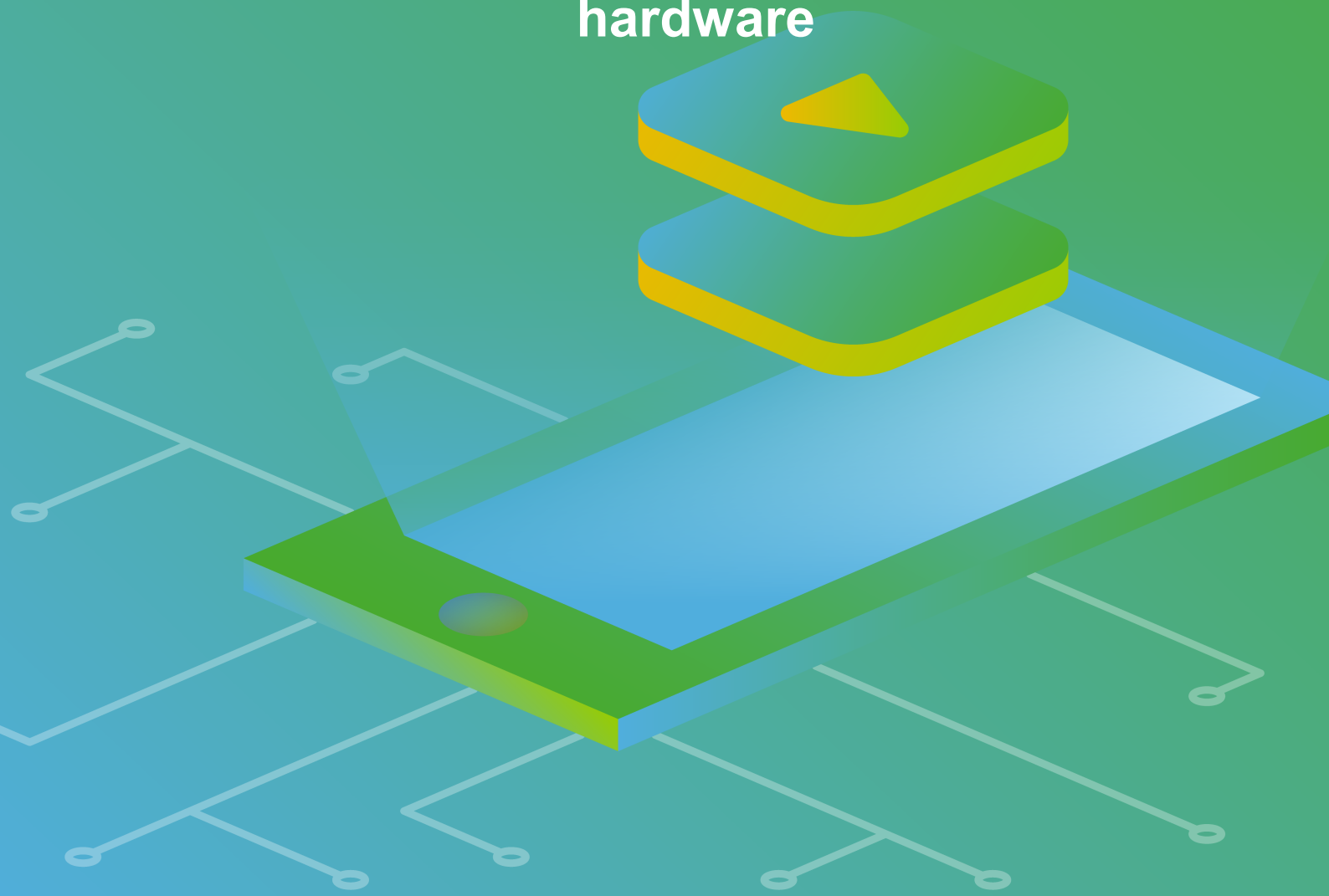
POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

Course

« Introduction to Biomedical Engineering»

Dr. Kirill Aristovich

Section 3: Microcontrollers/Arduino
Lecture 3.2: System components and hardware



System components and hardware

First of all, every modern microcontroller would let you to access its internal features, such as: memory, digital ports, ADC (analog –to-digital converter), DAC, or digital-to-analog converter. Both of those lets you interfacing with analogue electronics, which is handy and exactly what we want. Also, you have the access to interrupt control (we have talk about it in previous lecture), and digital timers and clocks. The other important bit is the ability to transmit information to- and from- other digital devices, such as other controllers and computer. Most of those is usually organized using serial interface.

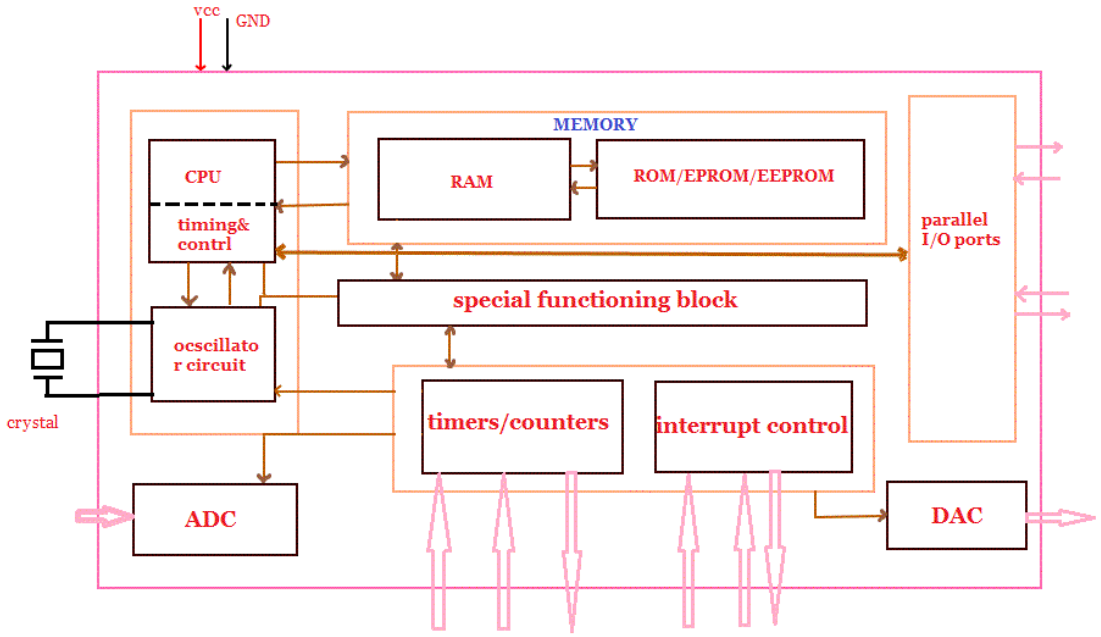


Figure 1 - Microcontroller structure

The general idea for designing embedded system interface is a microcontroller, which has several sensors attached to the inputs, and controls the outputs such as motors, LEDs, speakers, relays, or transistor-based power circuits. In addition, there is external communication to a computer for performing complex calculations, or human interaction.

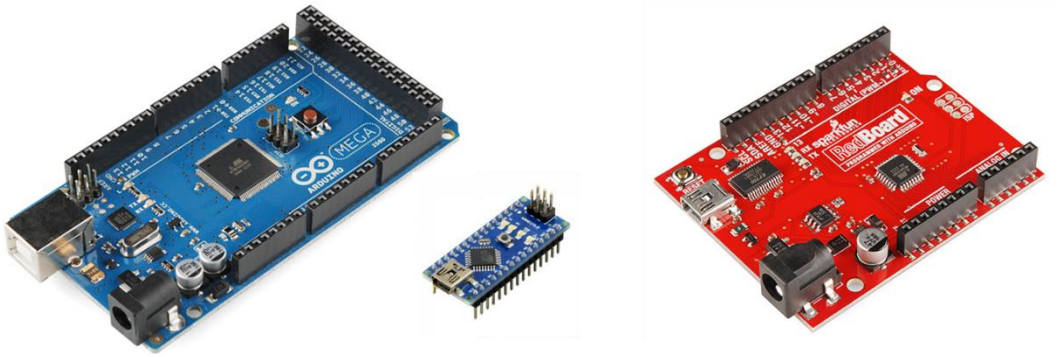


Figure 2 - Arduino boards

You can obviously connect everything you like to your controller using digital and analog pins, but Arduino has a wide range of standard plug-in sensors and components called shields. So you do not need to use kilometers of jumper cables or design complex PCBs in order to create your own systems. Just plug the shields on top of each other and job done.

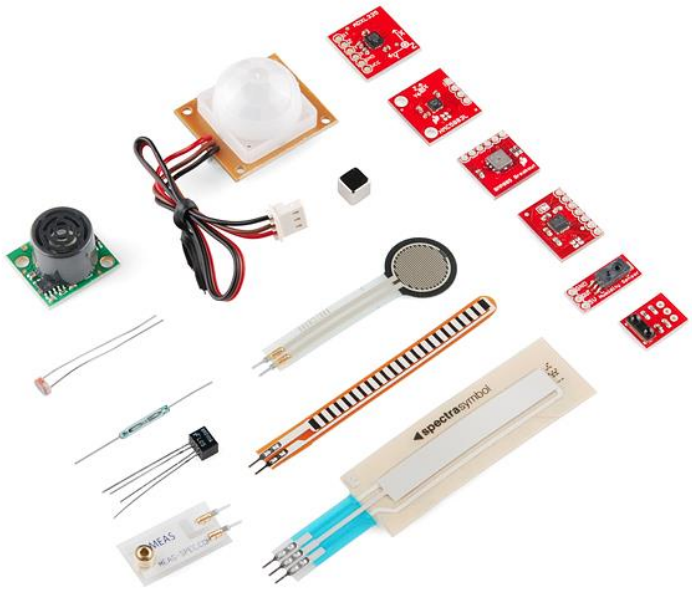


Figure 3 - sensors

Well, enough talking, let's do some examples. You can find information below on what is Tincercad, and how to start working with its free circuit and Arduino simulator. I would advise to try to simulate every diagram and code we are discussing here. This will not require much time, but gives you practical hands-on experience even if you do not physically possess Arduino board right now.

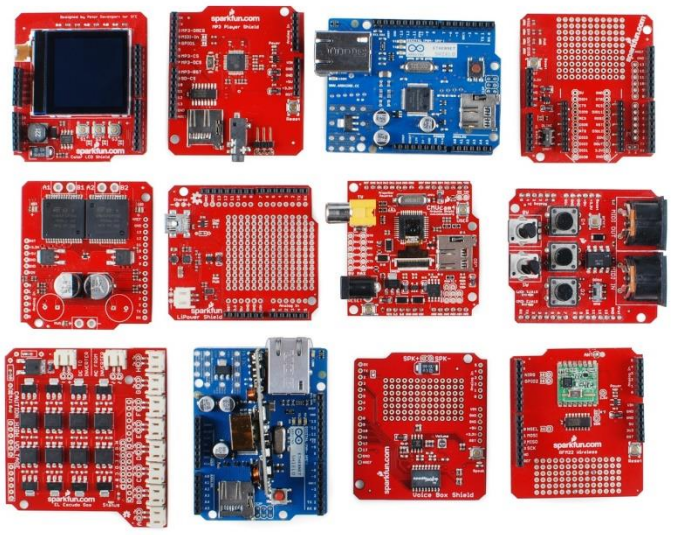


Figure 4 – shields

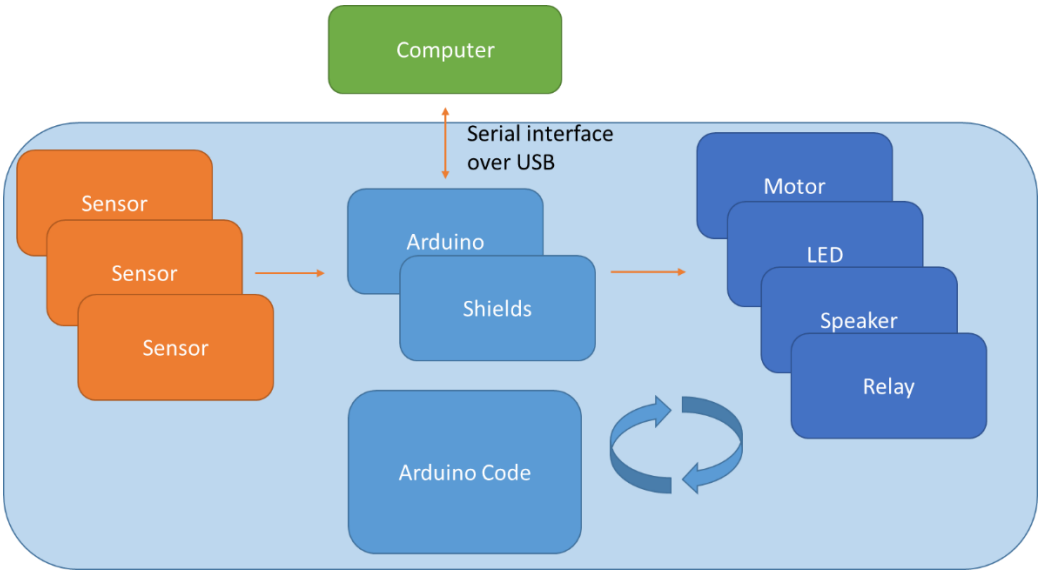
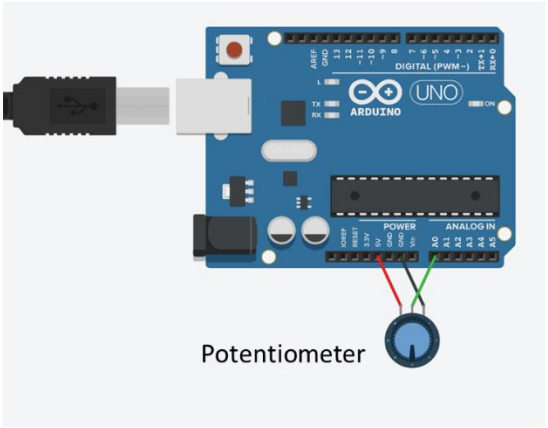


Figure 5 - General idea: real-time embedded/robotic systems

Let's start with exploring the sensor part. The simplest sensor is the potentiometer, or voltage divider. We will connect it between 0 and 5V, and will sense the voltage on the middle pin, which should change from 0 to 5 depending on the knob position. We connect this to the analogue input pin A0.

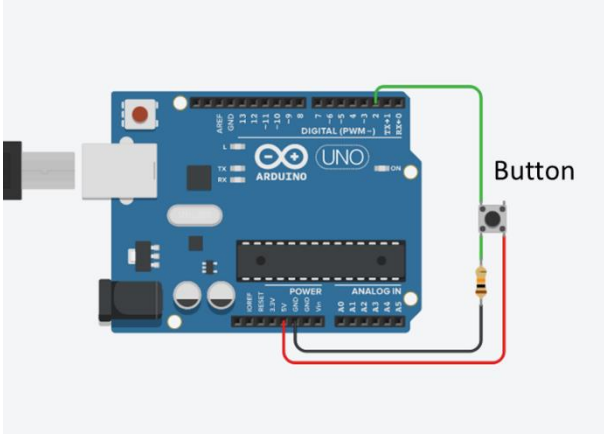



```
int sensorValue = 0;

void setup()
{
  pinMode(A0, INPUT);
  pinMode(13, OUTPUT);
}

void loop()
{
  // read the value from the sensor
  sensorValue = analogRead(A0);
  // turn the LED on
  digitalWrite(13, HIGH);
  // stop the program for the <sensorValue>
  // milliseconds
  delay(sensorValue); // Wait for sensorValue millisecond(s)
  // turn the LED off
  digitalWrite(13, LOW);
  // stop the program for the <sensorValue>
  // milliseconds
  delay(sensorValue); // Wait for sensorValue millisecond(s)
}
```

Figure 6 - Sensors example

This simple attached program changes the frequency of LED blinking depending on the position of the knob by assigning the amount of delay between 'on' and 'off' directly from the voltage value: Arduino has 8-bit ADC, so the range of values received from ADC is from 0 for 0V to 2^8-1 which is 255 for 5V.

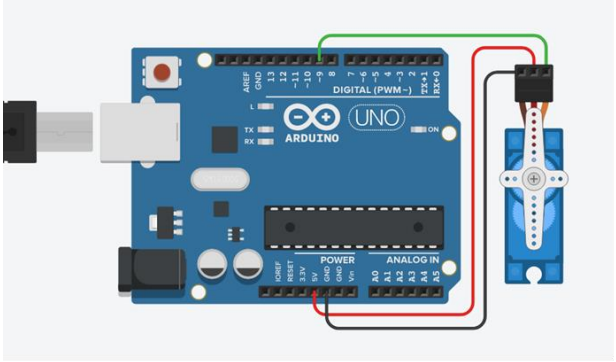
Moving on, here is the simple button indicator, which sends button state to the serial interface.




```
int buttonState = 0;

void setup()
{
  pinMode(2, INPUT);
  Serial.begin(9600);

}

void loop()
{
  // read the input pin
  buttonState = digitalRead(2);
  // print out the state of the button
  Serial.println(buttonState);
  delay(10); // Delay a little bit to improve simulation performance
}
```

Figure 7 - Serial interface: sending stuff

Next here is a bit more sophisticated example of using the sensor: This is a standard ultrasonic range detector. Similar topology that you might use for car cruise control or parking sensor. Again, the whole program only takes several lines and sends the detected distance over serial interface every 100 milliseconds.



```
#include <Servo.h>

int pos = 0;

Servo servo_9;

void setup()
{
  servo_9.attach(9);

}

void loop()
{
  // sweep the servo from 0 to 180 degrees in steps
  // of 1 degrees
  for (pos = 0; pos <= 180; pos += 1) {
    // tell servo to go to position in variable 'pos'
    servo_9.write(pos);
    // wait 15 ms for servo to reach the position
    delay(15); // Wait for 15 millisecond(s)
  }
  for (pos = 180; pos >= 0; pos -= 1) {
    // tell servo to go to position in variable 'pos'
    servo_9.write(pos);
    // wait 15 ms for servo to reach the position
    delay(15); // Wait for 15 millisecond(s)
  }
}
```

Figure 8 - Motors

Speaking of stuff that you can control, here is the basic servo control, which turns the servo slowly 180 degrees, and then back over and over again.

Before we have mentioned serial interface, let's discuss this in detail as we are going to use this over and over again. Serial is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): Serial. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB.

- Serial is used for communication between the Arduino board and a computer or other devices.
- All Arduino boards have at least one serial port (also known as a UART or USART): Serial.
- It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB.

Figure 9 - Serial in depth

Information from Arduino goes through the serial transmit pin, Tx, as a series of pulses or bits (hence serial), following the standard protocol. This information is then stored in the serial buffer on the computer side, and can be accessed from a computer. New information is coming on top of the old one, so the computer can access everything received recently down to the buffer size. Information flows the other direction exactly the same way, ending up being stored temporarily in the Arduino buffer, which can be accessed from the Arduino program.

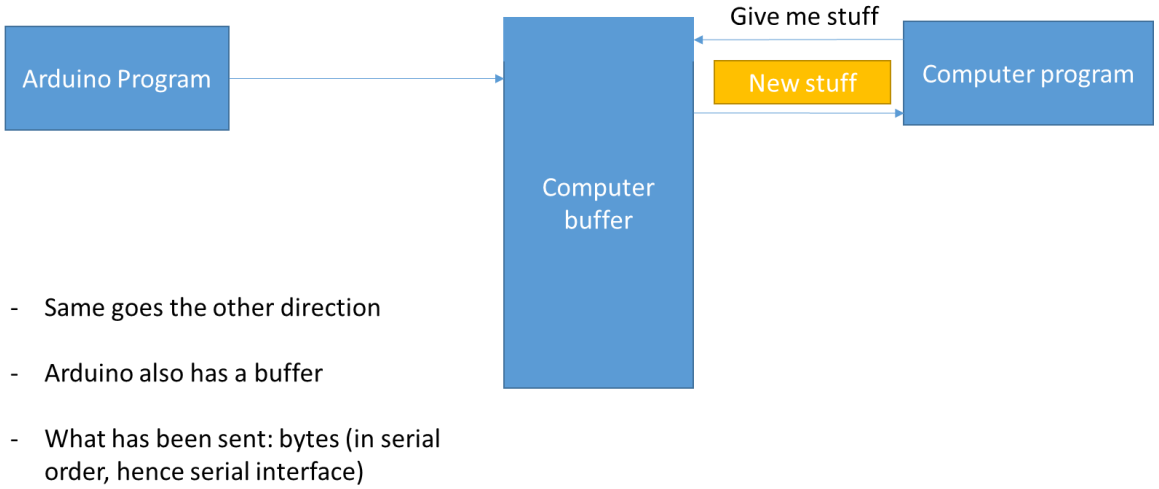


Figure 10 - How it works

In practise it is good to follow some common sense practices to avoid information misinterpretation and bugs: First, consider establishing handshake – a short pre-set message exchange to indicate that you have a connection that you are expecting. Then, consider confirming the reception of each command, and some basic checks for each message to make sure your message gets delivered and stuff is actually happening. Finally, buffer size is limited on either side, so avoid transmitting lots of information and cause buffer overflow.

Here is the simple suggested structure of communications Computer-Arduino, which you do not have to follow but will spare you lots of sleepless nights debugging the misbehaviour of your controller. On the Arduino side there are lots of functions to help you transmit and read serial stuff. Consider reading about those especially paying attention to data types which can be transmitted and how to read them back correctly.

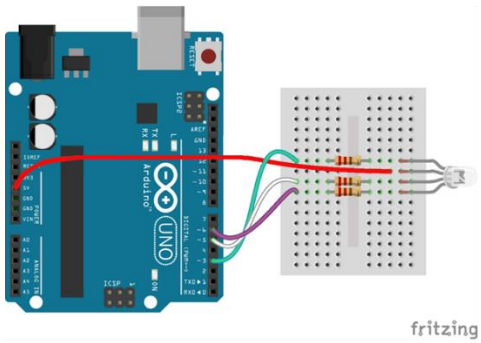
Follow the structure and rules:

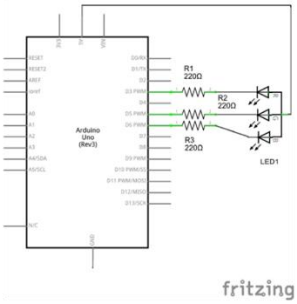
1) Connection, you make sure the Arduino is connected

Then, for each message you want to exchange, you send:

2) The type of order (e.g. “MOTOR”)
3) The parameters (e.g. speed)
4) And then you wait for an acknowledgment of the Arduino to send new orders.

Also, consider running the following example on tinkercad, which reads the serial and lights up the 3-colour LED accordingly.






```
// pins for the LEDs:
const int redPin = 3;
const int greenPin = 5;
const int bluePin = 6;

void setup() {
  // initialize serial:
  Serial.begin(9600);
  // make the pins outputs:
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

}

void loop() {
  // if there's any serial available, read it:
  while (Serial.available() > 0) {

    // look for the next valid integer in the incoming serial stream:
    int red = Serial.parseInt();
    // do it again:
    int green = Serial.parseInt();
    // do it again:
    int blue = Serial.parseInt();

    // look for the newline. That's the end of your sentence:
    if (Serial.read() == '\n') {
      // constrain the values to 0 - 255 and invert
      // if you're using a common-cathode LED, just use "constrain(color, 0, 255);"
      red = 255 - constrain(red, 0, 255);
      green = 255 - constrain(green, 0, 255);
      blue = 255 - constrain(blue, 0, 255);

      // fade the red, green, and blue legs of the LED:
      analogWrite(redPin, red);
      analogWrite(greenPin, green);
      analogWrite(bluePin, blue);

      // print the three numbers in one string as hexadecimal:
      Serial.print(red, HEX);
      Serial.print(green, HEX);
      Serial.println(blue, HEX);
    }
  }
}
```

Figure 11 - Example: read ASCII string

Couple of advises on debugging the serial: Use structures you can read and understand or have designed yourself. This way you know exactly what you expect. Particularly, ask yourself the following: are you sending raw bytes or strings? Did you convert strings to bytes? How about Integers? Floating point? Data converters will help you, also knowledge of the data types and their byte composition helps! Finally, Arduino IDE has built-in terminal, which lets you see what is in the buffer and send stuff to Arduino buffer from a computer. It is located in tools-> serial monitor.

In the end of this lecture, let's have a bit of fun applying some of the stuff we have learned before in Tinkercad. We will build a non-inverting amplifier based on the LM741 OpAmp located among standard components within the tinkercad environment. Adding 2 resistors, batteries, function generator for the input and monitoring the output with virtual scope, we can physically see how the input signal gets amplified, and use our knowledge about control theory and electronics to play around with the gain and bandwidth by changing the resistor values.

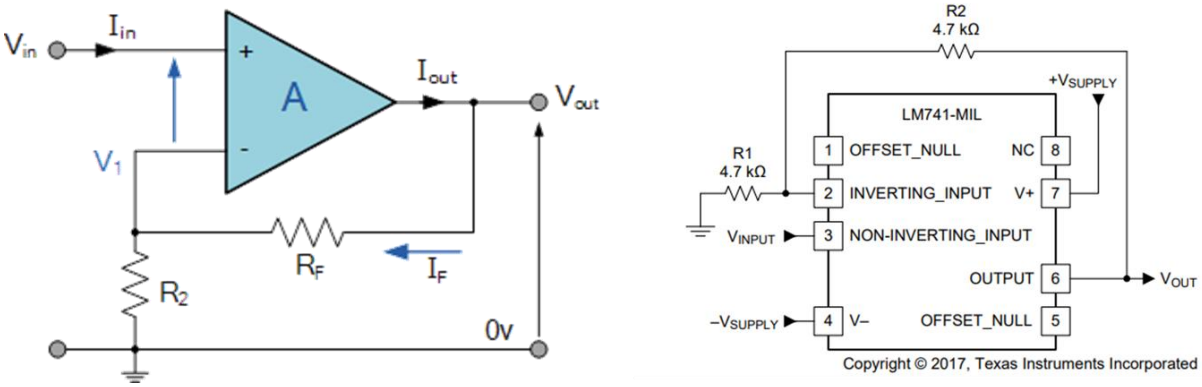


Figure 12 - Non-inverting amplifier based on OpAmp LM741

Here is the diagram and simple code for using Arduino to see the amplified signal and transmit it over the serial interface. I trust you will spend several minutes to understand it and play around with resistors.

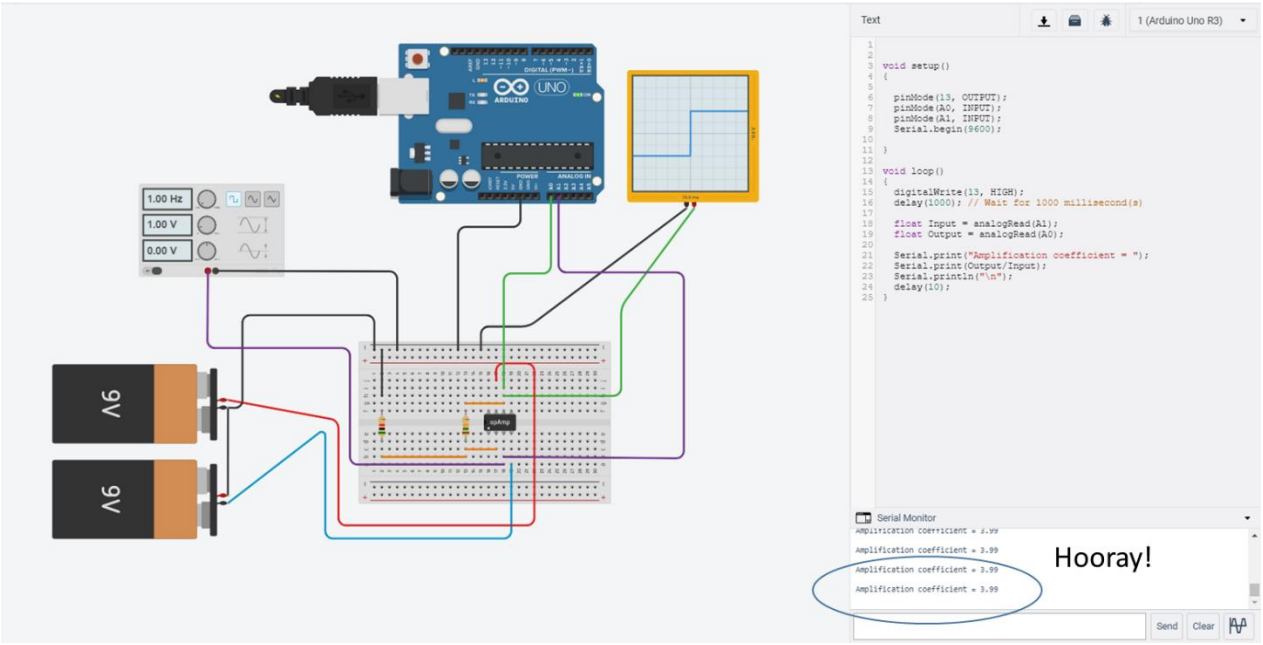


Figure 13 - AMP on arduino

Some graphic material used in the course was taken from publicly available online resources that do not contain references to the authors and any restrictions on material reproduction.

This course was developed with the support of
the "Open Polytech" educational project

Online courses from the top instructors of SPbPU