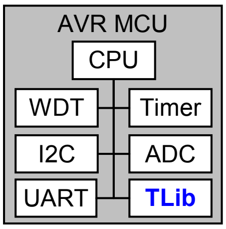
# Creating an Atmel QTouch$^{TM}$ Library Project using GCC or IAR

2009-11-17, v14, Paul Russell, Atmel QRG FAE

This document is intended only as a quick start guide for Atmel QTouch$^{TM}$ Library (TLib). This Guide will show you how to make a new TLib project from scratch and get to Debug&Run, using either IAR Embedded Workbench, or AVR Studio with WinAVR GCC. For further details please refer to the appropriate documents listed in Section 2.

## Section 1 What is TLib?



TLib is a Library Module that can be linked with your application to provide Capacitive Touch Sensing.

- For those not familiar with libraries, or libraries linked to hardware, then it may help to think of TLib as just another MCU Module that can optionally control some I/O pins, similar to hardware modules like WDT, I2C, UART, Timer, ADC …
- All the modules provide registers or settings to select which MCU pins they use.
- There are AppNotes with sample code showing how to use the hardware modules, similarly there are a Guide and examples showing how to use the software TLib module.
- Much of ROM and Stacks used by a TLib example would be required for a non-touch application, so incremental resource usage to add Touch is less than in the TLib guide.

## Section 2 Documents and Tools

Ensure you have all the required software installed, and the appropriate equipment and documentation

★ You may wish to completely uninstall previous versions of all software packages for a fresh start.

a) Install Atmel Tools: http://www.atmel.com ➔Products ➔ Touch Solutions ➔ QTouch Library ➔ QTouch Suite

| | | |
|---|---|---|
| QTouch Suite: http://www.atmel.com/products/touchsoftware/qtouchsuite.asp?family_id=702 | | |
| Download AVR Studio<br>http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725 | ○ Install AVR Studio (4.17 or newer)<br>▪ required for ICE support<br>▪ includes Debugger for GCC (or IAR) | |
| Download QTouch Library<br>http://www.atmel.com/dyn/products/tools_card_touch.asp?tool_id=4627 | ○ QTouch$^{TM}$ Library User Guide<br>○ Install QTouch Studio (with Visual Studio, .NET)<br>○ Install QTouch Library (3.1 or newer) | Optional:<br>○ Touch Sensors Design Guide<br>○ AVR252 (pdf and zip)<br>○ AVR254 (pdf and zip) |

b) Install Development Environment, choices:

IAR) IAR Embedded Workbench – See Section 3 TLib with IAR

- http://www.iar.com/website1/1.0.1.0/107/1/index.php – All Licenses OK: Full, 30 Day, Free Kick Start 4
- This Guide was prepared using: IAR Embedded Workbench for Atmel AVR, v. 5.30, 4K Kickstart edition [EWAVR-KS-WEB-5302.exe]
- This Guide uses IAR Embedded Workbench to do both Compiling and Debugging (Option: Debug using AVR Studio)

GCC) GCC with Debug using AVR Studio – See Section 4 TLib with GCC

- Reference: http://support.atmel.no/bin/customer?custSessionKey=&customerLang=en&noCookies=true&action=viewKbEntry&id=226
- WinAVR GCC: http://sourceforge.net/projects/winavr/files/
- This Guide was prepared using: WinAVR-20090313-install.exe and AVR Studio 4.17 (build 666) (112 MB, updated 7/09)

c) Prepare test PCB – TLib Demo unit, or your own PCB design with Atmel AVR MCU and Touch Sensor Patterns

- The Datasheet for the Atmel AVR IC you will use
- Please see Help in QTouch Studio for information about TLib Demos, including Schematics and Programming Tools.
- This Guide uses TLIB Demos AVRTS2080A and AVRTS2080B as examples (equal to EVK2080A/EVK2080B)

d) Prepare ICE or ISP compatible with your PCB or the TLib Demo

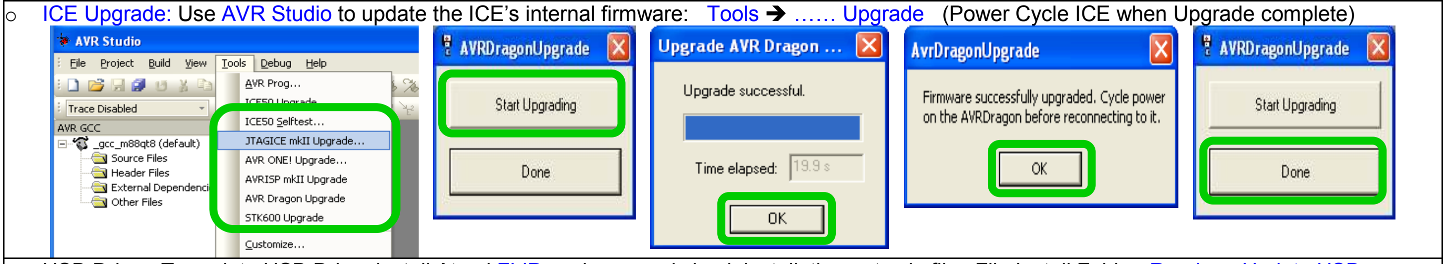
- For compatible Development Tools please refer to AVR IC's datasheet and AVR IC's webpage.
- Please see Help in QTouch Studio for Development Tools compatible with TLib Demos.
- Ensure ICE and drivers are up to date, see Updates section below
- Please ensure ICE ISP frequency set according to ICE Instructions (Typically ≤¼ target's frequency).
- ★ If CPU changes frequency "on the fly" please see Section 5 item t2.

e) Option: Install Additional Software Tools:

- Hawkeye – Viewer for two-wire data (Available from Atmel FAEs, can also display application variables beside TLib, step t4)
- Flip – Tool to In-Circuit Program AT90USB used in TLib Demos http://atmel.com/dyn/products/tools_card.asp?tool_id=3886
- 5030 USB Bridge – In-Circuit Bootload Package for AT90USB MCU as used in TLib Demos (Available from Atmel FAEs)

f) Updates: **Important:** Ensure you check **all** items indicated with: 🟩

- ICE Upgrade: Use AVR Studio to update the ICE's internal firmware: Tools ➔ …… Upgrade (Power Cycle ICE when Upgrade complete)



Upgrade successful.

Time elapsed: 19.9 s

Firmware successfully upgraded. Cycle power on the AVRDragon before reconnecting to it.

- USB Driver: To update USB Driver install Atmel FLIP package, and check installation notes in files Flip Install Folder: Readme, Update USB…
  - Disconnect other USB devices before doing this procedure.
  - USB port may show with different driver names. Plug/Unplug ICE or TLib Demo device to find which device in the list is correct.
- If issue connecting to ICE by USB, try ICE with RS-232 or a USB to RS-232 adapter (ex. JTAG ICE MKII has both USB and RS-232). USB issues are usually due to Windows Driver issues and corrected by updating the driver, but in some rare instance it is possible that some other software may conflict. A reformat and clean Windows install will certainly cure the issue by clearing software history, but that isn't always reasonable.

## Section 3 TLib with IAR

**Important:** Ensure you check **all** items indicated with: 🟩

i1. Create a **Project Folder** – a new empty folder for the project files, the folder location is your choice.

i2. Prepare project files: copy to Project Folder created in step i1, or add from original location: Header ***.h**, Library ***.r90**, Example ***.c**

- Refer to the Tables and Examples listed in the Atmel QTouch Library User Guide
- Default TLib install path: [TLib Path] = C:\Program Files\Atmel\Atmel QTouch Libraries 3.1\ (Use actual path of your TLib installation)
- **TLib API Header:** Copy to Project Folder, or step i4 do: Project➔Options➔C/C++Compiler➔Preprocessor➔Additional Include directories

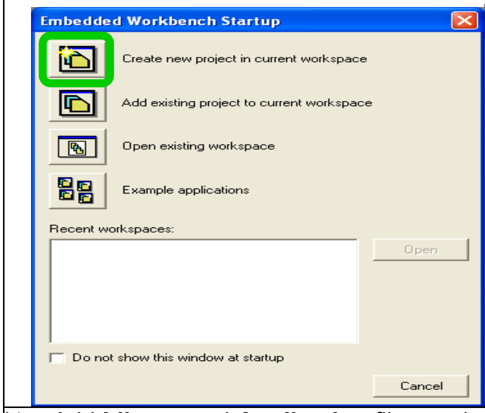
  All TLib projects: [TLib Path]\include\touch_api.h
- **Library:** Copy file to Project Folder, or at step i4 do: Project➔Options➔C/C++Compiler➔…

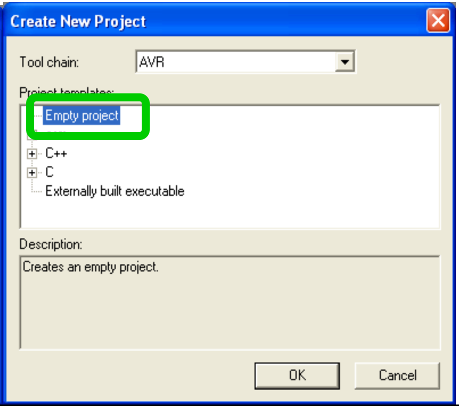
| | |
|---|---|
| AVRTS2080A Keys/Slider/Rotor: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QTouch library\library files\libv1g4-8qt-k-2rs.r90 |
| AVRTS2080A only Keys: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QTouch library\library files\libv1g4-8qt-k-0rs.r90 |
| AVRTS2080B Keys/Slider/Rotor: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QMatrix library\library files\libt88_8qm_4x_2y_krs_2rs_YL_LO_NIB.r90 |
| AVRTS2080B only Keys: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QMatrix library\library files\libt88_8qm_4x_2y_k_0rs_YL_LO_NIB.r90 |

- **Application:** Copy file to Project Folder: (Note: 2080 examples have different sensor assignments from other library examples)

| | |
|---|---|
| AVRTS2080A: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QTouch library\Example projects\2080A_iar_example\main_atmega88.c |
| AVRTS2080B: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QMatrix library\Example projects\TS2080B_qm_example_iar\main.c |

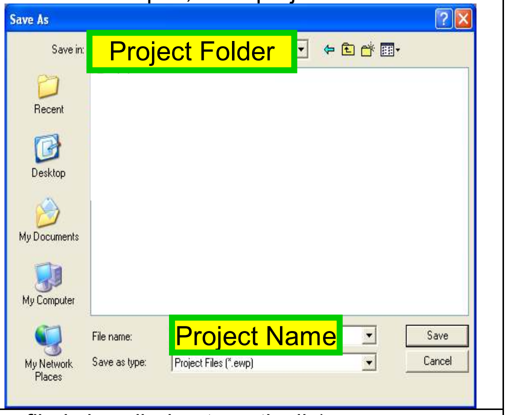
i3. Start IAR Embedded Workbench, create a new empty project, select Project Folder created in step i1, save project.


Embedded Workbench Startup
Create new project in current workspace
Add existing project to current workspace
Open existing workspace
Example applications
Recent workspaces:
Open
Do not show this window at startup
Cancel

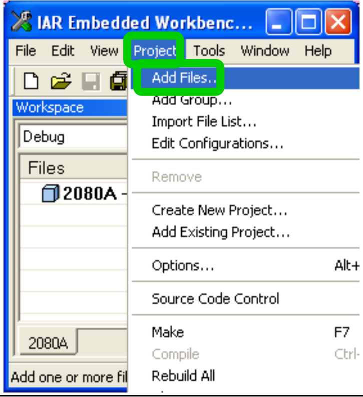


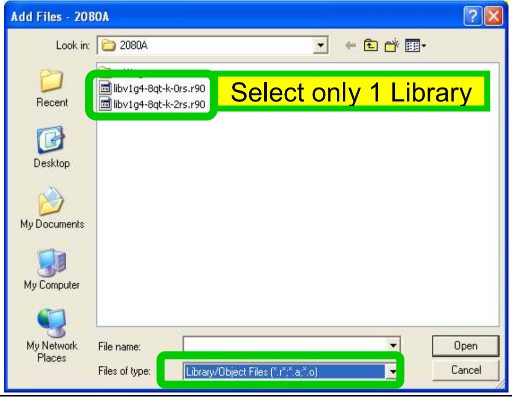
Tool chain: AVR

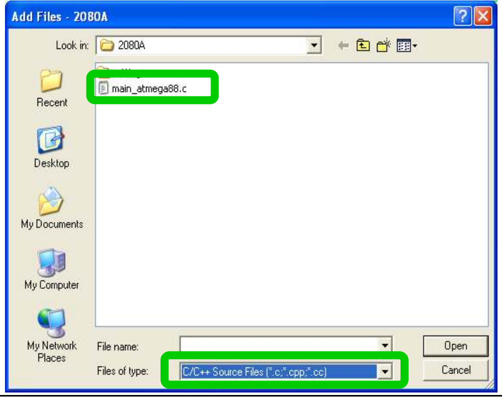
Description: Creates an empty project.



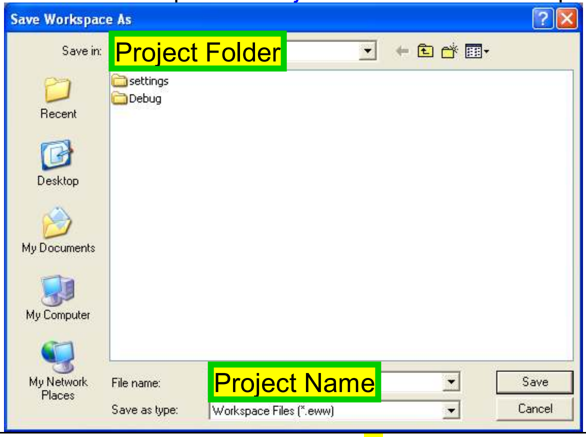
i4. Add **Library** and **Application** files copied in step i2: Project ➔ Add Files… (**Header** file is handled automatically)







i5. Save the Workspace in Project Folder created in step i1: File ➔ Save Workspace.

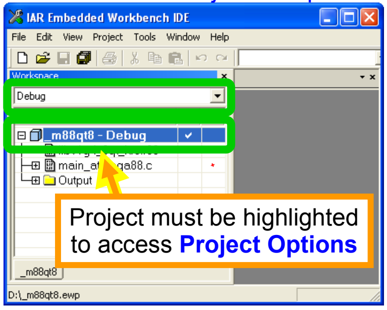


i6. For **Debug** Build continue at step i7, or for **Release** Build continue at step i21

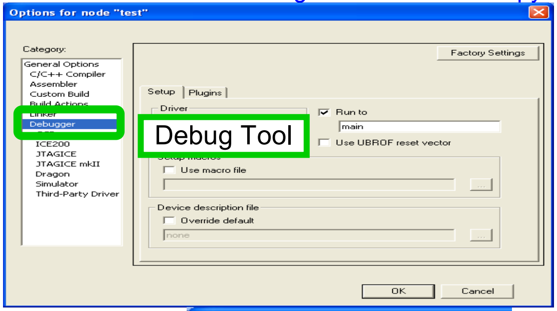
## IAR - Build for Debug

i7. For debugging in IAR Embedded Workbench select Debug Build with:

- ○ Select Debug Tool (ICE): Project ➔ Options ➔ Debugger = JTAG ICE MKII, Dragon …
- ○ Build format: Project ➔ Options ➔ Linker ➔ Format ⦿Debug Information for C-Spy (This is Default for new IAR Projects)

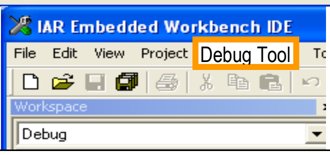


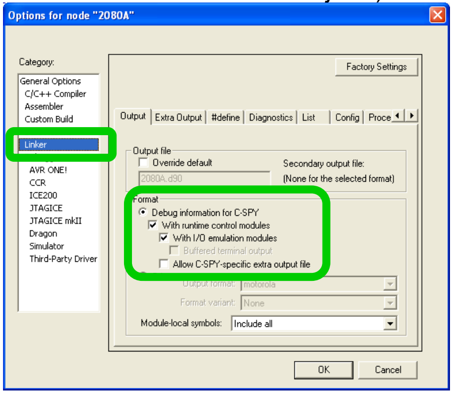
Project must be highlighted to access **Project Options**



Debug Tool

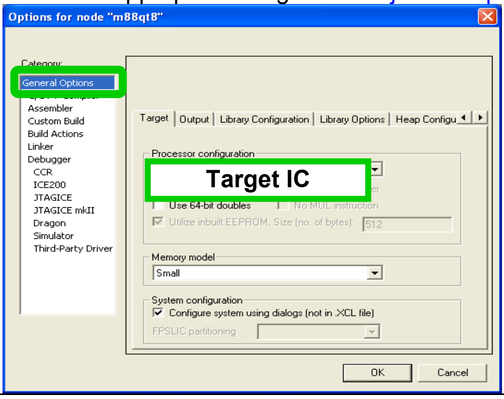
Selected Debug Tool is shown in the Menu bar





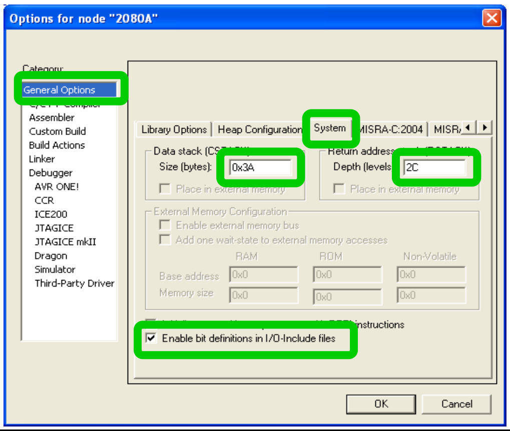
Format: ● Debug information for C-SPY; ☑ With runtime control modules; ☑ With I/O emulation modules; □ Buffered terminal output; □ Allow C-SPY-specific extra output file

i8. Select the appropriate Target IC: Project ➔ Options ➔ General options ➔ Target



- ○ AVRTS2080A - ATmega88
- ○ AVRTS2080B - ATtiny88

i9. Enable I/O Bit Definitions and set size of Stacks: Project ➔ Options ➔ General Options ➔ System



AVRTS2080A Keys/Slider/Rotor:

| | | |
|---|---|---|
| ○ CSTACK ≥ 0x40 (64) | Data |
| ○ RSTACK ≥ 0x2C (44) | Return Addresses |

AVRTS2080A only Keys:

| | |
|---|---|
| ○ CSTACK ≥ 0x30 (48) | Data |
| ○ RSTACK ≥ 0x28 (40) | Return Addresses |

AVRTS2080B Keys/Slider/Rotor:

| | |
|---|---|
| ○ CSTACK ≥ 0x35 (53) | Data |
| ○ RSTACK ≥ 0x20 (32) | Return Addresses |

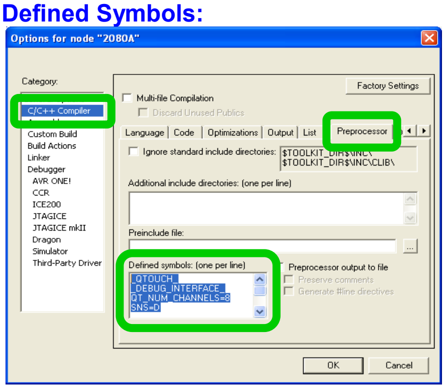
AVRTS2080B only Keys:

| | |
|---|---|
| ○ CSTACK ≥ 0x25 (53) | Data |
| ○ RSTACK ≥ 0x20 (32) | Return Addresses |

If application performs any additional functions then stack sizes should be appropriately adjusted.

**i10.** Set TLib build options: Project ➔ Options ➔ C/C++ compiler ➔ Preprocessor Tab (or put in C file before: `#include "touch_api.h"`)

Add to **Defined Symbols:**

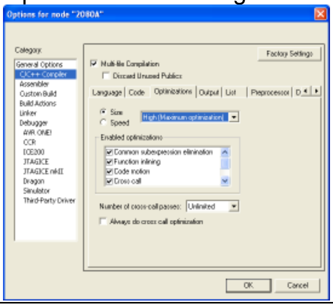


| TLib Configuration, see details in QTouch Library User Guide | |
|---|---|
| _DEBUG_INTERFACE_ | Remove to save memory,<br>Add for QTouch Studio or Hawkeye |
| QT_DELAY_CYCLES=** | Match to Sensor Tuning (QTouch Charge Time) |
| DELAY_CYCLES=** | Match to Sensor Tuning (QMatrix Dwell Time) |

| **QTouch - Keys/Slider/Rotor** | **QTouch - only Keys** |
|---|---|
| _QTOUCH_<br>QT_NUM_CHANNELS=8<br>QT_DELAY_CYCLES=1<br>SNS=D<br>SNSK=B<br>_DEBUG_INTERFACE_<br>_ROTOR_SLIDER_ | _QTOUCH_<br>QT_NUM_CHANNELS=8<br>QT_DELAY_CYCLES=1<br>SNS=D<br>SNSK=B<br>_DEBUG_INTERFACE_ |
| **QMatrix - Keys/Slider/Rotor** | **QMatrix - only Keys** |
| _QMATRIX_<br>QT_NUM_CHANNELS=8<br>DELAY_CYCLES=4<br>PORT_X=B<br>PORT_YA=D<br>PORT_YB=C<br>PORT_SMP=D<br>SMP_BIT=7<br>_DEBUG_INTERFACE_<br>QT_MAX_NUM_ROTORS_SLIDERS=2<br>_ROTOR_SLIDER_ | _QMATRIX_<br>QT_NUM_CHANNELS=8<br>DELAY_CYCLES=4<br>PORT_X=B<br>PORT_YA=D<br>PORT_YB=C<br>PORT_SMP=D<br>SMP_BIT=7<br>_DEBUG_INTERFACE_<br>QT_MAX_NUM_ROTORS_SLIDERS=0 |

i11. Optional Settings (Depending on project requirements):

Optimization: Size: High



Options for node "2080A"

Category: General Options, C/C++ Compiler, Assembler, Custom Build, Build Actions, Linker, Debugger, AVR ONE!, CCR, ICE200, JTAGICE, JTAGICE mkII, Dragon, Simulator, Third-Party Driver

☑ Multi-file Compilation
□ Discard Unused Publics

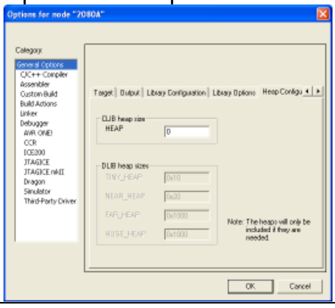
Language | Code | Optimizations | Output | List | Preprocessor | D

● Size ○ Speed: High (Maximum optimization)

Enabled optimizations:
☑ Common subexpression elimination
☑ Function inlining
☑ Code motion
☑ Cross call

Number of cross-call passes: Unlimited
□ Always do cross call optimization

Option: No Heap



Options for node "2080A"

Category: General Options

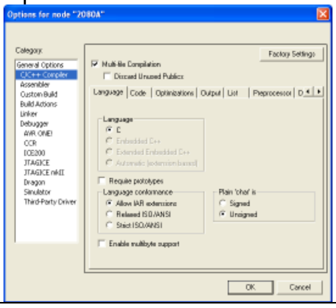
Target | Output | Library Configuration | Library Options | Heap Configu

CLIB heap size
HEAP: 0

DLIB heap sizes
TINY_HEAP: 0x10
NEAR_HEAP: 0x20
FAR_HEAP: 0x1000
HUGE_HEAP: 0x1000

Note: The heaps will only be included if they are needed.

Option: Multi-File



Options for node "2080A"

Category: C/C++ Compiler

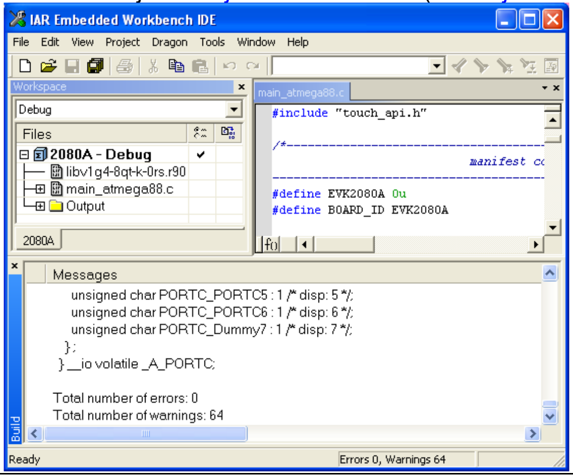
☑ Multi-file Compilation
□ Discard Unused Publics

Language | Code | Optimizations | Output | List | Preprocessor | D

Language: ● C ○ Embedded C++ ○ Extended Embedded C++ ○ Automatic (extension based)
□ Require prototypes
Language conformance: ● Allow IAR extensions ○ Relaxed ISO/ANSI ○ Strict ISO/ANSI
Plain 'char' is: ○ Signed ● Unsigned
□ Enable multibyte support

i12. Save the Workspace: File ➔ Save Workspace.

i13. Build the Project: Project➔Rebuild All (or: Project➔Make)



IAR Embedded Workbench IDE

Workspace: Debug

Files: 2080A - Debug ✔, libv1g4-8qt-k-0rs.r90, main_atmega88.c, Output

main_atmega88.c

```
#include "touch_api.h"

/*----------------------------------------------
                                    manifest cc
------------------------------------------------
#define EVK2080A 0u
#define BOARD_ID EVK2080A
```

Messages

```
unsigned char PORTC_PORTC5 : 1 /* disp: 5 */;
unsigned char PORTC_PORTC6 : 1 /* disp: 6 */;
unsigned char PORTC_Dummy7 : 1 /* disp: 7 */;
};
} __io volatile _A_PORTC;

Total number of errors: 0
Total number of warnings: 64
```

Ready | Errors 0, Warnings 64

- If any errors then check that you have a matching set of:
  Target AVR IC
  Header.h
  Library.r90
  Application.c

- **Known Issue – Linker Warning w6:**
  Builds for some AVR will present this warning, please ignore.

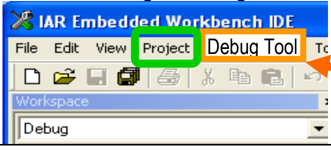
```
Linking
Warning[w6]: Type conflict for external/entry "_A_DDRC", in module main_atmega88
against external/entry in module burst_10_BC;
class/struct/union field/base types do not match for field/base ''; class/struct/union
field names do not match: DDRC_Dummy7 vs DDRC_DDC7
:
:
Total number of errors: 0
Total number of warnings: **
```

i14. Connect Programmer. For photos of ICE connections see QTouch Studio – Help – In Circuit reprogramming

- Powerdown Target and ICE (or unplug USB from each)
- Connect ICE to Target (ISP 6wire, or dW 2wire if debugWire was already enabled)
- Recheck Pin1 is correctly connected to Target (Label PCB and ICE Cable)
- Connect ICE to PC (USB or RS-232), If power switch then turn on ICE
- Powerup Target (Connect USB), If power switch then turn on Target

*Avoid misconnections:
Make pin alignment mark on PCB and ICE using a bright colour paint pen:
ISP6pin5: Gnd
ISP6pin6: Reset-/dW

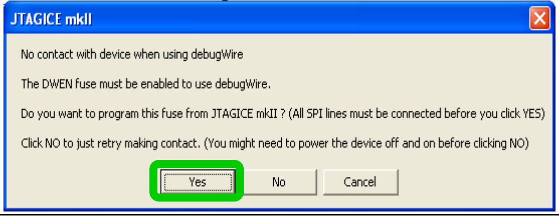
i15. Load into Programming Tool: Project➔Download and Debug



Selected Debug Tool is shown here

i16. You may need to enable Debug Wire – dW is disabled in new ICs, for Power Measurements, and for Release Builds
Project➔Download and Debug ➔Yes➔Power Down Target ➔ Power Up Target ➔OK

- To enable debugWire ensure ICE is connected to Target using ISP 6wire cable (Vcc, SPI/ISP, Reset/dW, Gnd)



JTAGICE mkII

No contact with device when using debugWire

The DWEN fuse must be enabled to use debugWire.

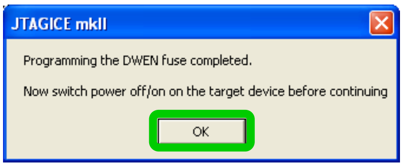
Do you want to program this fuse from JTAGICE mkII ? (All SPI lines must be connected before you click YES)

Click NO to just retry making contact. (You might need to power the device off and on before clicking NO)

Yes | No | Cancel

➔ Power Down Target
(Unplug USB from AVRTS2080)
🡓
Option: ICE cable change
ISP 6wire to dW 2wire
🡓
Power Up Target ➚
(Connect USB to AVRTS2080)

DWEN Fuse for dW:



JTAGICE mkII

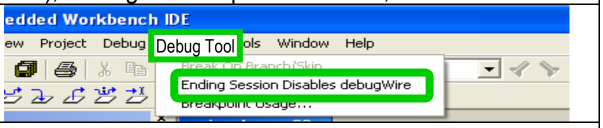
Programming the DWEN fuse completed.

Now switch power off/on on the target device before continuing

OK

i17. After enabling debugWire you can switch ICE to use dW 2wire cable (dW, Gnd), freeing the ISP pins for Touch, SPI…

i18. To disable debugWire for Low Power measurements or Release Builds:
- In Debug Tool Menu enable: ✓Ending Session Disables DebugWire
- Exit Debug mode: Debug ➔ Stop Debugging (auto disables DWEN fuse)
- ICE will need ISP 6wire cable for further actions



edded Workbench IDE
ew Project Debug Debug Tool ols Window Help
Break On Branch/Skip
Ending Session Disables debugWire
Breakpoint Usage...

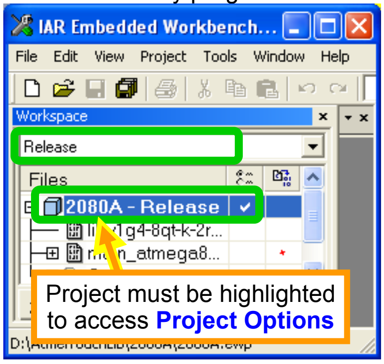
i19. Run the Program and do your tests: Debug➔Go
If your program outputs appropriate diagnostic data then you may view the data using QTouch Studio or Hawkeye
Note: Pins connected to ICE won't be able to Touch Detect. Use debugWire cable to free these pins (step i17).

i20. To stop debugging: Debug➔Stop Debugging
For Release Build continue at step i21, if finished continue at step i32

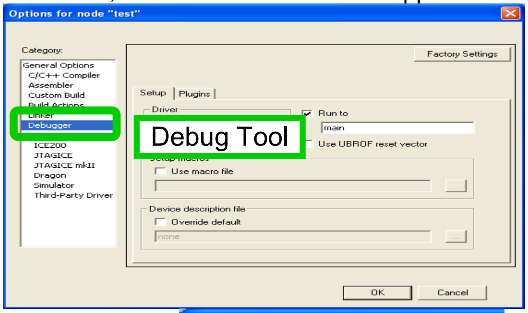
## IAR - Build for **Release** (or for Debugging in AVR Studio)

i21. Select **Release** Build to create a file for final testing and production (Option: use other software to FLASH, like AVR Studio)

- ○ Select Programming Tool (ISP, ICE): Project ➔ Options ➔ Debugger = STK600, JTAG ICE MKII, Dragon, …
- ○ Select output format. Typical formats include "motorola", "intel hex"… (Production output typically doesn't contain debug info)
- ○ Some tools only program ICs mounted on PCB, while tools like STK600 support in-circuit programming and programming sockets.

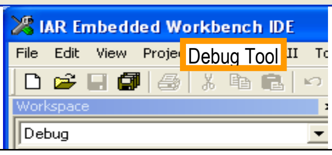


Project must be highlighted to access **Project Options**

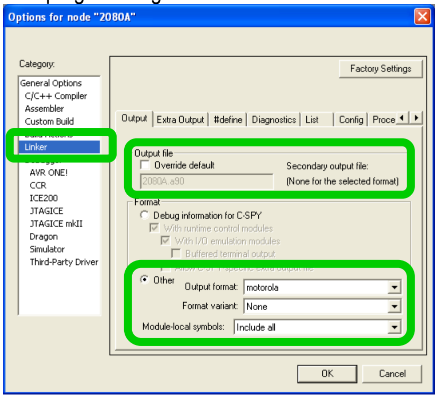


Debug Tool

Selected Debug Tool is shown in the Menu bar



For programming with IAR use Motorola format.



Output file: □ Override default, 2080A.a90
Secondary output file: (None for the selected format)
Format: ○ Debug information for C-SPY; ● Other
Output format: motorola
Format variant: None
Module-local symbols: Include all

i22. Select the appropriate Target IC, see Step i8: Project ➔ Options ➔ General options ➔ Target

i23. Enable I/O Bit Definitions and set size of Stacks, see Step i9: Project ➔ Options ➔ General Options ➔ System

i24. Set TLib build options for Release: Project ➔ Options ➔ C/C++ compiler ➔ Preprocessor Tab

| | |
|---|---|
| NDEBUG | This IAR definition is used for Release Builds |
| ★★★ | see Step i10 for required TLib Definitions |
| _DEBUG_INTERFACE_ | Release code usually has a dedicated interface, so if unused remove the TLib Diagnostics code |

i25. Save the Workspace: File ➔ Save Workspace.

i26. Build the Project, see Step i13: Project➔Rebuild All (or: Project➔Make)

i27. If you are using different software for programming then skip to step i32

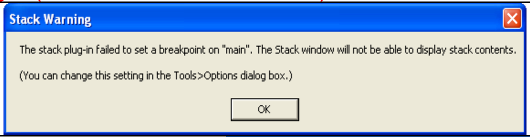
For programming with other software it may be necessary to disable DebugWire, see step i18

i28. Connect Programmer using ISP 6wire cable, see Step i14

Recheck Pin1 is correctly connected to Target (Label PCB and ICE Cable)

i29. Load into Programming Tool:
Project➔Download and Debug

Don't Click RUN



Stack Warning

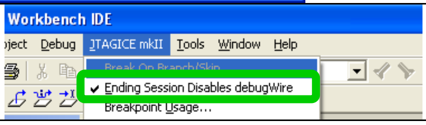
The stack plug-in failed to set a breakpoint on "main". The Stack window will not be able to display stack contents.

(You can change this setting in the Tools>Options dialog box.)

OK

Debug information isn't included in release build so may see this message.

i30. Set for debugWire disable upon exit to minimize IC power:
Debugger Menu enable: ✓Ending Session Disables debugWire.


Workbench IDE
oject Debug JTAGICE mkII Tools Window Help
Break On Branch/Skip
✓ Ending Session Disables debugWire
Breakpoint Usage…

i31. Finished Programming: Debug➔Stop Debugging (debugWire module in IC is disabled through setting in step i30)

## IAR - Finish

i32. Powerdown Target (Unplug USB from EVK2080)

i33. Power down Programmer.

i34. Disconnect Programmer from Target.

i35. Powerup Target and test it.

i36. Log results.

i37. Exit all software tools

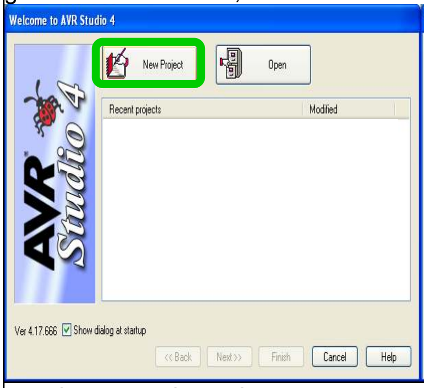
## IAR Completed

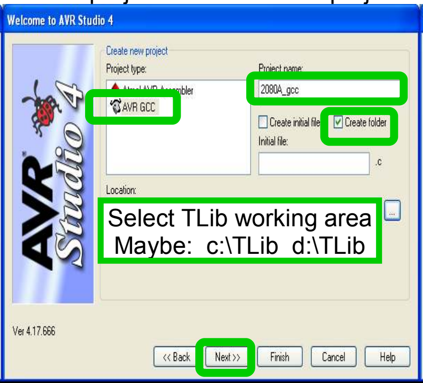
## Section 4 TLib with GCC

**Important:** Ensure you check **all** items indicated with: 🟩

- GCC Style Note:
  - For data in ROM using GCC see PROGMEM: http://www.nongnu.org/avr-libc/user-manual/pgmspace.html
  - AVR Libc Guide: http://www.nongnu.org/avr-libc/user-manual/index.html

g1. Start AVR Studio, create a new AVR GCC project in desired new project folder, select ICE and AVR IC, save project





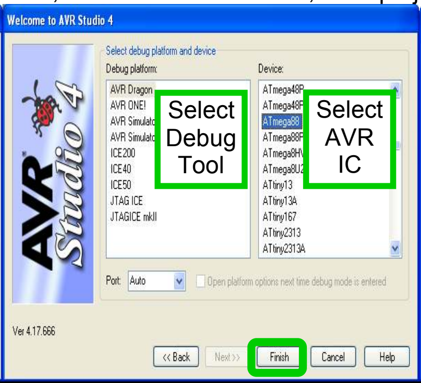
Project type: AVR GCC

Project name: 2080A_gcc

□ Create initial file ☑ Create folder

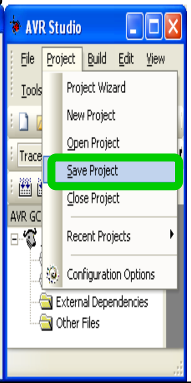
Location:

Select TLib working area
Maybe: c:\TLib d:\TLib



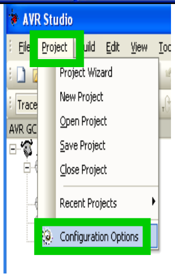
Select Debug Tool

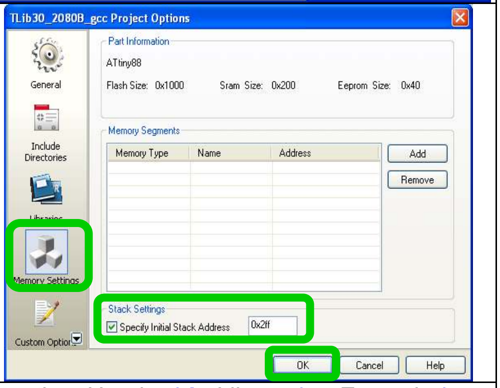
Select AVR IC



g2. Check the Stack Starting address matches the Datasheet:
Project ➔ Configuration Options ➔ Memory Settings ➔ Stack settings

For most AVR the stack is automatically set,
but for **ATtiny88** used in demo AVRTS2080B use:
☑ Specify Initial Stack address [ 0x2FF ]





Part Information

ATtiny88

Flash Size: 0x1000 Sram Size: 0x200 Eeprom Size: 0x40

Stack Settings

☑ Specify Initial Stack Address 0x2ff

g3. Prepare project files: copy to Project Folder created in step g1, or add from original location: Header ***.h**, Library ***.a**, Example ***.c**

- Refer to the Tables and Examples listed in the Atmel QTouch Library User Guide
- Default TLib install path: [TLib Path] = C:\Program Files\Atmel\Atmel QTouch Libraries 3.1\ (Use actual path of your TLib installation)
- **TLib API Header:** Copy to Project Folder

| | |
|---|---|
| All TLib projects: | [TLib Path]\include\touch_api.h |

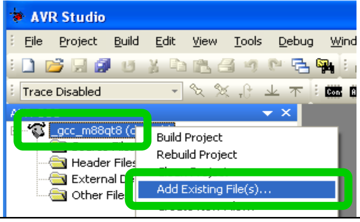
- **Library:** Copy file to Project Folder

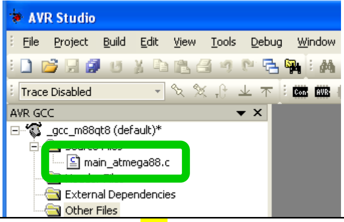
| | |
|---|---|
| AVRTS2080A Keys/Slider/Rotor: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QTouch library\library files\libavr4g2-8qt-k-2rs.a |
| AVRTS2080A only Keys: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QTouch library\library files\libavr4g2-8qt-k-0rs.a |
| AVRTS2080B Keys/Slider/Rotor: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QMatrix library\library files\libt88_8qm_4x_2y_krs_2rs_YL_LO_NIB.a |
| AVRTS2080B only Keys: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QMatrix library\library files\libt88_8qm_4x_2y_k_0rs_YL_LO_NIB.a |

- **Application:** Copy file to Project Folder: (Note: 2080 examples have different sensor assignments from other library examples)

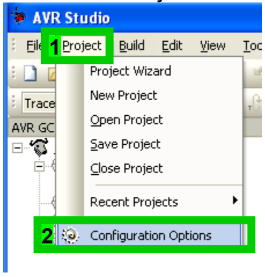
| | |
|---|---|
| AVRTS2080A: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QTouch library\Example projects\2080A_gnu_example \main_atmega88.c |
| AVRTS2080B: | [TLib Path]\megaAVR, tinyAVRand XMEGA libray\QMatrix library\Example projects\TS2080B_qm_example_gnu\main.c |

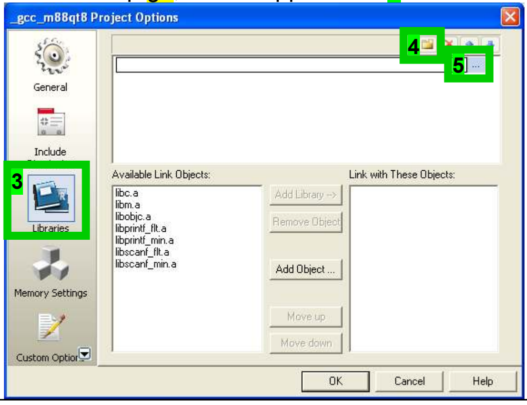
g4. Add Application file to Project: Right Click on project ➔ Add Existing File(s)… ➔ add **.c** from step g3

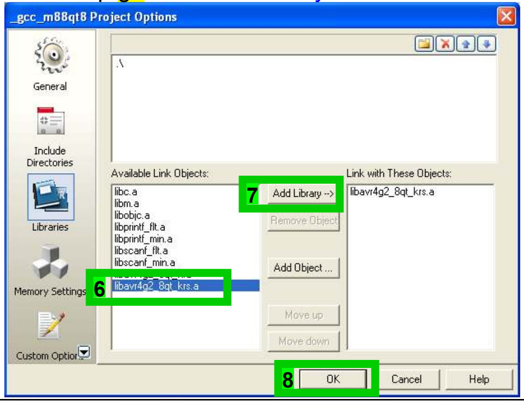




g5. Select Library Path, and add library **.a** from step g3
1.Project ➔ 2.Configuration Options ➔ 3.Libraries ➔ 4.Folder ➔ 5.Path
➔ Select Project folder created in step g1, Folder appears as .\ ➔ 6.Select **.a** from step g3 ➔ 7.Add Library ➔ 8.OK

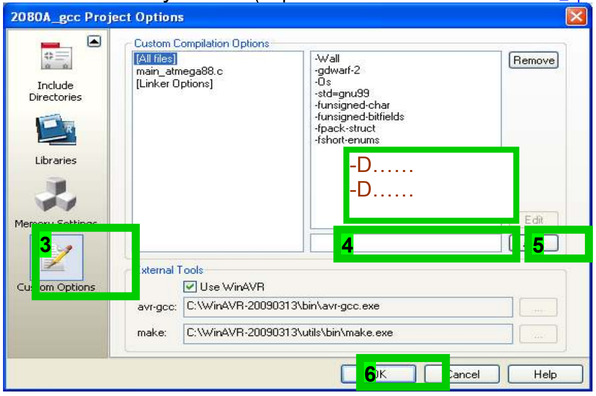






g6. Set TLib build options: 1 Project ➔ 2 Configuration Options ➔ 3 Custom Options ➔ 4 Item to Add: -D… ➔ 5 Add ➔ 6 OK
Repeat Steps 4 and 5 for each item. More…

Add to Defined Symbols: (or put in C file before: #include "touch_api.h")



2080A_gcc Project Options

Custom Compilation Options

[All files]
main_atmega88.c
[Linker Options]

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

External Tools

☑ Use WinAVR

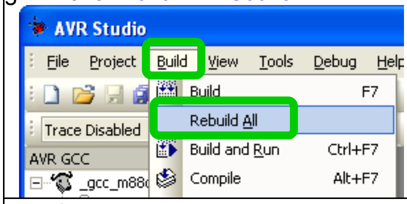
avr-gcc: C:\WinAVR-20090313\bin\avr-gcc.exe

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

| TLib Configuration, see details in QTouch Library User Guide | |
|---|---|
| -D_DEBUG_INTERFACE_ | Remove to save memory, or<br>add for QTouch Studio or Hawkeye |
| -DQT_DELAY_CYCLES=** | Match to Sensor Tuning (QTouch Charge Time) |
| -DDELAY_CYCLES=** | Match to Sensor Tuning (QMatrix Dwell Time) |

| **QTouch - Keys/Slider/Rotor** | **QTouch - only Keys** |
|---|---|
| -D_QTOUCH_ | -D_QTOUCH_ |
| -DQT_NUM_CHANNELS=8 | -DQT_NUM_CHANNELS=8 |
| -DQT_DELAY_CYCLES=1 | -DQT_DELAY_CYCLES=1 |
| -DSNS=D | -DSNS=D |
| -DSNSK=B | -DSNSK=B |
| -D_DEBUG_INTERFACE_ | -D_DEBUG_INTERFACE_ |
| -D_ROTOR_SLIDER_ | |
| **QMatrix - Keys/Slider/Rotor** | **QMatrix - only Keys** |
| -D_QMATRIX_ | -D_QMATRIX_ |
| -DQT_NUM_CHANNELS=8 | -DQT_NUM_CHANNELS=8 |
| -DDELAY_CYCLES=4 | -DDELAY_CYCLES=4 |
| -DPORT_X=B | -DPORT_X=B |
| -DPORT_YA=D | -DPORT_YA=D |
| -DPORT_YB=C | -DPORT_YB=C |
| -DPORT_SMP=D | -DPORT_SMP=D |
| -DSMP_BIT=7 | -DSMP_BIT=7 |
| -D_DEBUG_INTERFACE_ | -D_DEBUG_INTERFACE_ |
| -D QT_MAX_NUM_ROTORS_SLIDERS=2 | -DQT_MAX_NUM_ROTORS_SLIDERS=0 |
| -D_ROTOR_SLIDER_ | |

g7. Build: Build ➔ Rebuild All



If any errors then check that you have a matching set of:
Target IC, Header.h, Library.a, Main.c

WinAVR GCC Notes (compared to IAR):

- There is no difference between Debug Build and Release Build
- There is no need to set Stack sizes

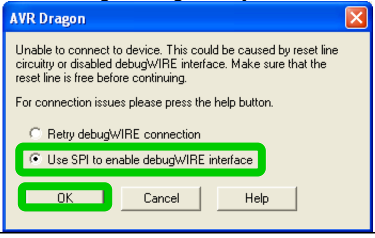
g8. Connect Programmer. For photos of ICE connections see QTouch Studio – Help – In Circuit reprogramming

- Powerdown Target and ICE (or unplug USB from each)
- Connect ICE to Target (ISP 6wire, or dW 2wire if debugWire was already enabled)
- Recheck Pin1 is correctly connected to Target (Label PCB and ICE Cable)
- Connect ICE to PC (USB or RS-232), If power switch then turn on ICE
- Powerup Target (Connect USB), If power switch then turn on Target

*Avoid misconnections:
Make pin alignment mark on PCB and ICE using a bright colour paint pen:
ISP6pin5: Gnd
ISP6pin6: Reset-/dW

g9. Load into Programming Tool: Debug ➔ Start Debugging (or Build ➔ Build and Run)

g10. First Time you use ICE with an AVR you may need to enable debugWire (DWEN Fuse for dW):
Debug ➔ Start Debugging ➔ ⦿ Use SPI ➔ OK ➔ Power Down Target ➔ Power Up Target ➔OK
To enable debugWire ensure ICE is connected to Target using ISP 6wire cable (Vcc, SPI/ISP, Reset/dW, Gnd)
After enabling debugWire you can switch to dW 2wire cable (dW, Gnd), freeing the ISP pins for Touch, SPI…



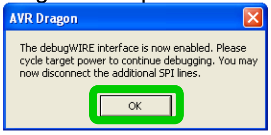
AVR Dragon

Unable to connect to device. This could be caused by reset line circuitry or disabled debugWIRE interface. Make sure that the reset line is free before continuing.

For connection issues please press the help button.

○ Retry debugWIRE connection
● Use SPI to enable debugWIRE interface

➔ Power Down Target
(Unplug USB from EVK2080)
↓
Option: ICE cable change
ISP 6wire to dW 2wire
↓
Power Up Target
(Connect USB to EVK2080) ➚



AVR Dragon

The debugWIRE interface is now enabled. Please cycle target power to continue debugging. You may now disconnect the additional SPI lines.

g11. Run the Program and do your tests: Debug ➔Run

- If your program outputs appropriate diagnostic data then you may view the data using QTouch Studio or Hawkeye
- Note: Pins connected to ICE won't be able to Touch Detect. Use debugWire cable to free these pins (see step g10).

g12. To stop debugging: Debug ➔ Break, then Debug ➔ Stop Debugging

g13. To disable debugWire for Low Power measurements or Release Builds:

- Stop any debugging session in progress: Debug ➔ Break, then Debug ➔ Stop Debugging
- Connect ICE to Target using 6wire ISP cable (You may wish to power down target while switching cables)
- Start fresh debug using: Debug ➔ Start Debugging
- Select ICE Options: ALT-O, or bottom option in Debug pulldown menu: Debug ➔ **** Options (****=ICE you are using)
- Select: Connection ➔ Disable debugWire ➔Yes ➔ wait for message "…leaving debug mode" ➔ OK ➔ OK

❖ In AVR Studio Help see: On-Chip Debugging with the JTAGICE mkII, subsection: Re-enabling the ISP Interface

**GCC - Finish**

g14. Powerdown Target (Unplug USB from EVK2080)

g15. Power down Programmer.

g16. Disconnect Programmer from Target.

g17. Powerup Target and test it.

g18. Log results.

g19. Exit all software tools

**GCC Completed**

# Section 5 **TLib Timing, Low Power, Diagnostics**

This section is still Under Construction - Feel free to suggest topics for this section (through QTouch Forum or avr@atmel.com)

t1. Some options for Timing and Low Power Control (Refer to AVR IC's Datasheet for Low Power modes):

a. Use __delay_cycles () with appropriate count for required period.
b. Like (a) but set CPU to slow before, then fast after, such that AVR is using Low Power during __delay_cycles()
c. Use Timer to generate an interrupt, and use __sleep() with Low Power Mode while waiting.
d. Use Watchdog with low speed oscillator as wakeup source, and use __sleep() with Low Power Mode while waiting
e. Use a pin as an External Interrupt Source, and use __sleep() with Lowest Power Mode while waiting
f. Set for wakeup upon communications event, and use __sleep() with Lowest Power Mode while waiting

For (e) and (f) the Touch Response timing and Low Power Mode are controlled by the External source, and can vary.

t2. If CPU changes frequency "on the fly" then ensure CPU switches back to starting frequency before using debugger functions.

Debug Option
- ○ Assign a spare AVR input pin as **Normal/FastDebug-** (This can be temporary assignment during code development)
- ○ Design code to run CPU as needed when pin is HI (internal pullup), but to keep CPU at full speed when this pin is LO.
- ○ Use a switch or jumper wire to connect Debug Pin to Gnd when need to interrupt code for debugging (Reload, Breakpoint, View memory…).

t3. Interface and Protocol options:

- In some projects the AVR will be both the Host processor and the Touch controller such that an interface isn't required, while in other cases the interface will be predefined by existing equipment, possibly as either a serial protocol or Pin-Per-Key interface (PPK). For cases where a simple new interface protocol is required the below ideas may help, as the diagnostic output for QTouch Studio and Hawkeye isn't suitable for real products do to the high volume of data output.
- An example of a simple single byte protocol, with optional host control of timing/power and sensor configuration:
  - ○ Tx: 1 byte of 8 touch status bits, or 1 slider position (255=NoTouch), or if AKS enabled the can Encode the Touch status like: 0x00~0x7F:Slider7bit, 0x80~0xBF:Wheel6bit, 0xC0~0xEF:Key0~47, 0xF0~0xFD:Error0~13, 0xFE:Calibrating, 0xFF:NoTouch
  - ○ Rx: 1~2 bits of received byte to control sampling response time (power): Fast, Low Power, and FullSleep(Can't Touch), see t1.
  - ○ Rx: Use some bits to select configuration so one firmware file can be used for many products or operating modes, reducing programmed IC stock and production costs. Also for run-time reassign of sensors, ex: Keys⇆Slider, use qt_reset_sensing()
- Multi-byte IO:
  - ○ Simple using Bit7 as First byte Flag: First byte Bit7=1, other bytes Bit7=0, Data in Bits0~6 of each byte, many Keys/Sliders/Wheels.
  - ○ Numerous other protocols exist that include byte synchronization and/or checksum: I2C, DLE Stuffing, STX-ETX, etc.

t4. Hawkeye may be used to monitor Touch Data and application specific data in near real time (Copy-Paste below to *.c and *.txt) This can be helpful if the project needs to be debugged in real time, or if the ICE pins are needed by the application.

Standard diagnostic data output modified for monitoring by Hawkeye to append 3 bytes of application data and a cycle counter.

```
static void report_debug_data( void )
{
   uint8_t i;
   int16_t sensor_delta;
   static uint8_t b_count; // Counter

   output_to_debugger( (uint8_t *) &board_info, (uint8_t) sizeof( board_info ) );
   output_to_debugger( (uint8_t *) &qt_measure_data.channel_signals[0], (uint8_t) sizeof( qt_measure_data.channel_signals ) );
   output_to_debugger( (uint8_t *) &qt_measure_data.channel_references[0], (uint8_t) sizeof( qt_measure_data.channel_references ) );
   for( i = 0u; i < QT_NUM_CHANNELS; i++ )
   {
      sensor_delta = qt_get_sensor_delta( i );
      output_to_debugger( (uint8_t *) &sensor_delta, sizeof( int16_t ) );
   }
   output_to_debugger( (uint8_t *) &qt_measure_data.qt_touch_status, (uint8_t) sizeof( qt_measure_data.qt_touch_status ) );
   output_to_debugger( (uint8_t *) &sensor_config[0], (uint8_t) sizeof( sensor_config ) );//#Channels
   send_debug_byte(b_appdata0);
   send_debug_byte(b_appdata1);
   send_debug_byte(b_appdata2);
   send_debug_byte(b_count ++);
}
```

Hawkeye Control File + 4Bytes

```
D, 1, 1, Model
D, 1, 2, ch_signals0
D, 1, 3, ch_signals1
D, 1, 4, ch_signals2
D, 1, 5, ch_signals3
D, 1, 6, ch_signals4
D, 1, 7, ch_signals5
D, 1, 8, ch_signals6
D, 1, 9, ch_signals7
D, 2, 2, ch_references0
D, 2, 3, ch_references1
D, 2, 4, ch_references2
D, 2, 5, ch_references3
D, 2, 6, ch_references4
D, 2, 7, ch_references5
D, 2, 8, ch_references6
D, 2, 9, ch_references7
-D, 2,11, sensor_deltas0
-D, 2,12, sensor_deltas1
-D, 2,13, sensor_deltas2
-D, 2,14, sensor_deltas3
-D, 2,15, sensor_deltas4
-D, 2,16, sensor_deltas5
-D, 2,17, sensor_deltas6
-D, 2,18, sensor_deltas7
B, 3, 11, sensor_states
D, 3, 12, rotor_slider0
D, 3, 13, rotor_slider1
B, 1, 11, sensorcfg0
B, 1, 12, sensorcfg1
B, 1, 13, sensorcfg2
B, 1, 14, sensorcfg3
B, 1, 15, sensorcfg4
B, 1, 16, sensorcfg5
B, 1, 17, sensorcfg6
B, 1, 18, sensorcfg7
B, 4, 11, AppDataB0
B, 4, 12, AppDataB1
B, 4, 13, AppDataB2
B, 4, 18, CycleCounter
```

Minimized Diagnostic Data output for one key (Ref, Sig, Delta, Status) and an application status word

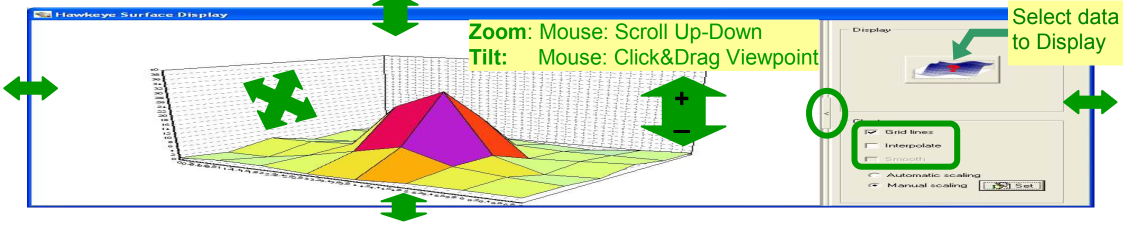
```
static void report_debug_data( void )
{
   int16_t sensor_delta;
   output_to_debugger( (uint8_t *) &qt_measure_data.channel_references[0], (uint8_t) sizeof( qt_measure_data.channel_references[0] ) );
   output_to_debugger( (uint8_t *) &qt_measure_data.channel_signals[0], (uint8_t) sizeof( qt_measure_data.channel_signals[0] ) );
   sensor_delta = qt_get_sensor_delta(0); output_to_debugger( (uint8_t *) &sensor_delta, sizeof( sensor_delta ) );
   output_to_debugger( (uint8_t *) &qt_measure_data.qt_touch_status, (uint8_t) sizeof( qt_measure_data.qt_touch_status ) );
   output_to_debugger( (uint8_t *) &w_appdata0, (uint8_t) sizeof( w_appdata0 ) ); /* 16 bit */
}
```

Minimized Hawkeye File

```
D, 1, 1, Ref
D, 1, 2, Signal
-D, 1,3, Delta
B, 1, 4, Sensor_States
D, 1, 6, AppDataW0
```

t5. Hawkeye Operation:

a. Start [ Hawkeye.exe ], use File→Open to select a Hawkeye Control file with format matching: report_debug_data( void )
b. Logging: Start☑, Finish□, Click **Open** to see data (automatic filename)
c. Hawkeye 3D Graph control using Mouse (click Display Icon to select data to display):



**Zoom**: Mouse: Scroll Up-Down
**Tilt:** Mouse: Click&Drag Viewpoint

Select data to Display

- End -