

# Debugging Techniques

---

**Marten van Dijk, Syed Kamran Haider**  
Department of Electrical & Computer Engineering  
University of Connecticut  
Email: {vandijk, syed.haider}@engr.uconn.edu

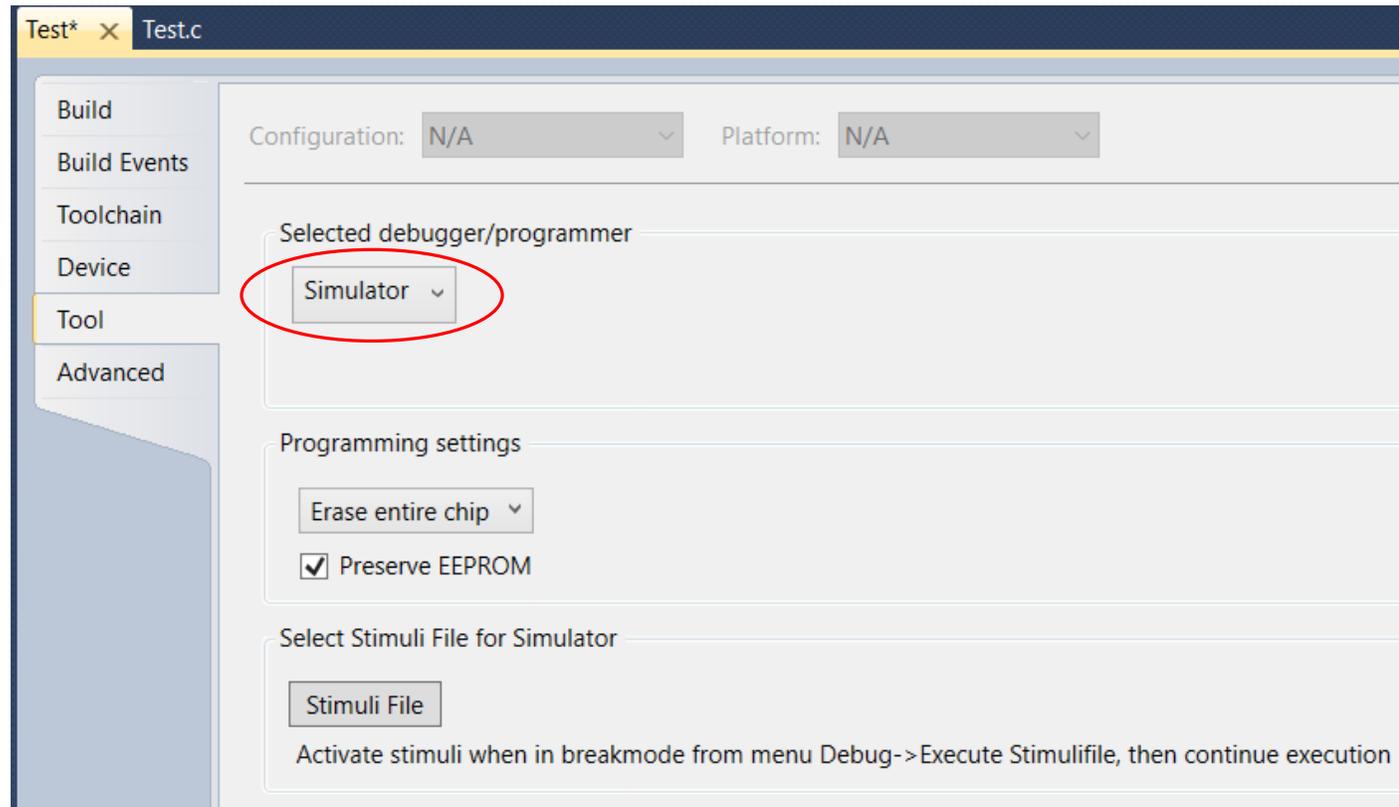
# Debugging Techniques

---

- Debugging in Atmel Studio
  - Simulator mode
  - On-chip debugging using debugWire interface for Xplained Mini kits
- Debugging using Assert library
- Debugging using Hardware Peripherals
  - LEDs, LCD
  - Observing output signals using Oscilloscope

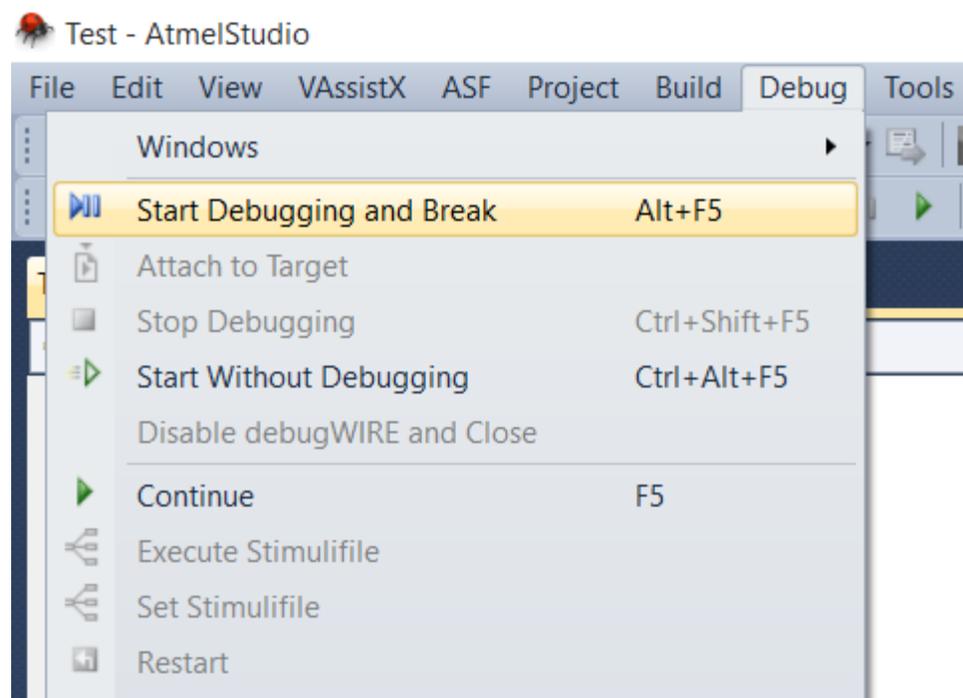
# Debugging in Atmel Studio (Simulator Mode)

- Create a new Atmel Studio project
- Select “Simulator” from the Tool Selection tab



# Starting a Debugging Session

- Build the project. (Hit F7)
- From Debug tab, select “Start Debugging and Break”
- The debugger pauses at the start of main.



```
#include <avr/io.h>

volatile int counter;
volatile int flag;

int main(void)
{
    // Initialize
    counter = 0;
    flag = 0;

    while(1)
    {
        // increment counter
        counter++;

        if (counter == 100)
        {
            // toggle flag
            flag ^= 1;

            // reset counter
            counter = 0;
        }
    }
}
```

# Various Windows in Debugging Session

The screenshot displays the Atmel Studio debugging environment. The main window shows a C program with a while loop. The 'Processor' window shows the current state of the processor, including the Program Counter, Stack Pointer, and various registers. The 'I/O View' window shows the status of various peripheral registers. A 'Watch' window is also visible at the bottom, showing variable values.

**Code in Disassembly Window:**

```
#include <avr/io.h>
int counter;
int flag;

int main(void)
{
    // Initialize
    counter = 0;
    flag = 0;

    while(1)
    {
        // increment counter
        counter++;

        if (counter == 100)
        {
            // toggle flag
            flag ^= 1;

            // reset counter
            counter = 0;
        }
    }
}
```

**Processor View Window:**

Name	Value
Program Counter	0x0000048
Stack Pointer	0x08FD
X Register	0x0104
Y Register	0x08FF
Z Register	0x0000
Status Register	0x0000
Cycle Counter	44
Frequency	16.000 MHz
Stop Watch	2.75 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00

**I/O View Window:**

Name	Value
AD_CONVERTER	
ANALOG_COMPARATOR	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
TWI	
USART0	
WATCHDOG	

**Watch Window:**

Value	Type

Watch Window shows variable values

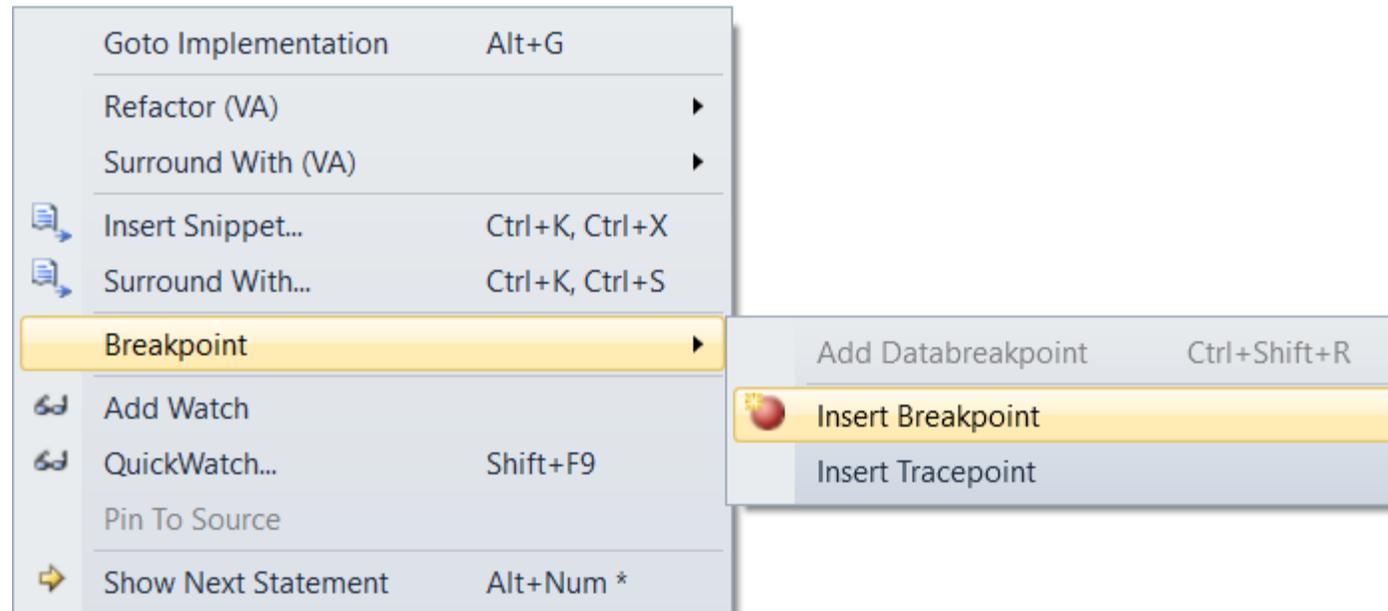
I/O view Window shows peripheral register values

Processor's view Window shows processor status

# Adding a Breakpoint in Debugging Session

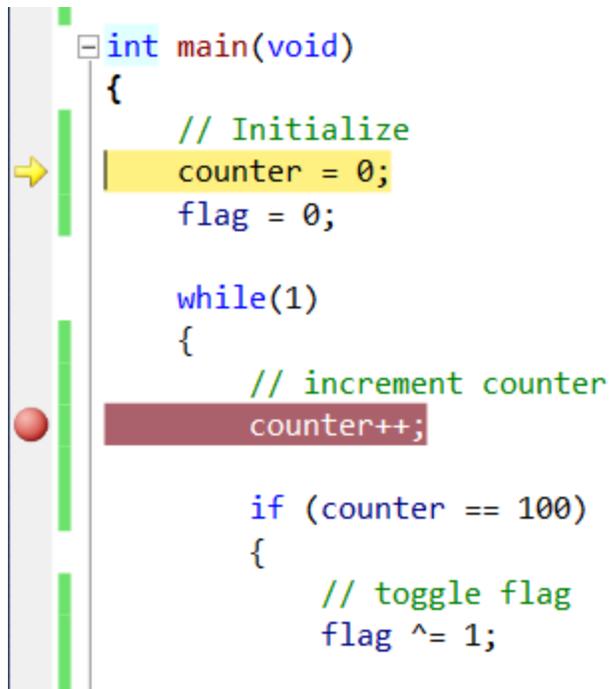
---

- Select any instruction in the code
- Right Click and insert a Breakpoint as follows



# Continue to the next Breakpoint

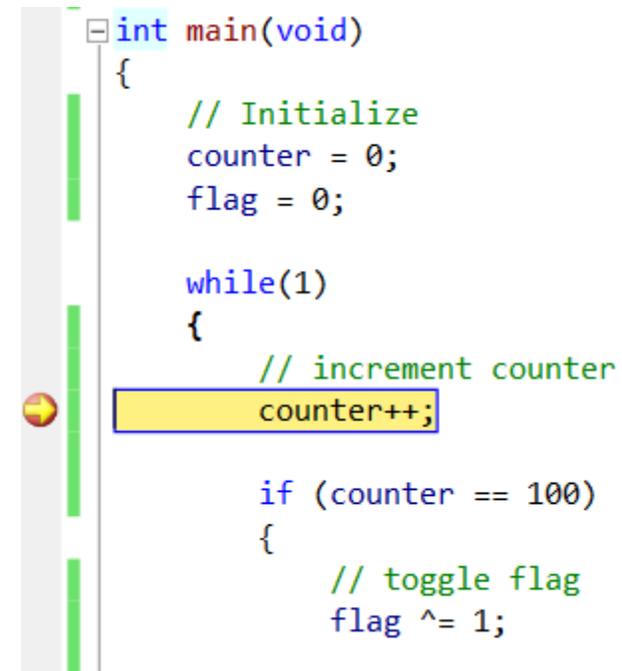
- After inserting a breakpoint, click Continue (F5)
- The program will stop at Breakpoint as shown in the right window.



```
int main(void)
{
    // Initialize
    counter = 0;
    flag = 0;

    while(1)
    {
        // increment counter
        counter++;

        if (counter == 100)
        {
            // toggle flag
            flag ^= 1;
        }
    }
}
```



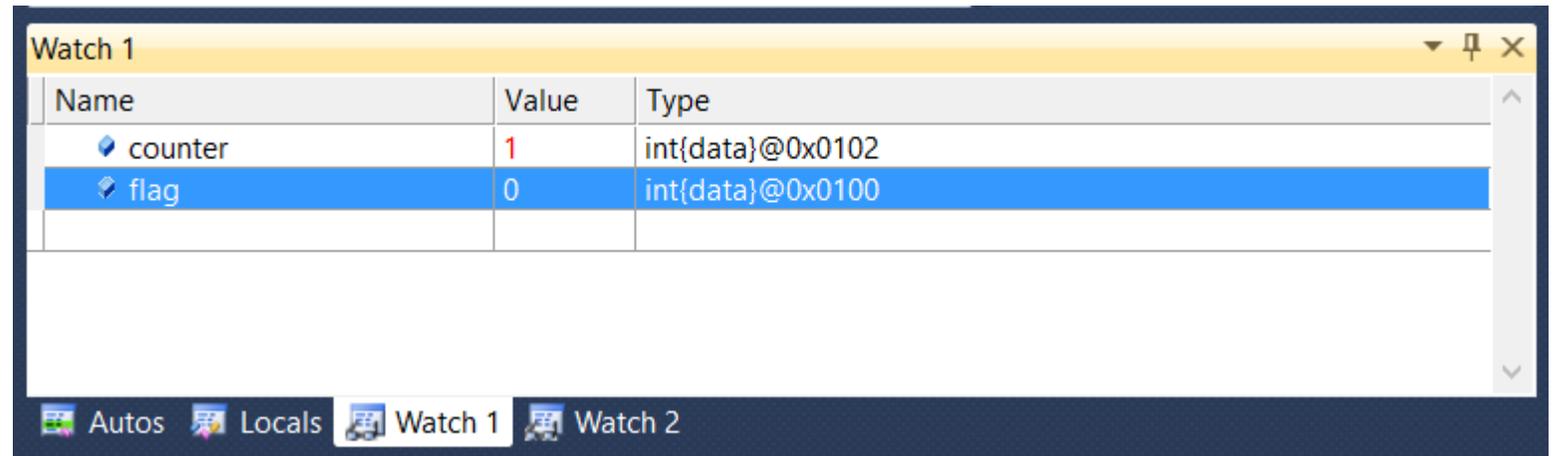
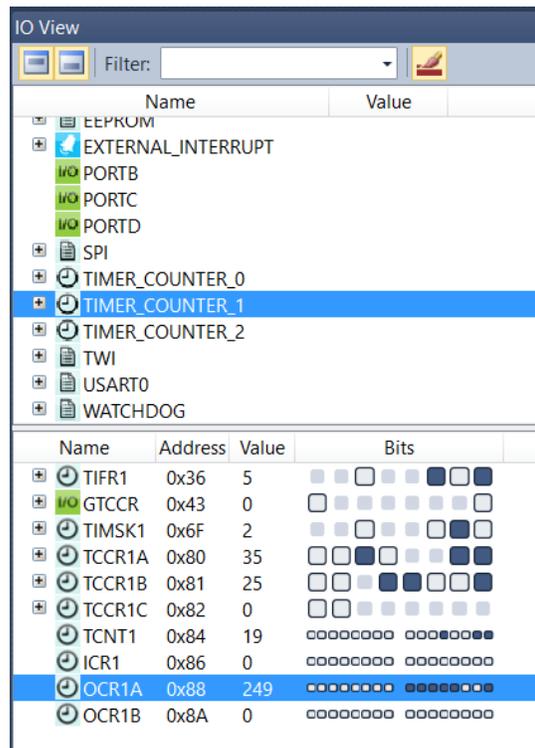
```
int main(void)
{
    // Initialize
    counter = 0;
    flag = 0;

    while(1)
    {
        // increment counter
        counter++;

        if (counter == 100)
        {
            // toggle flag
            flag ^= 1;
        }
    }
}
```

# Observing Register/Variable Values at a Breakpoint

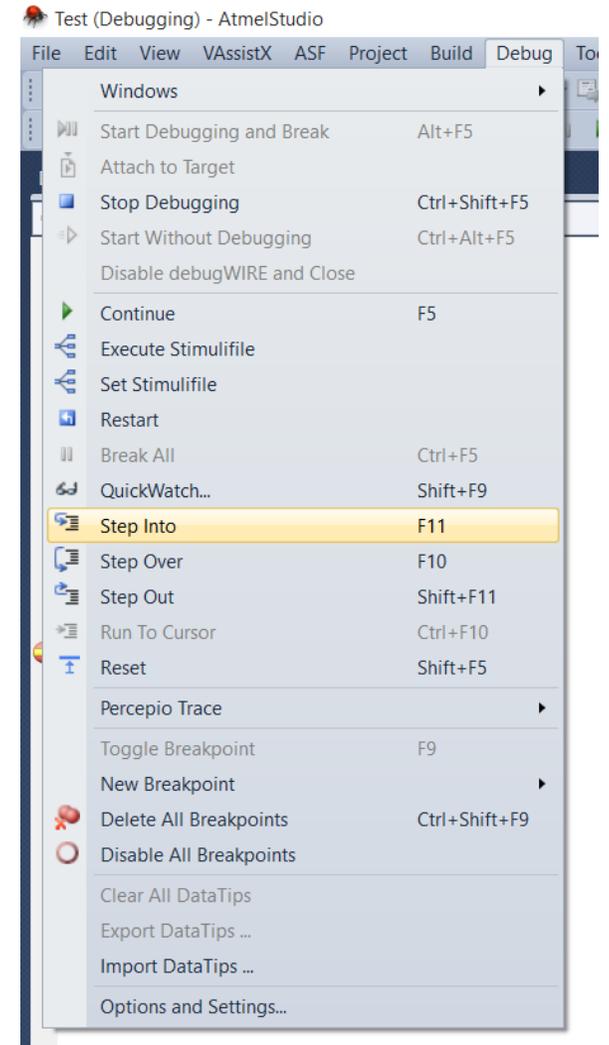
- Select particular peripheral and then the register to observe the value. (shown on left)
- Type variable names from your code in Watch Window to monitor their values. (shown on right)
  - Notice that I have ran through the loop once  $\rightarrow$  counter = 1



# Other Commands in Debugging Session

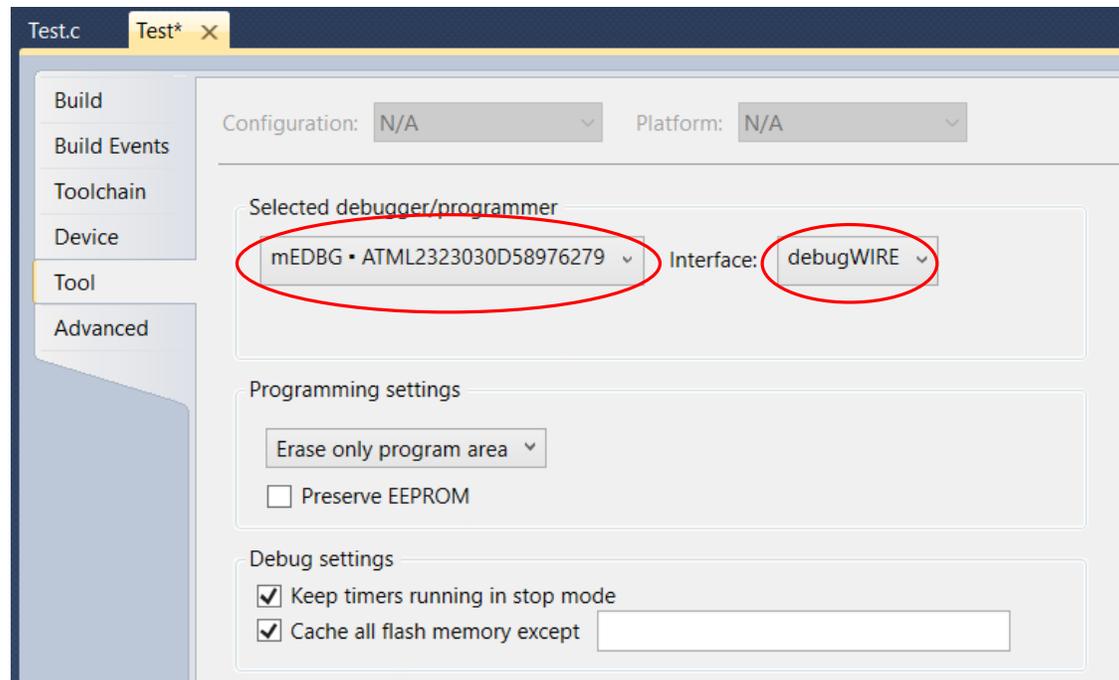
Inside 'Debug' tab, you'll see various useful debugging commands.

- 'Stop Debugging' exists the debugging session.
- 'Continue' run the code until the next breakpoint.
- 'Restart' restarts the debugging session and runs the code.
- 'Step Into' steps through the code line by line.
- 'Step Over' jumps over a function and stops after executing it.
- 'Step Out' returns from the current function and stops.
- 'Run to cursor' runs down to where the cursor is.
- 'Reset' command resets the current debugging session.



# Debugging in Atmel Studio (debugWire Mode)

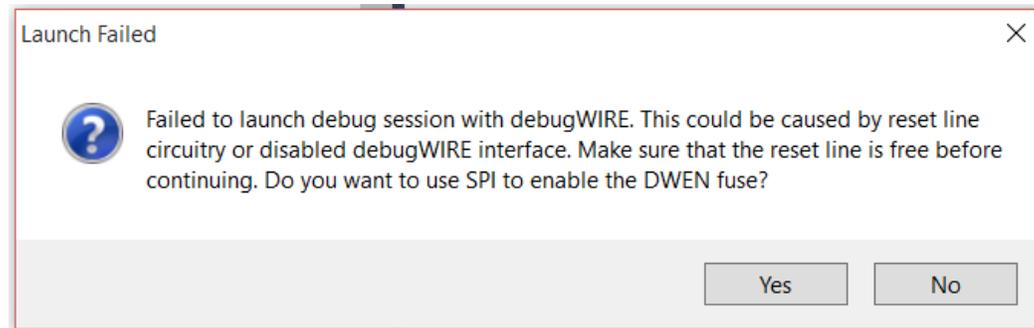
- On-chip debugging for Xplained Mini kits using debugWire interface is also quite similar to the simulator mode.
  - Simulator mode simulates the code as if it is running on the actual microcontroller
  - debugWire allows you to actually run the code on the microcontroller while you debug it step by step.
- Connect the Xplained Mini board with your computer
- Go to the Tool tab and select mEDBG with debugWire interface.



# Starting a Debugging Session (debugWire Mode)

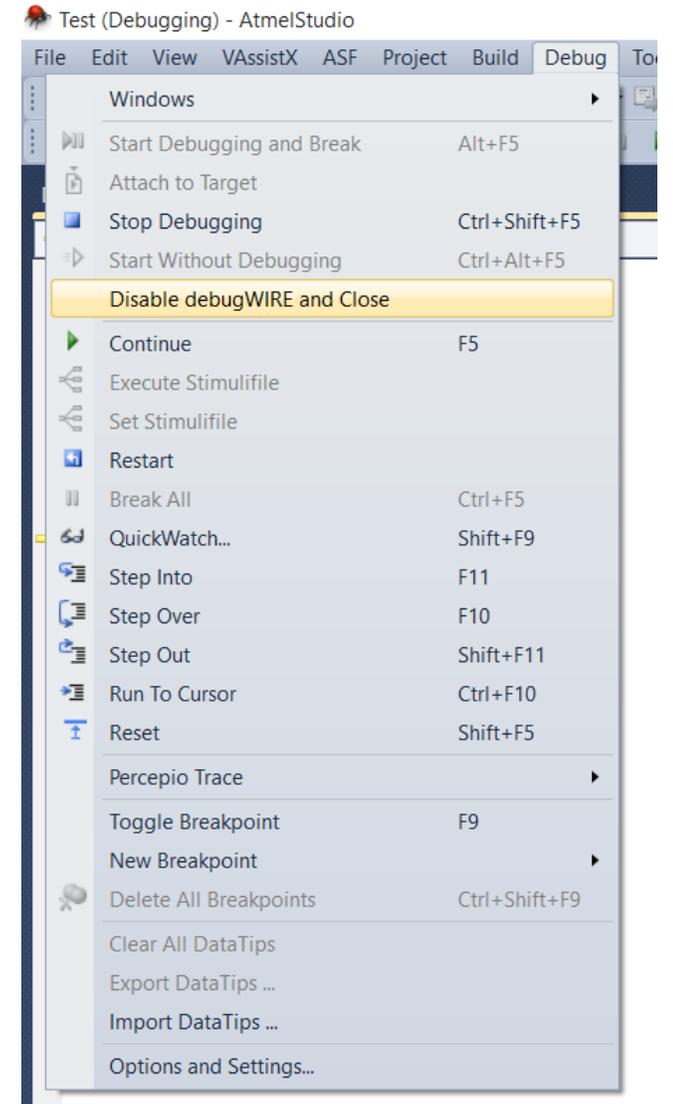
---

- Build the project (hit F7) and from Debug tab, select “Start Debugging and Break”
- Most likely you’ll see an error message asking you to enable DWEN fuse (as shown below).
  - DWEN fuse (debugWire Enable fuse) enables the debugWire interface on your microcontroller.
  - Click ‘Yes’ on the error message window and enable DWEN fuse.
- The debugger will pause at the start of main, just like simulator mode.
- Now you may use similar debugging techniques as done in Simulator mode
  - Use breakpoints to stop at a particular instruction.
  - Use Watch windows to observe/set program variables.
  - Use I/O view to observe/set the peripheral registers.



# Exiting a Debugging Session (debugWire Mode)

- It is really important to exit the debugWire debugging session in a proper way!
- To exit the debugging session, click on “Disable debugWire and Close”.
  - This will first disable the DWEN fuse in the microcontroller.
  - Then it will close the debugging session.
- If DWEN fuse is not disabled, you’ll not be able to program the microcontroller in ISP mode (which we want to use most frequently).



# Debugging using Assert library

---

- <http://people.ece.cornell.edu/land/courses/ece4760/Debugging/index.htm> has many great suggestions
- One can use the assert library (<http://en.wikipedia.org/wiki/Assert.h>) to test assertions in code
- Example:

```
//set up the debugging utility ASSERT
#define __ASSERT_USE_STDERR
#include <assert.h>

//test assertion - will print message if argument is NOT true;
assert(time<10);
```

# Debugging using Hardware Peripherals

---

- Debugging can also be performed by hardware peripherals.
  - By setting GPIO pins, for example, one can test the frequency of ISRs or certain program conditions (i.e. `PORTD |= 0x01`; when something happens) and measure results with an oscilloscope.
- Once the LCD lab has been done, one can also display variables and conditions on the screen as code is executed, if there are problems.