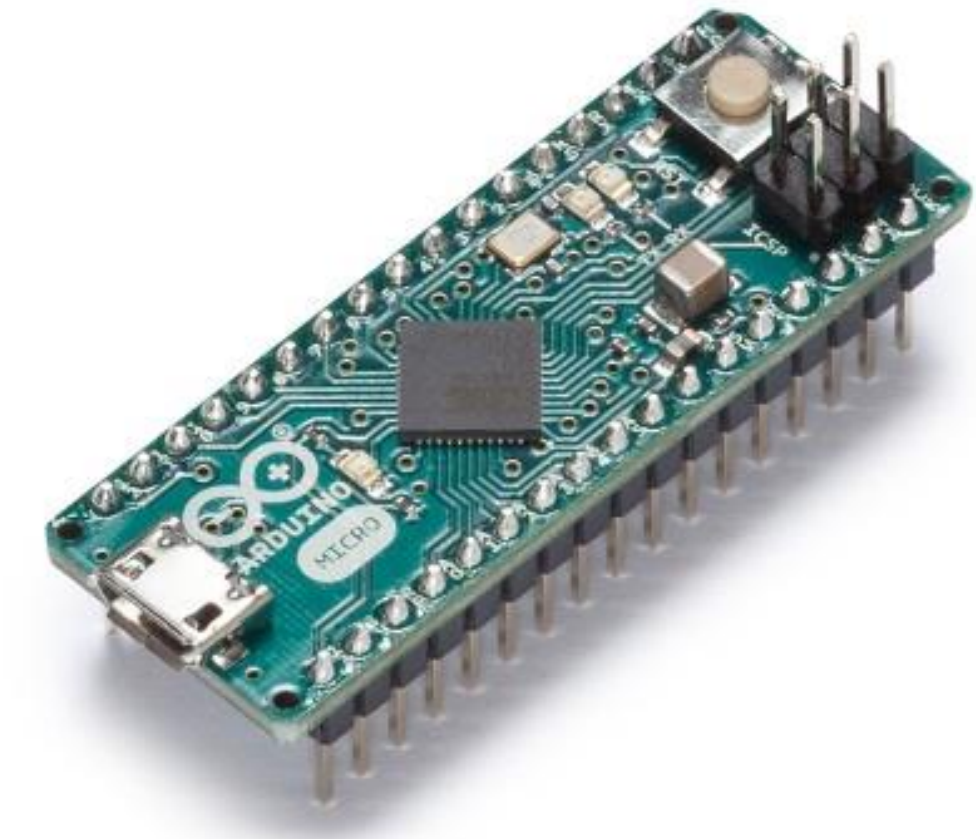
ARDUINO MICRO



OSH: Schematics, Reference Design, Board size

Arduino / Genuino Micro is open-source hardware! You can build your own board using the follwing files:

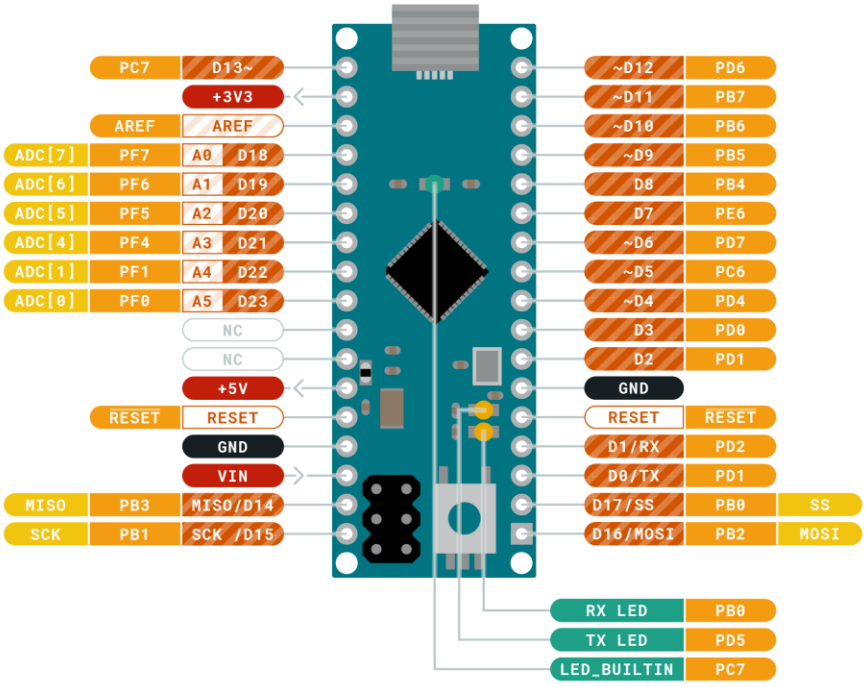
EAGLE FILES IN .ZIPSCHEMATICS IN .PDFBOARD SIZE IN .DXF

Pinout Diagram

Complete Pinout Diagram

ARDUINO

ARDUINO
MICRO



Ground
Power
LED
Internal Pin
SWD Pin
Other Pin
Digital Pin
Analog Pin
Default
Microcontroller's Port

Getting Started with the Arduino Leonardo, Leonardo ETH and Micro

Use your Leonardo, Leonardo ETH and Micro on the Arduino Web IDE

All Arduino boards, including this one, work out-of-the-box on the Arduino Web Editor, no need to install anything.

The Arduino Web Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow this simple guide to start coding on the browser and upload your sketches onto your board.

Use your Leonardo, Leonardo ETH and Micro on the Arduino Desktop IDE

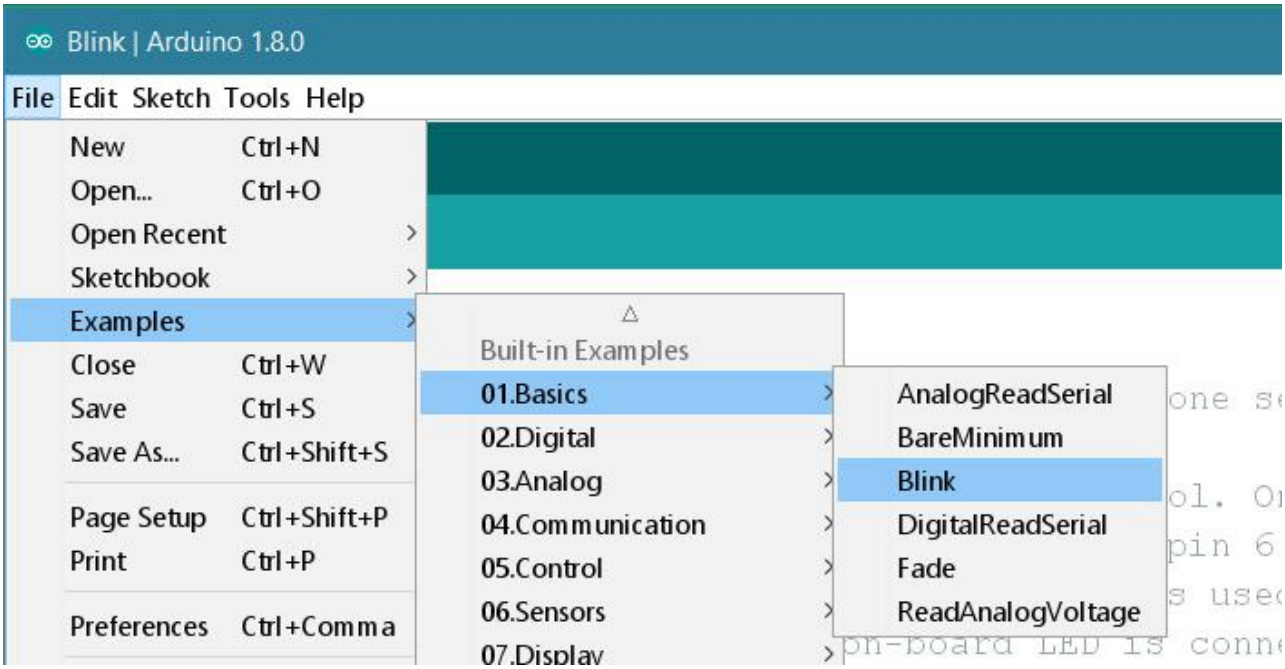
If you want to program your Leonardo, Leonardo ETH and Micro while offline you need to install the Arduino Desktop IDE.

Installing drivers for Leonardo, Leonardo ETH and Micro

Drivers should be automatically installed plugging with an USB cable the board to your PC, but with some version of the Windows operative system (like Windows 7, Vista and 10) it can happen that your board won't be recognized and you will get the message Unknown USB device. It is so necessary to manually install them following the guide Manually install Drivers on Windows.

Open the Blink example

Now that you've set up your online IDE let's make sure your computer can talk to the board, it's time to make sure you can upload a program. To do that let's open the LED blink example sketch: File > Examples > 1.Basics > Blink.

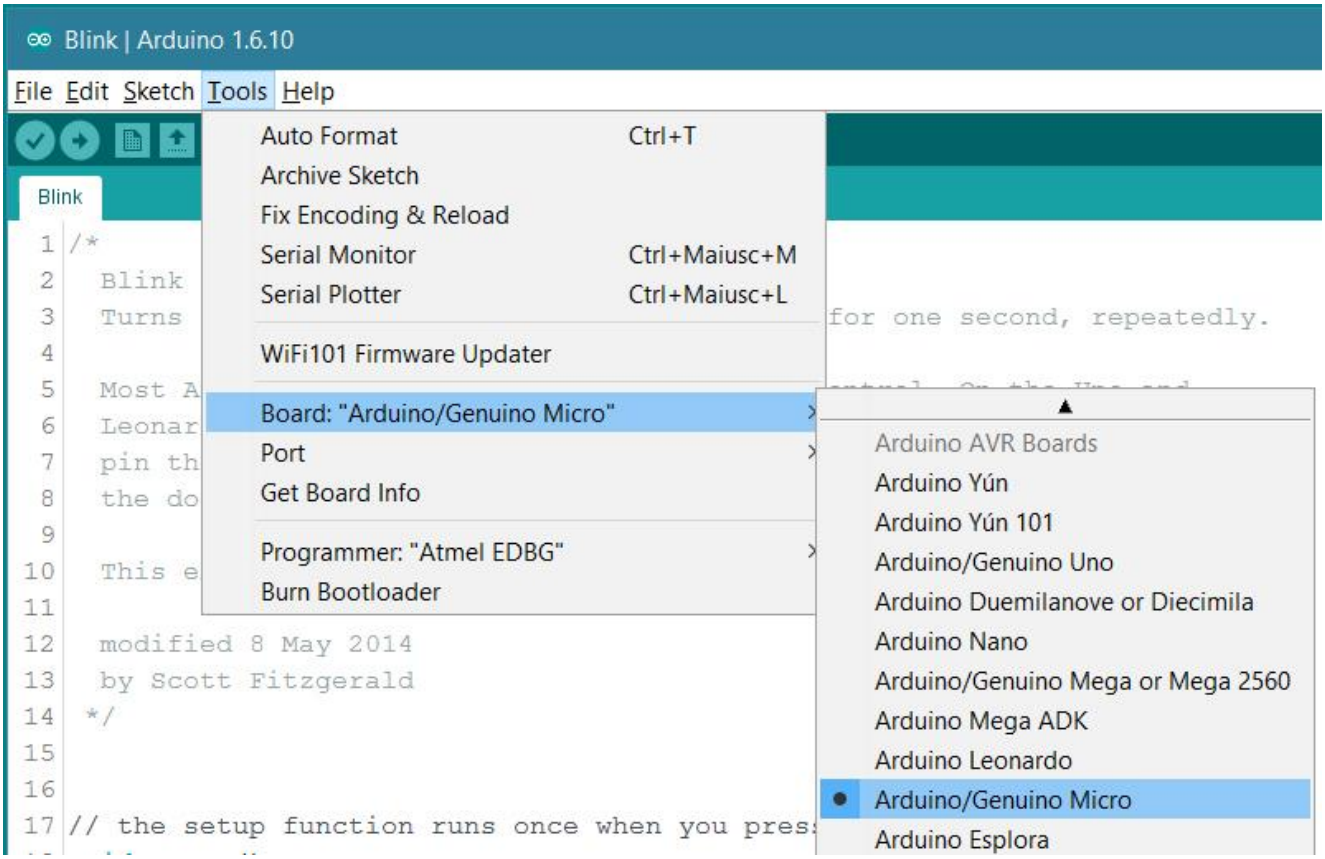


Select your board

You'll need to select your board in the Tools > Board menu:

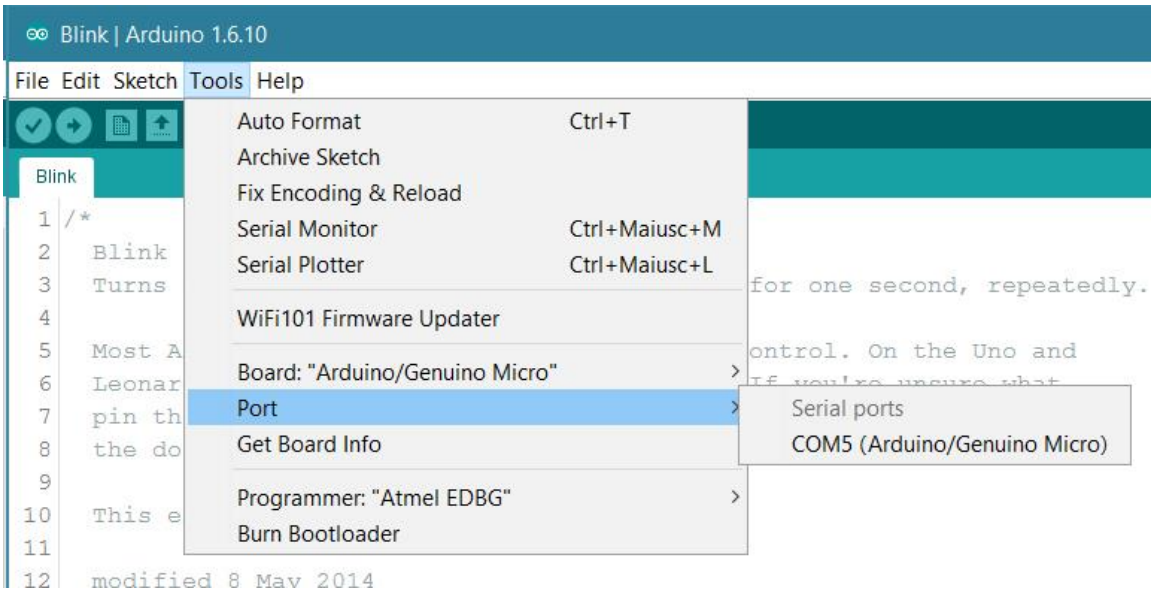
- Arduino Leonardo
- Arduino Leonardo ETH
- Arduino/Genuino Micro

according to the board you have.



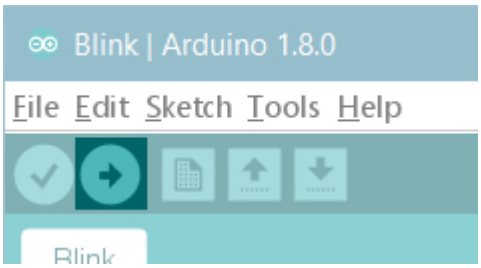
Select your serial port

Select the serial device of the board from the Tools > Serial Port menu.



Upload and Run your first Sketch

Click the Upload button in the upper left to load and run the sketch on your board:



After the compilation and upload process, you should see the message *Done Uploading* and the built-in LED of the board should start blinking.

Tutorials

Now that you have set up and programmed your Leonardo, Leonardo ETH or Micro board, you may find inspiration in our Project Hub tutorial platform.

More examples on the following library pages will help you in making very cool things!

- Keyboard - Send keystrokes to an attached computer.
- Mouse - Control cursor movement on a connected computer.
- Ethernet for connecting to the internet using the Arduino Ethernet Shield, Arduino Ethernet Shield 2 and Arduino Leonardo ETH

Please read...

Good Coding Practice With the Leonardo, Leonardo ETH and Micro

A word of caution on using the USB Mouse and Keyboard Libraries: if the Mouse or Keyboard library is constantly running, it will be difficult to program your board. Functions such as Mouse.move() and Keyboard.print() will move your cursor or send keystrokes to a connected computer and should only be called when you are ready to handle them. It is recommended to use a control system to turn this functionality on, like a physical switch or only responding to specific input you can control. When using the Mouse or Keyboard library, it may be best to test your output first using Serial.print(). This way, you can be sure you know what values are being reported. Refer to the Mouse and Keyboard examples for some ways to handle this.

Using the serial monitor effectively: Since serial is going through only one processor, the board is capable of filling your computer's serial buffer faster than the Uno or earlier boards. You may notice that if you send serial continually, for example like this:

```
void loop() {
  int sensorReading = analogRead(A0);
  Serial.println(sensorReading);
}
```

[Get Code]

the Serial Monitor in the IDE slows down considerably as it tries to keep up. If you encounter this, add a short delay to your loop so that the computer's serial buffer is not filled as fast. Even a millisecond delay will help:

```
void loop() {
  int sensorReading = analogRead(A0);
  Serial.println(sensorReading);
  delay(1);
}
```

[Get Code]

Serial applications using native libraries other than RXTX library read the serial buffer faster, so you may not encounter this error much outside of the Serial Monitor, Processing, or other RXTX-based serial applications.

Differences from the Arduino Uno

In general, you program and use the Leonardo, Leonardo ETH and Micro as you would other Arduino boards. There are, however, a few important differences.

Single processor for sketches and USB communication

The Leonardo, Leonardo ETH and Micro differ from other Arduino boards in that they use a single microcontroller to both run your sketches and for USB communication with the computer. The Uno and other boards use separate microcontrollers for these two functions, meaning that the USB connection to the computer remains established regardless of the state of the main microcontroller. By combining these two functions onto a single processor, the Leonardo allows for more flexibility in its communication with the computer. It also helps to lower the cost of the board by removing the need for an additional processor.

Serial re-enumeration on reset. Since the boards do not have a dedicated chip to handle serial communication, it means that the serial port is virtual -- it's a software routine, both on your operating system, and on the board itself. Just as your computer creates an instance of the serial port driver when you plug in any Arduino, the Leonardo/Micro creates a serial instance whenever it runs its bootloader. The board is an instance of USB's Connected Device Class (CDC) driver.

This means that every time you reset the board, the USB serial connection will be broken and re-established. The board will disappear from the list of serial ports, and the list will re-enumerate. Any program that has an open serial connection to the Leonardo will lose its connection. This is in contrast to the Arduino Uno, with which you can reset the main processor (the ATmega328P) without closing the USB connection (which is maintained by the secondary ATmega8U2 or ATmega16U2 processor). This difference has implications for driver installation, uploading, and communication; these are discussed below.

No reset when you open the serial port. Unlike the Arduino Uno, the Leonardo and Micro won't restart your sketch when you open a serial port on the computer. That means you won't see serial data that's already been sent to the computer by the board, including, for example, most data sent in the `setup()` function.

This change means that if you're using any Serial print(), println() or write() statements in your setup, they won't show up when you open the serial monitor. To work around this, you can check to see if the serial port is open after calling `Serial.begin()` like so:

```
Serial.begin(9600);
  // while the serial stream is not open, do nothing:
  while (!Serial) ;
```

[Get Code]

Keyboard and mouse emulation. One advantage of using a single chip for your sketches and for USB is increased flexibility in the communication with the computer. While the board appears as a virtual serial port to your operating system (also called CDC) for programming and communication (as with the Arduino Uno), it can also behave as a (HID) keyboard or mouse. See the "Good Coding Practice" section below for a warning about using this functionality.

Separation of USB and serial communication. On the Leonardo, Leonardo ETH and Micro, the main Serial class refers to the virtual serial driver on the board for connection to your computer over USB. It's not connected to the physical pins 0 and 1

as it is on the Uno and earlier boards. To use the hardware serial port (pins 0 and 1, RX and TX), use Serial1. (See the Serial reference pages for more information.)

Differences in pin capabilities. The Leonardo, Leonardo ETH and Micro has some slight differences in the capabilities and assignments of various pins (especially for SPI and TWI). These are detailed on the hardware page.

Uploading Code to the Leonardo, Leonardo ETH and Micro

In general, you upload code to the Leonardo or Micro as you would with the Uno or other Arduino boards. Click the upload button in the Arduino IDE and your sketch will be automatically uploaded onto the board and then started. This works more or less the same way as with the Uno: the Arduino software initiates a reset of the board, launching the bootloader - which is responsible for receiving, storing, and starting the new sketch.

However, because the serial port is virtual, it disappears when the board resets, the Arduino software uses a different strategy for timing the upload than with the Uno and other boards. In particular, after initiating the auto-reset of the Leonardo, Leonardo ETH or Micro (using the serial port selected in the Tools > Serial Port menu), the Arduino software waits for a new virtual (CDC) serial / COM port to appear - one that it assumes represents the bootloader. It then performs the upload on this newly-appeared port.

These differences affect the way you use the physical reset button to perform an upload if the auto-reset isn't working. Press and hold the reset button on the Leonardo or Micro, *then* hit the upload button in the Arduino software. Only release the reset button after you see the message "Uploading..." appear in the software's status bar. When you do so, the bootloader will start, creating a new virtual (CDC) serial port on the computer. The software will see that port appear and perform the upload using it. Again, this is only necessary if the normal upload process (i.e. just pressing the uploading button) doesn't work. (Note that the auto-reset is initiated when the computer opens the serial port at 1200 baud and then closes it; this won't work if something interferes with the board's USB communication - e.g. disabling interrupts.)