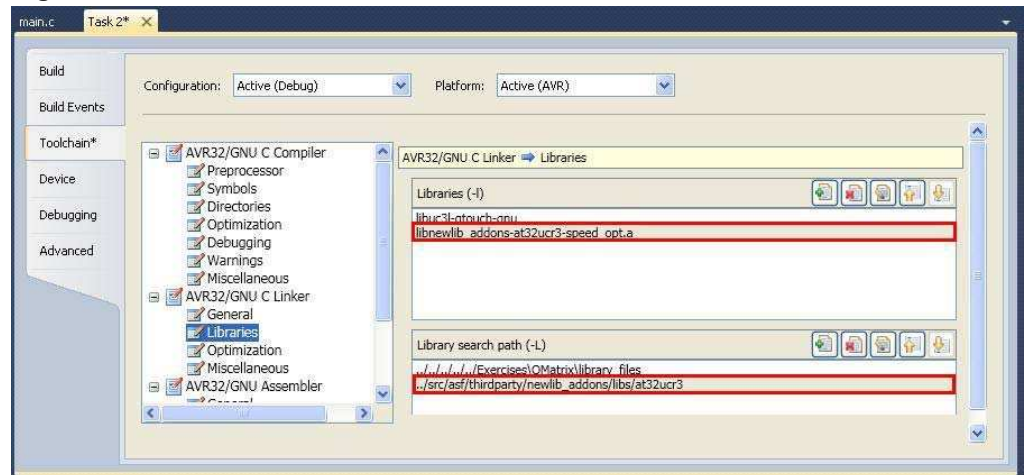
6. Next a summary of operations for the selected options will be displayed. You can see that the necessary include paths are being added. Press Finish to finish adding the ASF driver files.

Figure 4-22. Summary of operations for selected options.



- You can see that the corresponding directories and the library file for these files are added to the project. [Figure 4-23](#) shows the necessary library added in the project properties.

Figure 4-23. Included libraries.



Set the Optimization Level.

Please ensure the optimization level is set to Optimize for size (-Os) both in Release and Debug profile. This option is available under Project → Properties → Toolchain → AVR/GNU C Compiler → Optimization.



Tip Please ensure the optimization level is configured in the project.



Tip Now save and build the project, and if all the steps mentioned above have been followed, you should be able to build successfully with no errors.

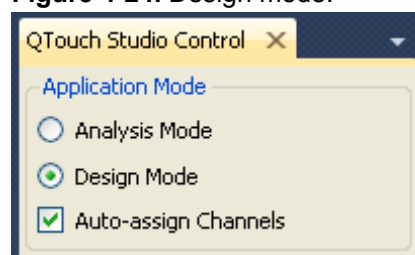
4.3 Task 3: Create a virtual kit in design mode and verifying the pin configuration



Create a virtual kit.

- Launch Atmel QTouch Studio. Select Design Mode.
- Design mode shall provide editable virtual kit. It allows users to add buttons, wheels, sliders, and kit background image and select kit technology (QTouch or QMatrix).
- Check the Auto-assign Channels option.

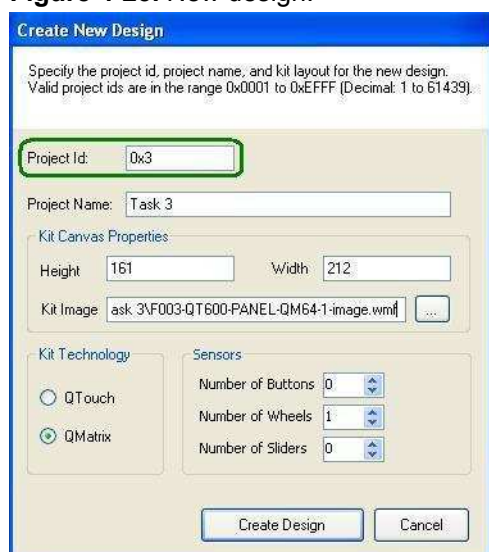
Figure 4-24. Design mode.





4. On the File Menu, click New Design. In the Create New Design window, specify the project ID, project name, kit canvas properties like Height and Width (millimeters) and background (kit) image, and kit technology, along with a valid number of sensors for the new project.
5. For this hands-on training, select QMatrix as the kit technology. Under sensors, select the number of buttons as 0, wheel as 1, and slider as 0.
6. For the background image, select the image provided in the Task 3 folder named F002-QT600-PANEL-QT16-image.
7. Assign channel numbers 0-3 to the wheel.

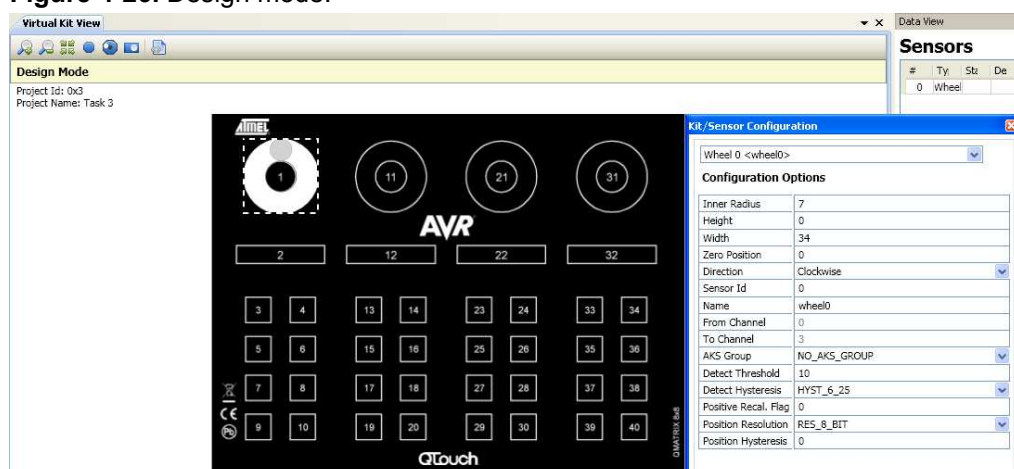
Figure 4-25. New design.



Note down the Project ID.

8. Click on each sensor and adjust the height and width accordingly in the kit/sensor configuration window. Refer to Figure 4-26.

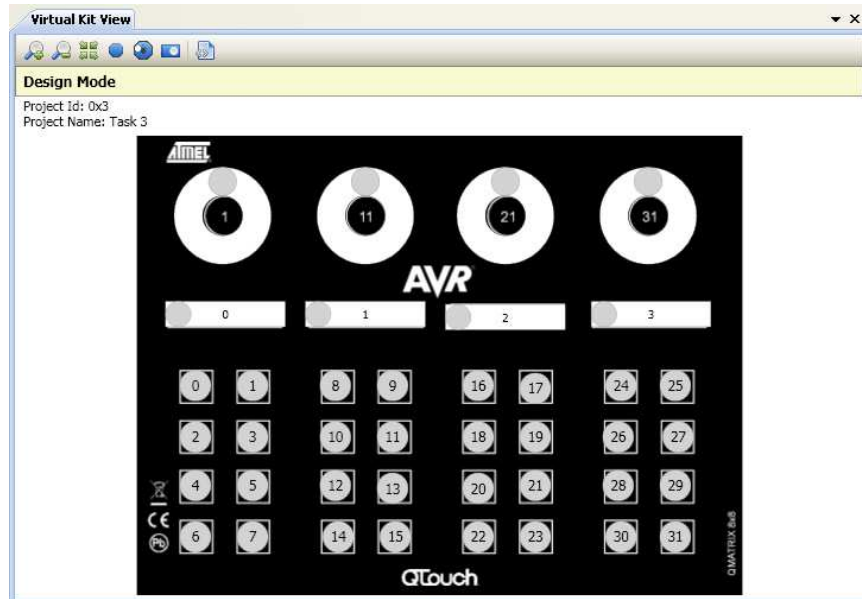
Figure 4-26. Design mode.



9. Save the design in the same Task 3 folder.

10. Notice that the wheel has been assigned to channels 0-3.
11. Notice the toolbar on the top left corner of the virtual kit window which is shown in [Figure 4-26](#) is used for adding wheels, sliders and buttons. Click on those icons to add the corresponding sensor.
12. Add the sliders, wheels and buttons in the sequential order as displayed on the sensor board.
13. Click on the slider icon and a slider gets added to the virtual kit. Similarly, click on the wheel icon to add a wheel to the virtual kit. Refer to [Figure 4-27](#).

Figure 4-27. Adding a slider and wheel.



14. Ensure that the channel numbers are assigned in the following sequence:
 - Wheel 0: 0-3
 - Slider 0: 4-7
 - Button group 3 to 10: 8-15
 - Wheel 1: 16-19
 - Slider 1: 20-23
 - Button group 13 to 20: 24-31
 - Wheel 2: 32-35
 - Slider 2: 36-39
 - Button group 23 to 30: 40-47
 - Wheel 3: 48-51
 - Slider 3: 52-55
 - Button group 33 to 40: 56-63



Ensure the sensors are added to the project in the above sequence so that the channels will also be assigned to the sensors in the above sequence.

15. Now adjust the height and width of the wheels and sliders.
16. Save the Design (File → Save Design) and close it.



Make a note of the project ID.



Make sure that the sensors have been assigned channels as mentioned above.



The macros for the pin configurability are defined in the
touch_config_at32uc31.h.

Now build the project, and if everything has been done as per the guidelines given, you should be able to build successfully with no errors.

4.4 Task 4: Configure the global options and enable/set the sensors

4.4.1 Setting up the code



Initialize the library.

After this initial configuration, we need to initialize the [Atmel QTouch Library](#). This is done by calling `touch_qm_sensors_calibrate ()`; Add the code given below under the label: `/* Initialize touch sensing. */` to initialize Touch Library.

```
touch_ret = touch_qm_sensors_calibrate ();
if (touch_ret != TOUCH_SUCCESS)
{
    while (1u);
}
```



Adding more Touch Library configuration.

Next, we need to set up some runtime global QMatrix parameters. For this add the following macros under the label: `/* QMATRIX GLOBAL SENSOR CONFIGURATION INFO. */` in the structure static `touch_qm_config_t qm_config`.

```
QM_DI,
QM_NEG_DRIFT_RATE,
QM_POS_DRIFT_RATE,
QM_MAX_ON_DURATION,
QM_DRIFT_HOLD_TIME,
QM_POS_RECAL_DELAY,
QM_RECAL_THRESHOLD,
```

These macros are defined in the file `touch_config_at32uc31.h`. Touch Library will shift the value specified to the corresponding register offset of the CAT Module.



Adding the timer ISR.

Call `init_timer()`; to configure timer ISR to fire regularly. Add this under the label `/* Configure timer to fire ISR regularly. */`

Find this function and add the following code to it:

```
volatile avr32_tc_t *tc = EXAMPLE_TC;
/* options for waveform generation. */
static const tc_waveform_opt_t WAVEFORM_OPT = {
    .channel = EXAMPLE_TC_CHANNEL, /* Channel selection. */
    .bswtrg = TC_EVT_EFFECT_NOOP, /* Software trigger effect on
TI0B. */
    .beevt = TC_EVT_EFFECT_NOOP, /* External event effect on
TI0B. */
}
```

```

        .bcpc = TC_EVT_EFFECT_NOOP,          /* RC compare effect on TIOB.
*/
        .bcpb = TC_EVT_EFFECT_NOOP,          /* RB compare effect on TIOB.
*/
        .aswtrg = TC_EVT_EFFECT_NOOP,       /* Software trigger effect on
TIOA. */
        .aeevt = TC_EVT_EFFECT_NOOP,        /* External event effect on
TIOA. */
        .acpc = TC_EVT_EFFECT_NOOP,         /* RC compare effect on TIOA:
toggle. */
        .acpa = TC_EVT_EFFECT_NOOP,         /* RA compare effect on TIOA:
toggle (other possibilities are none, set and clear). */
        .wavsel = TC_WAVEFORM_SEL_UP_MODE_RC_TRIGGER, /* Waveform
selection: Up mode with automatic trigger(reset) on RC compare. */
        .enetr = 0u,                        /* External event trigger enable. */
        .eevt = 0u,                          /* External event selection. */
        .eevtedg = TC_SEL_NO_EDGE, /* External event edge selection. */
        .cpcdis = 0u,                        /* Counter disable when RC compare.
*/
        .cpcstop = 0u,                      /* Counter clock stopped with RC
compare. */
        .burst = 0u,                        /* Burst signal selection. */
        .clki = 0u,                          /* Clock inversion. */
        .tcclks = TC_CLOCK_SOURCE_TC3 /* Internal source clock 3,
connected to fpba / 8. */
};

static const tc_interrupt_t TC_INTERRUPT = {
        .etrgrs = 0u,
        .ldrbs = 0u,
        .ldras = 0u,
        .cpcs = 1u,
        .cpbs = 0u,
        .cpas = 0u,
        .lovrs = 0u,
        .covfs = 0u
};

/* initialize the timer/counter. */
tc_init_waveform (tc, &WAVEFORM_OPT);

/* set the compare triggers. */
tc_write_rc (tc, EXAMPLE_TC_CHANNEL, EXAMPLE_RC_VALUE);

/* configure Timer interrupt. */
tc_configure_interrupts (tc, EXAMPLE_TC_CHANNEL, &TC_INTERRUPT);

/* start the timer/counter. */

```



```
tc_start (tc, EXAMPLE_TC_CHANNEL);
```

Find the function `ISR tc_irq(void)`; and add the following code to it:

```
/* update the current time */
current_time_ms_touch++;
/* every 25th ms */
if ((current_time_ms_touch % measurement_period_ms) == 0u)
{
    /* set flag: it's time to measure touch */
    time_to_measure_touch = 1u;
}
/* clear the interrupt flag. This is a side effect of reading the
TC SR. */
tc_read_sr (EXAMPLE_TC, EXAMPLE_TC_CHANNEL);
```



Now it's time to configure our first sensor.

1. We use the following API call to configure a set of buttons. The below is for configuring sensors channels 8 to 15 as buttons 2 to 9.

```
for (i = 8u; i < 16u; i++)
{
    touch_ret =
touch_qm_sensor_config (SENSOR_TYPE_KEY, (channel_t) i,
                        (channel_t) i, AKS_GROUP_1, 20u,
                        HYST_6_25, RES_1_BIT, 0u, &sensor_id);

    if (touch_ret != TOUCH_SUCCESS)
    {
        while (1u); /* Check API Error return code. */
    }
}
```

Where 'i' is the channel number assigned to a particular button. Place this under the label `/* Configure 8 single channel keys as Sensor's 2-9. */`.

This tells Atmel QMatrix that the sensor corresponding to the particular sensor id should be treated as a key and is assigned the i^{th} channel. We will look at the other parameters a little later.

Similarly configure the other buttons under the corresponding labels.

2. To enable a slider using channels 4, 5, 6, and 7 the following code is used:

```
touch_ret =
touch_qm_sensor_config (SENSOR_TYPE_SLIDER, (channel_t) 4u,
                        (channel_t) 7u, AKS_GROUP_1, 30u, HYST_6_25,
                        RES_8_BIT, 0u, &sensor_id);

if (touch_ret != TOUCH_SUCCESS)
{
    while (1u); /* Check API Error return code. */
}
```


Place this piece of code under the label
 /* Configure a 4 channel Slider as Sensor 1. */

Similarly configure the other sliders under the corresponding labels.

NOTE

The RES_8_BIT argument is the slider resolution.

3. Similarly, for enabling the rotor using channels 0, 1, 2, and 3 using the following code:

```
touch_ret =
touch_qm_sensor_config (SENSOR_TYPE_ROTOR, (channel_t) 0u,
                        (channel_t) 3u, AKS_GROUP_1, 25u, HYST_6_25,
                        RES_8_BIT, 0u, &sensor_id);
if (touch_ret != TOUCH_SUCCESS)
{
    while (1u);          /* Check API Error return code. */
}
```

Place this piece of code under the label
 /* Configure a 4 channel Rotor as Sensor 0. */

Similarly configure the other rotors under the corresponding labels. The other arguments are not too important right now.

4. Add the following piece of code after the `init_system();` call, after the label
 /* Configure the touch library sensors. */

```
touch_ret = config_64ch_touch_keys_rotors_sliders ();
if (touch_ret != TOUCH_SUCCESS)
{
    while (1u);          /* Check API Error return code. */
}
```



Configuring burst length.

1. Call `qm_burst_length`, to configure the burst length array starting index. Add this against the label `/* QMatrix burst length array starting index.*/` in the structure `static touch_qm_config_t qm_config`.

2. Find this array and define the burst lengths for the sensors as below.

```
32u,          /* x0, y0 */
32u,          /* x1, y0 */
32u,          /* x2, y0 */
32u,          /* x3, y0 */
48u,          /* x4, y0 */
48u,          /* x5, y0 */
48u,          /* x8, y0 */
48u,          /* x9, y0 */
```

3. Similarly enter the burst length values for all the sensors up to (x9, y7).



Somewhere to store time.

We are now almost ready to start taking measurements. Touch measurement is taken by calling `touch_qm_sensors_start_acquisition (current_time_ms_touch, &qm_dma, NORMAL_ACQ_MODE, touch_qm_measure_complete_callback)`, 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.





We are using `current_time_ms_touch` as a simple counter to keep track of time. `p_qm_dma` is a pointer to the dma channels used for Touch measurement. `qm_acq_mode` specifies whether raw acquisition mode or normal acquisition mode is being used. `measure_complete_callback` function is called by the Touch Library once the QMatrix Touch measurement is complete. It provides pointers related to the measured touch data and touch status. `measure_data` is QMatrix sensor status and measured data pointer. `touch_ret_t` shows QTouch Library Error status.



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 (possibly to fix sensor layout faults).

Add this macro, `QM_FILTER_CALLBACK`, in the structure `static touch_qm_config_t qm_config` after the label `/* Filter callback function pointer */`.



We are now ready to perform a touch measurement.

Inside the main loop, under the label `/* Start a touch sensors measurement process. */`, add the following piece of code:

```
touch_ret =
    touch_qm_sensors_start_acquisition (current_time_ms_touch,
                                        &qm_dma, NORMAL_ACQ_MODE,

    touch_qm_measure_complete_callback);
    if ((touch_ret != TOUCH_SUCCESS)
        && (touch_ret != TOUCH_ACQ_INCOMPLETE))
    {
        while (1u);
        /* Reaching this point can be due to -
        1. The api has returned an error due to a invalid input parameter.
        2. The api has been called during a invalid Touch Library state. */
    }
```



Build the code.

It should build successfully with no errors or warnings.

4.4.2 Further explanation

Most of the function calls used in this task are quite simple and self explanatory. The most complex line is the `touch_qm_sensor_config()` function, so let's look closer at this:

```
touch_ret_t touch_qm_sensor_config (sensor_type_t sensor_type,
                                    channel_t from_channel,
                                    channel_t to_channel,
                                    aks_group_t aks_group,
                                    threshold_t detect_threshold,
                                    hysteresis_t detect_hysteresis,
                                    resolution_t position_resolution,
                                    uint8_t position_hysteresis,
                                    sensor_id_t * p_sensor_id);
```

The parameters are as follows:

- `sensor_type` can be of type key, rotor or slider
- `from_channel` the first channel in the sensor group
- `to_channel` the last channel in the sensor group
- `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
- `p_sensor_id` The sensor id value of the configured sensor is updated by the Touch Library

NOTE The range of `position_hysteresis` value is from 0 to 7.

4.5 Task 5: Use of the Debug Interface

4.5.1 Introduction

The debug interface is used to communicate various touch sensor information to the computer running Atmel QTouch Studio. The debug interface as seen from the Atmel AT32UC3L is implemented completely in firmware. This is to be added only for debugging purposes.

4.5.2 Setting up the code



Add the source files.

- `QDebug.c`
- `QDebugTransport.c`
- `SPI_Master.c`

These source files with the corresponding header files are present in Task 5 folder.

1. `QDebug.c`, `QDebugTransport.c`, and `SPI_Master.c`, and its corresponding headers can also be found in the `qdebug` folder.
2. The debug interface is used to communicate various touch sensor information to a computer running Atmel QTouch Studio.
3. In the solution explorer tab in Atmel AVR Studio, right click the project and select Add Existing File, and then select the above mentioned source files and the corresponding header files from Task 5 folder. Alternatively, you can drag and drop the file onto the project. Similarly include the other source files.
4. The Atmel QT600 interface board and QTouch Studio can be used to read touch data from any application based on the [QTouch Library](#). Here we are using USART SPI mode as the communication interface.
5. Open the Project properties Window, select tool chain tab and click on Directories option below AVR/GNU C compiler, and specify the right paths for all three files (enter the relative path). Make sure those directories paths are included both in Release and Debug profile.



Add the Include files.

Include the following header file in the `main.c` file below the label:

```
/* Includes for Touch Debug interface */  
#include "QDebug.h"
```



Adding code.

1. In the `QDebugSettings.h` file, add the project ID (same project id given to the virtual kit image in Task3) similar to the below format, under the label
/* Define Custom project as the project id of the virtual kit */ #define
CUSTOM_PROJECT Project ID (ID given to the virtual kit).
2. Assign the project ID under the label
/* Set up project info */.
#define PROJECT_ID CUSTOM_PROJECT.
3. Define the communications protocol. In this hands-on training, we are using the SPI protocol.
#define QDEBUG_SPI, under the label /* Edit project Info */.
4. In `main.c`, add the code `INTC_register_interrupt (&touch_acq_done_irq, AVR32_CAT_IRQ, AVR32_INTC_INT3);` under the comment /*Register the Touch Library CAT interrupt handler to the interrupt controller... */ before #endif
5. In `main.c`, call the function `QDebug_Init();` under the label /*Initialize the debug interface */.
6. In `main.c`, add this piece of code under the label /* Send out the Touch debug information data each time when Touch measurement process is completed. */
`QDebug_SendData (p_qm_measure_data->acq_status);`
7. In `main.c`, call the function `QDebug_ProcessCommands();` under the label /* Process any commands received from QTouch Studio. */.



Build the code.

1. It should build successfully with no errors or warnings.
2. Now connect the kit as shown in Task 1.
3. Program the code.
4. Unplug the squid programming cable that is connected between Atmel QT600 MCU board and the Atmel AVR JTAGICE mkII. Insert the 10-pin cable for streaming touch data.
5. Launch Atmel QTouch Studio and switch to Analysis mode.
6. Check if your kit gets connected.
7. Switch to Analysis mode and click on Start Reading button.
8. Test whether you are able to detect touch and see it on QTouch Studio or not. In the Data view you can see the values of reference, signal, and delta and the state of each sensor.

4.6 Task 6: Analysis mode

4.6.1 Introduction





Analysis mode has a scalable image of the connected kit. It will show touch events such as button presses, wheel use, and slider use. The toolbar buttons in the virtual kit view window shall be used to perform zoom-in, zoom-out, auto fit and auto code generation operations.



The Analysis mode can be selected by using the radio buttons in the Control view. QTouch Studio reads out the channel to sensor mapping for an AVR

QTouch Kit when it is connected to the PC. Using this information, the image of the kit will be updated.

-  As explained during Task 1, sensors displayed in a light grey color are inactive, and sensors displayed in a light brown color are active sensors. Whenever a touch to an active sensor is detected a filled circle indicating where the user touches the sensor is drawn. The filled circle increases when the delta increases, and changes color when the threshold is reached.
-  When a touch to the active sensor is no longer detected, the color of the sensor will go back to light brown again. The user shall be able to edit sensor configuration options (non-design options) and kit configuration options (non-design options) in Analysis mode.

4.6.2 Configuring the kit/sensors



Adjacent Key Suppression® (AKS).

1. Open Kit/Sensor Configuration window, select Button0 and configure the AKS Group as AKS_GROUP_1.
2. Similarly, configure all the sensors as AKS_GROUP_1.
3. Now, if you touch more than 1 button at the same time, only one is detected.



In designs where the sensors are close together or set for high sensitivity, multiple sensors might report detect simultaneously if touch is near them. To allow applications to determine the intended single touch, the touch library provides the user the ability to configure a certain number of sensors in an AKS group.

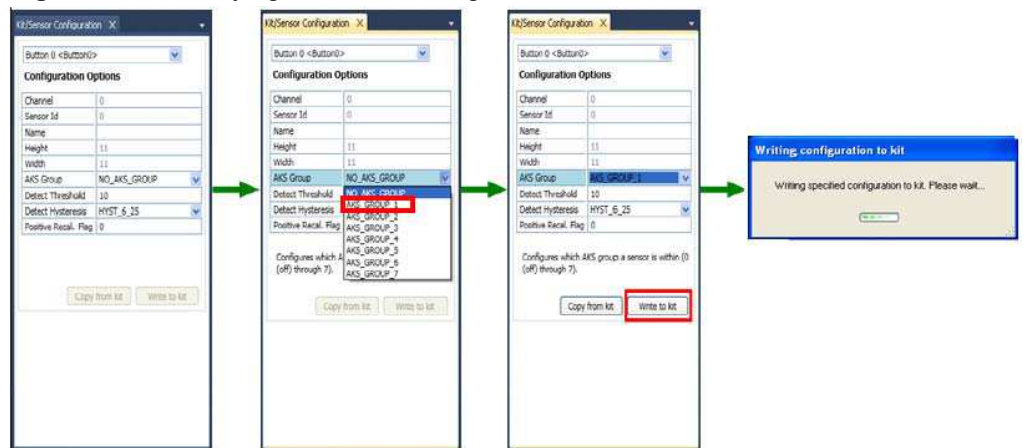


When a group of sensors are in the same AKS group, only the first, strongest sensor will report detection. The sensor reporting detection will continue to report detection even if another sensor's delta becomes stronger. The sensor stays in detect until its delta falls below its detection threshold, and then, if any more sensors in the AKS group are still in detect, the strongest will report detection.



And so at any given time, only one sensor from each AKS group will be reported to be in detect.

Figure 4-28. Modifying kit/sensor configuration.





Modify all global and sensor parameters runtime as shown in Figure 4-31.



Detect threshold.

1. Open the Kit/Sensor Configuration window, select any sensor and change the values of the detect threshold. Click on Write to kit.
2. Observe how sensitivity of that sensor changes with different values.
3. The lower the value, the more sensitive is the sensor.



A sensor's detect threshold defines how much its signal must drop below its reference level to qualify as a potential touch detect. It has a range of between 3 and 255.



For more details on all the configuration parameters, please refer to the Atmel QTouch Library User Guide.



Maximum ON duration.

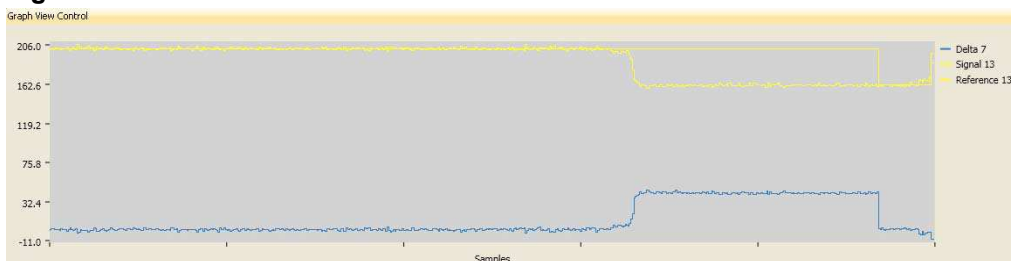
1. Click on the View Menu and click on Graph View. The QTouch Graph view window opens.
2. Select data set for channel 7 and observe what happens on touch.

Figure 4-29. Maximum ON duration = 0.



3. If a touch happens for a prolonged duration when the Maximum ON duration is zero, the sensor remains in detect.
4. Open the Kit/Sensor configuration window.
5. Change the Maximum ON duration to 30 and the sensor automatically recalibrates after six seconds.

Figure 4-30. Maximum ON duration = 30.



If an object unintentionally contacts a sensor and results in touch detection for a prolonged interval, it is usually desirable to recalibrate the sensor in order to restore its function, perhaps after a time delay of some seconds. The Maximum ON duration timer monitors such detections.

4.7 Task 7: Design validation

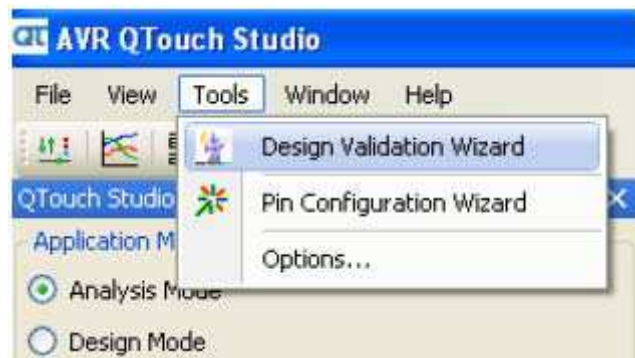
4.7.1 Introduction

The design validation wizard is a tool used to validate the design by measuring and logging values received from the kit.

4.7.2 Using the design validation wizard

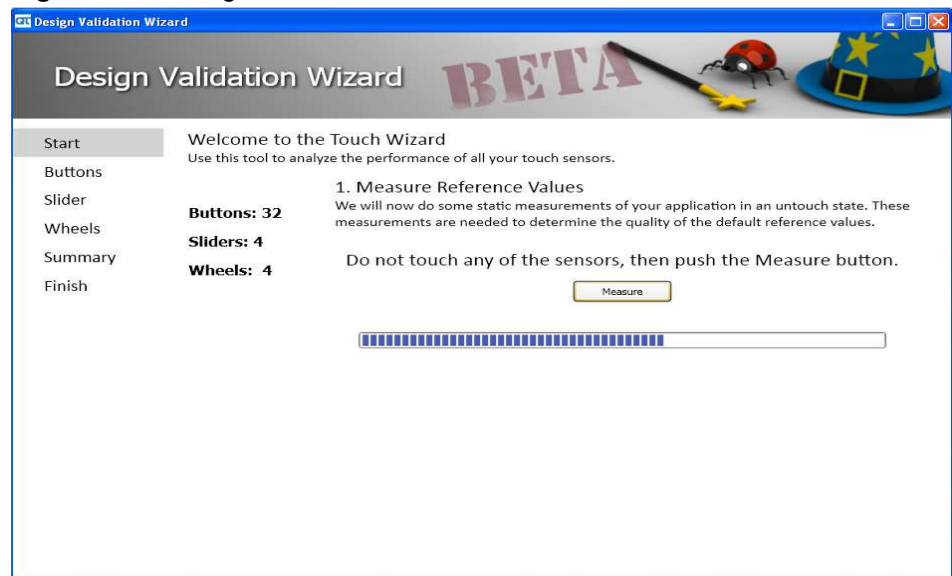
1. Connect the kit and start a touch debug session.
2. Launch Atmel QTouch Studio and start the design validation wizard from the Tools menu.

Figure 4-31. Selecting design validation wizard.



3. The design validation wizard opens.

Figure 4-32. Design validation wizard.



4. On the first page, it will ask you to click Measure button without touching any of the sensors. Follow the instructions.
5. Click Next and follow subsequent instructions.

6. At the end, you will get a summary with details about each sensor and recommended settings.
7. You can either write these recommended settings to the kit or project file.
8. Save the design and project file and reopen them again to see if the changes have been made.

4.8 Task 8: Log and analyze data



Logging the data.

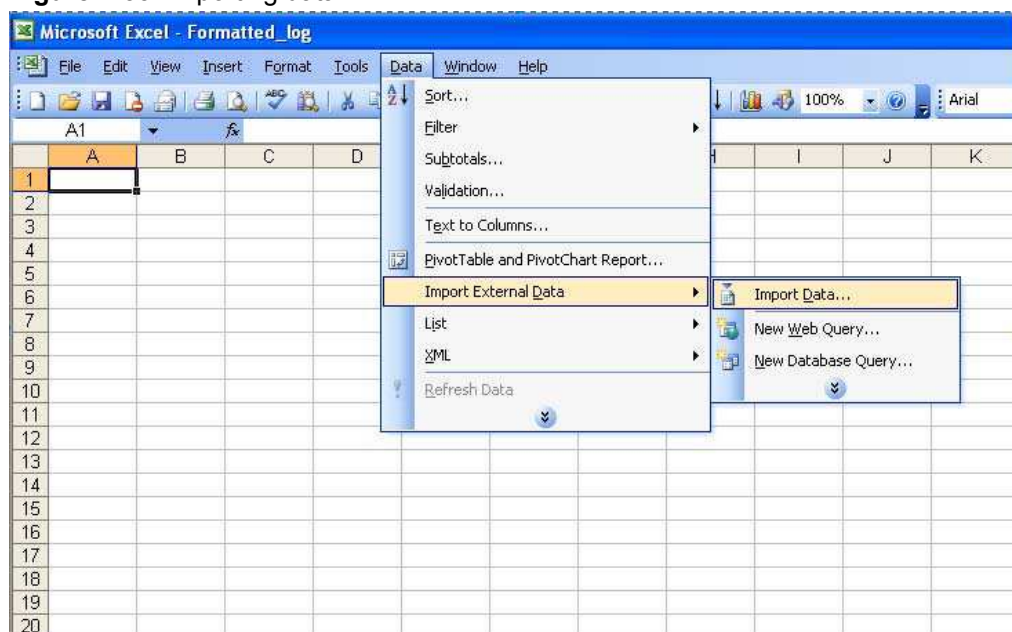
1. If the Auto-start Reading option is checked, then Atmel QTouch Studio will automatically start reading touch data when a kit is connected.
2. If unchecked, you must click the Start Reading button to make QTouch Studio start reading touch data. Checking the Log data to file checkbox will make QTouch Studio log data read from the kit.
3. As soon as reading has stopped, either because the user has clicked on the Stop Reading button or the kit has been disconnected from the PC, a dialog will popup to let the user specify a location and file name to save the logged data.
4. Note that clicking the Cancel button will simply discard all data logged and close the dialog.



Analysis and debugging.

1. Open a new Excel spreadsheet and import data from the logged data.
2. Select tab and semicolon as delimiters and save the file.

Figure 4-33. Importing data.



3. You can also create charts, which help in analyzing the logged data and observing variations. In the Excel spreadsheet named Formatted_Log_Chart, you can observe the delta variations for sensor 7 in the form of a chart as depicted.



Ensure the size of the logged data is less than 65535 as the Excel spreadsheet can accommodate up to 65535 data values on each column.

5 Summary

The main goal with this hands-on training was to learn how to use Atmel QT600 set up and use the Atmel QTouch suite (with Atmel AVR Studio 5) and the QT600 kit.

- Set up the hardware
- Create a project from scratch and include libraries and other header files
- Create a virtual kit, use the pin configuration wizard and configure technology options
- Configure global options and enable sensors
- Add buttons, sliders and wheels
- Use Analysis mode
- Use Design validation wizard
- Log and analyze data

In addition, you have learned how to use the Atmel QTouch Studio GUI, to write C code, debug your code and program the device.

NOTE

All the include files and other source files in this hands-on training have been taken from Atmel QTouch Library version 4.4. If a new library version is released, please use the latest files from it.



6 Atmel technical support center

Atmel has several support channels available:

- Web portal: <http://support.atmel.no/> All Atmel microcontrollers
- Email: touch@atmel.com All Atmel touch products
- Email: avr@atmel.com All Atmel AVR products
- Email: avr32@atmel.com All 32-bit Atmel AVR products

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

- A rich FAQ database
- Easy submission of technical support requests
- A history of all your past support requests
- Atmel microcontrollers' newsletters
- Information about available trainings and training material

7 Table of contents

Features	1
1 Introduction	1
2 Document overview	2
3 Requirements	3
3.1 Software	3
3.2 Hardware	3
3.3 Atmel QT600 system description	3
3.3.1 Block diagram.....	4
3.3.2 Pin schematic	4
4 Assignment	5
4.1 Task 1: Connect the tools and run a test application	5
4.1.1 Introduction.....	5
4.1.2 Connecting the hardware	8
4.2 Task 2: Set up a new project, select and include library, add touch files and ASF driver files	10
4.2.1 Introduction.....	10
4.2.2 Setting up the code	10
4.3 Task 3: Create a virtual kit in design mode and verifying the pin configuration	19
4.4 Task 4: Configure the global options and enable/set the sensors	22
4.4.1 Setting up the code	22
4.4.2 Further explanation	26
4.5 Task 5: Use of the Debug Interface.....	27
4.5.1 Introduction.....	27
4.5.2 Setting up the code	27
4.6 Task 6: Analysis mode	28
4.6.1 Introduction.....	28
4.6.2 Configuring the kit/sensors.....	29
4.7 Task 7: Design validation	31
4.7.1 Introduction.....	31
4.7.2 Using the design validation wizard	31
4.8 Task 8: Log and analyze data	32
5 Summary	33
6 Atmel technical support center	34
7 Table of contents	35

**Atmel Corporation**

2325 Orchard Parkway
San Jose, CA 95131
USA

Tel: (+1)(408) 441-0311

Fax: (+1)(408) 487-2600

www.atmel.com

Atmel Asia Limited

Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road

Kwun Tong, Kowloon

HONG KONG

Tel: (+852) 2245-6100

Fax: (+852) 2722-1369

Atmel Munich GmbH

Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY

Tel: (+49) 89-31970-0

Fax: (+49) 89-3194621

Atmel Japan

16F, Shin Osaki Kangyo Bldg.
1-6-4 Osaki Shinagawa-ku
Tokyo 104-0032

JAPAN

Tel: (+81) 3-6417-0300

Fax: (+81) 3-6417-0370

© 2012 Atmel Corporation. All rights reserved.

Atmel®, Atmel logo and combinations thereof, AVR®, AVR Studio®, Adjacent Key Suppression®, AKS®, megaAVR®, QTouch®, STK®, tinyAVR®, XMEGA®, and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Windows® and others are registered trademarks or trademarks of Microsoft Corporation in U.S. and or other countries. Other terms and product names may be trademarks of others.

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN THE ATMEL TERMS AND CONDITIONS OF SALES LOCATED ON THE ATMEL WEBSITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS AND PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.