



MSOE<sup>®</sup>

# USING THE ARDUINO UNO WITH ECLIPSE

Milwaukee School of Engineering

Created: June 2011

Last Update: 13 September 2013

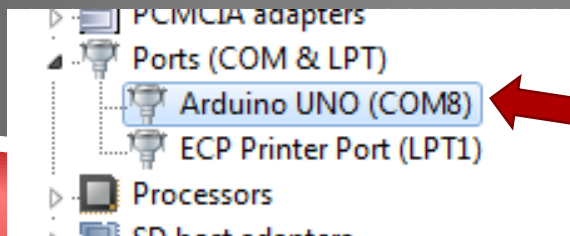
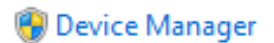
Author: Cory J. Prust, Ph.D.

# ECLIPSE OVERVIEW

- ▶ Eclipse is an open-source software development system
- ▶ Can be used to program the Arduino UNO board in C or C++
- ▶ Allows greater access to and control of microcontroller subsystems

# BEFORE CONTINUING...

- ▶ The Arduino UNO drivers must be installed!
  - ▶ For details, see the Arduino Installation tutorial:  
<https://faculty-web.msoe.edu/prust/arduino>
- ▶ Determine which COM port is assigned to the Arduino UNO board:
  - ▶ In Windows, click on the **Start Menu** and open the **Control Panel**
  - ▶ Choose **System and Security** and then, under **System**, open the **Device Manager**
  - ▶ Under **Ports (COM & LPT)** locate “Arduino UNO”



**IMPORTANT:**  
Note the “COM” port

# OVERVIEW

- ▶ **Step 1:** Installing WinAVR
- ▶ **Step 2:** “Installing” Eclipse
- ▶ **Step 3:** Using Eclipse
- ▶ **Step 4:** Building a Project and Testing the UNO
- ▶ **Step 5:** Creating a New Project
- ▶ **Step 6:** Using the MSOE Support Functions

# INSTALLING WINAVR

- ▶ WinAVR contains the AVR toolchain (plus many other useful tools):
  - ▶ avr-gcc: compiles our C programs
  - ▶ avr-dude: programs the Arduino UNO
- ▶ Eclipse will use WinAVR automatically – we just need to install it!
- ▶ Download the latest release of WinAVR from <https://faculty-web.msoe.edu/prust/arduino>
- ▶ Run the installation using default settings.

**IMPORTANT: You must use the default installation directory!**  
**C:\WinAVR-20100110**

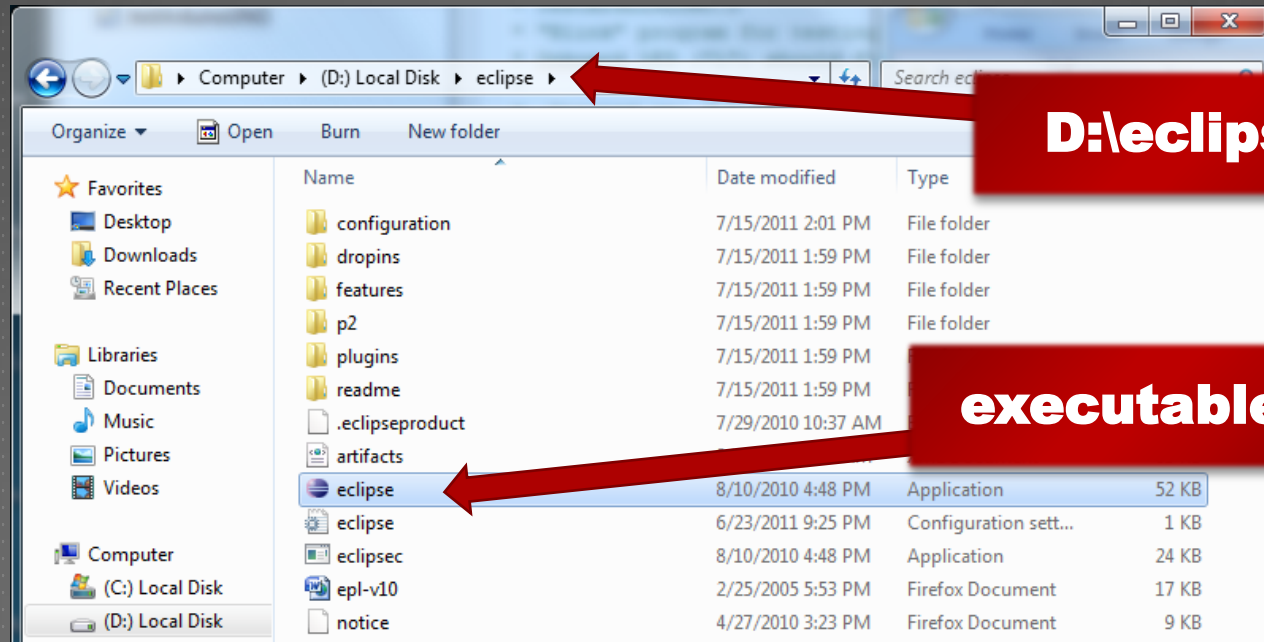
# “INSTALLING” ECLIPSE

- ▶ Download the Eclipse archive (.zip file) from <https://faculty-web.msoe.edu/prust/arduino>
- ▶ Extract the .zip file to your D:\ drive
- ▶ After unzipping, you should see two folders:
  - ▶ D:\eclipse
  - ▶ D:\ARDUINOworkspace

(continued...)

# “INSTALLING” ECLIPSE

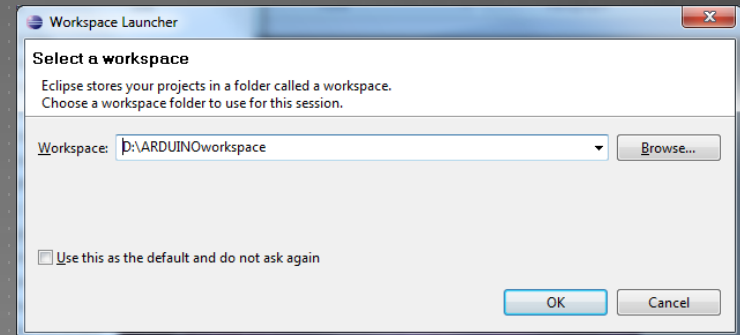
- ▶ The executable is located in D:\eclipse. Double-click to start Eclipse.



- ▶ For quicker access to Eclipse, create a shortcut to the executable.

# USING ECLIPSE

- ▶ You will see the following screen - it defines “D:\ARDUINOworkspace” as the location for your Arduino projects
  - ▶ Select “OK”



- ▶ You will then see the Eclipse workbench

(continued...)



# USING ECLIPSE

## Build Icon

Pressing this button results in a “build” of the active project. The result of a successful build is a “.hex” file which can be uploaded onto the UNO board.

## Upload Icon

Pressing this button sends the program (in the form of a “.hex” file) to the ATmega328p on the UNO board.

## Project Explorer

All projects in the current workspace (ARDUINOworkspace) are shown here. Only the selected project is “active”.

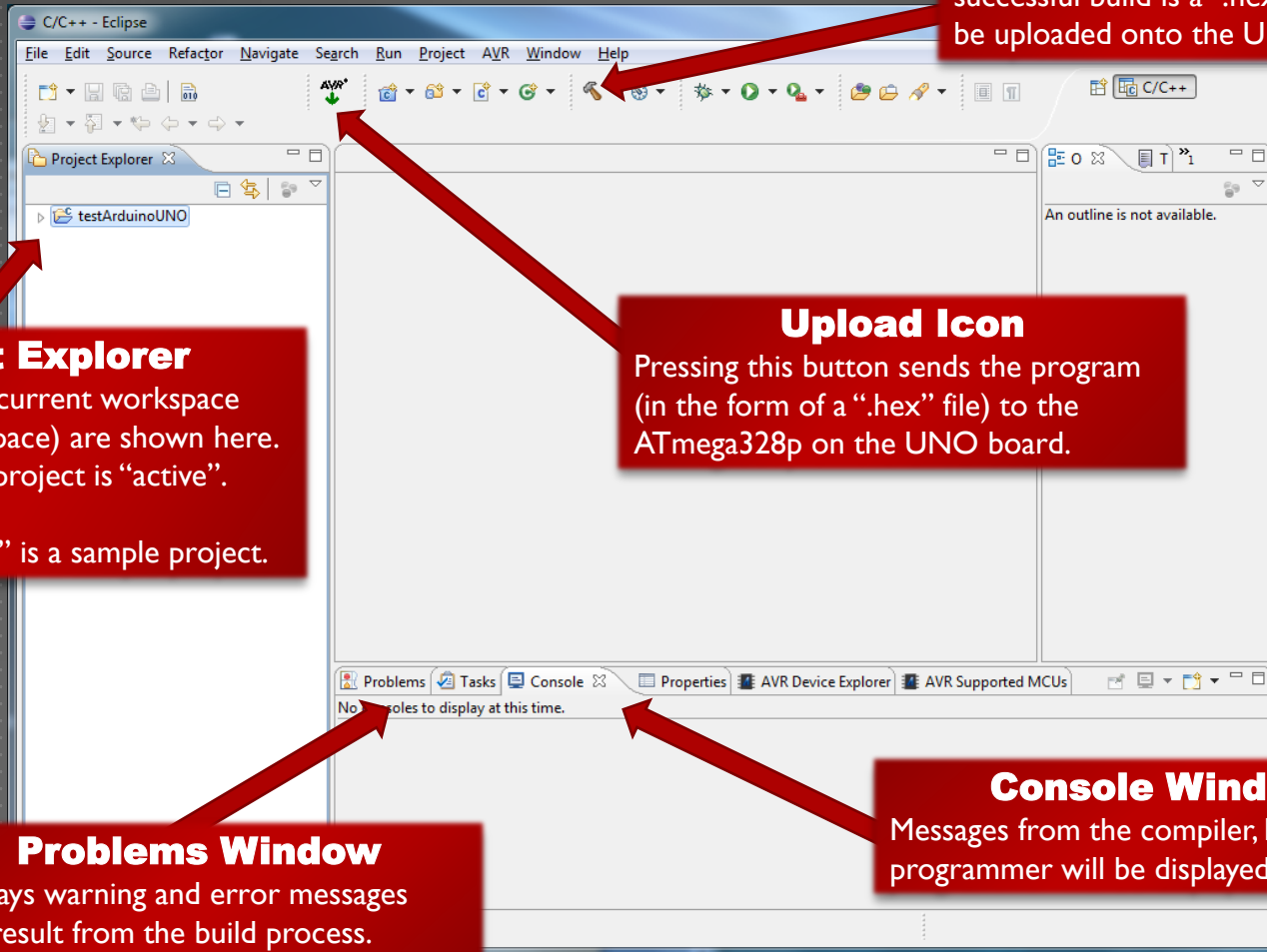
“testArduinoUNO” is a sample project.

## Problems Window

Displays warning and error messages that result from the build process.

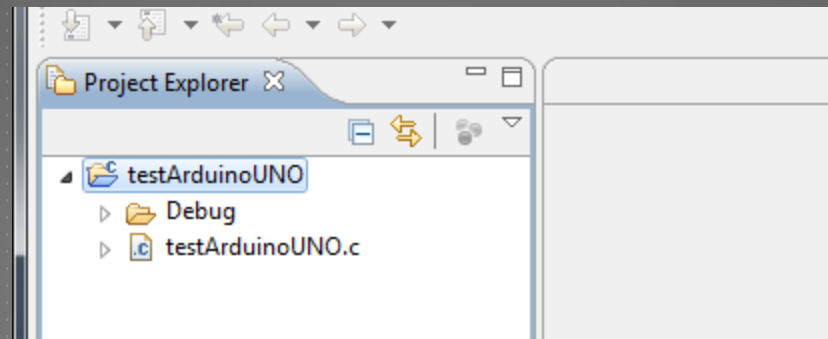
## Console Window

Messages from the compiler, linker, and programmer will be displayed here.



# BUILDING A PROJECT

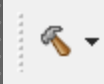
- ▶ The project “testArduinoUNO” has been included as a sample project
- ▶ Expand the project in the Project Explorer. You should see the following:

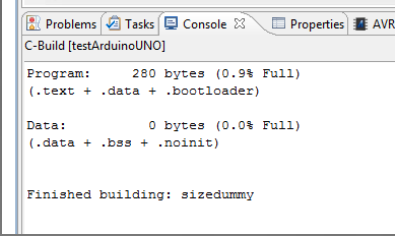


- ▶ “testArduinoUNO.c” is the source code. Double-click to open and examine the code.

(continued...)

# BUILDING A PROJECT

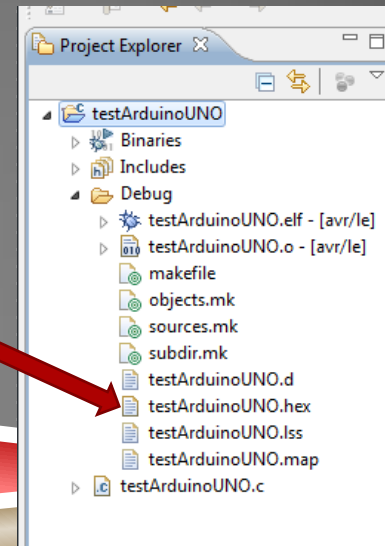
- ▶ To build the project, click the build icon 
  - ▶ The build may take a minute or two
- ▶ Information regarding the build process is shown in the console window.



```
Problems Tasks Console Properties AVR
C-Build [testArduinoUNO]
Program:      280 bytes (0.9% Full)
(.text + .data + .bootloader)
Data:         0 bytes (0.0% Full)
(.data + .bss + .noinit)
Finished building: sizedummy
```

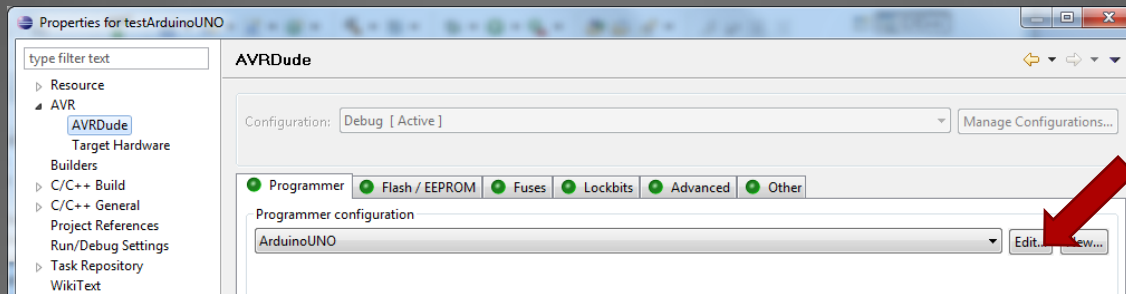
- ▶ The build process generates several files, which can be seen in the Project Explorer window:

**.hex file used to program the ATmega328p microcontroller**

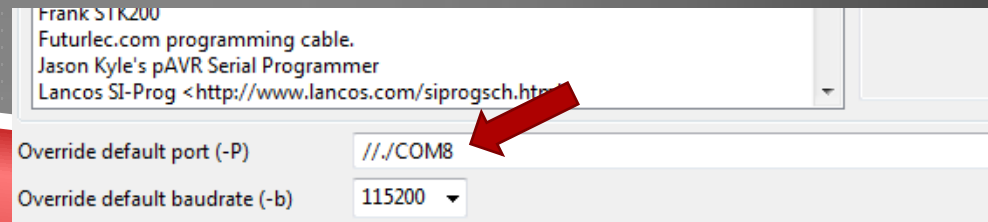


# TESTING THE UNO

- ▶ Eclipse must know which COM port the UNO board is connected to.
- ▶ Select “Project” then “Properties”
- ▶ Expand the “AVR” arrow and select “AVRDude”
  - ▶ The “Programmer configuration” should read “ArduinoUNO”. Click the “Edit” button



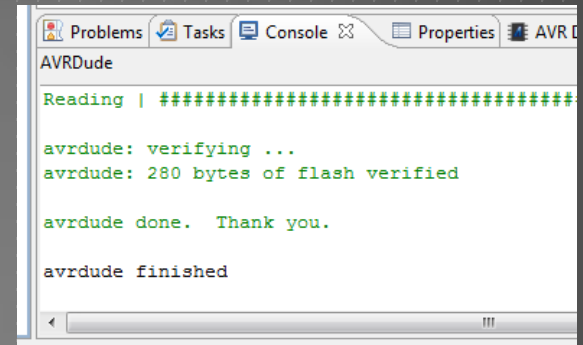
- ▶ Change the “Override default port (-P)” setting to the correct COM port, and select “OK” (twice)
  - ▶ For example, the correct setting for COM5 would be: `///COM5`



# TESTING THE UNO

- ▶ Connect your Arduino UNO board to your laptop
- ▶ Press the upload icon 
- ▶ Information regarding the upload process is shown in the console window.

- ▶ A successful upload results in the following message:



```
AVR Dede
Reading | #####
avrduede: verifying ...
avrduede: 280 bytes of flash verified

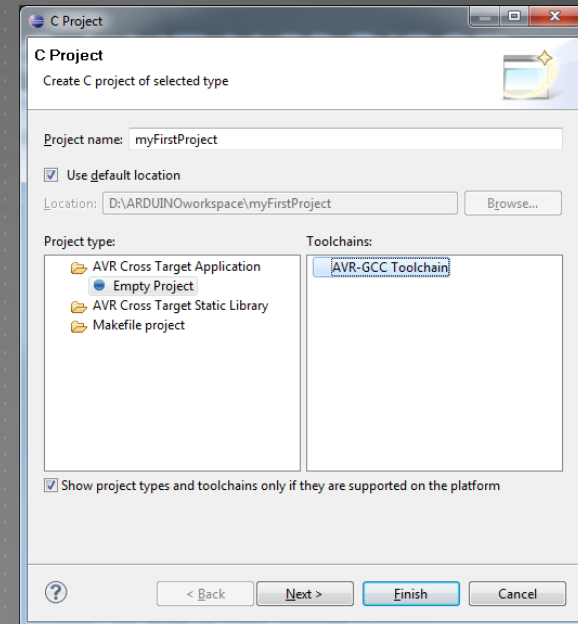
avrduede done. Thank you.

avrduede finished
```

- ▶ The **yellow** LED (marked “L”) should be blinking!

# CREATING A NEW PROJECT

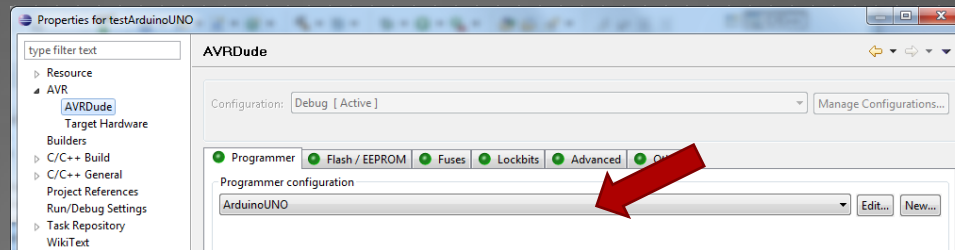
- ▶ You will need to create new projects in Eclipse, for example, when you begin a new laboratory assignment.
  - ▶ **IMPORTANT:** Keep all of your projects in “ARDUINOworkspace”!
- ▶ Select “File – New – C Project”
- ▶ Give the project a descriptive name
  - ▶ Do not use spaces or any special characters
    - “my First Project” **BAD**
    - “myFirstProject” **GOOD**
- ▶ Project type: “Empty Project”
- ▶ Toolchain: “AVR-GCC Toolchain”
- ▶ Select “Next”



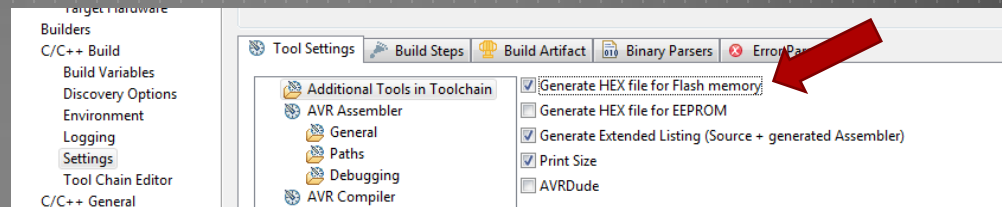
(continued...)

# CREATING A NEW PROJECT

- ▶ Click the “Advanced Settings” button
- ▶ Expand the “AVR” arrow and select “AVRDude”
- ▶ Under “Programmer configuration”, use the drop-down box to select “ArduinoUNO”



- ▶ Expand the “C/C++ Build” arrow and select “Settings”
- ▶ Under the “Tool Settings” tab, check the “Generate HEX file for FLASH memory” option

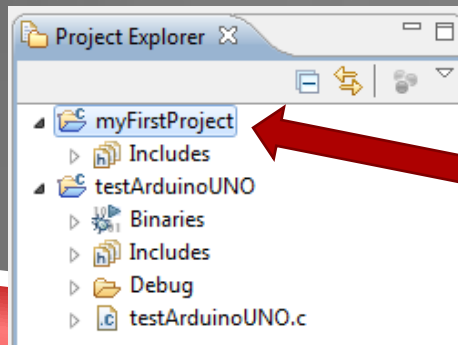
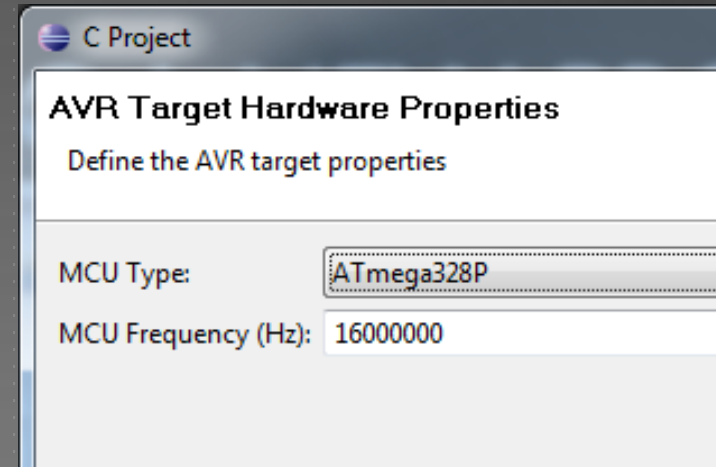


- ▶ Click “OK”, and “Next”

(continued...)

# CREATING A NEW PROJECT

- ▶ Set the MCU Type to “ATmega328p”
- ▶ Set the MCU Frequency (Hz) to “16000000”
  - ▶ The Arduino UNO board has a 16MHz crystal which provides the CPU clock to the ATmega328p
- ▶ Choose “Finish”
- ▶ You will see your new project in the Project Explorer



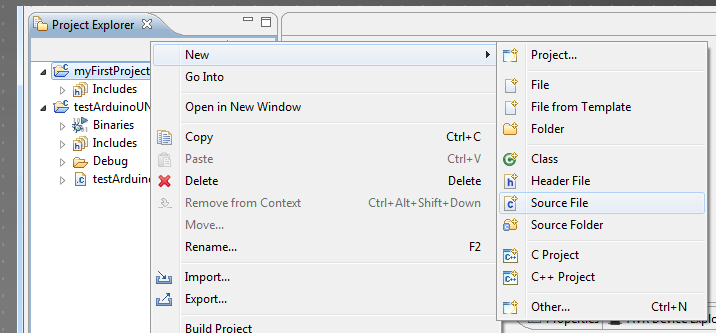
**Note: The selected project is “active”**

(continued...)

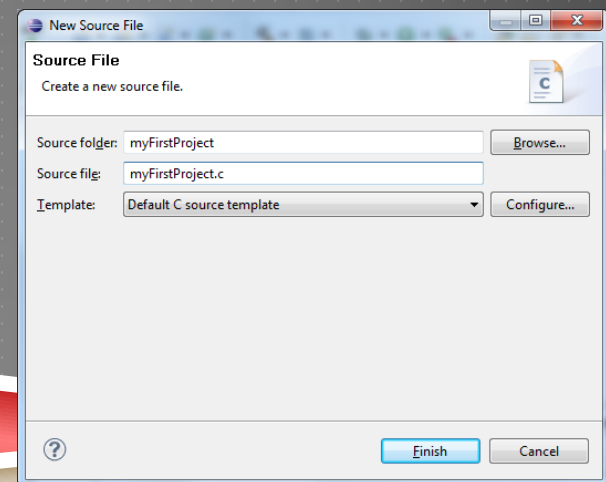


# CREATING A NEW PROJECT

- ▶ We now need to add a source file (.c)
- ▶ From within the Project Explorer, right-click on your project and select “New – Source File”



- ▶ Give your source file a descriptive name
  - ▶ e.g., “myFirstProject.c”
  - ▶ **IMPORTANT:** the file name must end with “.c”
- ▶ Choose “Finish”



# CREATING A NEW PROJECT

- ▶ The source file is now part of your project and will be used during the build process.
- ▶ As a test, copy/paste the source code from the “testArduinoUNO” project into your new project:
  - ▶ Experiment with the delay function to alter the blink rate and pattern
  - ▶ Build the new project
- ▶ Upload the .hex file to the UNO board

## **IMPORTANT:**

Only the “active” project will be built!

Make a project “active” by **selecting** it in **Project Explorer**

# SUMMARY: USING ECLIPSE

**CREATE NEW PROJECT**



**CONFIGURE SETTINGS**



**ADD "C" Source File**



**EDIT CODE**  
in C Source File

**Development  
Cycle**

**TEST and DEBUG**

**BUILD**  
Executable Hex File

**UPLOAD**  
Hex File to UNO



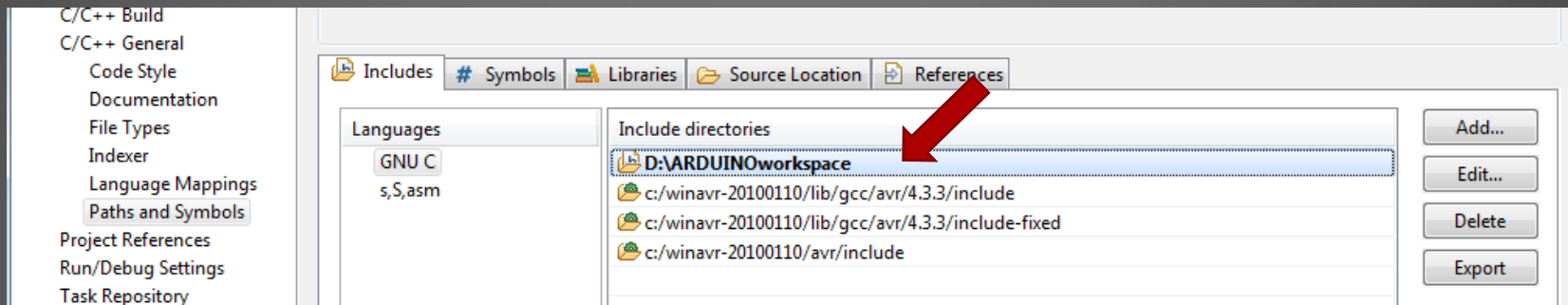
# USING THE “MSOE” SUPPORT FUNCTIONS

- ▶ Within “D:\ARDUINOworkspace” is a directory named “MSOE” containing a variety of functions:
  - ▶ delay.c – time delay functions
  - ▶ lcd.c – LCD control functions
  - ▶ bit.c – general purpose functions
- ▶ These functions can easily be used within Eclipse:
  - ▶ Must tell Eclipse their location on the filesystem
  - ▶ Must “include” the files in our source code
  - ▶ Must properly “call” the functions within our program
- ▶ Next, we will modify our program to allow precise timing of the “blink”

(continued...)

# USING THE “MSOE” SUPPORT FUNCTIONS

- ▶ Step 1: Tell Eclipse the location of the MSOE support functions
  - ▶ Select “Project” then “Properties”
  - ▶ Expand the “C/C++ General” arrow and select “Paths and Symbols”
  - ▶ In the “Includes” tab, “Add” an Include directory “D:\ARDUINOworkspace”

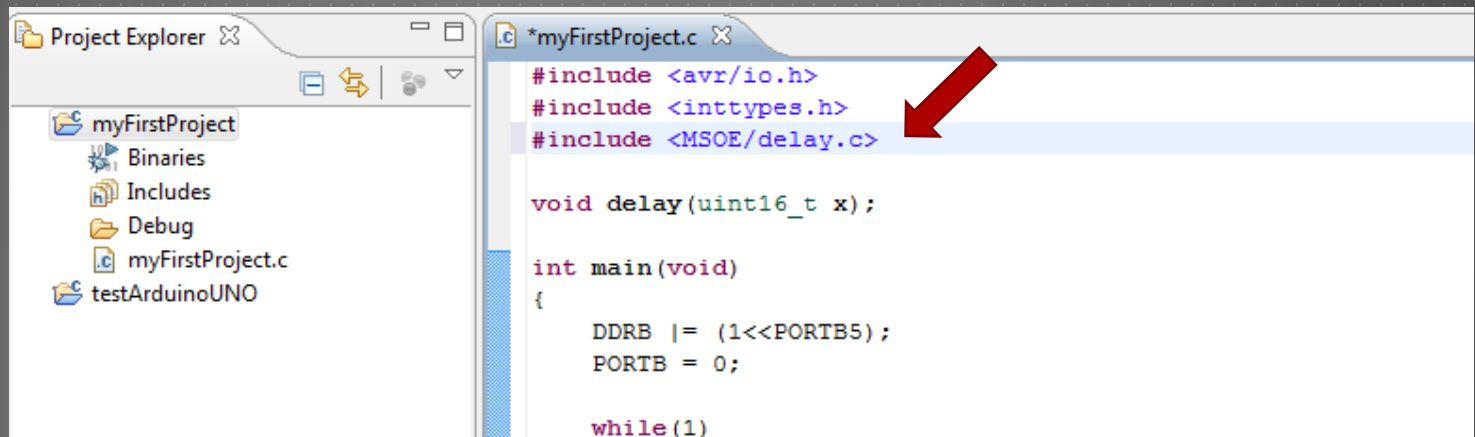


- ▶ Select “OK”
- ▶ You may be prompted to “rebuild”. Choose “Yes”.

(continued...)

# USING THE “MSOE” SUPPORT FUNCTIONS

- ▶ Step 2: Include the MSOE support functions in the source code
  - ▶ We will use the “delay\_ms()” function to control the timing
  - ▶ The function source code must be “included” within our program
    - ▶ The function itself is contained in a file called “delay.c”
- ▶ Add the following line of code:



The screenshot shows an IDE window with a Project Explorer on the left and a code editor on the right. The Project Explorer shows a project named 'myFirstProject' with subfolders for 'Binaries', 'Includes', 'Debug', and 'myFirstProject.c', and a file named 'testArduinoUNO'. The code editor shows the following C code:

```
#include <avr/io.h>
#include <inttypes.h>
#include <MSOE/delay.c>

void delay(uint16_t x);

int main(void)
{
    DDRB |= (1<<PORTB5);
    PORTB = 0;

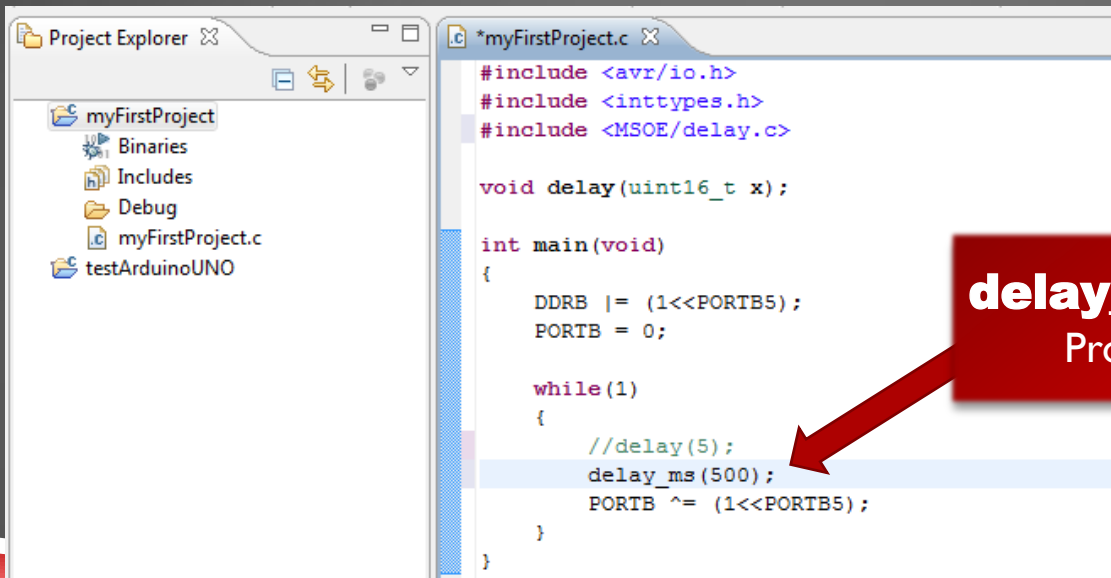
    while(1)
```

A red arrow points to the line `#include <MSOE/delay.c>`, which is highlighted in blue.

(continued...)

# USING THE “MSOE” SUPPORT FUNCTIONS

- ▶ Step 3: Call the function within our program
  - ▶ The “`delay_ms()`” function accepts an unsigned 16-bit integer parameter that controls the time delay (in milliseconds)
  - ▶ Make the following modifications:



The screenshot shows an IDE window with a Project Explorer on the left and a code editor on the right. The Project Explorer shows a project named 'myFirstProject' with subfolders for 'Binaries', 'Includes', 'Debug', and files 'myFirstProject.c' and 'testArduinoUNO'. The code editor shows the following C code:

```
#include <avr/io.h>
#include <inttypes.h>
#include <MSOE/delay.c>

void delay(uint16_t x);

int main(void)
{
    DDRB |= (1<<PORTB5);
    PORTB = 0;

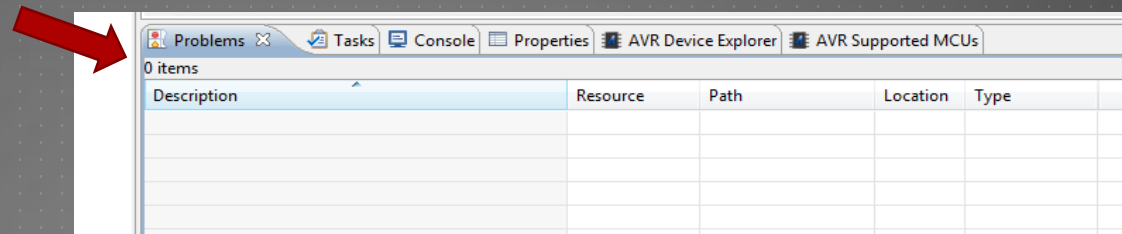
    while(1)
    {
        //delay(5);
        delay_ms(500);
        PORTB ^= (1<<PORTB5);
    }
}
```

**delay\_ms(500);**  
Produces a 500 millisecond delay

(continued...)

# USING THE “MSOE” SUPPORT FUNCTIONS

- ▶ Save your program and build it!
- ▶ The “Problems” tab will alert you to any errors or warnings that resulted from the build process:
  - ▶ “0 items” means a successful build!

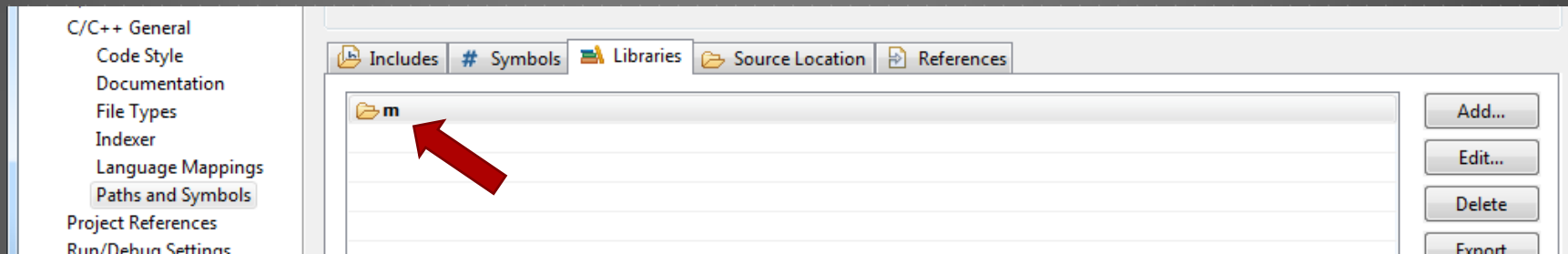


- ▶ Upload the .hex file to the UNO board
  - ▶ Experiment with the “delay\_ms()” function to alter the blink rate and pattern



# ONE FINAL NOTE: THE AVR MATH LIBRARY

- ▶ When a program is doing lots of calculations (e.g., floating point, calls to functions such as “sqrt”, “cos”, etc) it is a good idea to use the AVR Math Library
- ▶ The AVR Math Library contains code which has been optimized for use on the AVR microcontrollers – so things run fast and efficiently!
- ▶ To use it:
  - ▶ Select “Project” then “Properties”
  - ▶ Expand the “C/C++ General” arrow and select “Paths and Symbols”
  - ▶ In the “Libraries” tab, “Add” an entry named “m”



- ▶ Select “OK”

# CONGRATULATIONS!!!

- ▶ You now have a fully functioning Arduino UNO development system in Eclipse!
- ▶ Programming the Arduino board in C unlocks the full functionality of the ATmega328p microcontroller!
- ▶ Writing software in C also provides greater flexibility:
  - ▶ For example, your C programs could easily be “built” for a different microcontroller!