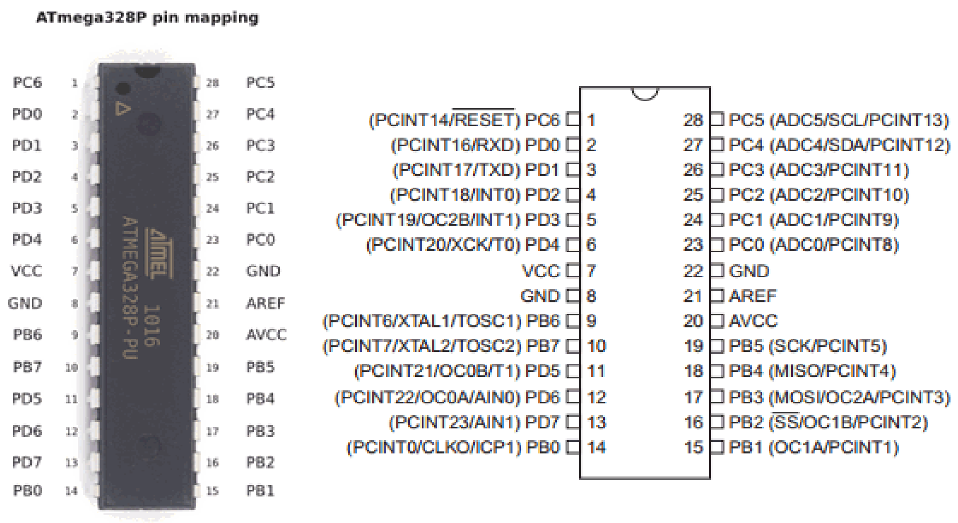
Lab no 07: Introduction to AVR Microcontroller (Atmega328P)

The purpose of this Lab is to introduce a general knowledge about the AVR microcontroller architecture and tools, as an example, we will study the famous **Atmega328P,** which is used in the Arduino Uno board.
We will install Atmel studio and know how to generate a hex file. Next, we will build our first embedded C application, the blink-led application. Then, we will learn how to mix c code with assembly code using inline assembly. Finally, we will install **Arduino builder** and use it to port hex to the Arduino kit.

Parts: -

1. Introduction to Atemega328P.
2. AVR studio setup and steps to generate hex files.
3. Blink application in C language.
4. Blink application in assembly language.
5. Configuration Steps for Atmel Studio for Flashing or Programming an Arduino Board.
6. Arduino builder to port hex to Arduino kit.

Part 1. Introduction to Atmega328P



In this lab, we will focus on GPIO, first download the Atmega328P datasheet from:

https://www.alldatasheet.com/datasheet-pdf/pdf/241077/ATMEL/ATMEGA328P.html

GPIO (General Purpose Input/Output) is a software-controlled interface that is usually found on microcontrollers and some microprocessor ICs or interface chipsets. Typically, the GPIO is one or more pins on the IC which have no special purpose in themselves. It facilitates an optional ability for device designers to create an interface/connection between the IC and a peripheral component by programming some hardware registers.

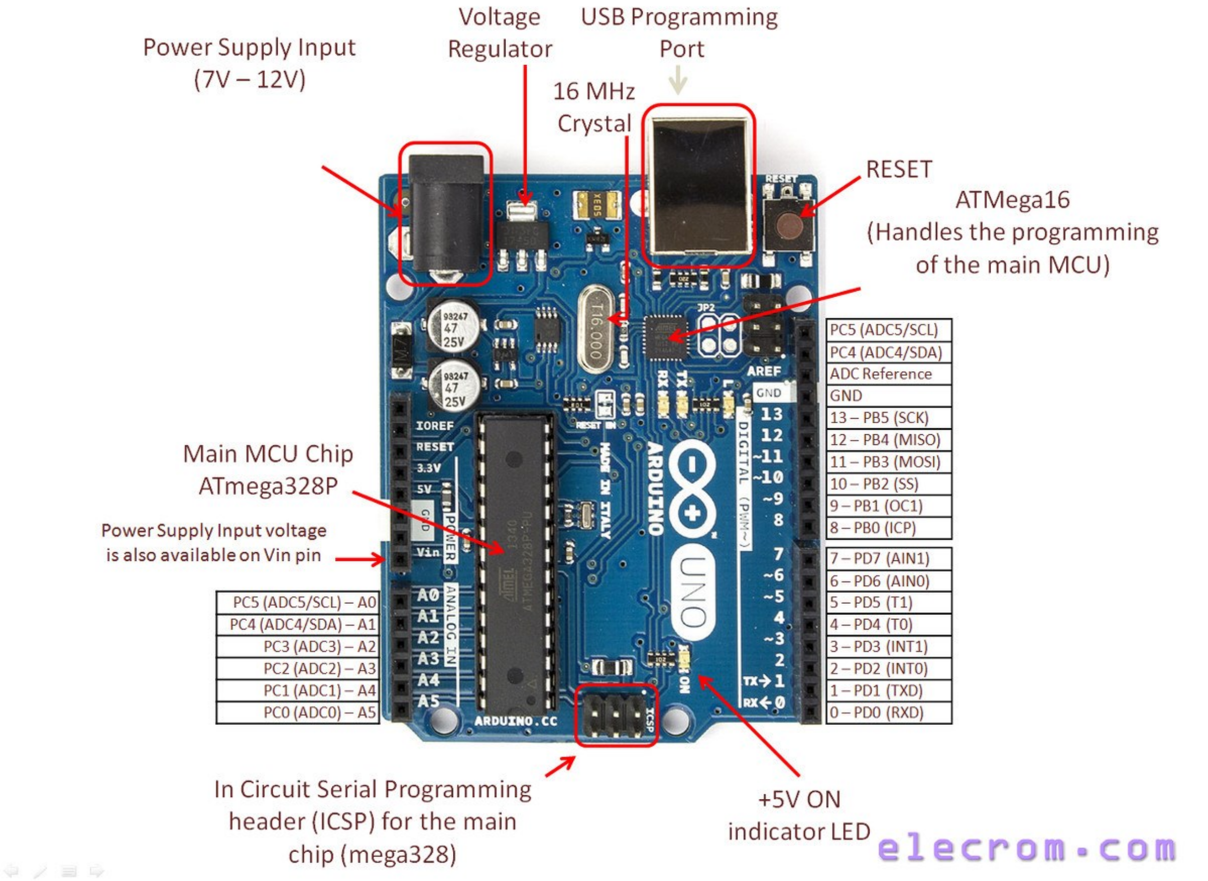
A GPIO pin is a generic pin whose value consists of one of two voltage settings (**high or low**) and whose behavior can be programmed through software.

`GPIOPin.INPUT`	The GPIO pin is configured for input and is only readable.
`GPIOPin.OUTPUT`	The GPIO pin is configured for output and is both readable and writable.

A GPIO port is a platform-defined grouping of GPIO pins (often 4 or more pins). However, GPIO pins that are part of a GPIO port cannot be retrieved or controlled individually as GPIO.

Arduino Uno ATmega328P Pin Mapping

ATmega328P is a very advance and feature-rich microcontroller. It is one of the famous microcontrollers of Atmel because of its use in the Arduino UNO board, shown in the figure below.

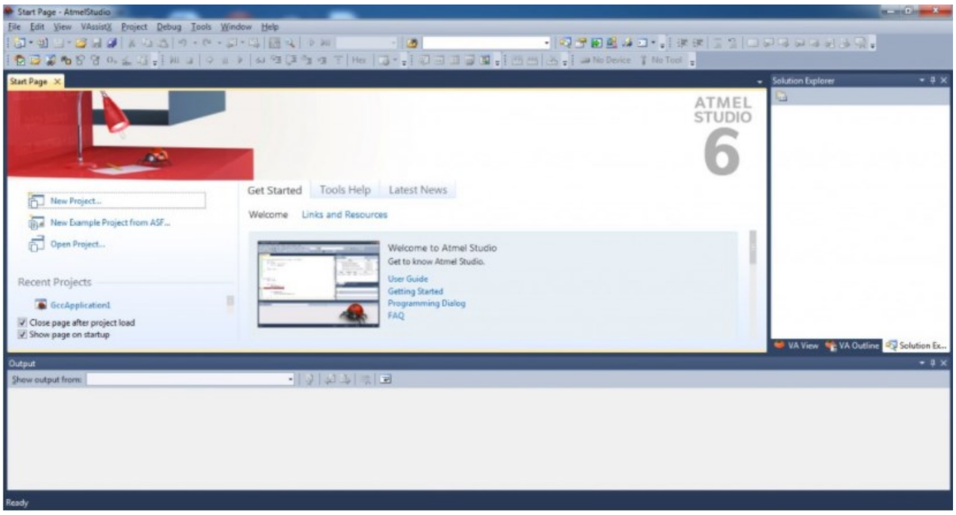


For more details about the datasheet and AVR microcontroller, kindly refer to the lab DataSheet - YouTube by Eng. Miada Aladl.

Part 2. AVR studio setup and steps to generate hex files

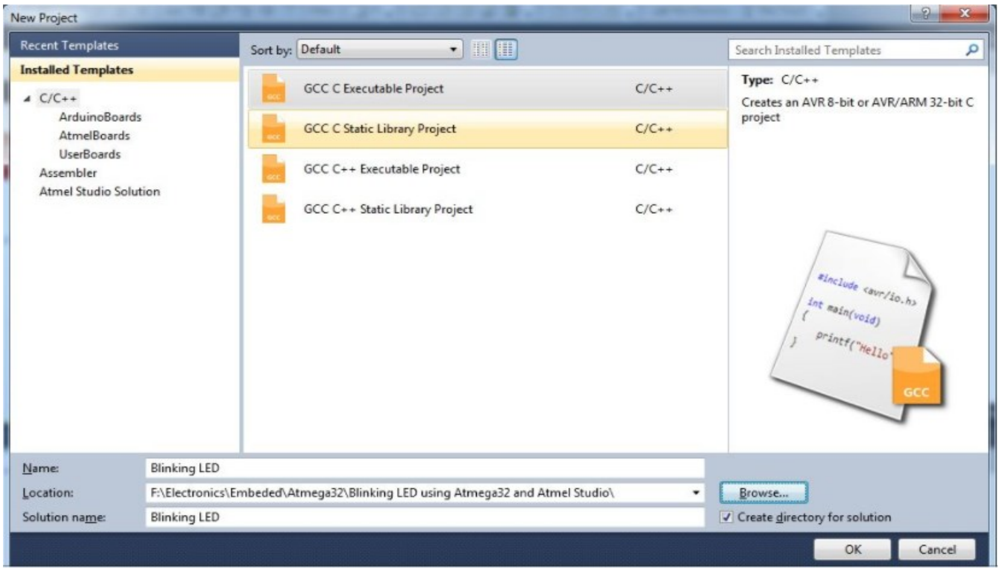
1. Download and Install Atmel Studio. You can download Atmel Studio from Atmel’s Website.

2. Open Atmel Studio



3. Select **New Project**

4. Select **GCC C Executable Project,** give a project name, solution name, and location in which the project is to be saved, and click **OK**.



Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

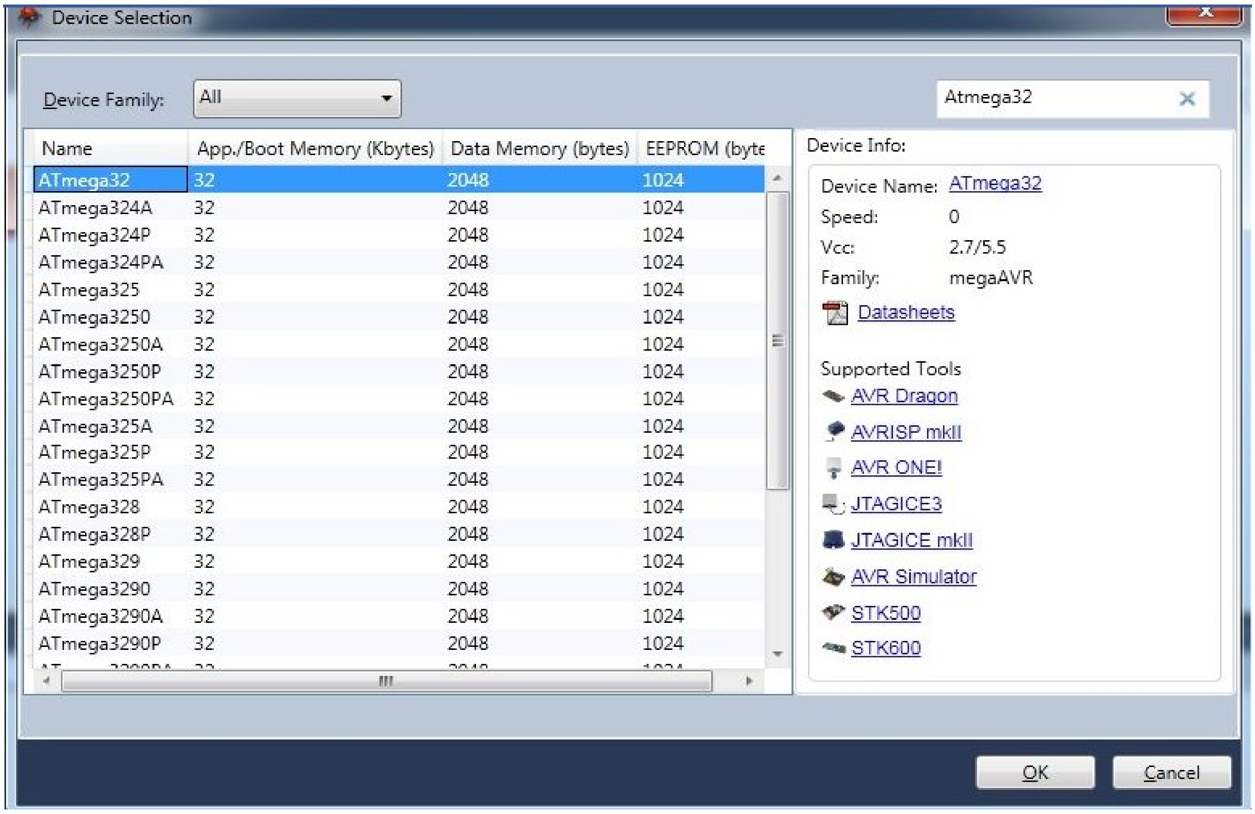
Name: Blinking LED

Location: F:\Electronics\Embeded\Atmega32\Blinking LED using Atmega32 and Atmel Studio\

Solution name: Blinking LED

☑ Create directory for solution

5. Selecting Microcontroller



Device Family: All

Atmega32

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (byte
ATmega32	32	2048	1024
ATmega324A	32	2048	1024
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

Device Info:

Device Name: ATmega32
Speed: 0
Vcc: 2.7/5.5
Family: megaAVR
Datasheets

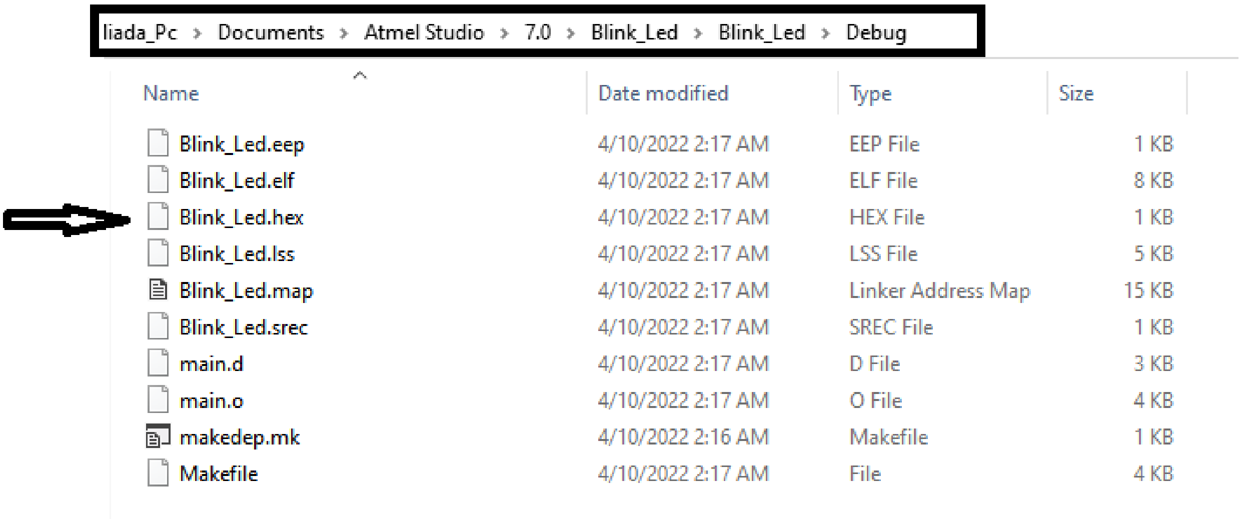
Supported Tools
- AVR Dragon
- AVRISP mkII
- AVR ONE!
- JTAGICE3
- JTAGICE mkII
- AVR Simulator
- STK500
- STK600

Choose the microcontroller that you are going to use, here we are using Atmega32. Then click **OK**.

6. Write the Program

```
#ifndef F_CPU
#define F_CPU 16000000UL // 16 MHz clock speed
#endif
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{

  while(1) //infinite loop
  {
          //code
  }
}
```

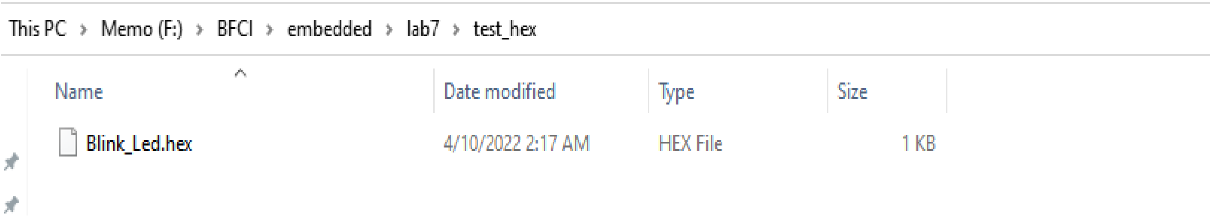
7. Then click on **Build >> Build Solution** or **Press F7** to generate the hex file.



liada_Pc › Documents › Atmel Studio › 7.0 › Blink_Led › Blink_Led › Debug

Name	Date modified	Type	Size
Blink_Led.eep	4/10/2022 2:17 AM	EEP File	1 KB
Blink_Led.elf	4/10/2022 2:17 AM	ELF File	8 KB
Blink_Led.hex	4/10/2022 2:17 AM	HEX File	1 KB
Blink_Led.lss	4/10/2022 2:17 AM	LSS File	5 KB
Blink_Led.map	4/10/2022 2:17 AM	Linker Address Map	15 KB
Blink_Led.srec	4/10/2022 2:17 AM	SREC File	1 KB
main.d	4/10/2022 2:17 AM	D File	3 KB
main.o	4/10/2022 2:17 AM	O File	4 KB
makedep.mk	4/10/2022 2:16 AM	Makefile	1 KB
Makefile	4/10/2022 2:17 AM	File	4 KB

8. Save Blink_Led.hex to a separated folder to use in **part 5, 6**

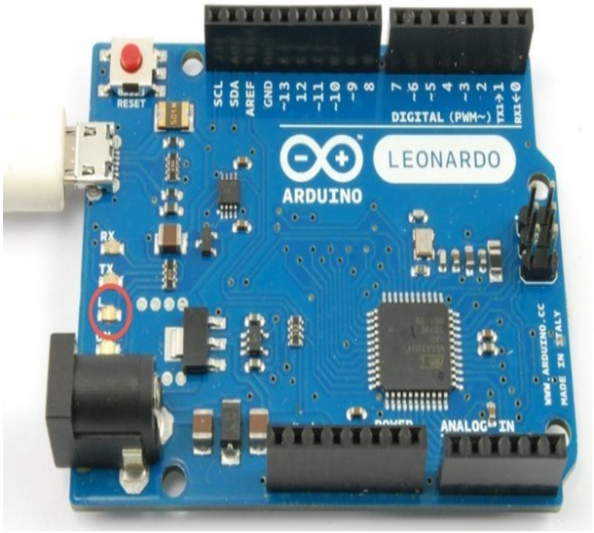


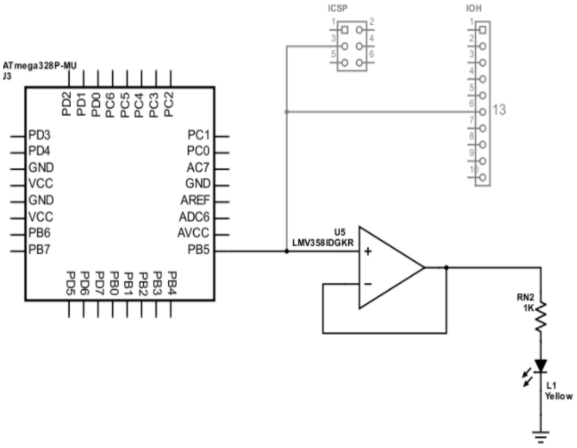
This PC › Memo (F:) › BFCI › embedded › lab7 › test_hex

Name	Date modified	Type	Size
Blink_Led.hex	4/10/2022 2:17 AM	HEX File	1 KB

Part 3. Blink application on C Language

Arduino has a single LED that you can control from your program. This LED is built onto the Arduino board and is often referred to as the 'L' LED as this is how it is labelled on the board. The position of this LED is circled in red on the pictures of the Arduino Uno and Leonardo below.





Arduino digital I/O pin 13 corresponds to Input/Output High (IOH) header pin 6. This pin connects directly to the ATmega328P microcontroller's port B bit 5 pin, labeled PB5 in the diagram.

Blink Application

In this application, the LED is turned on for 5000 ms (5 sec) and is then turned off again. You need to import two header files, one for the IO ports definition and the other for the delay function.

Notice that when programing in pure C, you need to define the exact pin where you want to write to. In our case, Arduino digital pin 13 is pin 5 of PORTB (or in binary: 0B100000).

Code of ‘L’ led blinking

```
#include <avr/io.h>
#define F_CPU 16000000
#define BLINK_DELAY_MS 5000

#include <util/delay.h>

int main (void)
{
  // Arduino digital pin 13 (pin 5 of PORTB) for output
  DDRB |= 0B100000; // PORTB5

  while(1) {
    // turn LED on
    PORTB |= 0B100000; // PORTB5
    _delay_ms(BLINK_DELAY_MS);

    // turn LED off
    PORTB &= ~ 0B100000; // PORTB5
    _delay_ms(BLINK_DELAY_MS);
  }
}
```

Part 4. Blink application in assembly language

It is possible to mix assembler and GCC in different ways:

- **Separate file**. Write a pure assembler *.s file and link it with the main C file.
- **Inline assembler**. Write inline assembler directly into the C code
- **Assembler macro**. Write an assembler macro and instantiate it in C (reusable inline assembler).

Example of Inline assembler:

We will replace C delay function in the blink code written in the previous part by assembly delay function as follows:

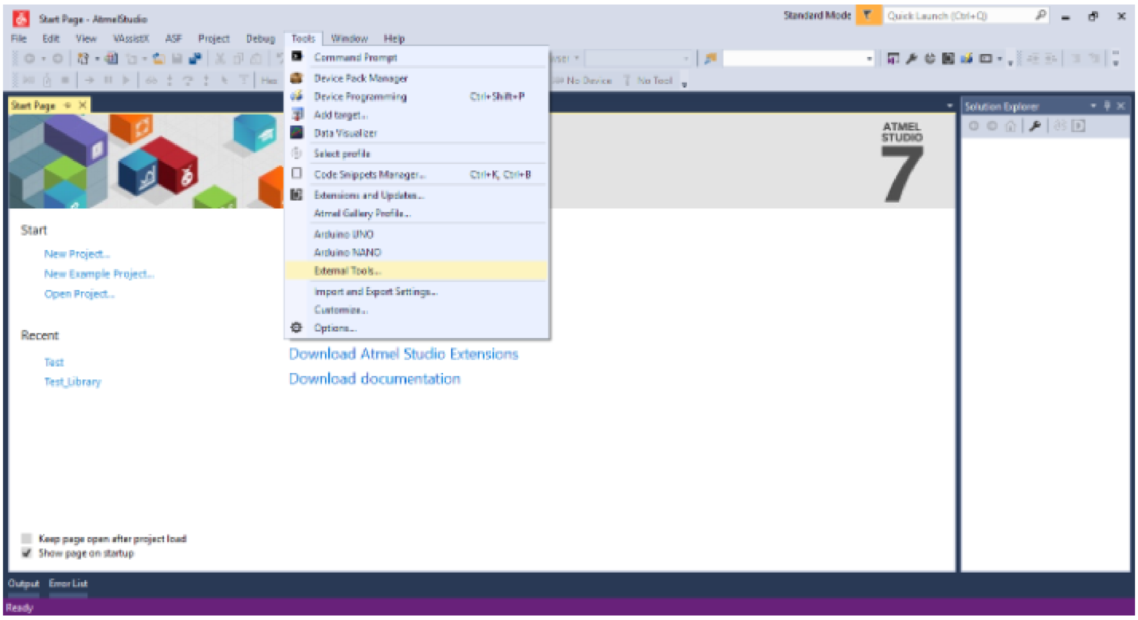
```
#include<avr/io.h>
#define F_CPU 16000000UL
#include<util/delay.h>
void delay_ms(unsigned char ms)
{
      ms;
      asm(
      "ldi r16, 31          \n"
      "OUTER_LOOP:            \n"
      "ldi r24, lo8(1021)    \n"

      "ldi r25, hi8(1021)    \n"
      "DELAY_LOOP:            \n"

      "adiw r24, 1           \n"

      "brne DELAY_LOOP    \n"
      "dec r16               \n"
      "brne OUTER_LOOP       \n"
      "ret                   \n");
}
int main()
{
      DDRD=0x04;
      while(1)
      {
             PORTD=0x00;
             delay_ms(1000);    //assemply delay for test
             PORTD=0x04;
             delay_ms(1000);
      }
}
```

- Then click on **Build >> Build Solution** or **Press F7** to generate the hex file.

Part 5. Configuration Steps for Atmel Studio for Flashing or Programming an Arduino Board

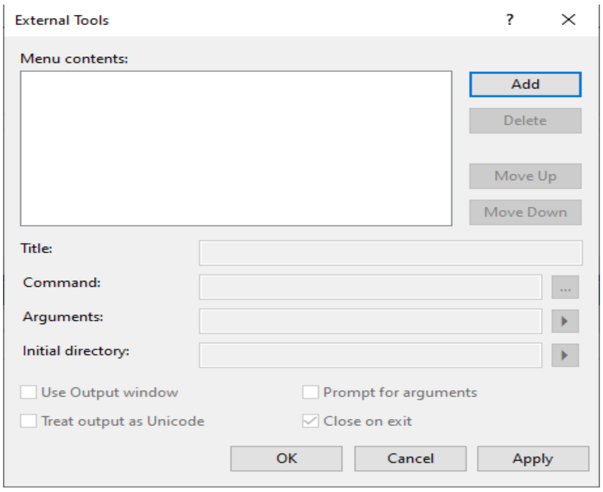
1. **Open** your installed Atmel Studio IDE. Go to External Tools Menu

   On the Menu Bar go to *Tools -> External Tools*



2. **External** Tool Window

   You should see a window like this but will be empty for you.



□ Use Output window
□ Prompt for arguments
□ Treat output as Unicode
☑ Close on exit

3. **Add** a new Tool for Arduino UNO

Click 'Add' to add a new Tool. And fill the text boxes as below.

For Title, you can give any title you want.

For Command, It should have the path to avrdude.exe that will be in the location where you have installed the Arduino IDE.

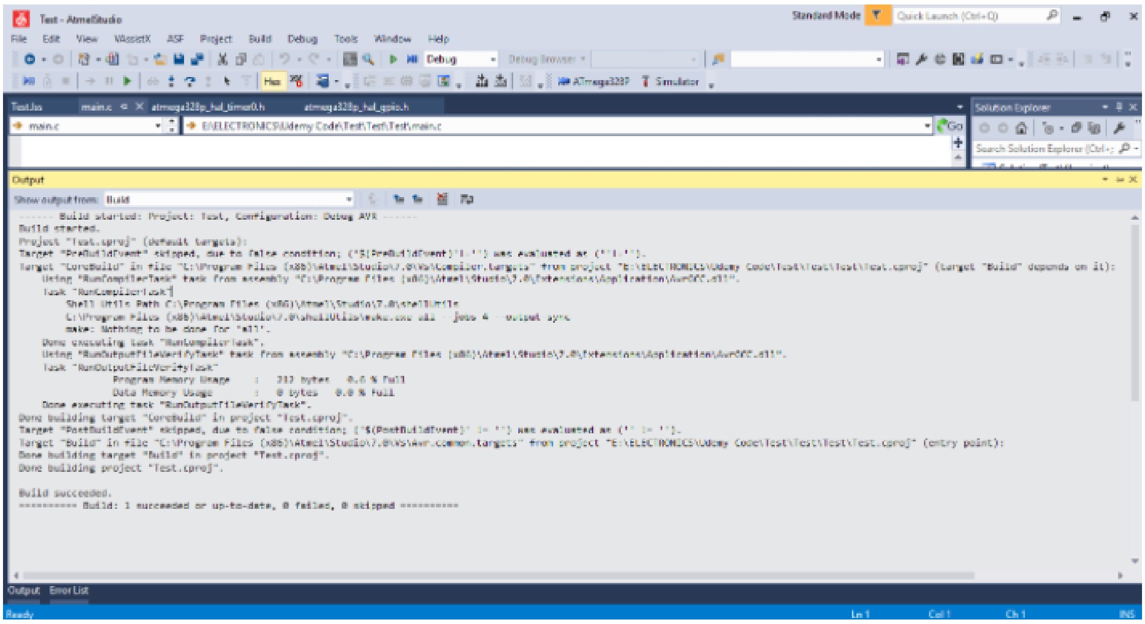
For Arguments, It should have the 3 most important parameters, the Microcontroller which is dependent on the Arduino board you are using, COM Port, and Baud Rate. COM Port system is dependent and can be determined from Device Manager. Baud Rate should be 115200.

For example:

- Title: Arduino UNO
- Command: C:\Program Files (x86)\Arduino\hardware\tools\avr\bin\avrdude.exe
- Arguments: -C"C:\Program Files (x86)\ arduino\ hardware\ tools\avr\etc\ avrdude.conf" -v -patmega328p -carduino -PCOM10 -b115200 -D -Uflash:w:"$(ProjectDir)Debug\$(TargetName).hex":i

4. **Select** 'Use Output Window'

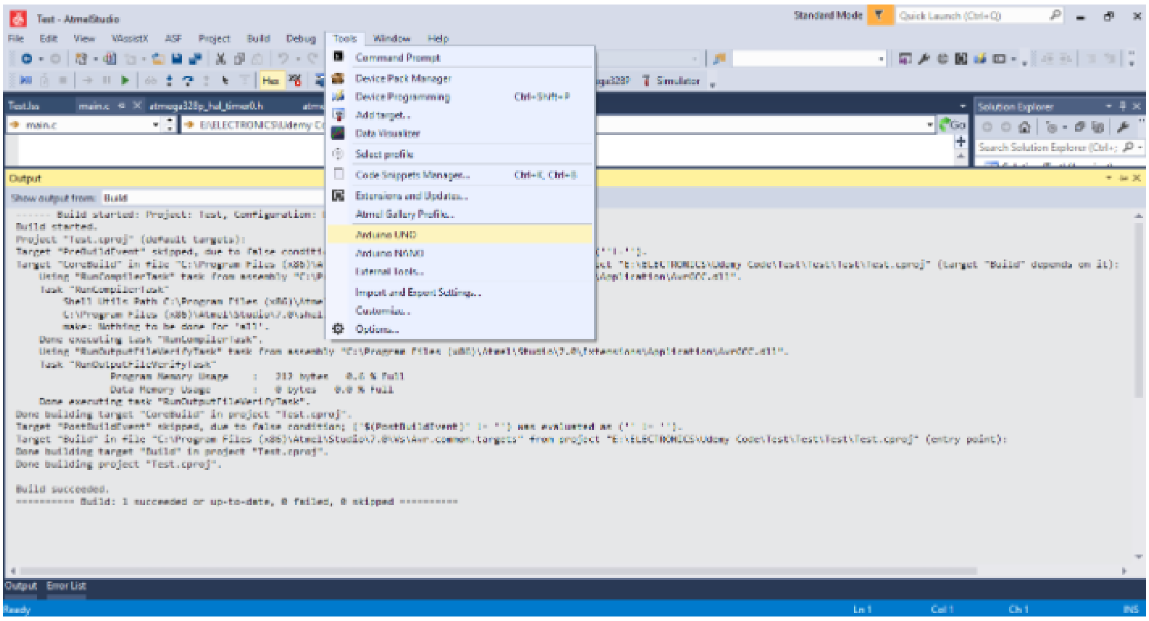
5. **Build** Your Program



```
------ Build started: Project: Test, Configuration: Debug AVR ------
Build started.
Project "Test.cproj" (default targets):
Target "PreBuildEvent" skipped, due to false condition; ('$(PreBuildEvent)'!='') was evaluated as (''!='').
Target "CoreBuild" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Compiler.targets" from project "E:\ELECTRONICS\Udemy Code\Test\Test\Test\Test.cproj" (target "Build" depends on it):
    Using "RunCompilerTask" task from assembly "C:\Program Files (x86)\Atmel\Studio\7.0\Extensions\Application\AvrGCC.dll".
    Task "RunCompilerTask"
        Shell Utils Path C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils
        C:\Program Files (x86)\Atmel\Studio\7.0\shellUtils\make.exe all --jobs 4 --output-sync
        make: Nothing to be done for 'all'.
    Done executing task "RunCompilerTask".
    Using "RunOutputFileVerifyTask" task from assembly "C:\Program Files (x86)\Atmel\Studio\7.0\Extensions\Application\AvrGCC.dll".
    Task "RunOutputFileVerifyTask"
                Program Memory Usage    :   212 bytes   0.6 % Full
                Data Memory Usage       :   0 bytes   0.0 % Full
    Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "Test.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "E:\ELECTRONICS\Udemy Code\Test\Test\Test\Test.cproj" (entry point):
Done building target "Build" in project "Test.cproj".
Done building project "Test.cproj".

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

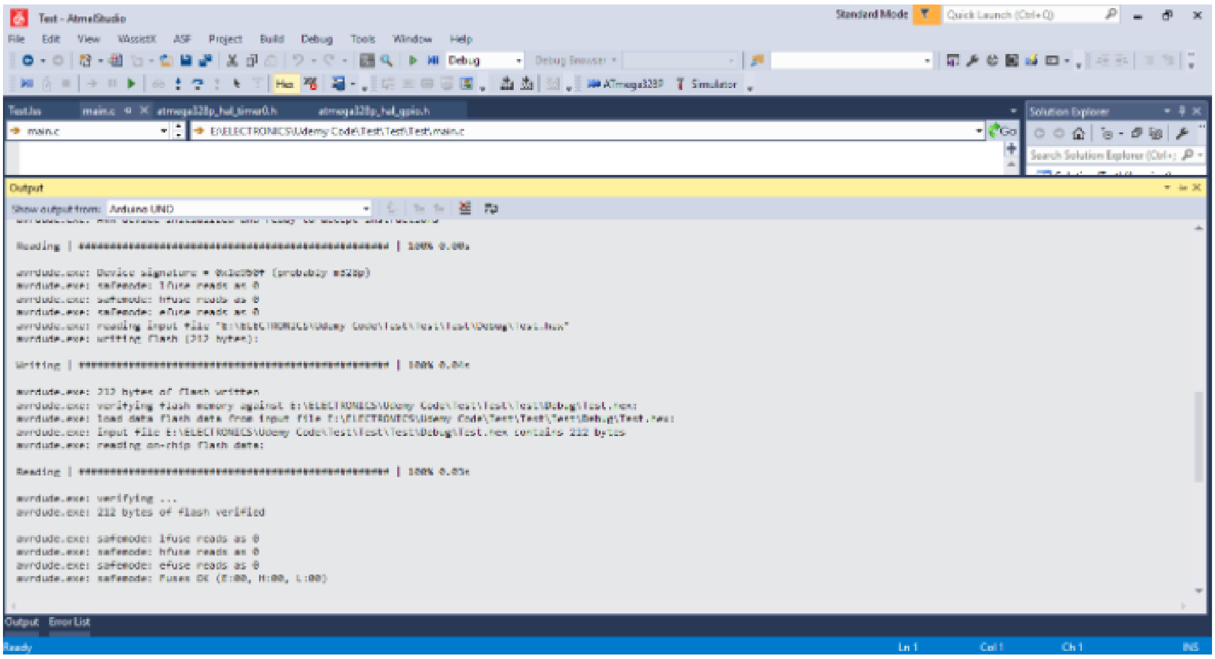
6. **Flash** Arduino and Test

Go to Menu -> Tools and Select the board you want to Test.



```
------ Build started: Project: Test, Configuration:
Build started.
Project "Test.cproj" (default targets):
Target "PreBuildEvent" skipped, due to false conditi
Target "CoreBuild" in file "C:\Program Files (x86)\A
    Using "RunCompilerTask" task from assembly "C:\P
    Task "RunCompilerTask"
        Shell Utils Path C:\Program Files (x86)\Atme
        C:\Program Files (x86)\Atmel\Studio\7.0\shel
        make: Nothing to be done for 'all'.
    Done executing task "RunCompilerTask".
    Using "RunOutputFileVerifyTask" task from assembly "C:\Program Files (x86)\Atmel\Studio\7.0\Extensions\Application\AvrGCC.dll".
    Task "RunOutputFileVerifyTask"
                Program Memory Usage    :   212 bytes   0.6 % Full
                Data Memory Usage       :   0 bytes   0.0 % Full
    Done executing task "RunOutputFileVerifyTask".
Done building target "CoreBuild" in project "Test.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files (x86)\Atmel\Studio\7.0\Vs\Avr.common.targets" from project "E:\ELECTRONICS\Udemy Code\Test\Test\Test\Test.cproj" (entry point):
Done building target "Build" in project "Test.cproj".
Done building project "Test.cproj".

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

If Everything is fine, you should get a message like this



Show output from: Arduino UNO

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

avrdude.exe: Device signature = 0x1e950f (probably m328p)
avrdude.exe: safemode: lfuse reads as 0
avrdude.exe: safemode: hfuse reads as 0
avrdude.exe: safemode: efuse reads as 0
avrdude.exe: reading input file "E:\ELECTRONICS\Udemy Code\Test\Test\Test\Debug\Test.hex"
avrdude.exe: writing flash (212 bytes):

Writing | ################################################## | 100% 0.04s

avrdude.exe: 212 bytes of flash written
avrdude.exe: verifying flash memory against E:\ELECTRONICS\Udemy Code\Test\Test\Test\Debug\Test.hex:
avrdude.exe: load data flash data from input file E:\ELECTRONICS\Udemy Code\Test\Test\Test\Debug\Test.hex:
avrdude.exe: input file E:\ELECTRONICS\Udemy Code\Test\Test\Test\Debug\Test.hex contains 212 bytes
avrdude.exe: reading on-chip flash data:

Reading | ################################################## | 100% 0.03s

avrdude.exe: verifying ...
avrdude.exe: 212 bytes of flash verified

avrdude.exe: safemode: lfuse reads as 0
avrdude.exe: safemode: hfuse reads as 0
avrdude.exe: safemode: efuse reads as 0
avrdude.exe: safemode: Fuses OK (E:00, H:00, L:00)
```

Part 6. Arduino builder to port hex to Arduino kit

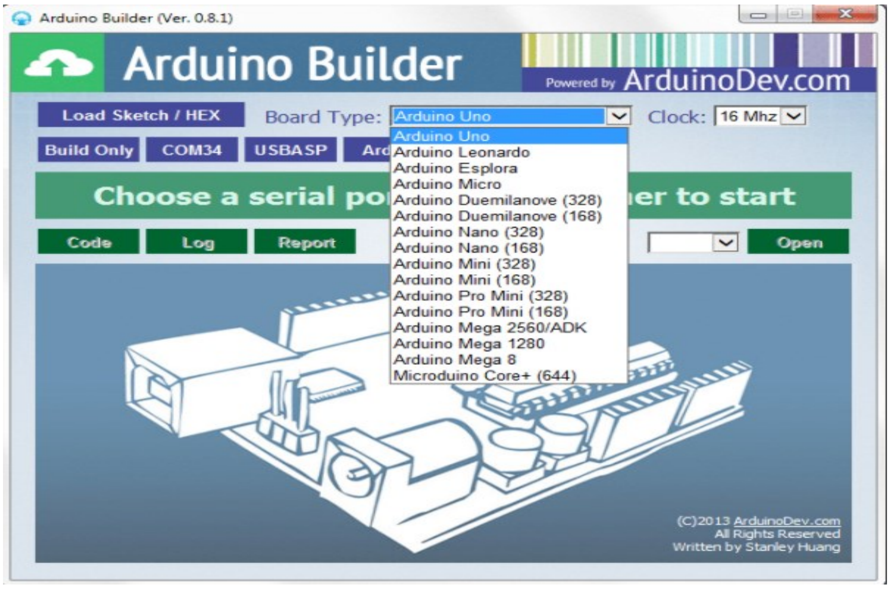
Arduino Builder is a tool for viewing and compiling Arduino sketch (source code) and programming the Arduino board with the compiled code (HEX code).

- Download Arduino builder from

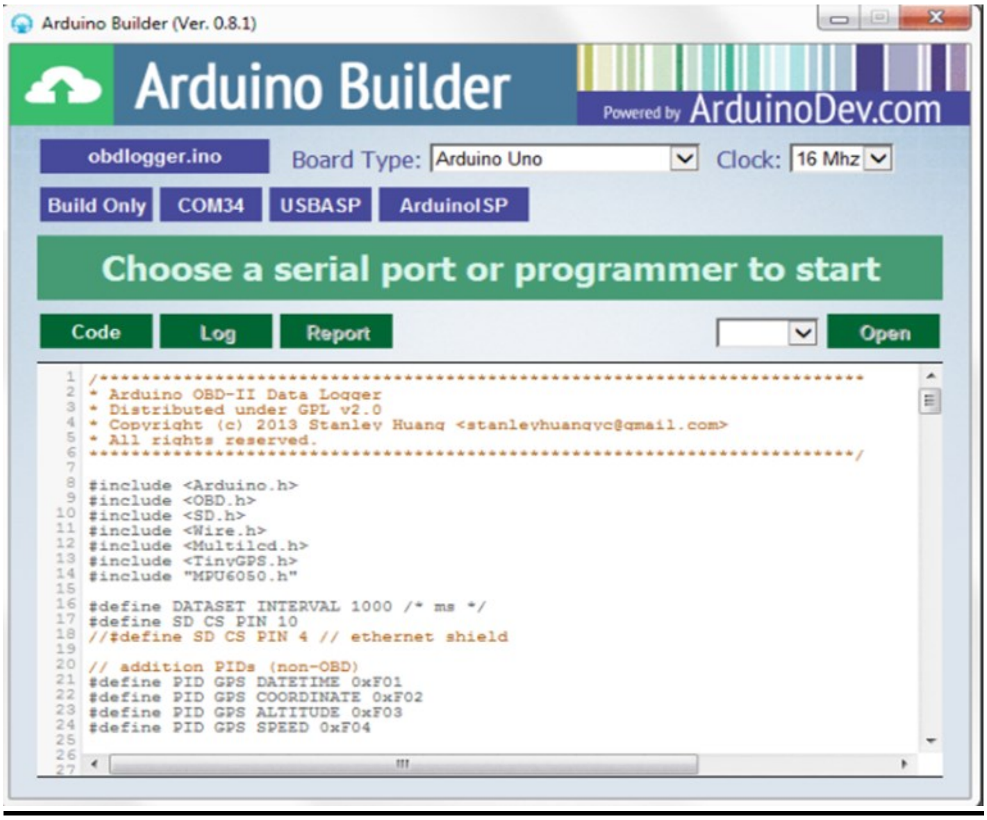
  https://sourceforge.net/projects/arduinodev/files/ArduinoBuilder/

➢ It's only 3 steps for compiling your Arduino sketch and programming the Arduino with the compiled code

STEP 1: Set board type and operating frequency if necessary



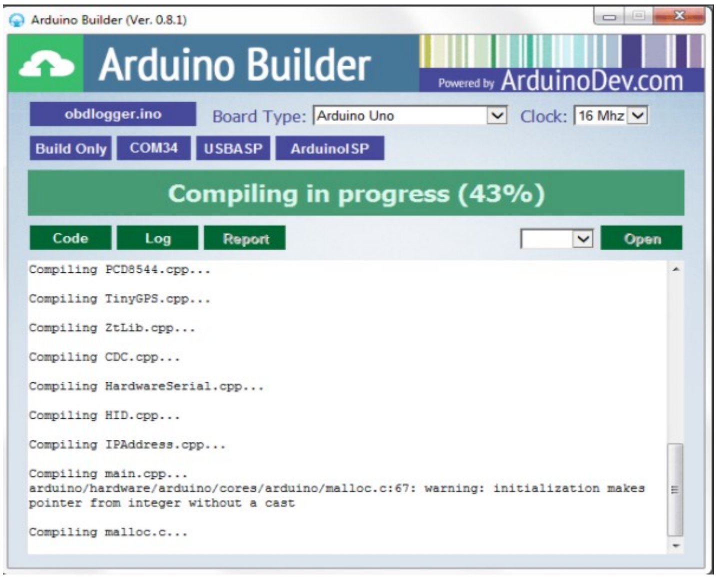
STEP 2: Load your sketch or HEX file, Click the “Load Sketch / Hex” button to choose a file to load and the content of the file will be displayed.



```
/**************************************************************************
* Arduino OBD-II Data Logger
* Distributed under GPL v2.0
* Copyright (c) 2013 Stanley Huang <stanleyhuangyc@gmail.com>
* All rights reserved.
*************************************************************************/

#include <Arduino.h>
#include <OBD.h>
#include <SD.h>
#include <Wire.h>
#include <Multilcd.h>
#include <TinyGPS.h>
#include "MPU6050.h"

#define DATASET INTERVAL 1000 /* ms */
#define SD CS PIN 10
//#define SD CS PIN 4 // ethernet shield

// addition PIDs (non-OBD)
#define PID GPS DATETIME 0xF01
#define PID GPS COORDINATE 0xF02
#define PID GPS ALTITUDE 0xF03
#define PID GPS SPEED 0xF04
```

STEP 3: Choose serial port or programmer, Click one of the serial port or programmer button shown the whole procedure will begin. If you only want to compile the code into HEX file, click *Build Only* button.

Once started, the program will switch to a console view in which the process of compiling and programming will be displayed as well as an error message if there is any.

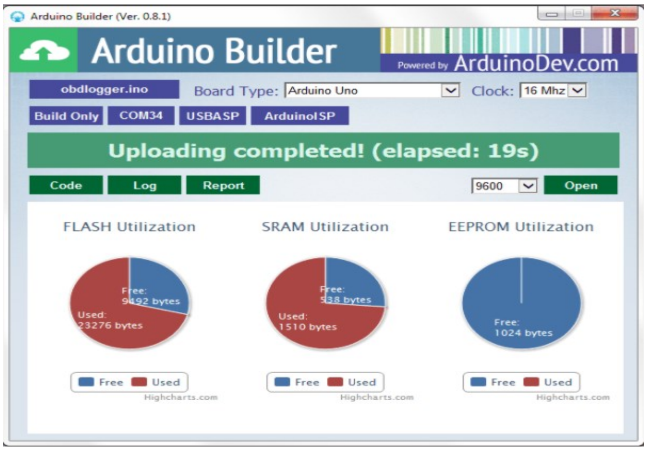


Board Type: Arduino Uno
Clock: 16 Mhz

Compiling in progress (43%)

```
Compiling PCD8544.cpp...

Compiling TinyGPS.cpp...

Compiling ZtLib.cpp...

Compiling CDC.cpp...

Compiling HardwareSerial.cpp...

Compiling HID.cpp...

Compiling IPAddress.cpp...

Compiling main.cpp...
arduino/hardware/arduino/cores/arduino/malloc.c:67: warning: initialization makes
pointer from integer without a cast

Compiling malloc.c...
```

When the process completes with no error, a report with 3 pie charts will be shown from which you can grab an overview of the memory consumption for your Arduino's FLASH, SRAM and EEPROM space.



Board Type: Arduino Uno
Clock: 16 Mhz

Uploading completed! (elapsed: 19s)

9600