

Introduction to the Atmel Studio 6 Environment

Colin Tan

(colin.ky.tan@gmail.com)

1. Introduction

In this document you will be introduced to the Atmel Studio 6 environment. Atmel Studio 6 is a powerful integrated development environment from Atmel consisting of an AVR compiler that can produce code for the Atmel Atmega328 that you are using, a debugger that lets you step through your code and simulate execution on the Atmega328, and several other useful features as well.

You will also be introduced to avrdude (AVR Downloader/UploaDEr), a tool that lets you transfer programs written on your PC/notebook onto the Arduino board.

2. Downloading and Installing the Arduino Environment

Arduino is also an open-source software standard, comprising of a simple software development environment, tools to upload written programs to the Arduino board, and a set of libraries that simplify access to the hardware. If you do not have the Arduino IDE, download it from this link:

<http://arduino.cc/en/Main/Software>

Installation instructions for Windows users:

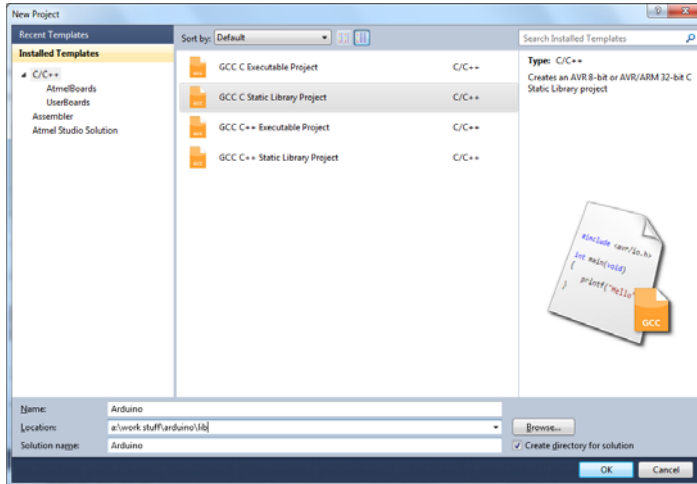
<http://arduino.cc/en/Guide/Windows>

I will assume that you have installed the Arduino software in c:\. The version used in this report is v1.0.1, and the software is installed in c:\arduino-1.0.1. In newer versions of Arduino for Windows installed using the Windows installer, the Arduino is installed in c:\Program Files\Arduino. Modify the instructions to suit your actual installation.

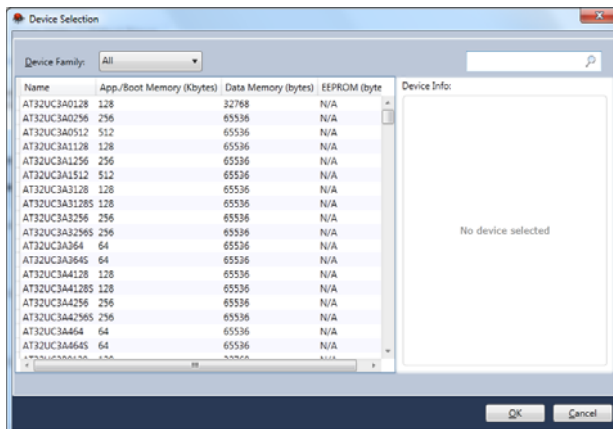
3. Compiling the Arduino Libraries

The first thing we must do now is to create a project to compile all the Arduino libraries into a single static library that we can link into our C programs. To do this:

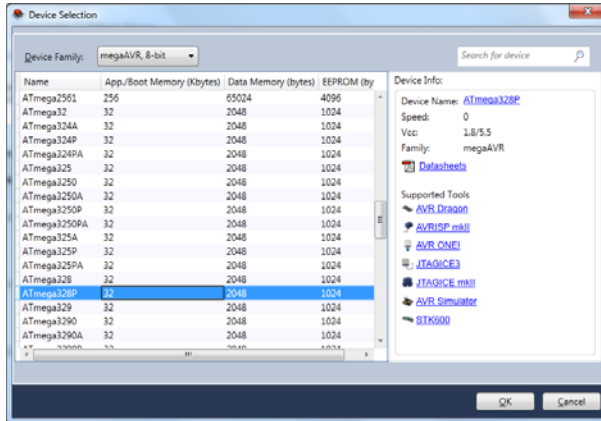
- i. Start up Atmel Studio.
- ii. Click File->New->Project, and the following dialog box will pop up:



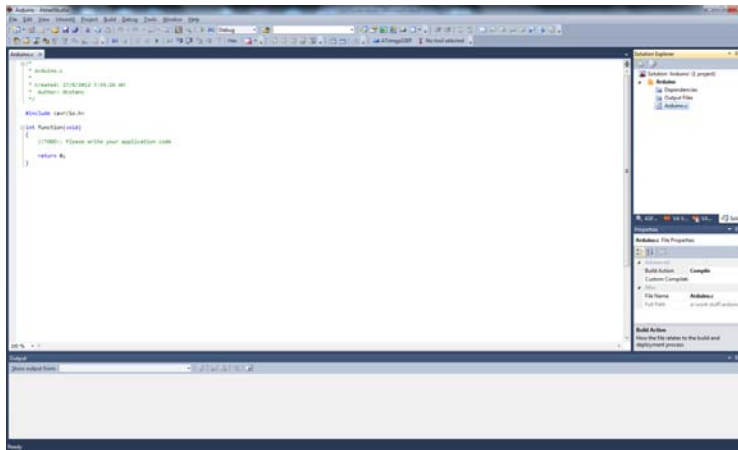
- iii. Select “GCC C Static Library Project”, and enter “arduino” in “Name”. Choose your work directory in “Location”. It does not have to be exactly as shown above. You can call your Location “c:\project” as long as the path already exists.
- iv. Click OK. This will bring up the Device Selection dialog box, shown below:



Under “Device Family”, select “megaAVR, 8-bit”, then select “Atmega328p” from the list below. Your dialog box will look like this:

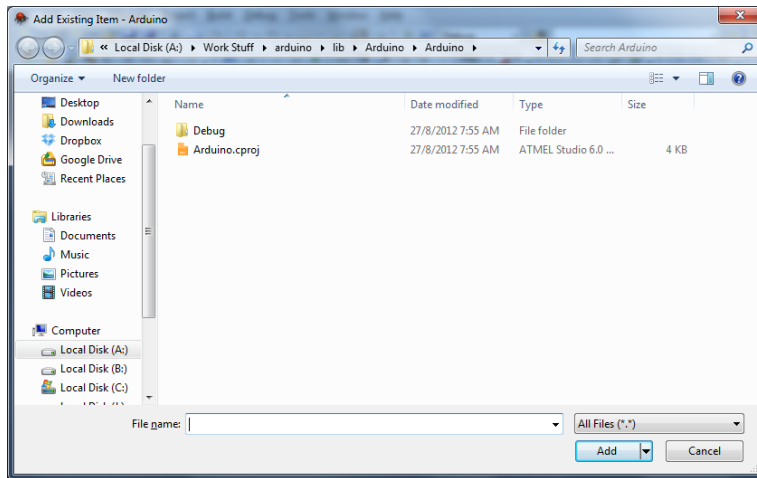


Click “OK”. This will bring up a dummy source code file like the one shown below:



RIGHT click on “arduino.c” in the solution explorer pane (Click “View->Solution Explorer” if the pane is missing), then click on “Remove” to remove the arduino.c file from your solution. Click “Delete” on the dialog box that appears.

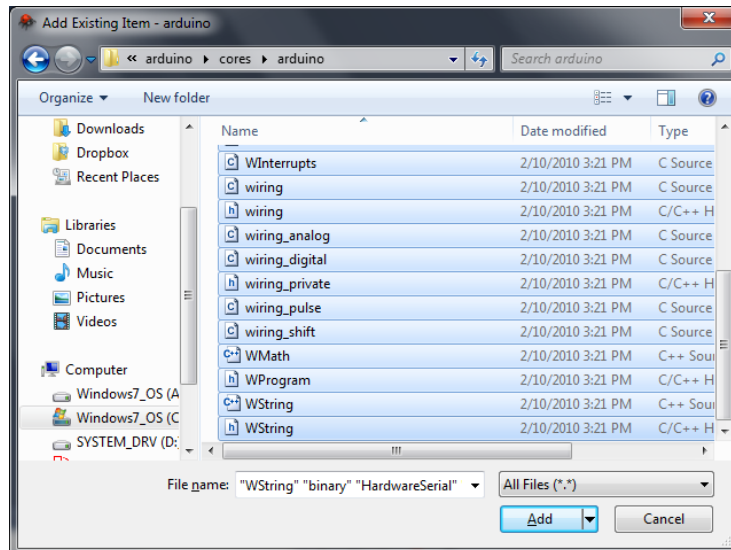
- v. Now RIGHT-CLICK on the bold word “arduino” with the yellow symbol next to it, and choose Add->Existing Item from the context menu. This will bring up the following dialog box:



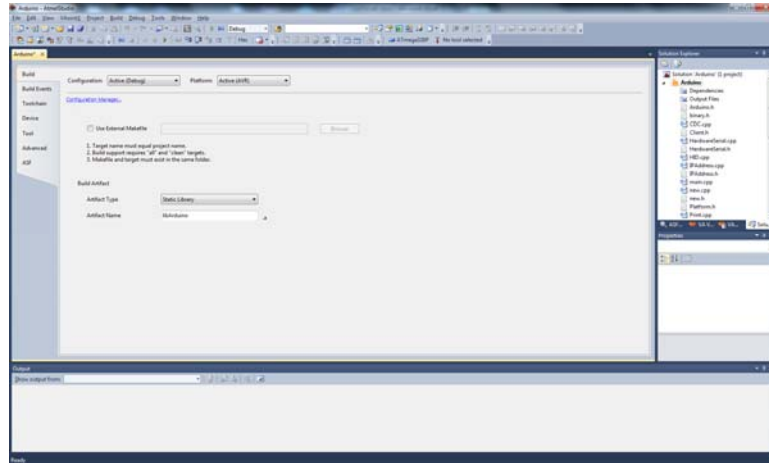
Navigate to the directory containing the source code for the Arduino library. This will be at “<Arduino root directory>\hardware\arduino\cores\arduino”. If you have installed Arduino in c:\arduino-1.0.1, then the full path is:

C:\arduino-1.0.1\hardware\arduino\cores\arduino

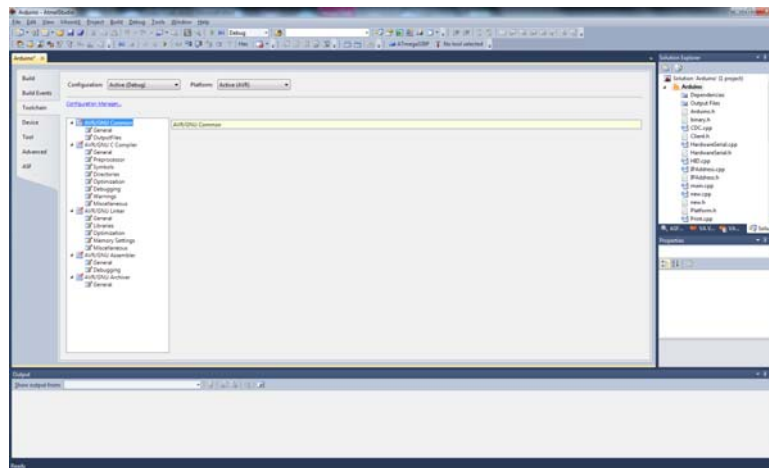
Select all the files and add click Add.



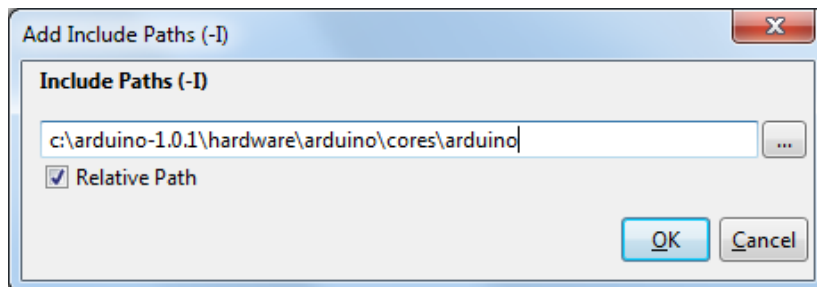
vi. You now need to set up the Include directories in your project. To do this, the bold “Arduino” word in the Solution Explorer pane, then click on Project->Properties, which brings up this dialog box:



Click on the “Toolchain” tab on the left, giving you:



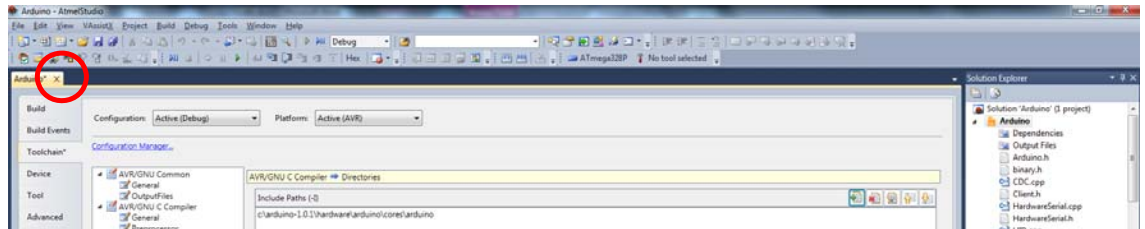
Now click on “Directories” under “AVR/GNU C Compiler”, and a textbox labelled “Include Paths (-I)” appears on the right panel, together with some buttons. Click on the first button (it should show a green “+”), which brings up a dialog box that says “Add Include Paths (-I)”. Enter the full path to the Arduino library source code:



Click OK.

Repeat, and this time add c:\arduino-1.01\hardware\arduino\variants\standard if you are compiling for the ATmega168 or ATmega328 (e.g. for the Arduino Uno), and c:\arduino-1.01\hardware\arduino\variants\mega if you are compiling for the ATmega1280 or ATmega2560 (e.g. for the Arduino Mega).

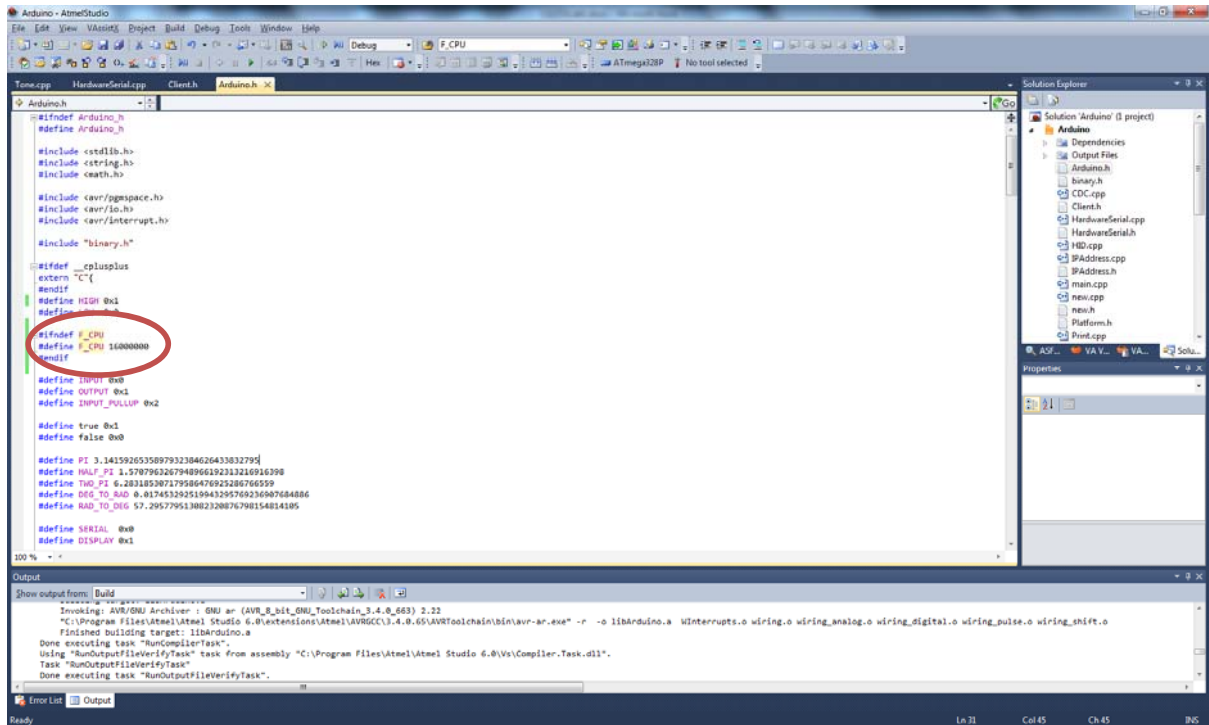
- vii. Now click the small “X” next to the word “arduino”, circled below, to close this dialog box.



- viii. Double click “Arduino.h” in the right Solution Explorer panel to bring up the source code for this file. Locate the line “#define LOW 0x0”, then type in the following:

```
#ifndef F_CPU
#define F_CPU 1600000L
#endif
```

Your screen will look like this. The added code is circled:



- ix. Finally click Build->Build Solution to compile the library. If all goes well the Output pane at the bottom of the IDE should show “Build Succeeded”.
- x. You will be able to locate your library file in the “<Location>\arduino\arduino\debug” directory. So if your Location that you set when you first created the project is c:\projects, then the full path to the library file is:

C:\projects\arduino\arduino\debug

(Note: If you can't find the debug directory, it might be c:\projects\arduino\arduino\arduino\debug”. AS6 appears to add an extra level of directories)

If you navigate to this directory, you will find a file called “libarduino.a”. This is the static library file that contains all the arduino routines you will need for the next section.

4. Creating a New Compiler Project

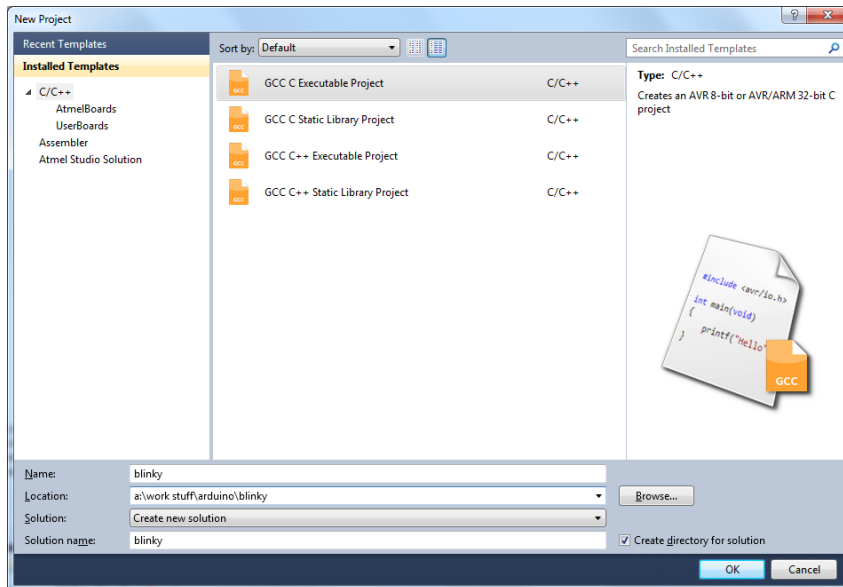
We will now create our own version of the Blinky program that you saw earlier, except that this time we will build a circuit with two LEDs (one red and one green), and alternately flash the two LEDs.

4.1 Building the Circuit

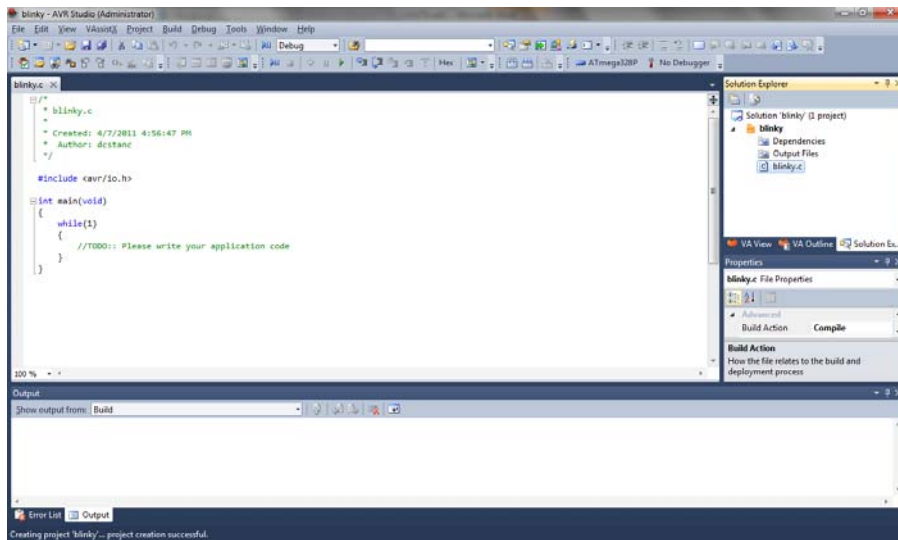
To try this demo, connect an LED to digital pins 12 and 13.

4.2 Creating the Project in AS6

Close any open projects by clicking “File->Close Solution” in Atmel Studio 6, then click “File->New->Project”. This time choose “GCC C Executable Project”, and enter “blinky” in the Name field. You can choose any directory for your Location.



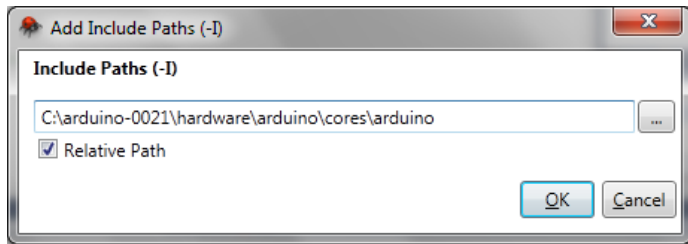
Click “OK” to continue, and again choose “megaAVR, 8-bit” and “ATmega328P” from the Device Selection dialog box. Click OK, and once again an empty source file appears:



We will use the Arduino library to access the hardware.

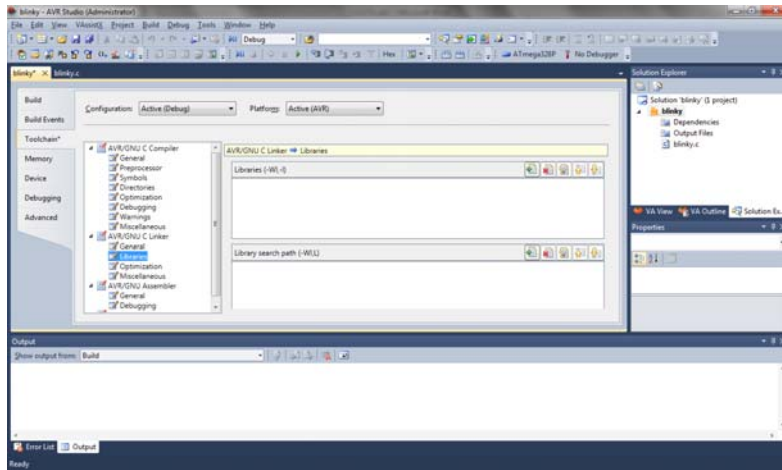
4.3 Setting Up the Include Path, Library and Library Path

As before, we must set up our Include path. To do this, click on Project->blinky Properties, then click on the “Toolchains” tab. Under AVR/GNU C Compiler click on “Directories”, then add in the path to your Arduino library source code. In the example here it is “c:\arduino-1.0.1\hardware\arduino\cores\arduino”, your own particular path may be slightly different, but it is always “<Arduino root directory>\hardware\arduino\cores\arduino”.

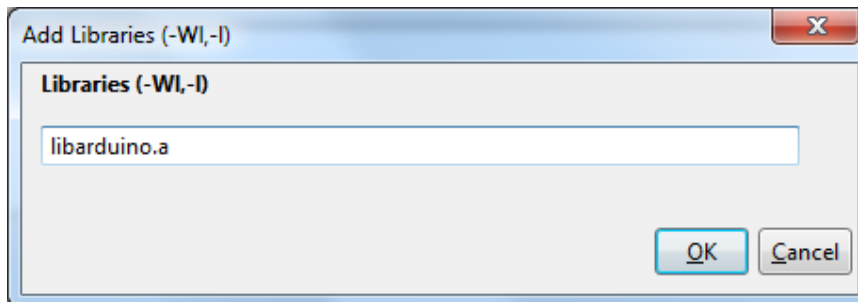


Click OK. Also add in `c:\arduino-1.0.1\hardware\arduino\variants\standard`.

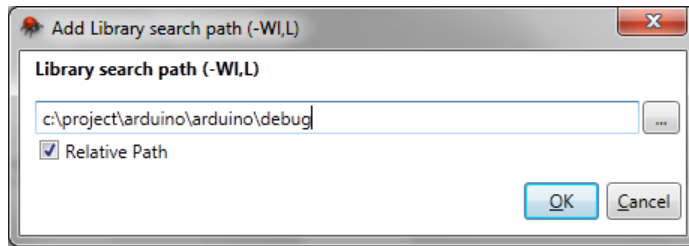
Now click on “Libraries” under “AVR/GNU C Linker”:



In the “Libraries (-WL, -l)” field, click on the first button (the one with the green +), and enter “libarduino.a”. Click OK.

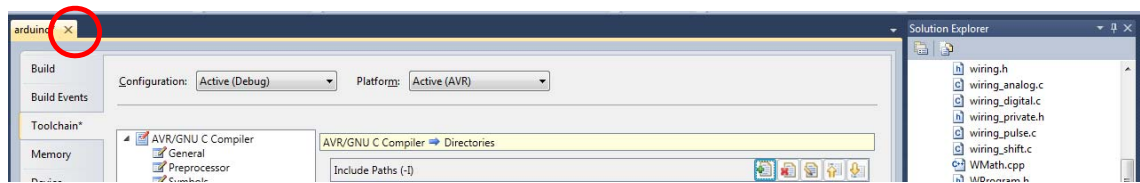


In the “Library Search Path (-WL, L)”, click on the first button (again with the green +), and enter the path to the library you compiled earlier. In our example it is `c:\project\arduino\arduino\debug`:



Click OK.

Close the properties dialog by clicking on the “X”.



4.4 Keying in The Code and Compiling

Now that you have the project properties set up properly, key in the following program:

```

/*
 * blinky.c
 *
 * Created: 24/8/2012 4:56:47 PM
 * Author: Colin Tan
 */

#include <avr/io.h> // Header file to access Atmega328 I/O registers
#include <Arduino.h> // Header file for the Arduino library
#define GREEN_PIN 12
#define RED_PIN 13

void blink_led(unsigned pinnum)
{
    digitalWrite(pinnum, HIGH); // Set digital I/O pin to a 1
    delay(1000); // Delay
    digitalWrite(pinnum, LOW); // Set digital I/O pin to a 0
    delay(1000); // Delay
}

void setup()
{
    pinMode(GREEN_PIN, OUTPUT); // Set digital I/O pins 12
    pinMode(RED_PIN, OUTPUT); // and 13 to OUTPUT.
}

int main(void)
{
    init(); // Initialize arduino. Important!
    setup(); // Set up the pins

    while(1)
    {

```

```
    blink_led(RED_PIN);  
    blink_led(GREEN_PIN);  
}  
}
```

Click “Build->Build Solution” to build the project. If all goes well the “Output” window will show “Build succeeded.”. If not you will have to check your typing, and ensure that all the directories and libraries are set up properly according to the instructions above.

4.5 Keying in The Code and Compiling

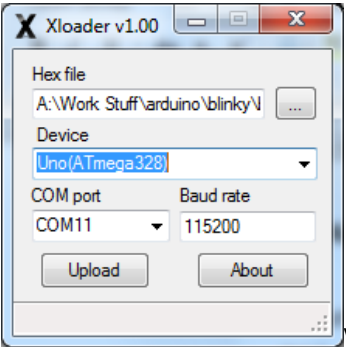
Assuming that the code compiled properly, connect the Arduino board now to your PC/notebook using the USB cable provided. If you have not already done so, follow the instructions in the link below:

<http://arduino.cc/en/Guide/Windows>

Ensure that you know which COM port the Arduino is connected to. The examples shown here will assume COM11, but it is DIFFERENT for different computers! In fact if you connect two different Arduino boards to the same computer, the COM ports will be different!

4.6 Uploading the Code using XLoader

- a. Download XLoader from <http://russemotto.com/xloader/>.
- b. Unzip the XLoader directory to a convenient place like your desktop.
- c. Launch XLoader.exe, choose “Uno(ATmega328)” under “Device”, the correct COM port number of the Arduino in COM port, and a baud rate of 115.200.



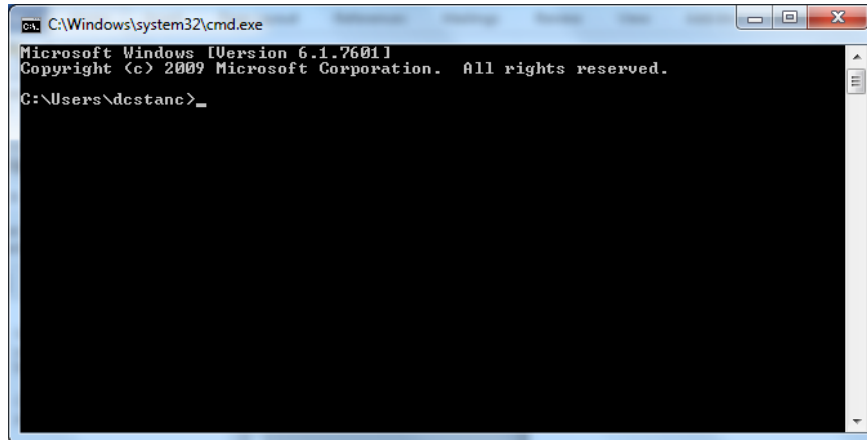
- d. Click the “...” button at Hex file, navigate to the directory where the .hex file is, click on the .hex filename, then click Upload. This will upload the file onto the Arduino.

4.7 Uploading the Code using avrdude

If you want better control you can also upload code using avrdude. You will now need to shell to DOS. To do so in Windows 7, press CTRL-ESC to bring up the Windows menu, and

enter "cmd" in the "Search programs and files" field, and press ENTER. On Windows 7 this may take more than one attempt as it needs to find cmd.exe.

For Windows XP, click on "Start->Run", then enter "cmd" and click "OK". You will get a black DOS box similar to this:



You now need to determine the path to your Arduino "bin" directory. This is always "<Arduino root directory>\hardware\tools\avr\bin". For example, if <Arduino root directory> is "c:\arduino-1.0.1", then the full path to the bin directory is:

```
"c:\arduino-1.0.1\hardware\tools\avr\bin"
```

Enter the following command in the DOS window to properly set up the path:

```
set PATH=%PATH%;c:\arduino-1.0.1\hardware\tools\avr\bin
```

You need one more step; you need to copy the avrdude.conf file from the c:\arduino-1.0.1\hardware\tools\avr\etc directory. To do this:

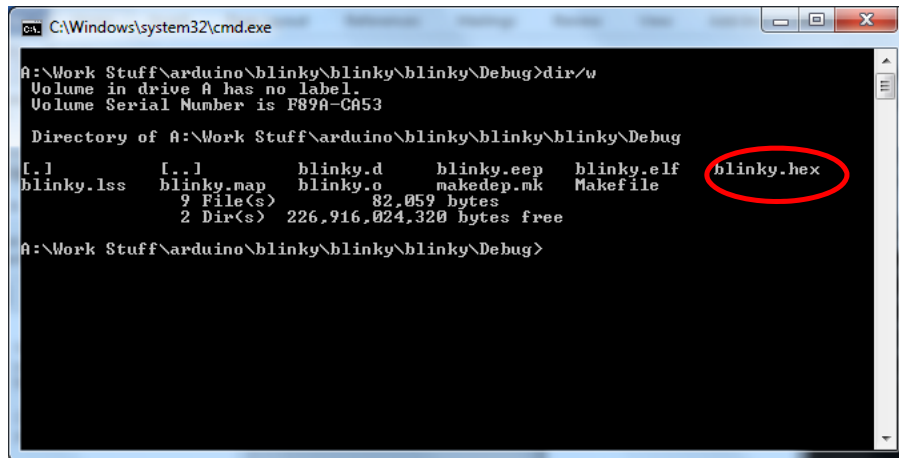
```
cd c:\arduino-1.0.1\hardware\tools\avr\bin
copy c:\arduino-1.0.1\hardware\tools\avr\etc\* .
```

You should see "c:\arduino-1.0.1\hardware\tools\avr\etc\avrdude.conf 1 file(s) copied". Now change to the debug directory of your project. For this project it is "<Location>\blinky\blinky\debug". If your "Location" is c:\project, then your debug directory is "c:\project\blinky\blinky\debug". Enter the following command in the DOS prompt.

```
cd c:\project\blinky\blinky\debug
```

(Note: If you can't find the debug directory, it might be c:\project\blinky\blinky\blinky\debug". AS6 appears to add an extra level of directories)

Type "dir/w" to see the contents of the directory. In particular you should see a "blinky.hex" file, which is the executable code to be uploaded to the Arduino.



```
C:\Windows\system32\cmd.exe
A:\Work Stuff\arduino\blinky\blinky\Debug>dir/w
Volume in drive A has no label.
Volume Serial Number is F89A-CA53

Directory of A:\Work Stuff\arduino\blinky\blinky\Debug
[.]                [.]                blinky.d            blinky.eep          blinky.elf          blinky.hex
blinky.lss         blinky.map          blinky.o            makedep.mk         Makefile
                   9 File(s)          82,059 bytes
                   2 Dir(s)    226,916,024,320 bytes free

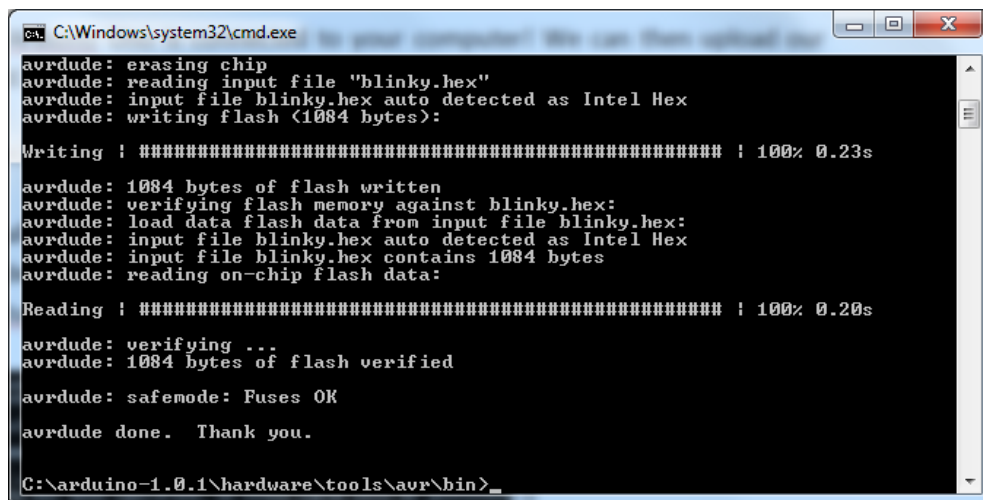
A:\Work Stuff\arduino\blinky\blinky\Debug>
```

(Note: The location given here is actually a:\work stuff\arduino instead of c:\project).

Make sure that the Arduino Uno is connected to your computer! We can then upload our program to the board. To do this, type in the following command:

```
avrdude -p m328p -b 115200 -c arduino -P\\.\COM11 -U flash:w:blinky.hex
```

Press enter, and if all goes well you will see the L, TX and RX LEDs on the board flashing quickly while the code is being uploaded. When completed the red and green LEDs will flash alternately. The avrdude program will also show the uploading progress:



```
C:\Windows\system32\cmd.exe
avrdude: erasing chip
avrdude: reading input file "blinky.hex"
avrdude: input file blinky.hex auto detected as Intel Hex
avrdude: writing flash (1084 bytes):

Writing | ##### | 100% 0.23s

avrdude: 1084 bytes of flash written
avrdude: verifying flash memory against blinky.hex:
avrdude: load data flash data from input file blinky.hex:
avrdude: input file blinky.hex auto detected as Intel Hex
avrdude: input file blinky.hex contains 1084 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 0.20s

avrdude: verifying ...
avrdude: 1084 bytes of flash verified

avrdude: safemode: Fuses OK
avrdude done. Thank you.

C:\arduino-1.0.1\hardware\tools\avr\bin>
```

The following table explains what the arguments to avrdude mean:

Argument	Meaning
-p m328p	We are uploading to an Atmega328p.
-b 115200	At a speed of 115,200 bits per second (bps)
-c arduino	Using the arduino upload protocol, which is understood by Arduino boards.
-P \\.\COM11	To COM port 11. This will probably be different on your machine. Use the Arduino environment to find out the correct port. See http://arduino.cc/en/Guide/Windows for details.
-U flash:w:blinky.hex	Write the file "blinky.hex" to flash memory.

There are other possible arguments to avrdude. For example "-D" prevents avrdude from erasing all the flash before uploading, saving some time. Type "avrdude -?" to see all of the options available.

5. Debugging

The Atmel Studio environment includes an integrated debugger that can debug using a simulator, as well as do debugging on the target boards. Unfortunately the Arduino Uno has no support for JTAG, which is needed for on-board debugging. You can therefore only debug using the simulator.

6.1 Starting the Debugger

Before starting the debugger, comment out the two "delay(1000);" statements in blink_led() as these will cause the debugger to hang:

```
void blink_led(unsigned pinnum)
{
    digitalWrite(pinnum, HIGH); // Set digital I/O pin to a 1
    // delay(1000); // Delay commented out
    digitalWrite(pinnum, LOW); // Set digital I/O pin to a 0
    // delay(1000); // Delay commented out
}
```

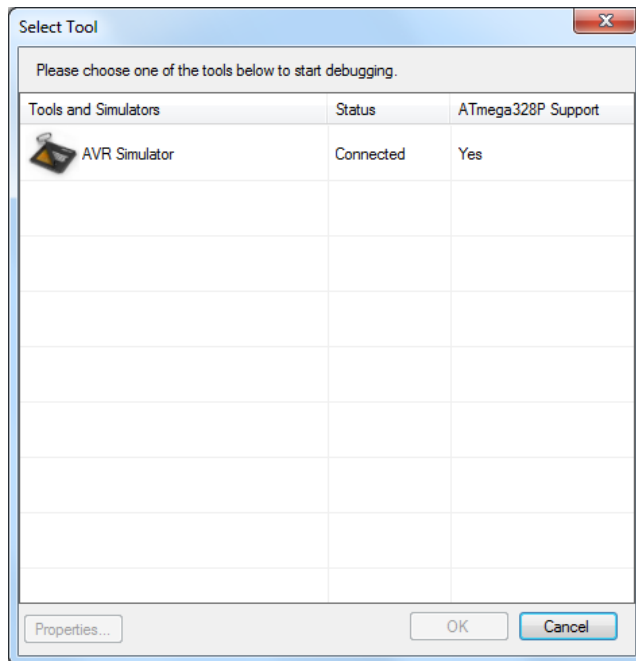
In addition add the following function:

```
void update()
{
    static int x=0;
    x=x+1;
}
```

Add the following line to “main” between the two calls to blink_led:

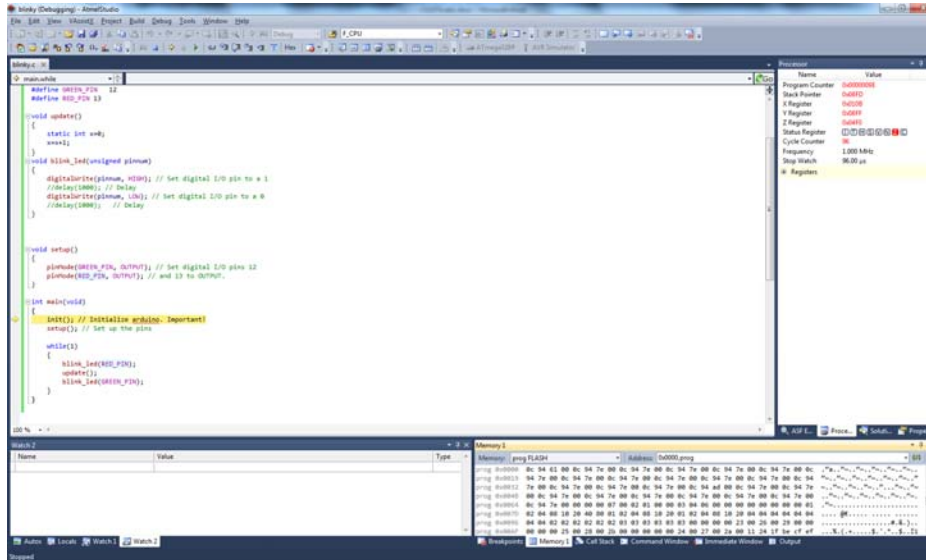
```
while(1)
{
    blink_led(RED_PIN); // existing code
    update();           // new line
    blink_led(GREEN_PIN); // existing code
}
```

Recompile the code using Build->Build Solution. To start the debugger, click Debug->Start Debugging and Break. Sometimes the debugger will ask you to choose a Debug Tool:



Click “AVR Simulator” and then click OK.

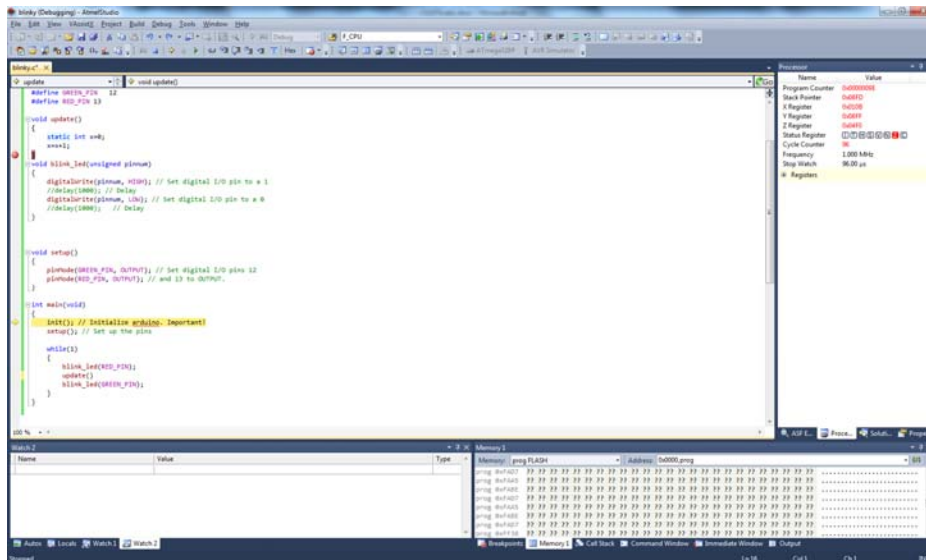
The execution will begin and break at the first statement in main. A yellow arrow appears indicating which statement is currently being executed.



In addition there should be a Breakpoint window showing all the currently set break points, a Processor window showing the values in all the CPU registers (I/O registers are not shown!), and a Watch window showing values of variables. If any of these windows are missing, you can use the View menu to open them.

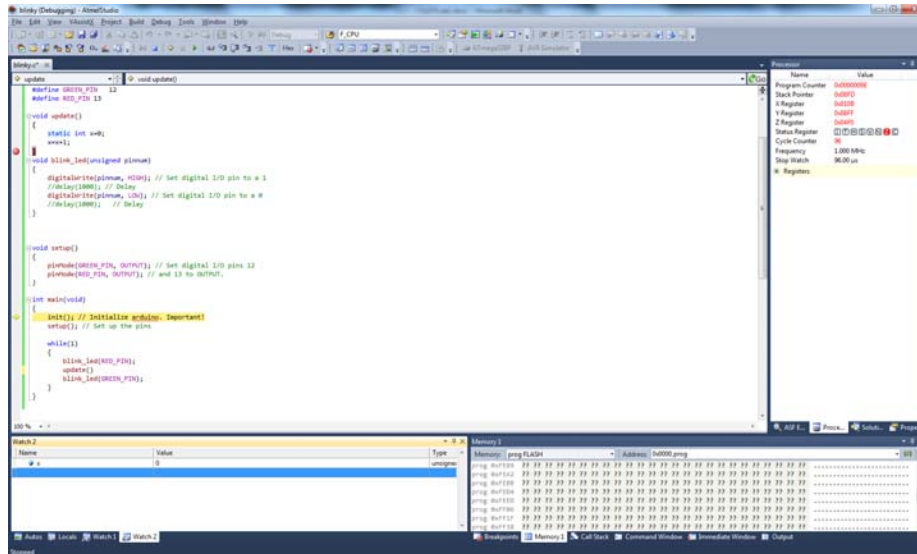
5.2 Adding Breakpoints

Execution stops whenever the debugger hits a break-point. To set a break point, LEFT click on the statement you want to break at, then RIGHT click, select Breakpoint->Insert Breakpoint. Do this now for the `x=x+1` statement in the newly inserted `update()` function. A red ball appears at the statement you've chosen:



5.3 Inserting Watches

A “watch” lets you monitor the value of a variable. To insert a watch, click on the “Name” field of an empty row and enter the variable name. In this case we will enter “x”, giving us:



It may say “Unknown identifier” if “x” is local to update() and we are currently outside of the update() function.

5.4 Stepping Through Code

You can now press “F10” or “F11” to step through the code. F10 will step OVER functions – i.e. it will execute the function as a whole unit. F11 will step INTO functions. I.e. the debugger will enter the function and step through every statement in it.

Play around with both F10 and F11 until you understand how they work. Other debug options are available from the Debug menu.

This document does not cover debugging using the I/O View, CPU View or the Disassembly View, all of which are valuable. You are encouraged to look these up yourself.