

# ECED3204 – Lab #1

---

*STUDENT NAME(s):*\_\_\_\_\_.

*STUDENT NUMBER(s):* *B00*\_\_\_\_\_.

## **Pre-Lab Information**

It is recommended that you read this entire lab ahead of time. Doing so will save you considerable time during the lab.

## **Overall Objective**

This lab has three main objectives:

- Learn about the development environment you will be using in this class.
- Write a simple program in C to blink an LED.
- Write a simple program in ASM to blink an LED.

## Part #1: Introduction to the Development Environment in C

### Objective

- Familiarize yourself with the microprocessor module, power requirements, and breadboard

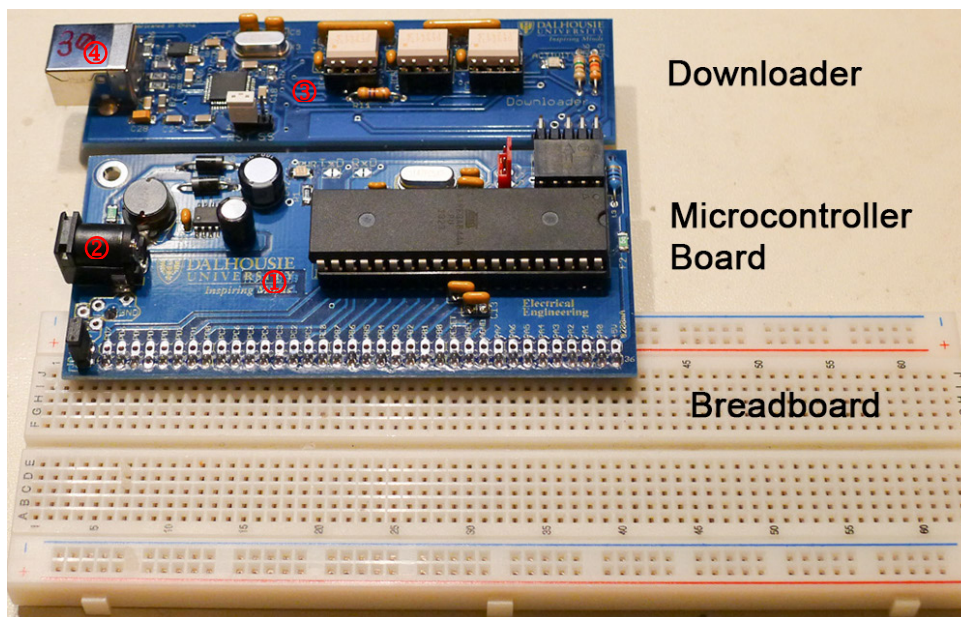
### Required Materials

- Microprocessor Module with Programmer
- Breadboard
- USB Cable
- Power Supply
- Computer with Atmel Studio 6.2 and Programmer Utility installed

### Background

A microcontroller is basically a small computer, capable of executing instructions you program into the device. A variety of low-cost microcontroller boards are available on the market, and the Dalhousie microcontroller board is a version with some special features you will use in later classes. This includes a protection circuit which prevents you from blowing up your computer if you apply an incorrect voltage!

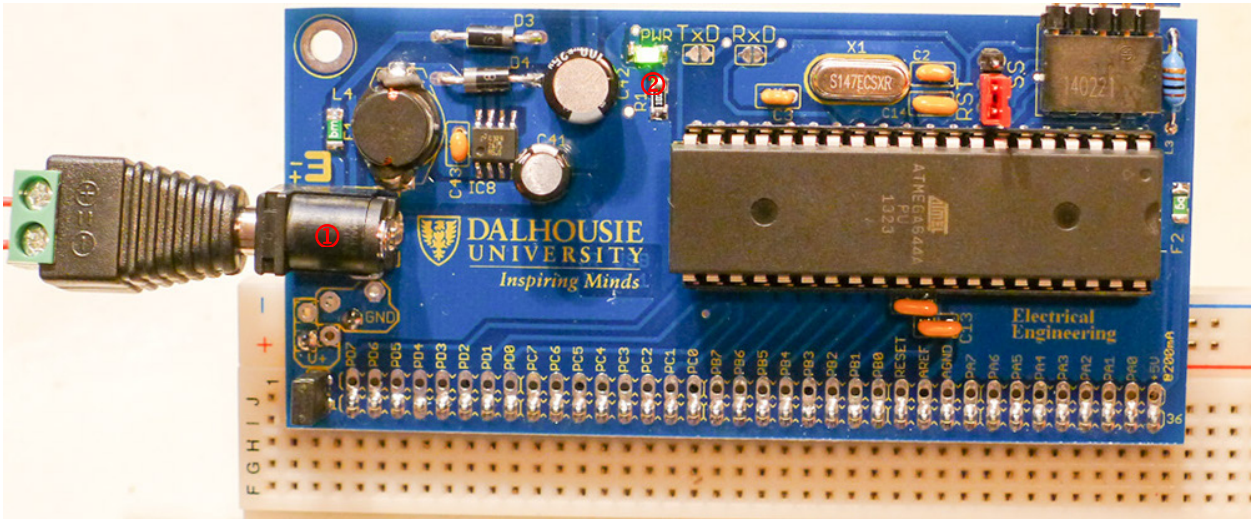
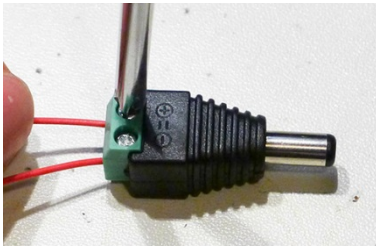
The board is shown in the following photo, which notes the three things of importance: ① the microcontroller board with input power jack ②, the programmer board ③ with USB connection ④.



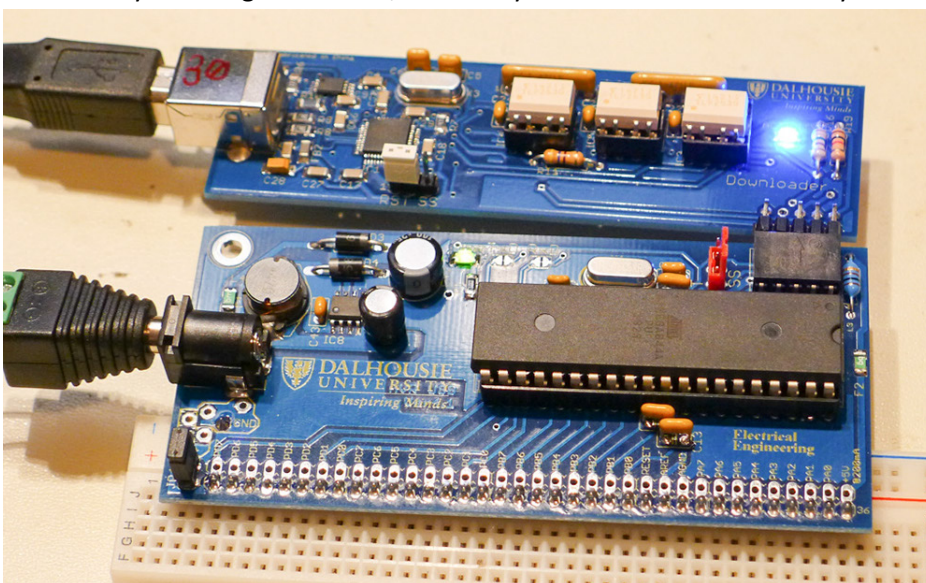
### Procedure

1. Plug your programmer board together (if not already plugged together)
2. Connect a suitable power source to the power jack on the microcontroller board, such as 10 V DC Source ①, and check the LED ② comes on. Note the +/- on the DC Power jack connector is

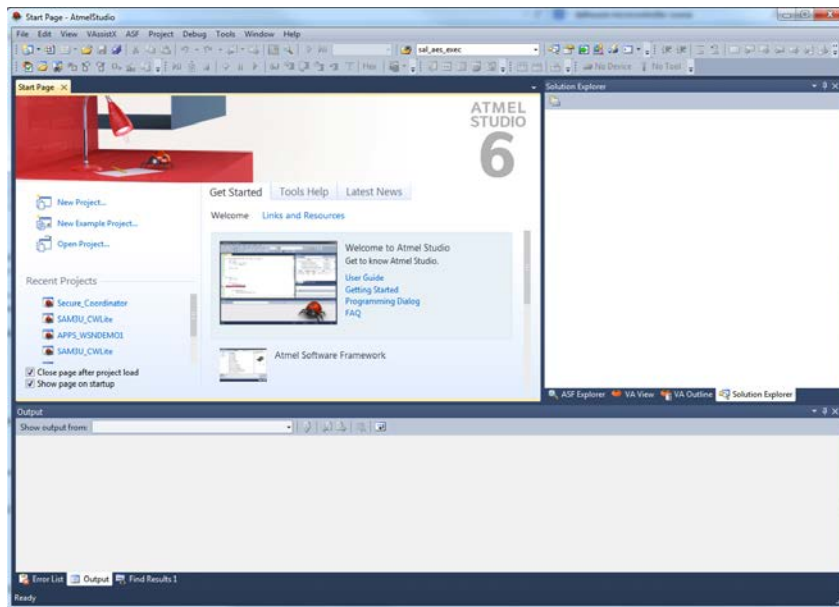
backwards from what is required, but you will not damage the board if connecting the wrong way around. If the LED fails to illuminate, simply try swapping the + and – power supply leads.



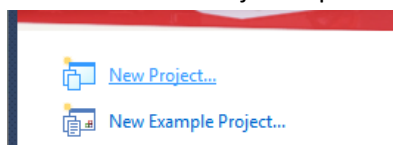
3. Plug the USB-A cable into your computer and programmer board. Depending on jumper settings the 'Activity' LED might come on, but it may not so continue either way!



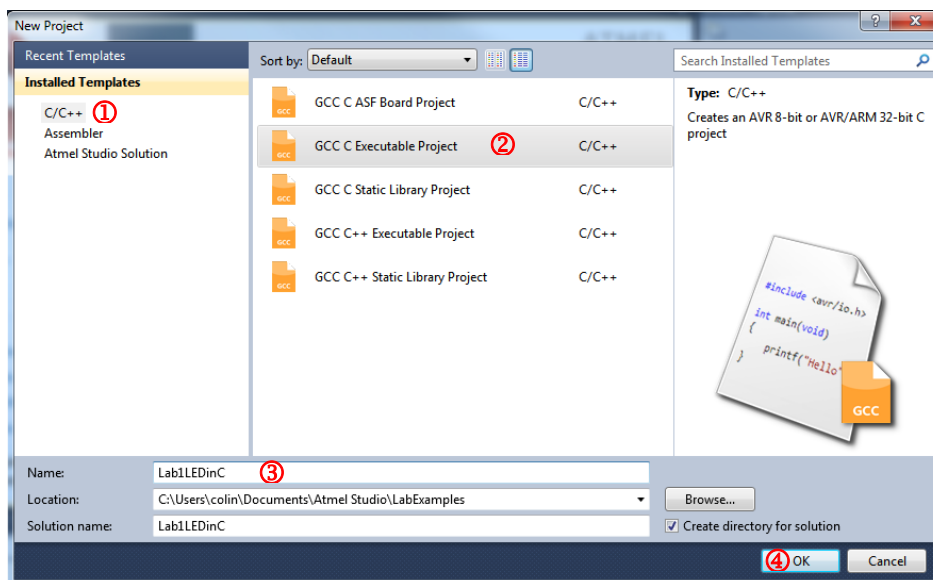
4. Start Atmel Studio 6.2, the main page will look something like this:



5. Select the 'New Project' Option:

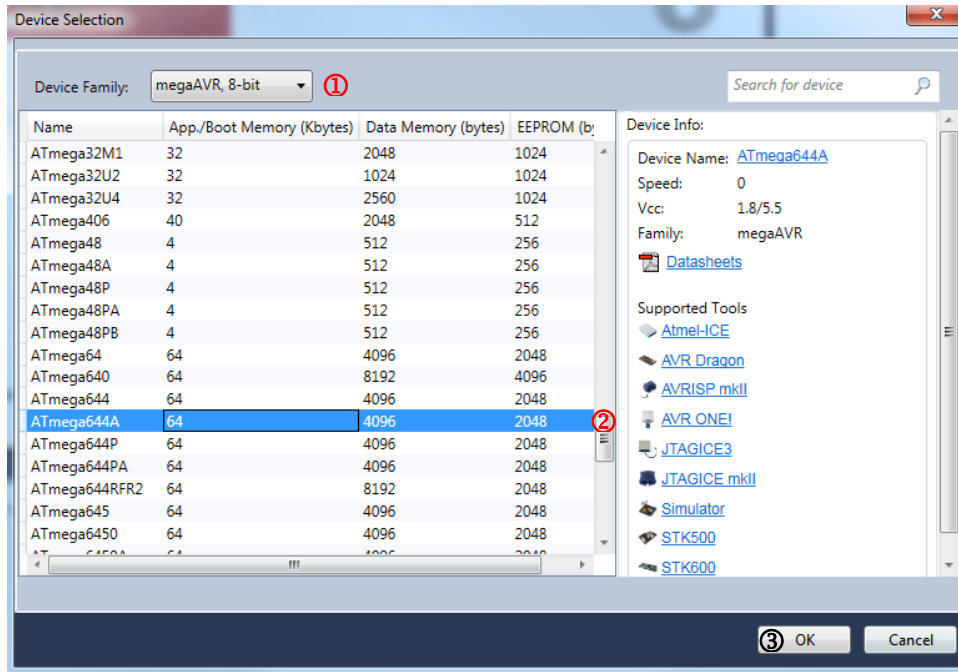


6. On the 'New Project' window, select the **C/C++** Option ①, and the select a **GCC C Executable Project** ②. You should set a suitable project name and location ③, and then hit **OK** ④.

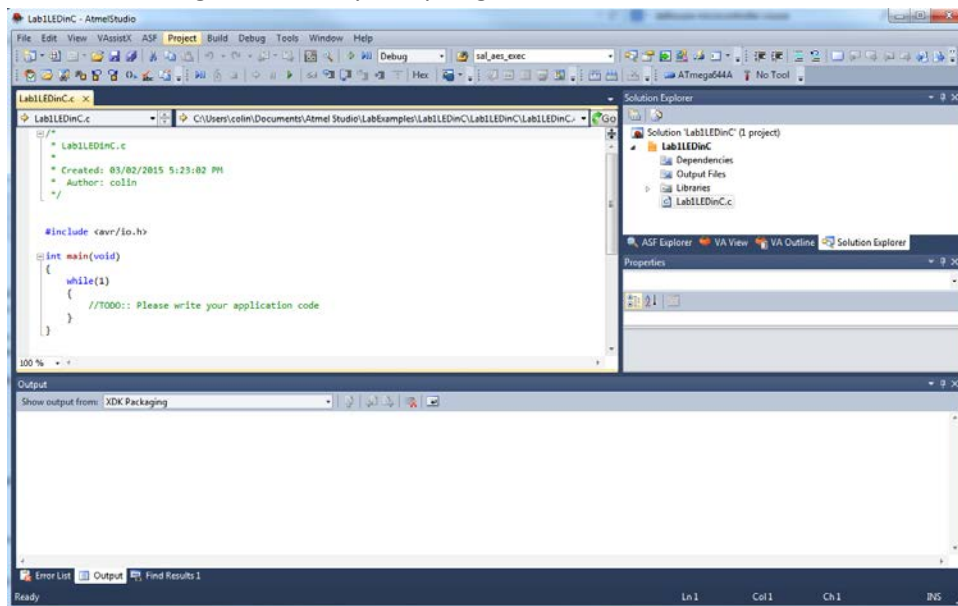




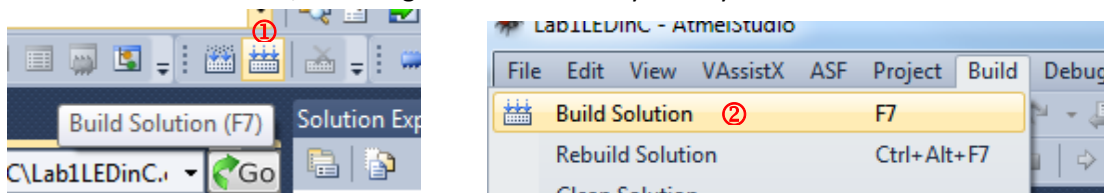
7. On the next page, you need to select the AVR device for the project. Change the Device Family to **megaAVR, 8-bit** ①, and find the **ATmega644A** device ②. Select this device and hit OK ③.



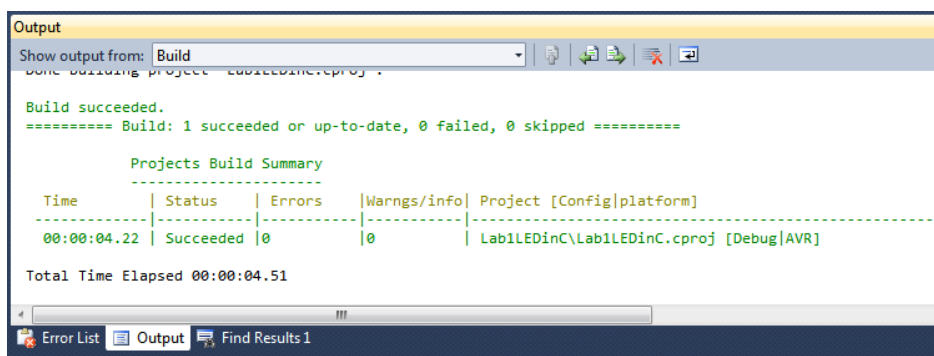
8. At this point – you should have a working project! All you need to do now is write all the code. But before doing that we'll try compiling it.



9. You can compile the code by either selecting the **Build Project** icon ①, Selecting **Build Solution** ② from the **Build Menu**, or hitting the **F7** button on your keyboard.



10. This should work, which if you look at the bottom of the window shows you the build was successful:



11. Now it's time to blink an LED. Write the following code into the main window:

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

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

    while(1)
    {
        PORTB = 0x00;
        _delay_ms(500);
        PORTB = 1<<0;
        _delay_ms(500);
    }
}
```

Here is a copy of the C Code:

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

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

    while(1)
    {
        PORTB = 0x00;
        _delay_ms(500);
        PORTB = 1<<0;
        _delay_ms(500);
    }
}
```

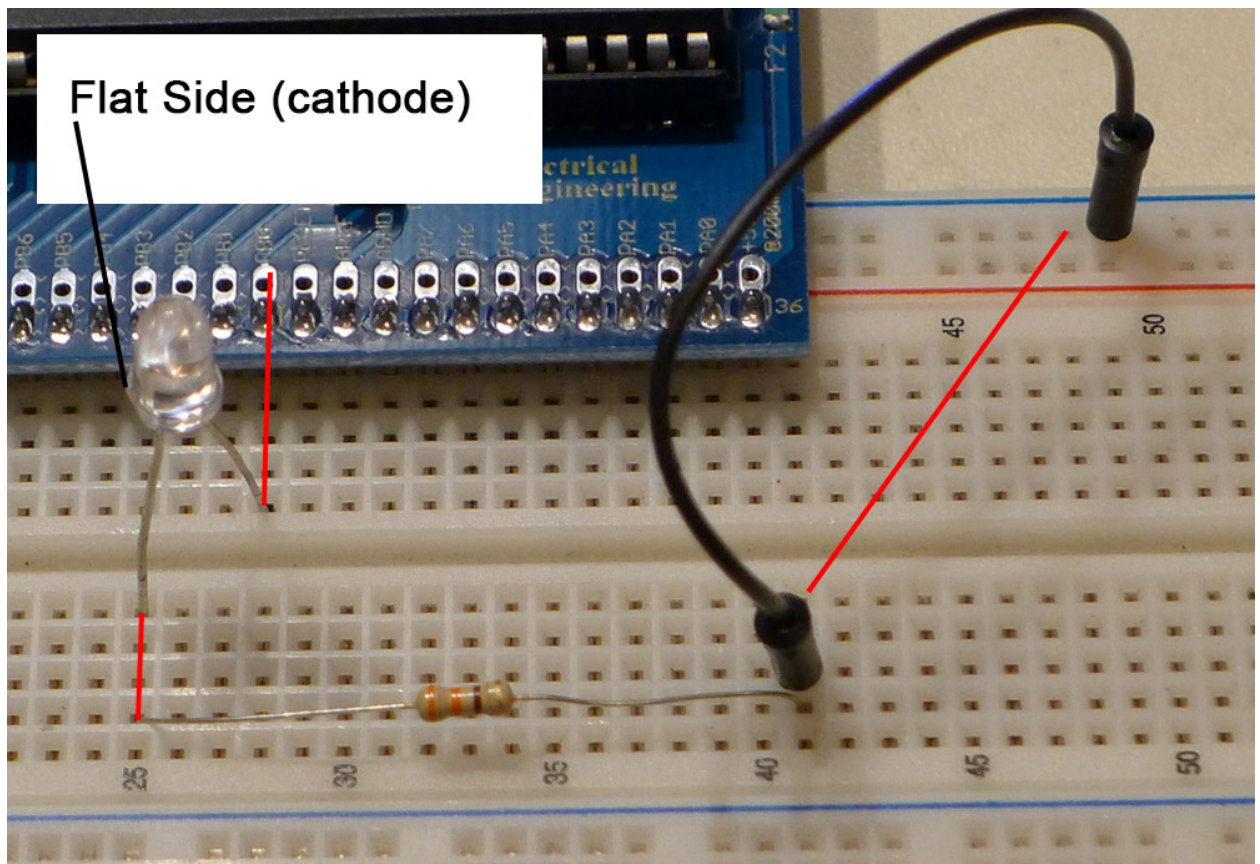
12. Attempt to build the project again. You might see a warning here, but you should still see a build succeeded. If things go wrong check you've entered the code correctly.

```
Build succeeded.
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====

      Projects Build Summary
-----
Time      | Status   | Errors | Warnings/info | Project [Config|platform]
-----
00:00:01.79 | Succeeded | 0      | 1              | Lab1LEDinC\Lab1LEDinC.cproj [Debug|AVR]


Total Time Elapsed 00:00:01.91
```

13. The next step is to setup the hardware. We need to connect an LED and 330-ohm resistor to PORTB, Pin 0 (PB0). Remember that LEDs have a polarity – either look for the flat side to indicate negative, or the longer lead to indicate positive:

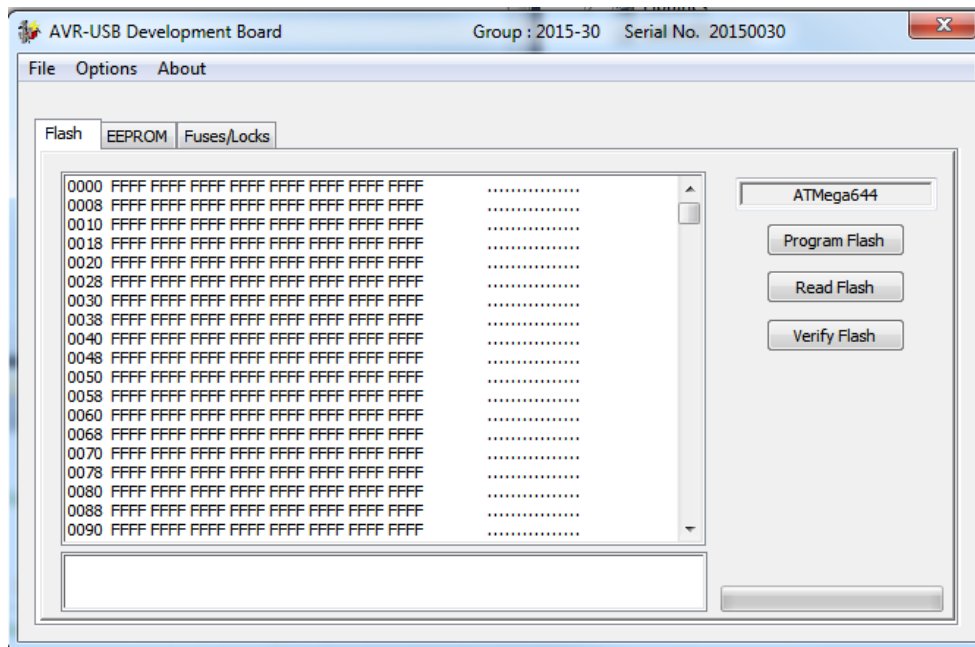


14. You can now program the board. Run the **AVRUSBProg** program which can program the device:

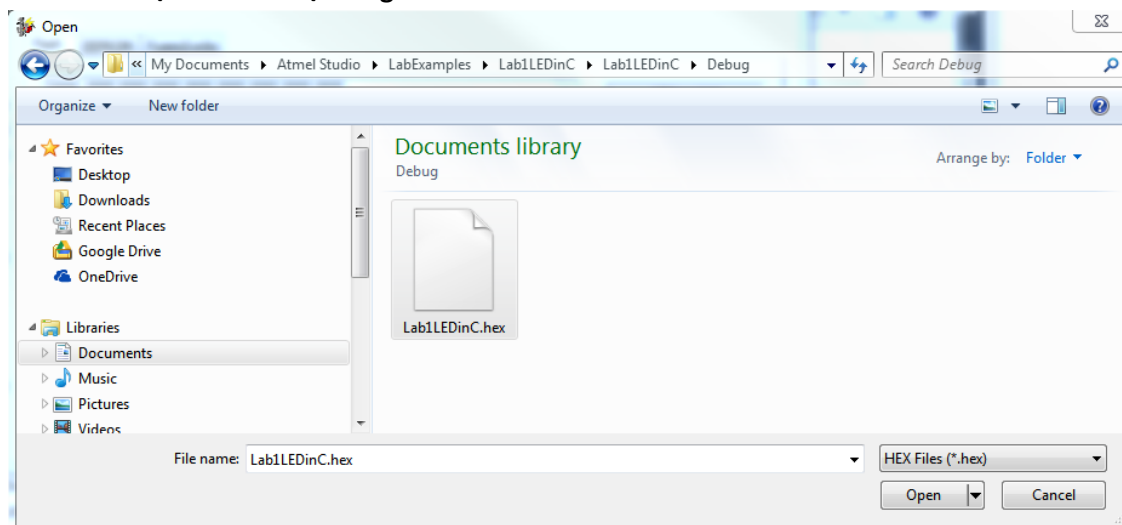
## Programs (1)

 AVRUSBProg.exe

15. The program should connect to the board, with a main window that looks something like this:

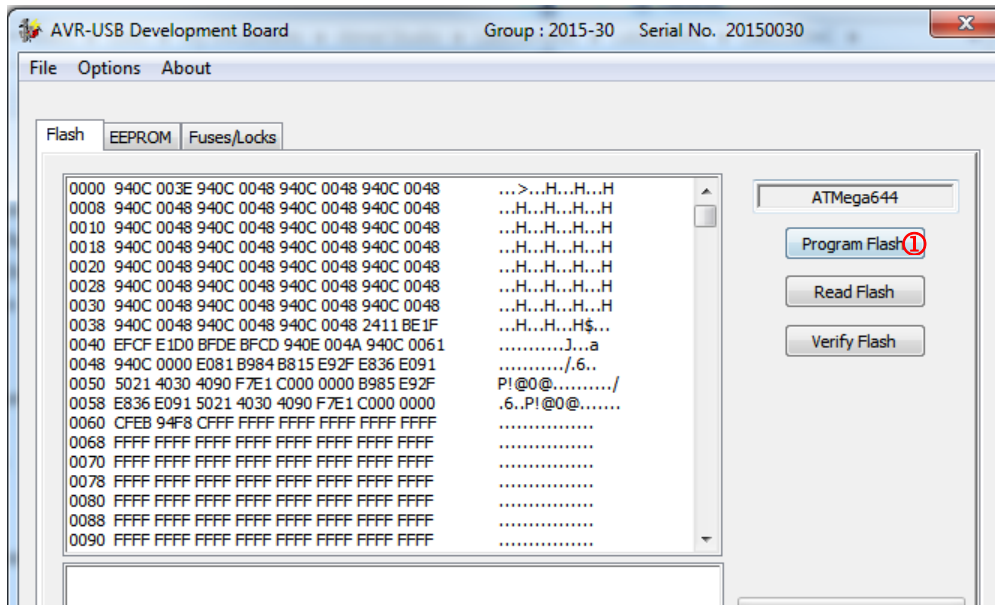


16. You need to find a **.hex** file located in your project directory. Remember where you saved it from way back where you created the project? If not, try saving the project again and see where the directory is located! The **.hex** file will be in a few subdirectories, here it's located in **Lab1LEDinC\Lab1LEDinC\Debug**.

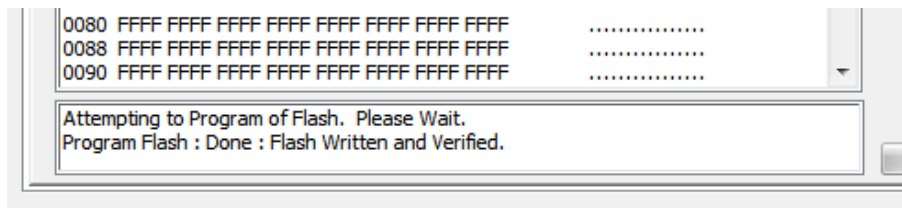




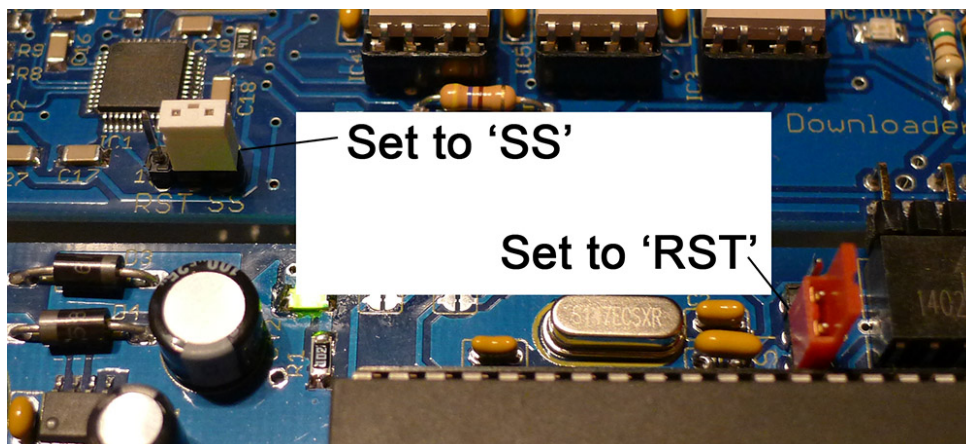
17. Select the file and hit **Open**. The main window should look something like the following, and hit the **Program Flash** button ①:



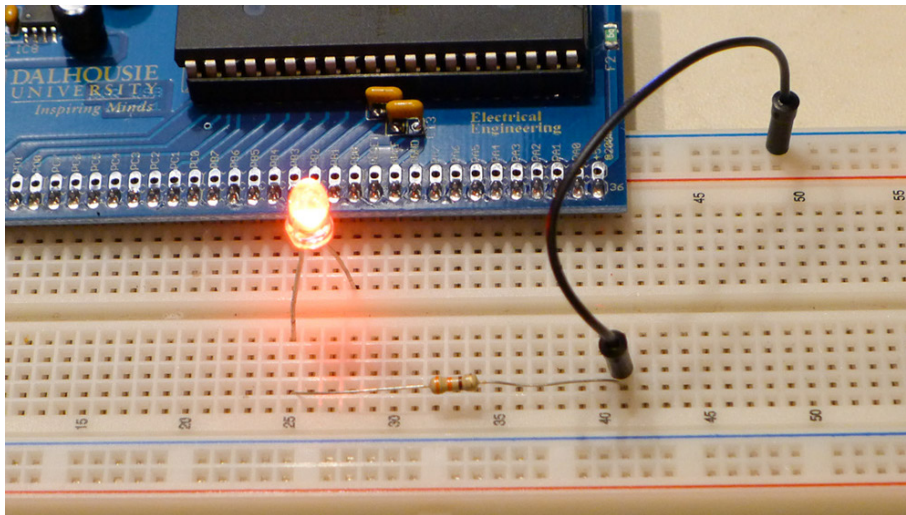
18. Hopefully you'll see a note that the program was successful:



19. Finally – we need to let the chip run the program. To do this either unplug the 'Downloader' board, or even better, move the 'RST' jumper over to the 'S.S.' position on the downloader. The jumper on the microcontroller should be in the 'RST' position.. This will cause your code to automatically run. You will want to remove this later when controlling a robot, otherwise your robot might move away!



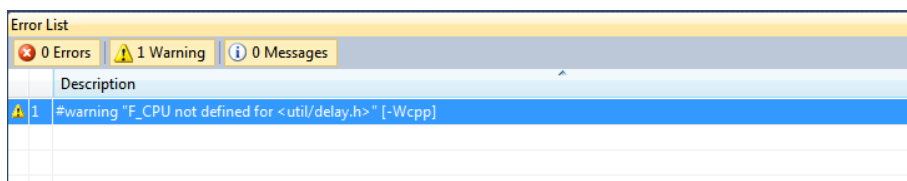
20. You should be rewarded with a very quickly flashing LED:



21. What happened? Our C code had this line:

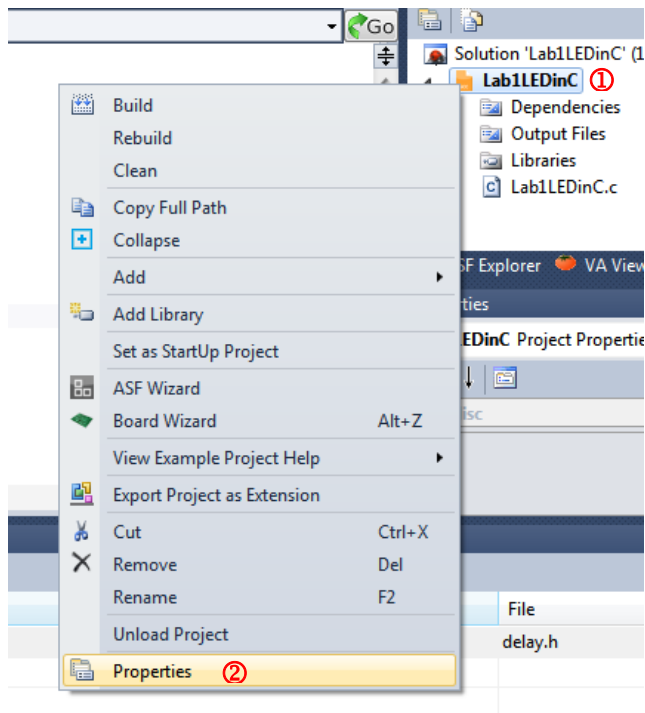
```
_delay_ms(500);
```

But the delay is much faster than that! The answer lies that if we look at the ‘Warning’ section, you should see the following:

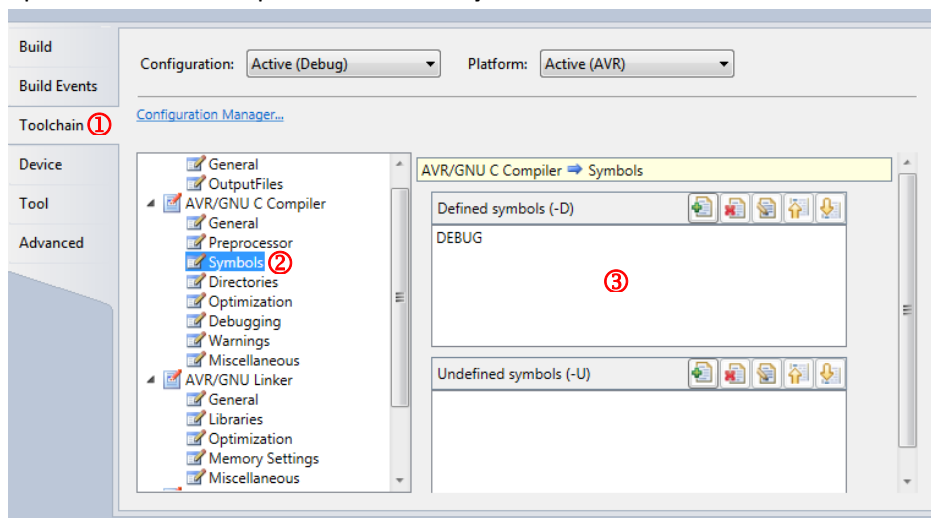


Let's fix that now.

22. Right-Click on your Project Name ① (here it is **Lab1LEDinC**), then Select **Properties** ②.



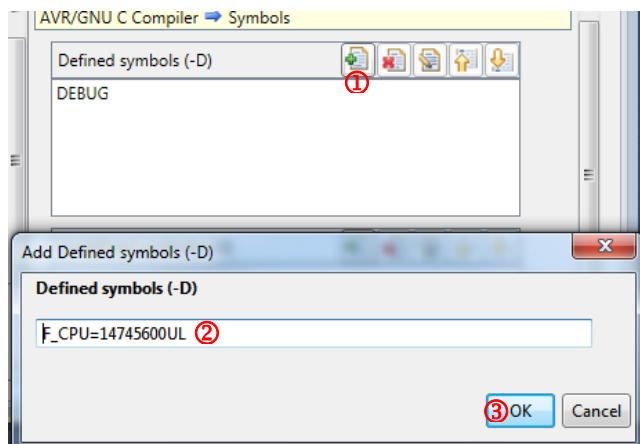
23. Select the **Toolchain** tab ①, then Select the **AVR/GNU C Compiler** section and the **Symbols** ② option. This should open the **Defined Symbols** tab ③.



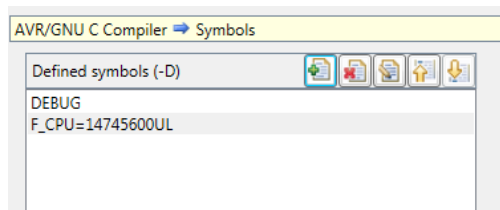
24. On this tab, hit the New Symbol option ①, then you can add a new symbol ② with the following specification:

**F\_CPU=14745600UL**

Finally hit OK ③.



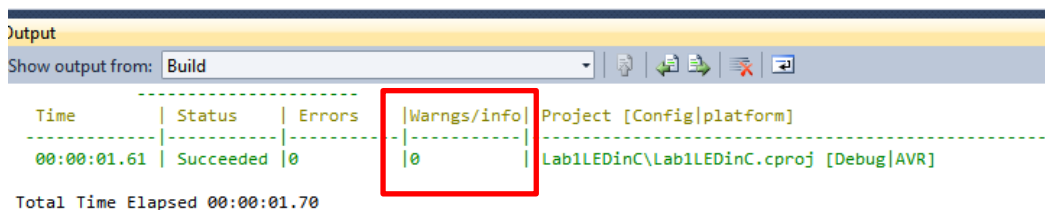
25. The results of this should be the symbol is added to your project:



26. Save the project:



27. Rebuild it. You should note there is no additional warnings now!



28. Re-Program this using previously mentioned steps. You should notice the LED is flashing much slower now. This completes the first part of the Lab, you will now complete this using assembly language.

## Part #2: Introduction to the Development Environment in Assembly

### Objective

- Familiarize yourself with the microprocessor module, power requirements, and breadboard

### Required Materials

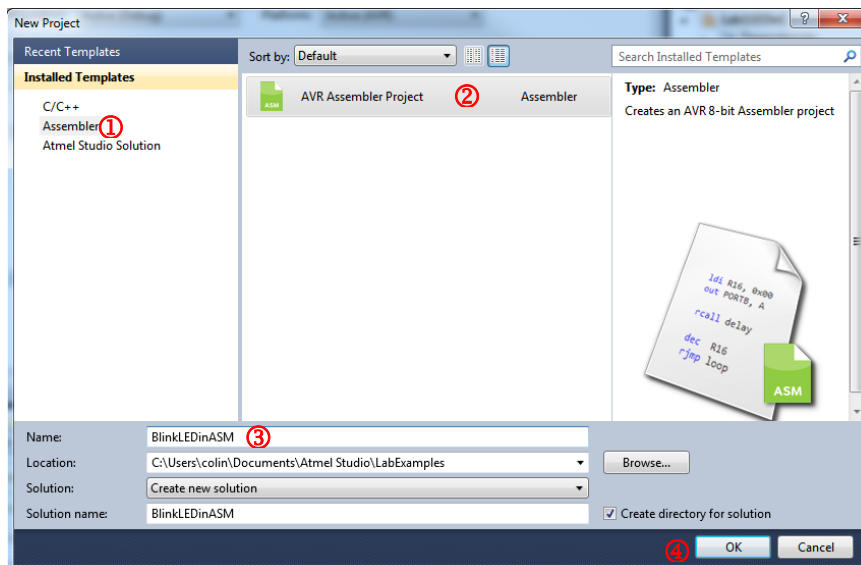
- Microprocessor Module with Programmer
- Breadboard
- USB Cable
- Power Supply
- Computer with Atmel Studio 6.2 and Programmer Utility installed

### Background

This section of the lab assumes you have completed Part #1 already, as in Part #1 you learned about programming.

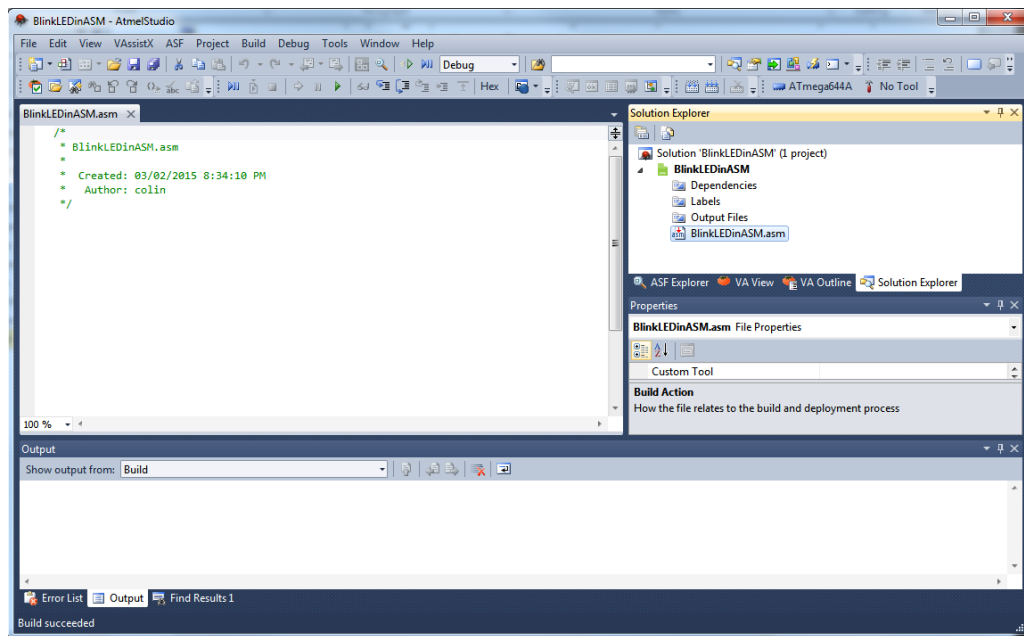
### Procedure

- Setup the board as in Part #1 – that is connected to the computer, with the power supply, and LED connected. If you have just completed Part #1 this will already be properly configured.
- Generate a new project again, but on the 'New Project' window, select the **Assembler** Option ①, and then select a **AVR Assembler Project** ②. You should set a suitable project name and location ③, and then hit **OK** ④.



- Once again select the ATmega644A Microcontroller as the part, and generate a new project.
- Your new project should look something like this:





Copy the following code into the editor window:

```

.include "m644Adef.inc"
.def temp = R16

.equ dlyconstant = 1000000

.org $0000 jmp RESET

RESET:
    ldi temp, high(RAMEND)
    out SPH, temp
    ldi temp, low(RAMEND)
    out SPL, temp

    ldi temp, 0b00000001
    out DDRB, temp

MAIN:
    sbi PortB, 0
    rcall DLY

    cbi PortB, 0
    rcall DLY

    rjmp MAIN

DLY:
    ldi r17, byte3 (dlyconstant)
    ldi r18, high (dlyconstant)
    ldi r19, low (dlyconstant)

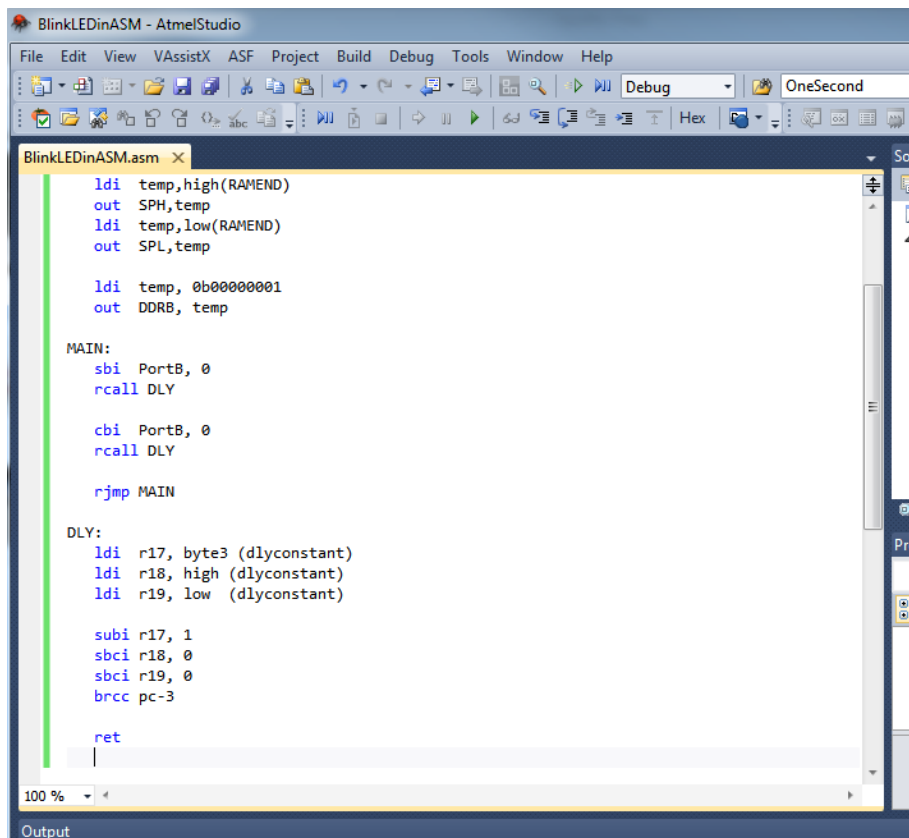
    subi r17, 1
    sbci r18, 0

```

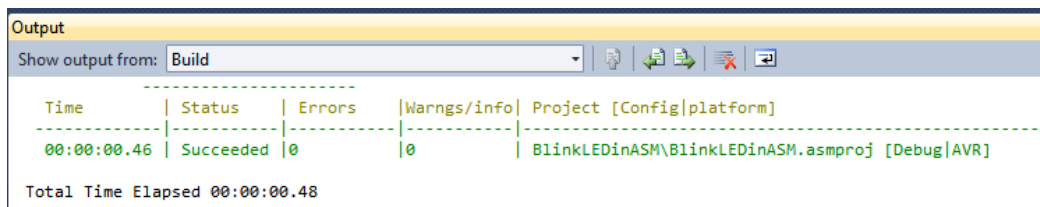
```
sbci r19, 0
brcc pc-3

ret
```

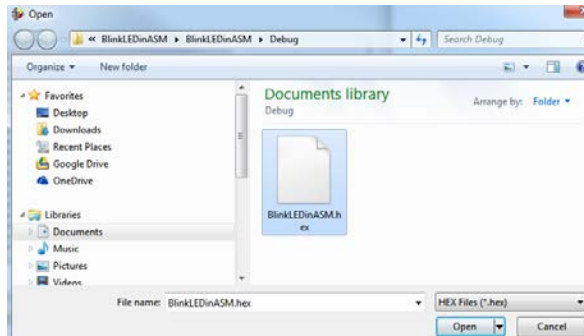
Which should now look like this:



5. Hit the 'Build Solution' button again, check you have no errors/warnings:



6. Open the AVRusbProg Programmer software, and select the BlinkLEDinASM.hex file to download to the board.



7. Check the speed the LED is blinking at, is it faster or slower than before?

## Lab Questions

1. In part 1, we set the clock frequency with the `F_CPU` directive. In MHz what was the clock frequency of the board?
2. What effect did setting the correct frequency have on the `_delay_ms( )` function?
3. Looking at the AVR Programmer Utility, you should have noticed a difference between the C Code (from Part 1), and the Assembly Code (from Part 2). Which HEX file seemed to contain more data (i.e. was larger)?
4. What was the relative blinking speed of the LED from Part 2 (using Assembly) to Part 1 (using C) (i.e. faster/slower)?