CG2271 Real Time Operating Systems
Lab 1 – Introduction to Arduino

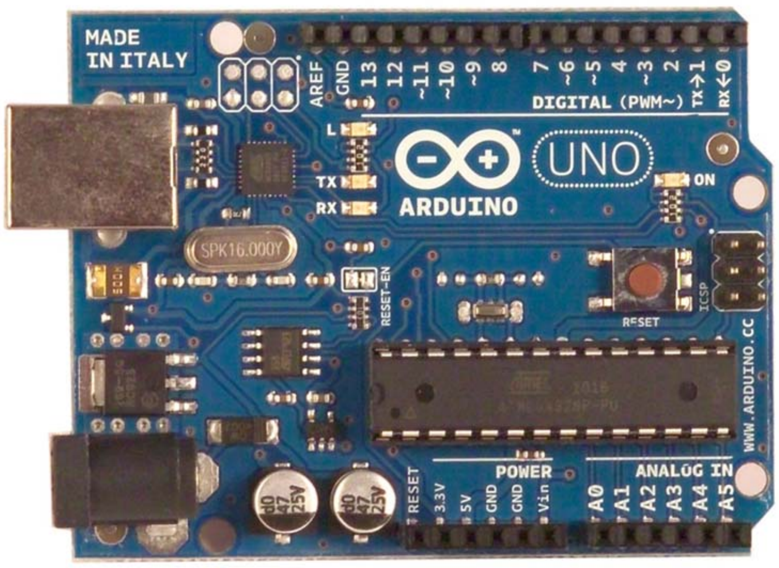
1. Introduction

In this lab you will be introduced to the AVR Studio 5 environment. AVR Studio 5 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 too.

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

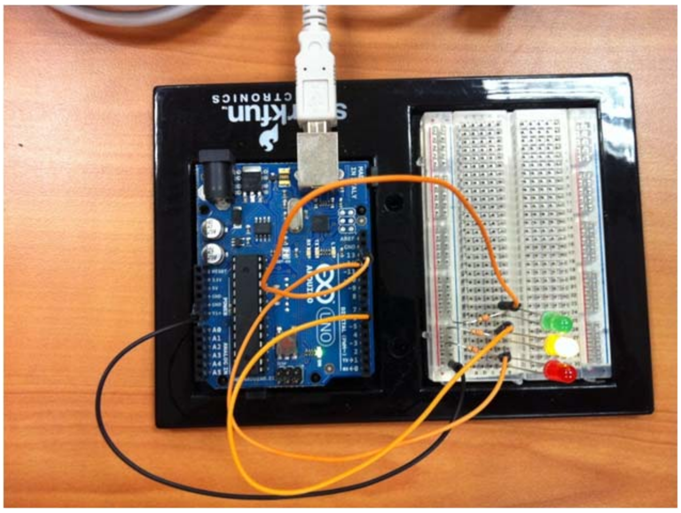
Your primary development platform will be the Arduino platform. Arduino is an open-source hardware/software platform that is designed to make building embedded systems simple. Each team of 2 persons is provided with a Sparkfun Inventor Kit, made up of an Arduino Uno processor board, a breadboard, wires and assorted electronic components to help you build up your projects. The Arduino Uno board is shown below:



The top row of sockets are labeled "0" to "13", then "GND" and "AREF". The pins labeled "0" to "13" are digital input/output pins, while GND provides electrical ground. The AREF pin provides a reference voltage for the analog to digital converters, which we will not use here.

The bottom row is labeled "A5" to "A0", then Vin, GND, GND, 5V, 3.3V and RESET. The A5 to A0 inputs are analog inputs, while Vin is to power the board from an external source. GND provides electrical ground, 5V and 3.3V provide a 5-volt and 3.3-volt electrical source, and RESET, when pulled low, causes the board to reboot. There is also a RESET button on the board for this purpose.

You will assemble your circuits on the breadboard that is provided. Connections are made between the breadboard and the Arduino using the wire jumpers provided. An example circuit is shown below:



Note that you should ALWAYS color code your wires. E.g. black for GND, red for 5V/3.3V, green for inputs, orange for outputs, etc. This makes your circuits much easier to debug.

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. For the first two labs we will be using the libraries, so you will need to download the Arduino software from this link:

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

Read through this page to gain an overview of the Arduino Project.

http://arduino.cc/en/Guide/HomePage

Follow the platform specific instructions to try out the board. 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 v0.21, and the software is installed in c:\arduino-0021. Modify the instructions to suit your actual installation.

Throughout this course I will be assuming that you are using Windows. If you are using Mac OS X or Linux, please find out the equivalent steps for yourself.

3. <u>Downloading and Installing AVR Studio 5</u>

In the previous section you would have tried out the blinky example on the Uno, using the Arduino development environment. For the rest of this course you will be writing programs using AVR Studio 5 (AS5) for the following reasons:

i. AS5 is an industrial-standard IDE for writing programs for AVR microcontrollers.
ii. AS5 provides you with a powerful debugger, which is absent in the Arduino IDE.
iii. The Arduino IDE supports the Arduino language, which is translated into C, but is not _exactly_ C.
iv. We will be moving away from the Arduino libraries as you get more proficient in programming the AVR "bare-metal", and that makes using the Arduino environment awkward.

To download AS 5, click on the following link:

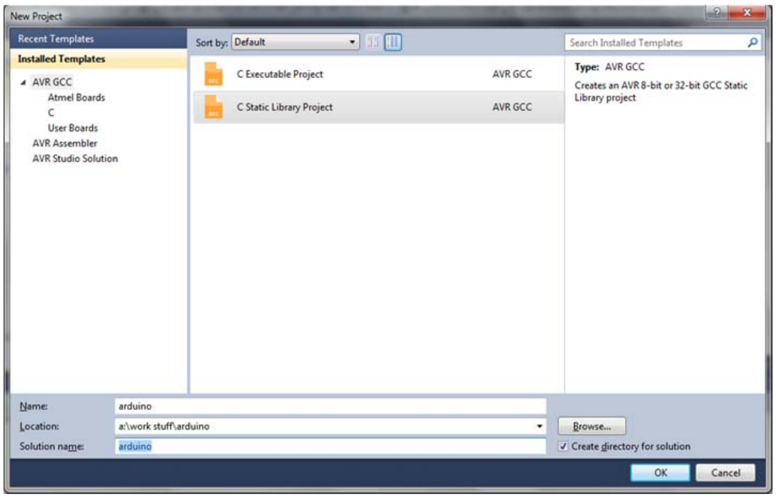
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=17212&source=avr_5_studio_overview

Register yourself, download the software and follow the instructions for installation.

4. <u>Compiling the Arduino Libraries</u>

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


New Project

Recent Templates
Installed Templates
- AVR GCC
  - Atmel Boards
  - C
  - User Boards
- AVR Assembler
- AVR Studio Solution

Sort by: Default

Search Installed Templates

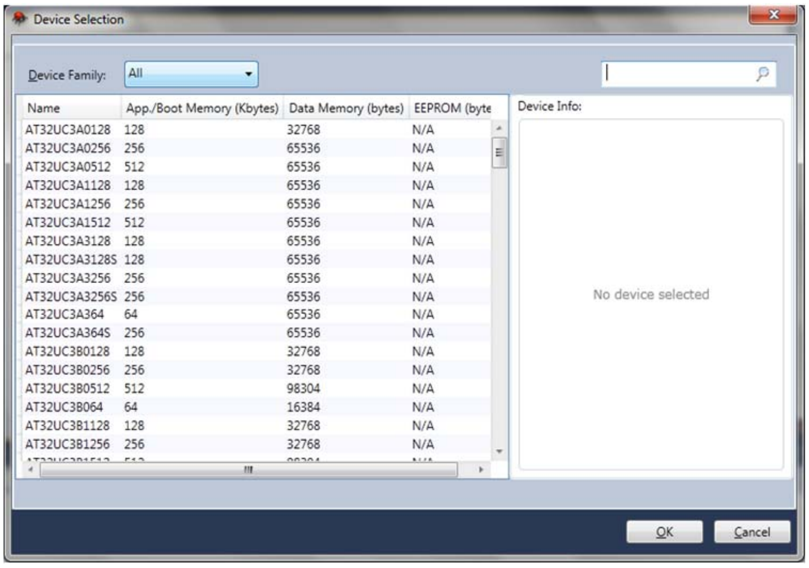
C Executable Project	AVR GCC
C Static Library Project	AVR GCC

Type: AVR GCC
Creates an AVR 8-bit or 32-bit GCC Static Library project

Name: arduino
Location: a:\work stuff\arduino
Solution name: arduino
[✓] Create directory for solution

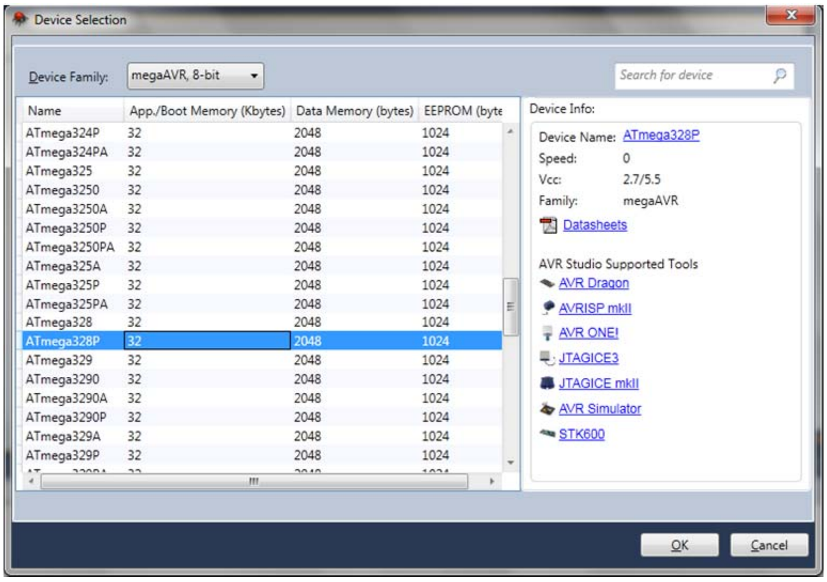
iii. Select "C Static Library Project", and enter "arduino" in "Name". Choose your work directory in "Location". It does not have to be exactly as shown above. I will assume that your “Location” is “a:\work stuff\arduino”. You can call your Location “c:\project” as long as the path already exists.

iv. Click OK. This will bring up the Device Selection dialog box, shown below:



Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (byte
AT32UC3A0128	128	32768	N/A
AT32UC3A0256	256	65536	N/A
AT32UC3A0512	512	65536	N/A
AT32UC3A1128	128	65536	N/A
AT32UC3A1256	256	65536	N/A
AT32UC3A1512	512	65536	N/A
AT32UC3A3128	128	65536	N/A
AT32UC3A3128S	128	65536	N/A
AT32UC3A3256	256	65536	N/A
AT32UC3A3256S	256	65536	N/A
AT32UC3A364	64	65536	N/A
AT32UC3A364S	256	65536	N/A
AT32UC3B0128	128	32768	N/A
AT32UC3B0256	256	32768	N/A
AT32UC3B0512	512	98304	N/A
AT32UC3B064	64	16384	N/A
AT32UC3B1128	128	32768	N/A
AT32UC3B1256	256	32768	N/A

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

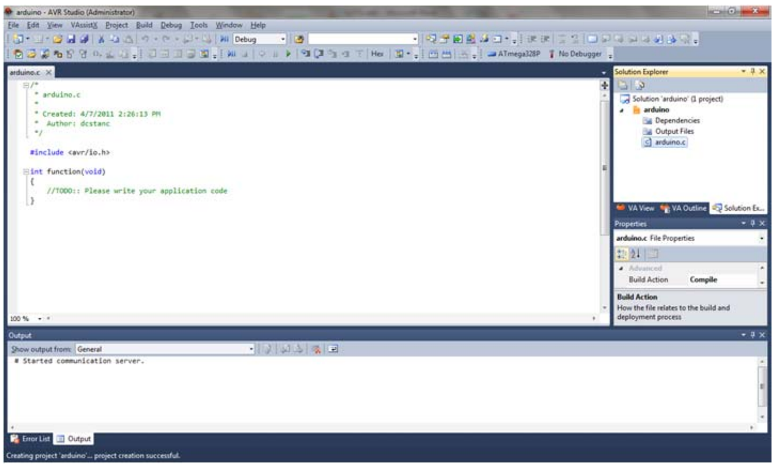


Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (byte
ATmega324P	32	2048	1024
ATmega324PA	32	2048	1024
ATmega325	32	2048	1024
ATmega3250	32	2048	1024
ATmega3250A	32	2048	1024
ATmega3250P	32	2048	1024
ATmega3250PA	32	2048	1024
ATmega325A	32	2048	1024
ATmega325P	32	2048	1024
ATmega325PA	32	2048	1024
ATmega328	32	2048	1024
ATmega328P	32	2048	1024
ATmega329	32	2048	1024
ATmega3290	32	2048	1024
ATmega3290A	32	2048	1024
ATmega3290P	32	2048	1024
ATmega329A	32	2048	1024
ATmega329P	32	2048	1024

Device Name: ATmega328P
Speed: 0
Vcc: 2.7/5.5
Family: megaAVR
Datasheets

AVR Studio Supported Tools
AVR Dragon
AVRISP mkII
AVR ONE!
JTAGICE3
JTAGICE mkII
AVR Simulator
STK600

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



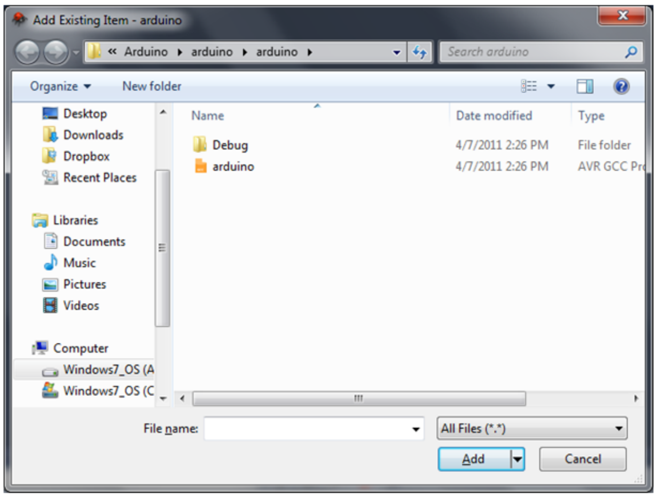
```
/*
 * arduino.c
 *
 * Created: 4/7/2011 2:26:13 PM
 *  Author: dcstanc
 */

#include <avr/io.h>

int function(void)
{
    //TODO:: Please write your application code
}
```

```
* Started communication server.
```

Creating project 'arduino'... project creation successful.

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:

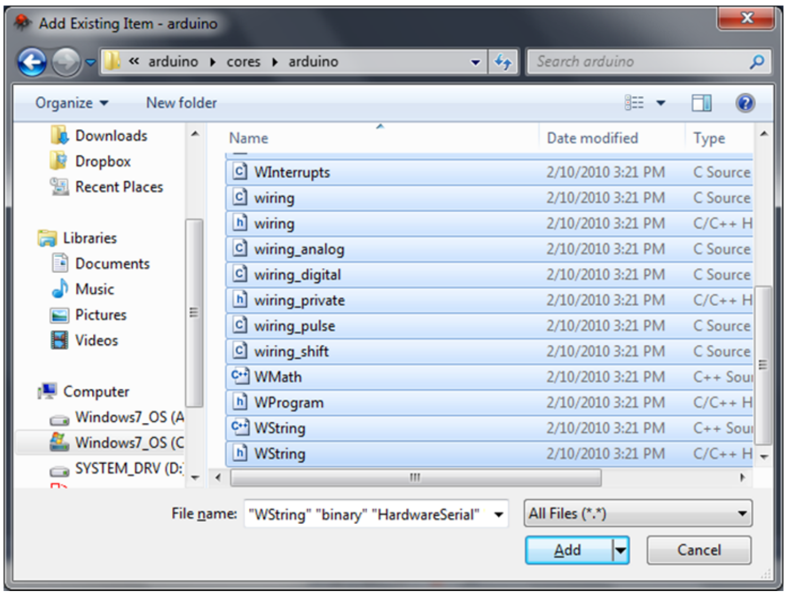


Name	Date modified	Type
Debug	4/7/2011 2:26 PM	File folder
arduino	4/7/2011 2:26 PM	AVR GCC Pr

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-0021, then the full path is:

C:\arduino-0021\hardware\arduino\cores\arduino

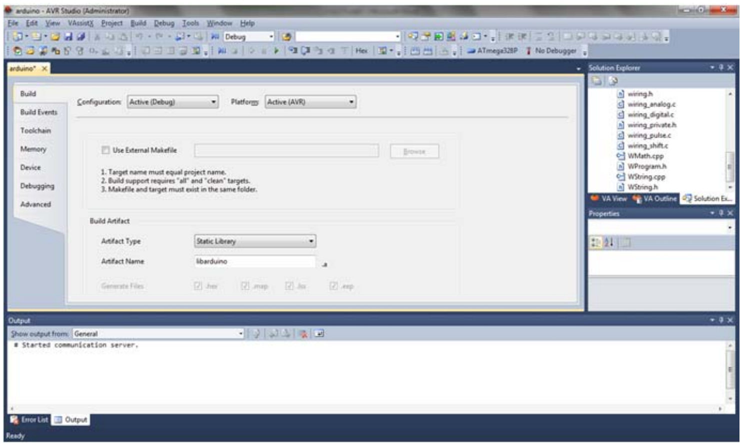
Select all the files and add click Add.



Name	Date modified	Type
WInterrupts	2/10/2010 3:21 PM	C Source
wiring	2/10/2010 3:21 PM	C Source
wiring	2/10/2010 3:21 PM	C/C++ H
wiring_analog	2/10/2010 3:21 PM	C Source
wiring_digital	2/10/2010 3:21 PM	C Source
wiring_private	2/10/2010 3:21 PM	C/C++ H
wiring_pulse	2/10/2010 3:21 PM	C Source
wiring_shift	2/10/2010 3:21 PM	C Source
WMath	2/10/2010 3:21 PM	C++ Sour
WProgram	2/10/2010 3:21 PM	C/C++ H
WString	2/10/2010 3:21 PM	C++ Sour
WString	2/10/2010 3:21 PM	C/C++ H

File name: "WString" "binary" "HardwareSerial"

vi. You now need to set up the Include directories in your project. To do this, click on Project->Properties, which brings up this dialog box:



Configuration: Active (Debug)   Platform: Active (AVR)

Use External Makefile

1. Target name must equal project name.
2. Build support requires "all" and "clean" targets.
3. Makefile and target must exist in the same folder.

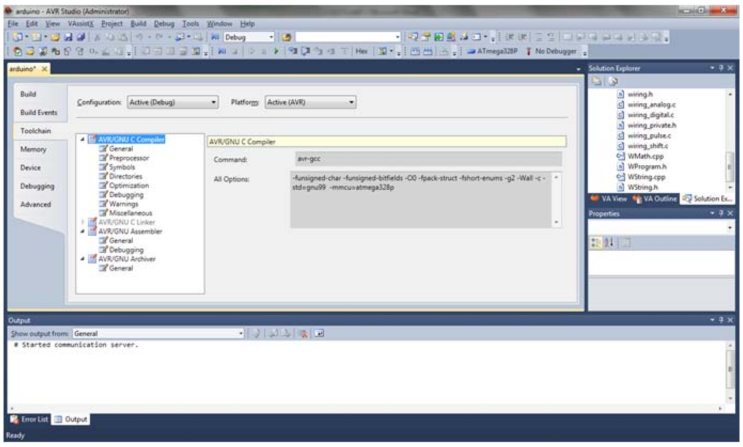
Artifact Type: Static Library

Artifact Name: libarduino

Show output from: General

```
# Started communication server.
```

Click on the "Toolchain" tab on the left, giving you:



Configuration: Active (Debug)   Platform: Active (AVR)

AVR/GNU C Compiler

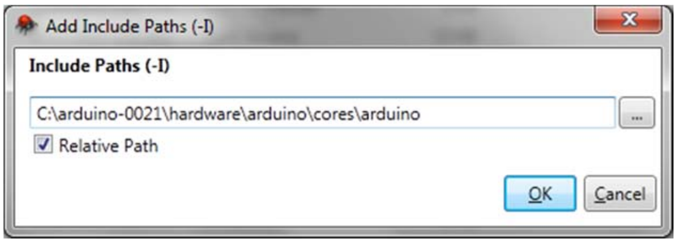
Command: avr-gcc

All Options: -funsigned-char -funsigned-bitfields -O0 -fpack-struct -fshort-enums -g2 -Wall -c -std=gnu99 -mmcu=atmega328p

Show output from: General

```
# Started communication server.
```

Now click on "Directories" under "AVR/GNU C Compiler", and a textbox labeled "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:

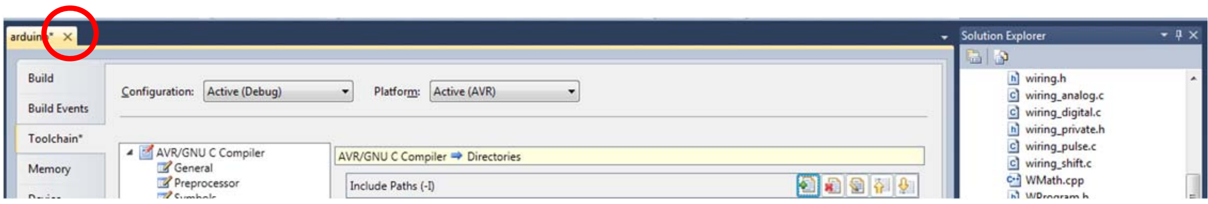


C:\arduino-0021\hardware\arduino\cores\arduino

Relative Path [✓]

Click OK.

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



Configuration: Active (Debug)   Platform: Active (AVR)

AVR/GNU C Compiler ➡ Directories

Include Paths (-I)

viii. 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”.

ix. 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

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.

5. <u>Creating a New Compiler Project</u>

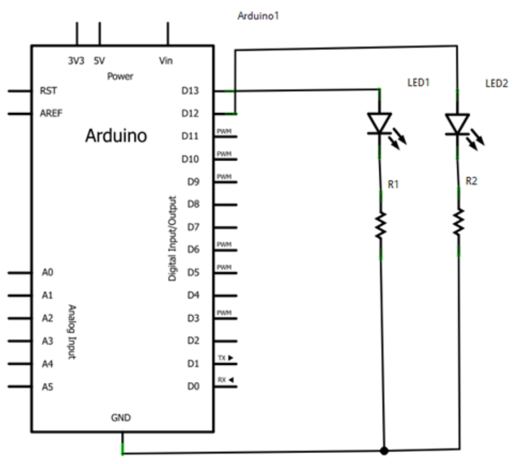
We will now create our own version of the Blinky program that you saw earlier, except that this time we will build a circuit with two LEDs (one red and one green), and alternately flash the two LEDs.

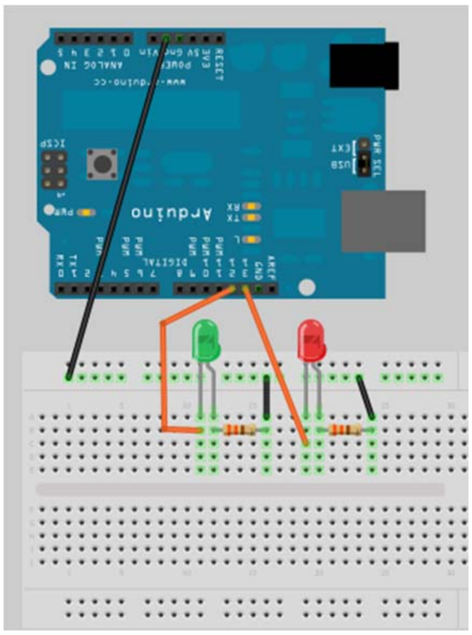
5.1 Building the Circuit

The assemble this circuit you will need:

Component	Quantity
LED	1 red 1 green (or yellow)
Resistor 330 ohm (body has orange-orange-brown stripes)	2
Wires	3 black 2 orange

The circuit diagram is shown below, followed by the layout on the breadboard.



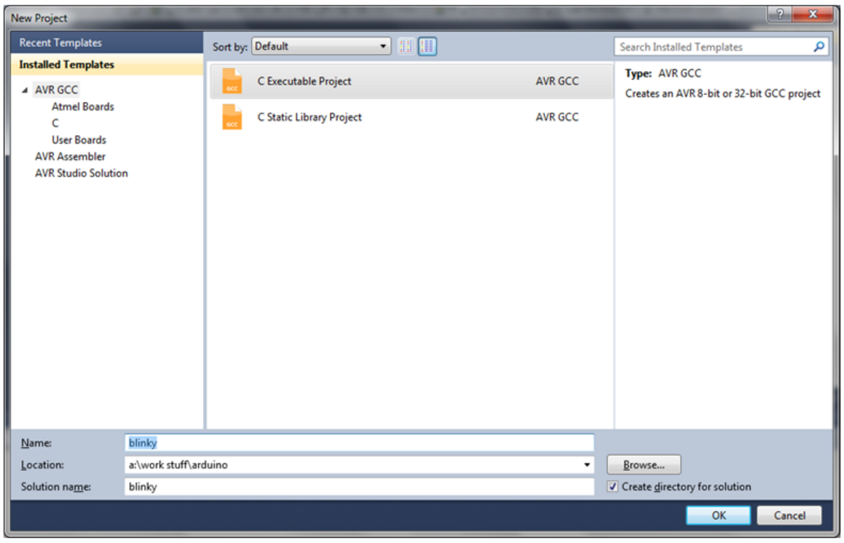


The positive (longer) legs of the red and green (or yellow) LEDs are connected to digital pins 13 and 12 respsectively on the Arduino, while the negative (shorter) legs are each connected to a 330 ohm resistor (orange-orange-brown). The other leg of the resistor is connected to GND via the common ground line on the breadboard. The resistors are important to prevent burning the LEDs.

DO NOT CUT ANY OF THE WIRES YOU ARE GIVEN!

5.2 Creating the Project in AS5

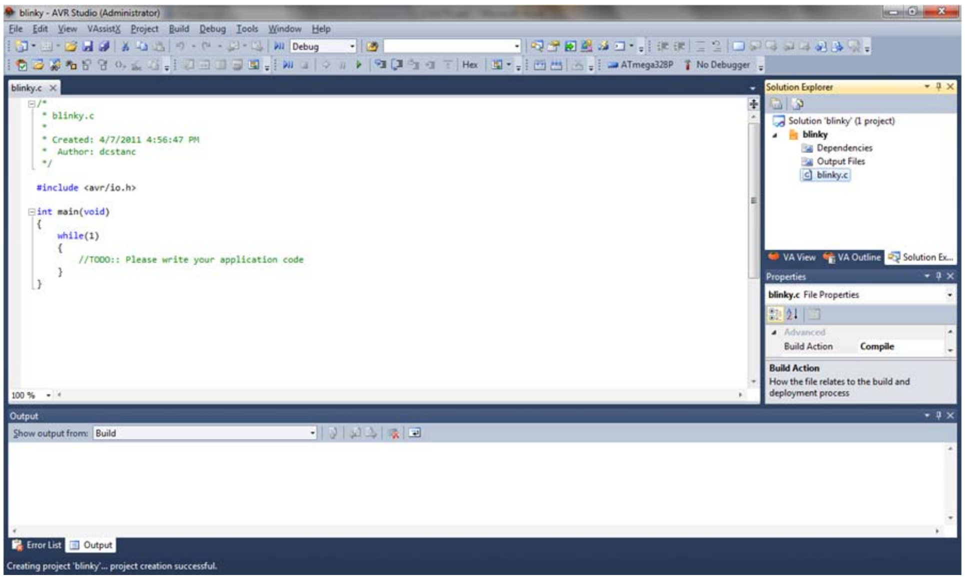
Close any open projects by clicking “File->Close Solution” in AVR Studio 5, then click “File->New->Project”. This time choose “C Executable Project”, and enter “blinky” in the Name field. You can choose any directory for your Location.



Type: AVR GCC
Creates an AVR 8-bit or 32-bit GCC project

Name: blinky
Location: a:\work stuff\arduino
Solution name: blinky
✓ Create directory for solution

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:



```
/*
 * blinky.c
 *
 * Created: 4/7/2011 4:56:47 PM
 *  Author: dcstanc
 */

#include <avr/io.h>

int main(void)
{
    while(1)
    {
        //TODO:: Please write your application code
    }
}
```

Build Action: Compile

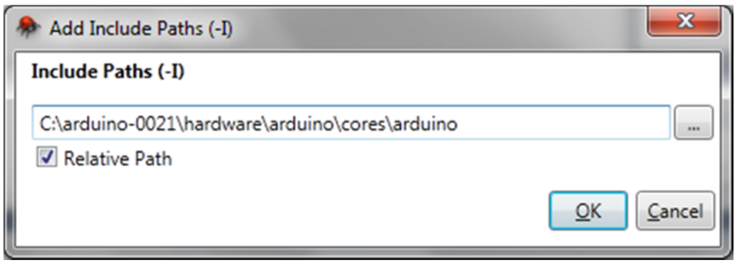
Build Action
How the file relates to the build and deployment process

Creating project 'blinky'... project creation successful.

In our first program we will use the Arduino library to access the hardware. In subsequent labs you will replace these library calls with actual “bare-metal” code to directly access the hardware.

5.3 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-021\hardware\arduino\cores\arduino”, your own particular path may be slightly different, but it is always “<Arduino root directory>\hardware\arduino\cores\arduino”.


Add Include Paths (-I)

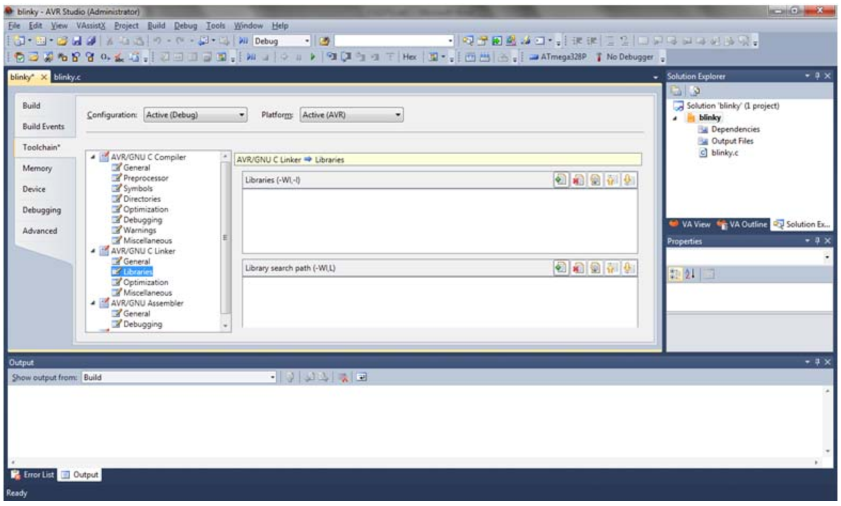
Include Paths (-I)

C:\arduino-0021\hardware\arduino\cores\arduino

☑ Relative Path

Click OK.

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


blinky - AVR Studio (Administrator)

Configuration: Active (Debug)

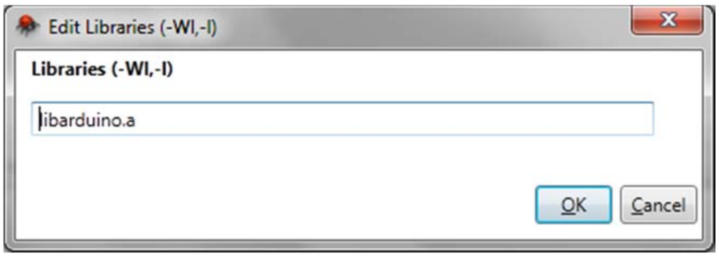
Platform: Active (AVR)

AVR/GNU C Linker ➔ Libraries

Libraries (-Wl,-l)

Library search path (-Wl,L)

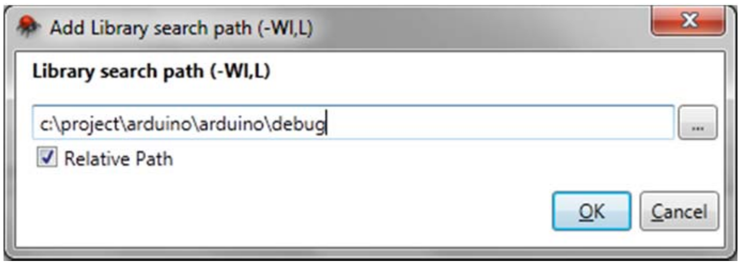
In the “Libraries (-WI, -l)” field, click on the first button (the one with the green +), and enter “libarduino.a”. Click OK.


Edit Libraries (-Wl,-l)

Libraries (-Wl,-l)

libarduino.a

In the “Library Search Path (-WI, 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”:


Add Library search path (-Wl,L)

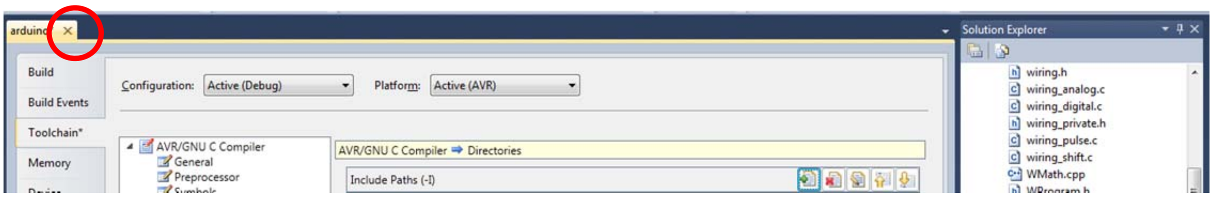
Library search path (-Wl,L)

c:\project\arduino\arduino\debug

☑ Relative Path

Click OK.

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



Configuration: Active (Debug)

Platform: Active (AVR)

AVR/GNU C Compiler ⇒ Directories

Include Paths (-I)

5.4 Keying in The Code and Compiling

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

```
/*
 * blinky.c
 *
 * Created: 4/7/2011 4:56:47 PM
 *  Author: dcstanc
 */

#include <avr/io.h> // Header file to access Atmega328 I/O registers
#include <WProgram.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.

5.5 Keying in The Code and Compiling

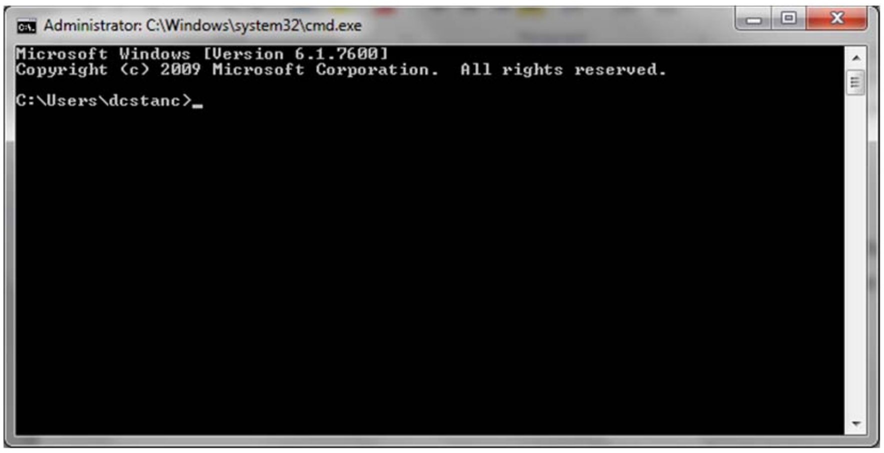
Assuming that the code compiled properly, connect the Arduino board now to your PC/notebook using the USB cable provided. If you have not already done so, follow the instructions in the link below:

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!

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.

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



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\dcstanc>_
```

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-0021”, then the full path to the bin directory is:

“c:\arduino-0021\hardware\tools\avr\bin”

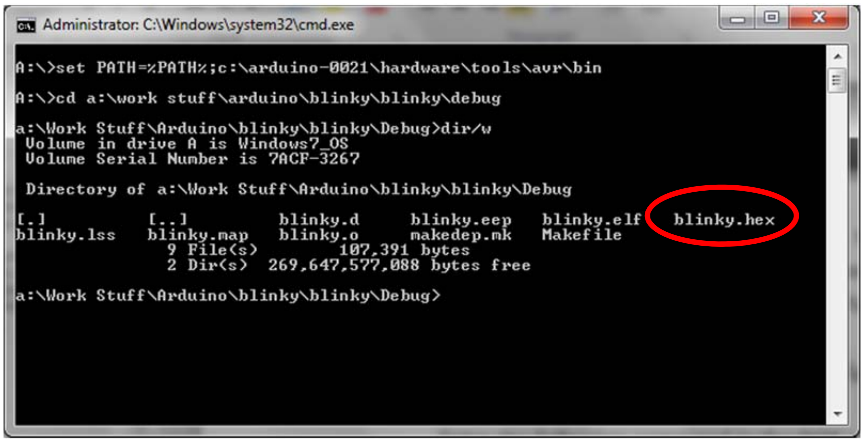
Enter the following command in the DOS window to properly set up the path:

```
set PATH=%PATH%;c:\arduino-0021\hardware\tools\avr\bin
```

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
```

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.



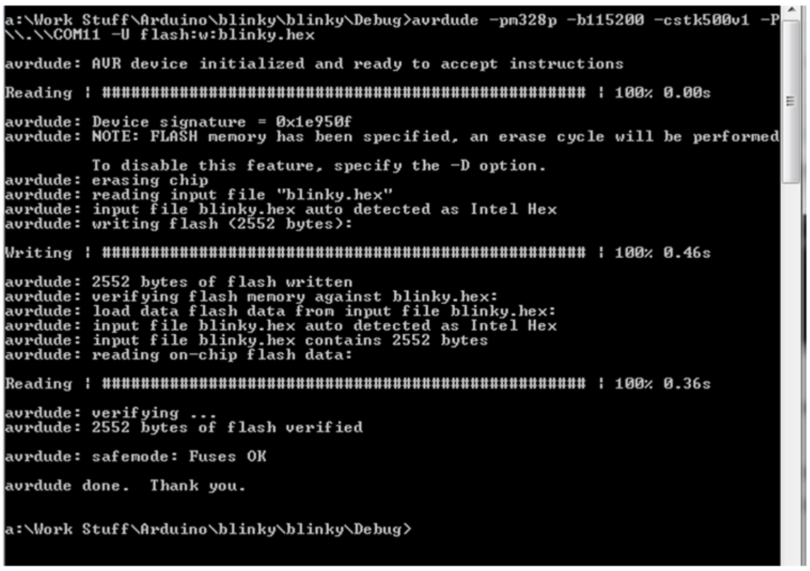
```
A:\>set PATH=%PATH%;c:\arduino-0021\hardware\tools\avr\bin

A:\>cd a:\work stuff\arduino\blinky\blinky\debug

a:\Work Stuff\Arduino\blinky\blinky\Debug>dir/w
 Volume in drive A is Windows7_OS
 Volume Serial Number is 7ACF-3267

 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)        107,391 bytes
               2 Dir(s)  269,647,577,088 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 stk500v1 –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:



```
a:\Work Stuff\Arduino\blinky\blinky\Debug>avrdude -pm328p -b115200 -cstk500v1 -P
\\.\COM11 -U flash:w:blinky.hex

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed

         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "blinky.hex"
avrdude: input file blinky.hex auto detected as Intel Hex
avrdude: writing flash (2552 bytes):

Writing | ################################################## | 100% 0.46s

avrdude: 2552 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 2552 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.36s

avrdude: verifying ...
avrdude: 2552 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.


a:\Work Stuff\Arduino\blinky\blinky\Debug>
```

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 stk500v1	Using the STK500 version 1 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.

6. Debugging

The AVR 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
}
```

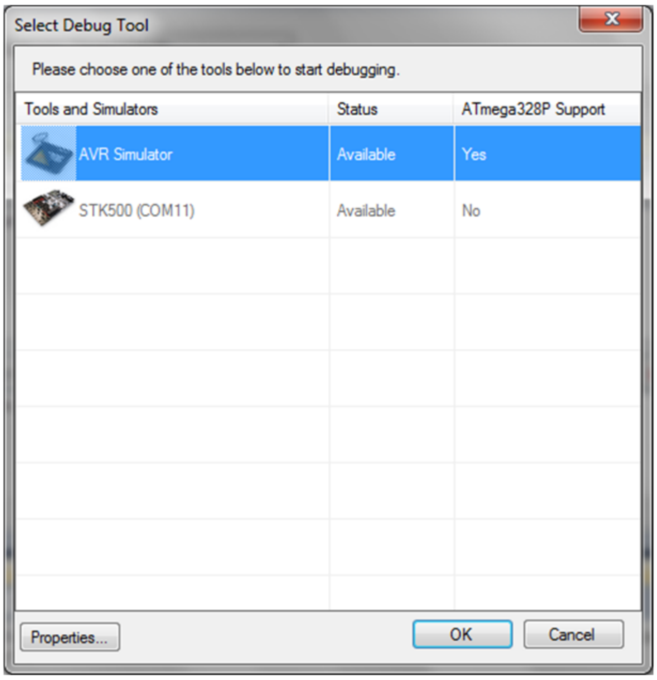
In addition add the following function:

```
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:

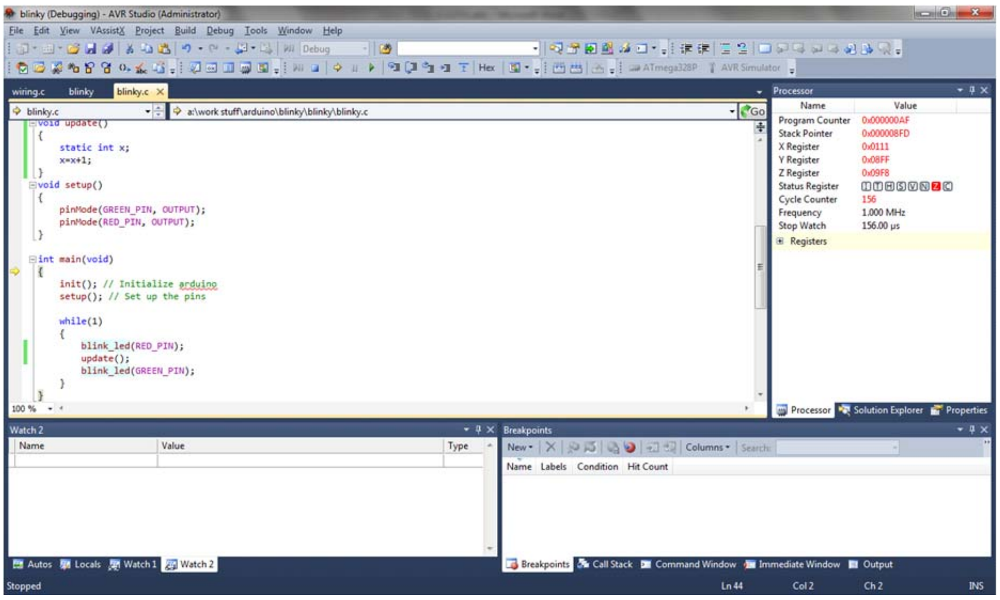

Select Debug Tool

Please choose one of the tools below to start debugging.

Tools and Simulators	Status	ATmega328P Support
AVR Simulator	Available	Yes
STK500 (COM11)	Available	No

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.



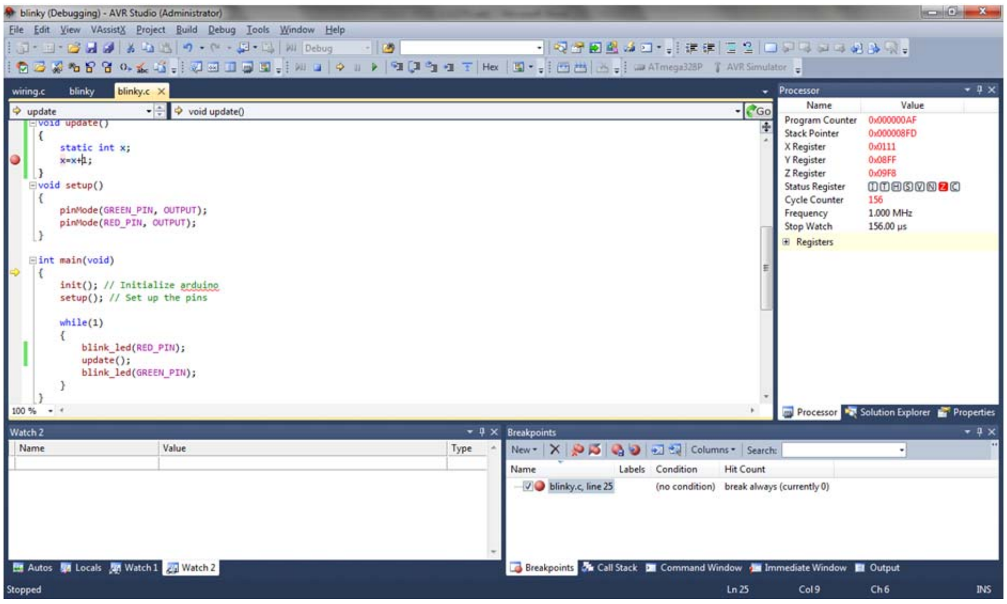
```
void update()
{
    static int x;
    x=x+1;
}
void setup()
{
    pinMode(GREEN_PIN, OUTPUT);
    pinMode(RED_PIN, OUTPUT);
}

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

    while(1)
    {
        blink_led(RED_PIN);
        update();
        blink_led(GREEN_PIN);
    }
}
```

Name	Value
Program Counter	0x000000AF
Stack Pointer	0x000008FD
X Register	0x0111
Y Register	0x08FF
Z Register	0x09F8
Status Register	I T H S V N Z C
Cycle Counter	156
Frequency	1.000 MHz
Stop Watch	156.00 µs
Registers	

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

6.2 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:

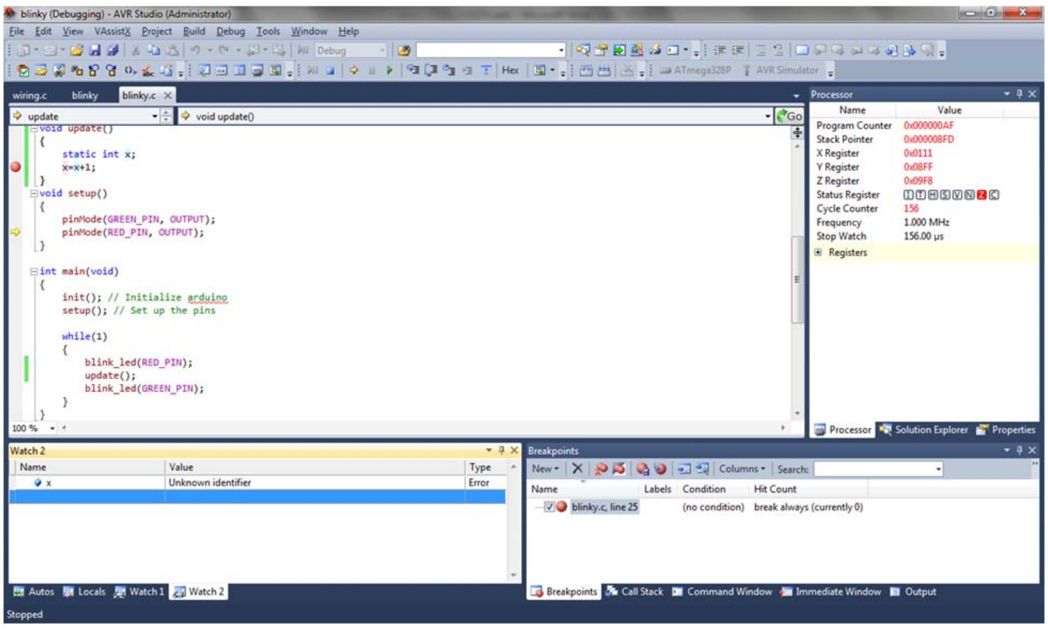


```
void update()
{
    static int x;
    x=x+1;
}
void setup()
{
    pinMode(GREEN_PIN, OUTPUT);
    pinMode(RED_PIN, OUTPUT);
}

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

    while(1)
    {
        blink_led(RED_PIN);
        update();
        blink_led(GREEN_PIN);
    }
}
```

Name	Labels	Condition	Hit Count
blinky.c, line 25		(no condition)	break always (currently 0)

6.3 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:



```
void update()
{
    static int x;
    x=x+1;
}
void setup()
{
    pinMode(GREEN_PIN, OUTPUT);
    pinMode(RED_PIN, OUTPUT);
}

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

    while(1)
    {
        blink_led(RED_PIN);
        update();
        blink_led(GREEN_PIN);
    }
}
```

Processor

Name	Value
Program Counter	0x000000AF
Stack Pointer	0x000008FD
X Register	0x0111
Y Register	0x08FF
Z Register	0x09F8
Status Register	I T H S V N Z C
Cycle Counter	156
Frequency	1.000 MHz
Stop Watch	156.00 µs
Registers	

Watch 2

Name	Value	Type
x	Unknown identifier	Error

Breakpoints

Name	Labels	Condition	Hit Count
blinky.c, line 25		(no condition)	break always (currently 0)

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

6.4 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.

7. Summary

In this lab you have set up the Arduino and AVR Studio environments, and have gotten a feel for assembling a circuit and programming the Sparkfun Inventor Kit.

As this is an introductory lab, we have made use of the functions available in the Arduino libraries to access the digital I/O pins. In subsequent labs as you learn more about accessing the hardware directly (“bare-metal”), we will progressively replace these library routines with our own routines.

The reason for this is simple: while using the Arduino library is convenient, it doesn’t teach you anything about how to program hardware. Writing “bare-metal” programs on the other hand teach you a lot about the Atmega328p, and much of this knowledge is transferrable to other microcontrollers.