TRAINING MANUAL

From Maker to Manufacture: Bridging the Gap from Arduino to AVR

AN-12077

Prerequisites

- **Hardware Prerequisites**
  - ATmega328P Xplained Mini Board
  - IO1 Xplained Pro extension board
  - Arduino Xplained Pro board
  - Micro-USB cable
- **Software Prerequisites**
  - Atmel® Studio version 6.2 or later
  - Arduino IDE 1.6.0
  - Arduino Extension for Atmel Studio
  - Terminal Window Extension
- **Estimated Completion Time**
  - Two hours

Introduction

This hands-on will demonstrate how to develop Arduino using Atmel Studio along with the rich user interface and other great development tools that it provides.

Arduino is an open-source electronics prototyping platform based on flexible, easy- to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments. Key fact about Arduino is that these boards are based on the Atmel microcontroller family and underlying software is based on Atmel development tools.

Answer for 'Why should I switch from Arduino?' is:

The Arduino IDE:

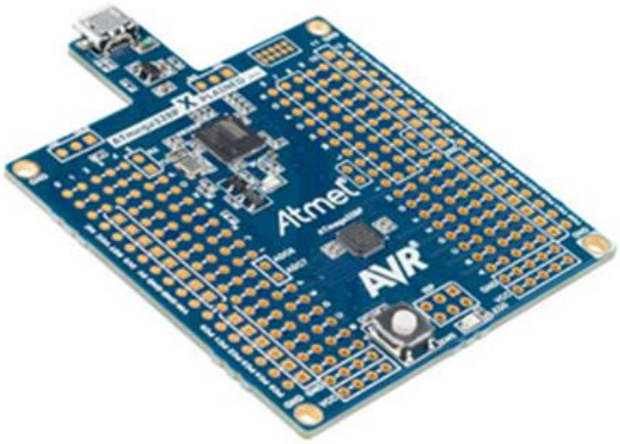
- quite limiting for experienced programmers capabilities
- lack of compiler warnings and debugging capabilities
- (Serial.println() after every statement doesn't count), make life hard when working on advanced projects
- Atmel Studio is a great choice for users that have outgrown the integrated Arduino IDE
- some of those Arduino libraries are just so darn *convenient*

So why not have the best of both worlds? Arduino: a wrapper on top of C/C++ with debugging capabilities. It is possible to combine any Arduino sketch or library with your own custom code on advanced projects.

Atmel has a unique, privileged position in Arduino and responsibility to the “going pro” community to provide a bridge, help transition from hobbyist to developer.

The Atmel Xplained Mini family is a perfect “bridge” for easy transition to C and C++. It has very similar architecture to Arduino and offers most of the features of an Arduino board. It is possible to run Arduino sketches when the IDE is set up properly. On-board hardware debugger/programmer is also available. It uses the incredibly popular AVR® microcontroller family and it is relatively inexpensive.

Figure 1. ATmega328P Xplained Mini Board



The training material is composed of four assignments.

In the **first assignment** we will see how to connect ATmega328P Xplained Mini to Arduino IDE.

In the **second assignment** we will create Arduino sketches in Atmel Studio using Studio’s Arduino Extension.

In the **third assignment** we will configure Atmel Studio to directly take in sketches. We will discuss how to transit to Atmel Studio with existing Arduino sketches. Also, we will check how to insert break points and how to debug in the Atmel studio.

In the **fourth assignment** we will edit the file and create a simple application with peripherals: ADC, I^2C, and SPI. The application is to read a light sensor through ADC, read a temperature sensor through an I^2C interface, and store data in the SD card through a SPI interface.

Table of Contents

Icon Key Identifiers

 **INFO**	**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.**
 **EXECUTE**	**Highlights actions to be executed of the target when necessary.**

1 Training Module Architecture

This training material can be retrieved through different Atmel deliveries:

- As an Atmel Studio Extension (**.vsix** file) usually found on the Atmel Gallery web site (http://gallery.atmel.com/) or using the Atmel Studio Extension manager
- As an Atmel Training Executable (**.exe** file) usually provided during Atmel Training sessions

Depending on the delivery type, the different resources needed by this training material (hands-on documentation, datasheets, application notes, software, and tools) can be found on different locations.

1.1 Atmel Studio Extension (.vsix)

Once the extension is installed, you can open and create the different projects using *"New Example Project from ASF..."* in Atmel Studio.

The projects installed from an extension are normally found under "*Atmel Training > Atmel Corp. Extension Name*".

There are different projects which can be available depending on the extension:

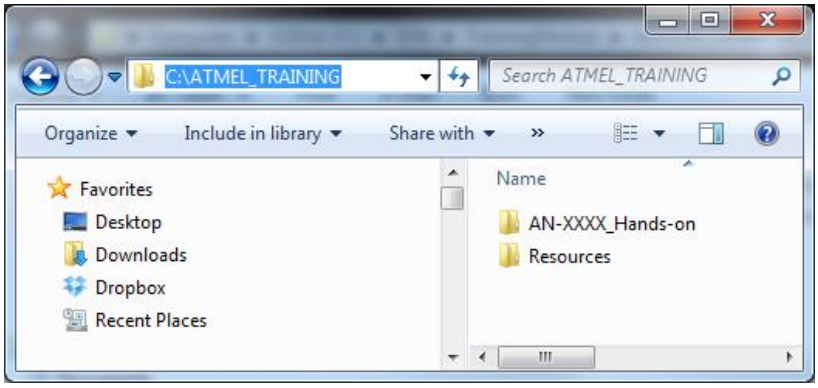
- **`Hands-on Documentation`**: contains the documentation as required resources
- **`Hands-on Assignment`**: contains the initial project that may be required to start
- **`Hands-on Solution`**: contains the final application which is a solution for this hands-on

Each time a reference is made to some resources in the following pages, the user must refer to the `Hands-on Documentation` project folder.

1.2 Atmel Training Executable (.exe)

Depending on where the executable has been installed, you will find the following architecture which is composed by two main folders:

- `AN-12077_Hands-on`: contains the initial project that may be required to start and a solution
- **`Resources`**: contains required resources (datasheets, software, tools…)



Unless a specific location is specified, each time a reference is made to some resources in the following pages, the user must refer to this `Resources` folder.

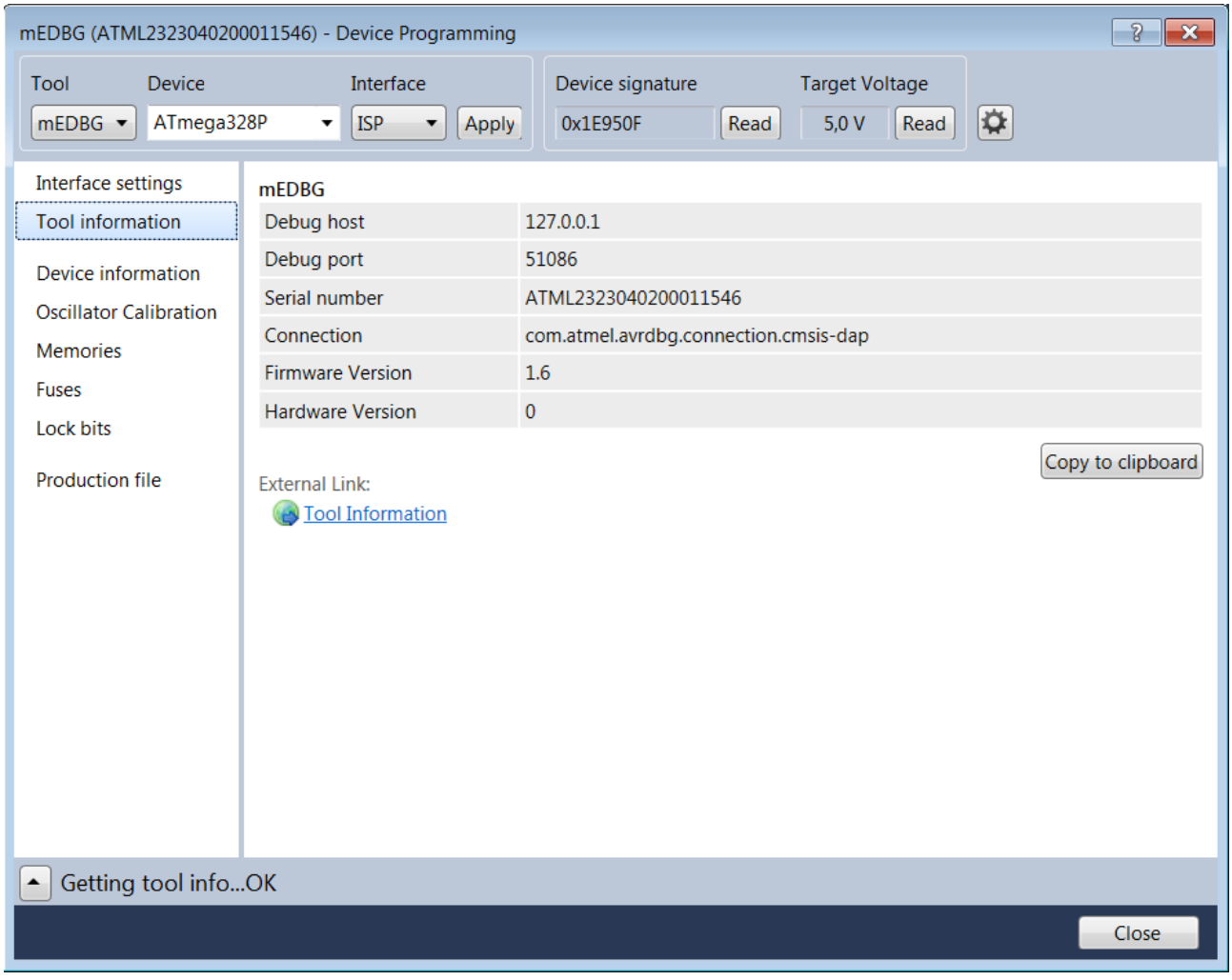
2 Assignment 1: How to connect the ATmega328P Xplained Mini to the Arduino IDE

2.1 Arduino IDE

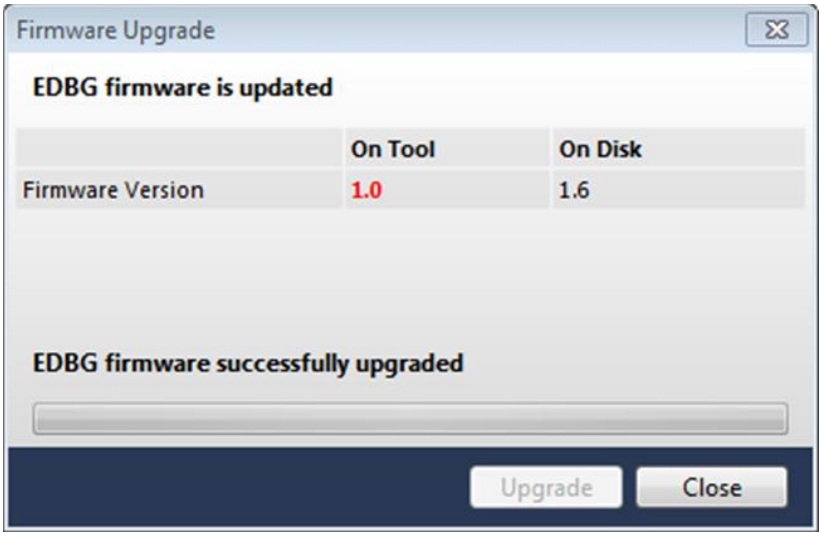
Download the Arduino IDE from www.arduino.cc.

2.2 mEDBG Firmware upgrade on the ATmega328P Xplained Mini

1. Go to Atmel spaces http://spaces.atmel.com/gf/project/avr_xp_mini/frs/ .From the package “medbgdebugger” select medbg_fw.zip and download.
2. Overwrite the zip package medbg_fw.zip in the Atmel Studio installation folder (e.g.: C:\Program Files (x86)\Atmel\Atmel Studio 6.2\tools\mEDBG).
3. Start Atmel Studio.
4. Connect the ATmega328P Xplained Mini to the computer.
5. In Atmel Studio, select Tools → Device programming (alt.: Ctrl+Shift+P).
6. In the Device Programming window, set Tool to mEDBG and click “Apply”.



INFO **Atmel Studio will now ask you if you want to upgrade the firmware.**



7. Select Upgrade.

INFO **There is a bug in the serial number on some of the ATmega328P Xplained Mini boards making them not recognizable by Atmel Studio programming/debugging. If your board has an unknown character in the serial number there is a fix in the "Releases" folder. To see your board serial number, start Atmel Studio and go to Tools → Device programming → select tool, this will list the mEDBG with serial number, if some of the characters have a "?" on black background, download the package from serial number fix and follow the instruction from How_to_change_serial_Number.pdf.**

2.3 Set the Bootloader Fuses in ATmega328P

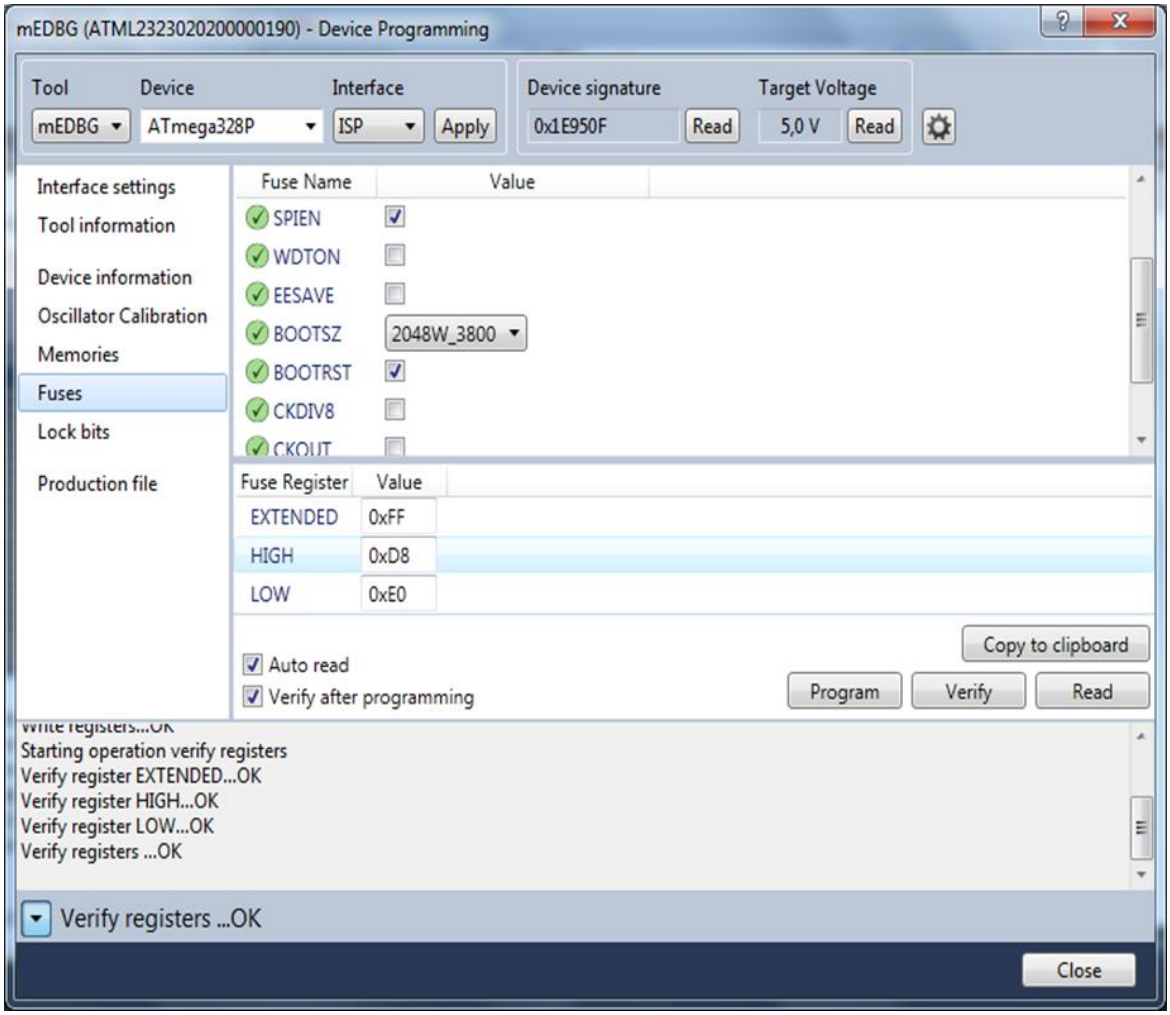
1. Now In the 'Device Programming' window, select 'Fuses'.
2. Change value on EXTENDED, HIGH, and LOW as below and click Program.

   EXTENDED = 0xFF

   HIGH = 0xD8

   LOW= 0xE0

Figure 2-1. Device Programming: Fuses



2.4 Program the Bootloader

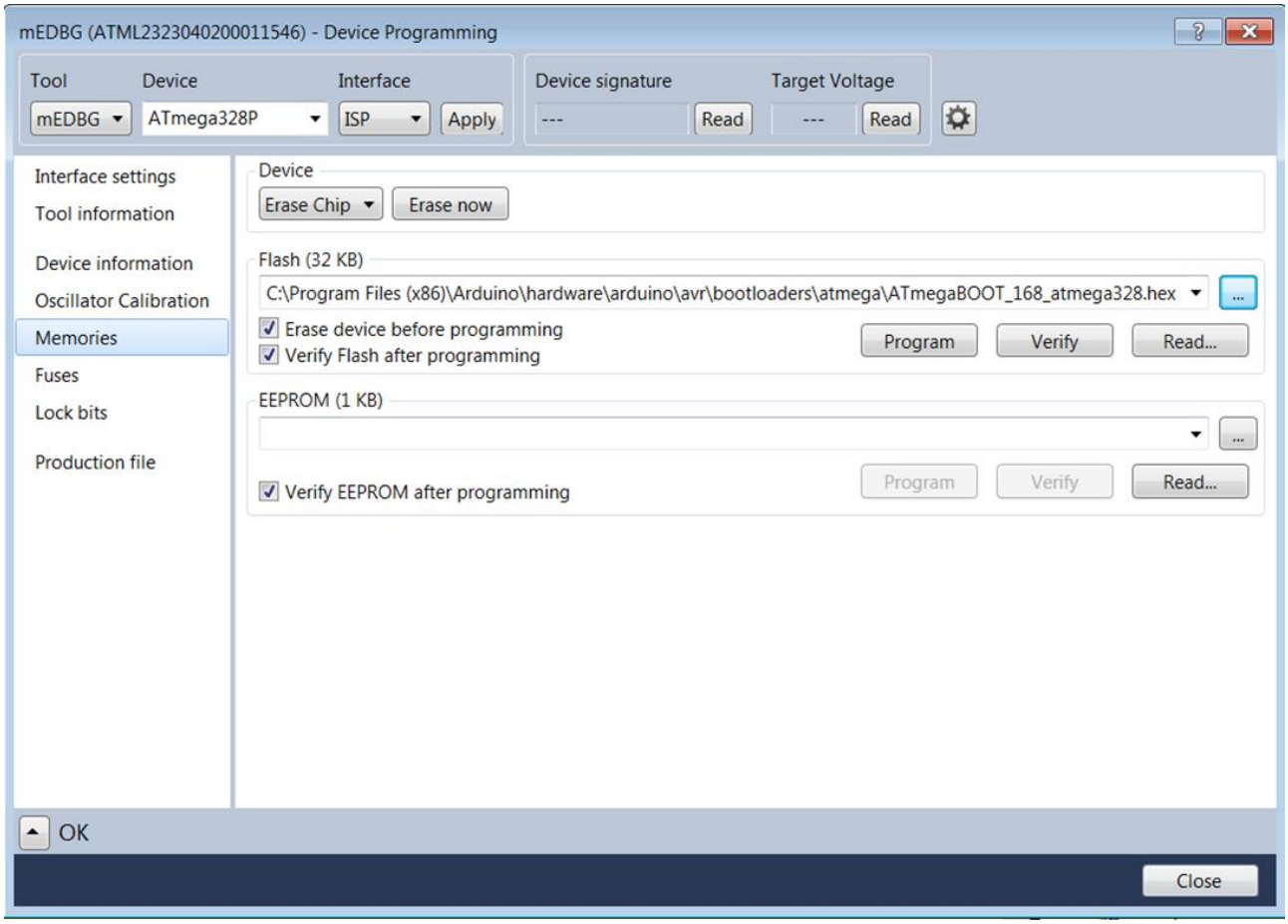
INFO **The bootloader hex file is located in the Arduino IDE folder: C:\Program Files (x86)\Arduino\hardware\arduino\avr\bootloaders\atmega*.hex. Bootloader can be selected according to board configurations as listed in Table 2-1.**

Table 2-1. Bootloader

Xplained Mini	Bootloader
ATmega328P/5V/16MHz	ATmegaBOOT_168_atmega328.hex
ATmega168PB/5V/16MHz	ATmegaBOOT_168_ng.hex
ATmega168PB/3.3V/8MHz	ATmegaBOOT_168_pro_8MHz.hex

1. In the ‘Tools → Device Programming’ window, select tab ”Memories”.
2. Browse for C:\Program Files (x86)\Arduino\hardware\arduino\avr\bootloaders\atmega\ ATmegaBOOT_168_atmega328.hex.
3. Click program.

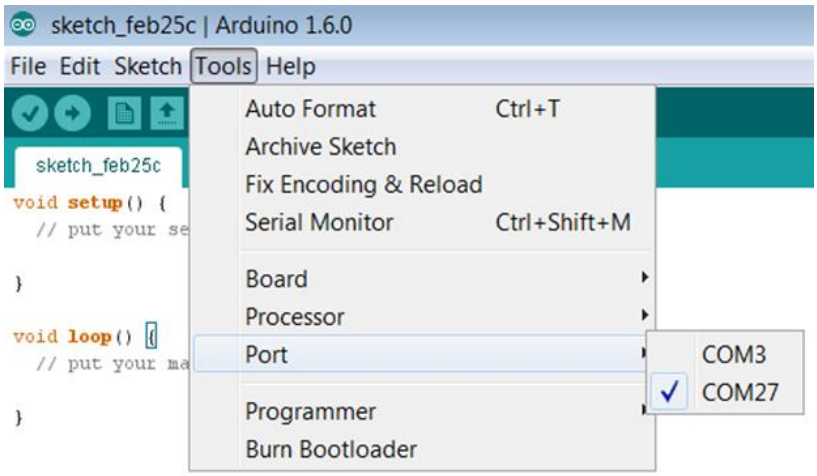
Figure 2-2. Device Programming: Memories



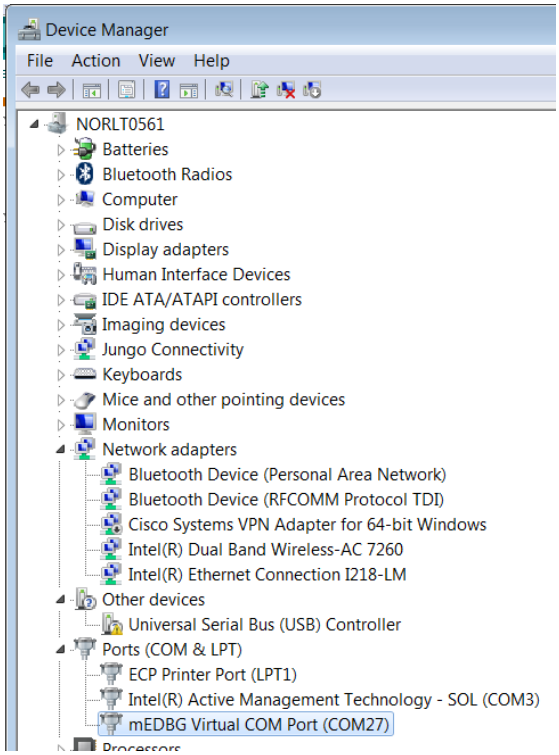
2.5 Configure Arduino IDE

1. Start the Arduino IDE from the Windows® start menu, or the folder the Arduino IDE was installed in.
2. From menu Tools → Port select the correct COM port for the mEDBG. (See tip below on how to verify the COM port used for the mEDBG.)

Figure 2-3. Arduino IDE: Serial Port



mEDBG com port can be viewed from Start → Control panel → Device Manager → Ports as below:



3. Board can be selected according to Table 2-2.

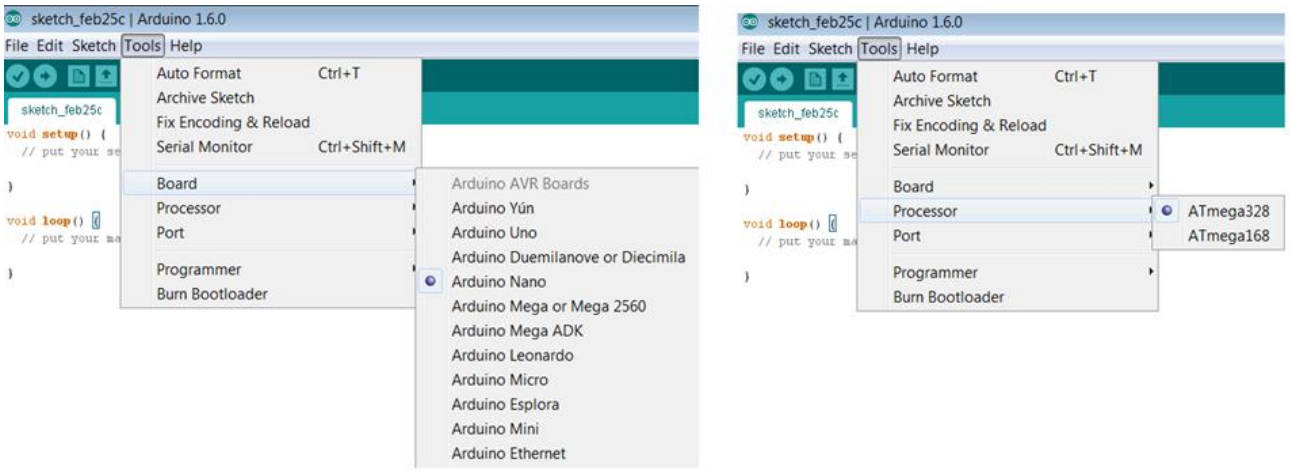
Table 2-2. Arduino IDE: Board

Xplained Mini	Board to select in Arduino IDE
ATmega328P Xplained Mini	Arduino Nano: ATmega328
ATmega168PB Xplained Mini	Arduino Nano: ATmega168

Here, select: Tools → Boards → Arduino Nano.

Select: Tools → Processor → ATmega328.

Figure 2-4. Arduino IDE: Board and Processor

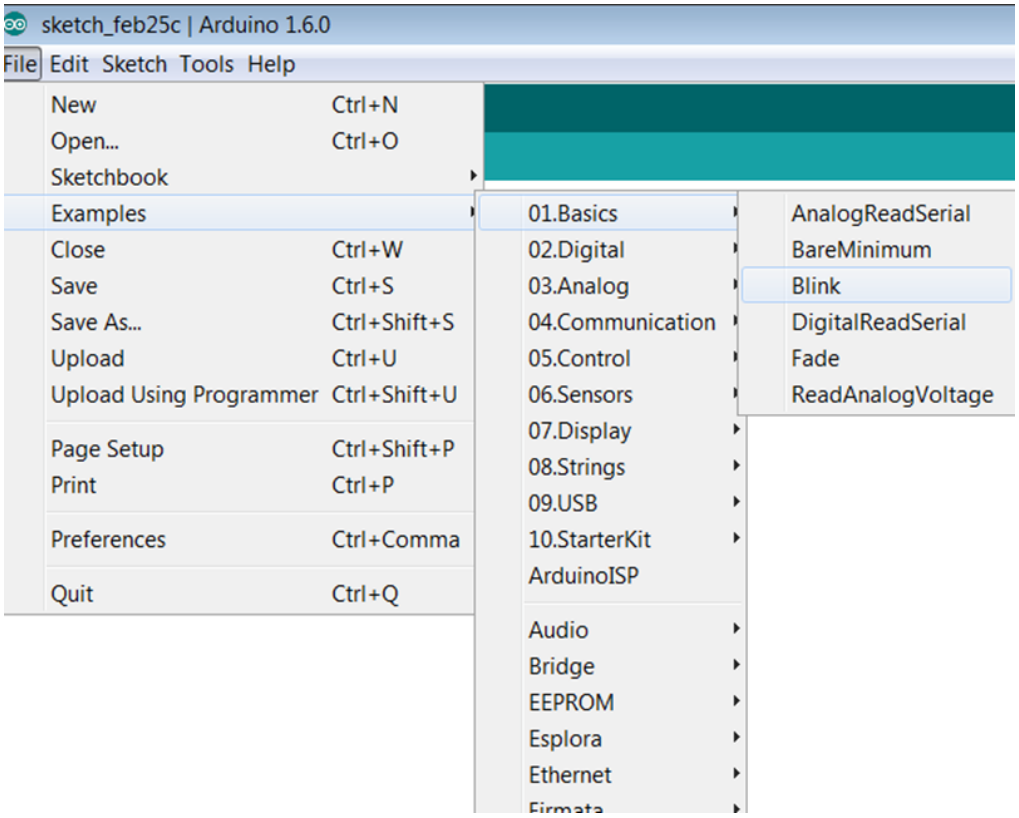


2.6 Upload the Program

RESULT **ATmega328P Xplained Mini board will be connected to Arduino IDE and will start blinking LED.**

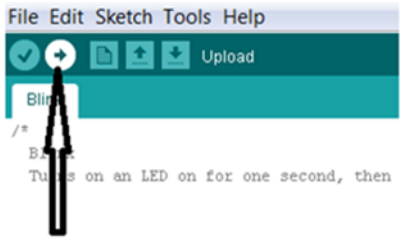
1. In Arduino IDE, select File → Examples → 01.Basics → Blink.

Figure 2-5. Arduino IDE: Examples-Blink



2. Upload the sketch by clicking “Upload” button in Arduino IDE.

Figure 2-6. Arduino IDE: Upload



RESULT **Messages will be appeared at the bottom of window ‘Compiling Sketch’, ‘Uploading’, ’Done uploading’.**

3. Select File → Save As... and save the sketch ‘Blink’. You can select any path to save sketch.
4. Observe the blinking LED in ATmega328P Xplained Mini board.

We have successfully connected ATmega328P Xplained Mini board to the Arduino IDE.

3 Assignment 2: Creating Arduino Sketches in Atmel Studio

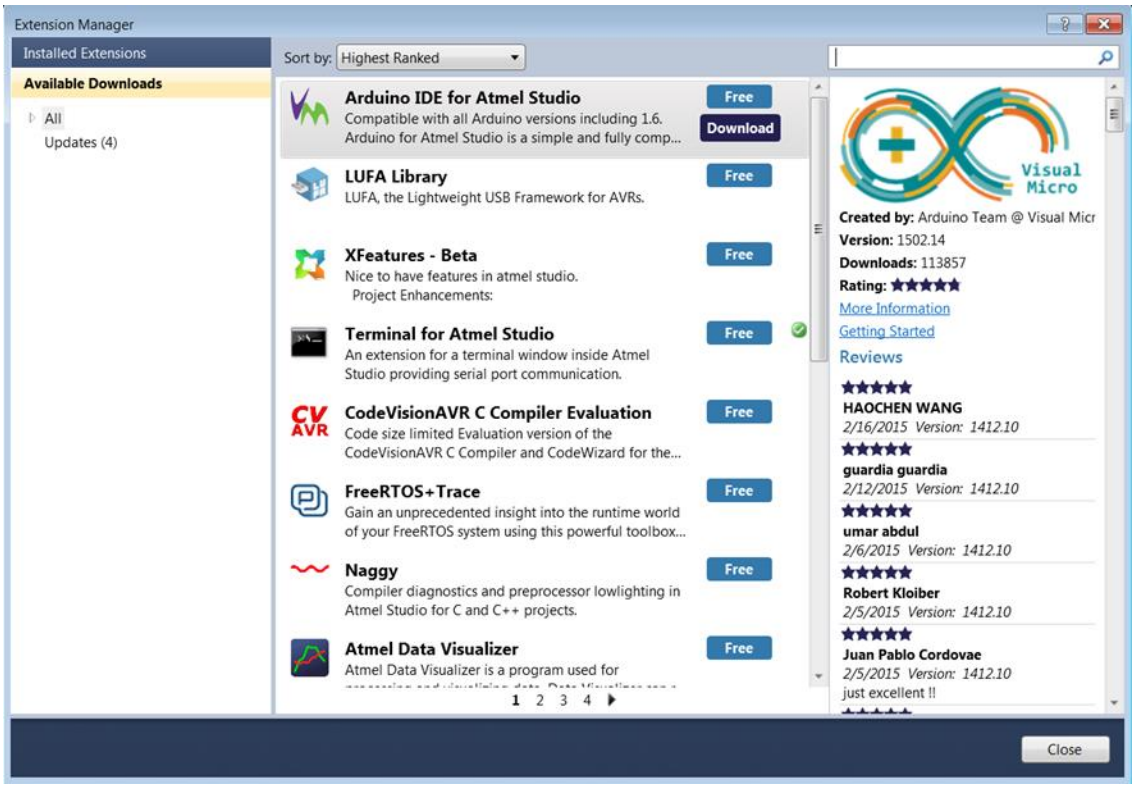
INFO

Atmel Studio's Arduino Extension allows any Arduino sketch to be written, compiled and uploaded to any Arduino while inside Atmel Studio with rich user interface and professional features that it provides.

3.1 Download Extension

1. Open Atmel Studio.
2. Select Tools → Extension Manager. The Extension Manager Window opens and by default it shows the installed extensions.
3. Click the "Available Downloads" option.

Figure 3-1. Atmel Studio: Extension Manager



4. Select "Arduino IDE for Atmel Studio" and click the "Download" icon.
5. The "Sign in to Extension Manager Dashboard" window opens and asks you to sign-in/register. Then Sign-in using the email-id and password provided while registering.

INFO

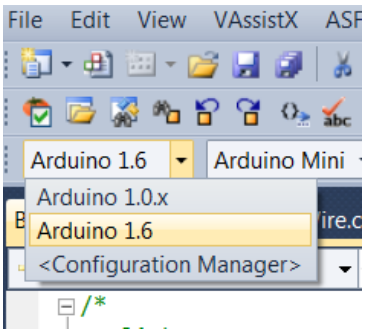
If you have not registered already, kindly register. Then close the "Sign in to Extension Manager Dashboard" window. Atmel will send you a confirmation email to the email-id provided. Click the link to confirm your email-id. Repeat steps 2 to 4 and in the "Sign in to Extension Manager Dashboard" and Sign-in using the email-id and password provided while registering.

6. The "Arduino IDE for Atmel Studio" extension will be downloaded, then install it.

3.2 Create Sketch

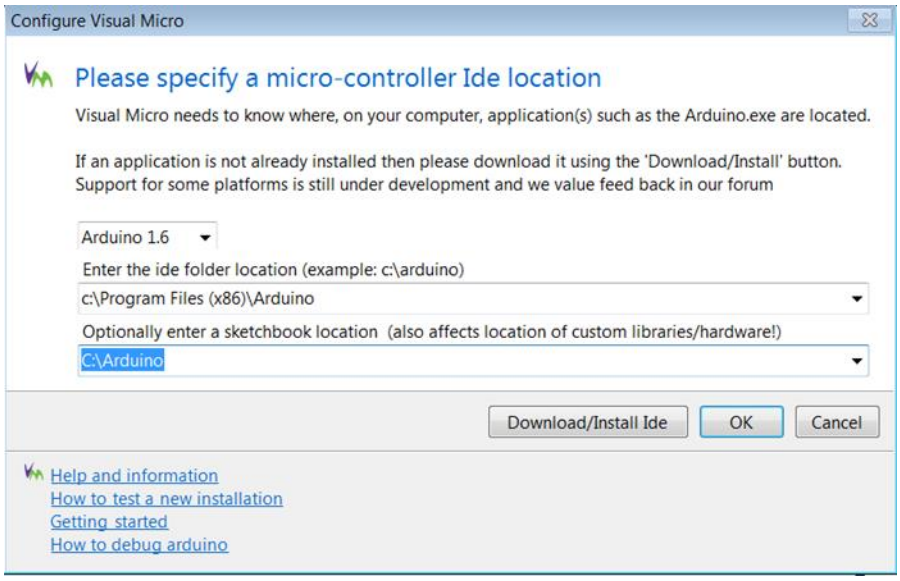
1. Open Atmel Studio.

TO DO **Verify Arduino 1.6 is listed in Atmel Studio and select it.**

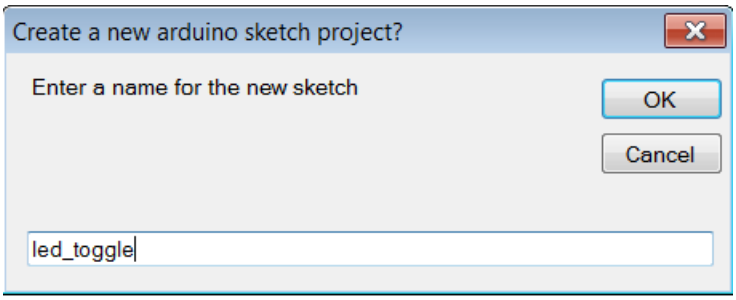


INFO **If it is not listed, click <Configuration Manager>. Select Arduino 1.6. Add the Arduino installation directory path as shown in below figure. Select 'OK'.**

Figure 3-2. Configure Visual Micro



2. Select File → New → Sketch Project.
3. Enter name for sketch project as shown in below figure and select 'OK'.

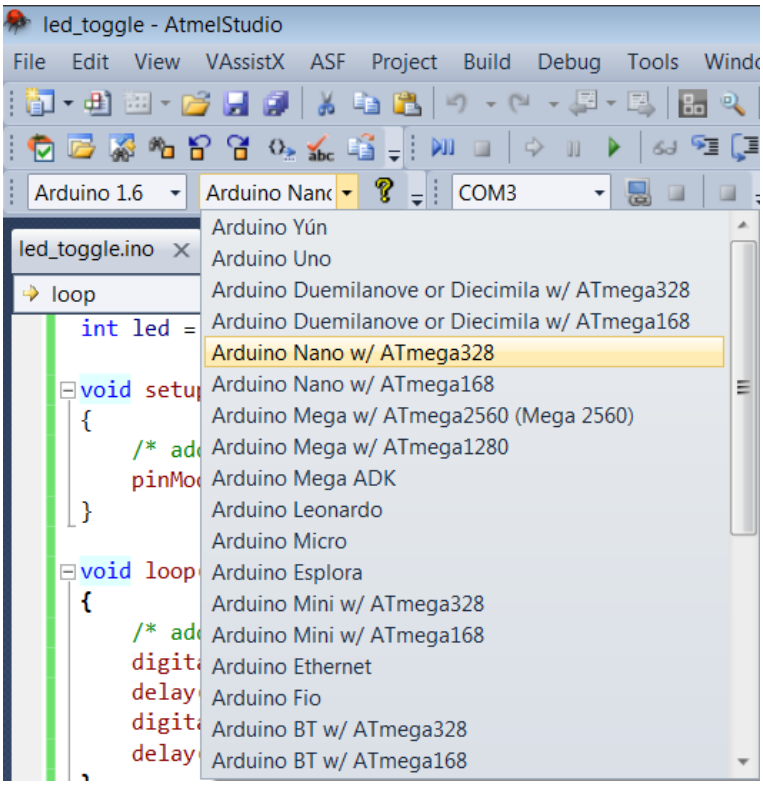


INFO **led_toggle.ino sketch will be created at location c:\Arduino as it is a default location as shown in Figure 3-2.**

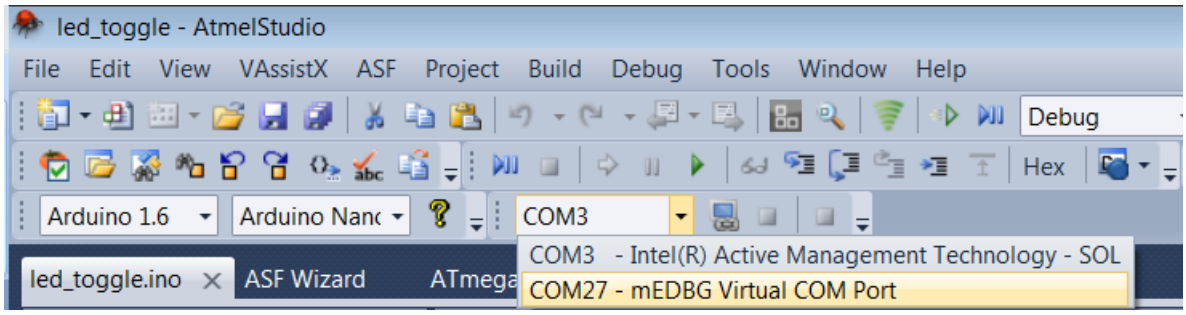
4. Edit the code to toggle the LED.

```
int led = 13;

void setup()
{
  /* add setup code here */
  pinMode(led, OUTPUT);
}

void loop()
{
  /* add main program code here */
  digitalWrite(led,HIGH);
  delay(500);
  digitalWrite(led, LOW);
  delay(500);
}
```

5. Select the board as Arduino Nano w/ATmega328.



6. Select mEDBG COM port number.



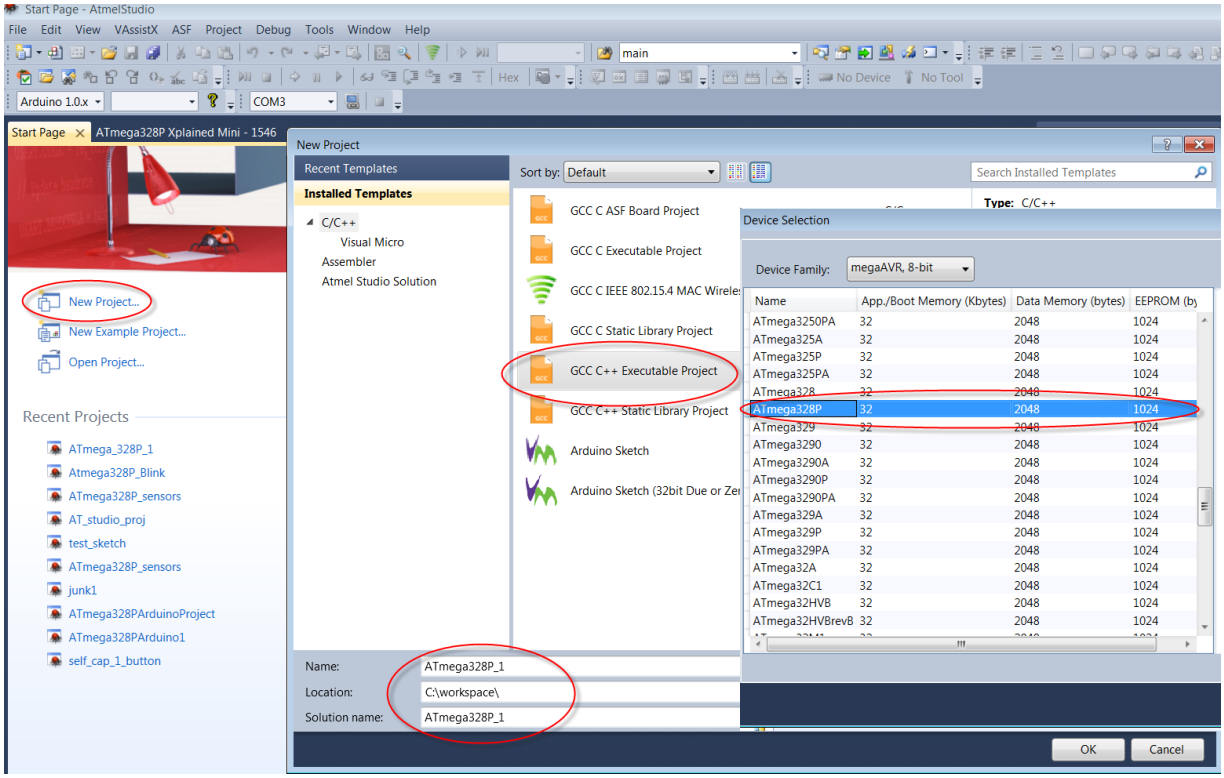
7. Download program by clicking icon  .

RESULT **LED on ATmega328P Xplained Mini board will start toggling.**

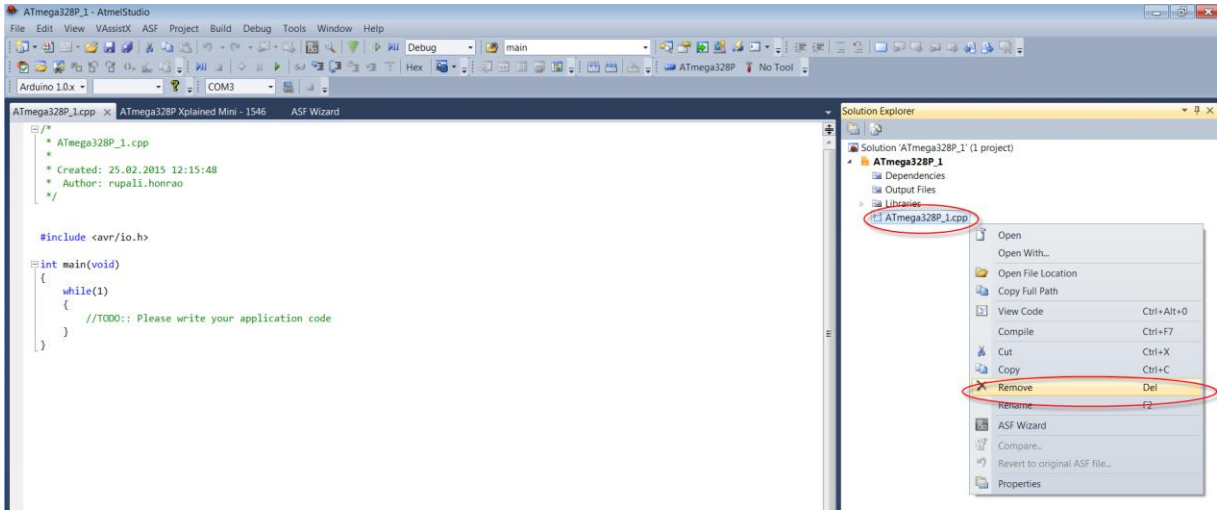
4 Transit to the Atmel Studio IDE with Existing Arduino Sketches

4.1 Project Creation

1. Open Atmel Studio and create a new “GCC C++ Executable Project give the Project”, give the project a reasonable name and select the ATmega328P as the device.
2. Make sure to make a note of the folder where the project is created.

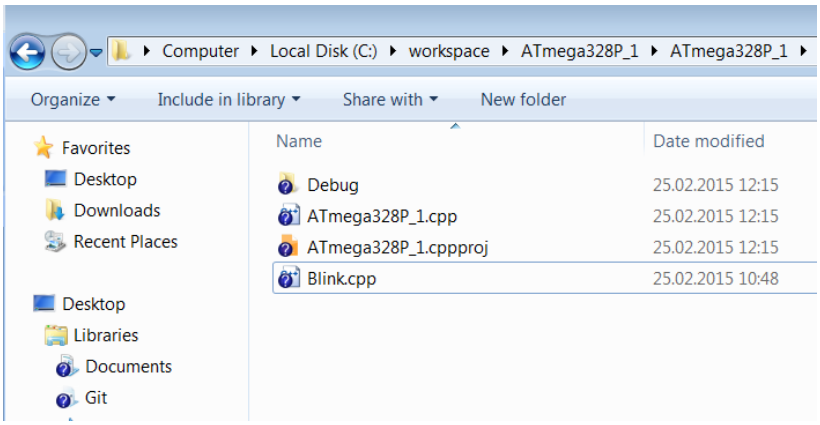


3. In the resulting solution, right click and remove the project cpp file (here *ATmega328P_1.cpp*) as shown in below figure.

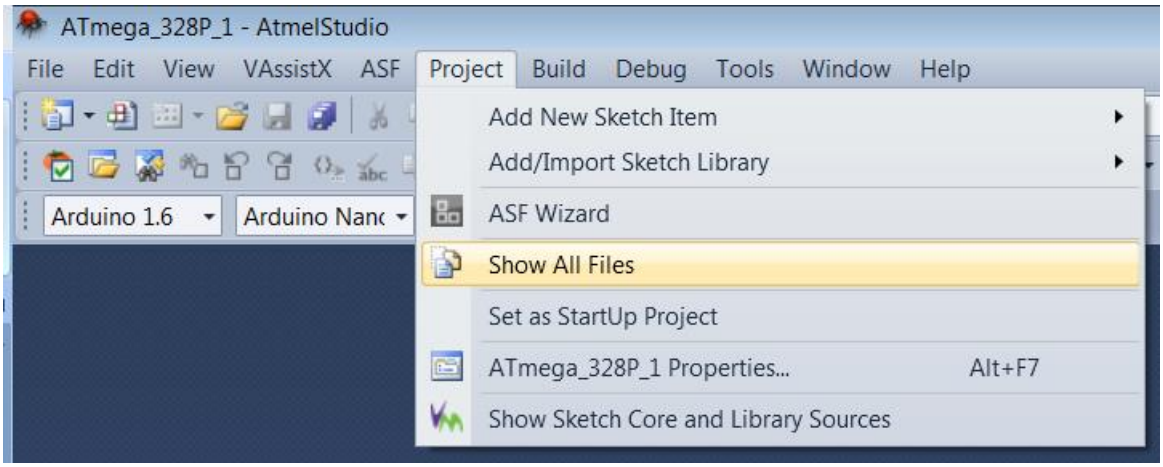


4. In Windows Explorer:
   a. Copy your sketch (earlier saved Arduino sketch Blink/Blink.ino) and place it to your new Atmel Studio project subdirectory, where *ATmega328P_1.cpp* is (still there in Explorer, even though it was removed from the Atmel Studio project).

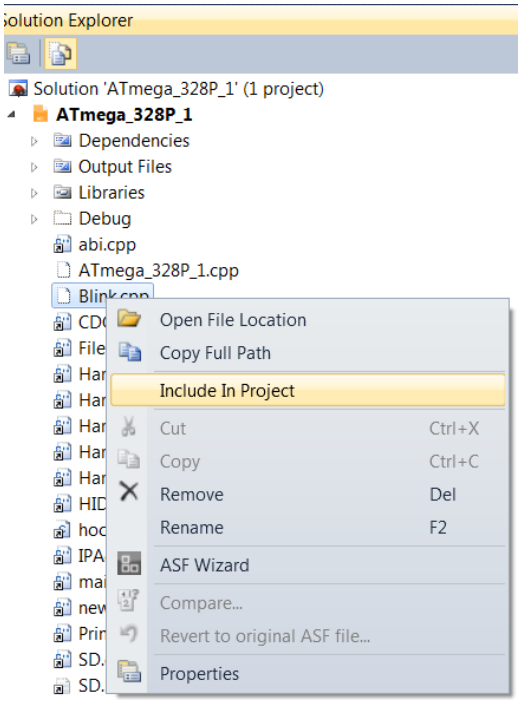
b. Rename your sketch extension to .cpp: Blink.ino → Blink.cpp. See below figure.



5. In the Atmel Studio select Project → Show All Files.



6. In the Solution Explorer window ,right click Blink.cpp file and select “Include In Project” as shown in figure below:



4.2 Configuring Compiler Symbols

1. In the Solution Explorer right click the project and select Properties or go to Project → [project name] Properties… in the file menu (Alt+F7).
2. Under Toolchain → AVR/GNU C Compiler → Symbols , add:

   F_CPU=16000000L

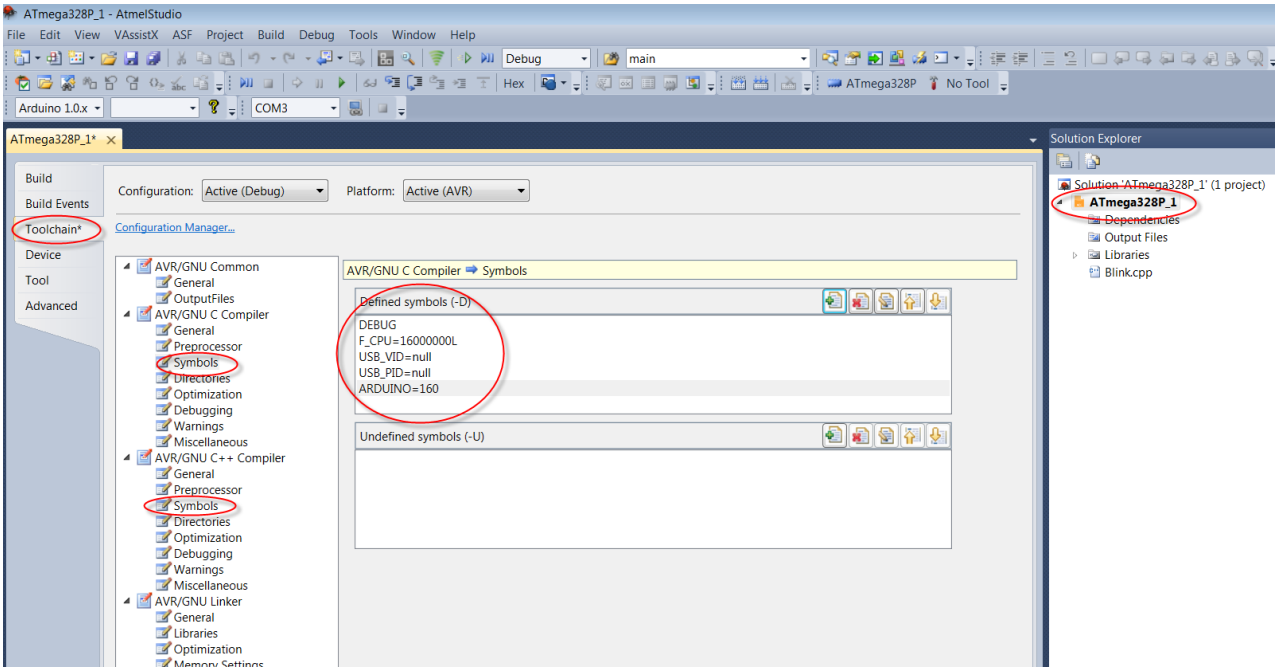
   USB_VID=null

   USB_PID=null

   ARDUINO=160

INFO **These are Arduino Specific Symbols.**

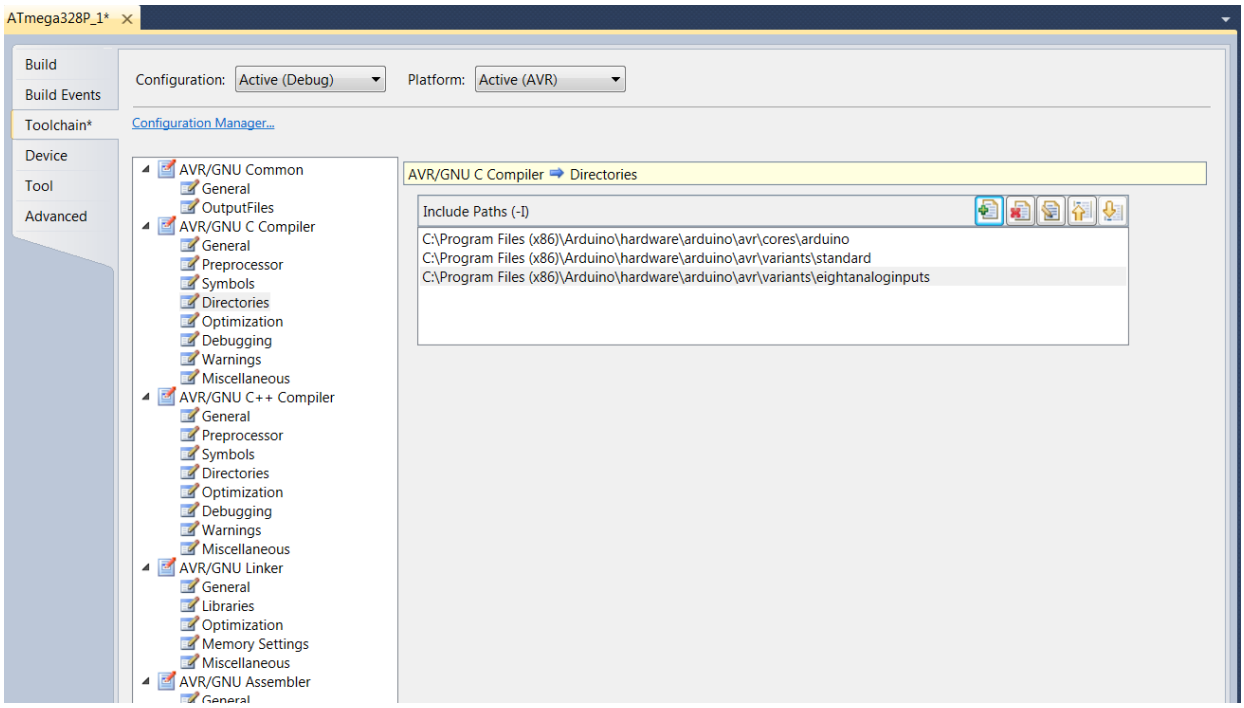
3. Repeat for Toolchain → AVR/GNU C++ Compiler → Symbols.



4.3 Configuring Compiler Directories

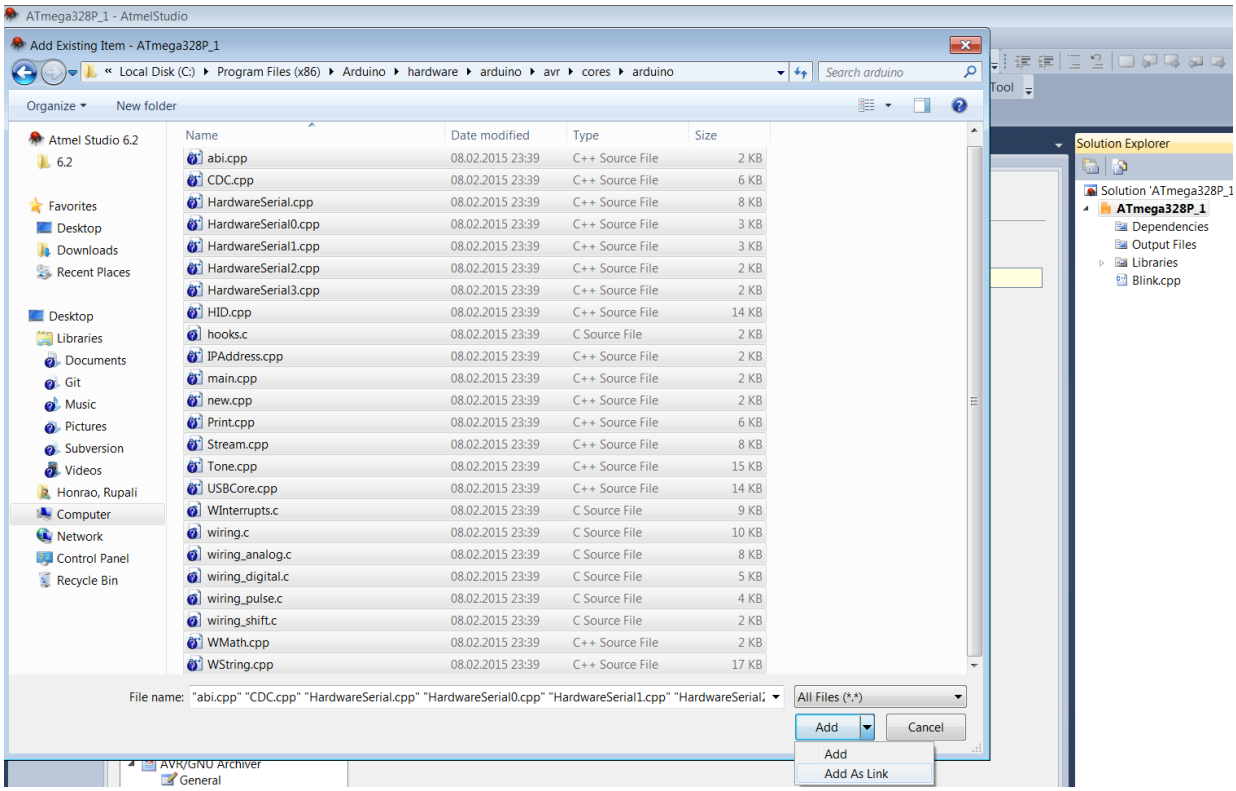
1. Under Toolchain → AVR/GNU C Compiler → Directories add.
2. C:\Program Files (x86)\Arduino\hardware\arduino\avr\cores\arduino.
3. C:\Program Files (x86)\Arduino\hardware\arduino\avr\variants\standard.
4. C:\Program Files (x86)\Arduino\hardware\arduino\avr\variants\eightanaloginputs.
5. Repeat for Toolchain → AVR/GNU C++ Compiler → Directories.

WARNING **Make sure you deselect “Relative Path” while adding these paths.**



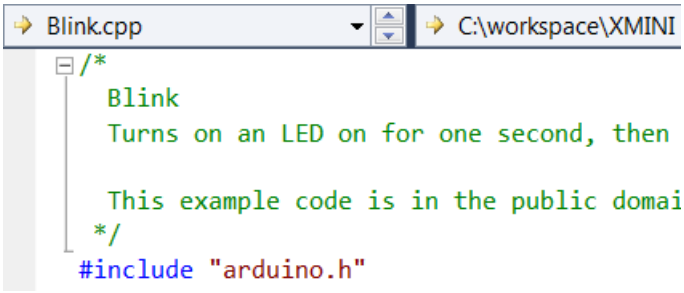
4.4 Add Arduino Dependency Files

1. Right click the project; go to “Add → Existing Item…”
2. In the dialog box that opens, aim the browser at C:\Program Files (x86)\Arduino\hardware\arduino\avr\cores\arduino
   a. In the “File Name” box, type *.c* <enter>
   b. Multi-select all files and select “Add as Link”



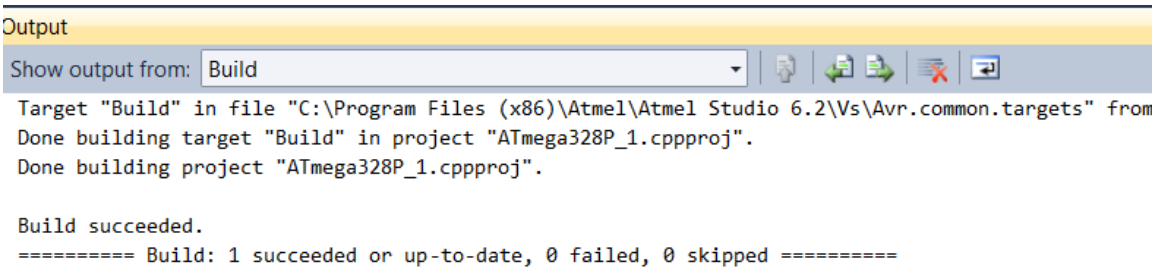
4.5 Build Solution

1. Add # include "arduino.h" at the top of Blink.cpp



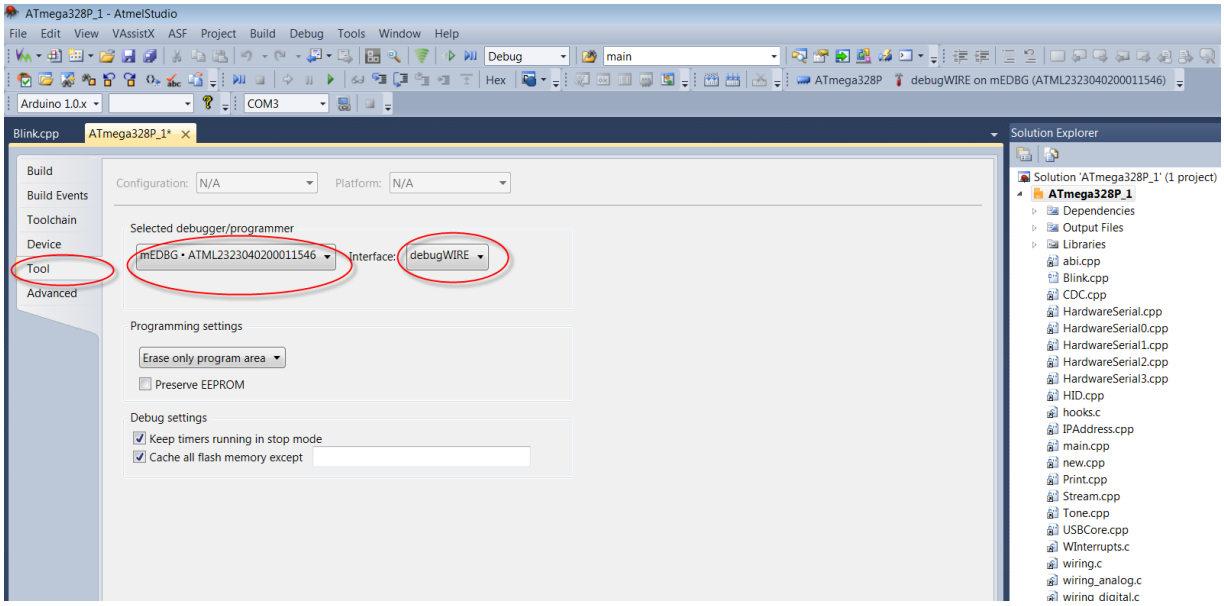
INFO **Now all the Arduino dependencies have been added and project can be compiled and loaded into the target.**

2. In the file menu, select Build → Build Solution. The build should finish successfully with no errors.



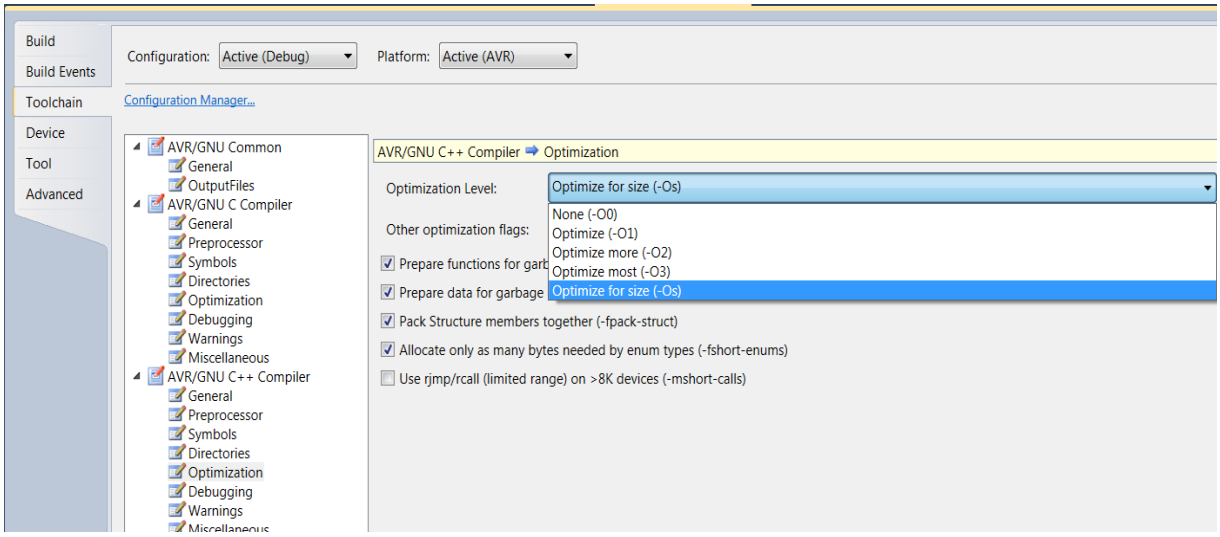
4.6 Plug in ATmega328P Xplained Mini Board

1. In the Solution Explorer right click the project and select Properties or go to Project→ [project name] Properties… in the file menu (Alt=F7).
2. Under Tools, select mEDBG and debugWIRE as interfaces.



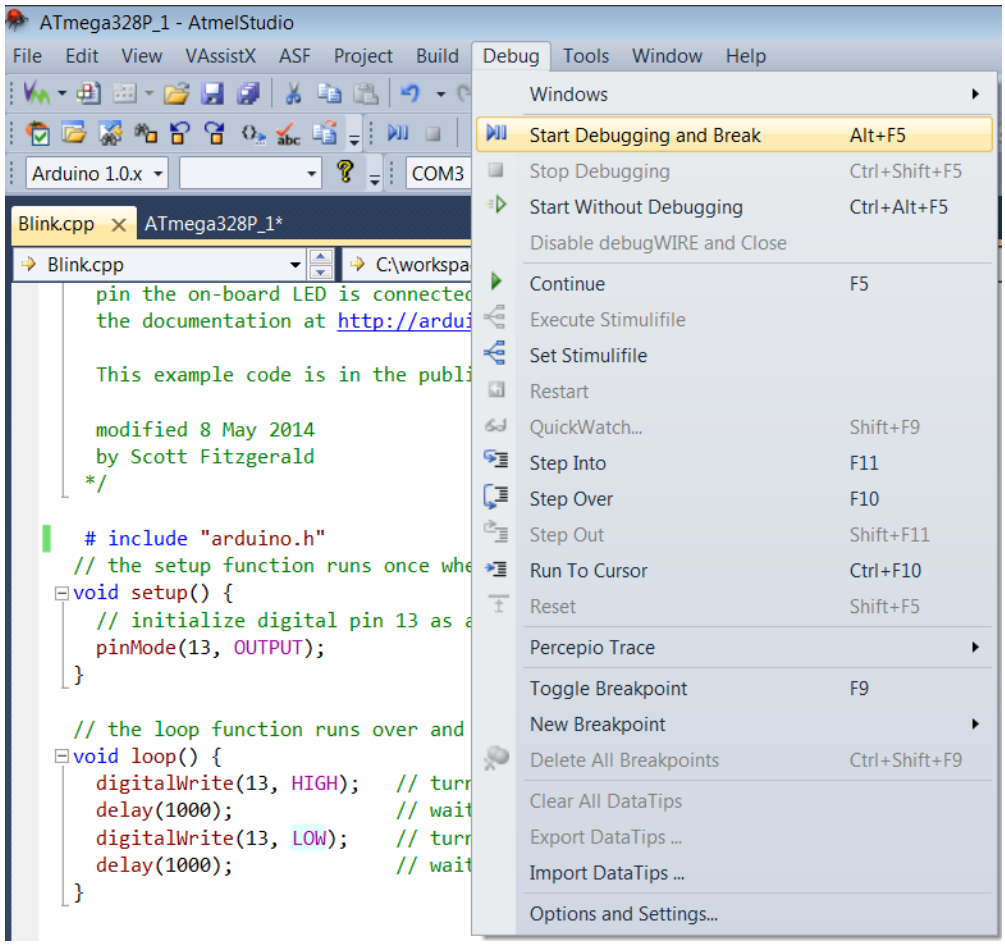
INFO **At some point the system may want to update the debugger. Let it update.**

3. Under Toolchain → AVR/GNU C Compiler → Optimization. Select Optimization level ‘Optimize for size(-Os)’.
4. Under Toolchain → AVR/GNU C++ Compiler → Optimization. Select Optimization level ‘Optimize for size(-Os)’.

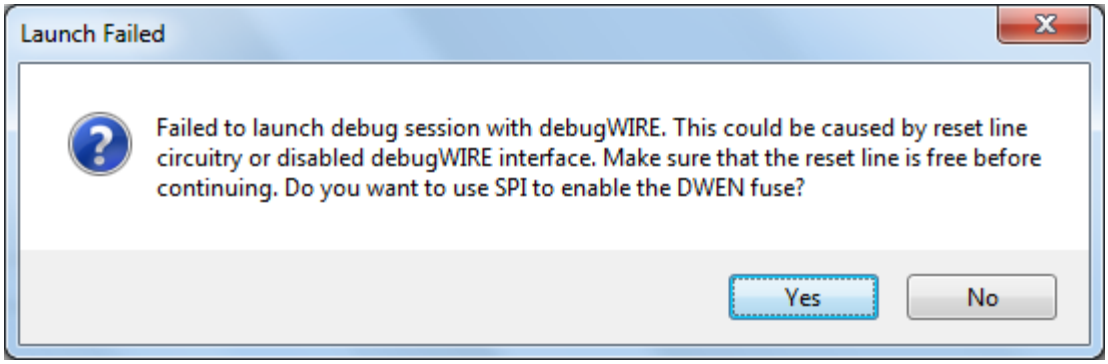


4.7 Debugging

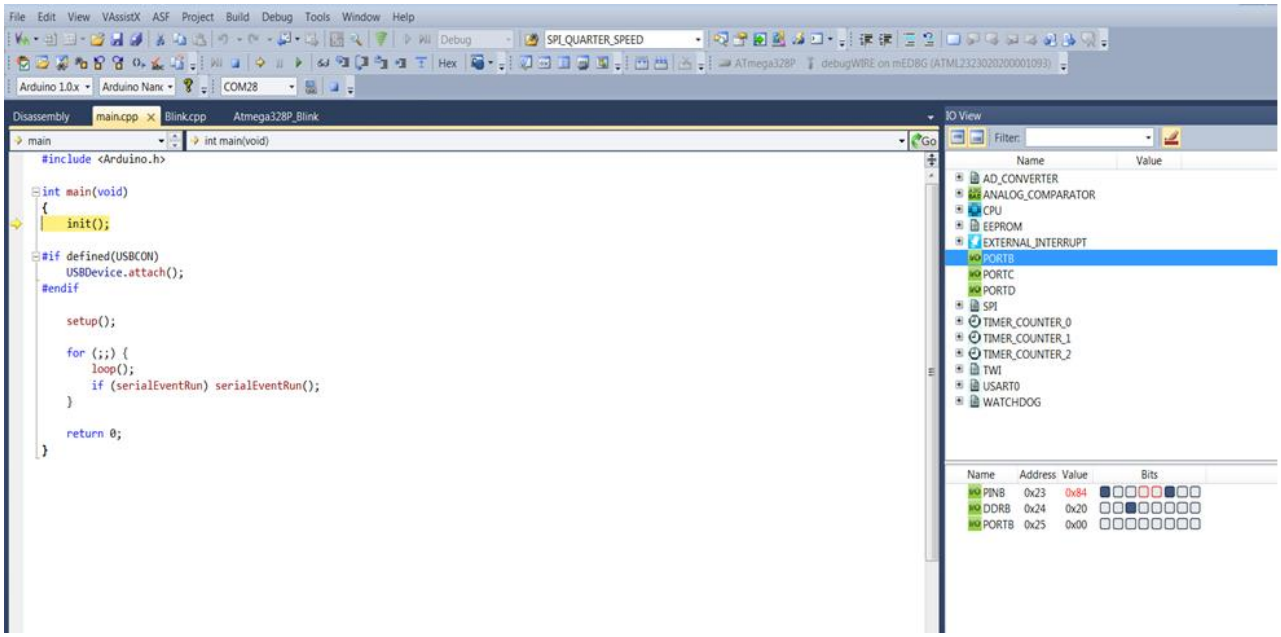
1. Select debug and click ‘Start Debugging and Break’.



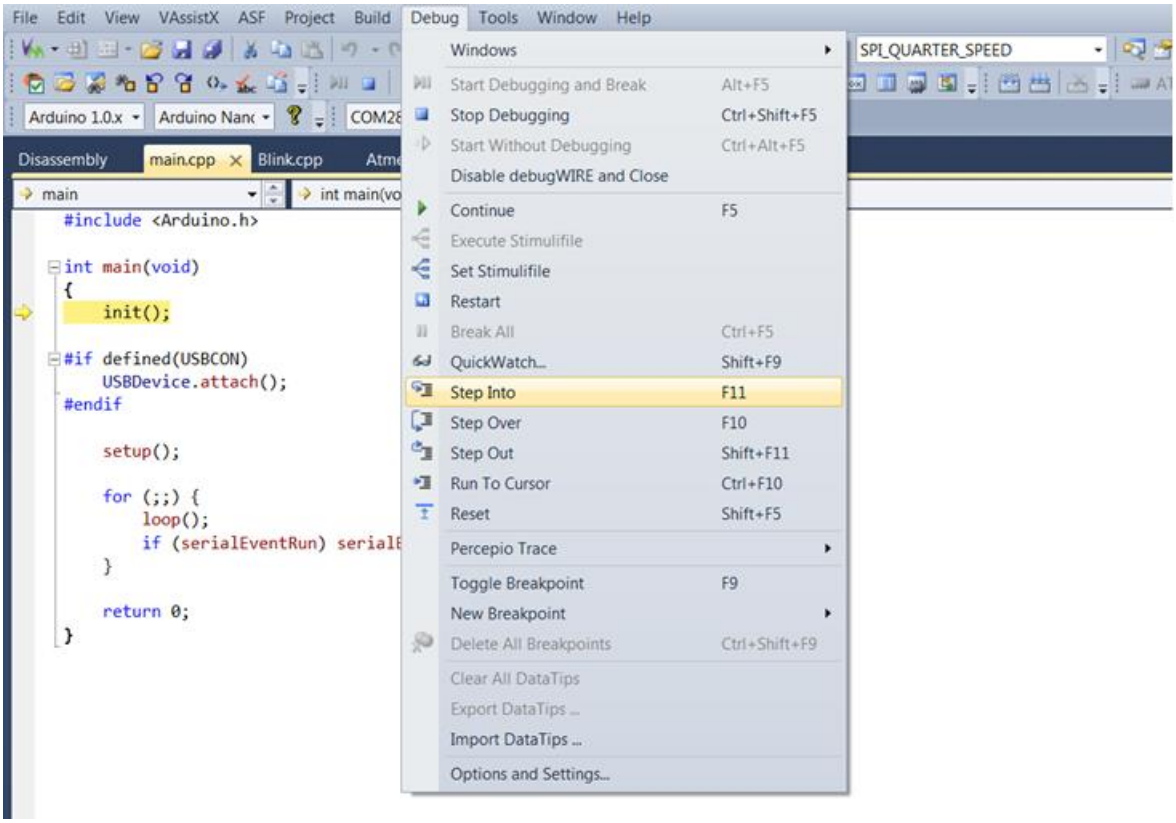
WARNING **If the DWEN fuse is not enabled and error message is displayed. Click 'Yes' and Studio will use the ISP to set the fuse as shown below.**



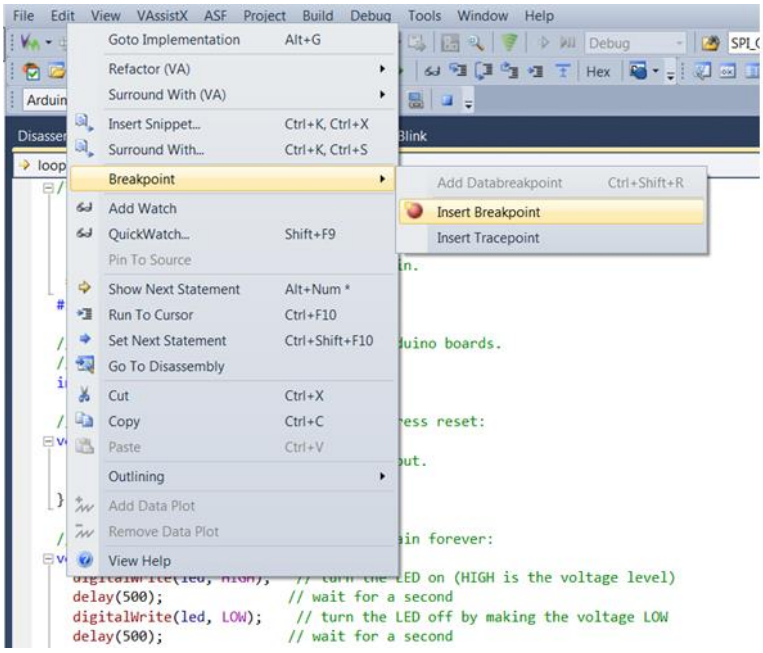
RESULT **The Debugger is started and breaks in main. You are now ready to start debugging.**



INFO **Different debug options are available in the debug menu.**



2. (In the 'Solution Explorer' file Blink.cpp) Go to the line in the source code where you want to insert a breakpoint, right click and select Breakpoint → Insert Breakpoint.

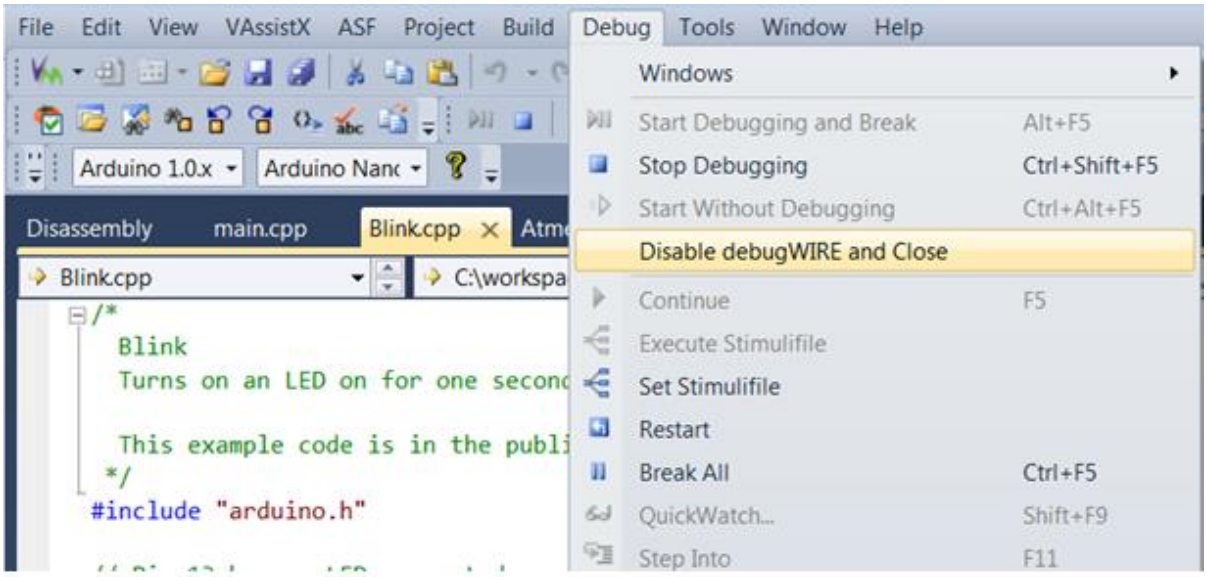


RESULT **Breakpoint is inserted.**

```
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

3. Run to Breakpoint by clicking “continue”. You can pause and continue the execution as per requirement.

4. Exit debug mode: Select Debug → Disable debugWIRE and Close.



WARNING **It is important to disable debugWIRE.**

INFO **Disabling debugWIRE resets the target and the DWEN fuse is reset, you will be able to use the ISP interface again. Having the DWEN fuse programmed enables some parts of the clock system to be running in all sleep modes. This will increase the power consumption of the AVR while in sleep modes. The DWEN Fuse should therefore always be disabled when debugWIRE is not used.**

5. Reset the board and observe that the LED blinking.

5 ATmega328P Application

We are going to create a simple application to read a light sensor using the ADC, read a temperature sensor using the I²C interface and store data in the SD card using the SPI interface. Also the light sensor value and temperature value is transmitted through the EDBG COM port.

For the I²C the “Wire” library from Arduino is required and for SD card the “SD library” is required to be included.

5.1 Compiler Setup

1. Right Click Project (*here ATmega328P_1*) and select Properties.
2. Under Toolchain → AVR/GNU C Compiler → Directories add (note that the previously added directories should still be in the list)

   C:\Program Files (x86)\Arduino\hardware\arduino\avr\libraries\Wire

   C:\Program Files (x86)\Arduino\hardware\arduino\avr\libraries\Wire\utility

   C:\Program Files (x86)\Arduino\hardware\arduino\avr\libraries\SPI

   C:\Program Files (x86)\Arduino\libraries\SD\src

   C:\Program Files (x86)\Arduino\libraries\SD\src\utility
3. Repeat for Toolchain → AVR/GNU C++ Compiler → Directories.

WARNING **Make sure you uncheck “Relative Path” while adding these paths.**

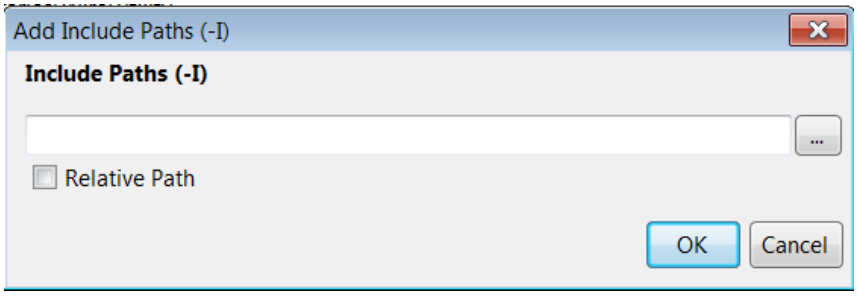
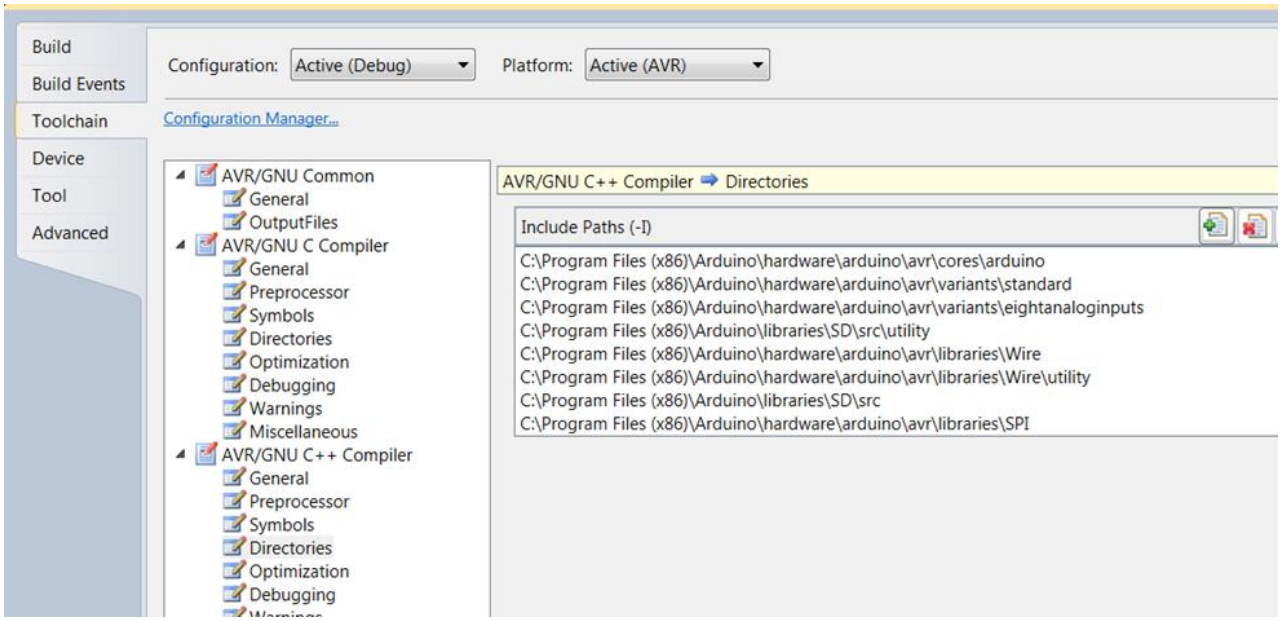


Figure 5-1. Atmel Studio: Compiler Setup

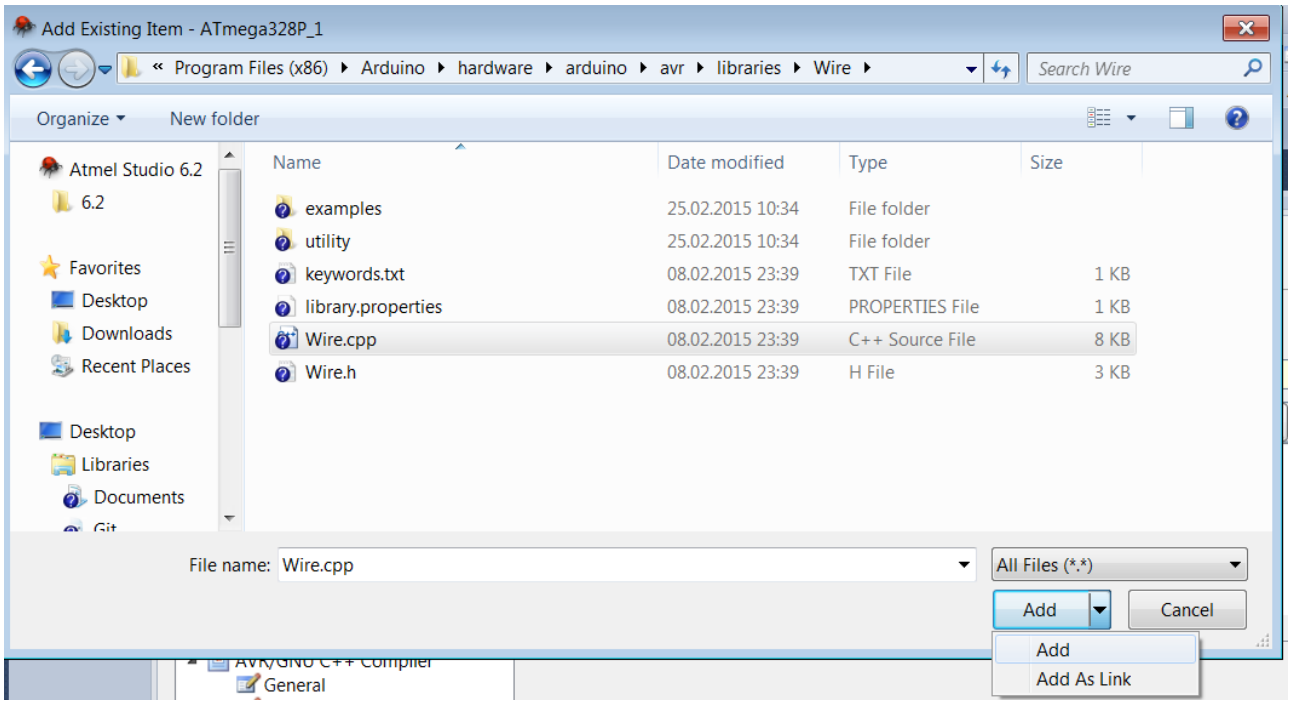


5.2 Add Dependency Files

TO DO **Add the .cpp source files from libraries to your actual Atmel Studio project.**

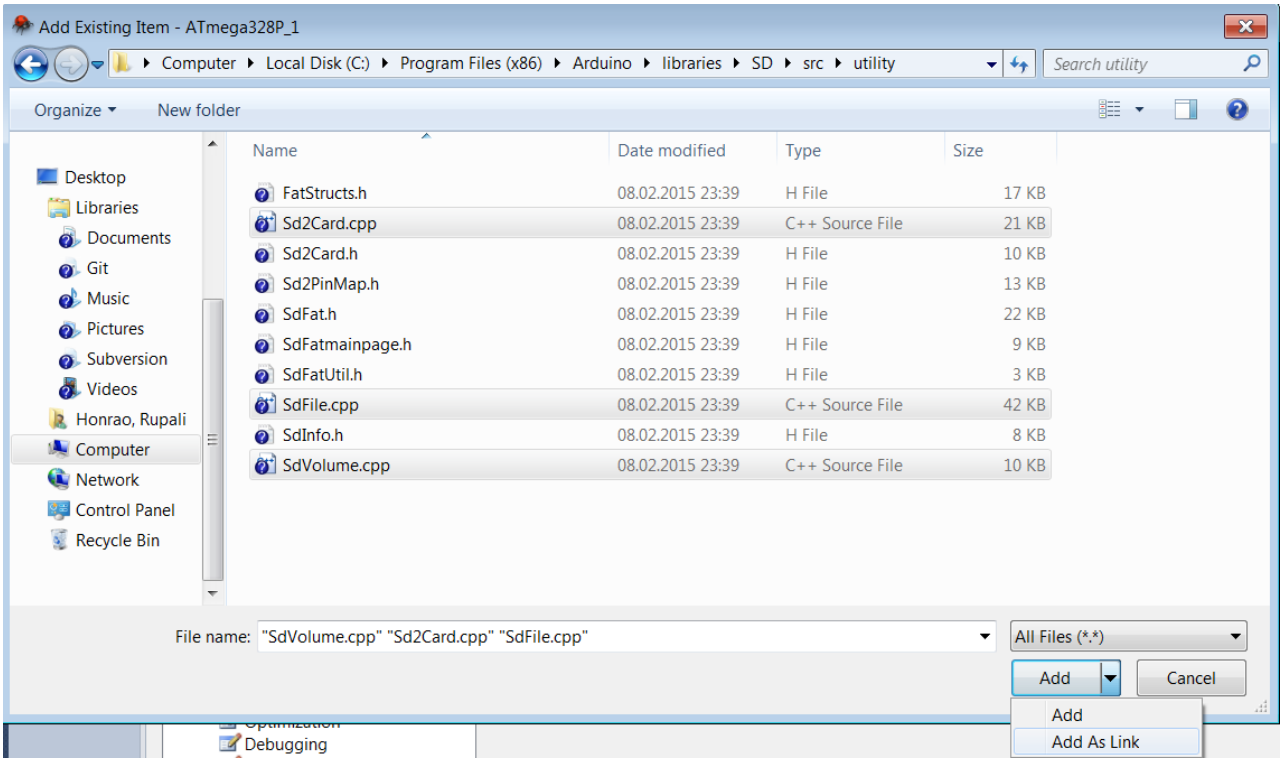
5.2.1 For Wire Library

1. Right click the project; go to “Add → Existing Item…”
2. In the dialog box that opens, aim the browser at
   a. C:\Program Files (x86)\Arduino\hardware\arduino\avr\libraries\Wire
   b. Select Wire.cpp file and add ‘As a link’
   c. Repeat the same for “wire\utility” subdirectory, select twi.c file and add ‘As a link’



5.2.2 For SD Library

1. Right click the project; go to “Add → Existing Item…”
2. In the dialog box that opens, aim the browser at
   a. C:\Program Files (x86)\Arduino\hardware\arduino\avr\libraries\SPI
3. Select SPI.cpp file and add ‘As a link’
4. Repeat the same for C:\Program Files (x86)\Arduino\libraries\SD\src
   a. Multiple Select SD.cpp, File.cpp file.
   b. Repeat the same for same for “SD\utility” subdirectory, multiple select SdVolume.cpp, Sd2Card.cpp, SdFile.cpp, and add ‘As a link’.



5.3 Developing Application

EXECUTE **Rename current Blink.cpp file to sensors.cpp file and delete LED blinking code. (I.e. delete setup() and loop().)**

EXECUTE **Include header files for SD and Wire library at the top in sensors.cpp file.**

```
#include <Wire.h>
#include <SD.h>
```

EXECUTE **Add definitions regarding temperature sensors.**

```
#define AT30TSE_TEMPERATURE_TWI_ADDR      0x4F
#define AT30TSE_TEMPERATURE_REG           0x00
#define AT30TSE_TEMPERATURE_REG_SIZE      2
#define AT30TSE_NON_VOLATILE_REG          0x00

#define AT30TSE_CONFIG_RES_9_bit          0
#define AT30TSE_CONFIG_RES_10_bit         1
#define AT30TSE_CONFIG_RES_11_bit         2
#define AT30TSE_CONFIG_RES_12_bit         3
uint16_t resolution = AT30TSE_CONFIG_RES_12_bit;
```

EXECUTE **Define constants for used pins.**

```
const int analogInPin = A0;  // Analog input pin
const int chipSelect = 10;    //SPI slave select pin
```

EXECUTE **Define variable to use the SD library.**

```
File myFile;
```

EXECUTE **Define variables to store ADC result and temperature.**

```
double temp_result;
int sensorValue = 0;        // value read from ADC A0
```

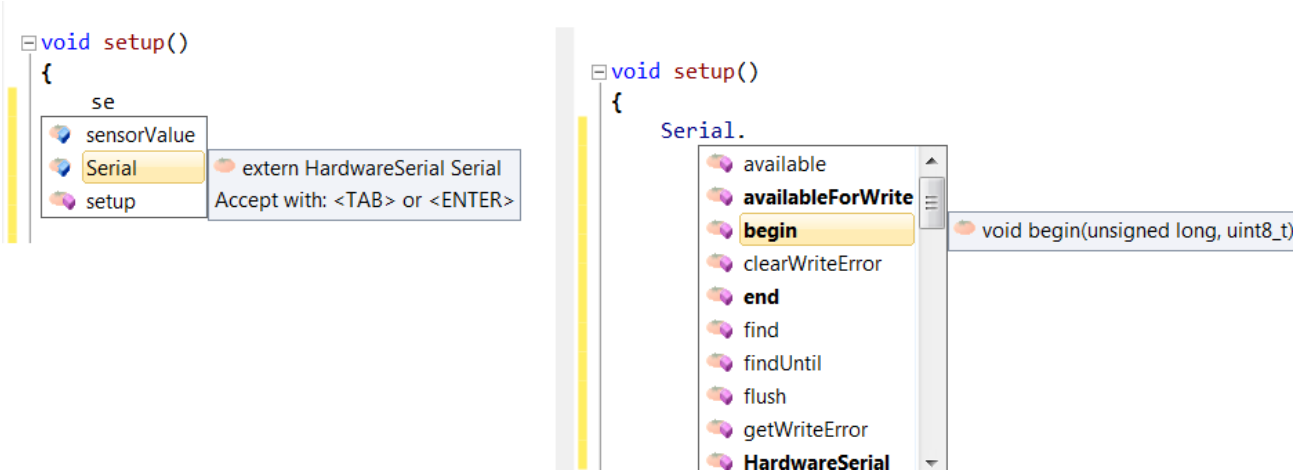
EXECUTE **Add functions to read temperature sensor.**

```
uint16_t at30tse_read_register(uint8_t reg, uint8_t reg_type, uint8_t reg_size)
{
	uint8_t buffer[2],i=0;
	buffer[0] = reg | reg_type;
	buffer[1] = 0;

	/* Internal register pointer in AT30TSE */
	Wire.beginTransmission(AT30TSE_TEMPERATURE_TWI_ADDR);
	Wire.write(buffer[0]);
	Wire.endTransmission();


	Wire.requestFrom(AT30TSE_TEMPERATURE_TWI_ADDR, reg_size);

	while(Wire.available())
	{
		buffer[i] = Wire.read(); // receive a byte as character
		i++;
	}

	return (buffer[0] << 8) | buffer[1];
}
```

```
double at30tse_read_temperature()
{
	/* Read the 16-bit temperature register. */
	uint16_t data = at30tse_read_register(AT30TSE_TEMPERATURE_REG,
	AT30TSE_NON_VOLATILE_REG,
	AT30TSE_TEMPERATURE_REG_SIZE);

	double temperature = 0;
	int8_t sign = 1;

	/*Check if negative and clear sign bit. */
	if (data & (1 << 15)){
		sign *= -1;
		data &= ~(1 << 15);
	}

	/* Convert to temperature  */
	switch (resolution){
		case AT30TSE_CONFIG_RES_9_bit:
		data = (data >> 7);
		temperature = data * sign * 0.5;
		break;
		case AT30TSE_CONFIG_RES_10_bit:
		data = (data >> 6);
		temperature = data * sign * 0.25;
		break;
		case AT30TSE_CONFIG_RES_11_bit:
		data = (data >> 5);
		temperature = data * sign * 0.125;
		break;
		case AT30TSE_CONFIG_RES_12_bit:
		data = (data >> 4);
		temperature = data * sign * 0.0625;
		break;
		default:
		break;
	}
	return temperature;
}
```

EXECUTE **Add initializing function setup().**

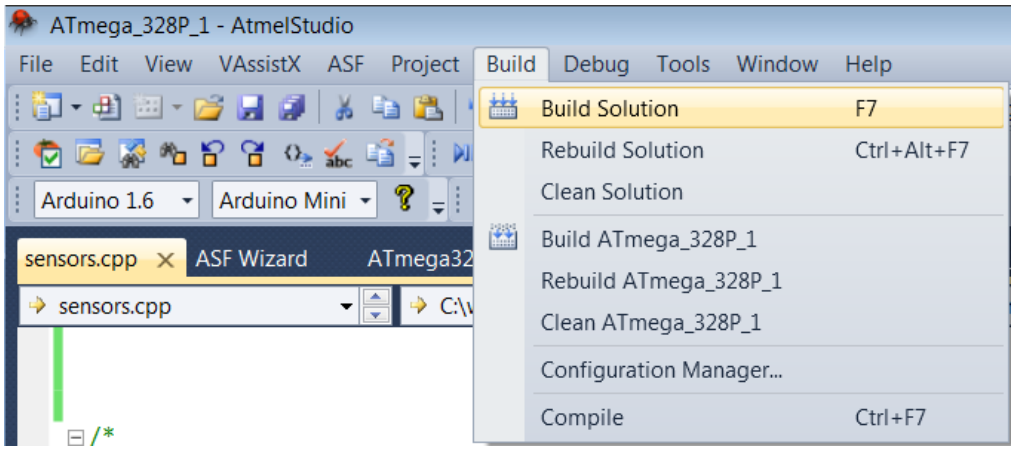
INFO **As soon as we start writing a function name in Atmel Studio possible function names are listed as shown below.**




```
void setup()
{
       Serial.begin(9600);         // Open serial communications

        if (!SD.begin(chipSelect)) {
               Serial.println("SD Card initialization failed!");
               return;
        }
        Serial.println("SD Card initialization done.");


       SD.remove("data.txt");

        Wire.begin();
}
```

EXECUTE **Add a variable and function loop().**

```
bool sensor_flg=0;

void loop() {

	sensorValue = analogRead(analogInPin);                  // read the ADC:
	temp_result = at30tse_read_temperature();       // read temperature

	if (sensorValue>=500)
	{
		sensor_flg=1;
	}

	if (sensor_flg==1)                // write data in file on particular sensor vale
	{
		sensor_flg=0;

		 myFile = SD.open("data.txt", FILE_WRITE);

		 if (myFile)         // if the file opened okay, write to it:
		 {
			myFile.print("sensor = " );
			myFile.print(sensorValue);

			myFile.print("    temp = ");
			myFile.print(temp_result);
			myFile.print("\n");

			myFile.close();     // close the file:

		 }
		 else
		 {
			// if the file didn't open, print an error:
			Serial.println("error opening data.txt");
		 }

		 // re-open the file for reading:
		 myFile = SD.open("data.txt");
		 if (myFile)
		 {
			// read from the file until there's nothing else in it:
			while (myFile.available()) {
				Serial.write(myFile.read());
			}
			// close the file:
			myFile.close();
			} else {
			// if the file didn't open, print an error:
			Serial.println("error opening data.txt");
		 }
	 }
	delay(500);
}
```



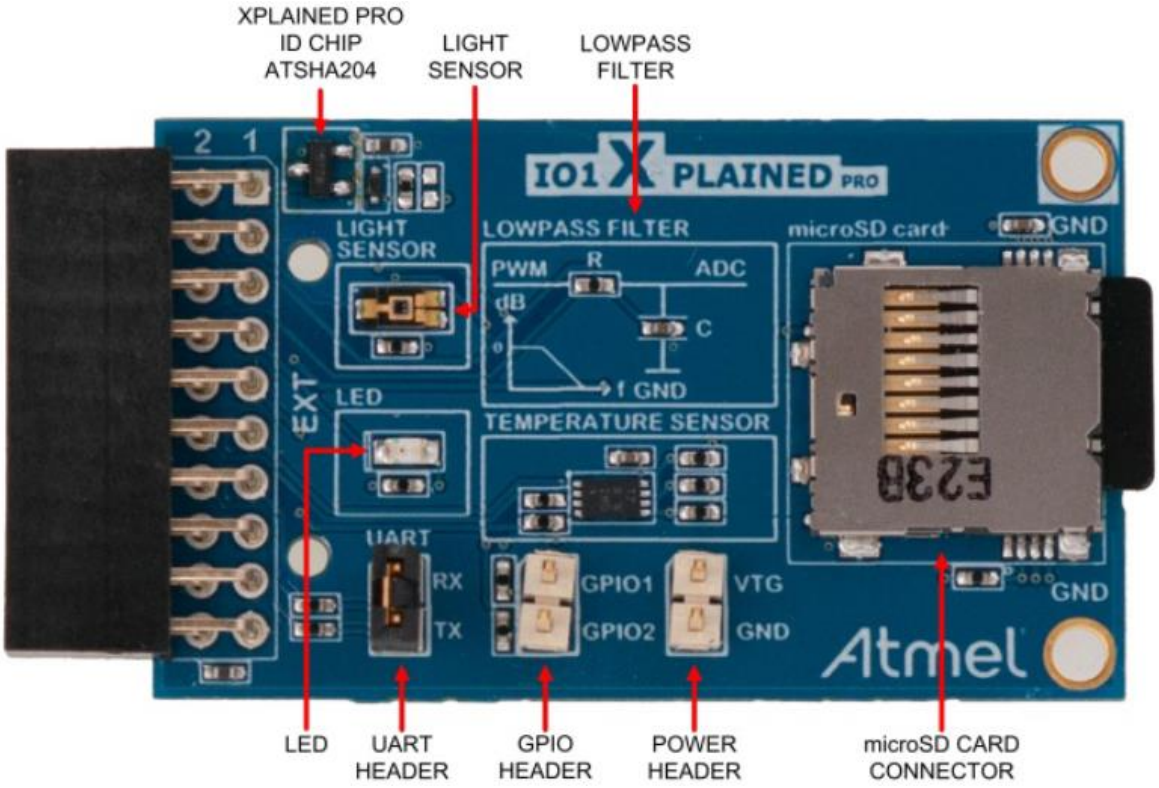
RESULT **The build should finish successfully with zero errors.**

5.4 Hardware Connection

Light sensor and temperature senor are present on IO1 Xplained Pro board.

IO1 Xplained Pro is an extension board to the Atmel Xplained Pro evaluation platform. It is designed to give a wide variety of functionality to Xplained Pro MCU boards, including a microSD card, a temperature sensor, a light sensor and more. The IO1 extension board is shown in Figure 5-2.

Figure 5-2. IO1 Xplained Pro



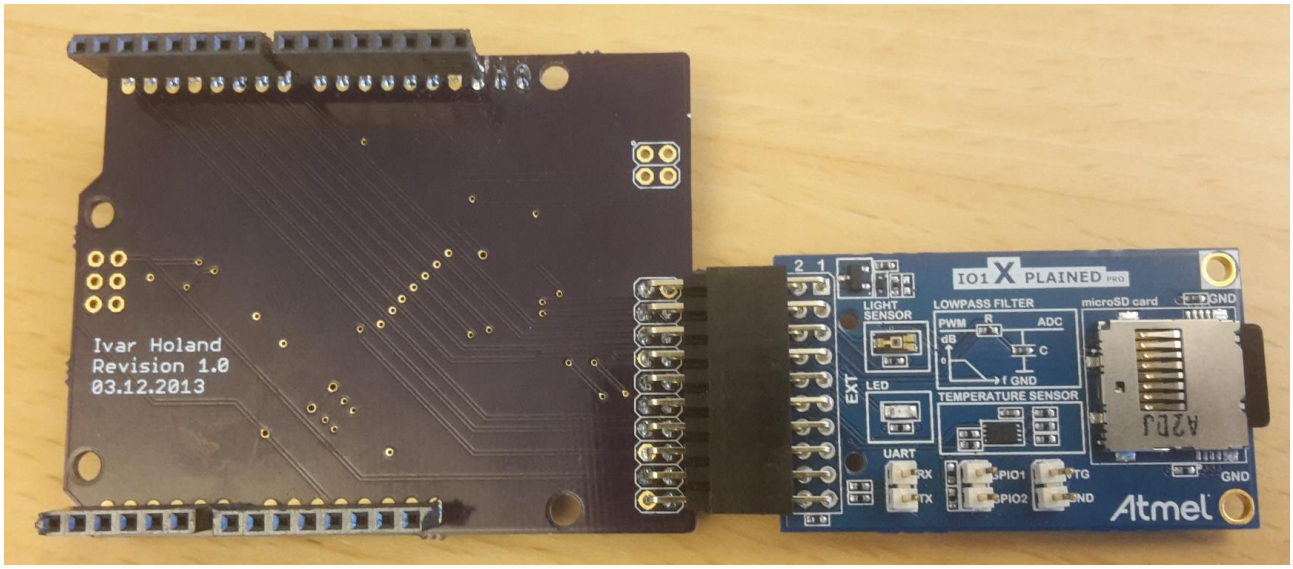
Actual connections of IO1 Xplained Pro board to ATmega328P Xplained Mini are as shown in Table 5-1.

Table 5-1. Connection: IO1 Xplained Pro- ATmega328P Xplained Mini

IO1 Xplained Pro		ATmega328P Xplained Mini	
Pin number	**Name**	**Pin**	**MCU pin**
3	ADC	PC0	A0
11	TWI_SDA	PC4	SDA
12	TWI_SCL	PC5	SCL
2	GND	GND	
20	VCC	3V3	
15	SPI_SS_A	PB2	SPI_SS(D10)
16	SPI_MOSI	PB3	SPI_MOSI
17	SPI_MISO	PB4	SPI_MISO
18	SPI_SCK	PB5	SPI_SCK

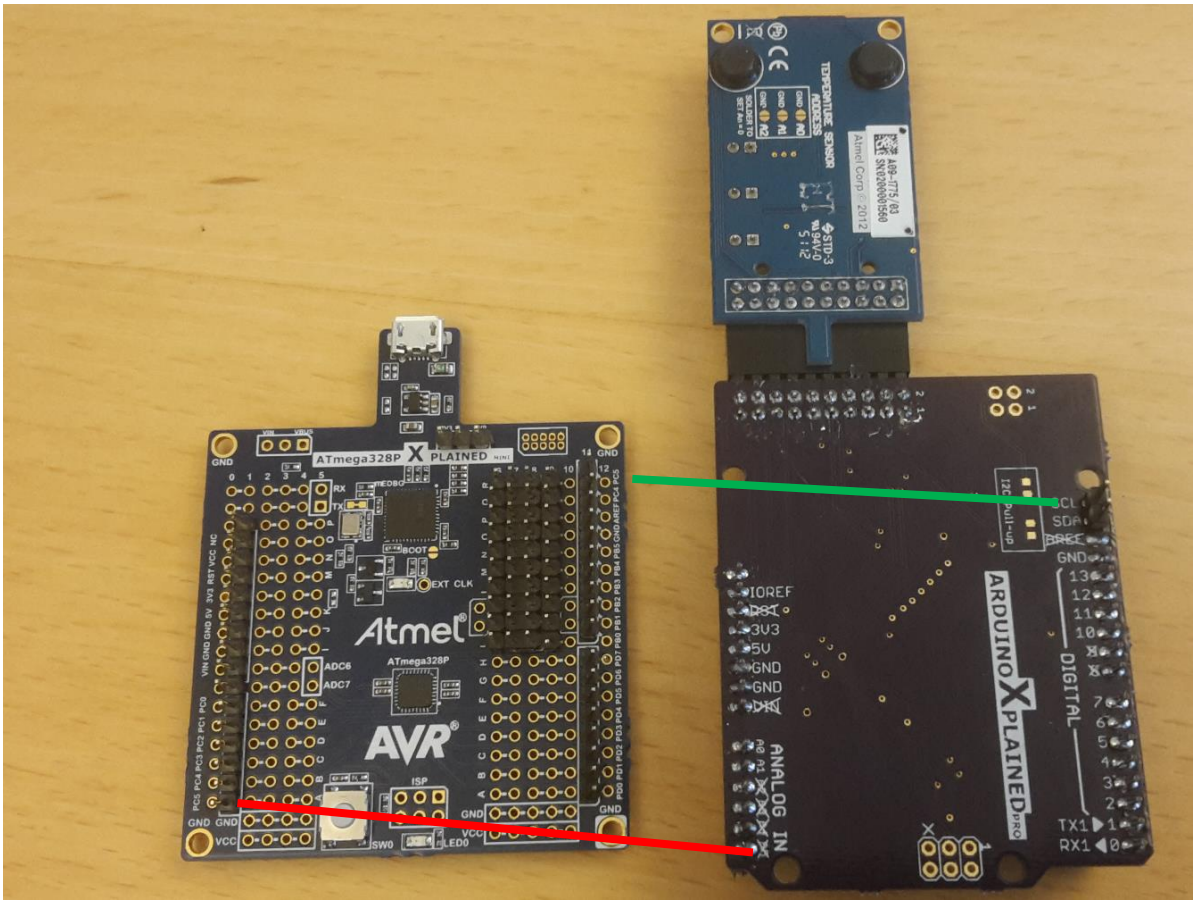
5.4.1 Connection: IO1 Xplained Pro – Arduino Xplained Pro

Connect the IO1 Xplained Pro board to Arduino Xplained Pro board as shown in below figure.



5.4.2 Connection: ATmega328P Xplained Mini - Arduino Xplained Pro

Now we need to connect ATmega328P Xplained Mini to the Arduino Xplained Pro board. Connect so that PC5 from column 1 on the Xplained Mini board goes in A5 of the Arduino Xplained Pro (red colored connection) and PC5 from column 11 on the Xplained Mini goes in SCL of to the Arduino Xplained Pro (green colored connection).



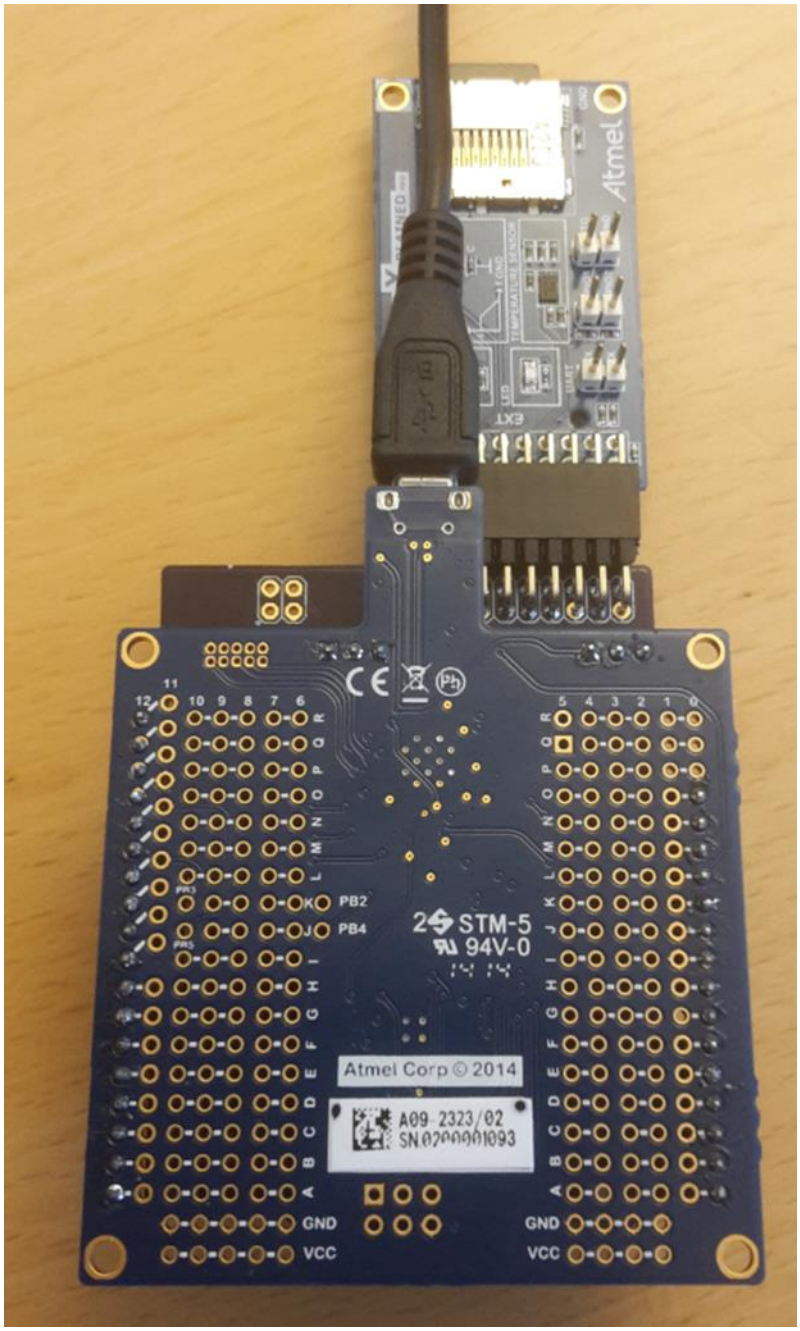
Connection would be as below.

5.4.3 Connection: USB cable

Connect the USB cable to the ATmega328P Xplained Mini board and place the board as shown below.

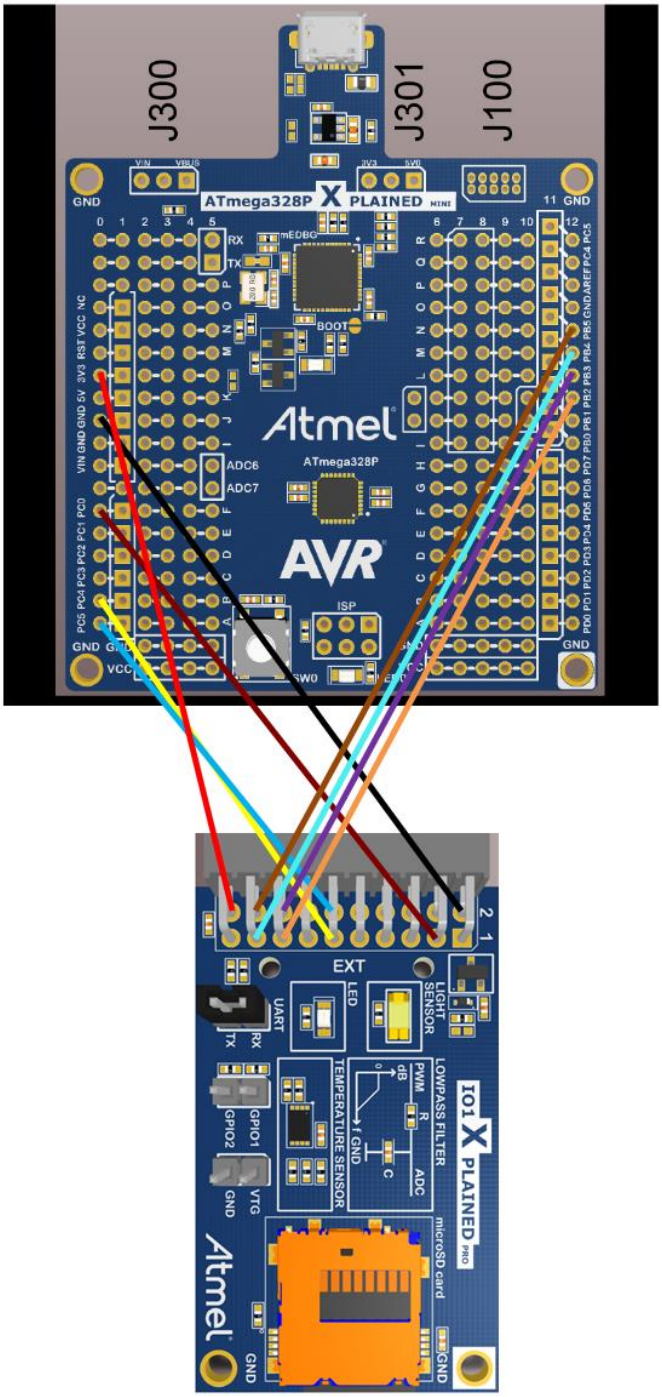
WARNING **Make sure SD card is properly inserted in the socket.**

Figure 5-3. Visual Representation of Hardware Connection for Assignment 4



INFO **Board is placed in this way that the light sensor wouldn't be in dark.**

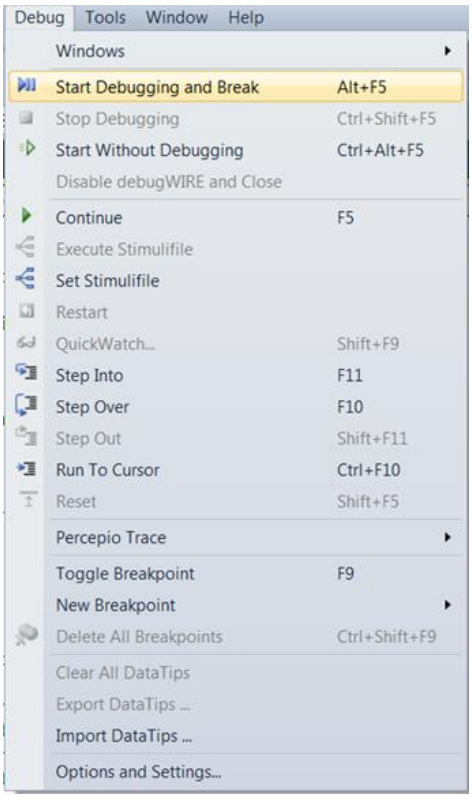
INFO **Alternative way of connection: User can also connect IO1 Xplained Pro board using nine male to female connectors without using Arduino Xplained Pro board as shown in figure below.**



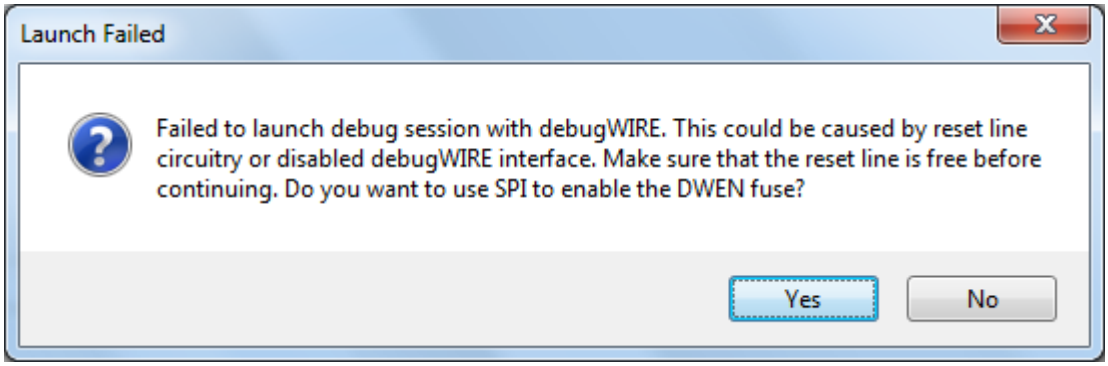
5.5 Debugging the Application

Now we will debug the application. Code has been written such that at particular condition (when sensor value > 500) the light sensor value and temperature value will be stored in the SD card. We will place a breakpoint at this condition and check that the breakpoint is hit.

TO DO **Select Debug and click 'Start Debugging and Break'.**

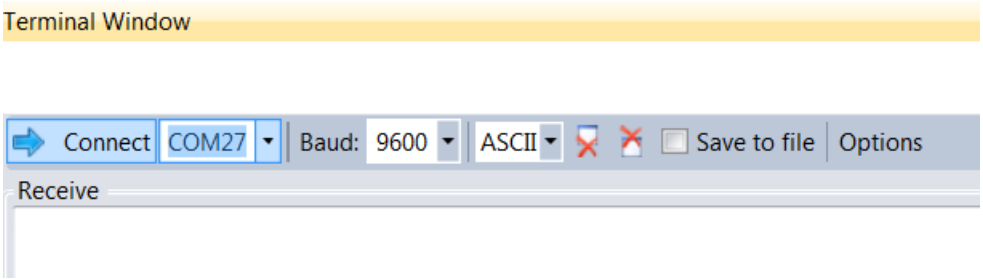


WARNING **If the DWEN fuse is not enabled and error message is displayed. Click 'Yes' and Studio will use the ISP to set the fuse as shown below.**



RESULT **The Debugger is started and breaks in main. You are now ready to start debugging.**

TO DO **In Atmel Studio select View → Terminal Window. Select the mEDBG COM Port, Baud: 9600 and select "Connect".**



INFO **Sensor data will be received through the mEDBG COM port.**

TO DO **Open sensors.cpp file from the Solution Explorer and place the breakpoint as shown in the figure below:**

```
void loop() {

    sensorValue = analogRead(analogInPin);
    temp_result = at30tse_read_temperature();

    if (sensorValue>=500)
    {
        sensor_flg=1;
    }

    if (sensor_flg==1)          // write data
    {
```

TO DO **Now start debugging by selecting symbol  or by pressing F5 key on keyboard.**

RESULT **In the Terminal window a message will appear: 'SD Card initialization done'.**

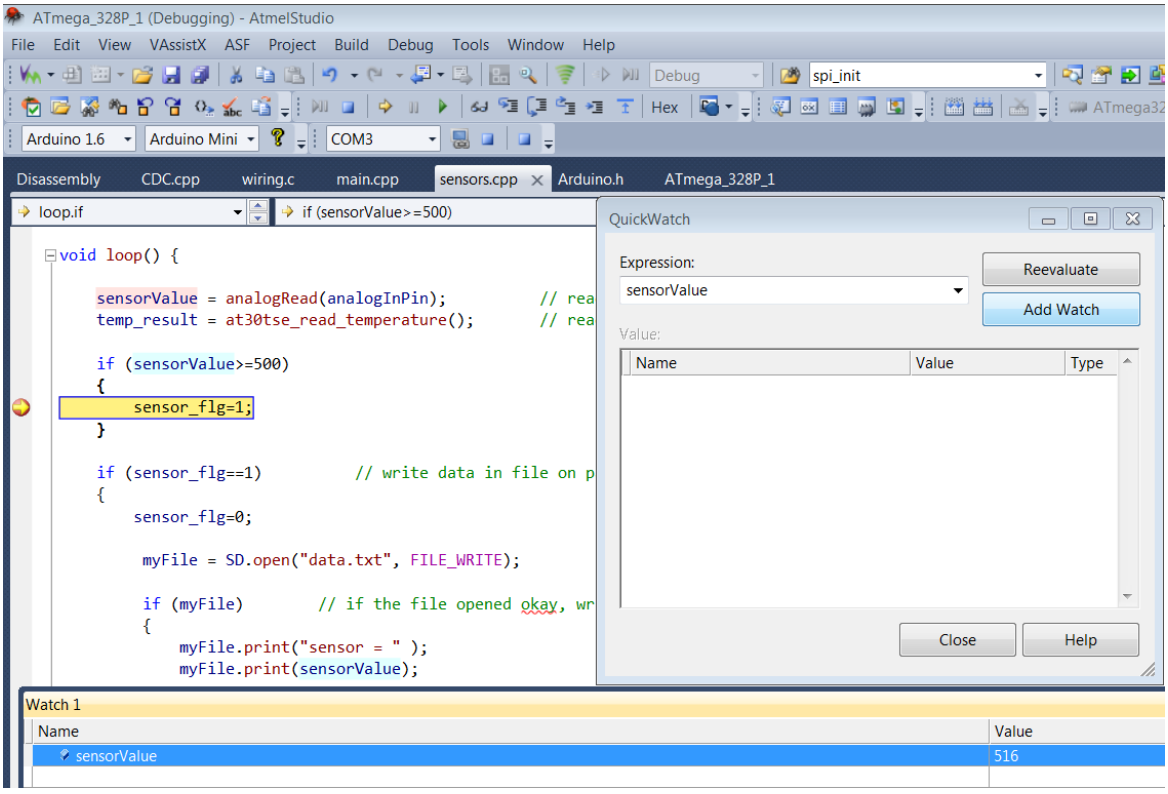
INFO **A light sensor is present on the IO1 Xplained Pro and sensor value increases when it is dark and decreases when it is bright.**

TO DO **Cover the sensor with your finger to increase the sensor value.**
As the sensor value is increased the condition if (sensorValue ≥500) will be true and program execution will hit the breakpoint as shown below.

```
void loop() {

    sensorValue = analogRead(analogInPin);          // read th
    temp_result = at30tse_read_temperature();       // read te

    if (sensorValue>=500)
    {
        sensor_flg=1;
    }

    if (sensor_flg==1)          // write data in file on parti
    {
        sensor_flg=0;
```

TO DO **To view the sensor value while debugging, select Debug → QuickWatch… Enter the expression 'sensorValue' and select button 'Add Watch'.**

RESULT **In Watch1 window value of variable 'sensorValue' will be displayed.**

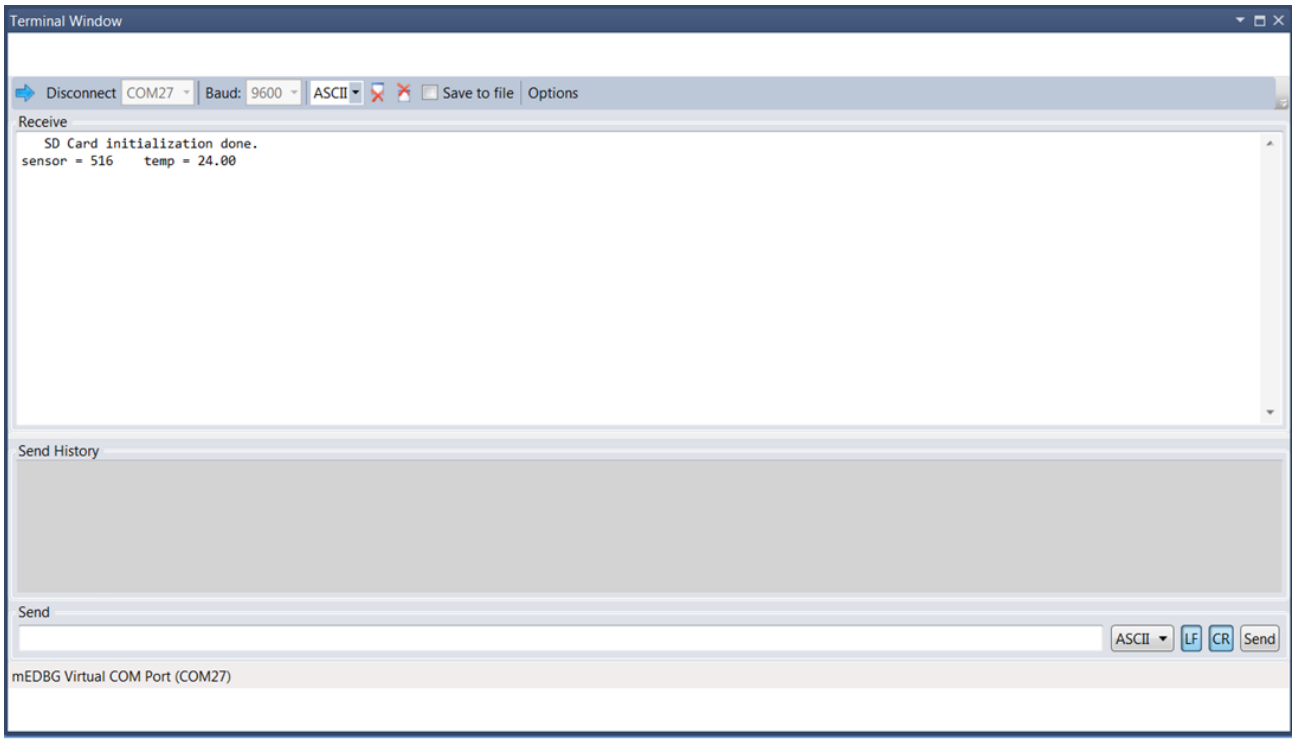


TO DO **Close the 'Watch1' window and 'QuickWatch' window.**

TO DO **Continue the execution by pressing F5 on keyboard.**

RESULT **Sensor value and temperature value will be stored in SD card also transmitted through EDBG COM port. Terminal window will show sensor values and temperature values.**

Figure 5-4. Atmel Studio: Terminal Window



TO DO **Exit the debug mode by selecting Debug → Disable debugWIRE.**

WARNING **It is important to disable debugWIRE.**

We have successfully added Arduino project on an ATmega328P Xplained Mini with the Atmel Studio.

Appendix A. Complete Solution to Assignment 4

Sensors.cpp file.

```
#include "arduino.h"
#include <Wire.h>
#include <SD.h>

#define AT30TSE_TEMPERATURE_TWI_ADDR      0x4F
#define AT30TSE_TEMPERATURE_REG           0x00
#define AT30TSE_TEMPERATURE_REG_SIZE      2
#define AT30TSE_NON_VOLATILE_REG          0x00

#define AT30TSE_CONFIG_RES_9_bit          0
#define AT30TSE_CONFIG_RES_10_bit         1
#define AT30TSE_CONFIG_RES_11_bit         2
#define AT30TSE_CONFIG_RES_12_bit         3

uint16_t resolution = AT30TSE_CONFIG_RES_10_bit;
double temp_result;
int sensorValue = 0;        // value read from ADC A0


const int analogInPin = A0;  // Analog input pin
const int chipSelect = 10;


// set up variable to use SD utility library functions:
File myFile;



void setup()
{
       Serial.begin(9600);         // Open serial communications

        if (!SD.begin(chipSelect)) {
               Serial.println("SD Card initialization failed!");
               return;
        }
        Serial.println("SD Card initialization done.");


       SD.remove("data.txt");

        Wire.begin();
}
```

```
uint16_t at30tse_read_register(uint8_t reg, uint8_t reg_type, uint8_t reg_size)
{
	uint8_t buffer[2],i=0;
	buffer[0] = reg | reg_type;
	buffer[1] = 0;

	/* Internal register pointer in AT30TSE */
	Wire.beginTransmission(AT30TSE_TEMPERATURE_TWI_ADDR);
	Wire.write(buffer[0]);
	Wire.endTransmission();

	Wire.requestFrom(AT30TSE_TEMPERATURE_TWI_ADDR, reg_size);

	while(Wire.available())
	{
		buffer[i] = Wire.read();
		i++;
	}

	return (buffer[0] << 8) | buffer[1];
}

double at30tse_read_temperature()
{
	/* Read the 16-bit temperature register. */
	uint16_t data = at30tse_read_register(AT30TSE_TEMPERATURE_REG,
	AT30TSE_NON_VOLATILE_REG,
	AT30TSE_TEMPERATURE_REG_SIZE);

	double temperature = 0;
	int8_t sign = 1;

	/*Check if negative and clear sign bit. */
	if (data & (1 << 15)){
		sign *= -1;
		data &= ~(1 << 15);
	}

	/* Convert to temperature */
	switch (resolution){
		case AT30TSE_CONFIG_RES_9_bit:
		data = (data >> 7);
		temperature = data * sign * 0.5;
		break;
		case AT30TSE_CONFIG_RES_10_bit:
		data = (data >> 6);
		temperature = data * sign * 0.25;
		break;
		case AT30TSE_CONFIG_RES_11_bit:
		data = (data >> 5);
		temperature = data * sign * 0.125;
		break;
		case AT30TSE_CONFIG_RES_12_bit:
		data = (data >> 4);
		temperature = data * sign * 0.0625;
		break;
		default:
		break;
	}
	return temperature;
}
```

```
bool sensor_flg=0;

void loop() {

       sensorValue = analogRead(analogInPin);                  // read the ADC:
       temp_result = at30tse_read_temperature();       // read temperature

       if (sensorValue>=500)
       {
              sensor_flg=1;
       }

       if (sensor_flg==1)                // write data in file on particular sensor vale
       {
              sensor_flg=0;

               myFile = SD.open("data.txt", FILE_WRITE);

               if (myFile)         // if the file opened okay, write to it:
               {
                      myFile.print("sensor = " );
                      myFile.print(sensorValue);

                      myFile.print("    temp = ");
                      myFile.print(temp_result);
                      myFile.print("\n");

                      myFile.close();     // close the file:

               }
               else
               {
                      // if the file didn't open, print an error:
                      Serial.println("error opening data.txt");
               }

               // re-open the file for reading:
               myFile = SD.open("data.txt");
               if (myFile)
               {
                      // read from the file until there's nothing else in it:
                      while (myFile.available()) {
                             Serial.write(myFile.read());
                      }
                      // close the file:
                      myFile.close();
                      } else {
                      // if the file didn't open, print an error:
                      Serial.println("error opening data.txt");
               }
        }
       delay(500);
}
```

6 Conclusion

In this hands-on training we have accomplished the following tasks:

- added support form Arduino style coding to Atmel studio
- added debugging capability to Arduino project
- based an Arduino project on Atmel Xplained Mini and Atmel Studio without changing any code
- added the ability to write more complex and complicated programs

7 Revision History

Doc. Rev.	Date	Comments
42439A	08/2015	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**