كلية الهندسة
قسم هندسة الروبوت والأنظمة الذكية

تطبيقات ميكاترونيك1

محاضرة 1

Microcontrollers
Arduino

د. فادي متوج

د.عيسى الغنام

2024-2023

توزيع الدرجات ومواعيد الاختبارات

النظري: 50   العملي: 50

اختبار1 في الأسبوع السادس	%15
اختبار2 في الأسبوع الثاني عشر	%15
مشروع تطبيقي بحثي يتضمن عرض تقديمي أمام لجنة حكم في الأسبوع الرابع عشر	%20
امتحان نهائي كتابي	%50

تحتاج للتجريب والتدريب والدراسة بشكل ذاتي منزلي وخلال الجلسات.

الموضوعات الرئيسة التي يغطيها المقرر

القسم الأول

Arduino

1- Getting started and Blinking with the Arduino

- Exploring the Arduino
- Creating our First Program

2- Digital Inputs, Outputs, and Pulse-Width Modulation

- Digital Outputs
- Pulse-Width Modulation with analogWrite()
- Reading Digital Inputs
- Building a Controllable RGB LED Nightlight

3- Reading Analog Sensors

- Understanding Analog and Digital Signals
- Reading Analog Sensors with the Arduino: analogRead()

4- Driving Motors

- Driving DC Motors
- Driving Servo Motors

الموضوعات الرئيسة التي يغطيها المقرر

القسم الأول

Arduino

5- USB and Serial Communication

- Understanding the Arduino's Serial Communication Capabilities
- Listening to the Arduino
- Talking to the Arduino

6- Interfacing with Liquid Crystal Displays

- Setting Up the LCD
- Using the LiquidCrystal Library to Write to the LCD
- Building a Personal Thermostat

7- Control Arduino Board using an Android Phone and a Bluetooth Module

- Getting Started with MIT App Inventor
- Creating Android App using MIT app inventor
- A Simple Project : Control LED using MIT App Inventor and Arduino

Raspberry Pi

Getting Started with Raspberry Pi

- Raspberry Pi setup & hardware overview
- Raspbian (Linux)
- Python, Hello in Python
- Raspberry Pi General Purpose IO (GPIO) Pins

الموضوعات الرئيسة التي يغطيها المقرر

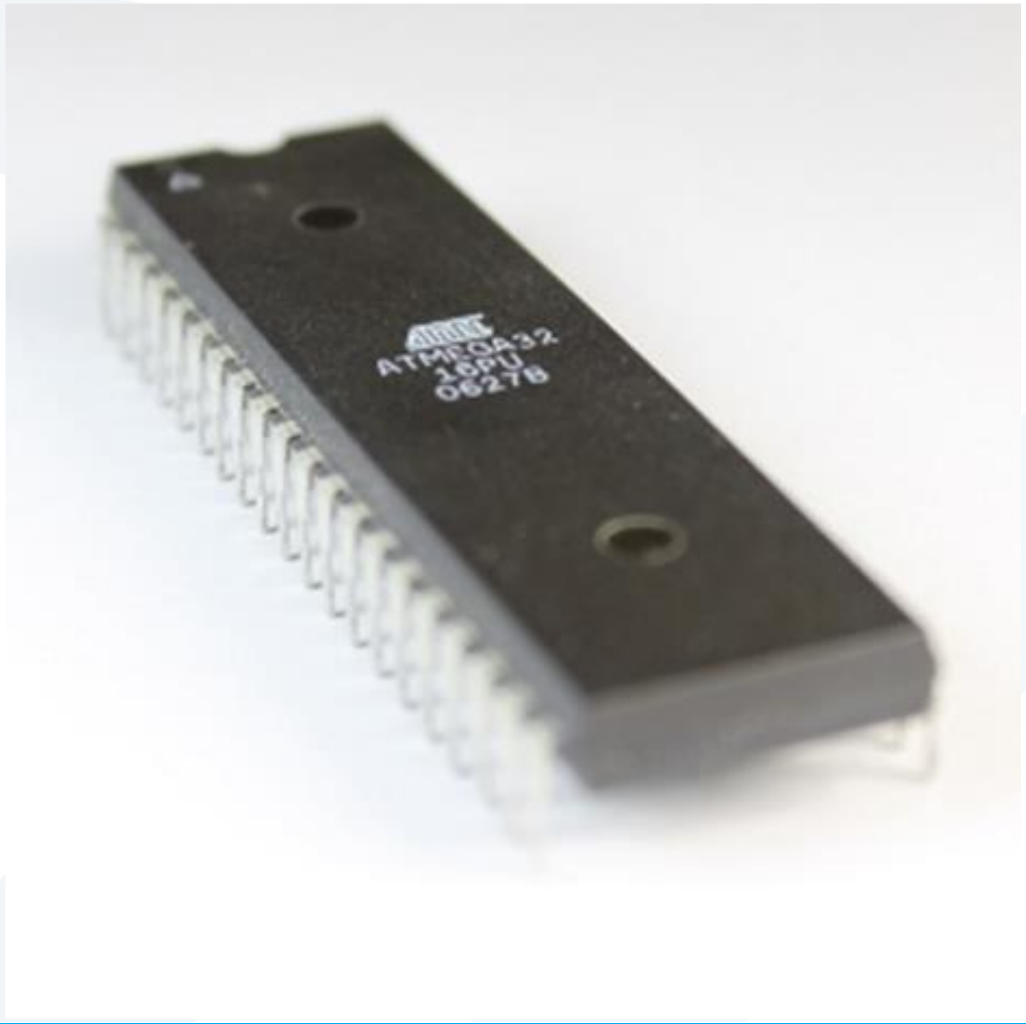
القسم الثاني

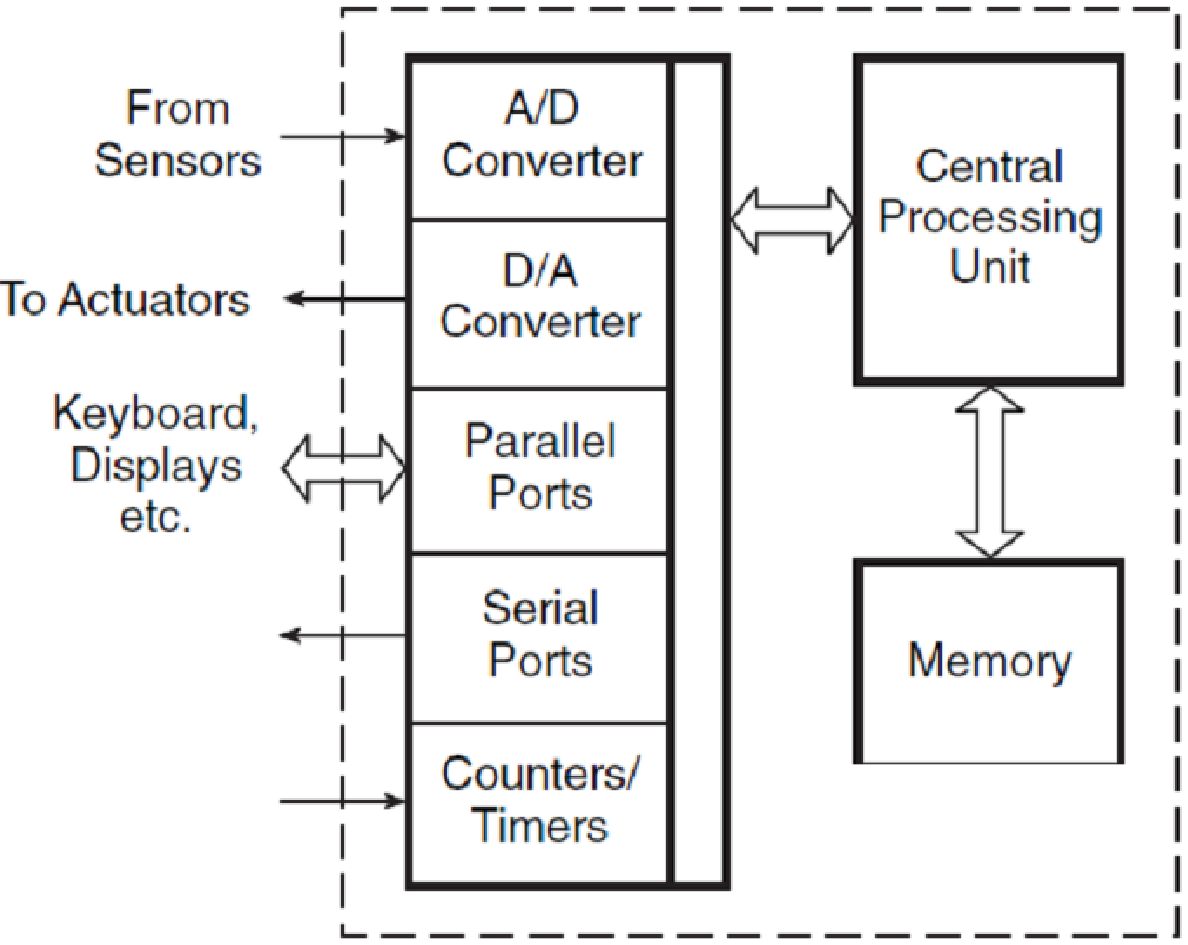
- برامج التصميم الإلكتروني Eagle and Altium Designer.
- المقاطعات والتحكم PID بمحرك تيار مستمر وبدرجة الحرارة
- التايمرات في المتحكمات وتوظيفها
- قيادة روبوت على عجلات تخطيط مسار وتحكم حركة
- ROS2

المشروع

- منافسة بين روبوتين للوصول إلى آخر المسار
- توزع النقاط عل التصميم وعلى التنفيذ والبرمجة وإنجاز المهام.
- استخدام تقنيات حساسات مركبة على روبوت تفاضلي بعجلات عادية للتعرف على بيئة مجهولة نسبيا.
- الروبوت مقيد بالمشي على مسار مرسوم (خط أسود)
- البيئة عبارة عن حلبة تحوي عوائق تكون موضوع بجانب أو على الخط الأسود بحيث تعيق حركة الروبوت
- العوائق عبارة عن علبة كرتونية لا يجب أن يصطدم الروبوت بها.
- عندما يصادف الروبوت عائق عليه العودة إلى تقاطع ليجرب مسار آخر حتى الوصول للأخير
- يجب على الروبوت تجاوز شرط فجائي معين.

Microcontrollers

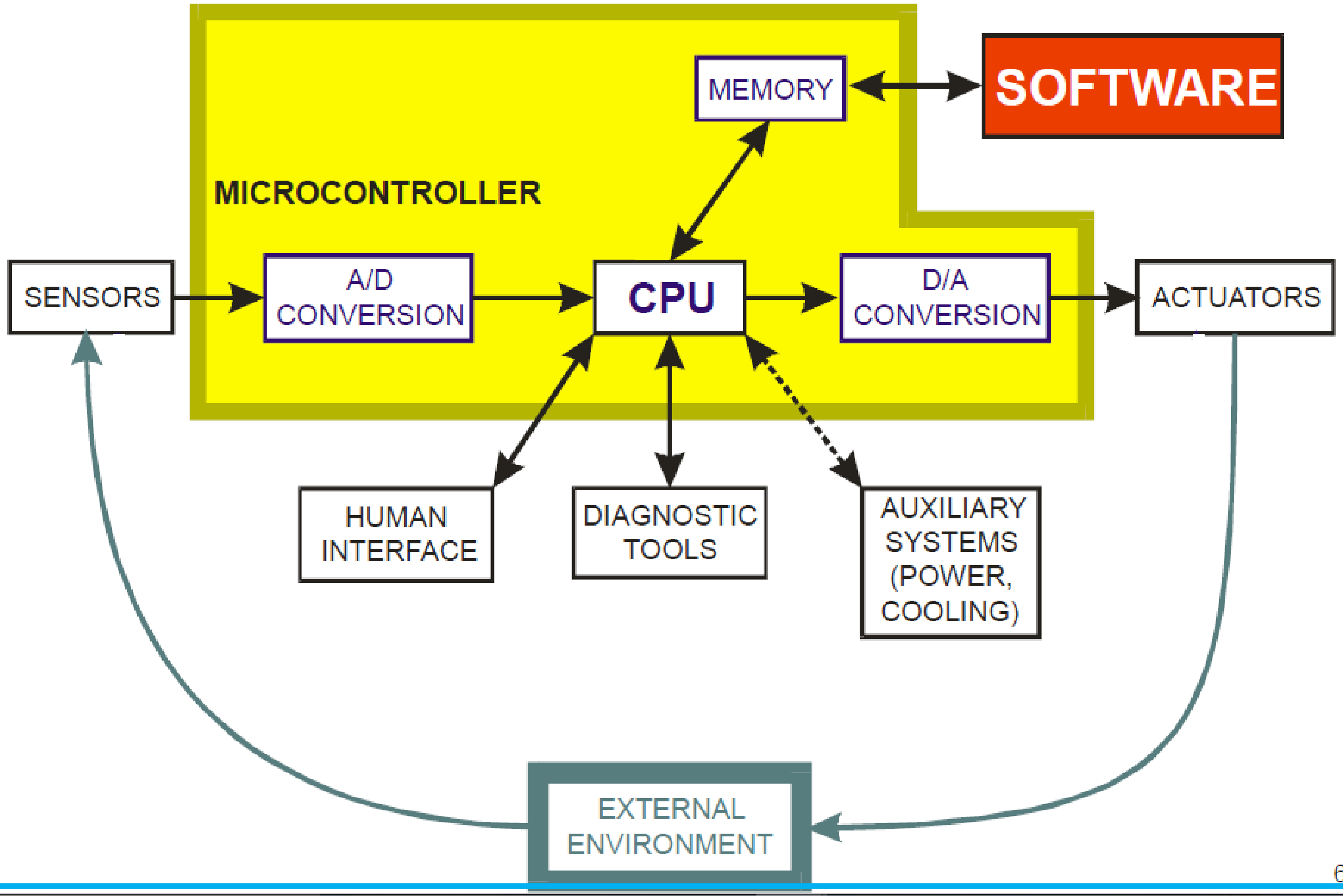




- Microcontrollers are found everywhere. You can find them in microwaves ovens, automobiles, televisions, etc.
- These microcontrollers control and sense the surrounding electronics and environment.
- The microcontroller is considered to be a computer on a chip. The microcontroller is able to execute a set of instructions in the form of a program.

Embedded System =
Computers Inside a Product

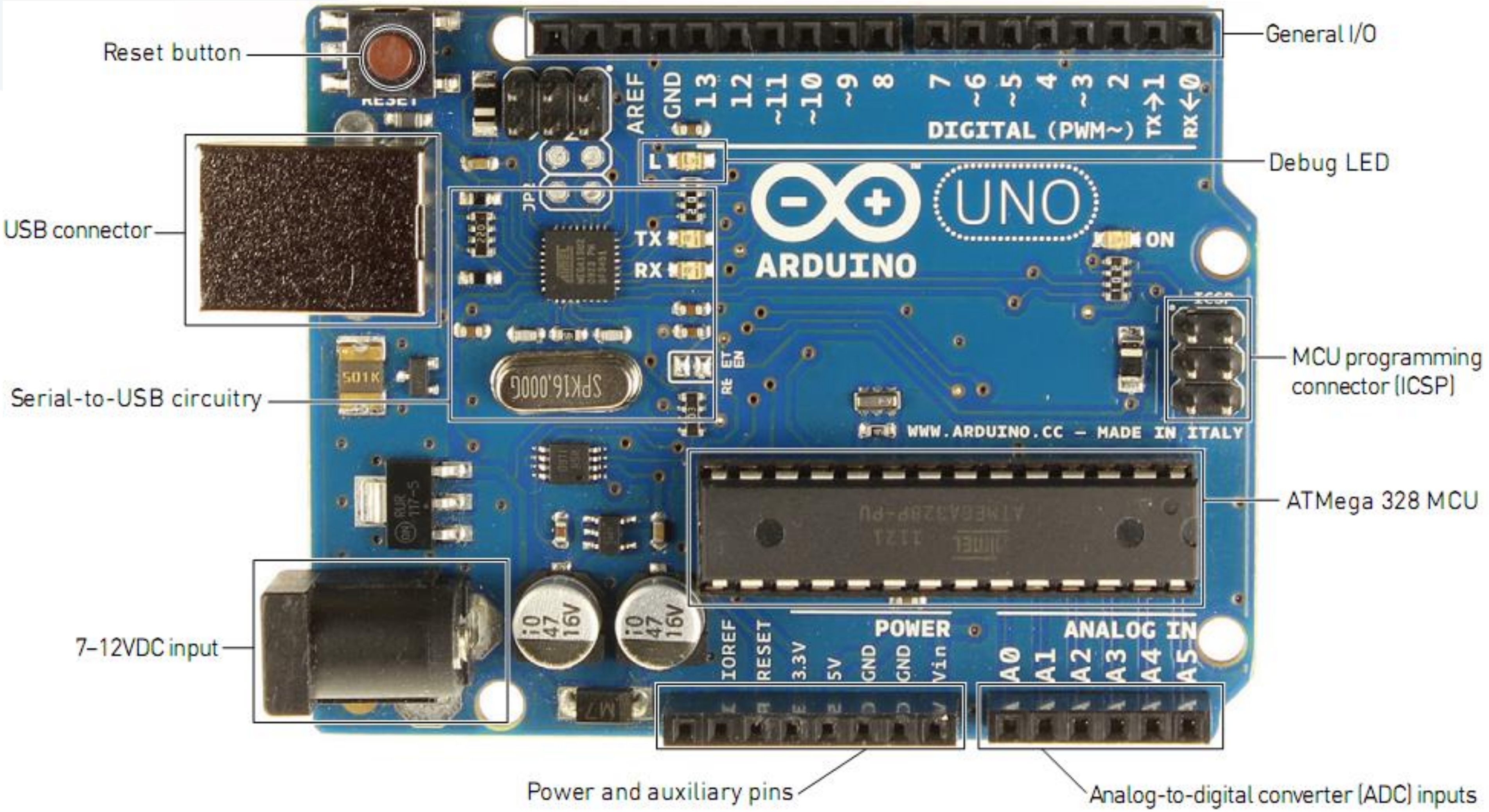
What is an embedded system?



What is an Arduino?

- Arduino platform is whatever you want it to be.
- The Arduino could be an automatic plant-watering control system. It can be a web server. It could even be a quadcopter autopilot.
- The Arduino is a microcontroller development platform paired with a programming language that we develop using the Arduino integrated development environment (IDE).
- By equipping the Arduino with sensors, actuators, lights, speakers, shields, and other integrated circuits, we can turn the Arduino into a programmable "**brain**" for just about any control system.

Basic components



Basic components

- Atmel microcontroller
- USB programming/communication interface(s)
- Voltage regulator and power connections
- I/O pins
- Debug, Power, and RX/TX LEDs
- Reset button
- In-circuit serial programmer (ICSP) connectors.

Atmel Microcontroller

- At the heart of every Arduino is an Atmel microcontroller unit (MCU)
- Most Arduino boards, including the Arduino Uno, use an **AVR ATMega** microcontroller.
- The Arduino Uno uses an ATMega 328p.
- The Arduino Due is an exception; it uses an **ARM Cortex** microcontroller.
- A 16 MHz ceramic resonator is wired to the ATMega's clock pins, which serves as the reference by which all program commands execute.
- You can use the Reset button to restart the execution of your program.
- Most Arduino boards come with a **debug LED** already connected to **pin 13**, which enables you to run your first program (blinking an LED) without connecting any additional circuitry.

Programming Interfaces

- Ordinarily, ATMega microcontroller programs are written in C or Assembly and programmed via the **ICSP** interface using a dedicated programmer.

AVR ISP MKII programmer

- Perhaps the most important characteristic of an Arduino is that we can program it easily via USB, without using a separate **programmer**.
- This functionality is made possible by the Arduino **bootloader**.
- The bootloader is loaded onto the ATMega at the factory (using the ICSP header), which allows a serial **USART** (Universal Synchronous/Asynchronous Receiver/Transmitter) to load our program on the Arduino without using a separate programmer.

The Arduino Bootloader

- A bootloader is a chunk of code that lives in a reserved space in the program memory of the Arduino's main MCU.
- In general, AVR microcontrollers are Programmed with an ICSP, which talks to the microcontroller via a serial peripheral interface (SPI).
- Programming via this method is fairly straight-forward, but necessitates the user having a hardware programmer such as an STK500 or an AVR ISP MKII programmer.

- In the case of the Arduino Uno and Mega 2560, a secondary microcontroller (an ATMega 16U2 or 8U2) serves as an interface between a USB cable and the serial USART pins on the main microcontroller.
- In older Arduino boards, an FTDI brand USB-to-serial chip was used as the interface between the ATMega's serial USART port and a USB connection.

General I/O and ADCs

- The general-purpose I/O pins can serve as digital inputs and outputs.
- The ADC pins can also act as analog inputs that can measure voltages between 0 and 5V
- Many of these pins serve additional functions. These special functions include various communication interfaces, serial interfaces, pulse-width-modulated outputs, and external interrupts.

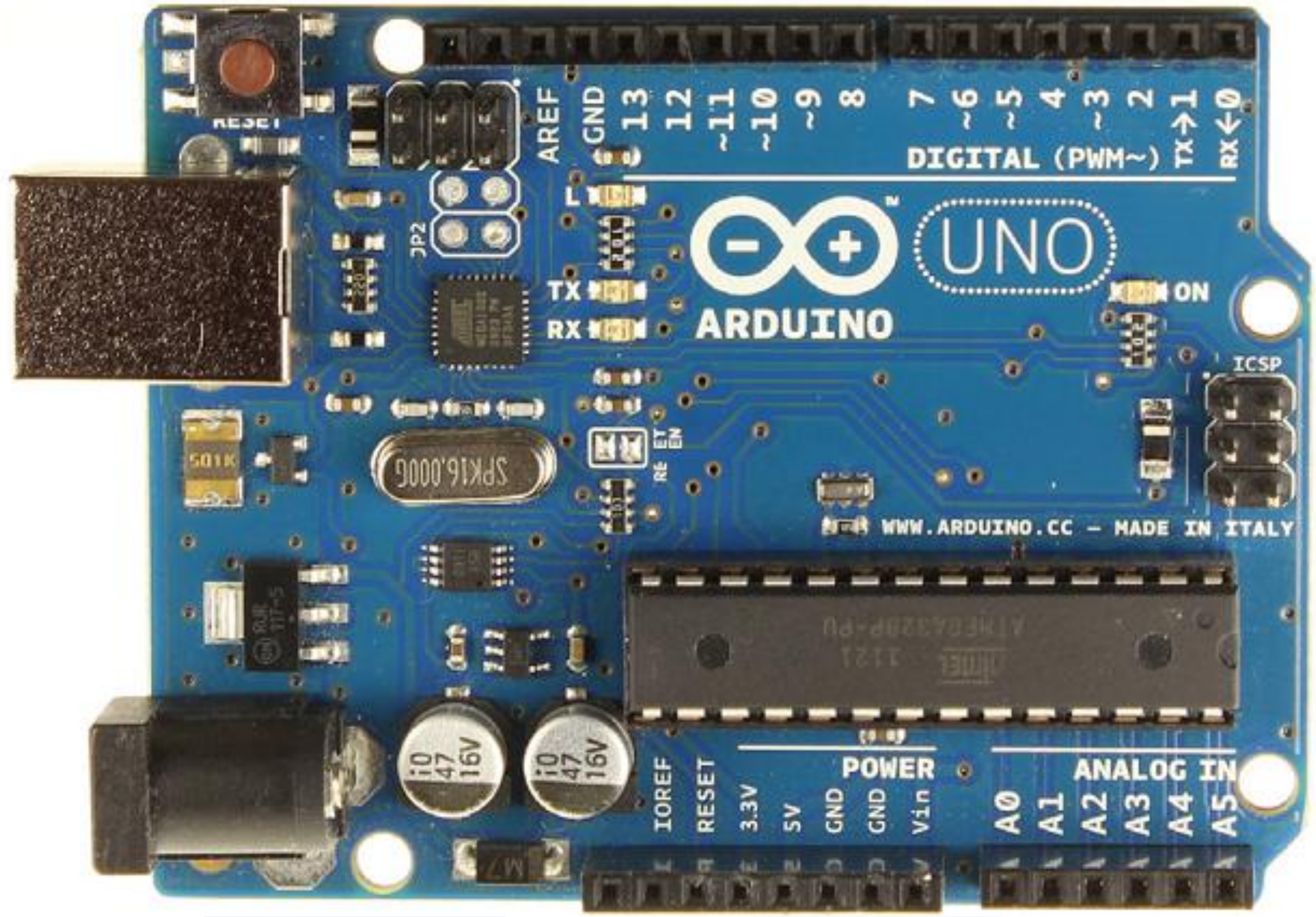
Power Supplies

- For the majority of your projects, you will simply use the 5V power that is provided over your USB cable.
- When you're ready to separate your project from a computer, you have other power options. The Arduino can accept between 6V and 20V (**7-12V recommend**) via the direct current (DC) barrel jack connector, or into the Vin pin.
- The Arduino has built-in 5V and 3.3V regulators:
- 5V is used for all the logic on the board. In other words, when you toggle a digital I/O pin, you are toggling it between 5V and 0V.
- 3.3V is broken out to a pin to accommodate 3.3V shields and external circuitry.

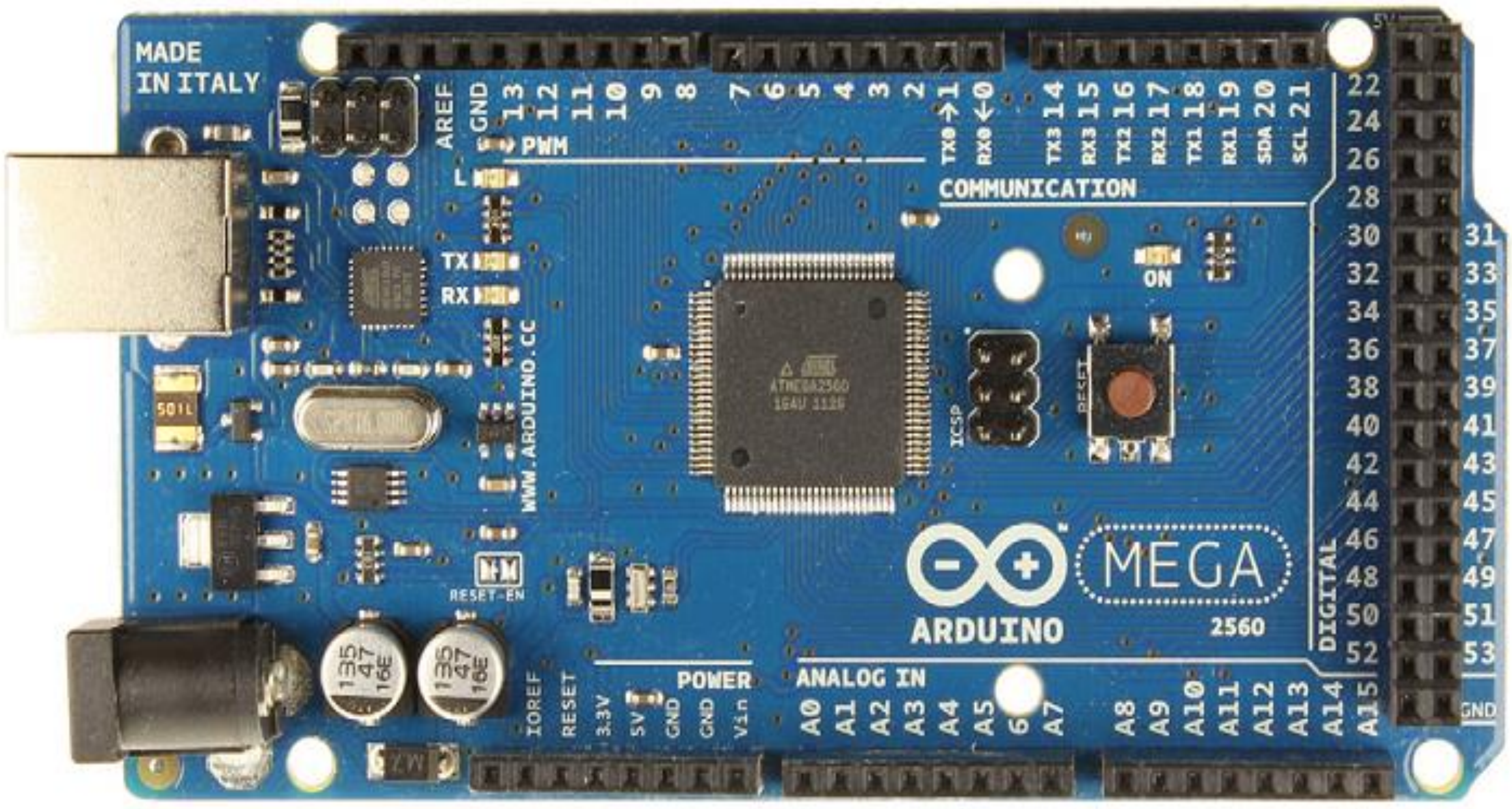
جَـامعة
المَـنارة
MANARA UNIVERSITY

Arduino Boards

Arduino Uno

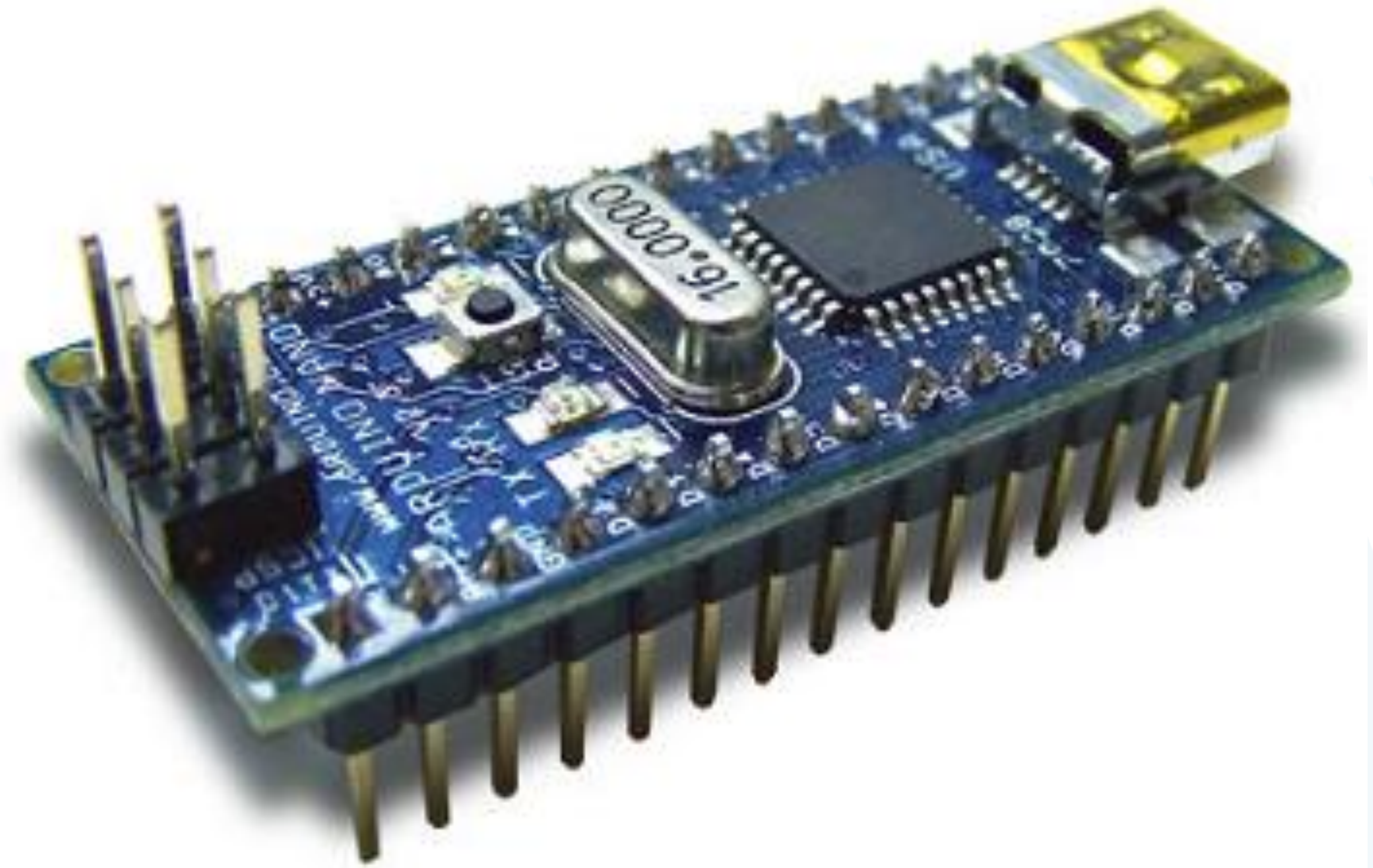

RESET
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
ICSP
JP2
WWW.ARDUINO.CC – MADE IN ITALY
IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
ANALOG IN
A0
A1
A2
A3
A4
A5

Arduino Mega 2560



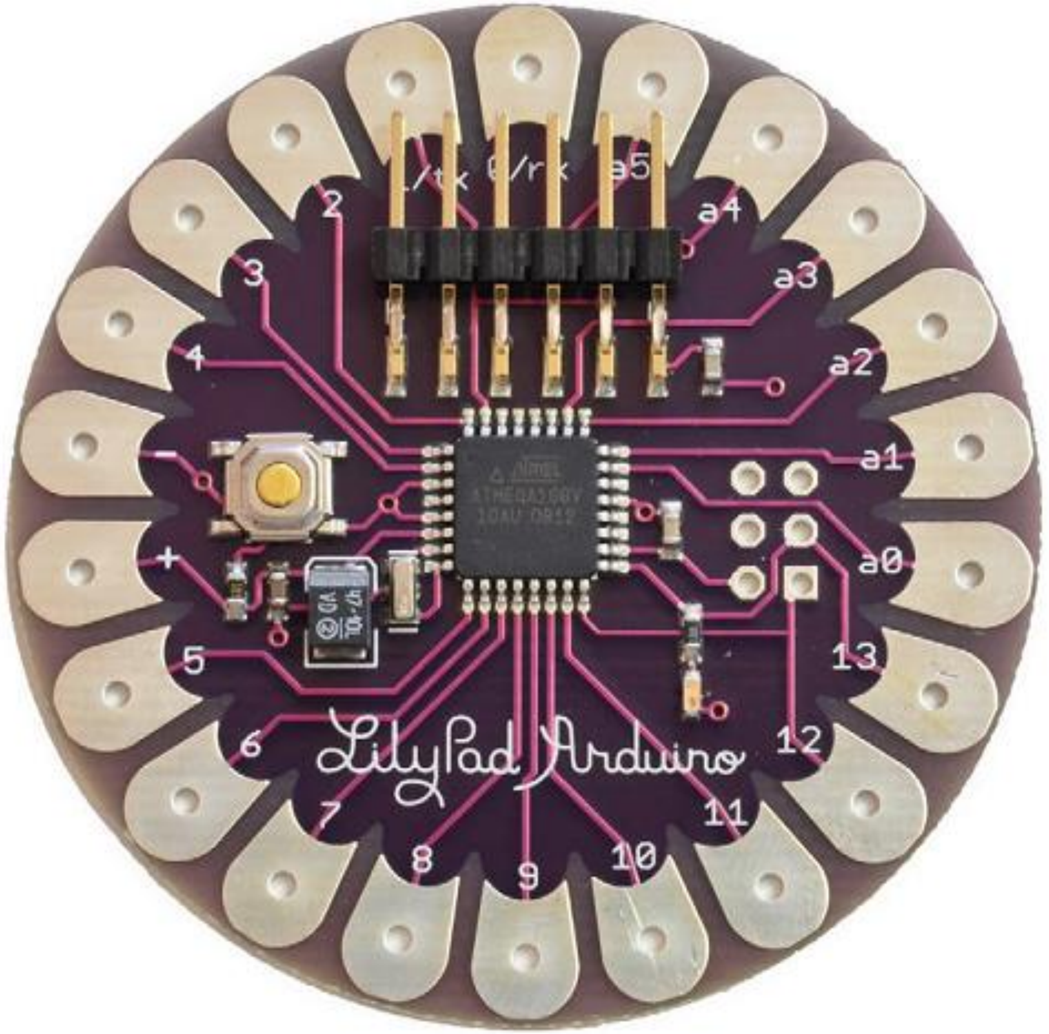
- The Mega 2560 employs an ATMega 2560 as the main MCU, which has 54 general I/Os to enable us to interface with many more devices.
- The Mega also has more ADC channels, and has four hardware serial interfaces (unlike the one serial interface found on the Uno)

Arduino Nano



- The Nano is designed to be mounted right into a breadboard socket.
- Its small form factor makes it perfect for use in more finished projects.

LilyPad Arduino



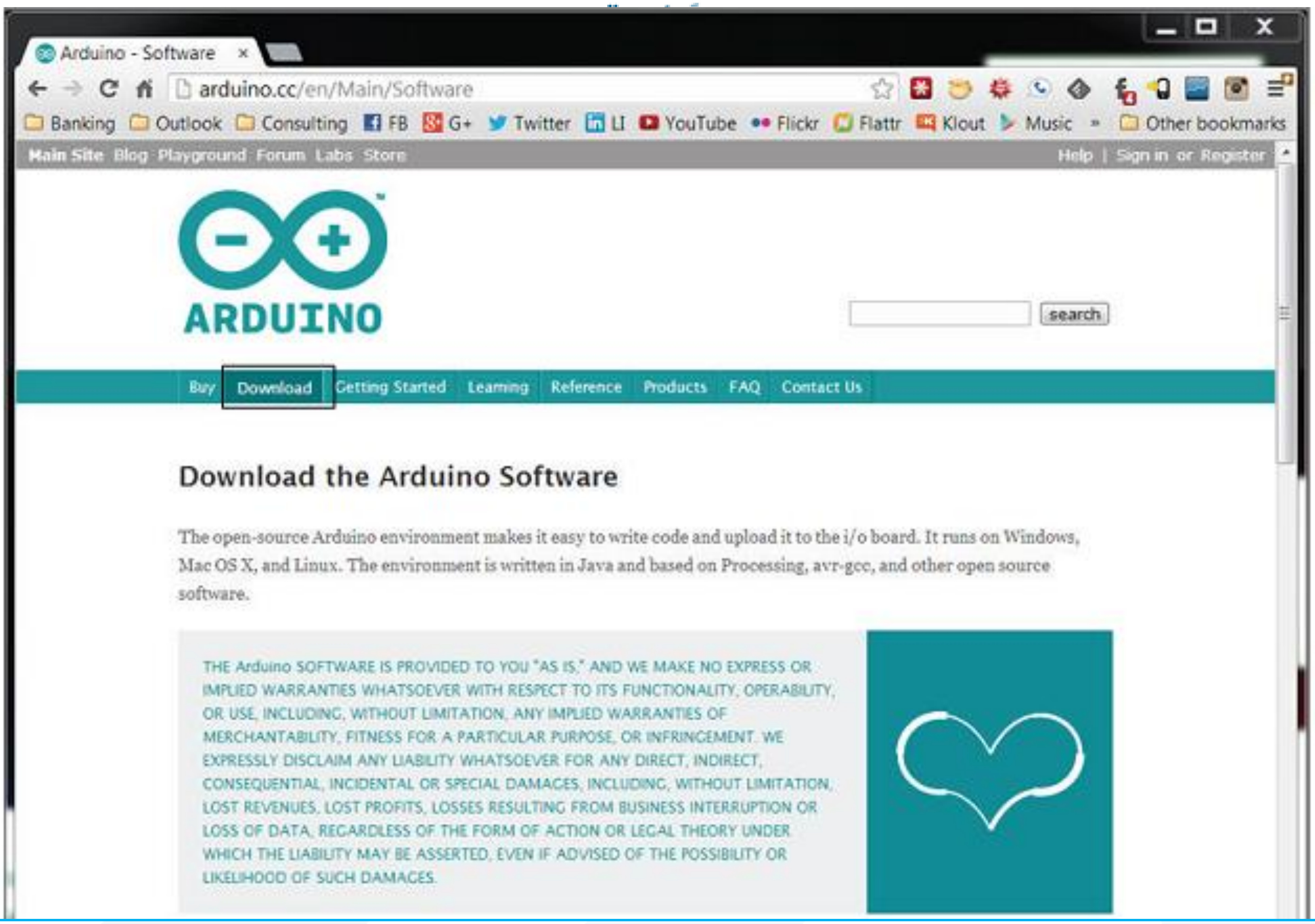
- The LilyPad is unique because it is designed to be sewn into clothing.
- Using conductive thread, we can wire it up to sewable sensors, LEDs, and more.
- To keep size down, we need to program it using an FTDI cable.

- The Arduino is **open source hardware**. As a result, we can find dozens and dozens of “Arduino compatible” devices available for sale that will work just fine with the Arduino IDE.
- Some of the popular third-party boards include the Seeduino, the adafruit 32U4 board, and the SparkFun Pro Mini Arduino boards.
- Many third-party boards are designed for very particular applications, with additional functionality already built in to the board. For example, the **ArduPilot** is an autopilot board for use in autonomous DIY quadcopters.

Quadcopter and ArduPilot Mega controller

Downloading and Installing theArduino IDE

- Access the Arduino website at www.arduino.cc and download the newest version of the IDE from the Download page.
- After completing the download, unzip it. Inside, you'll find the Arduino IDE.


Arduino - Software
arduino.cc/en/Main/Software
Banking Outlook Consulting FB G+ Twitter LI YouTube Flickr Flattr Klout Music Other bookmarks
Main Site Blog Playground Forum Labs Store
Help | Sign in or Register
ARDUINO
search
Buy Download Getting Started Learning Reference Products FAQ Contact Us

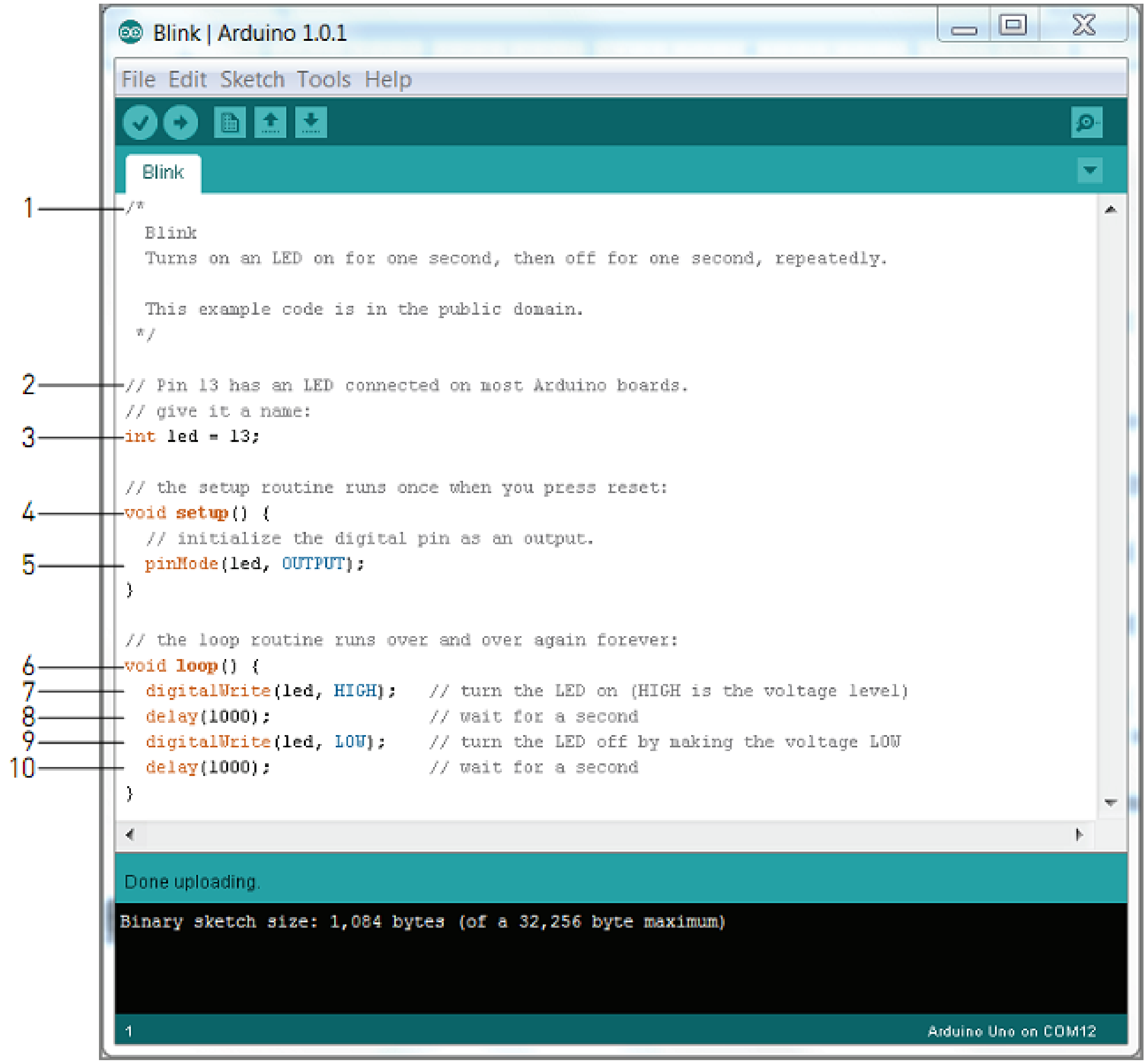
Download the Arduino Software

The open-source Arduino environment makes it easy to write code and upload it to the i/o board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing, avr-gcc, and other open source software.

THE Arduino SOFTWARE IS PROVIDED TO YOU "AS IS," AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEORY UNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY OR LIKELIHOOD OF SUCH DAMAGES.

Running the IDE and Connecting to the Arduino

Our First Program
Blink program



```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Done uploading.

```
Binary sketch size: 1,084 bytes (of a 32,256 byte maximum)
```

1. This is a multiline comment:
   - Comments are important for documenting your code. Everything you write between these symbols will not be compiled or even seen by your Arduino.
   - Multiline comments start with /* and end with */
   - Multiline comments are generally used when you have to say a lot (like the description of this program).
2. This is a single-line comment.
   - When you put // on any line, the compiler ignores all text after that symbol on the same line.
   - This is great for annotating specific lines of code or for “commenting out” a particular line of code that you believe might be causing problems.

3- This code is a variable declaration.

- A variable is a place in the Arduino's memory that holds information.
- Variables have different types. In this case, it's of type int, which means it will hold an integer. In this case, an integer variable called led is being set to the value of 13, the pin that the LED is connected to on the Arduino Uno.
- Throughout the rest of the program, we can simply use led whenever we want to control pin 13.
- Setting variables is useful because you can just change this one line if you hook up your LED to a different I/O pin later on; the rest of the code will still work as expected

4. **void setup()** is one of two functions that must be included in every Arduino program.
   - A function is a piece of code that does a specific task.
   - Code within the curly braces of the setup() function is executed once at the start of the program.
   - This is useful for one-time settings, such as setting the direction of pins, initializing communication interfaces, and so on.

5. The Arduino’s digital pins can function as input or outputs.
   - To configure their direction, use the command `pinMode()`.
   - This command takes two arguments.
   - The first argument to pinMode determines which pin is having its direction set.
   - The second argument sets the direction of the pin: **INPUT** or **OUTPUT**.
   - Pins are inputs by default, so we need to explicitly set them to outputs if we want them to function as outputs.

6. The second required function in all Arduino programs is `void loop()`.
   - The contents of the loop function repeat **forever** as long as the Arduino is on.
   - If we want our Arduino to do something once at boot only, we still need to include the loop function, but we can leave it empty.

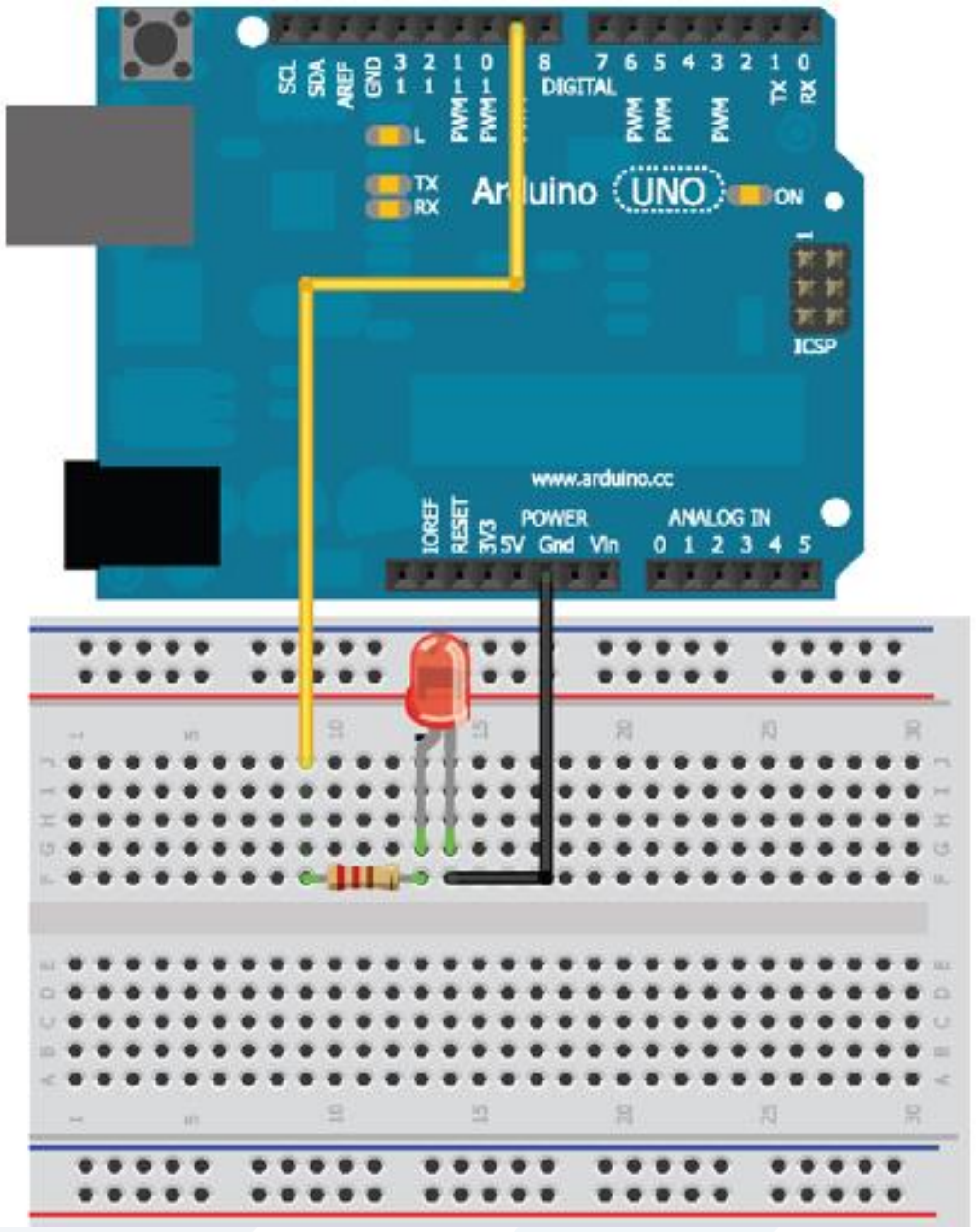
7. **digitalWrite()** is used to set the state of an output pin.

- It can set the pin to either **5V** or **0V**.
- When an LED and resistor is connected to a pin, setting it to 5V will enable you to light up the LED.
- The first argument to digitalWrite() is the pin we want to control.
- The second argument is the value we want to set it to, either **HIGH** (5V) or **LOW** (0V).
- The pin remains in this state until it is changed in the code

8. The **delay()** function accepts one argument: a delay time in milliseconds.
   - When calling delay(), the Arduino stops doing anything for the amount of time specified.
   - In this case, we are delaying the program for 1000ms, or 1 second.
   - This results in the LED staying on for 1 second before we execute the next command.
9. Here, **digitalWrite()** is used to turn the LED off, by setting the pin state to LOW.
10. Again, we delay for 1 second to keep the LED in the off state before the loop repeats and switches to the on state again

Digital Inputs, Outputs, and Pulse-Width Modulation

- LEDs are polarized; in other words, it matters in what direction we connect them. The positive lead is called the anode, and the negative lead is called the cathode.
- If we look at the clear top of the LED, there will usually be a flat side on the lip of the casing. That side is the cathode.
- Another way to determine which side is the anode and which is the cathode is by examining the leads. The shorter lead is the cathode.
- LEDs must always be wired in series with a resistor to serve as a current limiter. The larger the resistor value, the more it restricts the flow of current and the dimmer the LED glows.


SCL
SDA
AREF
GND
13
12
11
10
8
7 6 5 4 3 2 1 0
DIGITAL
PWM
TX
RX
L
Arduino UNO
ON
ICSP
www.arduino.cc
IOREF
RESET
3V3
5V Gnd Vin
POWER
ANALOG IN
0 1 2 3 4 5

Turning on an LED

```
const int LED=9;              //define LED for pin 9
void setup()
{
pinMode (LED, OUTPUT);   //Set the LED pin as an output
digitalWrite(LED, HIGH);    //Set the LED pin high
}
void loop()
{
//we are not doing anything in the loop!
}
```

LED with Changing Blink Rate

```
const int LED=9;      //define LED for Pin 9
void setup()
{
pinMode (LED, OUTPUT);     //Set the LED pin as an output
}
void loop()
{
for (int i=100; i<=1000; i=i+100)
{
digitalWrite(LED, HIGH);
delay(i);
digitalWrite(LED, LOW);
delay(i);
}
}
```

Using For Loops

The for loop declaration always contains three semicolon-separated entries:

■ The first entry sets the index variable for the loop. In this case, the index variable is i and is set to start at a value of 100.

■ The second entry specifies when the loop should stop. The contents of the loop will execute over and over again while that condition is true. <= indicates less than or equal to. So, for this loop, the contents will continue to execute as long as the variable i is still less than or equal to 1000.

■The final entry specifies what should happen to the index variable at the end of each loop execution. In this case, i will be set to its current value plus 100.

To better understand these concepts, consider what happens in two passes through the for loop:

1. i eqals 100.
2. The LED is set high, and stays high for 100ms, the current value of i.
3. The LED is set low, and stays low for 100ms, the current value of i.
4. At the end of the loop, i is incremented by 100, so it is now 200.
5. 200 is less than or equal to 1000, so the loop repeats again.
6. The LED is set high, and stays high for 200ms, the current value of i.
7. The LED is set low, and stays low for 200ms, the current value of i.
8. At the end of the loop, i is incremented by 100, so it is now 300.
9. This process repeats until i surpasses 1000 and the outer loop function repeats, setting the i value back to 100 and starting the process again.

Pulse-Width Modulation with analogWrite()

- Digital control of pins is great for blinking LEDs, controlling relays, and spinning motors at a constant speed.
- But what if we want to output a voltage other than 0V or 5V?
- Well, we can't—unless we are using an external digital-to-analog converter (**DAC**) chip.
- However, we can get pretty close to generating analog output values by using a trick called ***pulse-width modulation*** (**PWM**).

- On each Arduino, there are pins that can use the **analogWrite()**command to generate PWM signals that can emulate a pure analog signal when used with certain peripherals.
- These pins are marked with a ~ on the board.
- On the Arduino Uno, Pins **3, 5, 6, 9, 10,** and **11** are PWM pins.
- The **analogWrite()** command accepts two arguments: the pin to control and the value to write to it.
- The PWM output is an 8-bit value. In other words, we can write values from 0 to $2^8 - 1$, or 0 to 255.

Use PWM to change LED brightness

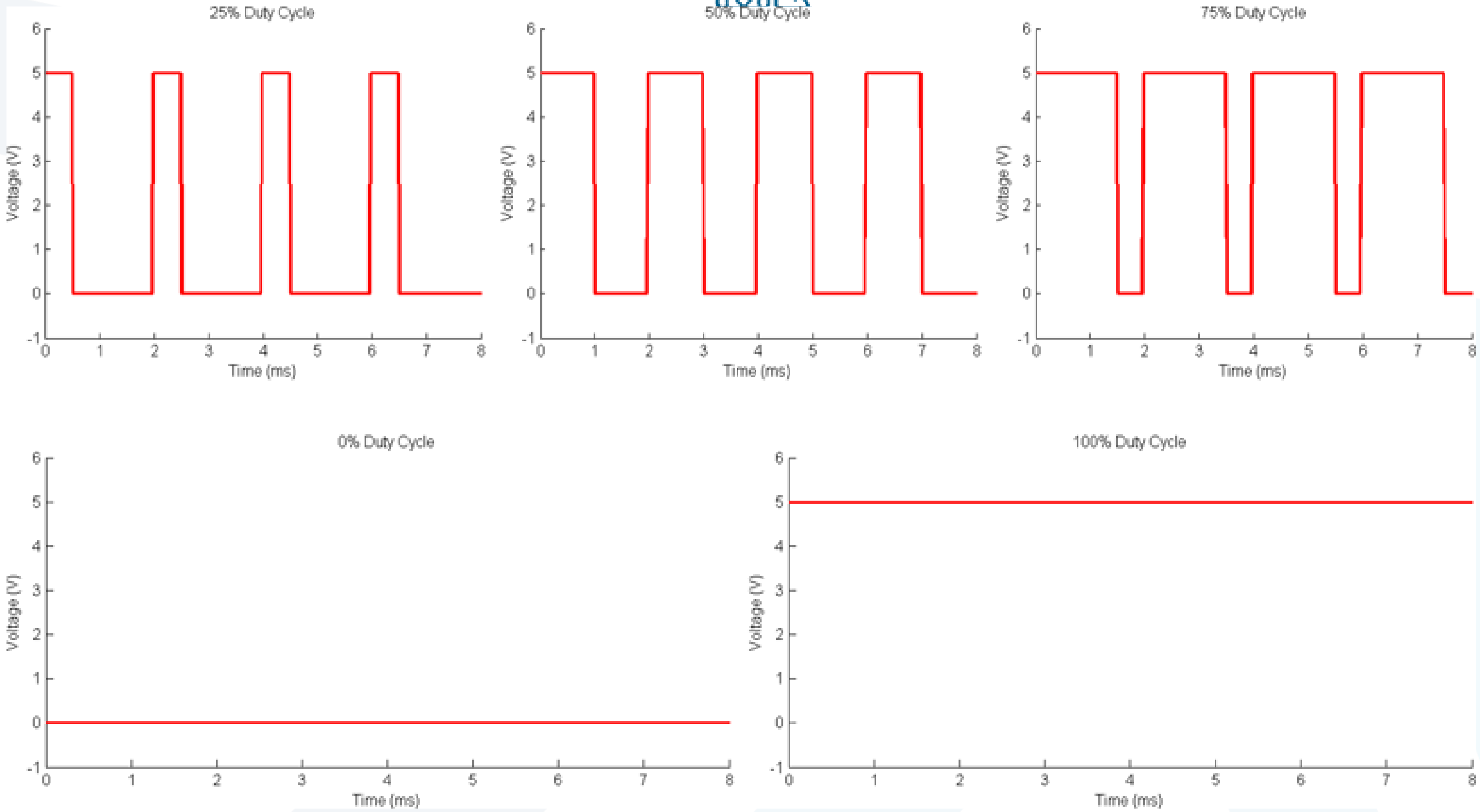
```
const int LED=9;    //define LED for Pin 9
void setup()
{
pinMode (LED, OUTPUT);     //Set the LED pin as an output
}
void loop()
{
for (int i=0; i<256; i++)
{
analogWrite(LED, i);
delay(10);
}
for (int i=255; i>=0; i--)
{
analogWrite(LED, i);
delay(10);
}}
```

What does the LED do when we run this code?

- we should observe the LED fading from off to on, then from on to off.
- Of course, because this is all in the main loop, this pattern repeats.

- PWM control can be used in lots of circumstances to **emulate** pure analog control, but it cannot always be used when we actually need an analog signal.
- For instance, PWM is great for driving direct current (DC) motors at variable speeds, but it does not work well for driving speakers unless we supplement it with some external circuitry.

- PWM works by modulating the **duty cycle** of a square wave.
- Duty cycle refers to the percentage of time that a square wave is high versus low.
- We are probably most familiar with square waves that have a duty cycle of 50%—they are high half of the time, and low half of the time.



PWM signals with varying duty cycles

The analogWrite()command sets the duty cycle of a square wave depending on the value we pass to it:

- Writing a value of **0** with analogWrite() indicates a square wave with a duty cycle of 0 percent (always low).
- Writing a **255** indicates a square wave with a duty cycle of 100 percent (always high).
- Writing a **127** indicates a square wave with a duty cycle of 50 percent (high half of the time, low half of the time).

- For a signal with a duty cycle of 25 percent, it is high 25 percent of the time, and low 75 percent of the time.
- The frequency of this square wave, in the case of the Arduino, is about **490Hz**. In other words, the signal varies between high (5V) and low (0V) about 490 times every second.

➢ So, if we are not actually changing the voltage being delivered to an LED, why do we see it get dimmer as we lower the duty cycle?

- It is really a result of our eyes playing a trick on us!
- If the LED is switching on and off every 1ms (which is the case with a duty cycle of 50 percent), it appears to be operating at approximately half brightness because it is blinking faster than our eyes can perceive. Therefore, our brain actually averages out the signal and tricks us into believing that the LED is operating at half brightness.