Embedded systems

From Wikipedia: An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured components of embedded systems.

- Limited computation power and memory
- Low power
- Small size
- Rugged operating range
- Low cost

- cell phones
- smart watches
- appliances
- automotive / avionics
- cameras and photography
- HVAC systems
- factory automation
- Internet of Things (IoT)

Analog vs. Digital

A key concept in beginning to design and use microcontrollers in embedded systems is understanding the difference between analog and digital signals.

Analog - a continuously varying energy or quantity of material. Much of how we interact with the surrounding physical world is analog in nature – sound, sight, smell, taste. We use sensors to convert physical analog quantities to analog voltages or currents. (Microphones, photosensors, temperature sensors, etc.) Just like the physical quantity, the voltage or current is defined at every point in time. Detecting an analog voltage or current requires requires precisely measuring the volts or amps at a particular time. A simple example of an analog signal is a sine wave.

Digital - The signal is either high or low (on/off or true/false or 0 /1). There are not many digital signals in the physical world. However, in terms of encoding information into signals, digital is much more robust. There is more leeway in interpreting whether a voltage is high or low. "High" means simply being "high enough" — above some threshold level. "Low" means simply being "low enough" — below some threshold level. Interpreting the information from a digital signal is less susceptible to errors because the voltages don't have to be known precisely.

The systems that we have developed to collect, manipulate, and transmit information work best by collecting the analog information from the surrounding physical environment, converting it digital form, processing the information digitally, and then converting back to analog (if needed).

Data Converters

In embedded systems, there is a constant need to transform information between analog and digital forms.

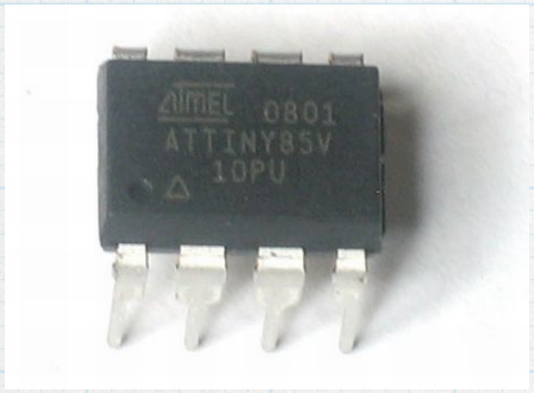
A sensor may be measuring some analog signal in the real world. For example, a temperature sensor would produce an analog voltage (or current) that is somehow proportional to the temperature in some system. An analog-to-digital converter (ADC) is needed to turn the temperature-dependent voltage into a digital number that can be used by the processor of the microcontroller.

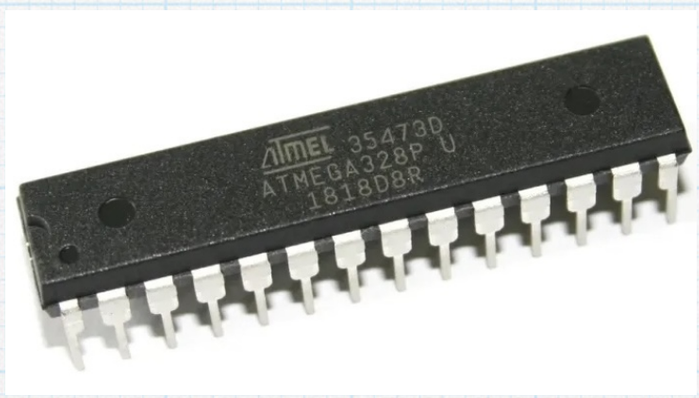
All microcontrollers have built-in ADCs to facilitate measurement of real-world signals. These can be viewed as little voltmeters.

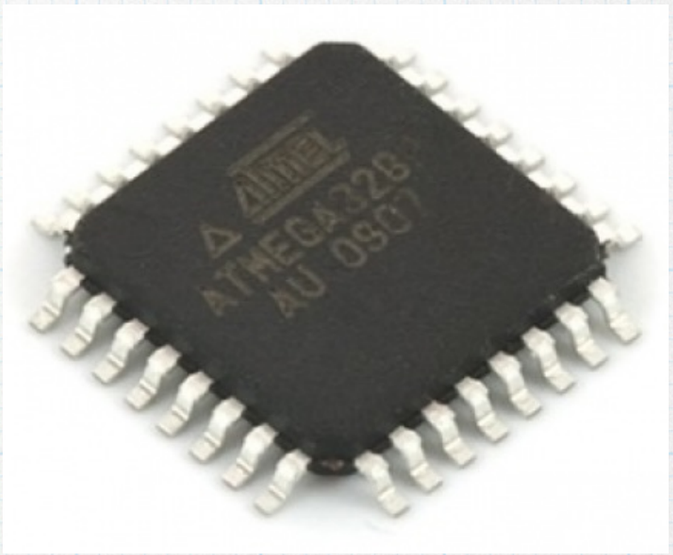
Sometimes, it may be necessary to convert digital information into an analog signal. (e.g. generating sounds.) This requires a digital-to-analog converter (DAC) circuit. Digital-to-analog conversion is less common in embedded systems. Having built-in DACs is more of an optional feature — some microcontrollers have them, some do not.

Microcontroller

A microcontroller is a one-chip computer. In a single package, it has the computing core and memory (of various types) needed to store and run small programs. There will be connections for power, clock control, digital I/O, analog in and maybe analog out.







Microcontroller

Microcontroller packages can have as few as 8 pins to as many 144 pins (or more). The basic pin functions can be broken into 3 categories.

Digital input/output

- Most of the pins of a microcontroller are digital I/O.
- These produce or detect “high” or “low” voltages that are interpreted as digital bits. The low, or logic 0, is usually close to ground (0 V) and high, or logic 1, is close to the power supply voltage (typically 3.3 V or 5 V).
- Digital pins can be configured as “input” to determine the logic level of voltage applied to the pin.
- Or they can be configured as “output” to produce a logic voltage on the pin to be applied to an external component.
- Digital pins can also be used to produce a pulse-width modulated (PWM) signal.
- If needed, the pin configuration can be changed as the program runs.
- Two pins can be used to set a serial communications channel. Multiple pins can be used to form a parallel channel.

Microcontroller

Analog input / (output):

- These measure then voltage of a component attached to the input. Each is an ADC, converting an analog voltage to a digital number.
- The input voltage must be between 0 V (ground) and the power-supply voltage (typically 3.3 V or 5 V). If the sensor produces voltages outside this range, some sort of intermediate voltage-shaping circuit would be needed to constrain the voltages applied to the input.
- If the sensor voltage is small — much more typical — then an amplifier may be needed in order to boost the voltage in order for the ADC to obtain an accurate reading.
- ADC resolution is determine by the number of bits in the digital number that is produced — 8, 10, 12, 14 bit ADCs are common in microcontrollers. For example, ATMega328 uses 10-bit ADCs, so the largest digital number is 1111111111 = 1023. If the voltage range is 5 V, then the resolution is 5 V ÷ 1023 = 4.9 mV.
- Generally, microcontrollers will have a “handful” of analog input pins. (ATMega328 has 6.)
- There may also be analog outputs, which require a DAC circuit. Analog output is often not essential, so not all microcontrollers included them.

Microcontroller

Power and clock pins:

- DC power and ground connections are essential. A larger chip may have multiple, redundant power and ground pins. Common power supply voltages are: 5 V, 3.3 V, 1.8 V.
- All computer systems need a clock signal in order to function.
- Most microcontroller have the option of generating a clock signal without external components.
- Larger chips probably have two pins available for connecting an external clock oscillator crystal. Using an external crystal will create a more accurate — and probably faster — clock signal. External crystals are fairly standard components with microcontrollers.

Sensors

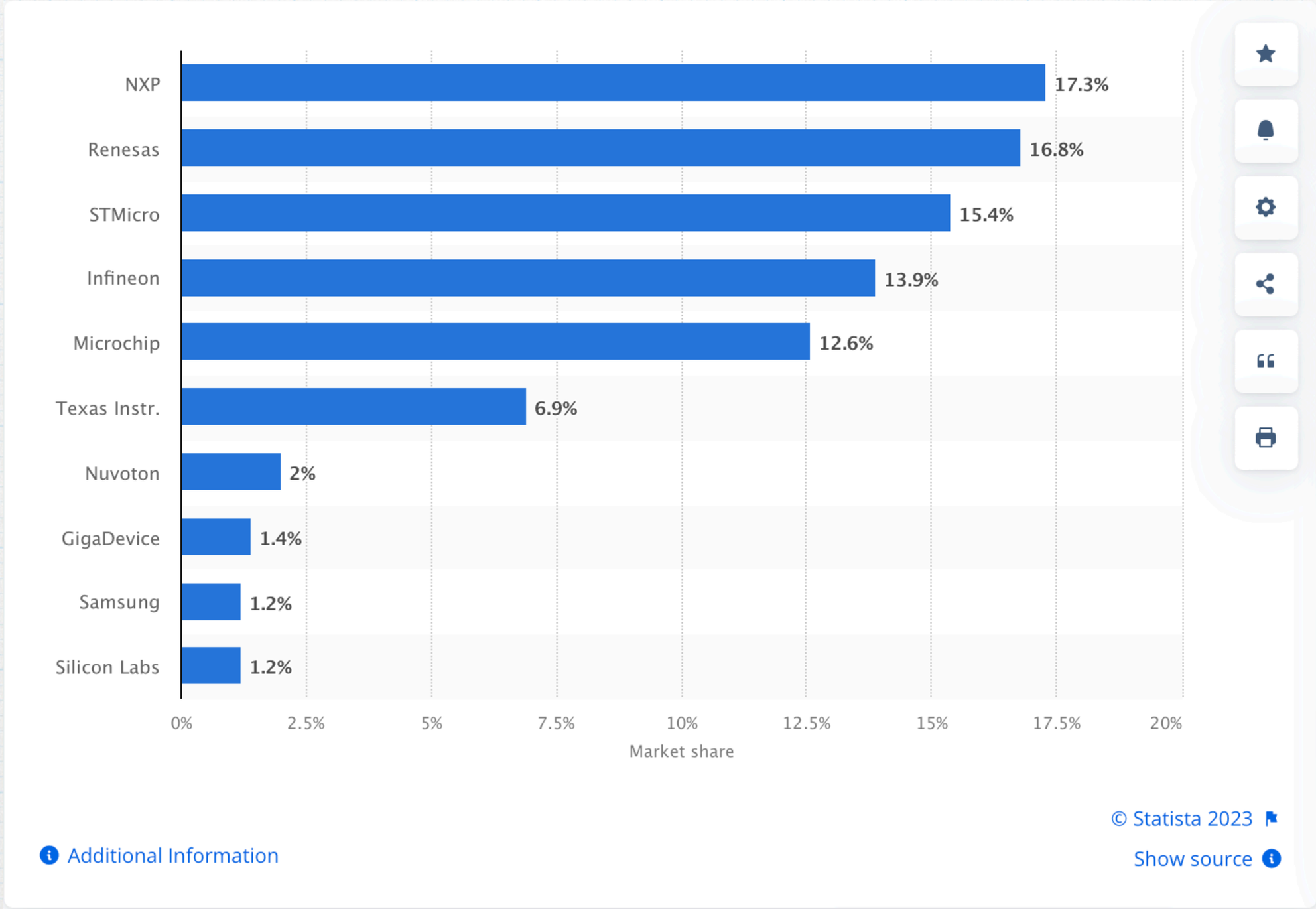
A key part of many embedded systems is measuring the surrounding environment. Sensors provide this type of input. There are many different kinds of sensors for measuring:

- voltage, current
- temperature (electronic, thermocouple, thermistor)
- humidity / pressure
- sound (microphone)
- light (photoresistor, photodiode, bolometer)
- distance (sonar, radar, lidar)
- mechanical motion (accelerometer, gyroscope)
- magnetic fields (coil, Hall-effect)
- chemical
- RFID tags

A bit of history

- **Microchip** was the original microcontroller company. The first controller was the PIC1650 in 1976. The company at that time was called General Instruments. Microchip was spun out as a separate company in the late 1980s. PIC originally stood for Peripheral Instrumentation Controller. Microchip sells more than 1 billion microcontrollers every year.
- **Atmel** was founded in 1984, specifically to compete in the microcontroller market. One of their major product line is the AVR family of controllers. Atmel gained lots of followers because it's chips were used in Arduino. Recently, Atmel was bought by Microchip.
- **ARM** is separate microcontroller architecture owned by ARM Holdings (from Great Britain) that is licensed by many companies. It is used in all smartphones and finds increasing use elsewhere.

Microcontroller manufacturers by market share - 2021



source: https://www.statista.com/statistics/1327509/top-mcu-suppliers-worldwide/

Microcontroller consideration

There are literally hundreds of microcontrollers available from the various companies. Things to consider when choosing.

- Price
- Architecture — 8-, 16-, 32, 64-bit
- Clock frequency — internal or with external oscillator crystal
- Memory — flash, static RAM
- Digital I/O pins
- Analog input — number of bits in ADC output
- PWM channels
- Package size and type
- Power requirements
- Analog output (optional)
- Built-in communications protocols

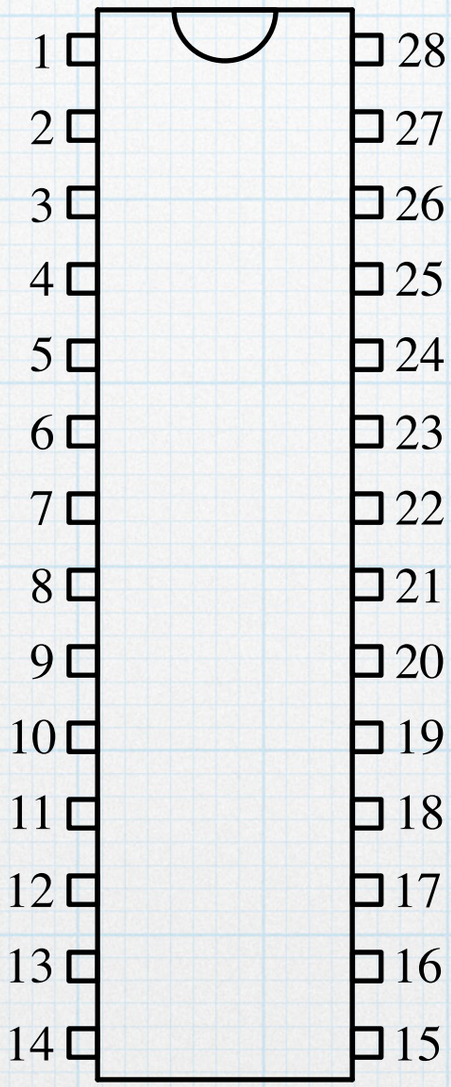
Atmega328

- Developed by Atmel. (Later acquired by Microchip)
- Used in Arduino Uno R3 (classic)
- 8-bit AVR architecture
- 8 – 20 MHz clock frequency (16 MHz is typical)
- 32 kbytes of flash memory (program storage)
- 2 kbytes of static memory
- 28-pin or 32-pin package
- 14 digital I/O
- 6 analog input (no analog out)
- 6 PWM outputs (part of the 14 digital I/0)

Pin assignment for Atmega328

Pin	data sheet	Arduino	function
1	PC6	Reset	Reset
2	PD0	digital 0	RX
3	PD1	digital 1	TX
4	PD2	digital 2	
5	PD3	digital 3	also PWM
6	PD4	digital 4	
7	VCC	VCC	power
8	GND	GND	ground
9	PB6	crystal	oscillator
10	PB7	crystal	oscillator
11	PD5	digital 5	also PWM
12	PD6	digital 6	also PWM
13	PD7	digital 7	
14	PB0	digital 8	

28-pin through-hole package



Pin	data sheet	Arduino	function
15	PB1	digital 9	also PWM
16	PB2	digital 10	also PWM
17	PB3	digital 11	MOSI, PWM
18	PB4	digital 12	MISO
19	PB5	digital 13	SCK
20	AVCC	VDD	power
21	AREF	analog ref	
22	GND	GND	ground
23	PC0	analog 0	ADC 0
24	PC1	analog 1	ADC 1
25	PC2	analog 2	ADC 2
26	PC3	analog 3	ADC 3
27	PC4	analog 4	ADC 4
28	PC5	analog 5	ADC 5

Developing embedded systems

Developing an embedded systems requires:

1. Hardware design. This includes the microcontroller, of course, along with any sensors, amplifiers, switches, displays, etc. Also requires design of the power system.
2. Software design. This code will run on the microcontroller and implement the desired system functionality. Usually written on a regular computer and then transferred to the microcontroller.

To facilitate embedded development, microcontroller vendors will provide:

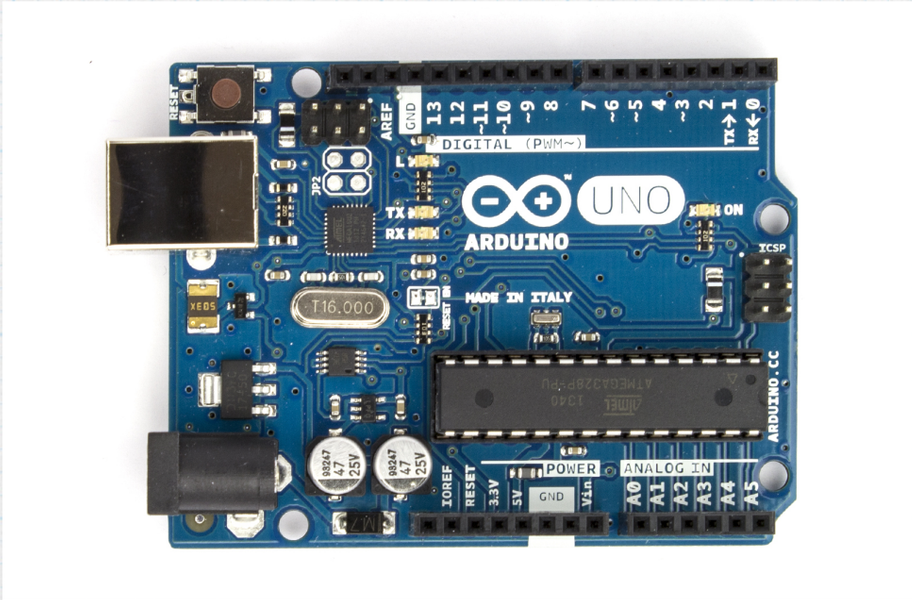
1. A hardware prototyping kit. This includes the microcontroller, power supply and connectors for easy connection to external portions of the hardware design.
2. A software Integrated Development Environment (IDE). Runs on a Windows, Mac, or Linux computer. Provides for compiling, error checking, and a channel to transfer completed code to the microcontroller.

These development tools can be expensive to purchase and might be difficult to use. Definitely intended for professional engineers. Students and hobbyists need not apply.

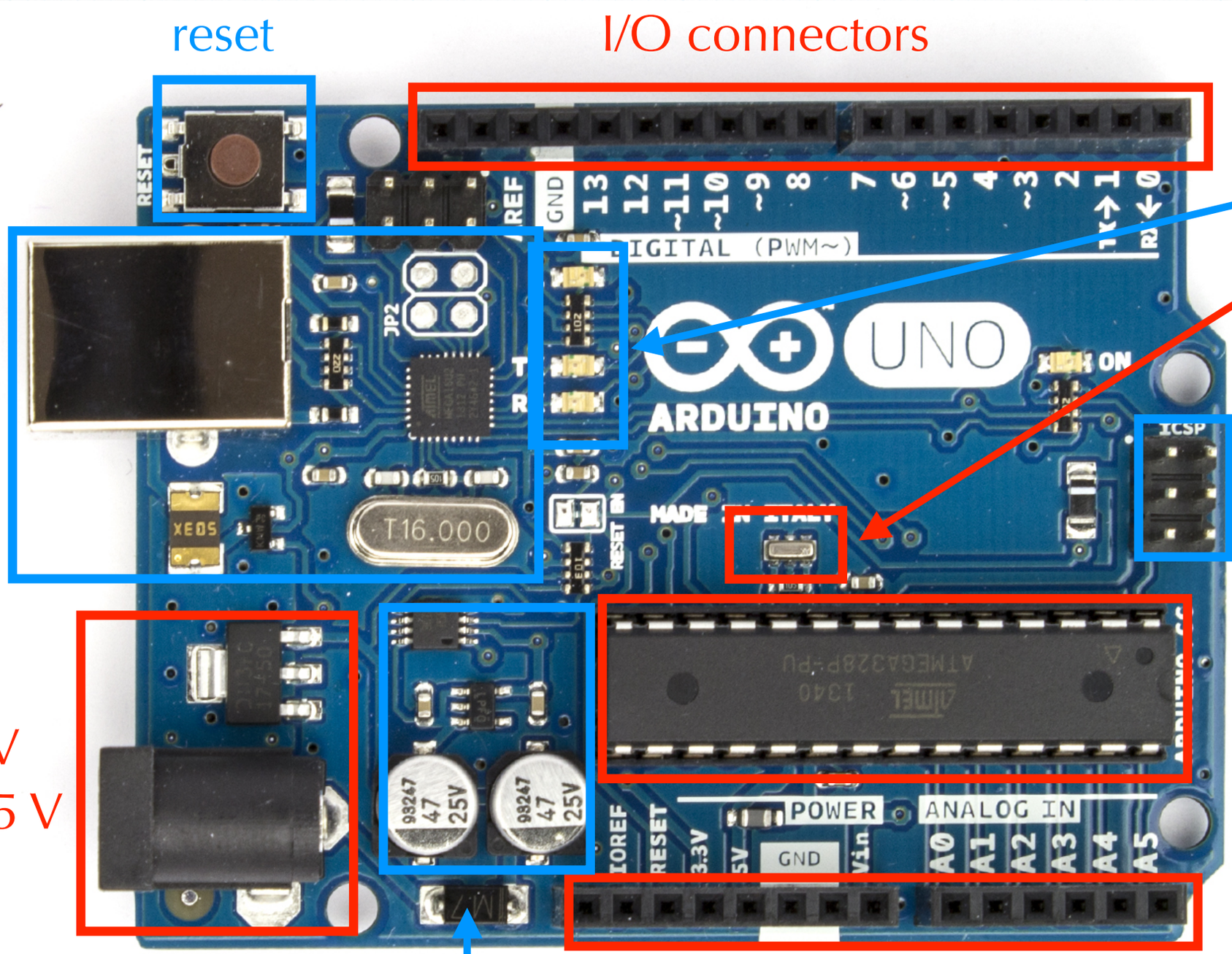
Enter Arduino

In 2005, a group at the Interactive Design Institute Ivrea in Ivrea, Italy put together a hardware / software platform that would make it much easier to build embedded systems prototypes. (The titular leader of the group was — and still is — Massimo Banzi, although there were four other founders.) They named the platform *Arduino*, after a bar that they frequented. They were focusing on making a system on which students and hobbyists could learn easily and would be affordable to non-professionals. Their efforts unleashed the current "maker" movement.

There have been several iterations on the basic platform over the years. For several years, the standard hardware arrangement is the Arduino Uno R3. (That may be about to change.)

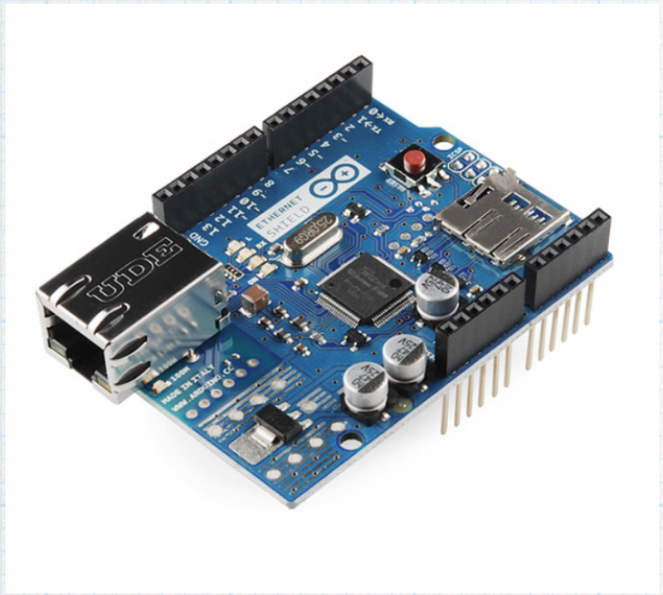


Arduino Uno hardware

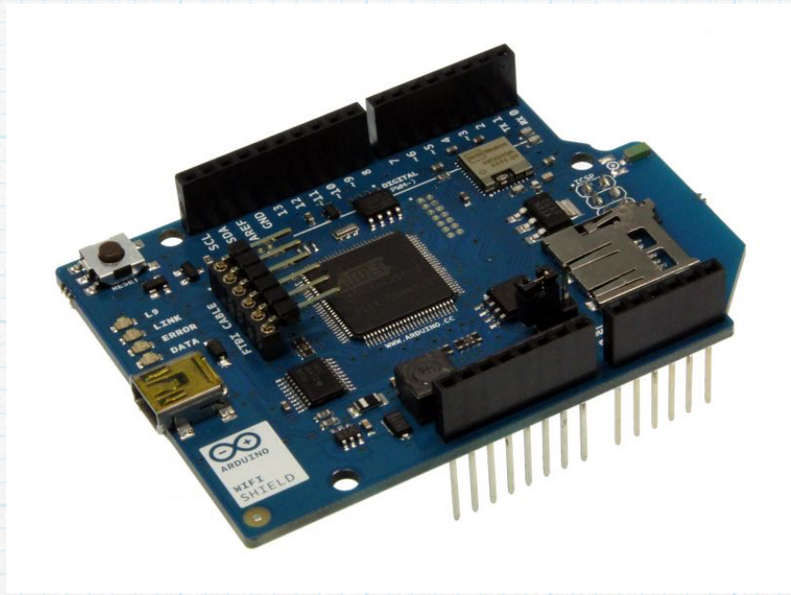


Also power and ground connections.
Can power 5-V and 3.3-V peripherals.

Hardware expansion through “shields”



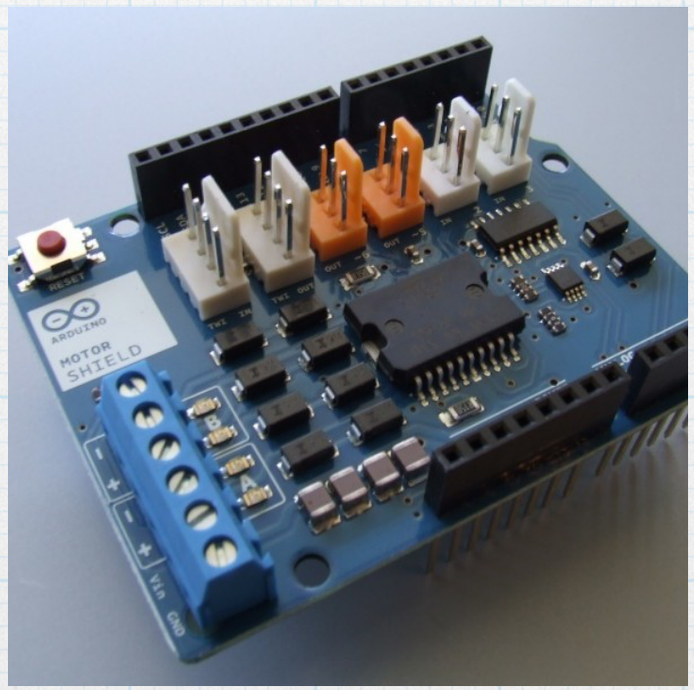
Ethernet



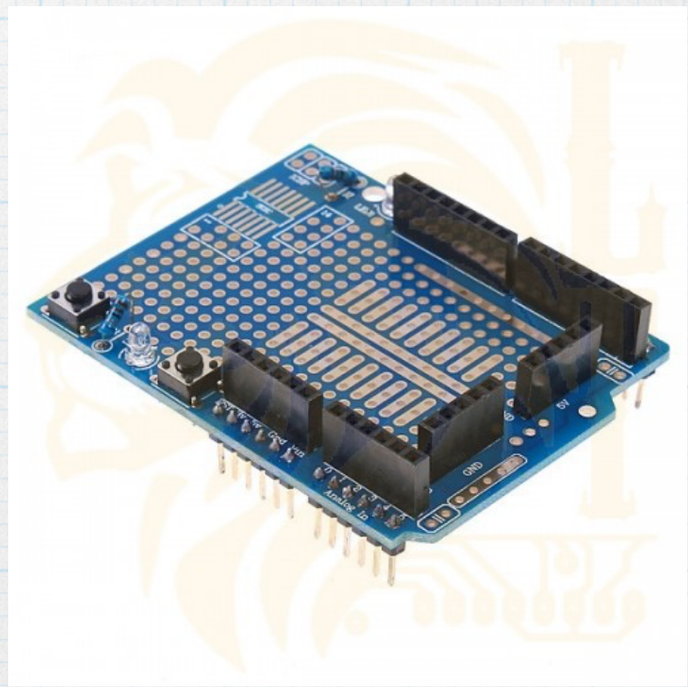
WiFi

Bluetooth

Touch display



Motor



Arduino Uno software

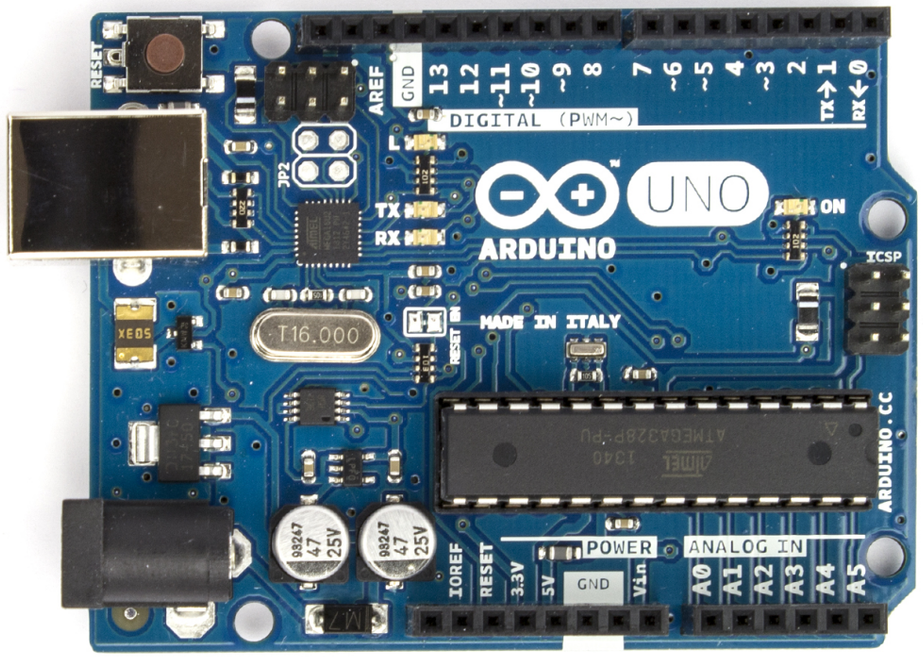
- The IDE for the Arduino is a Java application (runs identically on Windows / Mac / Linux).
- The programs (called sketches) are written in a subset of C. There are some special commands for defining the operation of I/O and for receiving/ sending information to those pins.
- There is a USB interface to facilitate transferring programs from the IDE on the computer to the controller. (No hardware dongle!) The USB interface also conveniently provides power when connected to a computer.
- There is an interface, called the serial monitor, that is analogous to the C console to interact with the controller as the code runs. It is rather crude.
- Note: The controller is meant to operate in a stand-alone mode. It is not intended to be tethered to the host computer indefinitely. But when developing the software, it is important to be able to "see" what is happening with the program.
- Everything is open-source, Including the hardware design.

What is needed to get started?

An Arduino Uno R3 board ~ $27

Get it directly from Arduino (https://arduino.cc). Or from many familiar vendors: Amazon, Digikey, Mouser, Newark, Adafruit, Jameco.

(Beware of knock-offs.)



USB - A to B cable

(Standard printer connection.)

Download and install the Arduino IDE

https://www.arduino.cc/en/Main/Software (Currently at ver. 2.2.1.)

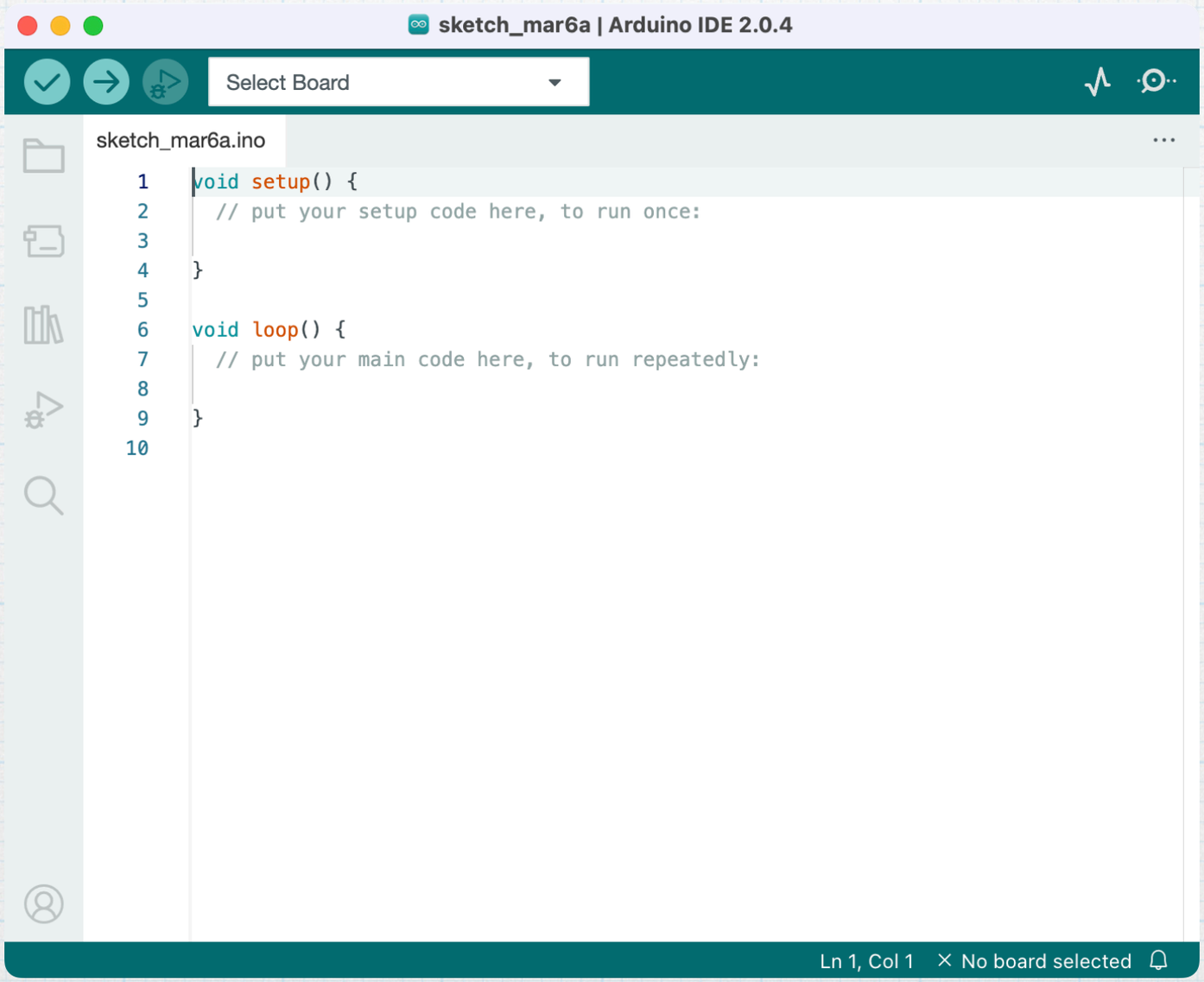
Structure of an Arduino program

An Arduino program (known as a “sketch”) is slightly different from a typical C program. There is no `main` function. Instead there are two separate functions — one called setup() and a second called loop(). Setup runs once at the beginning of execution and then loop() starts and runs in a continuous loop.

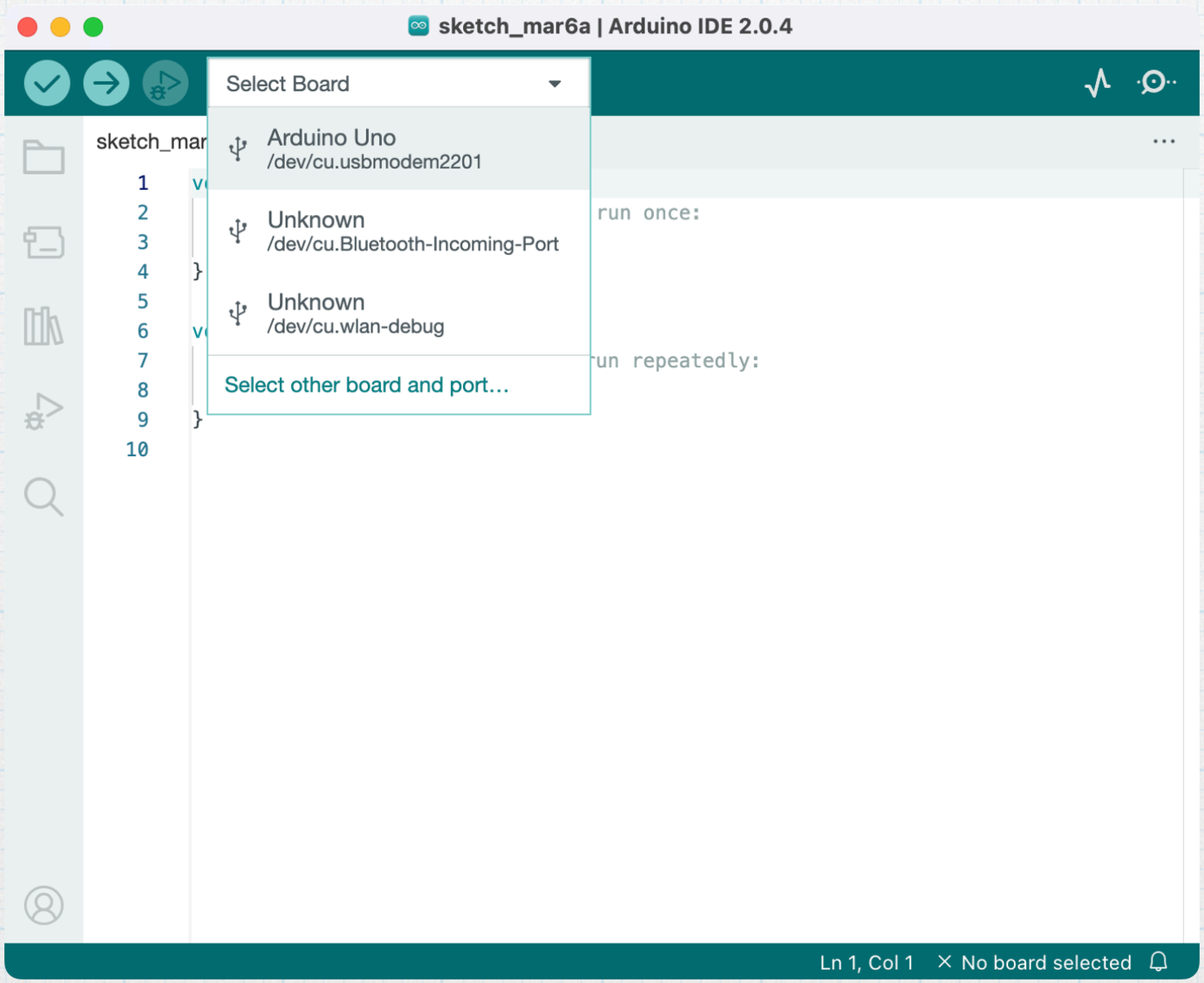
- The setup function behaves just like a regular C program. Program flow starts at the top and progresses through to the last line. (If there are any loops with setup, they behave according to the conditionals like we would expect.) As the name implies, the purpose of setup is to define variables, set the function of the pins, and take care of any preliminary housekeeping. When setup() is complete, program flow passes to loop().
- The program continues with the first line of the loop and then progresses through to the last line. Then, as the name implies, the program goes back to the first line of loop and runs again. And again and again. Forever. (Or until the power is removed.) It is as if there is an implied `while` loop whose conditional is always true.

Getting started with a simple sketch

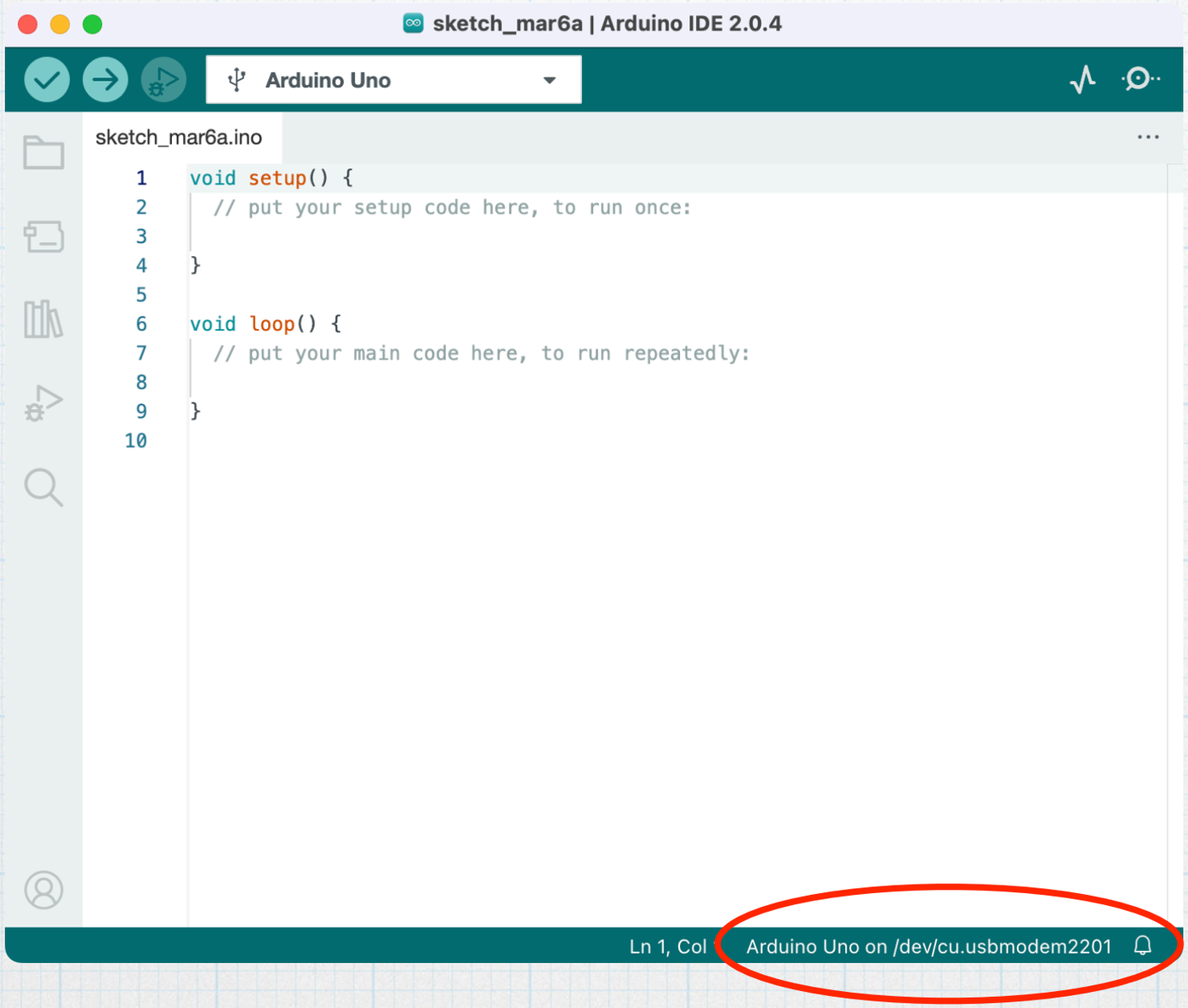
1. Download the Arduino IDE and install it on your computer.
2. Connect the Arduino board to your computer with the USB cable.
3. Launch Arduino IDE. A blank program template window appears. Empty versions of the two basic functions are in place. A line at the bottom of the window indicates that the board is not yet connected.



4. From the drop-down menu at the top of the window, choose “Arduino Uno”. “/dev/cu.usbmodemxxxx” indicates that the connection will be made through the USB port. (This is USB name for a Mac connection — the number may be different on different computers. A Windows computer may have a COM port.) The board and port selection can also be done using the “Tools” menu.

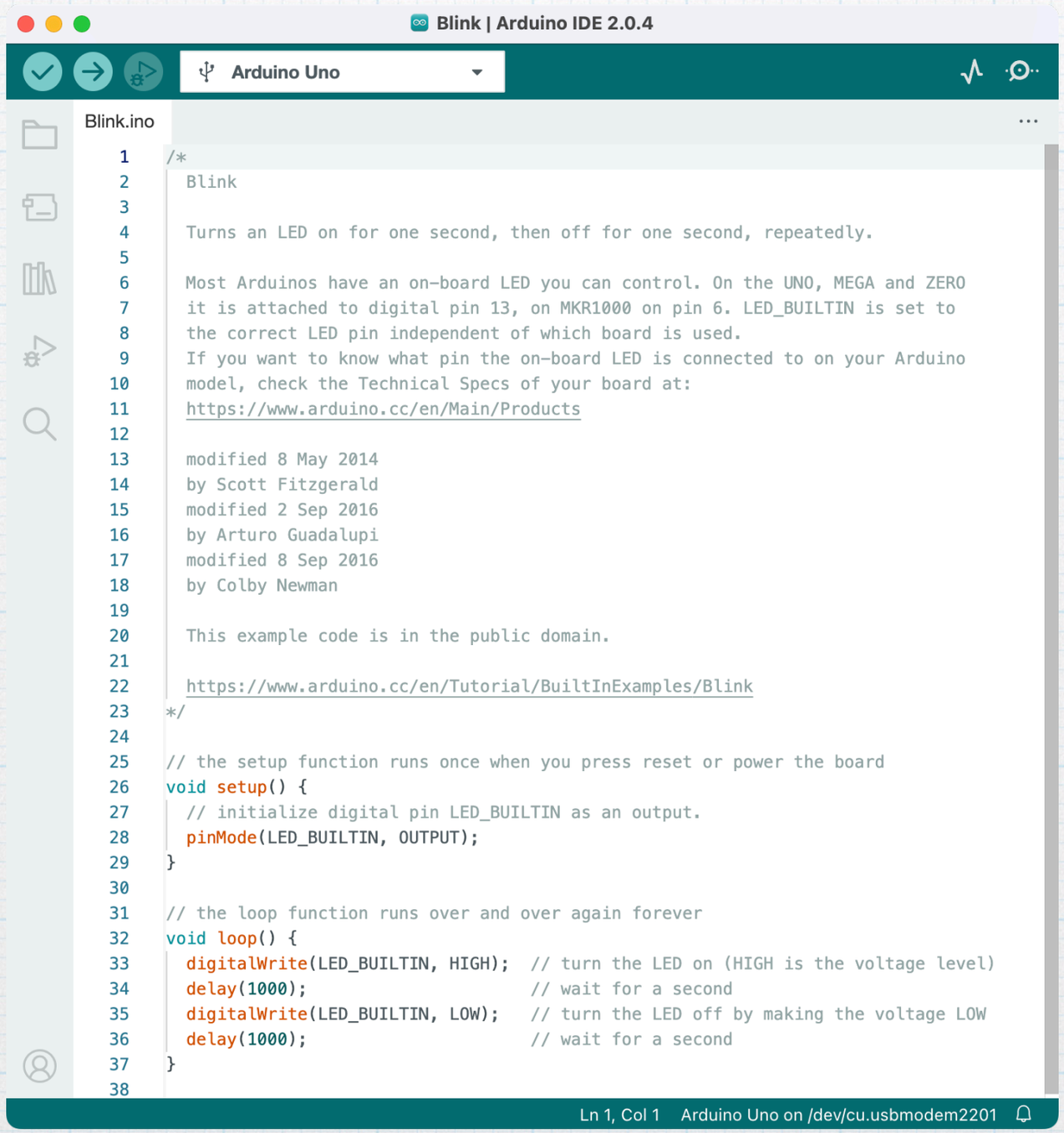


5. After making the board selection, the status line at the bottom of the window indicates that an Arduino Uno is connected via the USB port. We are ready to roll.



If the message still indicates that no board is connected, check the USB connection and trying selecting the board and port using the Tools menu.

6. A good way to start is to use some of the built-in programs. My favorite first program is “Blink” which simply blinks an LED on and off. It’s very simple and has all the basics of an Arduino sketch. From the file menu, select File→Examples→01.Basics→Blink. A new window opens with the Blink sketch.



6. There are many lines of comments. We should read through those. But to simplify presentation here, we show the sketch with most of the comments removed.

7. In the setup function, there is only one line:
pinMode(LED_BUILTIN, OUTPUT);

LED_BUILTIN is a pre-defined constant equal to 13. On the board, there is dedicated LED connected to digital pin 13. The pinMode command sets pin 13 to be a digital output.

8. In the loop function, there are two different commands.

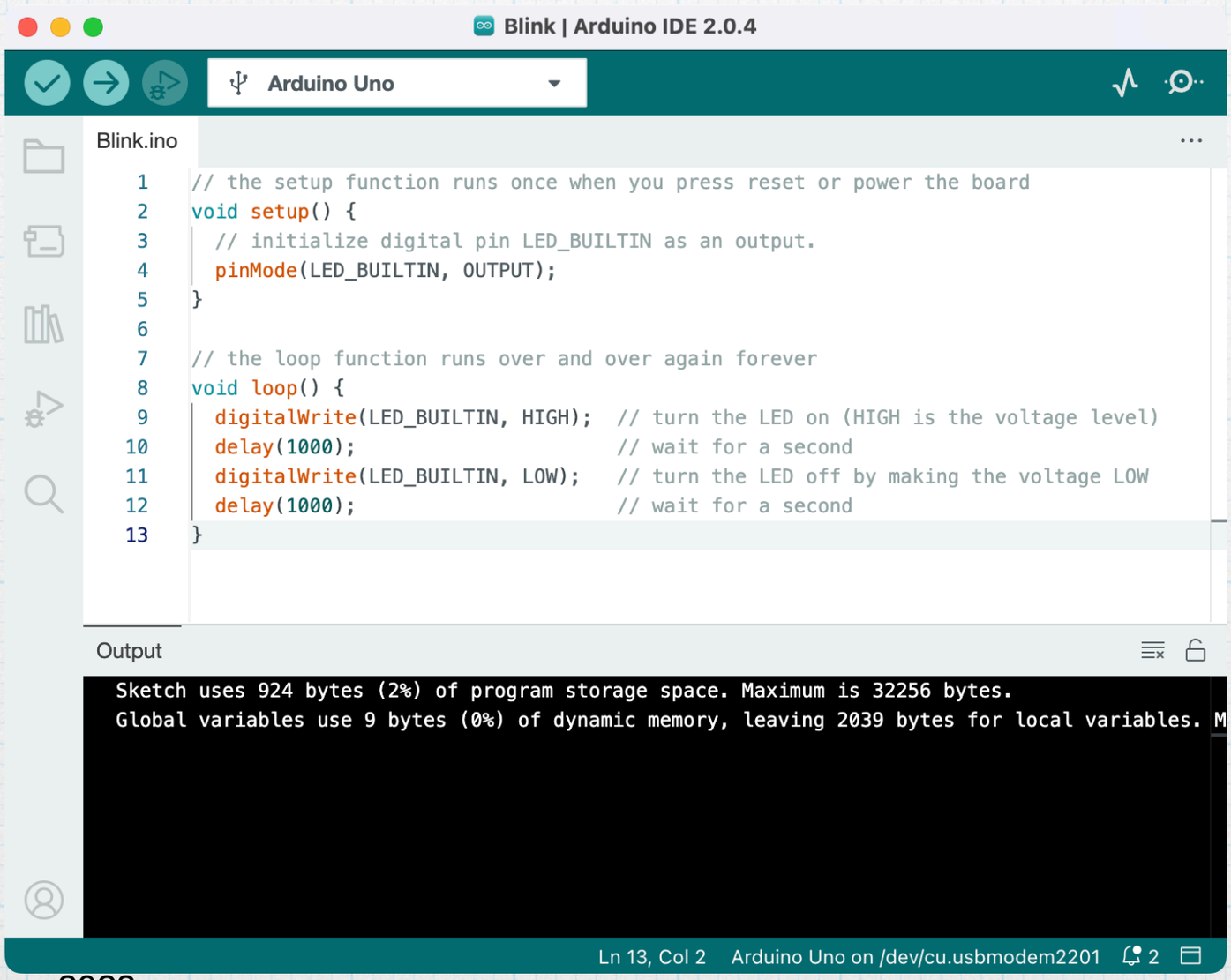
   digitalWrite(LED_BUILTIN, HIGH) sets the output of the LED_BUILTIN digital pin (#13) to the high state, meaning 5 V. This will turn the LED on. digitalWrite(LED_BUILTIN, LOW) sets the output of pin 13 to the low state, meaning 0 V. The LED turns off.

   delay(1000) means to do nothing for 1000 ms (= 1 s). The controller go into a time-killing while loop for the prescribed time.

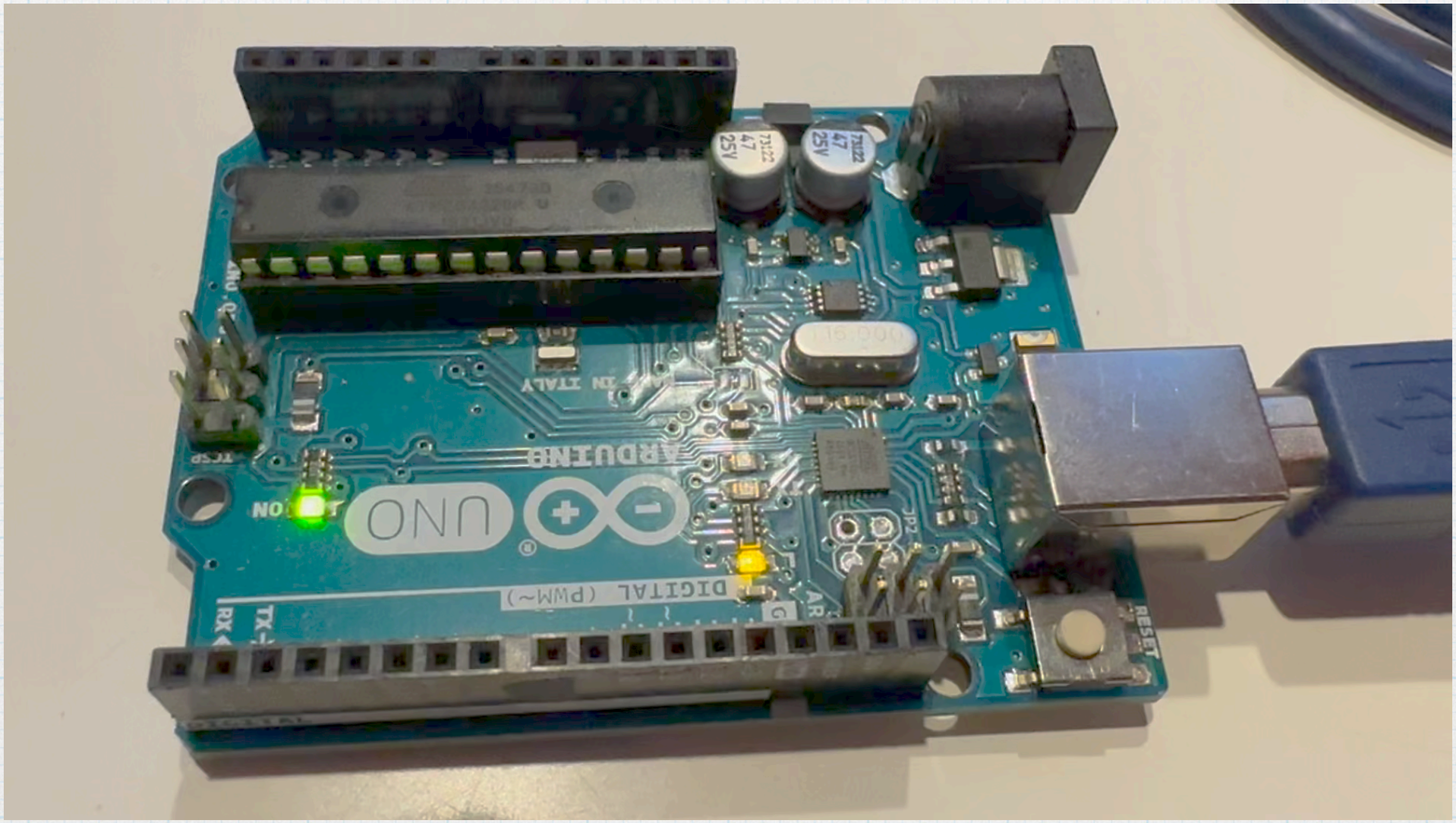
8. The loop sequence has the following actions:
   Turn on the LED
   Wait 1 second
   Turn off the LED
   Wait 1 second
   Repeat ad infinitum.

   That's it — the LED blinks.

9. Upload the sketch to the Arduino by clicking the right-arrow button at the top left of the window. It takes a couple of seconds to upload. The lower portion of the sketch window changes to display some commentary relating to the transfer of the program. If there were any errors in the compiling or transferring the code, those will be reported here. We also see some LEDs on the board blink as the code is transferred.



9. As soon as the code is completely transferred, the program will begin running on the board. There is a small yellow LED off to one side that will start blinking — 1 second on, 1 second off. That's the entirety of the exercise.



10. Play around with program by making some changes:
    - Substitute the number 13 for BUILTIN_LED in each line
    - Change the on and off times.
    - Add more lines to change the blink pattern.
    - Connect an external LED/limiting resistor combination to pin 13 so that two LEDs will blink.
    - Connect the external LED/resistor combination to other digital pins and make the corresponding changes the code.
    - Connect more LEDs and make fancy flashing patterns.

Note that after each change, the program will have to be uploaded again from the computer to the Arduino board.

With five minutes of playing around, we will understand everything about this simple program and a lot about how Arduino sketches work in general.

We are off to a good start towards learning about embedded systems, but there are many more fun things to learn.