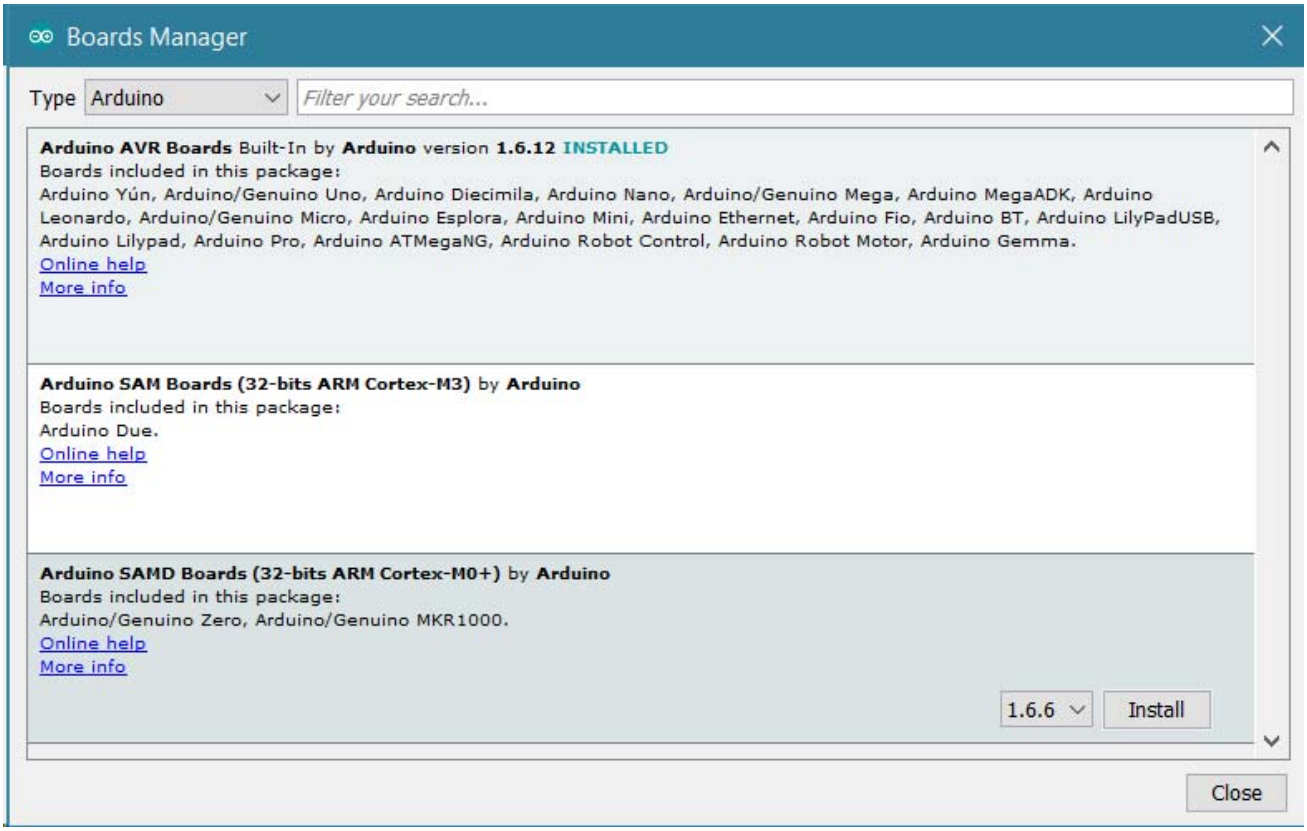
Getting started with the Arduino/Genuino Zero

This board is a simple and powerful 32-bit extension of the platform established by the UNO. Learn how to prepare your computer with all you need to start making your own projects. The Arduino Zero is programmed using the Arduino Software (IDE), our Integrated Development Environment common to all our boards and running both online and offline. For more information on how to get started with the Arduino Software visit the Getting Started page.

Use your Zero on the Arduino Desktop IDE

If you want to program your Zero while offline you need to install the Arduino Desktop IDE and add the Atmel SAMD Core to it. This simple procedure is done selecting **Tools menu**, then **Boards** and last **Boards Manager**, as documented in the Arduino Boards Manager page.

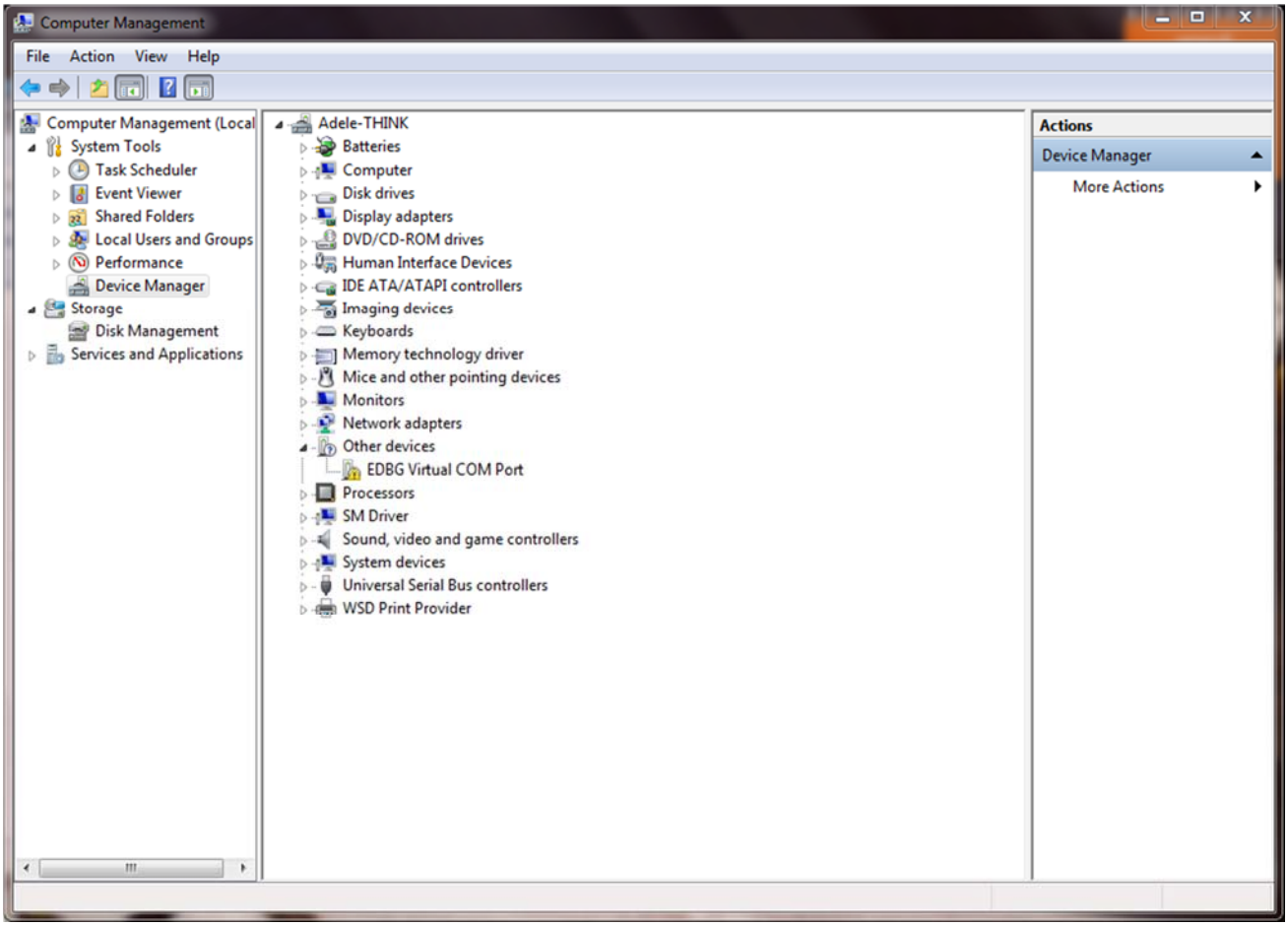


Installing Drivers for the Zero

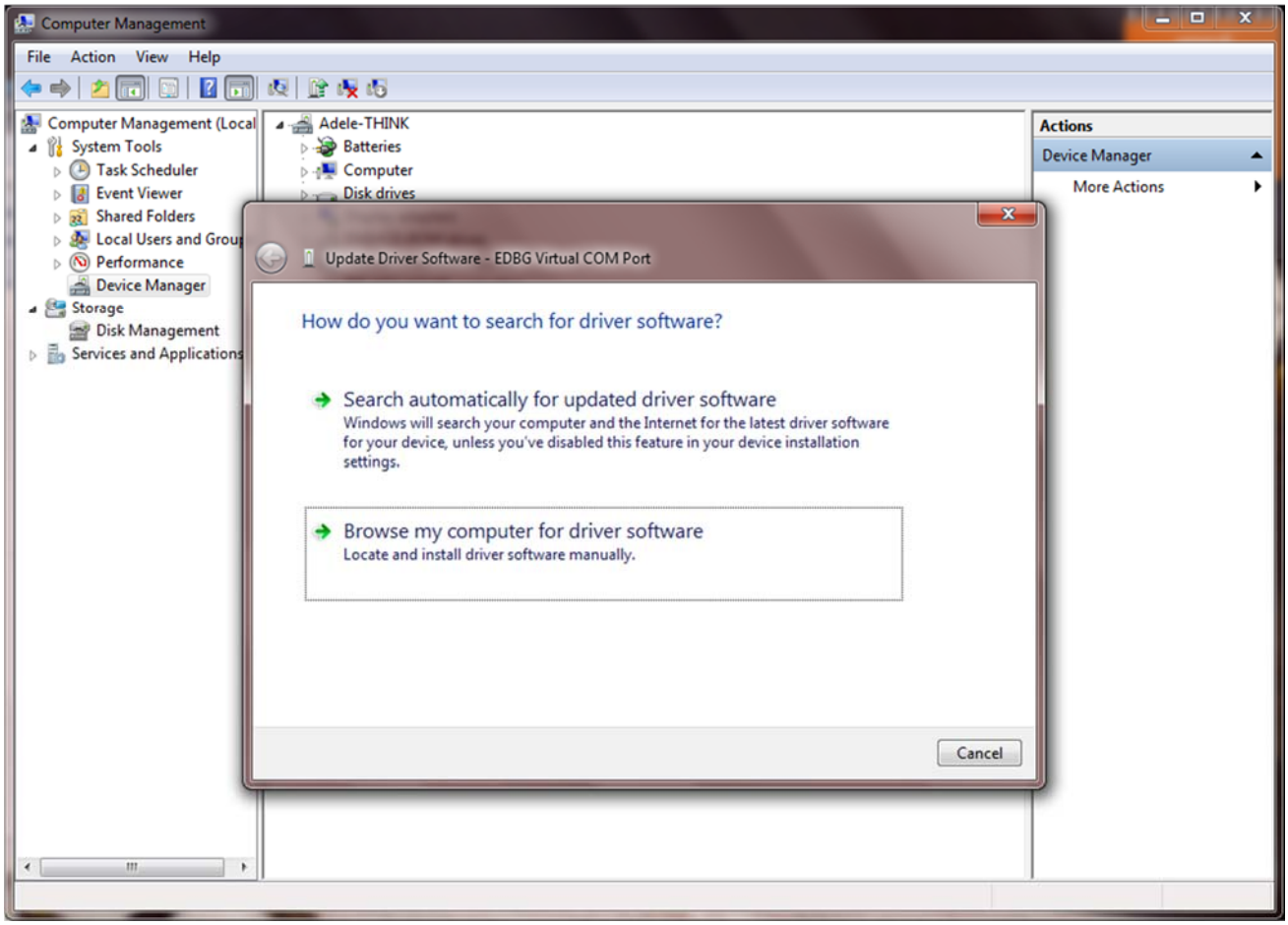
OSX
No driver installation is necessary on OSX. Depending on the version of the OS you're running, you may get a dialog box asking you if you wish to open the “Network Preferences”. Click the "Network Preferences..." button, then click "Apply". The Zero will show up as “Not Configured”, but it is still working. You can quit the System Preferences.

Windows (tested on XP, 7, Vista and 10)
Connect the Zero to your computer with a USB cable via the *Programming* port. Windows should initiate its driver installation process once the board is plugged in, but it won't be able to find the driver on its own. You'll have to tell it where the driver is. Click on the Start Menu and open the

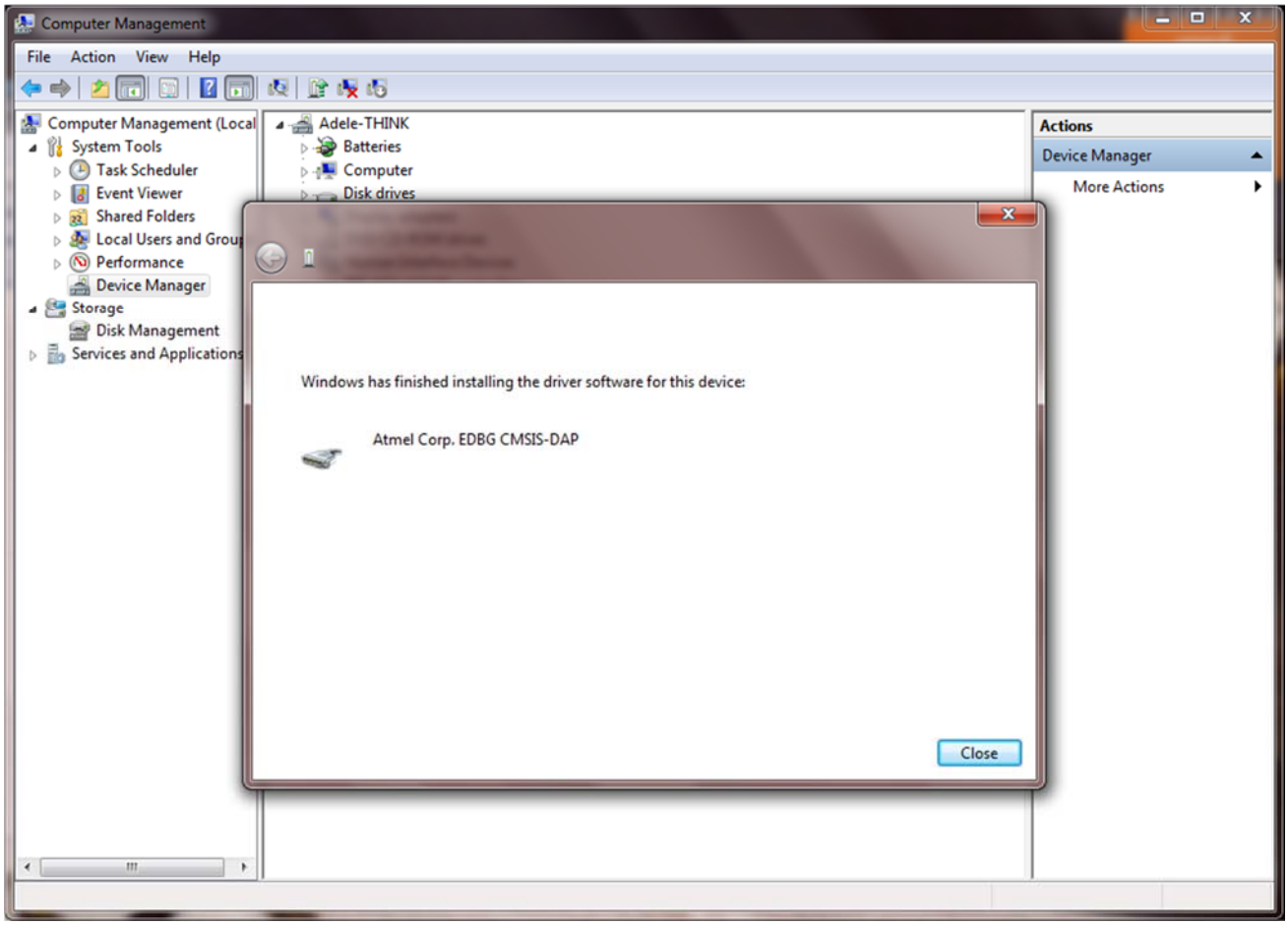
Control Panel Navigate to “System and Security”. Click on System, and open the Device Manager. Look for the listing named “Ports (COM & LPT)”. You should see an open port named “Arduino Zero Prog. Port”. Right click on the “Arduino Zero Prog. Port” and choose “Update Driver Software”.



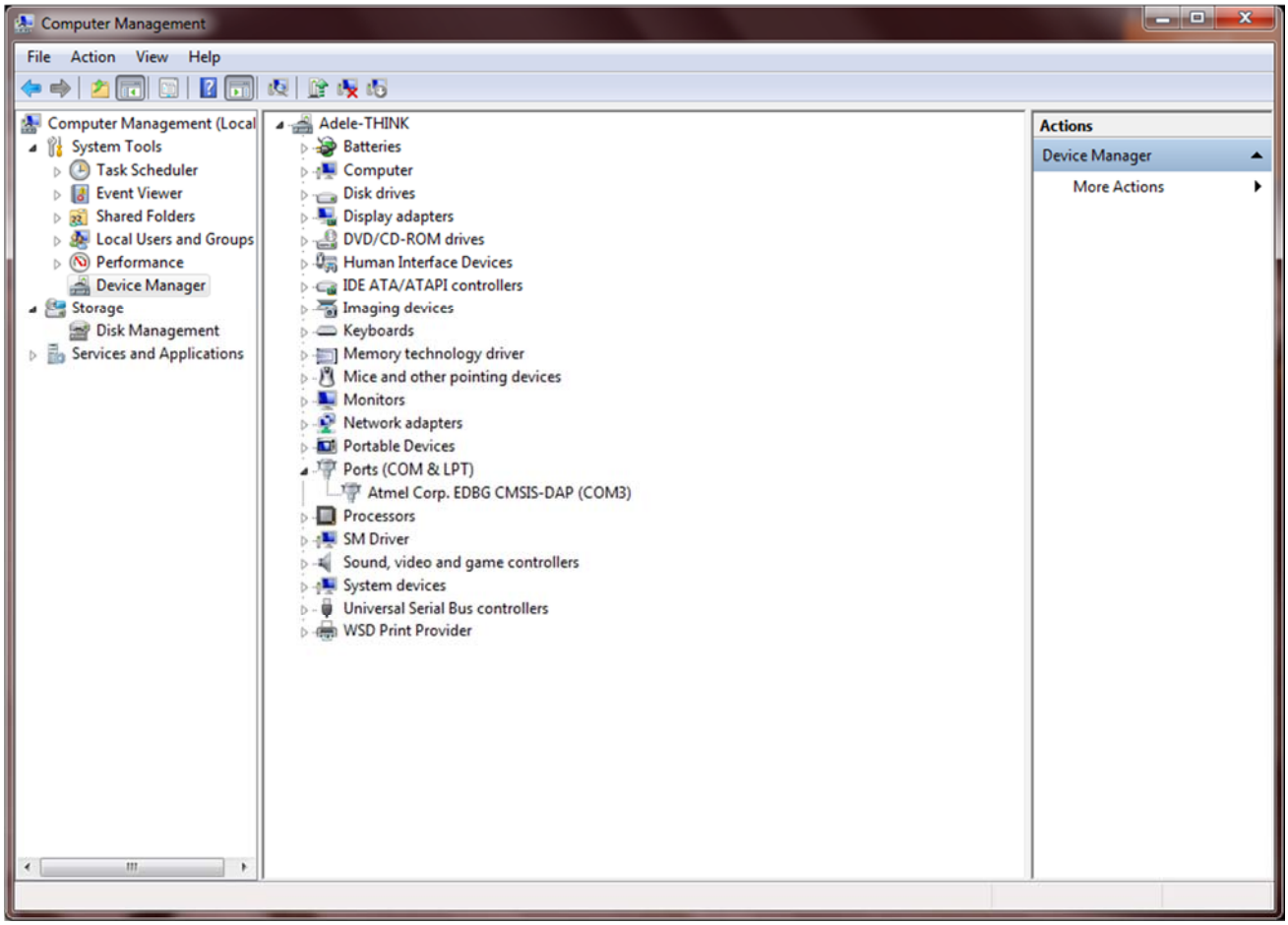
Select the “Browse my computer for Driver software” option.



Navigate to the folder with the Arduino IDE you downloaded and unzipped earlier. Locate and select the “Drivers” folder in the main Arduino folder (not the “FTDI USB Drivers” sub-directory). Press “OK” and “Next” to proceed. If you are prompted with a warning dialog about not passing Windows Logo testing, click “Continue Anyway”. Windows now will take over the driver installation.



You have installed the driver on your computer. In the Device Manager, you should now see a port listing similar to “Arduino Zero Programming Port (COM4)” If you have multiple COM devices, the Zero will probably be the COM port with the largest number.

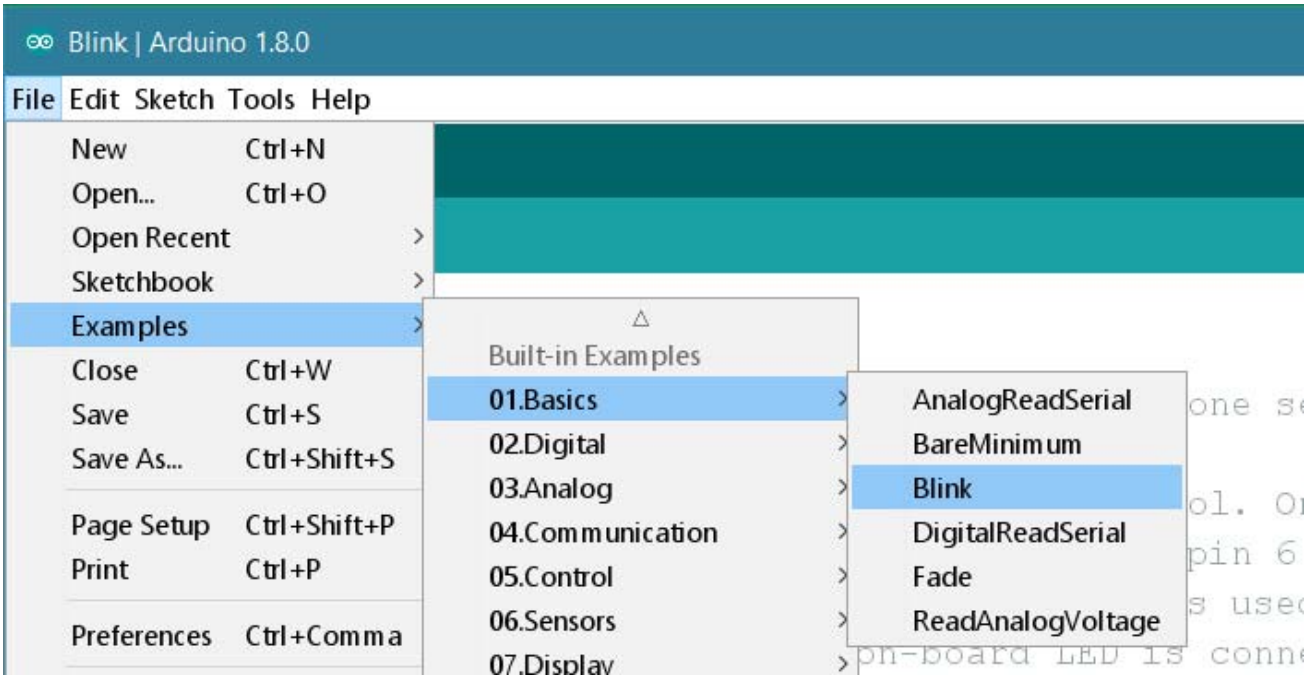


For what concern the **Native USB port** the procedure is the same, but in the device manager you will see an **unknown device**.

Linux
No driver installation is necessary for Linux.

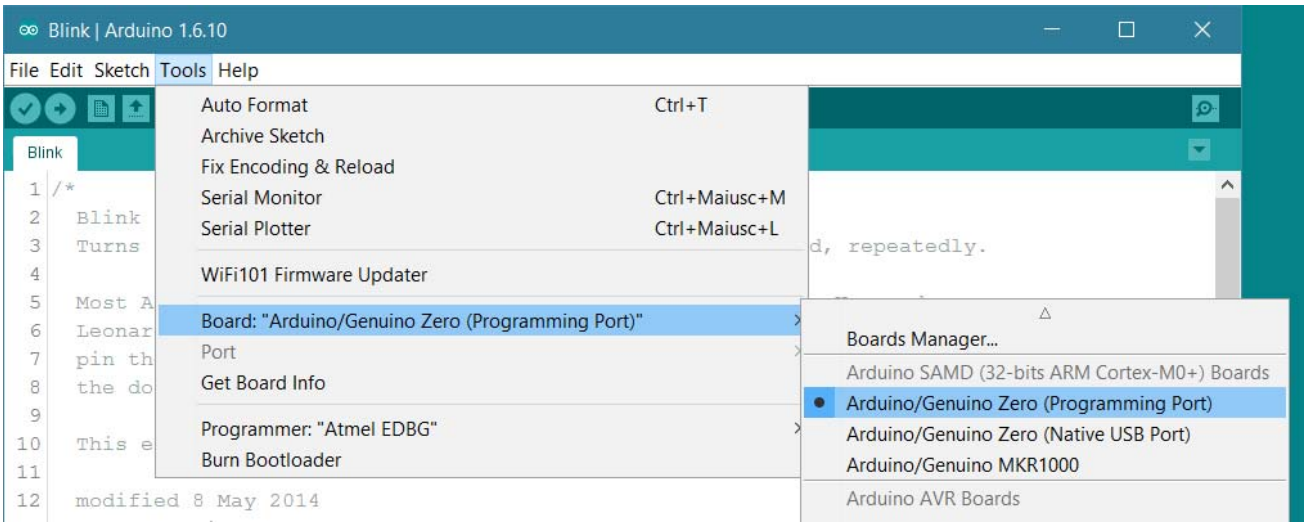
Open your first sketch

Open the LED blink example sketch: **File > Examples >01.Basics > Blink**.

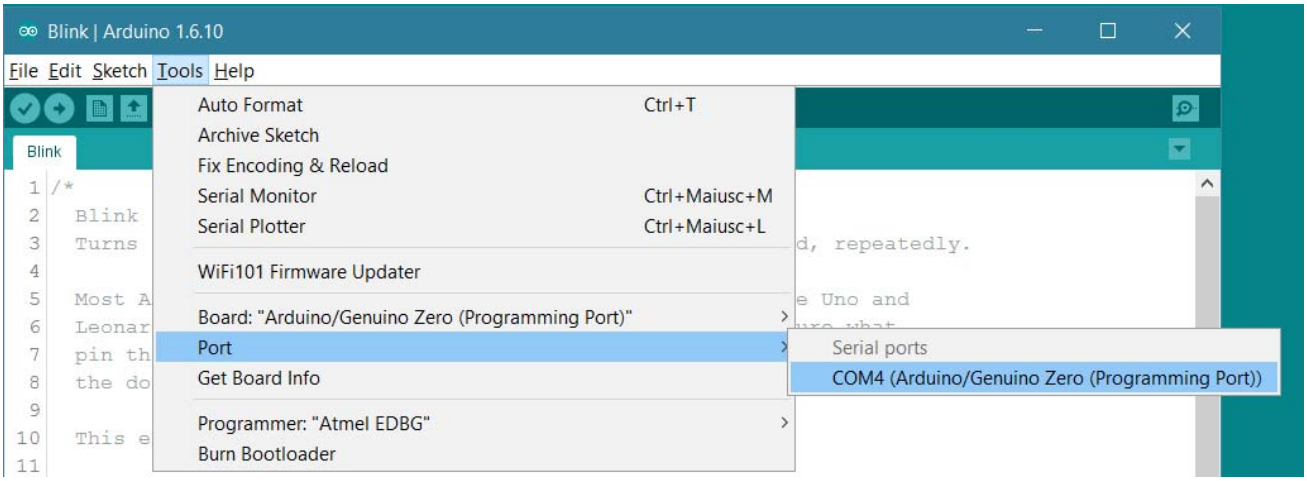


Select your board type and port

You'll need to select the entry in the **Tools > Board** menu that corresponds to your Arduino or Genuino board.



Select the serial device of the board from the Tools | Serial Port menu. This is likely to be **COM3** or higher (**COM1** and **COM2** are usually reserved for hardware serial ports). To find out, you can disconnect your board and re-open the menu; the entry that disappears should be the Arduino or Genuino board. Reconnect the board and select that serial port.



Upload the program

Now, simply click the "Upload" button in the environment. Wait a few seconds - you should see the RX and TX leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.

A few seconds after the upload finishes, you should see the on-board LED start to blink (in orange). If it does, congratulations! You've gotten your Zero board up-and-running. If you have problems, please see the troubleshooting suggestions.

Please Read...

The microcontroller on the Zero runs at 3.3V, which means that you must never apply more than 3.3V to its inputs or outputs. Care must be taken when connecting sensors and actuators to assure that this is never exceeded. **Connecting higher voltages, like the 5V commonly used with the other Arduino/Genuino boards, will damage the Zero.**

Differences from ATMEGA based boards

The Zero has the same footprint as the Arduino/Genuino Uno, and in general, you can program and use the Zero as you would do with other Arduino boards. There are, however, a few important differences and functional extensions, listed below.

Voltage

The microcontroller on the Zero runs at 3.3V, while a board like the Arduino Mega runs at 5V. You must never apply more than 3.3V to the Zero's inputs or outputs. When you connect sensors and actuators to the Zero always take care that the maximum voltage limits are not exceeded on the pins.

Connecting higher voltages, like the 5V commonly used with the other Arduino boards, will damage the Zero. If in doubt measure the IORef or the VCC pin, which supplies the voltage corresponding to the i/o of your board.

The board can take power from the USB connectors or from the DC plug (6-20V).

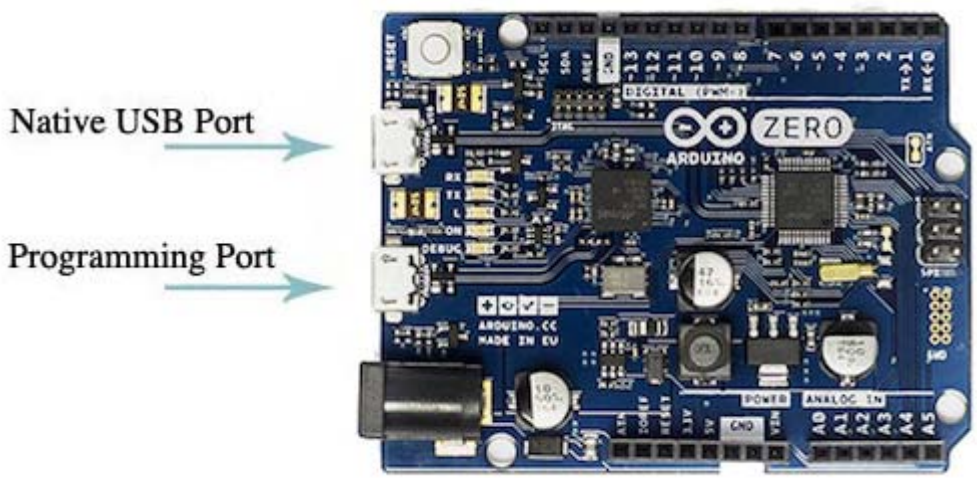
The Zero has an efficient switching voltage regulator, compliant with the USB host specification. Using the *Native* port as USB host implies that the board has to provide power to the device, for example a mouse or a keyboard.

Voltage/Current restrictions

Maximum source current is 46mA and maximum sink current is 65mA per cluster. A cluster is a group of GPIOs. For more information about what pins belong to what cluster, and what VDD/GND pins they draw power from, see page 936 this document.

Symbol	Parameter	Min.	Max.	Units
VDD	Power supply voltage	0	3.8	V
IGND	Current into a VDD pin	-	92	mA
IVDD	Current out of a GND pin	-	130	mA
VPIN	Pin voltage with respect to GND and VDD	GND-0.3V	VDD+0.3V	V

Serial ports on the Zero



The Zero has two USB ports available. The *Native* USB port (which supports CDC serial communication using the *SerialUSB* object) is connected directly to the SAMD21 MCU. The other USB port is the *Programming* port. It is connected to the ATMEL embedded debugger (EDBG), the onboard programmer and debugger which can also acts as a USB-to-Serial converter. This *Programming* port is the default for uploading sketches and communicating with the board.

The USB-to-serial converter of the *Programming* port is connected to the first UART of the SAMD21. It's possible to communicate over this port using the "Serial" object in the Arduino programming language.

The USB connector of the *Native* port is directly connected to the USB host pins of the SAMD21. Using the *Native* port enables you to use the Zero as a client USB peripheral (acting as a mouse or a keyboard connected to the computer) or as a USB host device so that devices can be connected to the Zero (like a mouse, keyboard, or an Android phone). This port can also be used as a virtual serial port using the "SerialUSB" object in the Arduino programming language.

Native port
Opening and closing the *Native* port at the baud rate of 1200bps triggers a “soft erase” procedure: the flash memory is erased and the board is restarted with the bootloader. This procedure is managed by the MCU, so if the MCU is interrupted for any reason, it is likely that the soft erase procedure would fail.

Opening and closing the *Native* port at a baudrate other than 1200bps will not reset the SAMD21. To use the serial monitor, and see what your sketch does from the beginning, you'll need to add few lines of code inside the setup(). This will ensure the SAMD21 will wait for the SerialUSB port to open before executing the sketch:

```
while (!SerialUSB) ;
```

Pressing the Reset button on the Zero causes the SAMD21 to reset as well as resetting the USB communication. This interruption means that if the serial monitor is open, it's necessary to close and reopen it to restart the communication.

Programming port
The USB *Programming port* is connected to the Atmel EDBG, which is an integrated programmer and debugger. Through the *Programming port* you have the complete control of the SAMD21, for example you can use the EDBG to burn the bootloader or to access to the entire flash content. Beside these advanced features it also behave as an USB-to-serial converter connected to the first serial interface of the SAMD21. Uploading using the *Programming port* is the safest way to program the SAMD21. For example it works even if the sketch running on the main MCU is not responding.

To communicate serially with the *Programming port*, use the "Serial" object in the IDE. All existing sketches that use serial communication based on the Uno board should work similarly. On contrary of what happen on the Arduino UNO, opening the Serial Monitor (or any other serial communication) on the Zero doesn't cause the main MCU reset. Pressing the Reset button while communicating over the *Programming* port doesn't close a USB connection with the computer because only the SAMD21 is reset.

ADC and PWM resolutions
The Zero has the ability to change its analog read and write resolutions (defaults to 10-bits and 8-bits, respectively). It can support up to 12-bit ADC/PWM and 10-bit DAC resolutions. See the analog write resolution and analog read resolution pages for information.

For more details on the Arduino Zero, see the hardware page.

Adding more Serial Interfaces to SAMD microcontrollers (SERCOM)

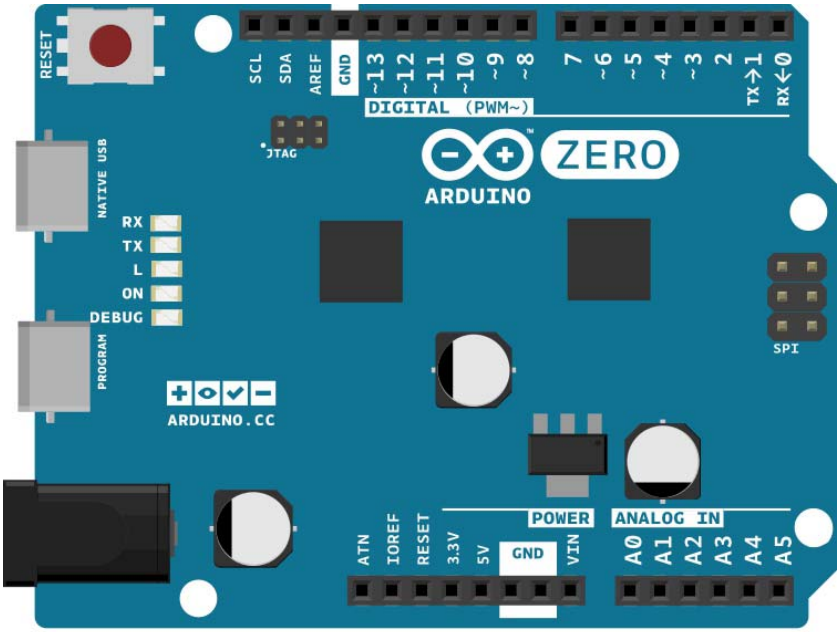
In this tutorial we explain you how to add further serial interfaces to your SAMD based board; these interfaces are hardware based and can be I2Cs, UARTs or SPIs types. This is possible because the SAMD microcontroller has six internal serial modules that can be configured individually and just four of them are already configured; the other two are available for mapping onto specific pins and in this tutorial we explain you how to do it.

Hardware Required

- Arduino or Genuino Zero, MKRZero or MKR1000 Board

Circuit

Only your Arduino or Genuino Board is needed for this example.



Pin functions
One of the advantages of the Arduino platform is the simplification of the hadware, assigning to each microcontroller pin one of the many possible functions. You can find the various functions assigned to each pin in the variant.cpp file of each board. Let's see, for example, the MKR1000 one.

Focusing our attention on the SERCOM related pin we can extract the following information :

Pin number	MKR Board pin	Perip.C SERCOMx (x/PAD)	Perip.D SERCOMx (x/PAD)	Periph.G COM
00	D0	3/00	5/00	
01	D1	3/01	5/01	USB/SOF
02	D2	0/02	2/02	I2S/SCK0
03	D3	0/03	2/03	I2S/FS0
04	D4		4/02	I2S/MCK1
05	D5		4/03	I2S/SCK1
06	D6	5/02	3/02	I2S/SCK0
07	D7	5/03	3/03	I2S/FS0
	SPI			
08	MOSI	*1/00	3/00	
09	SCK	*1/01	3/01	
10	MISO	*1/03	3/03	I2S/SD0
	Wire			
11	SDA	*0/00	2/00	I2S/SD1
12	SCL	*0/01	2/01	I2S/MCK0
	Serial1			
13	RX		*5/03	
14	TX		*5/02	
16	A1		5/00	
17	A2		5/01	
18	A3		0/00	
19	A4		0/01	
20	A5		0/02	
21	A6		0/03	I2S/SD0
	ATWINC1501B SPI			
26	WINC MOSI	*2/00	4/00	
27	WINC SCK	*2/01	4/01	
28	WINC SSN	2/02	4/02	
29	WINC MISO	*2/03	4/03	
	ATWINC1501B PINS			
32	WINC WAKE		4/00	
33	WINC IRQN		4/01	

As you can see, SERCOMs can be routed almost everywhere, having more than one SERCOM routable on more than one pin.

Default assigned SERCOMs
On MKR1000 boards on the header you can find an SPI, I2C and UART interface positioned as follow:

SPI / *SERCOM 1*:
- MOSI on pin 8;
- SCK on pin 9;
- MISO on pin 10;

I2C / *SERCOM 0*:
- SDA on pin 11;
- SCL on pin 12;

UART / *SERCOM 5*:
- RX on pin 13;
- TX on pin 14;

Additionally there is another SPI interface internally connected to the WINC1500 module wired as follow:

WINC1500 SPI / *SERCOM 2*:
- MOSI on pin 26;
- SCK on pin 27;
- MISO on pin 29;

So removing from out table the pre-exisiting interfaces, since our aim is to add new interfaces instead of changing the pre-defined one, we obtain what follows:

```
/*
 +------------+------------------+---------+---------+----------+
 | Pin number |  MKR  Board pin  | Perip.C | Perip.D | Periph.G |
 |            |                  | SERCOMx | SERCOMx |    COM   |
 |            |                  | (x/PAD) | (x/PAD) |          |
 +------------+------------------+---------+---------+----------+
 | 00         | D0               |   3/00  |   5/00  |          |
 | 01         | D1               |   3/01  |   5/01  | USB/SOF  |
 | 02         | D2               |   0/02  |   2/02  | I2S/SCK0 |
 | 03         | D3               |   0/03  |   2/03  | I2S/FS0  |
 | 04         | D4               |         |   4/02  | I2S/MCK1 |
 | 05         | D5               |         |   4/03  | I2S/SCK1 |
 | 06         | D6               |   5/02  |   3/02  | I2S/SCK0 |
 | 07         | D7               |   5/03  |   3/03  | I2S/FS0  |
 +------------+------------------+---------+---------+----------+
 | 16         | A1               |         |   5/00  |          |
 | 17         | A2               |         |   5/01  |          |
 | 18         | A3               |         |   0/00  |          |
 | 19         | A4               |         |   0/01  |          |
 | 20         | A5               |         |   0/02  |          |
 | 21         | A6               |         |   0/03  | I2S/SD0  |
 +------------+------------------+---------+---------+----------+
 */
```

Adding a new communication interface
Let's now try to use the table above to add a new interface to our MKR1000 board.

Create a new Wire instance
As we can see the pin 0 and pin 1 can be driven by two SERCOMs. In particular by SERCOM3 and SERCOM5. Well looking at the SAMD21 datasheet we can figure out that the SERCOM PAD0 can be used as SDA and the SERCOM PAD1 as SCL so, we can do this using the example below.

```
/* Wire Slave Sender on pins 0 and 1 on MKR1000

 Demonstrates use of the Wire library and how to instatiate another Wire
 Sends data as an I2C/TWI slave device
 Refer to the "Wire Master Reader" example for use with this

 Created 20 Jun 2016
 by
 Arturo Guadalupi <a.guadalupi@arduino.cc>
 Sandeep Mistry <s.mistry@arduino.cc>
*/

#include <Wire.h>
#include "wiring_private.h"

TwoWire myWire(&sercom3, 0, 1);   // Create the new wire instance assigning it to
pin 0 and 1

void setup()
{
  myWire.begin(2);                // join i2c bus with address #2

  pinPeripheral(0, PIO_SERCOM);   //Assign SDA function to pin 0
  pinPeripheral(1, PIO_SERCOM);   //Assign SCL function to pin 1

  myWire.onRequest(requestEvent); // register event
}

void loop()
{
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
  myWire.write("hello "); // respond with message of 6 bytes
                          // as expected by master
}

// Attach the interrupt handler to the SERCOM
extern "C" {
  void SERCOM3_Handler(void);

  void SERCOM3_Handler(void) {
    myWire.onService();
  }
}
```

the two instructions use the internal function `pinPeripheral (pinnumber,function)` that reassign the pins

```
pinPeripheral(0, PIO_SERCOM);   //Assign SDA function to pin 0
pinPeripheral(1, PIO_SERCOM);   //Assign SCL function to pin 1
```

must be put in the `setup()` in order to override the standard Arduino pin assignment for this board (digital I/O) and to allow the SERCOM to drive them.

The callback

```
void SERCOM3_Handler(void)
{
     myWire.onService();
}
```

instead is used to allow the real I2C communication since the Wire library relies on interrupts.

Create a new Serial instance

```
/*
  AnalogReadSerial on ne UART placed on pins 0 and 1
  Reads an analog input on pin A0, prints the result to the serial monitor.
  Graphical representation is available using serial plotter (Tools > Serial
Plotter menu)
  Attach the center pin of a potentiometer to pin A0, and the outside pins to
+3.3V and ground.
  Short together pin 0 and pin 1 with a wire jumper

 Created 20 Jun 2016
 by
 Arturo Guadalupi <a.guadalupi@arduino.cc>

  This example code is in the public domain.
*/

#include <Arduino.h>
#include "wiring_private.h"

Uart mySerial (&sercom3, 0, 1, SERCOM_RX_PAD_1, UART_TX_PAD_0); // Create the new
UART instance assigning it to pin 0 and 1

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  mySerial.begin(9600);

  pinPeripheral(0, PIO_SERCOM); //Assign RX function to pin 0
  pinPeripheral(1, PIO_SERCOM); //Assign TX function to pin 1
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read on mySerial wired in loopback:
  mySerial.write(sensorValue);
```

```
  while (mySerial.available()) {
    Serial.print(mySerial.read());
  }
  Serial.println();
  delay(1);        // delay in between reads for stability
}

// Attach the interrupt handler to the SERCOM
void SERCOM3_Handler()
{
  mySerial.IrqHandler();
}
```

as we did for Wire, the two instructions

```
pinPeripheral(0, PIO_SERCOM); //Assign RX function to pin 0
pinPeripheral(1, PIO_SERCOM); //Assign TX function to pin 1
```

must be placed in order to override the standard Arduino pin assignment for this board (digital I/O) and to allow the SERCOM to drive them.

The callback

```
void SERCOM3_Handler()
{
    mySerial.IrqHandler();
}
```

instead is used to allow the real Serial communication since the Serial library relies on interrupts too.