Index slide

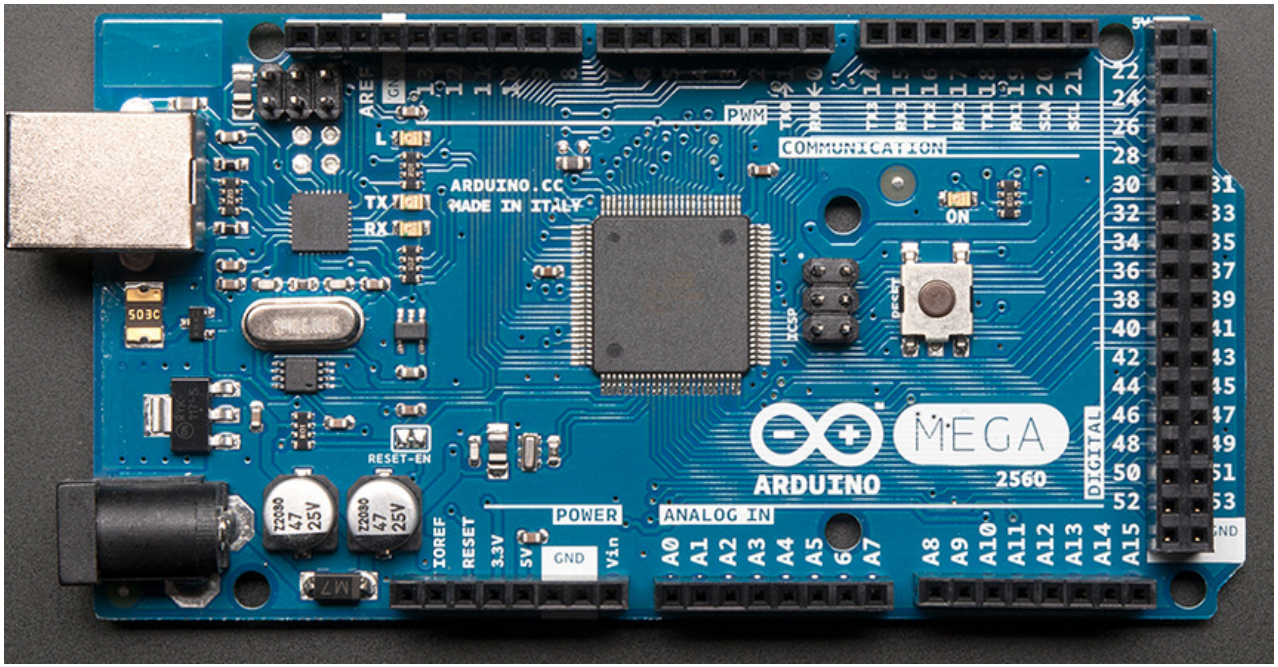
Slide Index

Week 3 -- 44 slides Michel Hanbury Sept. 21st 2017

Fall 2018 CAM8302E (Week 3) Microcomputer Interfacing Arduino Overview

Arduino Specifications and Hardware

Arduino Microcontroller



A popular inexpensive microcontroller used for embedded systems. It has 16 (10 bit A/D) inputs and synchronous and asynchronous serial ports. The device has 54 programmable digital I/O pins.

Arduino Mega Specs:

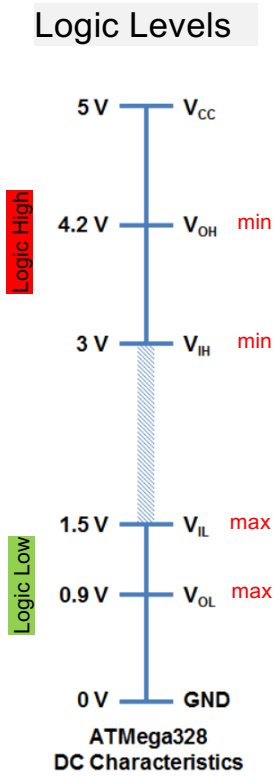
Specs (from http://arduino.cc/en/Main/ArduinoBoardMega):

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

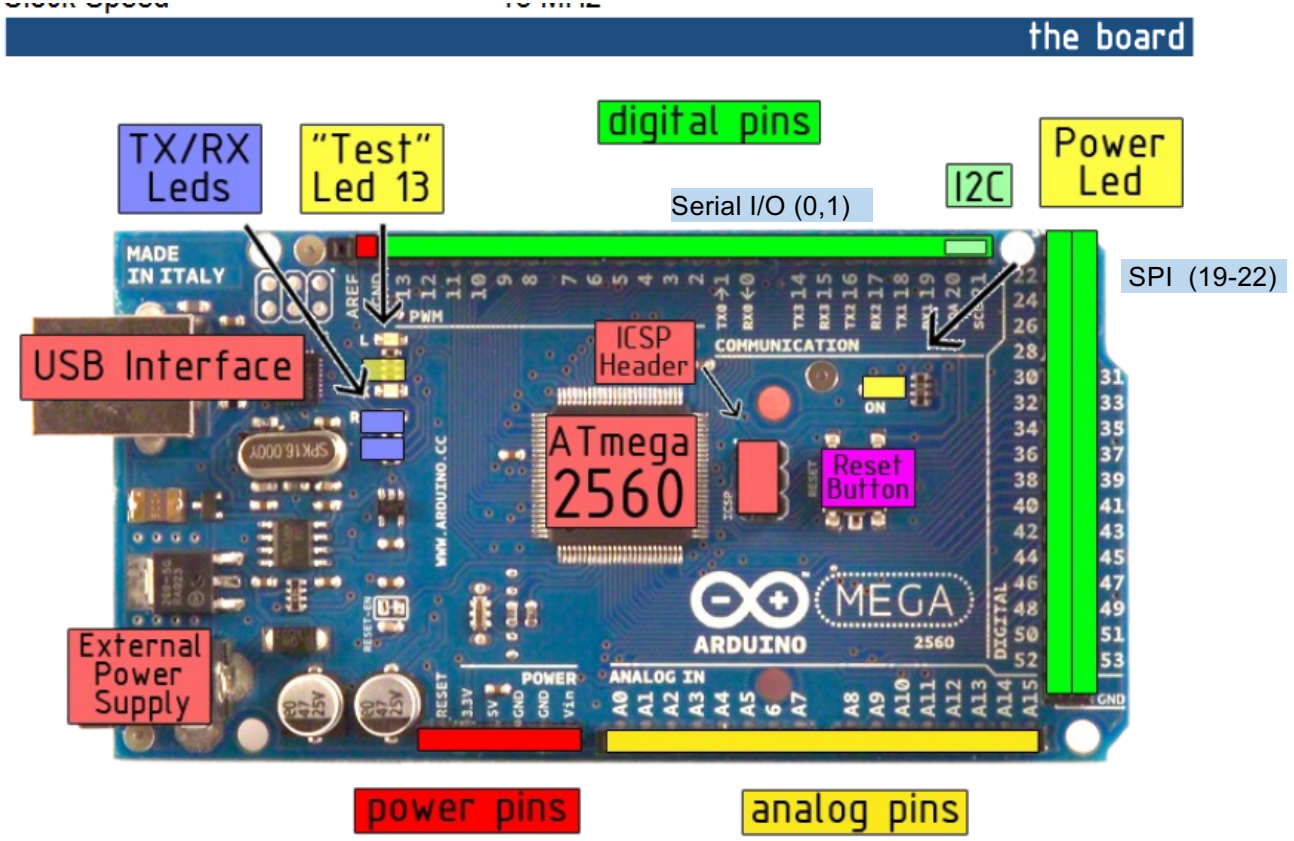
Indicates amount of memory, I/O pins and basic voltage and current specs.

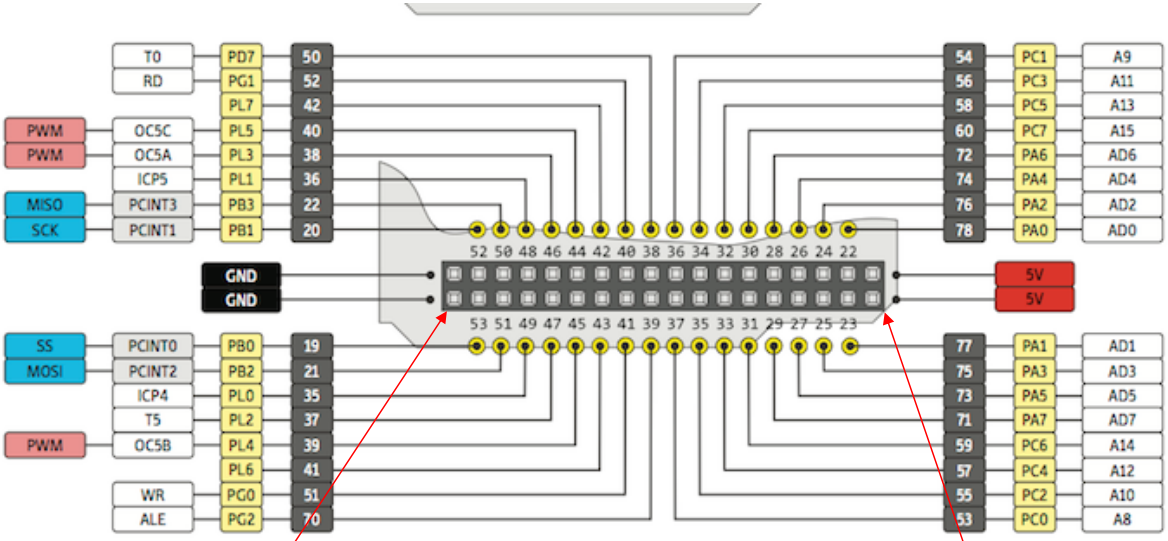
10 bit (0-5 volts)

Flash – non volatile program storage. SRAM – variable storage.



Arduino Hardware





Gnd

Connect to these points to power the circuits on your proto board.

+5

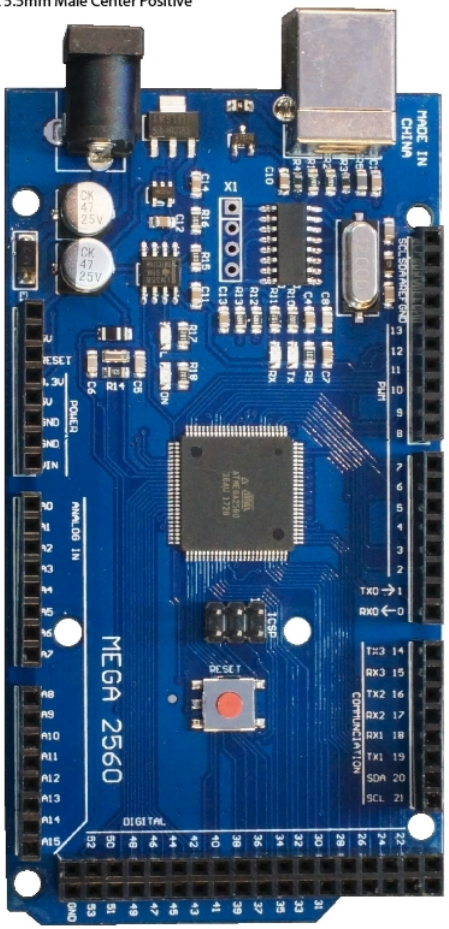
Disconnect the USB cable when making changes to your circuits. (Maximum current 100 mA)

DC Power Jack 7-12VDC Input
2.1mm x 5.5mm Male Center Positive

USB-B Port To Computer

Proto Supplies

Mega 2560 Pinout



Red numbers in parenthesis are the name to use when referencing that pin.
Analog pins are referenced as A0 thru A15 even when using as digital I/O

No Connection
I/O Reference Voltage for shields
Reset Input
3.3V Output @ 50mA
5V Output or Input
Ground
Ground
7-12V Output or Input

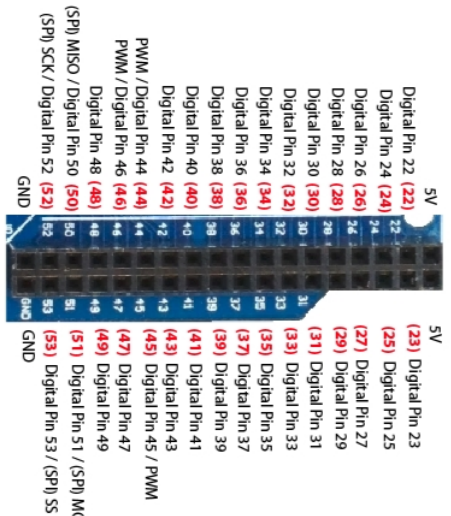
Analog Pin 0 / Digital Pin 54 **(A0)**
Analog Pin 1 / Digital Pin 55 **(A1)**
Analog Pin 2 / Digital Pin 56 **(A2)**
Analog Pin 3 / Digital Pin 57 **(A3)**
Analog Pin 4 / Digital Pin 58 **(A4)**
Analog Pin 5 / Digital Pin 59 **(A5)**
Analog Pin 6 / Digital Pin 60 **(A6)**
Analog Pin 7 / Digital Pin 61 **(A7)**

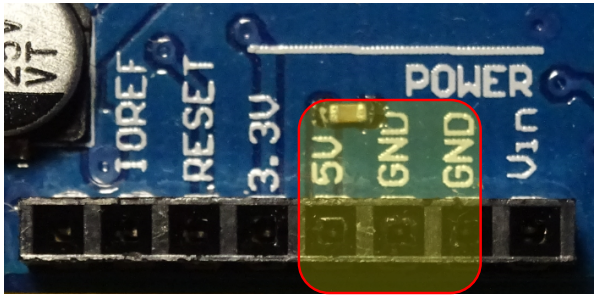
Analog Pin 8 / Digital Pin 62 **(A8)**
Analog Pin 9 / Digital Pin 63 **(A9)**
Analog Pin 10 / Digital Pin 64 **(A10)**
Analog Pin 11 / Digital Pin 65 **(A11)**
Analog Pin 12 / Digital Pin 66 **(A12)**
Analog Pin 13 / Digital Pin 67 **(A13)**
Analog Pin 14 / Digital Pin 68 **(A14)**
Analog Pin 15 / Digital Pin 69 **(A15)**

(I2C) SCL - Serial Clock
(I2C) SDA - Serial Data
Analog Reference Voltage
Ground
(13) Digital Pin 13 / PWM / Connected to on-board LED
(12) Digital Pin 12 / PWM
(11) Digital Pin 11 / PWM
(10) Digital Pin 10 / PWM
(9) Digital Pin 9 / PWM
(8) Digital Pin 8 / PWM

(7) Digital Pin 7 / PWM
(6) Digital Pin 6 / PWM
(5) Digital Pin 5 / PWM
(4) Digital Pin 4 / PWM
(3) Digital Pin 3 / PWM / Ext Int 5
(2) Digital Pin 2 / PWM / Ext Int 4
(1) Digital Pin 1 / Serial Port 0 TXD (Main Serial Port)
(0) Digital Pin 0 / Serial Port 0 RXD (Main Serial Port)

(14) Digital Pin 14 / Serial Port 3 TXD
(15) Digital Pin 15 / Serial Port 3 RXD
(16) Digital Pin 16 / Serial Port 2 TXD
(17) Digital Pin 17 / Serial Port 2 RXD
(18) Digital Pin 18 / Serial Port 1 TXD / Ext Int 3
(19) Digital Pin 19 / Serial Port 1 RXD / Ext Int 2
(20) Digital Pin 20 / (I2C) SDA / Ext Int 1
(21) Digital Pin 21 / (I2C) SCL / Ext Int 0



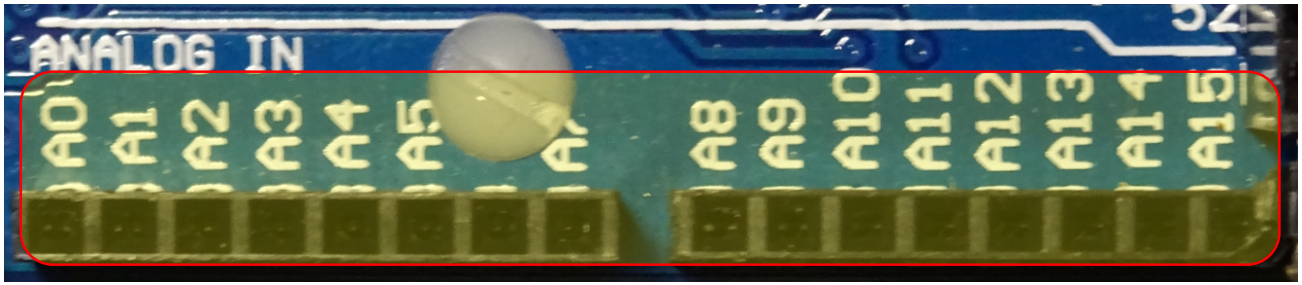


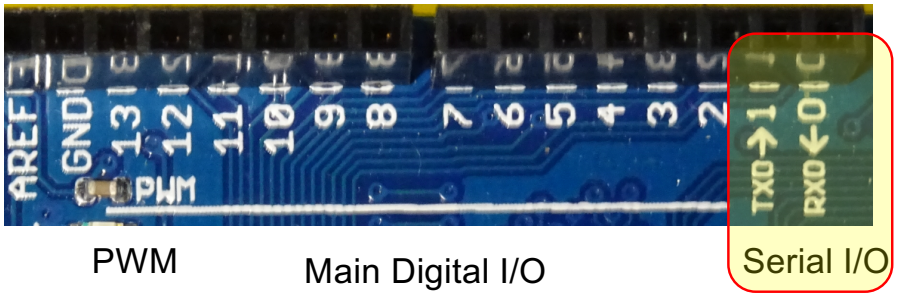
Connect power to your circuit from 5V and ground. Max of 100 mA through +5

The 5 volt pin is an output supply. Do not connect 5.0 volts from a power supply to this pin. Do not connect power to any other pin.

Arduino I/O Pins

16 analog to digital converter channels (0 to 5.0 volts)
0 to 1023 10 bits 4.88 mV resolution.

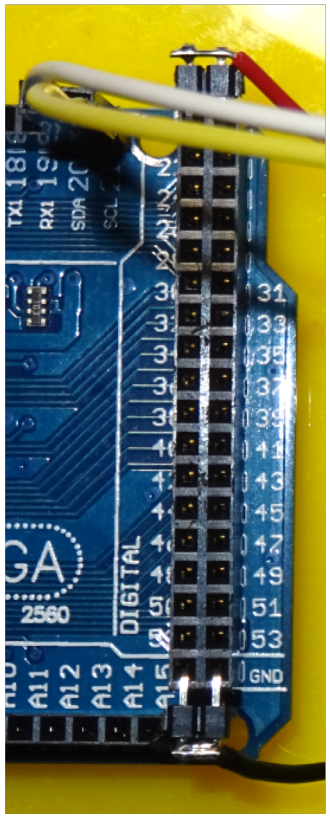




PWM

Main Digital I/O
TTL o/p max. 40 mA

Serial I/O



More Digital I/O

Arduino I/O Pins

Serial I/O

I²C

Arduino Specs:

** Read this section carefully.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- Vin. The input voltage to the board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- 5V. This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- GND. Ground pins.
- IOREF. This pin on the board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.

Arduino Specs:

Programming

The Arduino/Genuino 2560 Mega board can be programmed with the Arduino Software (IDE). For details, see the reference and tutorials.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader) 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Arduino Specs:

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the FTDI USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 2 to 13 and 44 to 46.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I2C: 20 (SDA) and 21 (SCL).** Support I2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I2C pins on the Duemilanove or Diecimila.

The Mega has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

Avoid Damaging the Arduino

** Read this section carefully.

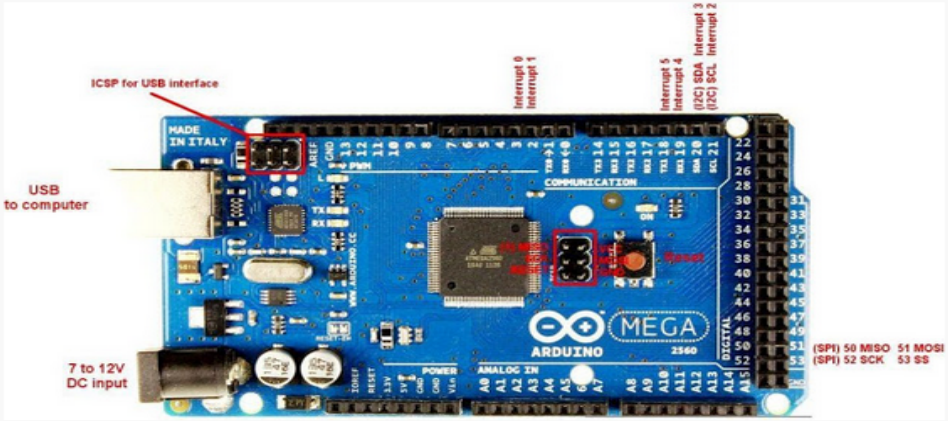
- Overloading a pin: If you try to *power* a device with a high power rating via the Arduino GPIO pins or the Vcc/GND pins, you are in danger of burning out the pin or the entire board. There's some information on pin current limits here, as well as a specific current-related problem here.
- Connecting more than 6V to *any* of the pins. If you want to use more power, use an H-bridge type chip like the L293D. The RESET pin can take up till 13V, though."The Perry Bad Journalism S"
- Shorting a GPIO pin set on OUTPUT, HIGH to GND, or one set on OUTPUT,LOW to Vcc
- Shorting two GPIO OUTPUT pins when one is HIGH and one is LOW
- Using +5V to power the Arduino instead of Vin/USB. This is actually OK, but only as long as you do not:
  - Put any load on Vin
  - Do this with the jumper set on USB

  The 5V pin is not as protected as the Vin pin, and can end up destroying things.
- Shorting anything but 0V to GND.
- Connecting more than Vcc to any GPIO pin (generally this is sort of safe till you go above Vcc+0.5V).
- Changing connections while the Arduino is on: It's *very* easy to accidentally short something.
- Switch polarities of Vcc and GND (This happens quite often when the power jack is soldered backwards.)
- Apply something other than 3.3V to the 3.3V pin. (This pin is for *output*, not input)

** Read this section carefully.

Ways in which you will possibly Destroy your Arduino!

We are not only talking about burning your Arduino Board in fire or crushing it with a hammer. But there are other method/ways to do it, that are more geeky. We know that you don't want to do it, but still these information will be helpful in order to avoid the situation, in which you would have burned/destroyed your Arduino accidentally. So don't let the smoke come out of your Arduino board, make it safe.



1: Shorting I/O Pins to Ground

Configure an I/O pin as output then set it high(Logical 1). Then short the pin to ground (GND). Now after doing that you have now created an **over-current condition** on the I/O pin & your arduino pin will be destroyed.

Reason: The MCU datasheet specifies an absolute maximum per-pin current of 40mA. With a typical internal resistance that is generally 25 ohms per pin, a short to ground can produce as much as 200mA of current to flow, more than enough to destroy the MCU pin.

2: Shorting I/O Pins to Each Other

Configure 2 **I/O pins** to be output, then set one of them to high (logic 1) & the other one to low (Logic 0). Now connect the pins together. You have now created an **over-current condition** on both I/O pins & they will get destroyed.

Reason: The MCU datasheet specifies an absolute maximum per-pin current of 40mA. With a typical internal resistance that is generally 25 ohms per pin, a short between I/O pins with +5 & 0 volts values, can produce as much as 200mA of current to flow, more than enough to destroy the MCU pin.

3: Apply Over-voltage to I/O Pins

Apply any voltage exceeding 5.5V to any I/O pin. The I/O pin is destroyed.

Reason: This forward-biases the ESD (electro-static discharge) protection diode built-in to the MCU. **Once** the voltage at the I/O pin is greater than the supply voltage (5V) by about 0.5V, the top diode starts to conduct current. This is OK for diverting a short-duration over-voltage event, like ESD (electro-static discharge), but that diode is not meant to be on all the time. It will simply burn out and stop protecting the pin. This high voltage can reach to other connected component with the MCU like USB Chip, LEDs & destroy them too.

4: Apply External Vin Power Backwards

Power your Arduino through the Vin connector pin, but with the reverse polarity of the **Vin/GND** power connection. This will fry up several components on the Arduino.

Reason: There is no reverse-voltage protection on voltages applied to the Vin connector pin. Current will flow from the GND pin of the ATmega328, through the 5V pin, back through the 5V regulator & to Vin. The same thing will happen with the ATmega8U2 MCU. Both MCUs & the 5V regulator will be destroyed.

5: Apply >5V to the 5V Connector Pin

Apply a voltage >5V to connector pin. Many components on the Arduino will be destroyed, and this voltage can also appear on your computer's USB port, and would possibly damage it.

Reason: There is no current protection on the 5V connector pin. The voltage applied is directly connected to the ATmega328P MCU, the ATmega8U2 USB interfaceMCU , and the 5V regulator, all of which can be damaged by voltages exceeding >5V, & the resulting currents that flow.

It is a common misconception that the Arduino 5V regulator will ensure that the 5V voltage remains at 5V, no matter what. IT WILL NOT! The only thing the 5V regulator can do is control current coming from the USB port or the external DC power jack. If the current is coming from an external power source directly connected to the 5V connector pin, the regulator can do nothing about it. Another consequence of applying more than 5V to the 5V connector pin is possible damage to the PC's USB port. If the Arduino is powered from USB then this excessive voltage can cause current to flow backwards through the voltage-switching MOSFET T1 and back to the PC's USB port.

6: Apply >3.3V to the 3.3V Connector Pin

Apply a voltage of 3.6V or higher to the 3.3V connector pin. Any 3.3V shields plugged in, or other devices powered from this pin, will be destroyed. If at least 9V is applied, this voltage can destroy the Arduino 3.3V regulator and also feed current back into the PC's USB port.

Reason: The 3.3V connector pin has no protection circuitry. This voltage is directly connected to the Arduino 3.3V regulator and any other shields or devices that are powered by this connector pin. If the voltage exceeds 9V, the 3.3V regulator will be destroyed and may allow current to flow backwards to the 5V node, and then backwards further to the PC's USB port. The excessive voltage will also destroy the two devices connected to the 5V node: the ATmega328P and ATmega8U2 microcontrollers.

7: Short Vin to GND

Power the Arduino from the DC power jack and short the Vin connector pin to GND. The Arduino blocking diode will be destroyed and traces on the Arduino PCB may melt and be destroyed.

Reason: There is no current limit protection on the Vin connector pin. A short circuit from Vin to GND effectively short circuits the DC power jack input, and exceeds the current rating of the blocking diode.

The amount of current that flows is limited only by the resistance of the Arduino PCB traces and the current capability of the power supply. If this is high enough, the diode D1 will be destroyed and PCB traces may melt due to the heat caused by this large current.

8: Apply 5V External Power with Vin Load

If you are powering the board from 5V applied to the 5V connector pin and you have circuitry connected to the Vin pin (or have shorted Vin to GND) then current will flow backwards through the 5V regulator and destroy it.

Reason: There is no reverse voltage protection on the 5V regulator thus current can flow from the 5V connector pin, backwards through the regulator, and to whatever is connected to Vin.

9: Apply >13V to the Reset Pin

Apply >13V to the Reset connector pin. The ATmega328P microcontroller will be damaged.

Reason: The Reset connector pin is directly connected to the reset pin on the ATmega328P. While this pin tolerates 13V, higher voltages will damage the device.

9: Apply >13V to the Reset Pin

Apply >13V to the Reset connector pin. The ATmega328P microcontroller will be damaged.

Reason: The Reset connector pin is directly connected to the reset pin on the ATmega328P. While this pin tolerates 13V, higher voltages will damage the device.

10: Exceed Total Microcontroller Current

Configure at least 10 I/O pins to be high and draw 20mA from each one (for example, by lighting 10 LED's). You have now exceeded the total supply current rating for the microcontroller and it will be damaged.

Reason: It's not enough to limit the current of each I/O pin -- the total current sourced from all I/O pins must not exceed 200mA, according to the ATmega328P datasheet.

11: Changing Connections While Powered Up

You should always avoid doing that. Otherwise it can burn that pin, in which a device is connected or even the whole board setup!

Reason: If a device is drawing current for the microcontroller's pins, and if we remove the wire from it. This will result in creation of back EMF (Due to circuit breaking action). This back emf will be of very large value if done very fast. So this can burn that pin.

12: Not Using FlyBack Diodes With Inductive Components Like motors, solenoids & relays

Use any of the inductive load, without Fly-back diodes. A condition can arise that a large back current will flow for the components, to the microcontroller pin. Ultimately destroying it.

Reason: Inductive load can produce back EMF of a very large value, thus will result in a very large current that can destroy the pic to which it is connected. The diode will make it sure to make itself in Reversed-Biased mode, whenever it happens. So a large current will be avoided.

Arduino IDE Install and Software (IDE) Integrated Development Environment

The Arduino IDE is a free open-source program.

The latest version is 1.8.7

Arduino Board Setup:

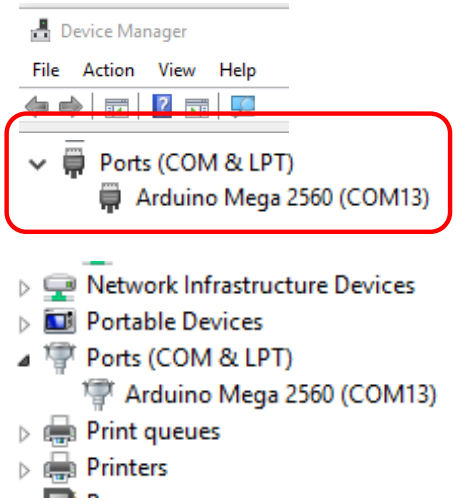
The Arduino converts the serial data to USB.

The PC converts the USB data to asynchronous serial.

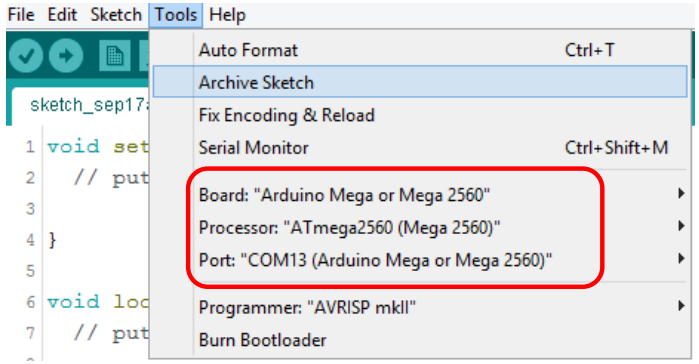
1

2

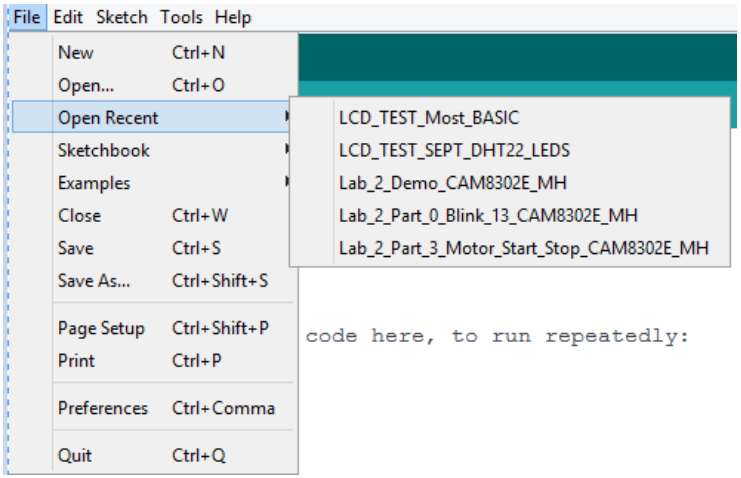
3



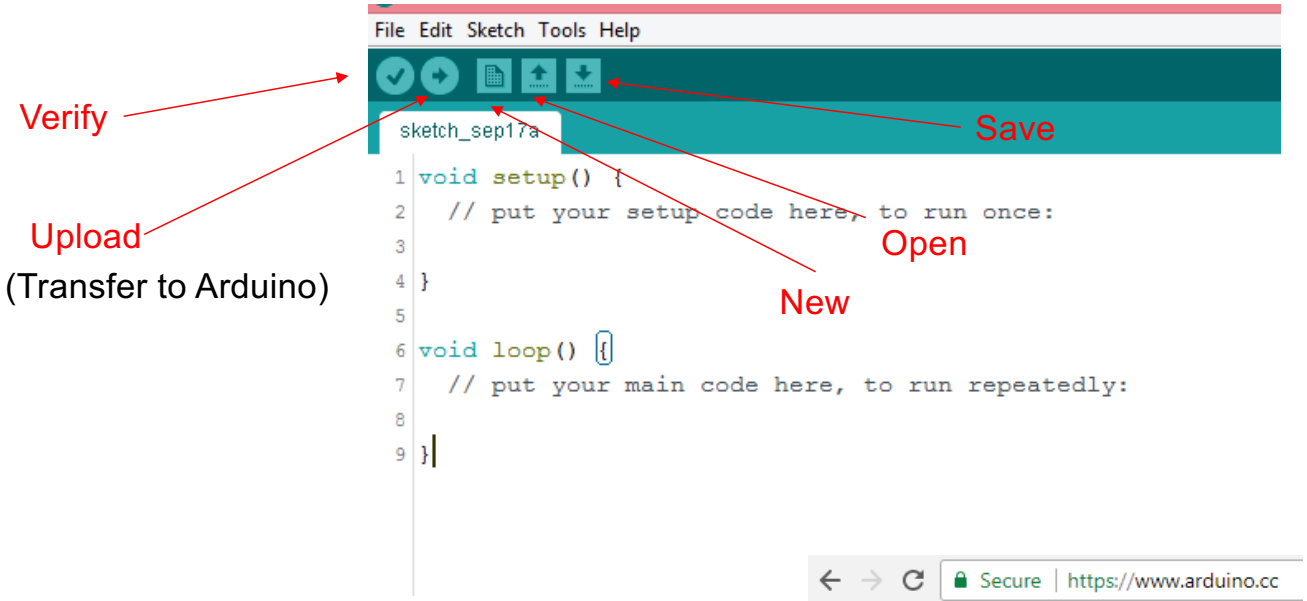
4



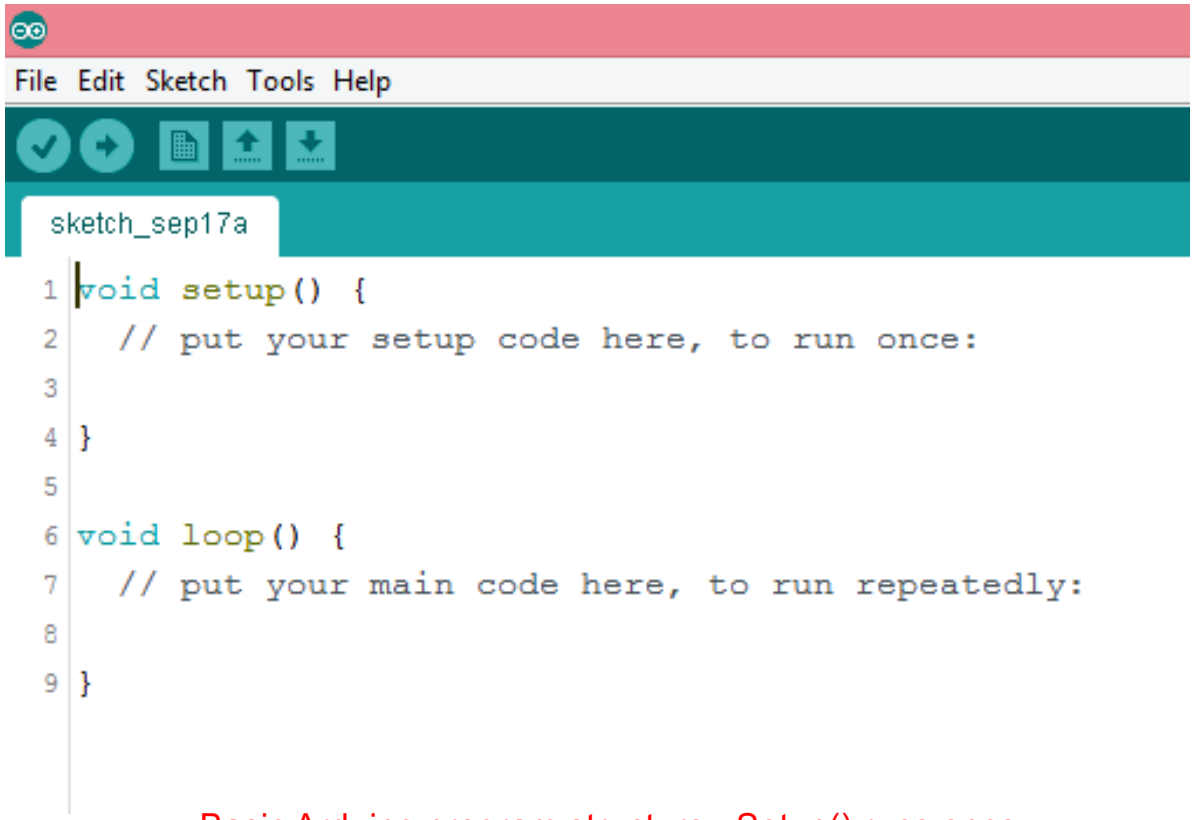
5



Arduino IDE (Integrated Development Environment)



Use the IDE to write, debug, verify and transfer your program.
The IDE can be downloaded for free:



Basic Arduino program structure. Setup() runs once and the loop() functions runs continuously once started.

Arduino Program (sketch.ino): Definitions, Libraries, Setup, Loop

Lab_2_Demo_CAM8302E_MH §

```
1   /*  Lab_2_Part_0_Blink_CAM8302E_MH.ino */    // comment
2
3
4   #define GrnLed = 5       // pin definitions
5   #define RedLed = 6
6   #define Start_Sw = 4
7
8   void setup()
9   {
10  Serial.begin(9600);      // configure the serial port
11
12  pinMode(Start_Sw,INPUT);    // difine line
13  pinMode(RedLed,OUTPUT);
14  pinMode(GrnLed,OUTPUT);
15  }
16
```

structure

The basic structure of the Arduino programming language is fairly simple and runs in at least two parts. These two required parts, or functions, enclose blocks of statements.

```
void setup()
{
  statements;
}

void loop()
{
  statements;
}
```

Where setup() is the preparation, loop() is the execution. Both functions are required for the program to work.

The setup function should follow the declaration of any variables at the very beginning of the program. It is the first function to run in the program, is run only once, and is used to set pinMode or initialize serial communication.

The loop function follows next and includes the code to be executed continuously – reading inputs, triggering outputs, etc. This function is the core of all Arduino programs and does the bulk of the work.

setup()

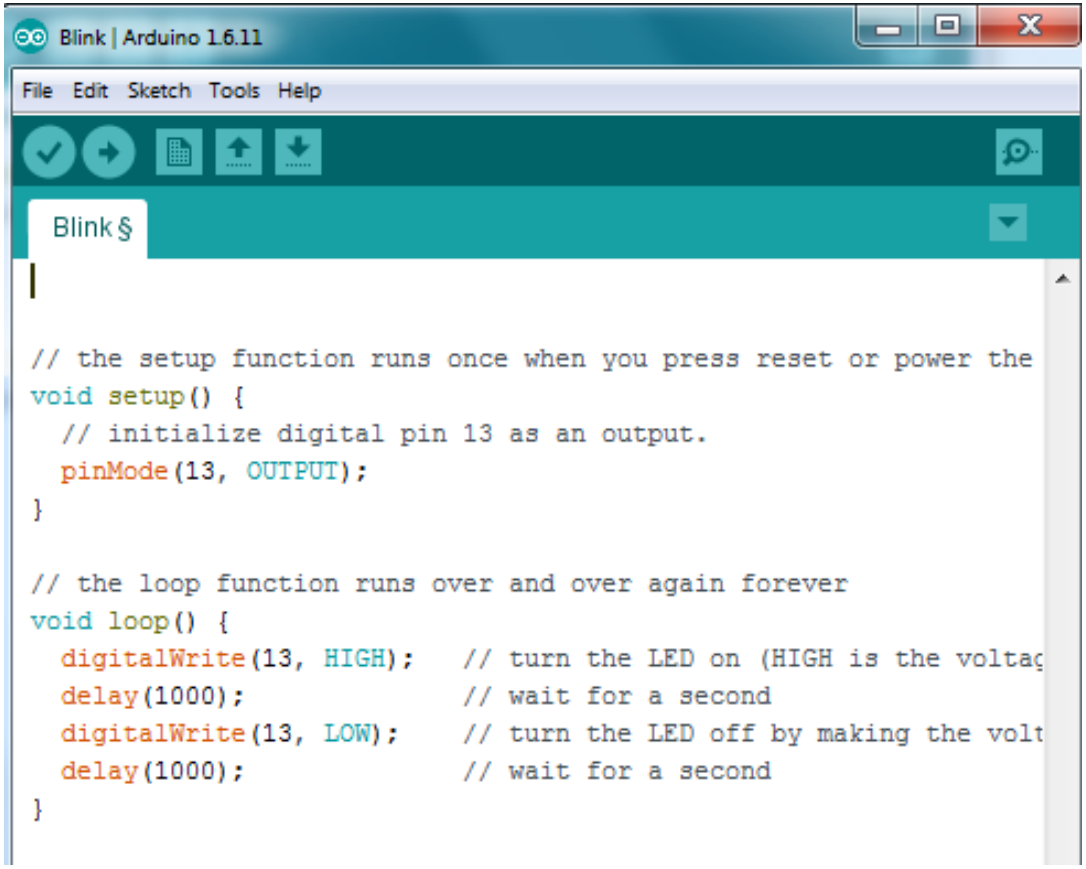
The setup() function is called once when your program starts. Use it to initialize pin modes, or begin serial. It must be included in a program even if there are no statements to run.

```
void setup()
{
  pinMode(pin, OUTPUT);      // sets the 'pin' as output
}
```

loop()

After calling the setup() function, the loop() function does precisely what its name suggests, and loops consecutively, allowing the program to change, respond, and control the Arduino board.

```
void loop()
{
  digitalWrite(pin, HIGH);   // turns 'pin' on
  delay(1000);               // pauses for one second
  digitalWrite(pin, LOW);    // turns 'pin' off
  delay(1000);               // pauses for one second
}
```


Blink | Arduino 1.6.11
File Edit Sketch Tools Help
Blink §
// the setup function runs once when you press reset or power the
void setup() {
// initialize digital pin 13 as an output.
pinMode(13, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltag
delay(1000); // wait for a second
digitalWrite(13, LOW); // turn the LED off by making the volt
delay(1000); // wait for a second
}


BareMinimum | Arduino 1.6.11
File Edit Sketch Tools Help
BareMinimum
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}

File Edit Sketch Tools Help

Upload

Basic_In_Out_Ain §

```
// the setup routine runs once when you press reset:

void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  pinMode(led, OUTPUT);
  pinMode(inPin, INPUT);
}
// the loop routine runs over and over again forever:

void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);
  // print out the value you read:
  reading = digitalRead(inPin);

// if (voltage < 2.5)   alterante to test voltage and not switch.

  if (reading == HIGH)
  {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  Serial.print ("High ");
  }
  else
  {
   digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
    Serial.print ("Low ");
  }

  Serial.println(voltage);
  delay (250);

}
```

Open source software is software that can be freely used, changed, and shared (in modified or unmodified form) by anyone. **Open source** software is made by many people, and distributed under licenses that comply with the **Open Source** Definition.

Lab_2_Demo_CAM8302E_F2016

```
/*  Lab_2_Part_0_Blink_CAM8302E_MH_F2016.ino                */
/*  September 15, 2016 , Michel Hanbury Section 030         */


#define WhtLed    5       // pin definitions for led (max. 40 mA)
#define BrdLed   13       // L on Arduino Board

#define Start_Sw 4        // 10k pull up required.
#define Stop_Sw  6        // 10k Pull up required.
```

```
// Convert the analog reading (which goes from 0 - 1023) to a voltage (0 to 5V):
float TempVolts = TempValue * (5.0 / 1023.0);   // float or real data type

Start_State = digitalRead(Start_Sw);  // read the input
Stop_State  = digitalRead(Stop_Sw);  //  read the input

if (!Start_State) // Wait for the PB to be pressed, causing it to go low.
   {
  digitalWrite(WhtLed, HIGH);  // turn on led if push button is pressed
  digitalWrite(BrdLed, HIGH);  // high turn on the led,pin is connected to anode.
   }
/*  || OR,  && AND,   ! NOT  */

if (TempVolts > 1.0 || !Stop_State) // turn off led if voltage > 1.5 volts or Stop_Sw is pressed.
   {
    digitalWrite(WhtLed, LOW);
    digitalWrite(BrdLed, LOW);
   }
```

Code that is run only once

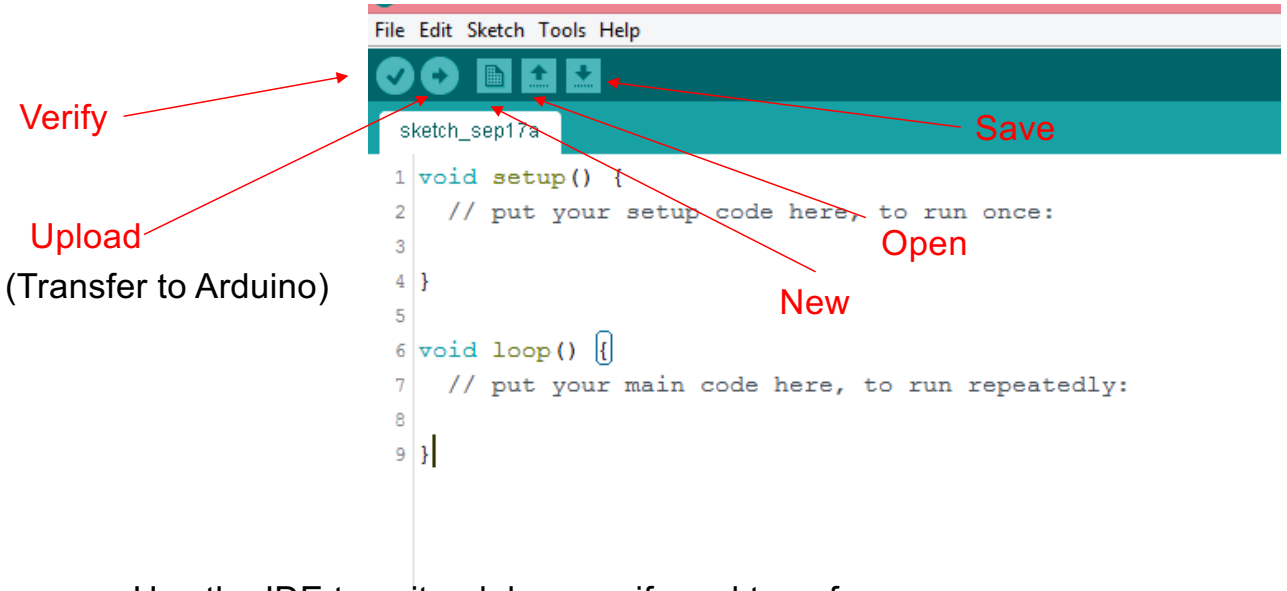
The code in the setup() function is executed once and is not part of a program loop.

```
void setup()
{
Serial.begin(9600);          // configure the serial port for 9600 bits / second

pinMode(Start_Sw,INPUT);     // define line input bit 4
pinMode(Stop_Sw,INPUT);
pinMode(WhtLed,OUTPUT);      // define line as output bit 5
pinMode(BrdLed,OUTPUT);
}
```

Software Example:

```
int TempValue = analogRead(A0);   // read analog input on CH A0 (16 bits)
int Start_State;                  // integer data type
int Stop_State;

// Convert the analog reading (which goes from 0 - 1023) to a voltage (0 to 5V):
float TempVolts = TempValue * (5.0 / 1023.0);   // float or real data type

Start_State = digitalRead(Start_Sw);  // read the input
Stop_State  = digitalRead(Stop_Sw);  //  read the input

if (!Start_State) // Wait for the PB to be pressed, causing it to go low.
   {
  digitalWrite(WhtLed, HIGH);  // turn on led if push button is pressed
  digitalWrite(BrdLed, HIGH);  // high turn on the led,pin is connected to anode.
   }
/*  || OR,  && AND,   ! NOT  */

if (TempVolts > 1.0 || !Stop_State) // turn off led if voltage > 1.5 volts or Stop_Sw is pressed.
   {
    digitalWrite(WhtLed, LOW);
    digitalWrite(BrdLed, LOW);
   }

Serial.print(TempVolts);  // display thermistor voltage to serial terminal
Serial.print("\t");   // tab
Serial.println("Volts");  //print voltage to screen
delay(100);  // wait 100 ms
```

Arduino IDE (Integrated Development Environment)



Use the IDE to write, debug, verify and transfer your program.

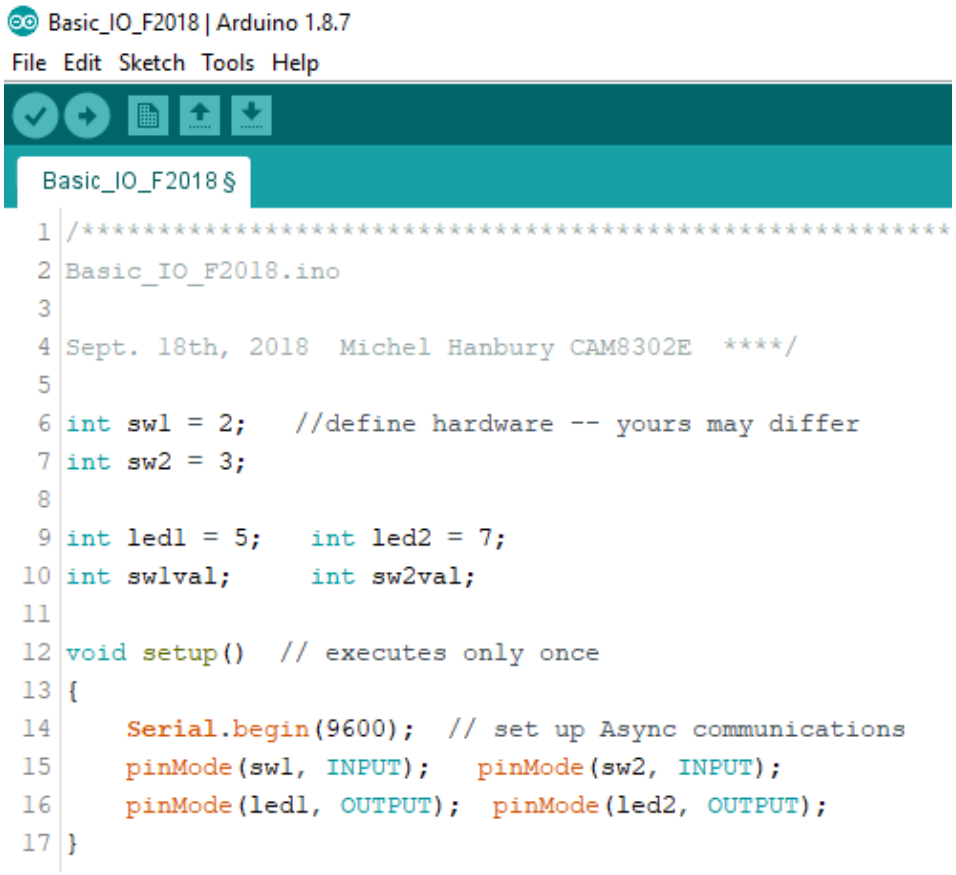
IDE -- Integrated Development Environment

Arduino Program (is called a Sketch): Definitions, Libraries, Setup, Loop

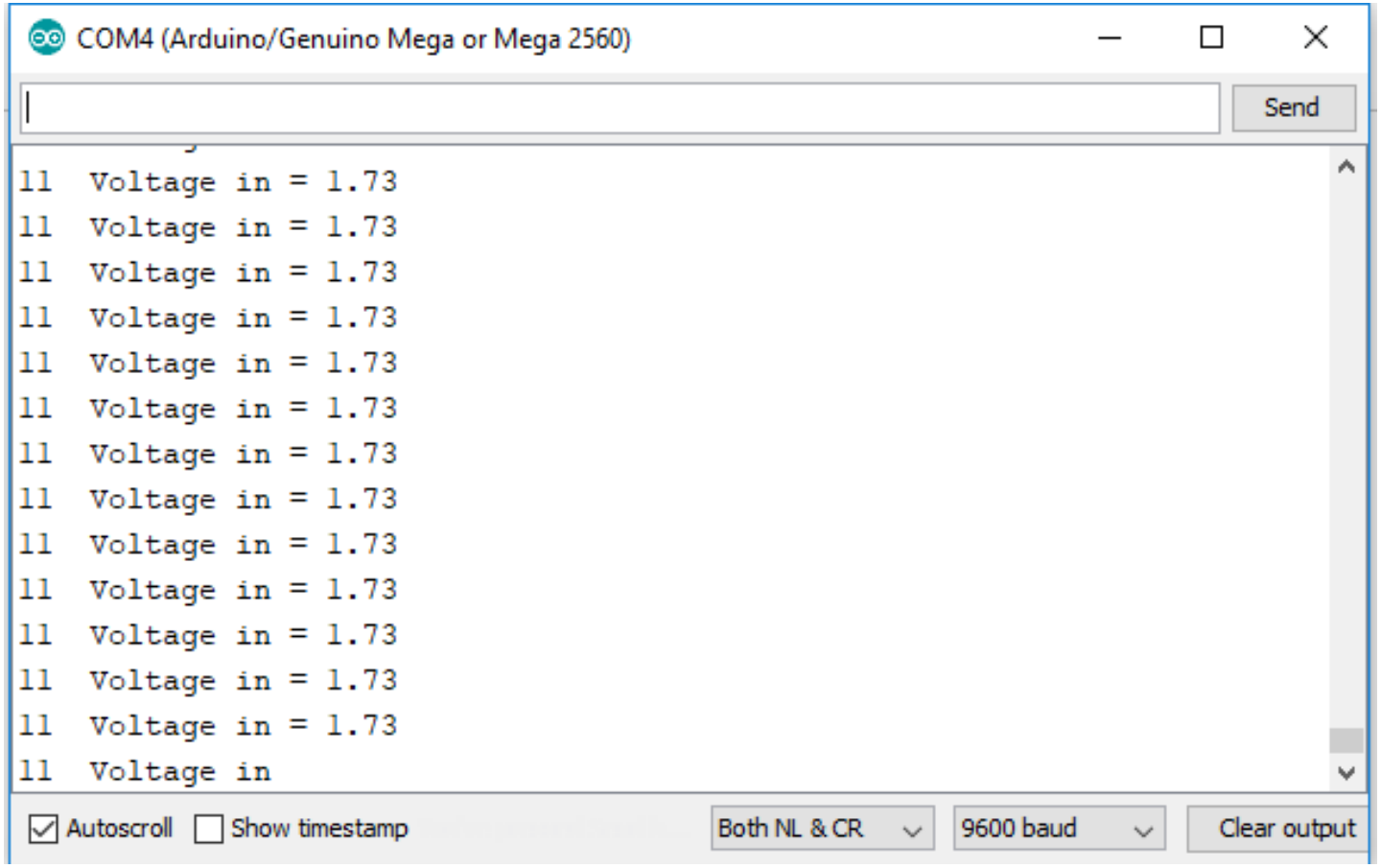
Lab_2_Demo_CAM8302E_MH §

```
1   /*  Lab_2_Part_0_Blink_CAM8302E_MH.ino */    // comment
2
3
4   #define GrnLed = 5       // pin definitions
5   #define RedLed = 6
6   #define Start_Sw = 4
7
8   void setup()
9   {
10  Serial.begin(9600);      // configure the serial port
11
12  pinMode(Start_Sw,INPUT);    // difine line
13  pinMode(RedLed,OUTPUT);
14  pinMode(GrnLed,OUTPUT);
15  }
16
```

```
lcd.setCursor(0,1);
lcd_key = read_LCD_buttons();


switch (lcd_key){

    case btnRIGHT:{
         lcd.print("RIGHT ");
         break;
    }
    case btnLEFT:{
          lcd.print("LEFT   ");
```


Basic_IO_F2018 | Arduino 1.8.7
File Edit Sketch Tools Help
Basic_IO_F2018 §
/***
Basic_IO_F2018.ino
Sept. 18th, 2018 Michel Hanbury CAM8302E ****/
int sw1 = 2; //define hardware -- yours may differ
int sw2 = 3;
int led1 = 5; int led2 = 7;
int sw1val; int sw2val;
void setup() // executes only once
{
Serial.begin(9600); // set up Async communications
pinMode(sw1, INPUT); pinMode(sw2, INPUT);
pinMode(led1, OUTPUT); pinMode(led2, OUTPUT);
}

```
18
19 void loop()  // runs endlessly
20 {
21   sw2val = digitalRead(sw2);  // read switch
22   swlval = digitalRead(swl);
23
24    int Photocell = analogRead(A0);     // 10 bit, 0-5 volts  4.9 mV resolution
25    float Vin = Photocell * (5.0 / 1023.0);
26
27 if (!swlval)  // test for low, pressed switch (switch pulled high)
28 {
29    digitalWrite(ledl, LOW);
30    digitalWrite(led2, HIGH);
31 }
32
33 if (!sw2val)
34 {
35     digitalWrite(ledl, HIGH);
36     digitalWrite(led2, LOW);
37 }
38 Serial.print(sw2val); Serial.print(swlval); Serial.print("  Voltage in = "); Serial.println(Vin);
39 }  // end of loop
```

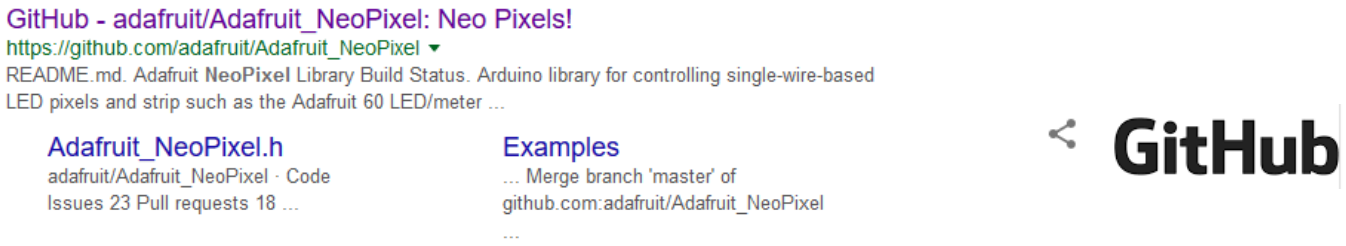

COM4 (Arduino/Genuino Mega or Mega 2560)
Send
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in = 1.73
ll Voltage in
Autoscroll
Show timestamp
Both NL & CR
9600 baud
Clear output

Arduino LCD Library

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

```
void setup(){

   lcd.begin(16, 2);              // start the library

}

void loop(){

```

```
lcd.setCursor(0,1);
lcd_key = read_LCD_buttons();


switch (lcd_key){

    case btnRIGHT:{
         lcd.print("RIGHT ");
         break;
    }
    case btnLEFT:{
          lcd.print("LEFT   ");
```



Many add-in libraries can be found on the GitHub web site.

These libraries are installed using the Arduino IDE.

The Arduino can call functions that are part of the library.