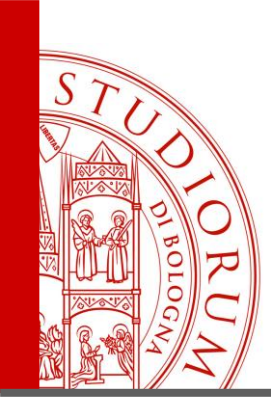


MEASUREMENTS WITH ARDUINO

Ing. Paolo Guidorzi

Topics covered

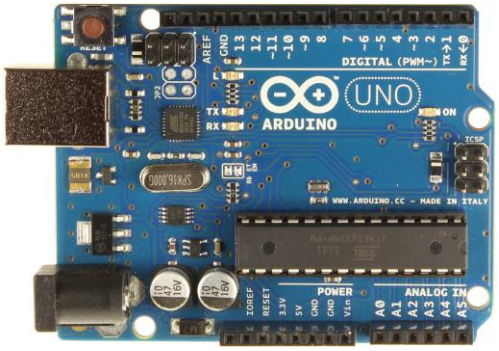
- Introduction: what is Arduino
- Embedded systems, microcontrollers, sensors, home automation...
- Ohm's law, resistors, LEDs, opamp, sensors, I^2C and SPI protocols
- Arduino : hardware, firmware, software (the "sketch"), the community
- Arduino and the outside world: analog and digital ports, the serial interface
- Sampling Theorem, Nyquist frequency, Antialiasing filters
- Arduino's programming language and its development environment
- The first experiments, breadboards, Prototype Board, PCB
- Reading the value of a potentiometer
- From the value of a potentiometer to the PWM output
- Brightness of an LED
- Continuously variable PWM output
- From the PWM signal to a direct voltage. «Poor man DAC»
- Using a button. Pull-up and pull-down resistors
- The voltage divider
- Resistive sensors: The photocell
- Resistive sensors: GAS sensor

MEASUREMENTS WITH ARDUINO

- The temperature sensor LM35 (analog)
- The thermocouple K temperature sensor (MAX6675 digital interface)
- The temperature and humidity sensors DHT11 / DHT22 (digital interface)
- The temperature sensor DS18B20 (1-Wire digital interface)
- Serial 7-segment display
- Color TFT graphic display (2.8") using Arduino UNO
- Connection of another TFT graphic display (1.8") using Arduino UNO
- Connection of another TFT graphic display (2.2") using Arduino MEGA2560
- External DAC and ADC converters
- Temperature and pressure sensor
- Bidirectional data communication between Arduino and computer (via serial port)
- Data communication from smartphone to Arduino via Bluetooth
- Using of a Multiplexer
- Example of a complete realization: the electronic nose:
  - Graphic display
  - 16 bit ADC - Humidity temperature pressure sensor
  - Data storage on microSD and real-time clock
  - Multiplexer
  - Gas sensors

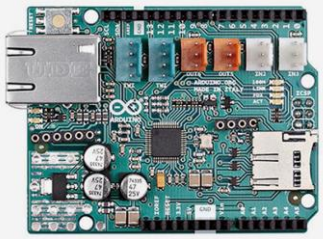
MEASUREMENTS WITH ARDUINO

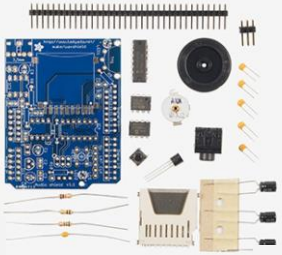
- Arduino is a rapid prototyping tool that allows to create small stand-alone interactive systems.
- It was created for artists, designers, scholars, researchers or anyone who needs a smart tool for a specific application.
- Both the Arduino hardware and software are open source, and many ready-made projects can be freely found online (for example on the www.arduino.cc and www.arduino.org sites)
- It is easy to use, you don't need to be an electronic engineer



MEASUREMENTS WITH ARDUINO

- It is programmed from computer with a language similar to C
- It has various inputs and outputs (digital and analog)
- It can be expanded using the so-called *Shields*, which expand the possibilities of use and interfacing with other devices and sensors
- Custom expansions for particular uses can be created by users

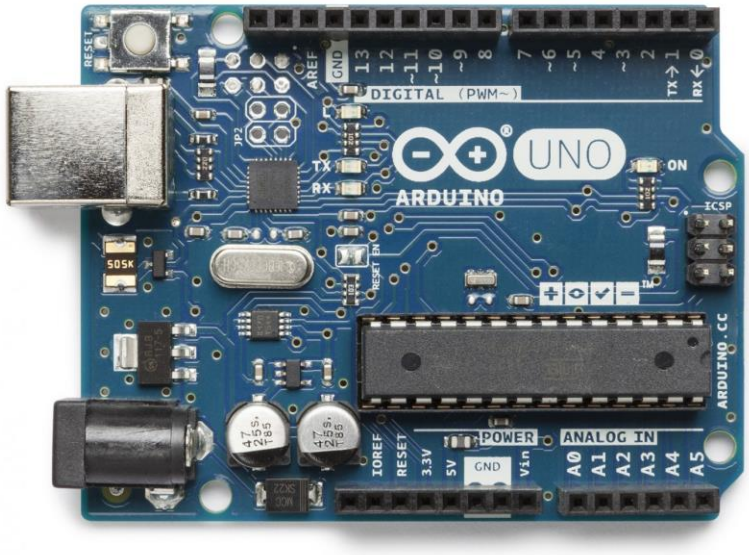




MEASUREMENTS WITH ARDUINO

There are many versions of Arduino, with different numbers of inputs and outputs and different computing powers. They all share the same programming language and environment.

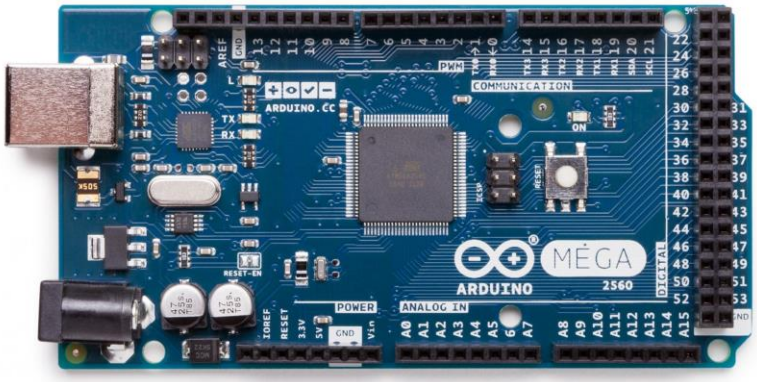
Arduino UNO:



Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

MEASUREMENTS WITH ARDUINO

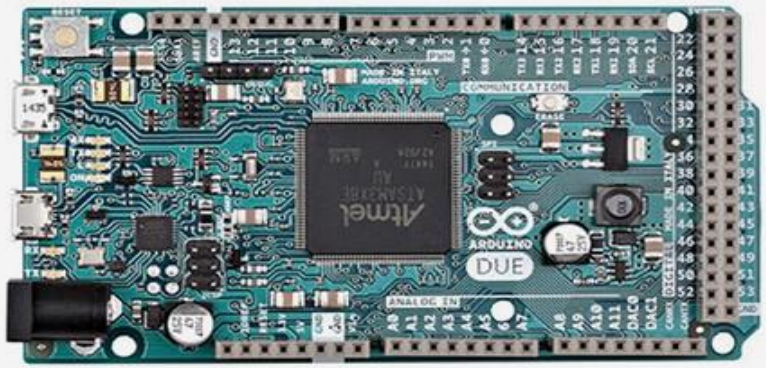
Arduino MEGA2560:



Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

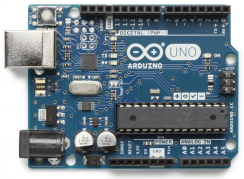
MEASUREMENTS WITH ARDUINO

Arduino DUE:

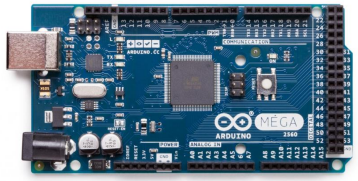


Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

MEASUREMENTS WITH ARDUINO

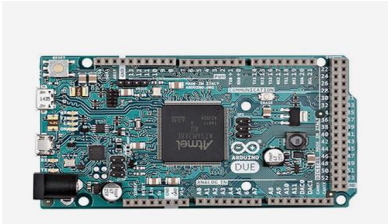


Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

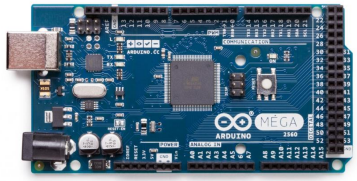


Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

MEASUREMENTS WITH ARDUINO

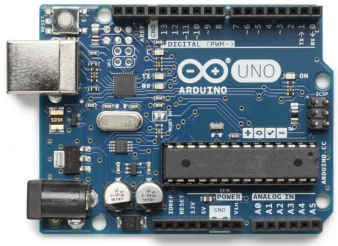


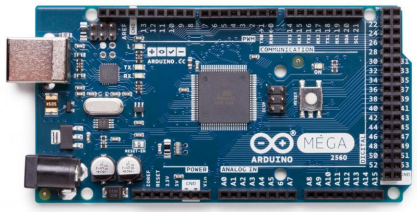
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

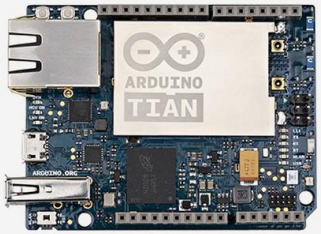


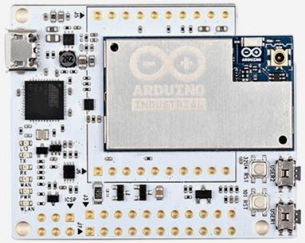
Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

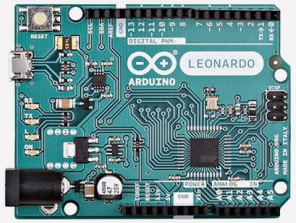
MEASUREMENTS WITH ARDUINO

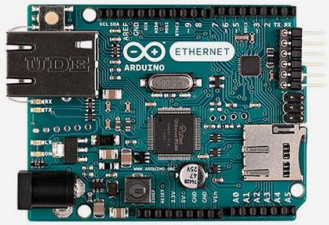


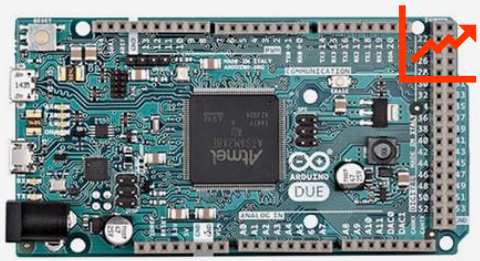












MEASUREMENTS WITH ARDUINO

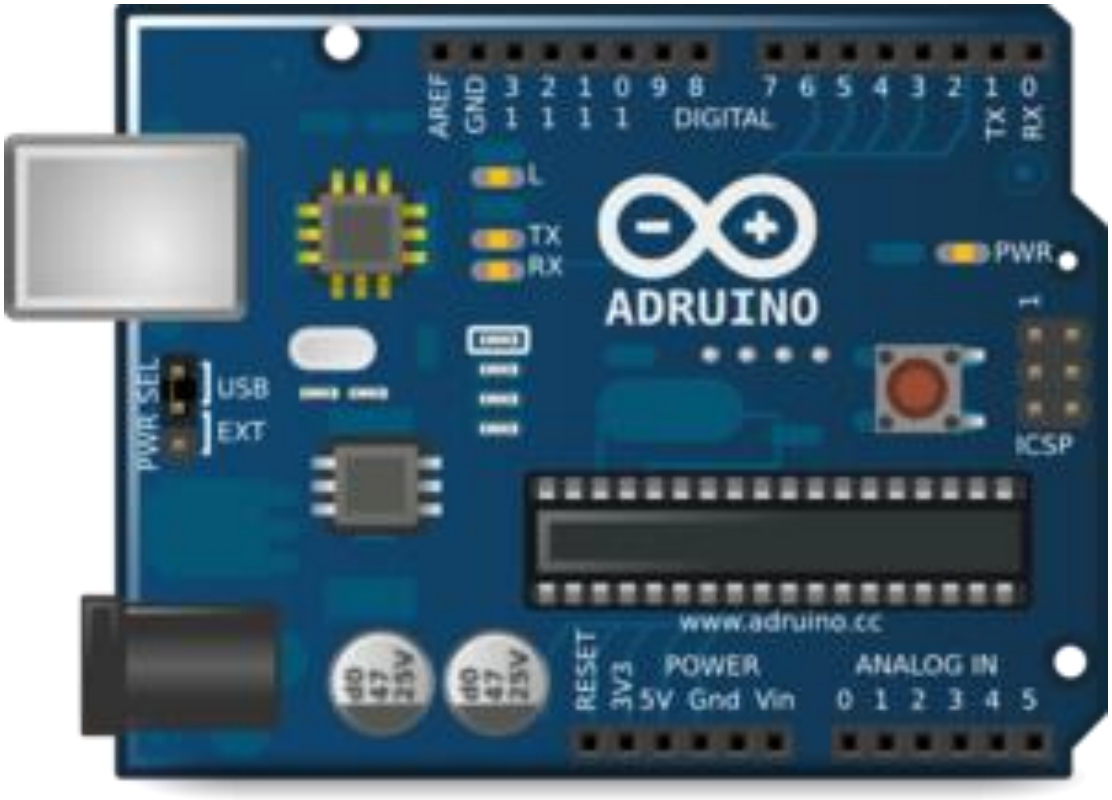
Compare board specs

This simple table shows a quick comparison between the characteristics of all the Arduino boards.

Name	Processor	Operating Voltage/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [KB]
Uno	ATmega328	5 V/7-12 V	16 Mhz	01/06/00	14/06/14	1
Due	AT91SAM3X 8E	3.3 V/7-12 V	84 Mhz	12/02/14	54/12	-
Leonardo	ATmega32u4	5 V/7-12 V	16 Mhz	01/12/00	20/07/14	1
Mega 2560	ATmega2560	5 V/7-12 V	16 Mhz	16/0	54/15	4
Mega ADK	ATmega2560	5 V/7-12 V	16 Mhz	16/0	54/15	4
Micro	ATmega32u4	5 V/7-12 V	16 Mhz	01/12/00	20/07/14	1
Mini	ATmega328	5 V/7-9 V	16 Mhz	01/08/00	14/06/14	1
Nano	ATmega168 ATmega328	5 V/7-9 V	16 Mhz	01/08/00	14/06/14	0.512 1
Ethernet	ATmega328	5 V/7-12 V	16 Mhz	01/06/00	14/04/14	1
Esplora	ATmega32u4	5 V/7-12 V	16 Mhz	-	-	1
ArduinoBT	ATmega328	5 V/2.5-12 V	16 Mhz	01/06/00	14/06/14	1
Fio	ATmega328P	3.3 V/3.7-7 V	8 Mhz	01/08/00	14/06/14	1
Pro (168)	ATmega168	3.3 V/3.35-12 V	8 Mhz	01/06/00	14/06/14	512
Pro (328)	ATmega328	5 V/5-12 V	16 Mhz	01/06/00	14/06/14	1
Pro Mini	ATmega168	3.3 V/3.35-12 V 5 V/5-12 V	8 Mhz 16Mhz	01/06/00	14/06/14	512
LilyPad	ATmega168V ATmega328V	2.7-5.5 V/2.7-5.5 V	8 Mhz	01/06/00	14/06/14	512
LilyPad USB	ATmega32u4	3.3 V/3.8-5V	8 Mhz	01/04/00	09/04/14	1
LilyPadSimple	ATmega328	2.7-5.5 V/2.7-5.5 V	8 Mhz	01/04/00	09/04/14	1
LilyPadSimpleS	ATmega328	2.7-5.5 V/2.7-5.5 V	8 Mhz	01/04/00	09/04/14	1

MEASUREMENTS WITH ADRUINO

Hardware e software open source..



MEASUREMENTS WITH ARDUINO

Arduino was born in 2005 from another platform for simplified prototyping, Wiring, created for non-experts, by Hernando Barragan, of which Massimo Banzi, creator of Arduino, was a supervisor.

Massimo Banzi, David Cuartielles, David Mellis, Tom Igoe, Gianluca Martino

The name Arduino comes from the name of a café in Ivrea where the team met in their free time. Arduino d'Ivrea was king of Italy from 1002 to 1014

Who are Arduino users?

Makers *are interesting people: they are not nerds, they are rather cool guys who are interested in technology, design, art, sustainability, alternative business models. They live on online communities, open source software and hardware but also on the dream of inventing something to produce on their own, to live off their own inventions. In a time of crisis they invent their work instead of looking for a classic one.*

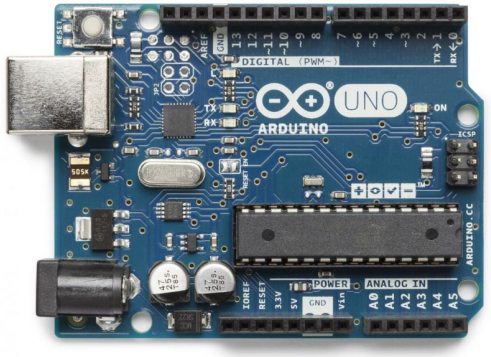
(from an interview with Massimo Banzi on *Wired*)

- **Computer**: general purpose, suitable for different needs. Does everything without being optimized for a particular task
- **Microprocessor**: data processing unit (generic). Requires external elements (RAM, peripherals, bus, etc.) to operate. In computers there is a microprocessor
- **DSP**: microprocessor specialized in mathematical operations related to signal processing
- **FPGA**: array of programmable logic ports, in some ways similar to the DSP
- **Microcontroller**: chip that includes all the main elements to work (volatile and non-volatile memory, bus, inputs and outputs, etc.)
- **Embedded system**: "intelligent" electronic device created for a precise function (e.g. a thermostat, a remote control). Created and optimized for a single task
- **Arduino** uses a microcontroller, so the board requires only few additional components to work (a quartz, i.e. the system clock, some voltage regulators, a chip to communicate with the computer via USB and little else)

MEASUREMENTS WITH ARDUINO

When to use a general purpose device (computer) and when a specialized one (embedded system) like Arduino?

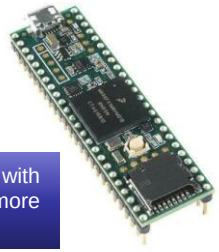
- Portability
- Special needs
- Miniaturization
- Energy Efficiency



Other examples of small embedded systems commonly used by hobbyists and experimenters:

Raspberry

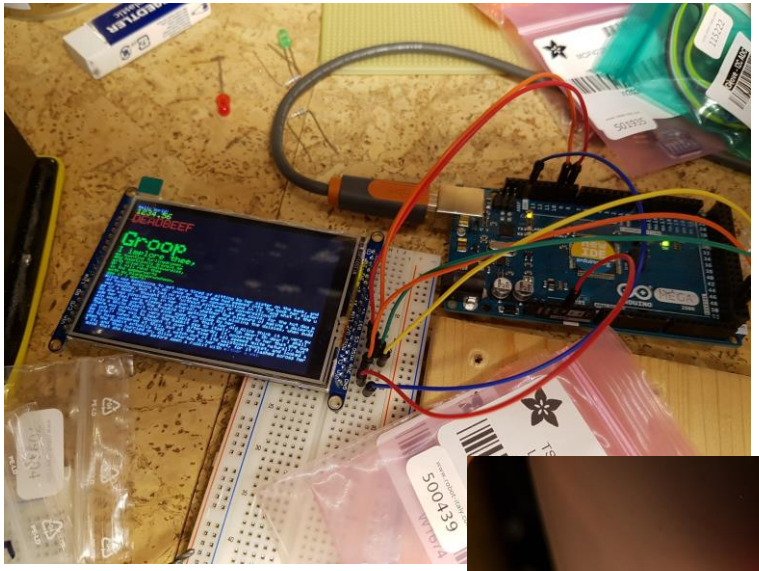
Teensy



Partially compatible with Arduino (but much more powerful)!

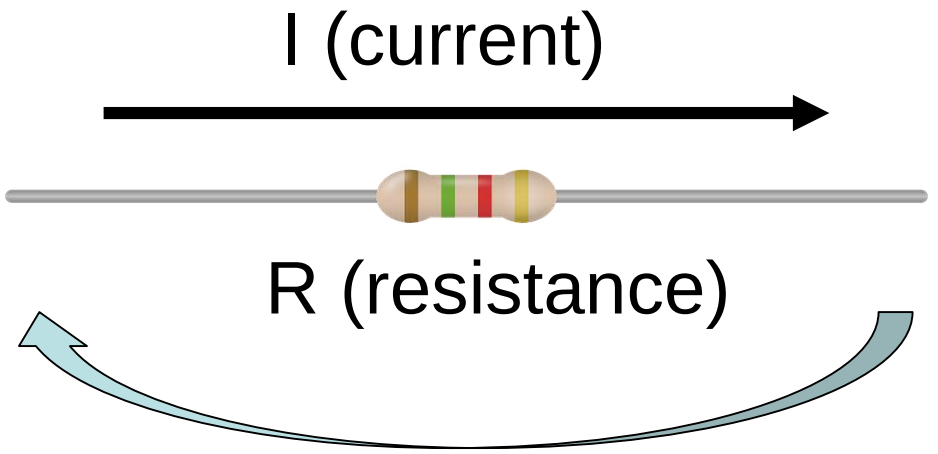
PIC

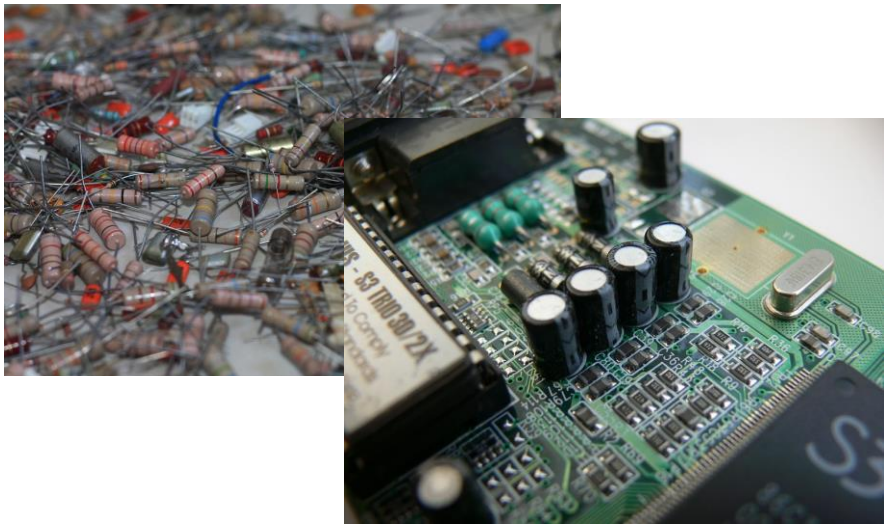
MEASUREMENTS WITH ARDUINO

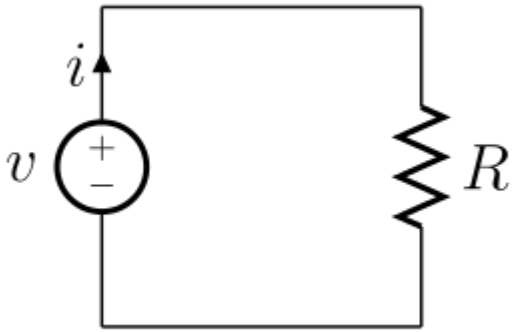


Ohm's Law V = R * I

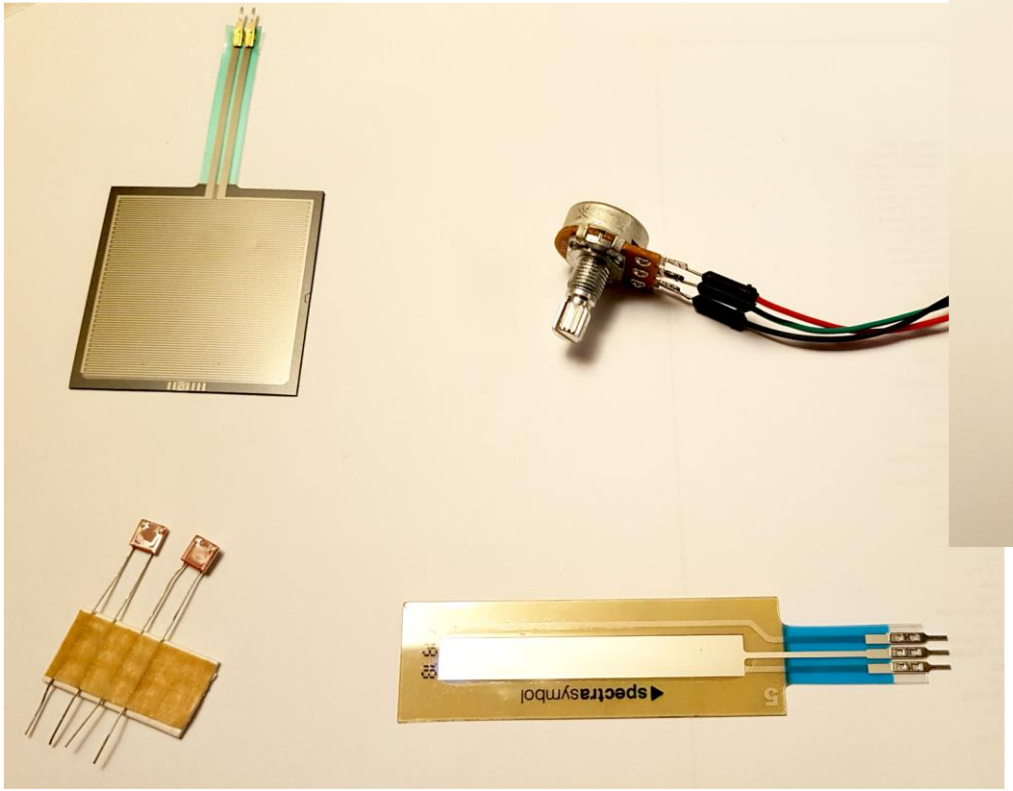
V Volt
I Ampere
R Ohm

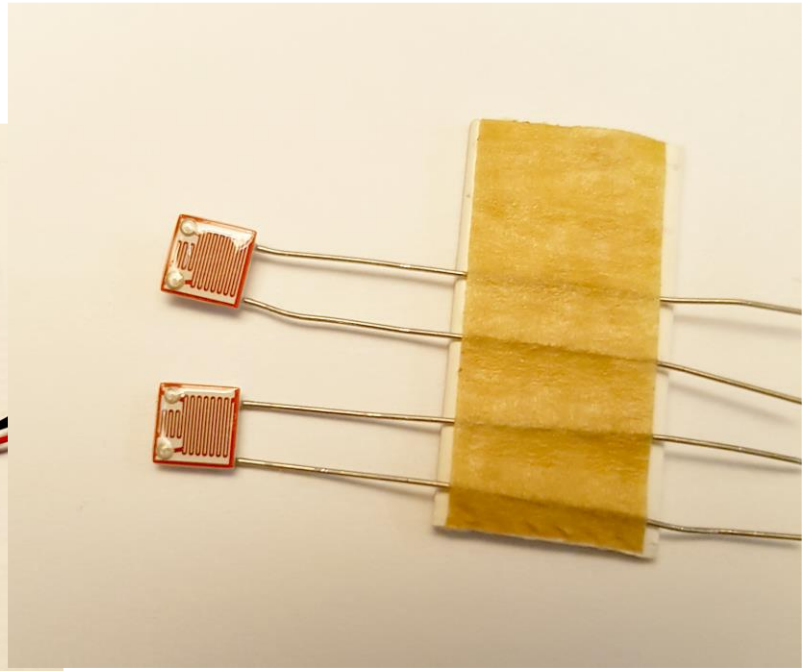






Variable resistors of various types

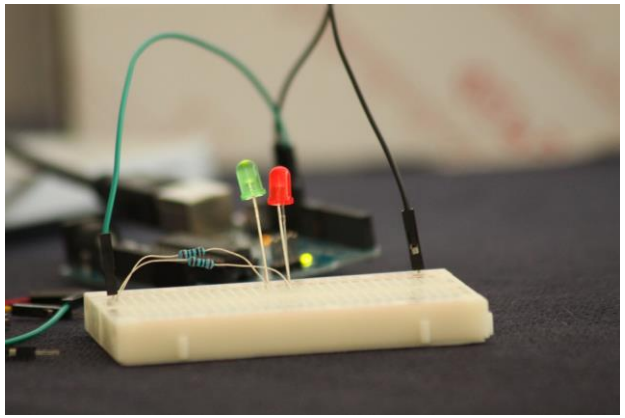


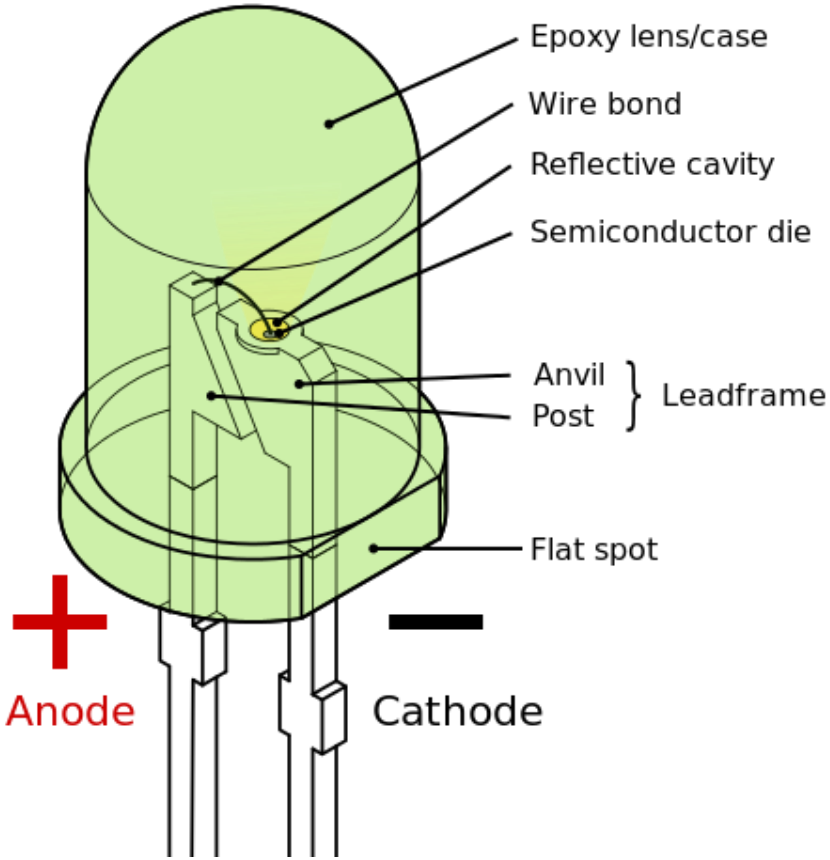


Force sensor, potentiometer, photocell, linear resistive sensor

LED (Light-Emitting Diode)

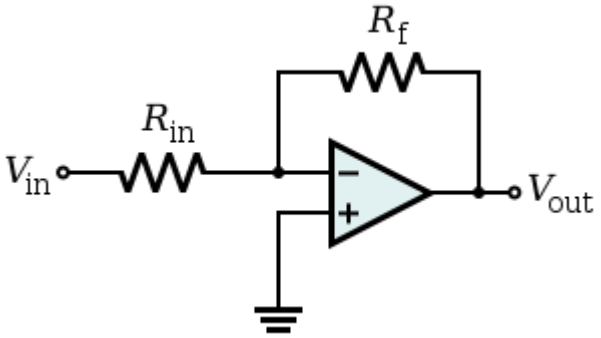
- It's a diode
- The current runs through it only in one direction
- The current must be limited using a resistance in series





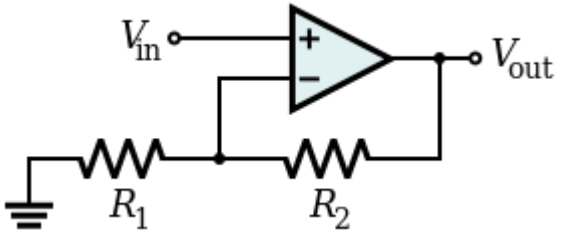
Operational Amplifiers

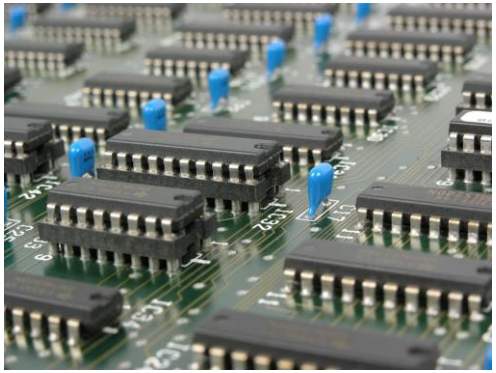
Inverting configuration: $V_{out} = -\frac{R_f}{R_{in}} V_{in}$

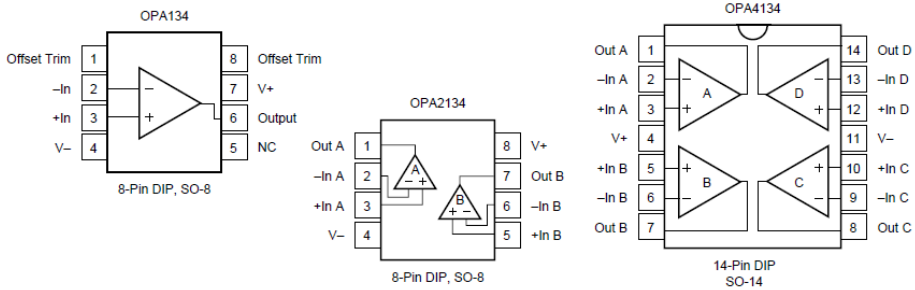


Non-inverting configuration: $V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}$

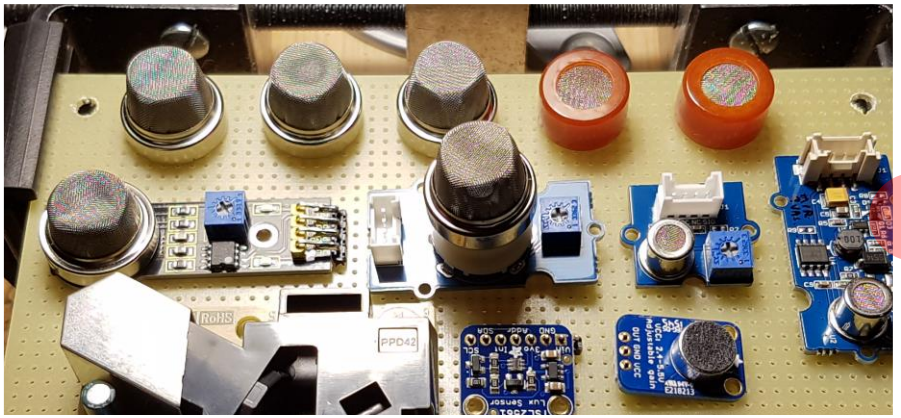
(if R1=infinite and R2=0 a unit gain buffer is obtained)

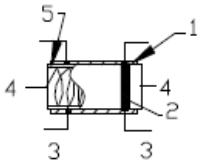


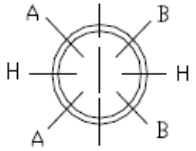


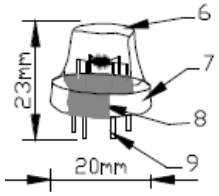


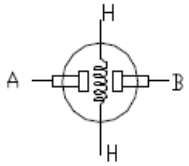
Gas sensors (analog)

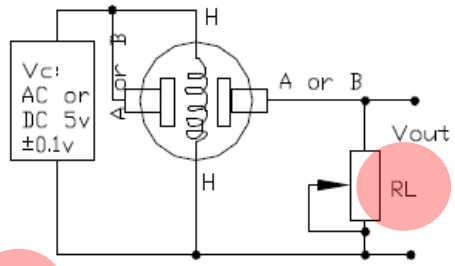








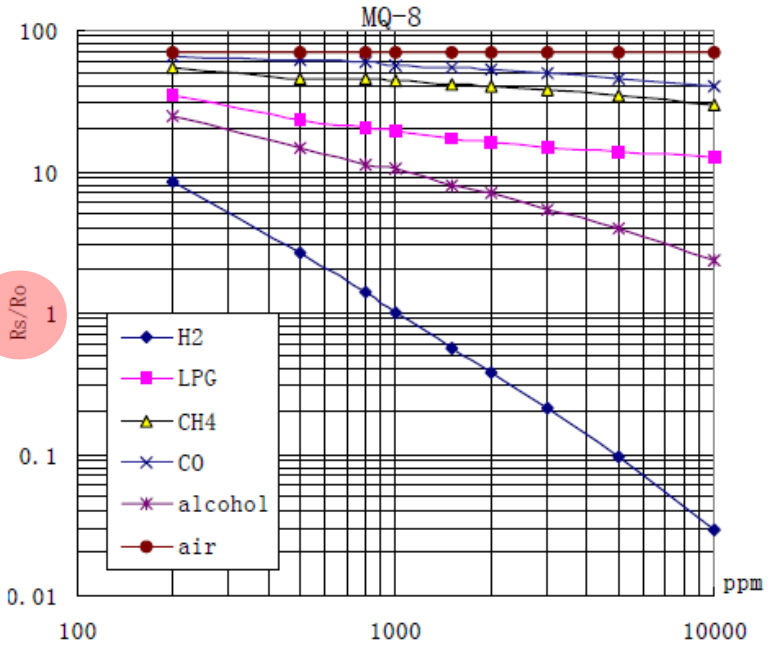


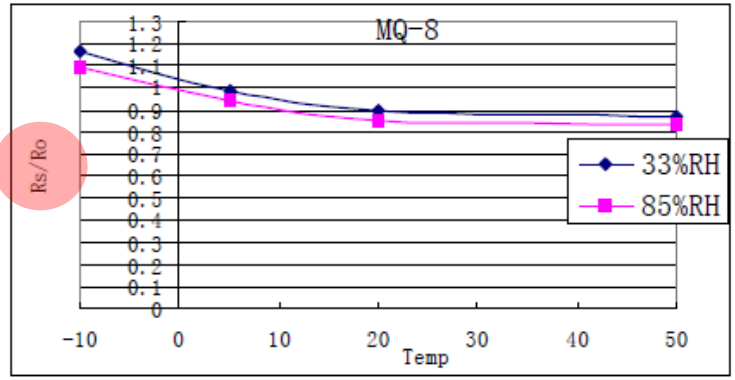


Ro: sensor resistance at 1000ppm of H_2 in the clean air.

Rs:sensor resistance at various concentrations of gases.

Rs	Sensing Resistance	10K Ω - 60K Ω (1000ppm H_2)





Digital communication devices

The most commonly used protocols for communication between intelligent electronics devices inside of a circuit are **I²C** e **SPI**

I²C: developed in the late ‘70s

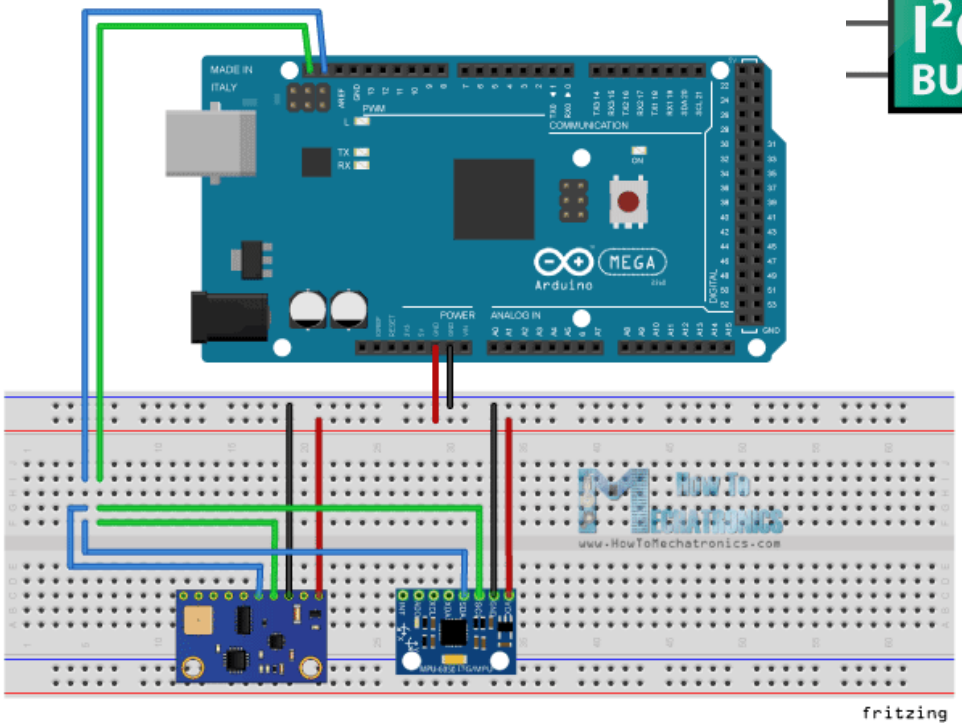
2 wires bus: **SDA** (Serial Data line)
SCL (Serial Clock line)

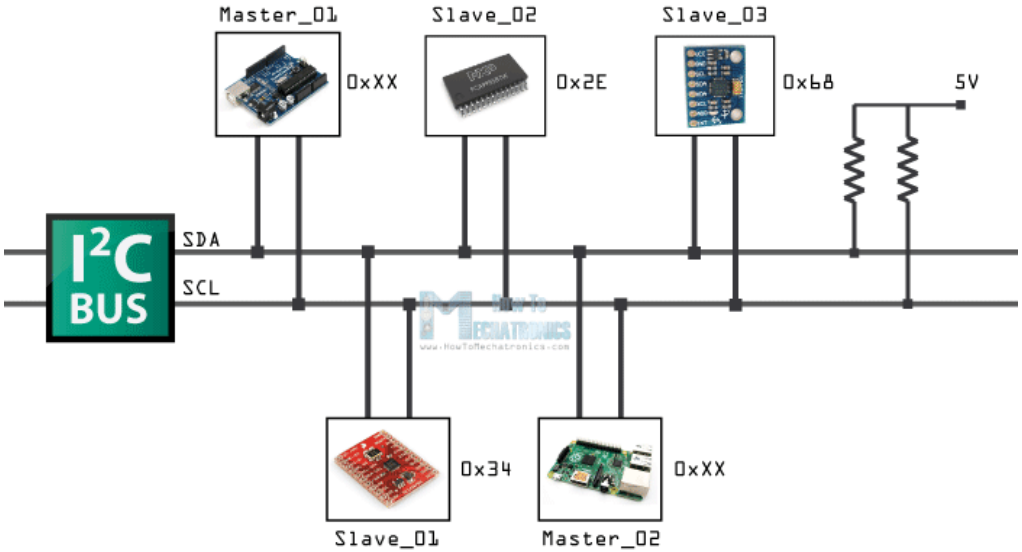
the devices on the bus are connected to these 2 wires

SPI: **S**erial **P**eripheral **I**nterface

4 wires bus: **MOSI** (Master Out Slave In), **MISO** (Master In Slave Out), **SCK** (Clock), **SS** (Slave Select, **SS1**, **SS2**, .., **SSn**)

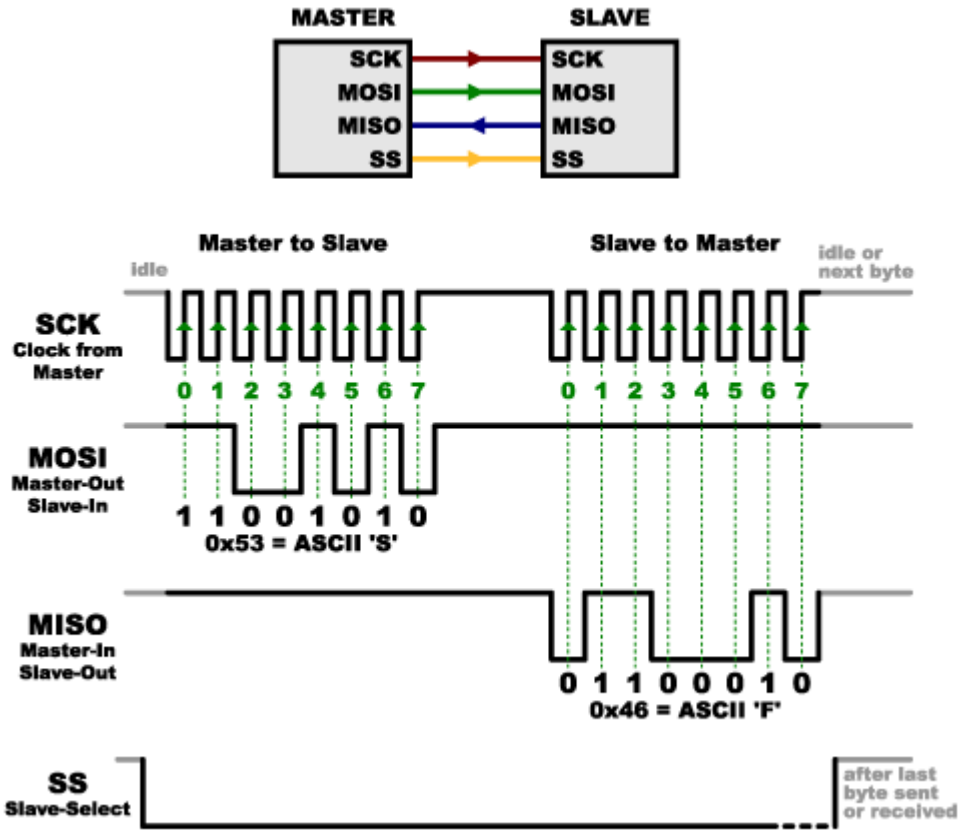
BUS I²C

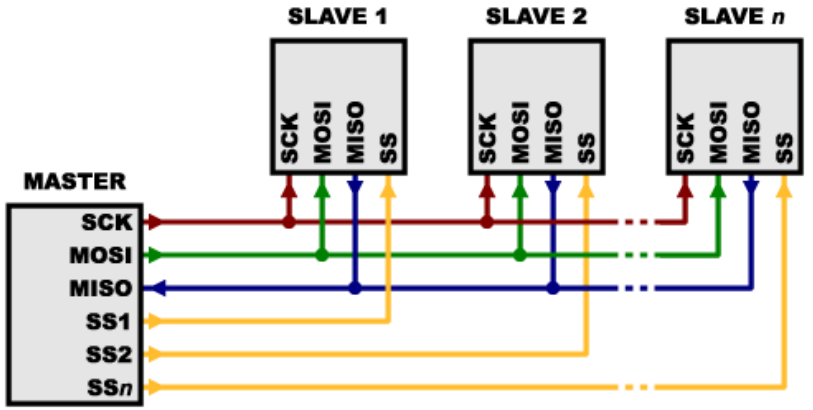


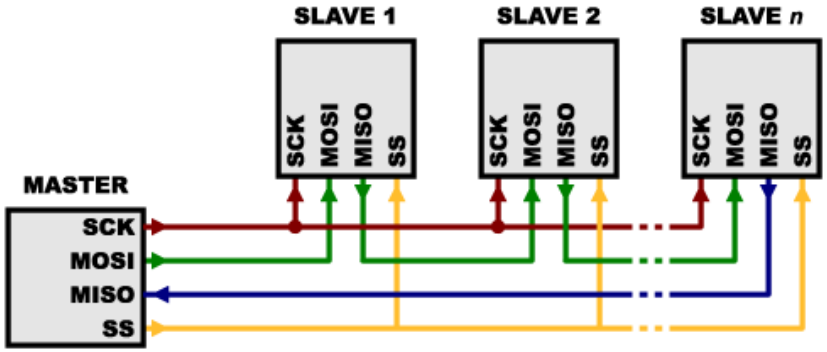


http://howtomechatronics.com/tutorials/arduino/how-i2c-communication-works-and-how-to-use-it-with-arduino/

BUS SPI







https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

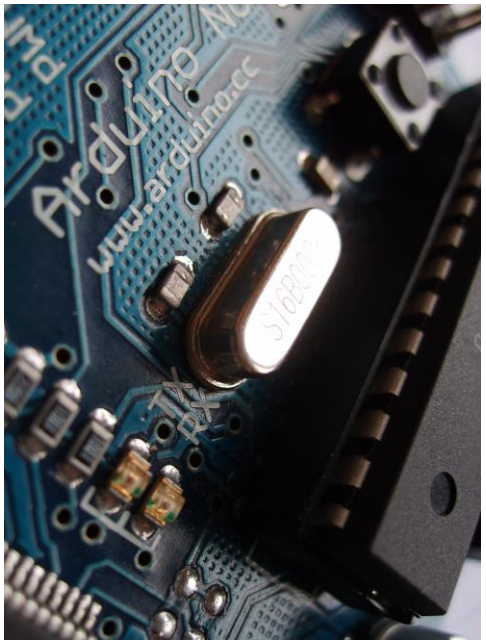
Comparison between BUS I²C and BUS SPI

- Both are Master / Slave type. The Master always starts communication

- **I²C** uses 2 wires: **SDA** (Serial Data line) and **SCL** (Serial Clock line).
  It's relatively slow (100-400 kHz)
  There may be multiple Masters and Slaves on the line

- **SPI** uses 4+ wires: **MOSI** (Master Out Slave In), **MISO** (Master In Slave Out), **SCK** (Clock), **SS** (Slave Select, **SS1**, **SS2**, .., **SSn**).
  It's fast, it can go up to 25 MHz
  One Master and multiple Slaves

Arduino is made of 3 elements:

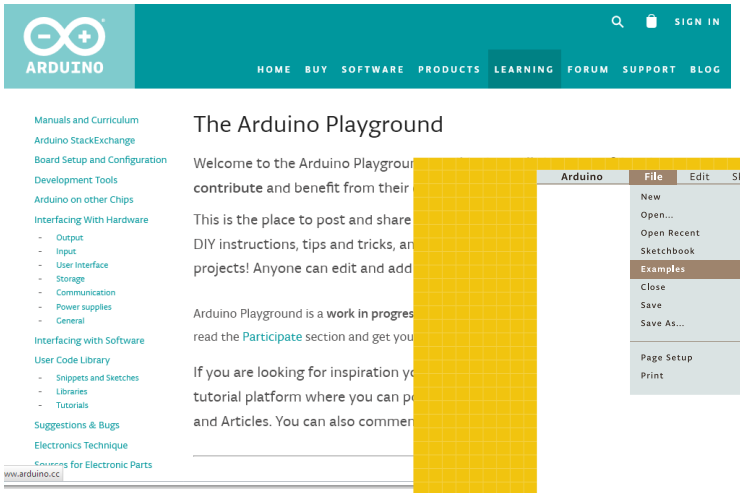


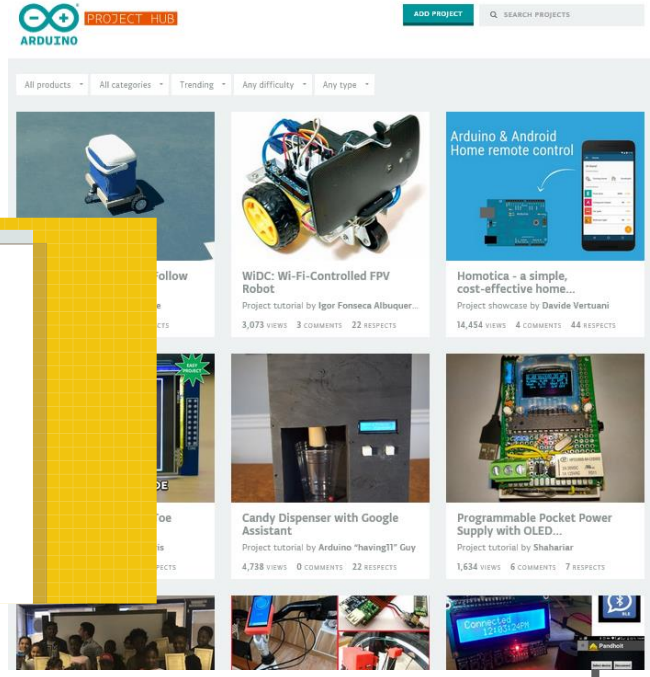
- Hardware
- Software
- Community

1. Hardware, i.e. the Arduino physical boards, can vary in terms of number of ports in/out or the power of the microcontroller, but they are all programmed with the same language (simplified C) and through the same development environment (IDE).
2. The software loaded on the microcontroller consists of 2 parts: a firmware, which remains resident and unchanged (similar to the computer BIOS) and performs the basic functions, including allowing communication with the computer via USB port and loading the software developed by the user, and the user program ("sketch").

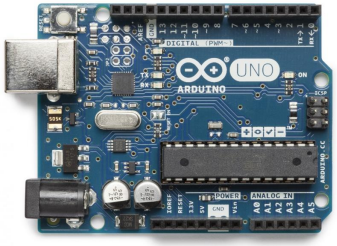
MEASUREMENTS WITH ARDUINO

3. The community is Arduino's real strength. The repository of other users' projects and the forum of the official website are good starting points not to start from scratch. The open source philosophy of the whole Arduino ecosystem pushes and invites the user to share in an open and free way (with various types of licenses) his projects with the whole community. Moreover, the development environment (the "IDE" of Arduino) already includes thousands of working examples.





Arduino UNO connections to and from the outside world

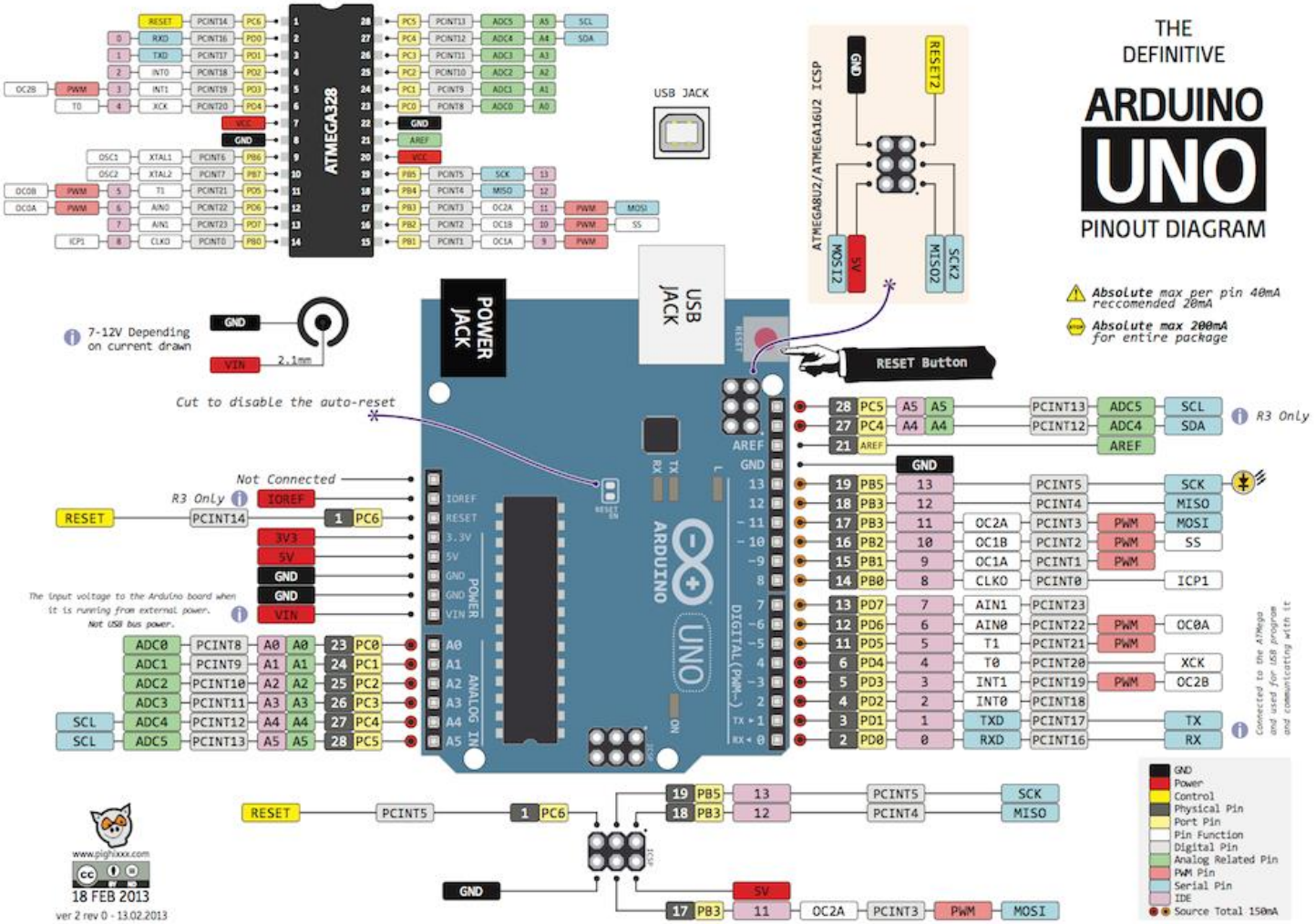


Atmega168 Pin Mapping

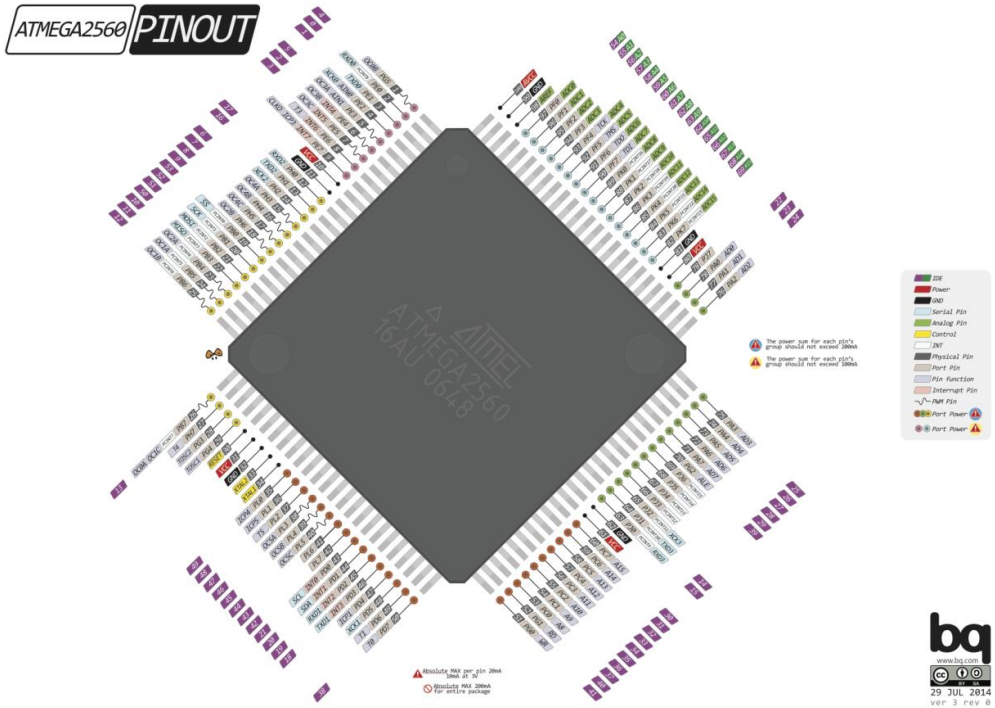
Arduino function					Arduino function
reset	(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22	GND	GND
GND	GND	8	21	AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16	PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

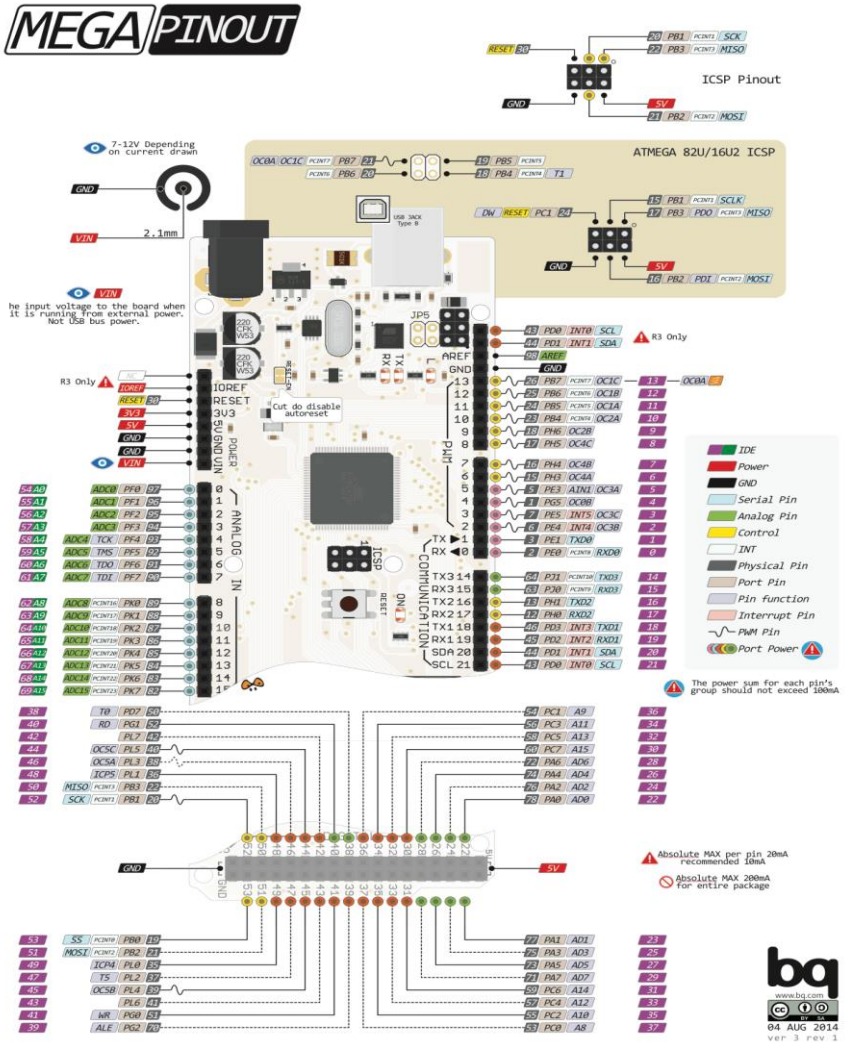
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

MEASUREMENTS WITH ARDUINO



www.pighixxx.com

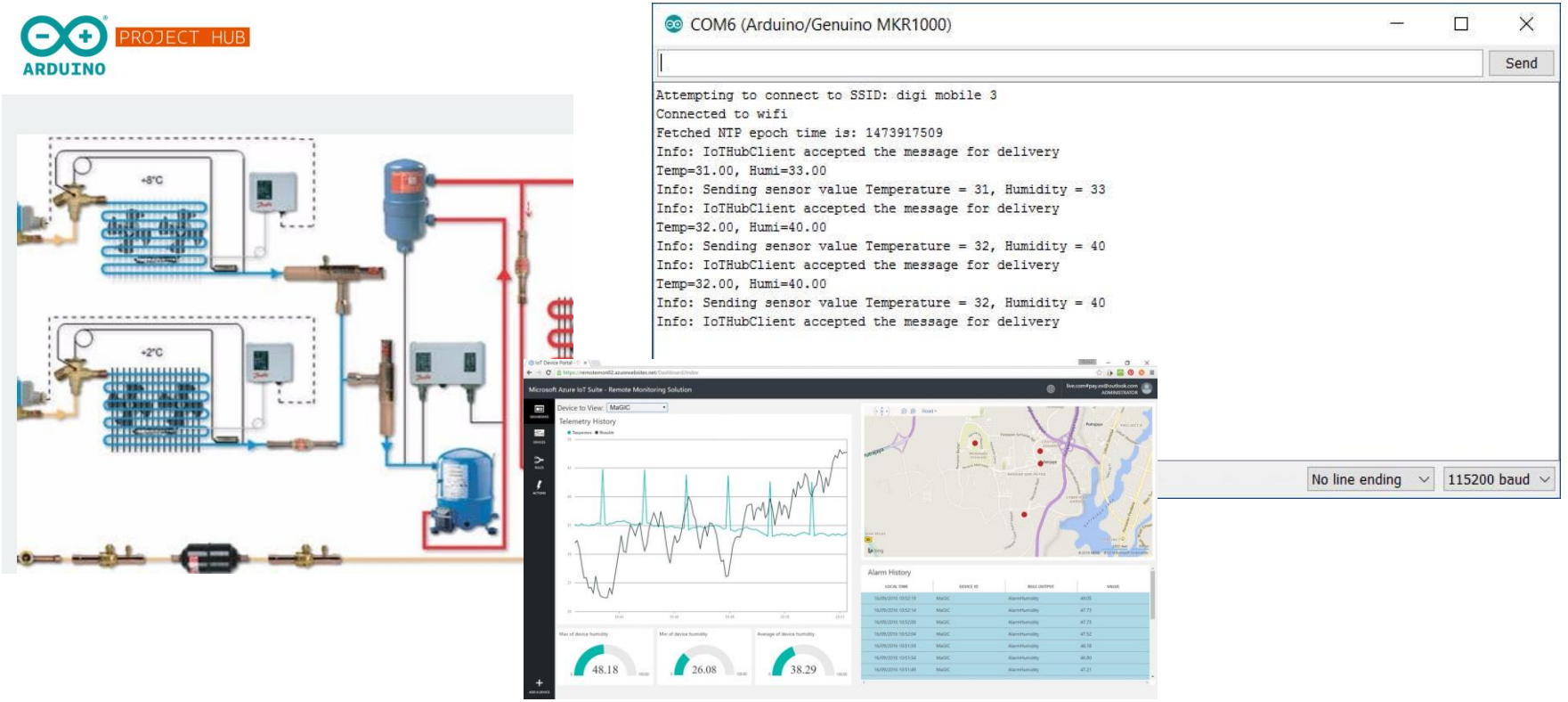




www.pighixxx.com

MEASUREMENTS WITH ARDUINO

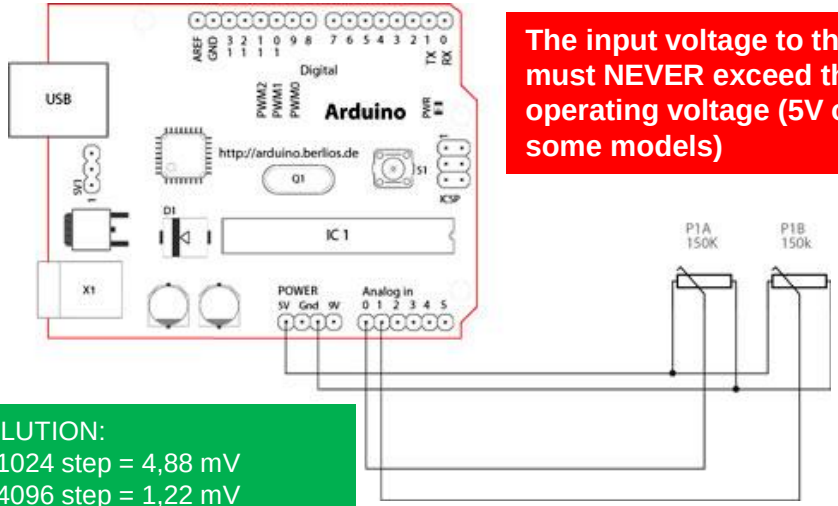
Arduino, in addition to communicating with the outside world through its ports, analog and digital (possibly connected to other devices such as displays, sensors, transducers, relays, ..), can also exchange data with the computer through its serial port (via USB). This function is often used when debugging the sketch, to display variable values or processing status.



https://create.arduino.cc/projecthub/wesee/project-kool-temperature-and-humidity-remote-monitoring-e5ddae

Analog inputs of Arduino

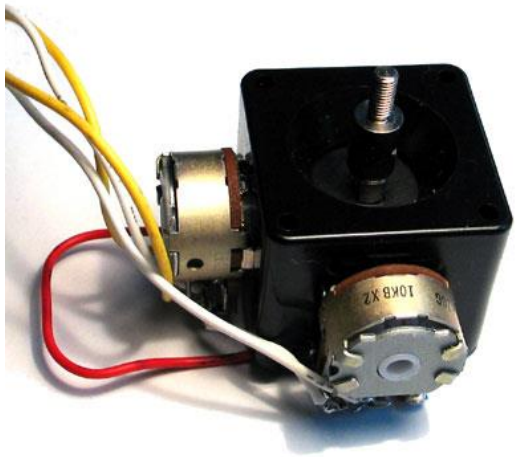
Arduino has some analog input pins, connected to ADC converters (inside the microcontroller). In the case of UNO and MEGA, they have 10 bit resolution, so they are capable of discretize an input voltage in 1024 intervals. If a higher resolution is required, an external ADC can be used, connected to with Arduino with a bus I^2C or SPI.



The input voltage to the ADC must NEVER exceed the card's operating voltage (5V or 3.3V for some models)

ADC RESOLUTION:
10 bit: 5V / 1024 step = 4,88 mV
12 bit: 5V / 4096 step = 1,22 mV
16 bit: 5V / 65536 step = 0,07 mV
24 bit: 5V / 16777216 step ≈ 0,0003 mV

Interfacing a Joystick



Arduino DUE, thanks to a most powerful microcontroller, has 2 x 12 bit ADCs and 2 x 12 bit DACs on the board

- Sampling Theorem
- Nyquist frequency
- Antialiasing filters

https://www.arduino.cc/en/Tutorial/JoyStick

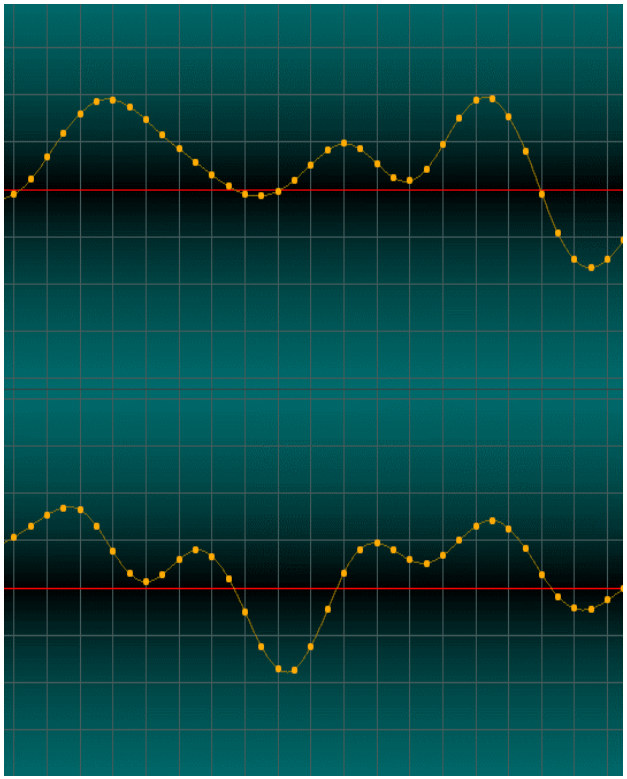
Arduino digital inputs and outputs

All Arduino models have some digital and analog inputs and outputs. The number and type of these inputs depends on the Arduino model. For example:

Arduino UNO
- 14 digital ports (6 PWM), configurable as IN or OUT
- 6 A/D converters with 10 bit resolution (1024 values)

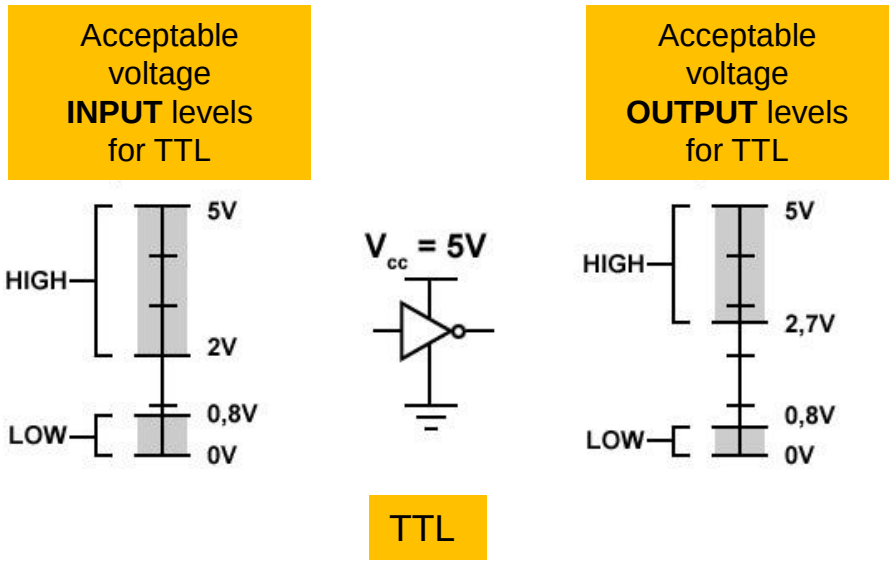
Arduino Mega2560
- 54 digital ports (15 PWM), configurable as IN or OUT
- 16 A/D converters with 10 bit resolution (1024 values)

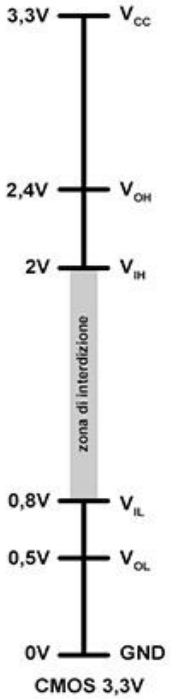
Arduino DUE
- 56 digital ports (12 PWM), configurable as IN or OUT
- 12 A/D converters with 12 bit resolution (4096 values)
- 2 D/A converters with 12 bit resolution (4096 values)

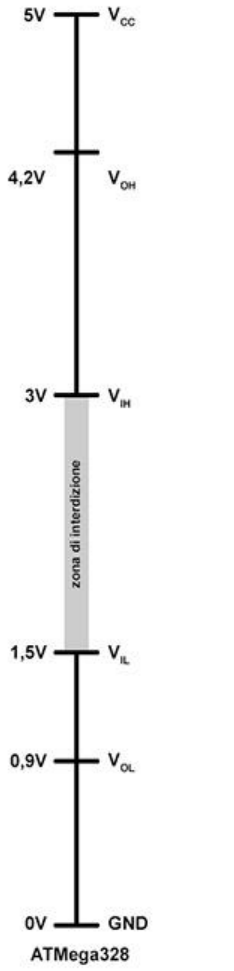


- Sampling Theorem
- Nyquist frequency
- Antialiasing filters

Electrically, the digital input and output levels correspond to values of voltage of 5 V (or 3.3 V for some Arduino models) for the HIGH level and 0 V for LOW level. There are actually tolerance ranges above and below to which the logical values HIGH and LOW are recognized:

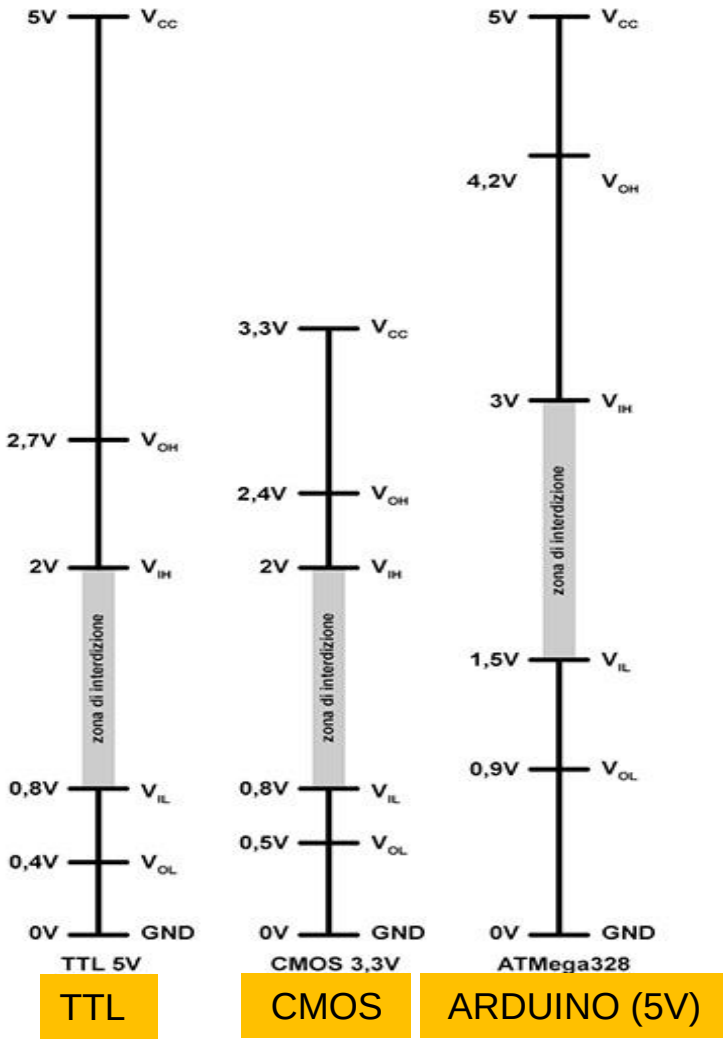






http://www.maffucci.it/2014/11/05/livelli-logici-ttl-e-cmos-cosa-si-nasconde-dietro-un-high-o-low-di-una-digitalwrite-di-arduino/

MEASUREMENTS WITH ARDUINO

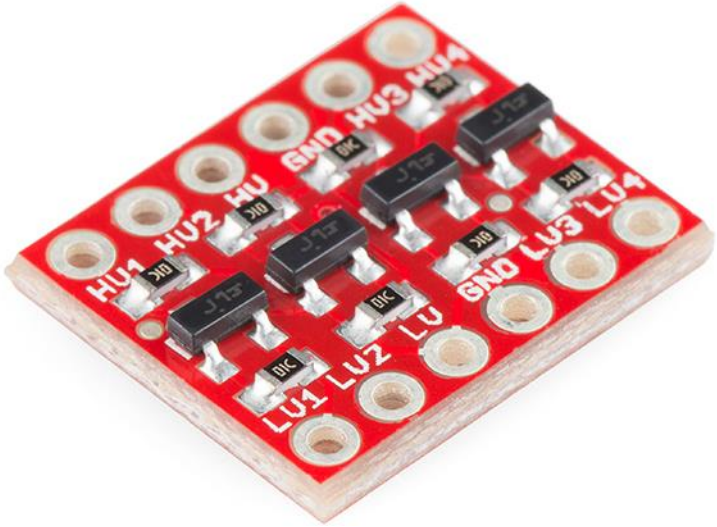


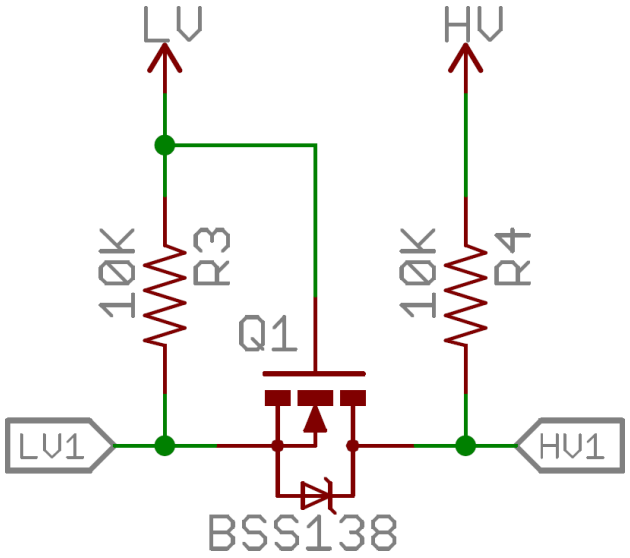
http://www.maffucci.it/2014/11/05/livelli-logici-ttl-e-cmos-cosa-si-nasconde-dietro-un-high-o-low-di-una-digitalwrite-di-arduino/

Conversion of logical levels

If Arduino is connected to devices that do not use the same logical levels, a *level shifting* must be done, using special components or *shields* created especially for this task. The conversion from 5 V to 3.3 V could also be done with a voltage divider formed by 2 resistors, but clearly this conversion goes only in a single direction. Note: the logic levels of communication, i.e. the digital lines, do not necessarily correspond to the supply voltages of the device.

There are components and shields that allow **bidirectional** level conversion (BD-LLC):



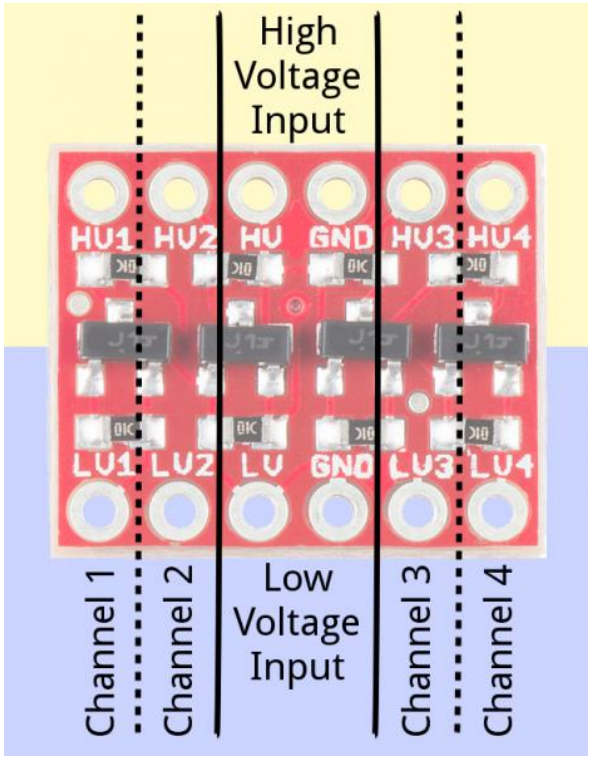


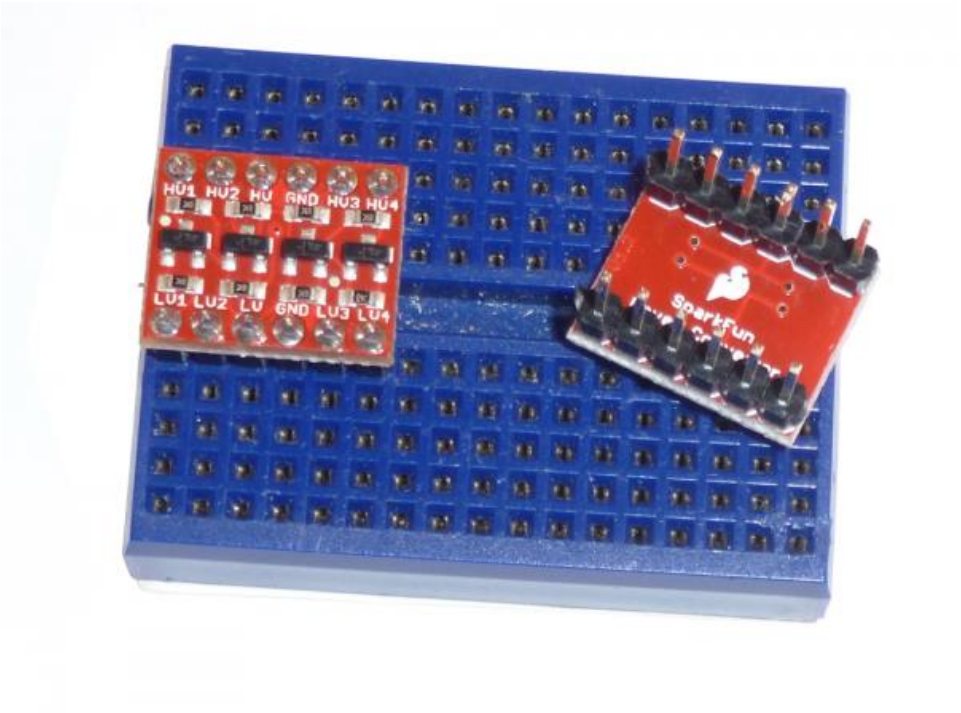
https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

MEASUREMENTS WITH ARDUINO

This shield board (and similar ones) convert the levels from 5 V logic to 3.3 V logic and vice versa.

The high voltage (5 V), low voltage (3.3 V) and ground must be connected to the board. The incoming 5 V logic levels from HVn ports are converted to 3.3 V logic levels at LVn ports, and the incoming 3.3 V logic levels from LVn ports are converted to 5 V logic levels at HVn ports. In this example there are 4 ports available.

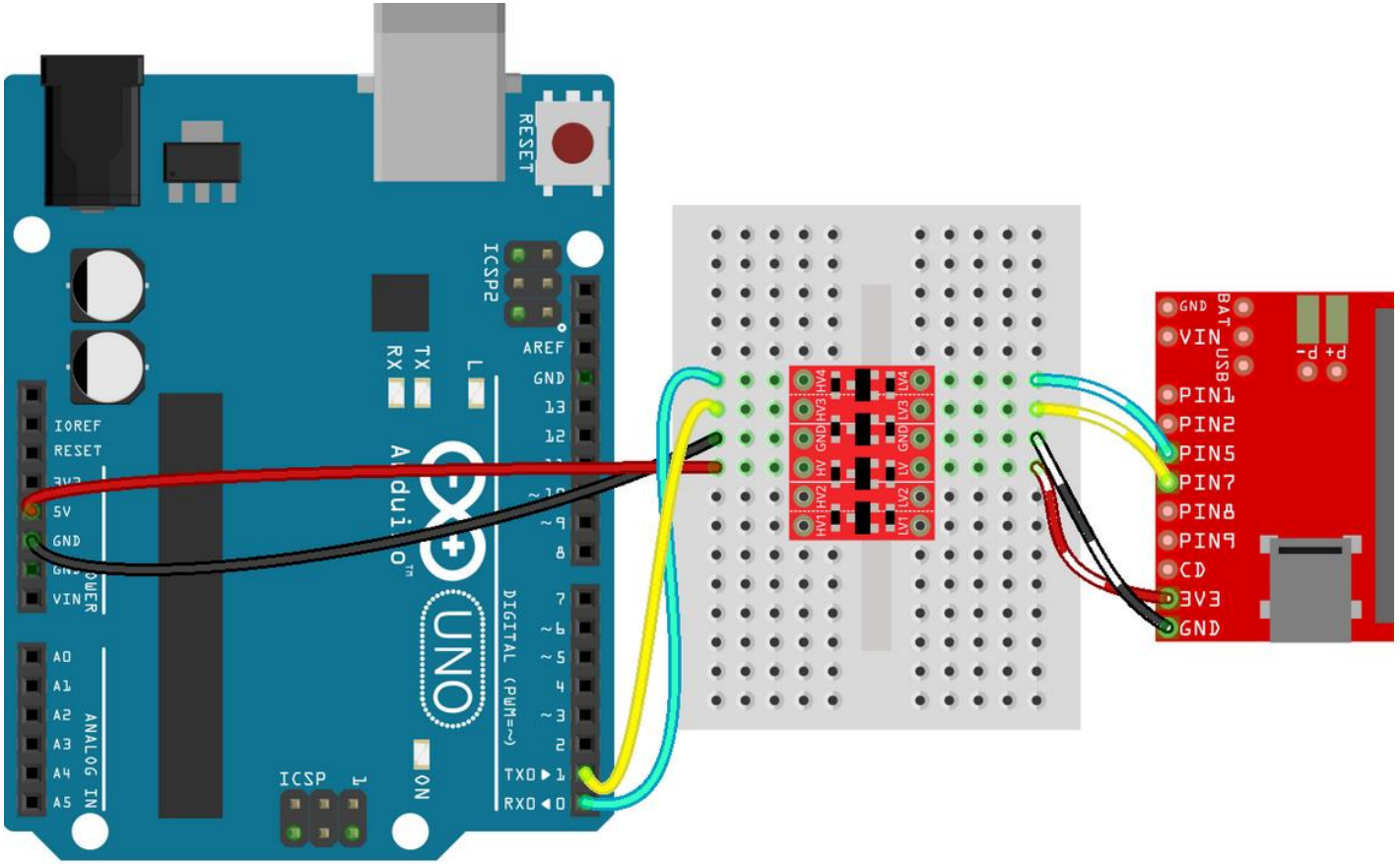




https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

MEASUREMENTS WITH ARDUINO

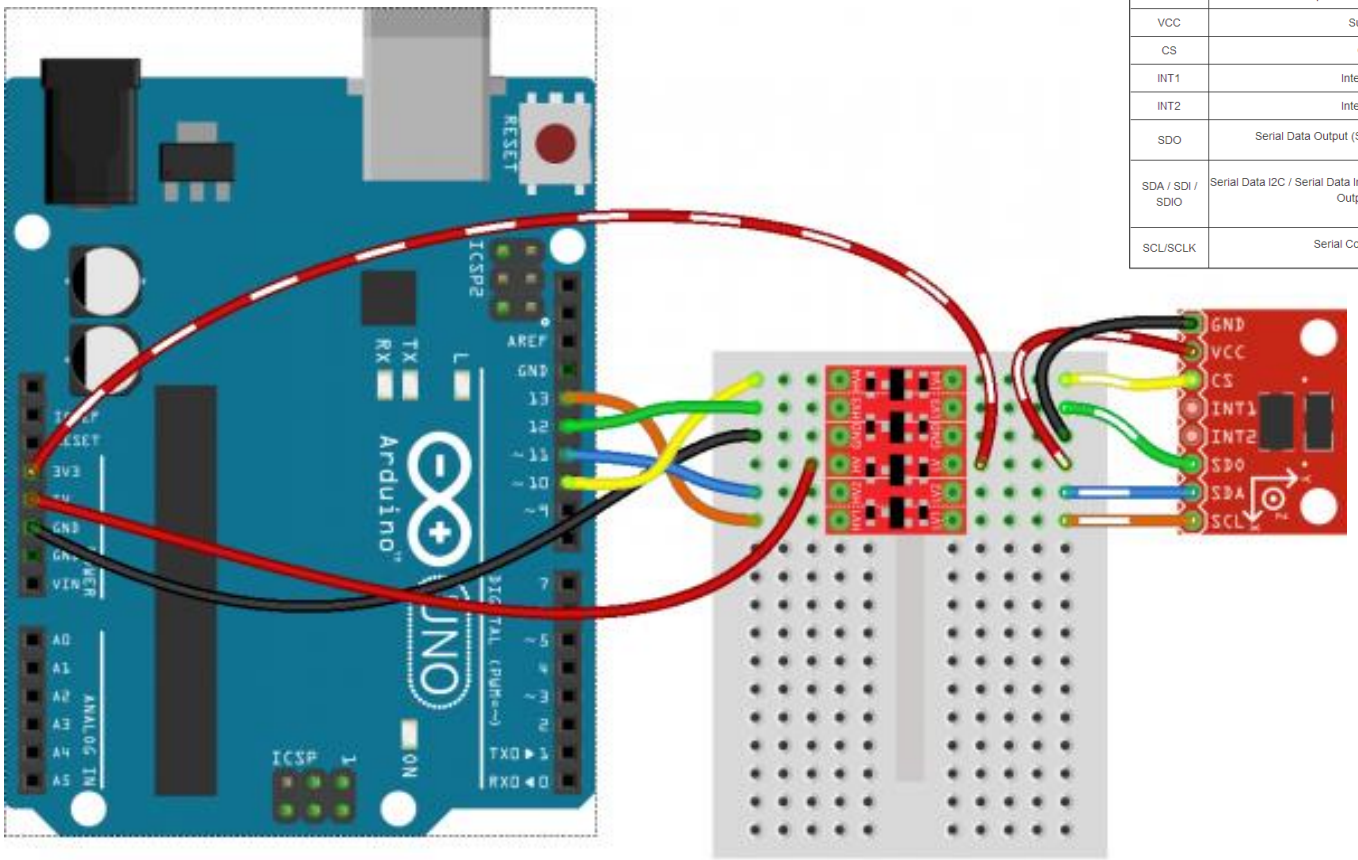
Example of connection between Arduino UNO (5 V logic) and a 3.3 V logic device. Communication from serial ports (TX0 and RX0). Note that the Arduino also has a pin in which 3.3 V are directly available (to power any device that requires this voltage).



https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

Example of connection between Arduino UNO (5 V logic) and a 3.3 V logic device. Communication via SPI protocol. 4 wires are used: MOSI (master out, slave in), MISO (master in, slave out), SCLK (serial clock) and CS (chip select).

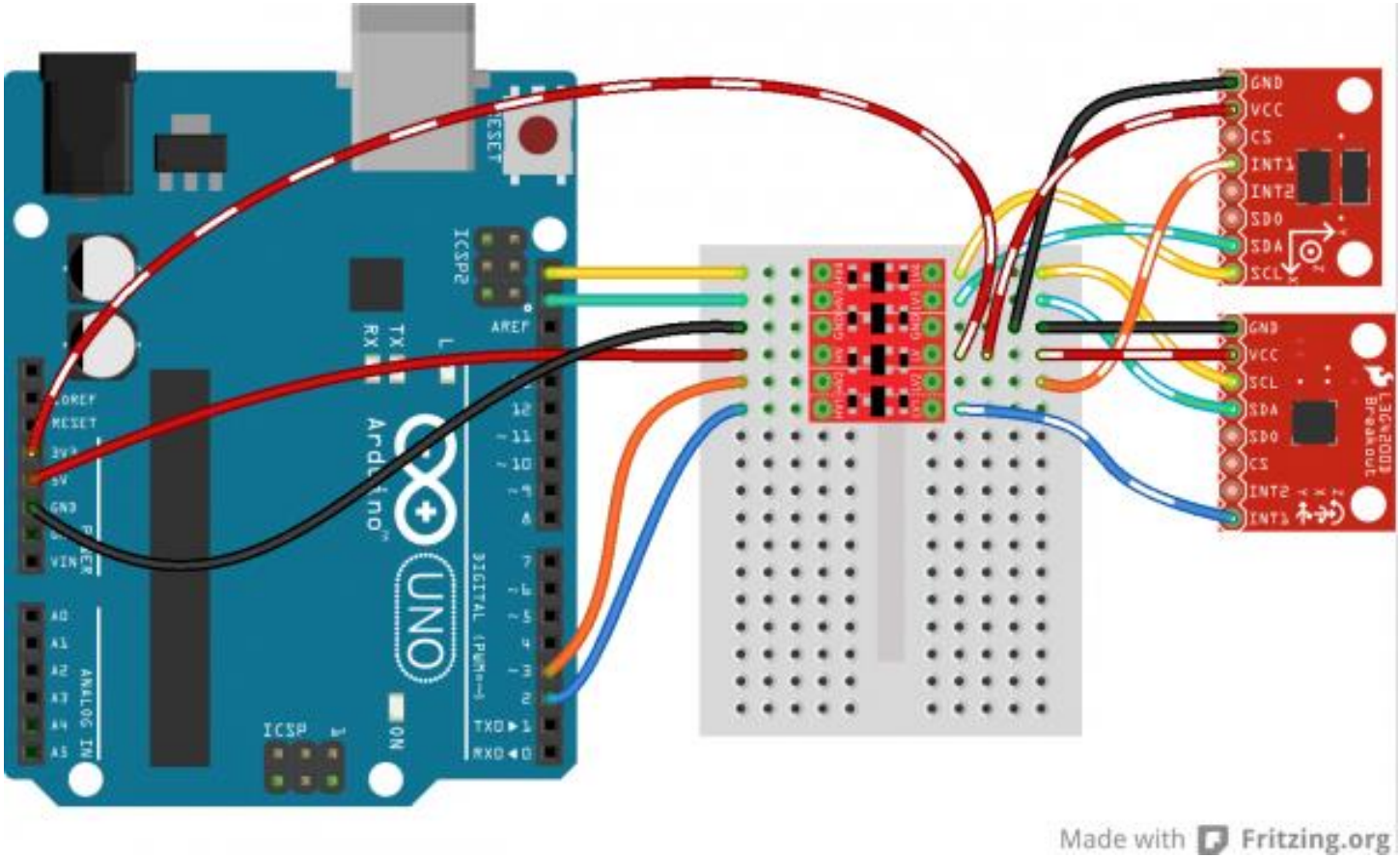
Mnemonic	Description
GND	This pin must be connected to ground
VCC	Supply Voltage
CS	Chip Select
INT1	Interrupt 1 Output
INT2	Interrupt 2 Output
SDO	Serial Data Output (SPI 4-Wire) / I2C Address Select
SDA / SDI / SDIO	Serial Data I2C / Serial Data Input (SPI 4-WIRE) / Serial Data Input and Output (SPI 3-Wire)
SCL/SCLK	Serial Communications Clock



https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

MEASUREMENTS WITH ARDUINO

Example of connection between Arduino UNO (5 V logic) and a 3.3 V logic device. Communication via I^2C protocol. In this case the data passing through the 2 wires required by the protocol (SDA and SCL) are bidirectional, supported by the conversion module.

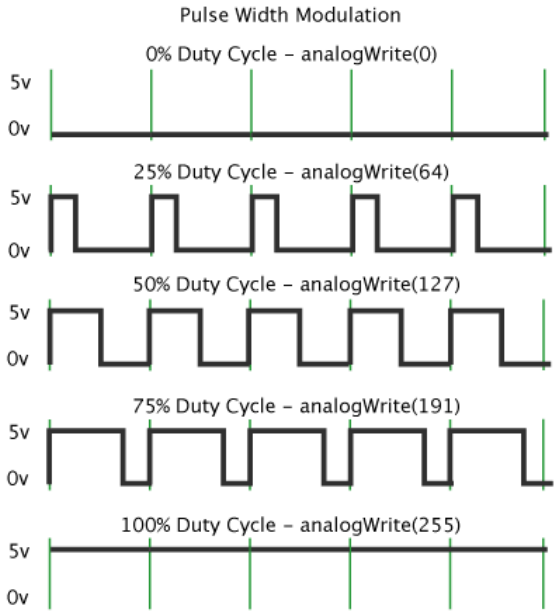


https://learn.sparkfun.com/tutorials/bi-directional-logic-level-converter-hookup-guide

PWM

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.



Sampling Theorem, Nyquist frequency, Antialiasing filters

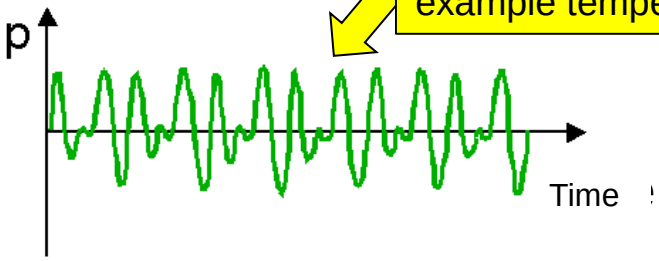
FREQUENCY ANALYSIS

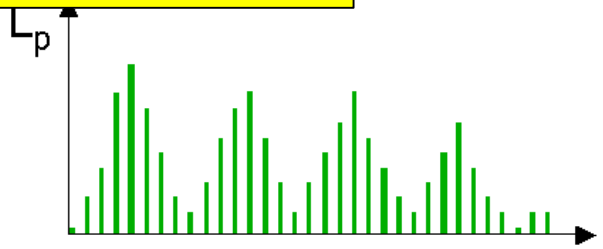
Periodic

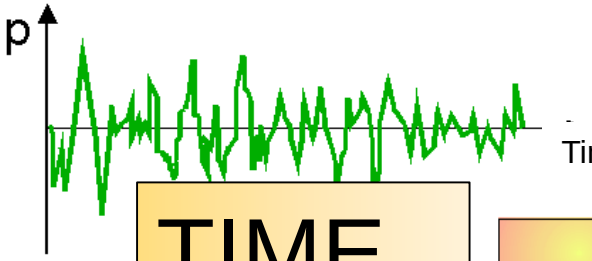
Random

Impulsive

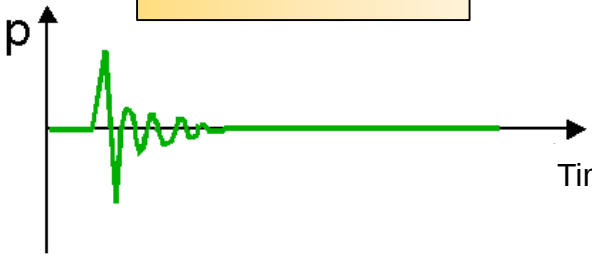
Signal that changes over time, for example temperature measurement

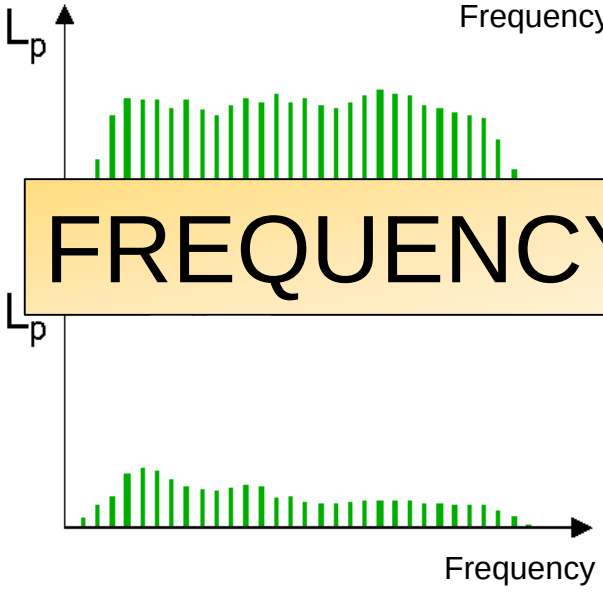






TIME





FREQUENCY

Image Courtesy of Brüel & Kjær

Sampling Theorem, Nyquist frequency, Antialiasing filters

Through a mathematical operation, the Fourier Transform, it is possible to analyze a signal, for example a sound or the time-varying temperature measurement from a sensor, by analyzing the spectral content (frequency domain) of the signal (time domain).The Fourier theory states that a time-varying signal can be represented by the sum of infinite sinusoids, with frequencies varying from zero to infinity, each with a given amplitude and phase. The amplitudes and phases of these sinusoids determine the waveform and spectrum of the signal. The spectral components of a given signal, each associated with a specific frequency, can be calculated, obtaining complex numbers (modulus and phase or real and imaginary part). In practice, not being possible to analyze all frequencies from zero to infinity, the analysis is limited to a restricted portion of frequencies. Fast calculation algorithms for Fourier transform exist: **FFT** = Fast Fourier Transform.


Baron Fourier.

Time domain ⇕ Frequency domain

$$X(f) = F[x(t)] = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

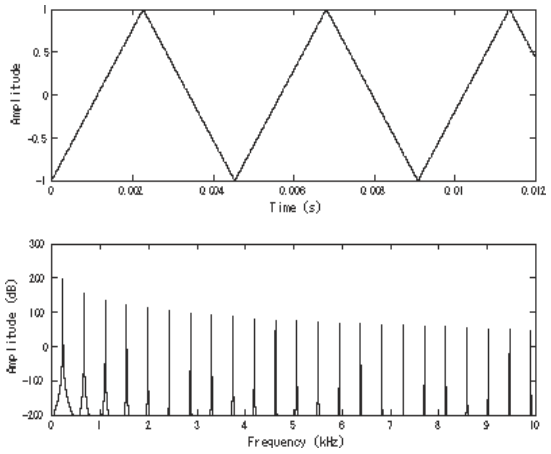
(Direct) Fourier Transform
Time ⇨ Frequency

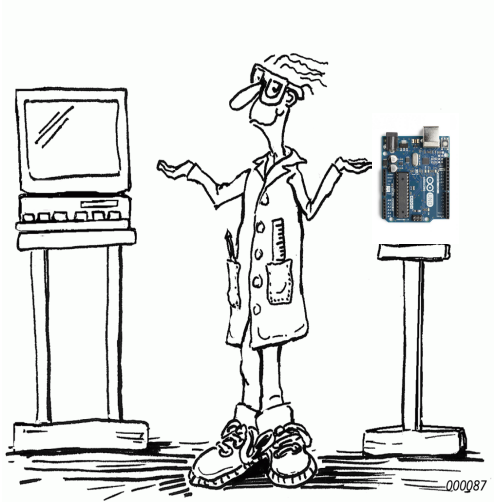
$$e^{j\omega t} = \cos(\omega t) + j\,\mathrm{sen}(\omega t)$$

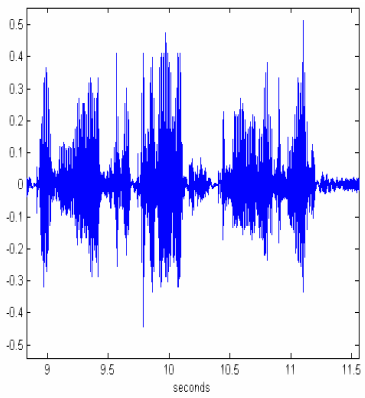
$$x(t) = F^{-1}[X(f)] = \int_{-\infty}^{\infty} X(f) e^{+j2\pi ft} df$$

Inverse Fourier Transform
Frequency ⇨ Time

Similar considerations can be made on sampled signals, discrete in time and amplitude (**DFT**, Discrete Fourier Transform). The integrals of the previous formulas become summations and the continuous signals become discretized. This means that with digital devices a signal (temporal sequence of data) can be sampled and analysed in the frequency domain (spectral analysis). The sampled signal can be represented as a sum of sinusoids (harmonics), each with a frequency multiple of a frequency called fundamental. Each harmonic has a certain amplitude and phase characteristic.







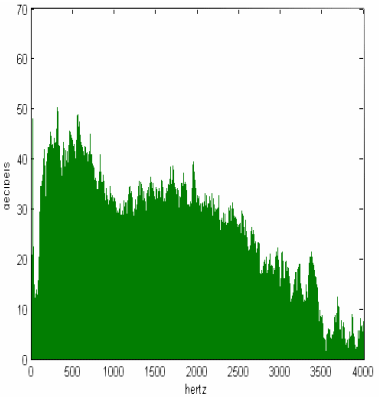
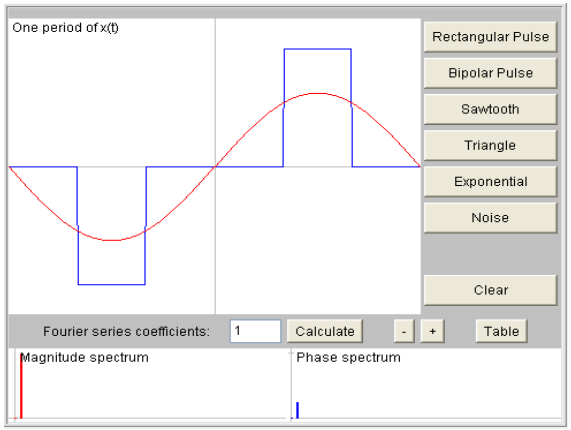
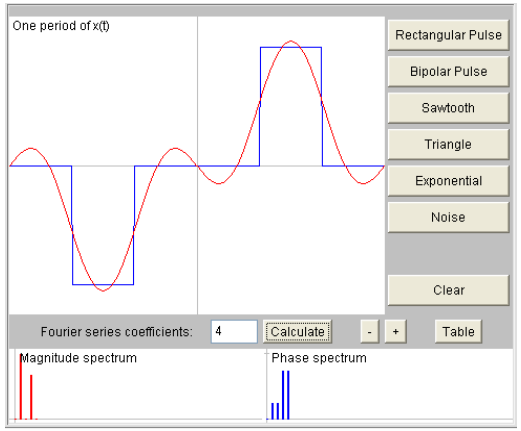
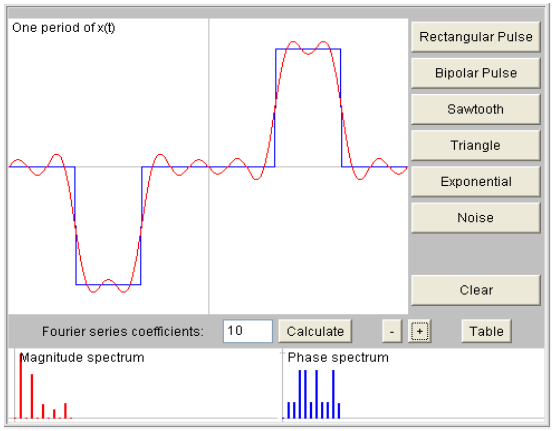


Image Courtesy of Brüel & Kjær

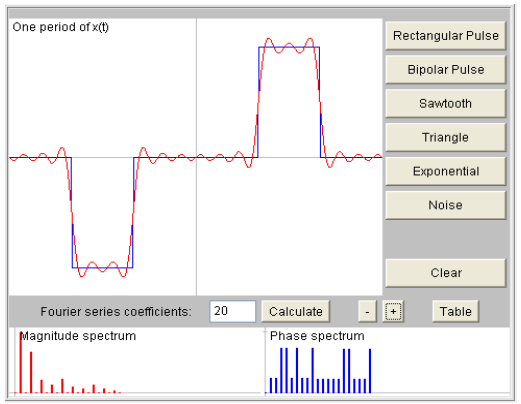
Approximation of a signal via Fourier Series







$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi f_k t}$$



http://www.jhu.edu/~signals/fourier2/index.html

Signal and its frequency spectrum

Example: sound wave

c = speed of sound in air

$$\lambda = \frac{c}{f}$$

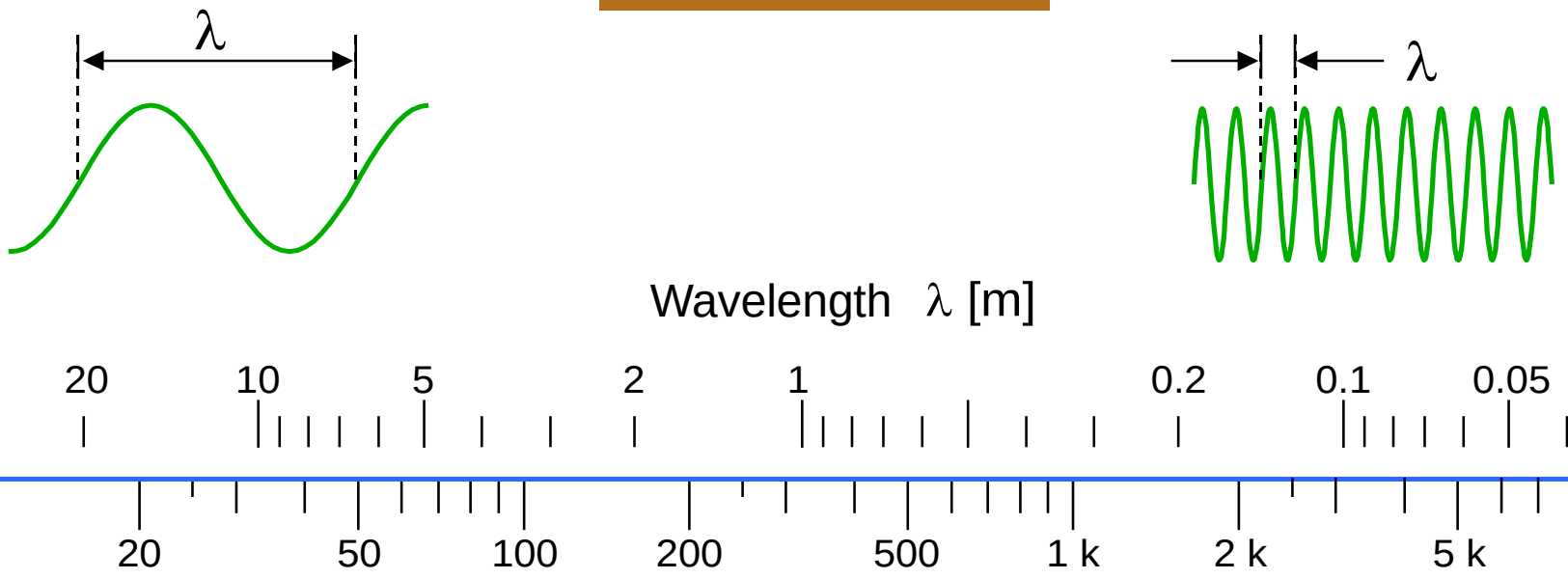
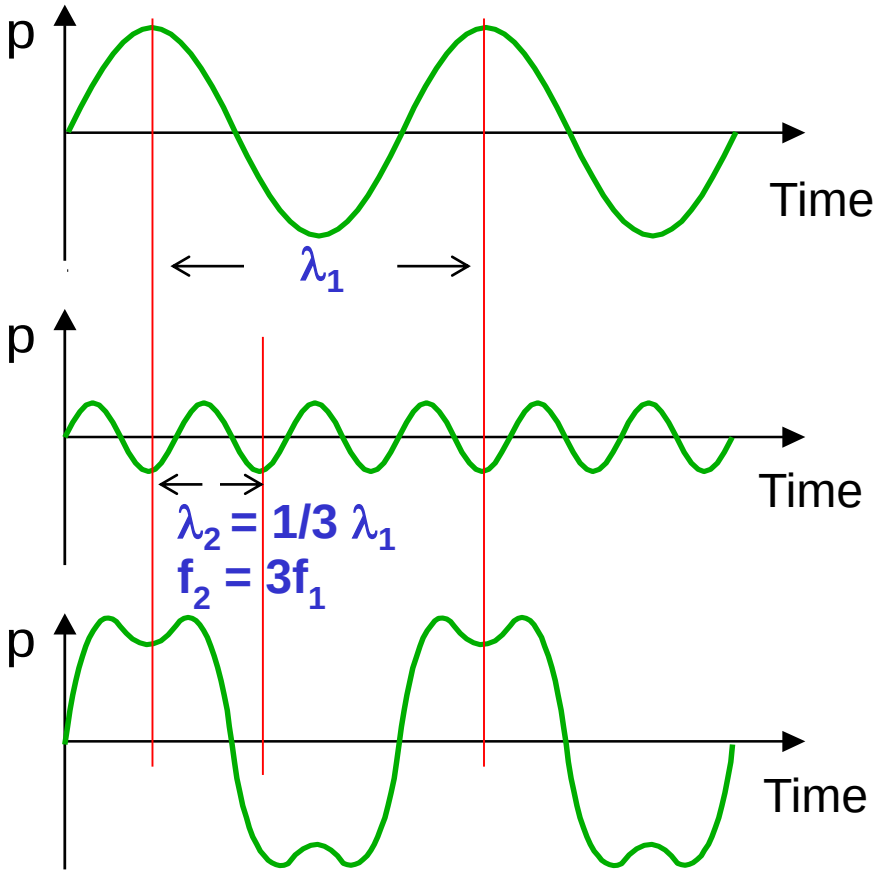


Image Courtesy of Brüel & Kjær

Signal and its frequency spectrum



Example: sound wave

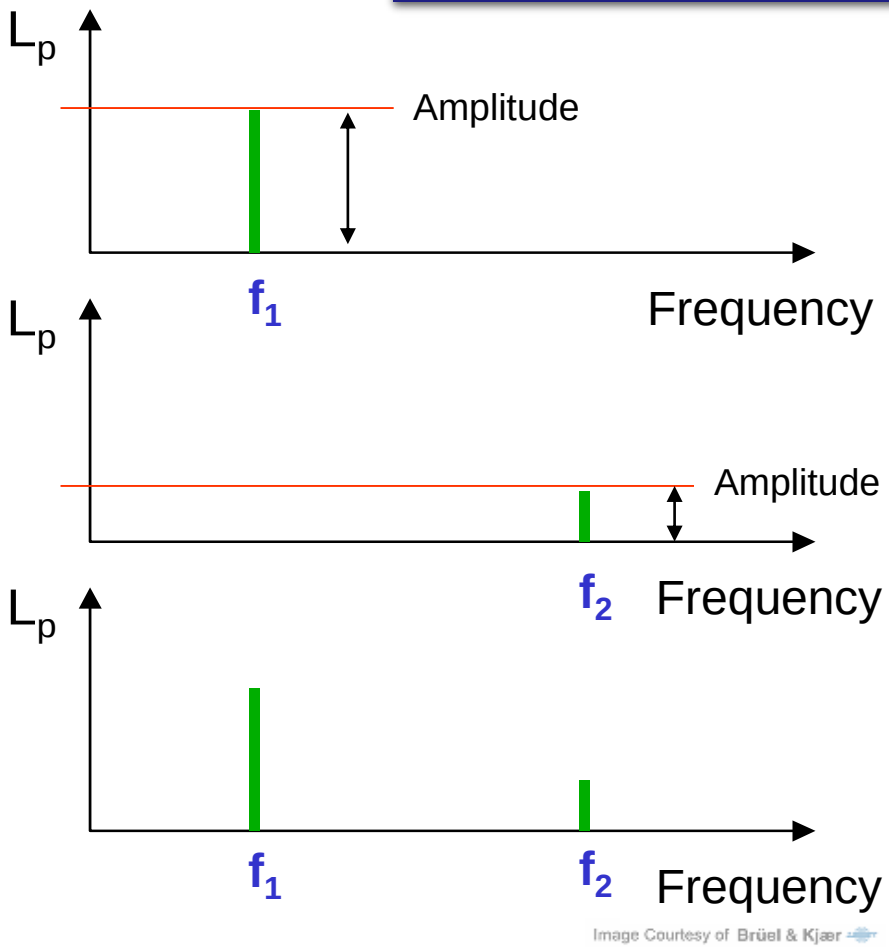
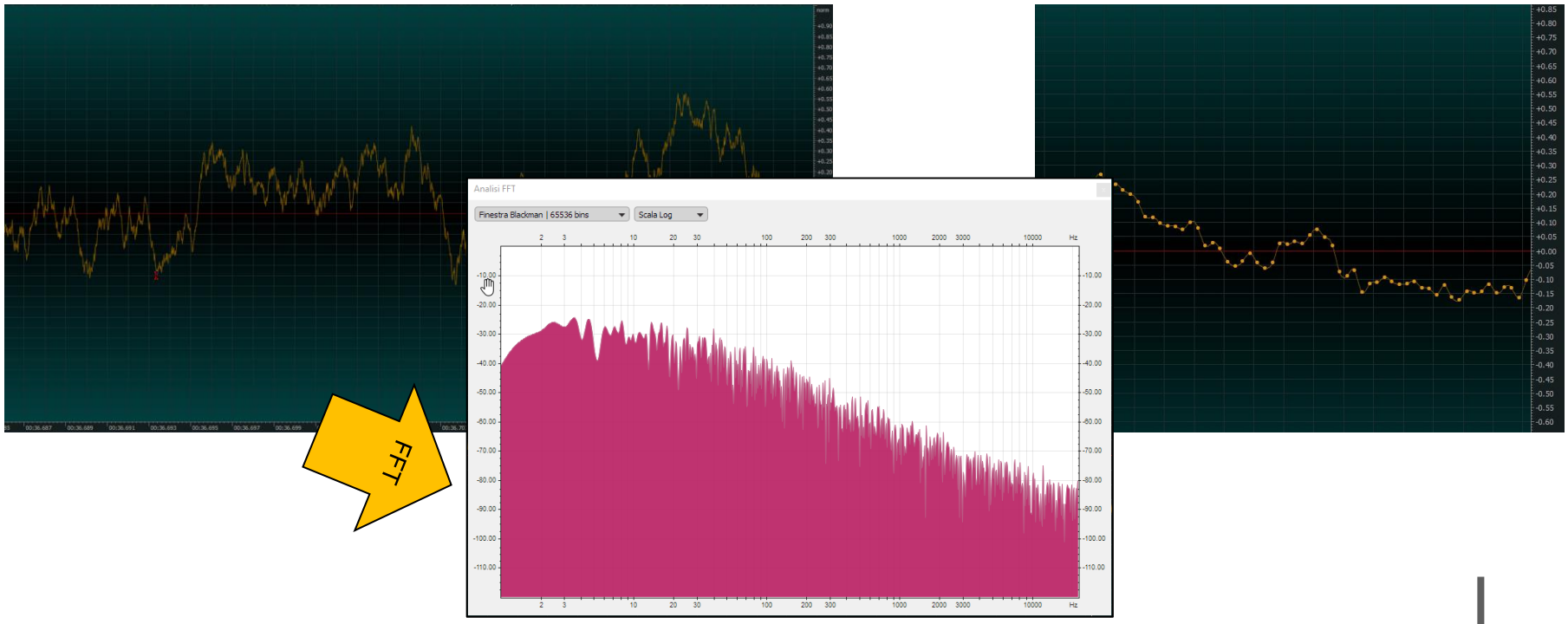


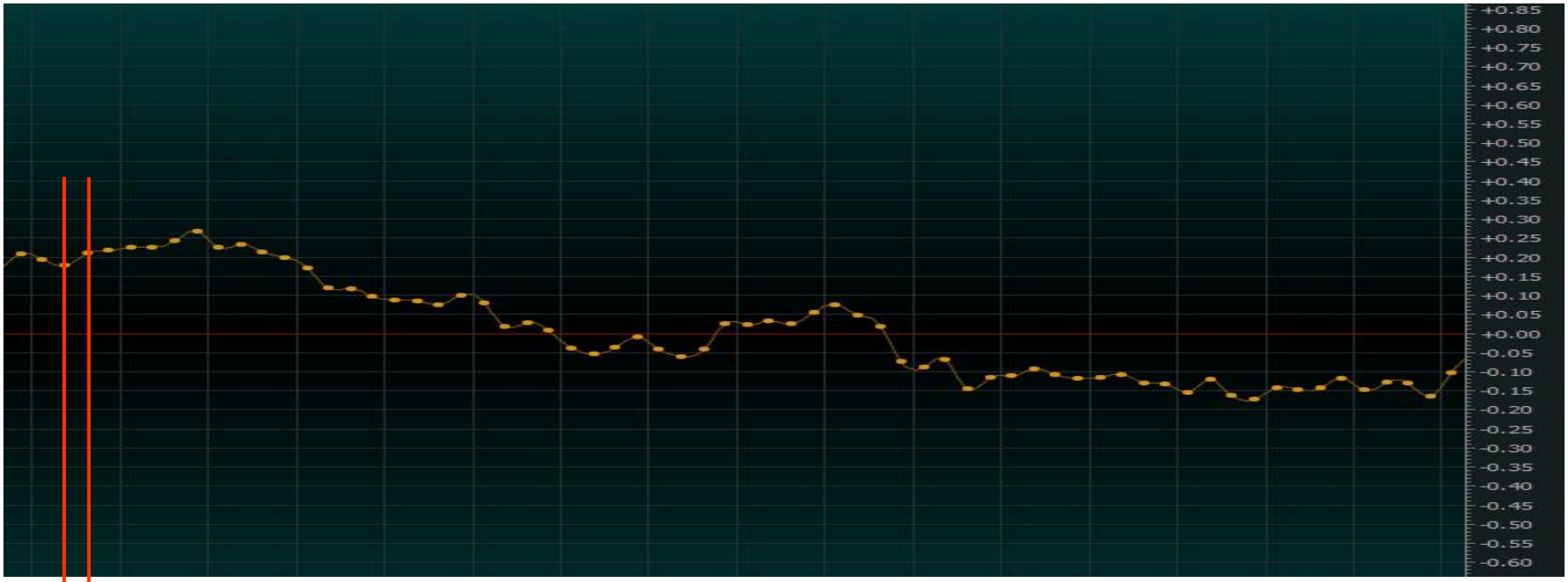
Image Courtesy of Brüel & Kjær

Time domain and Frequency domain

A variable electrical signal (voltage) at the input of an ADC (Analog to Digital Converter), coming for example from a sensor that measures a thermodynamic variable such as temperature or pressure, analyzed for a certain interval of time, has a spectrum that can have a more or less extended frequency bandwidth, which essentially depends on how fast this signal changes over time and on the type of variation.



Sampling frequency and Nyquist frequency



T = Sampling interval (s)

$f_s = \frac{1}{T}$ Sampling frequency (Hz)

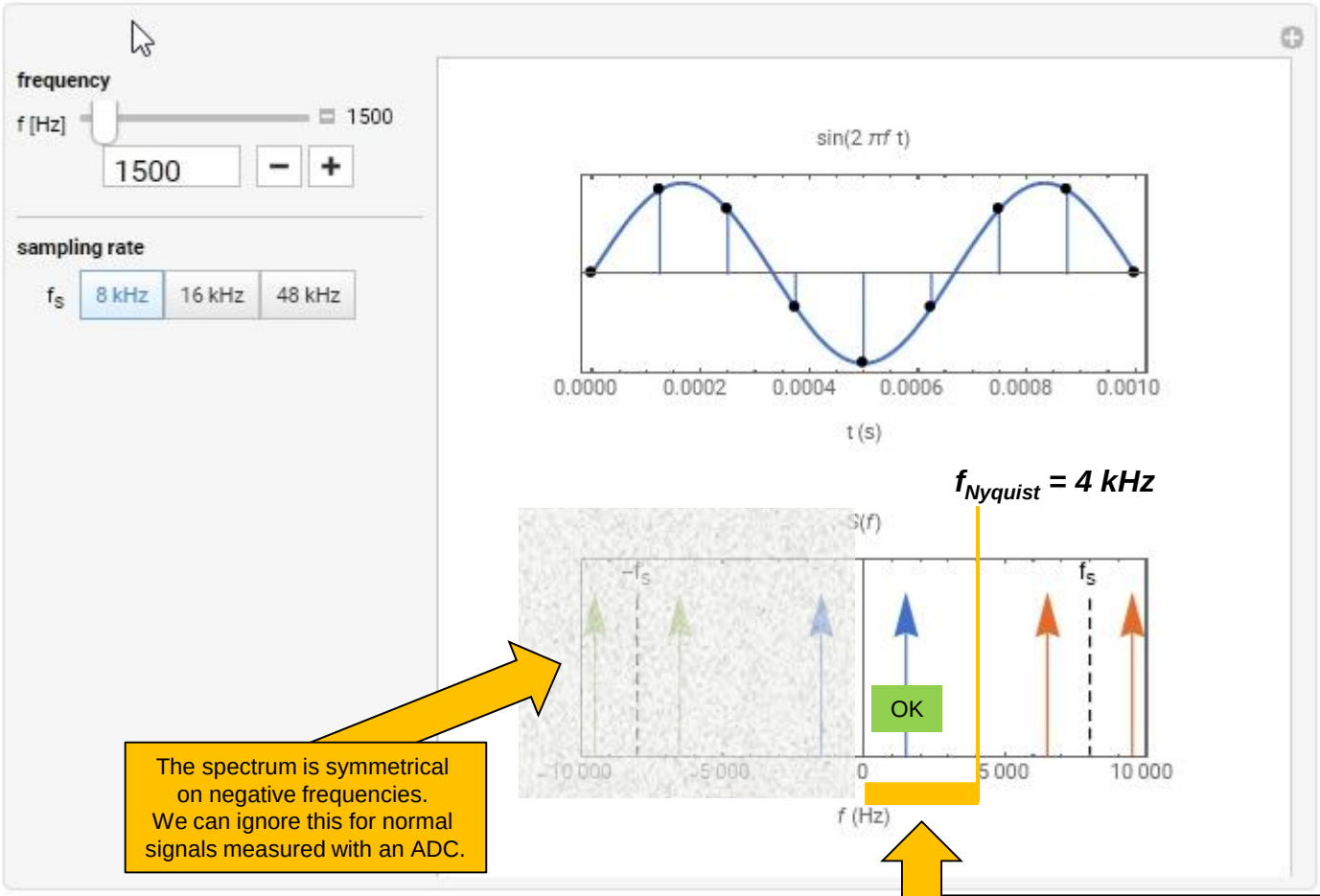
$f_N = \frac{f_s}{2}$ Nyquist frequency (Hz)

The **Nyquist-Shannon Theorem** states that to sample a signal at the sampling frequency f_s the signal must have a spectrum with **maximum frequency f_{max} lower than the Nyquist frequency f_N** (half of the sampling frequency). Otherwise, the part of the signal spectrum higher than the Nyquist frequency will "fall" into the sampled spectrum, creating artifacts in the measured values, the so called **aliasing**.

$$f_{\max} < \frac{f_s}{2}$$

If a signal is sampled at a certain sampling frequency, the measurement will have a bandwidth up to the Nyquist frequency, not beyond this limit.

Nyquist–Shannon sampling theorem example



$f_{sampling} = 8$ kHz

$f_{Nyquist} = 4$ kHz

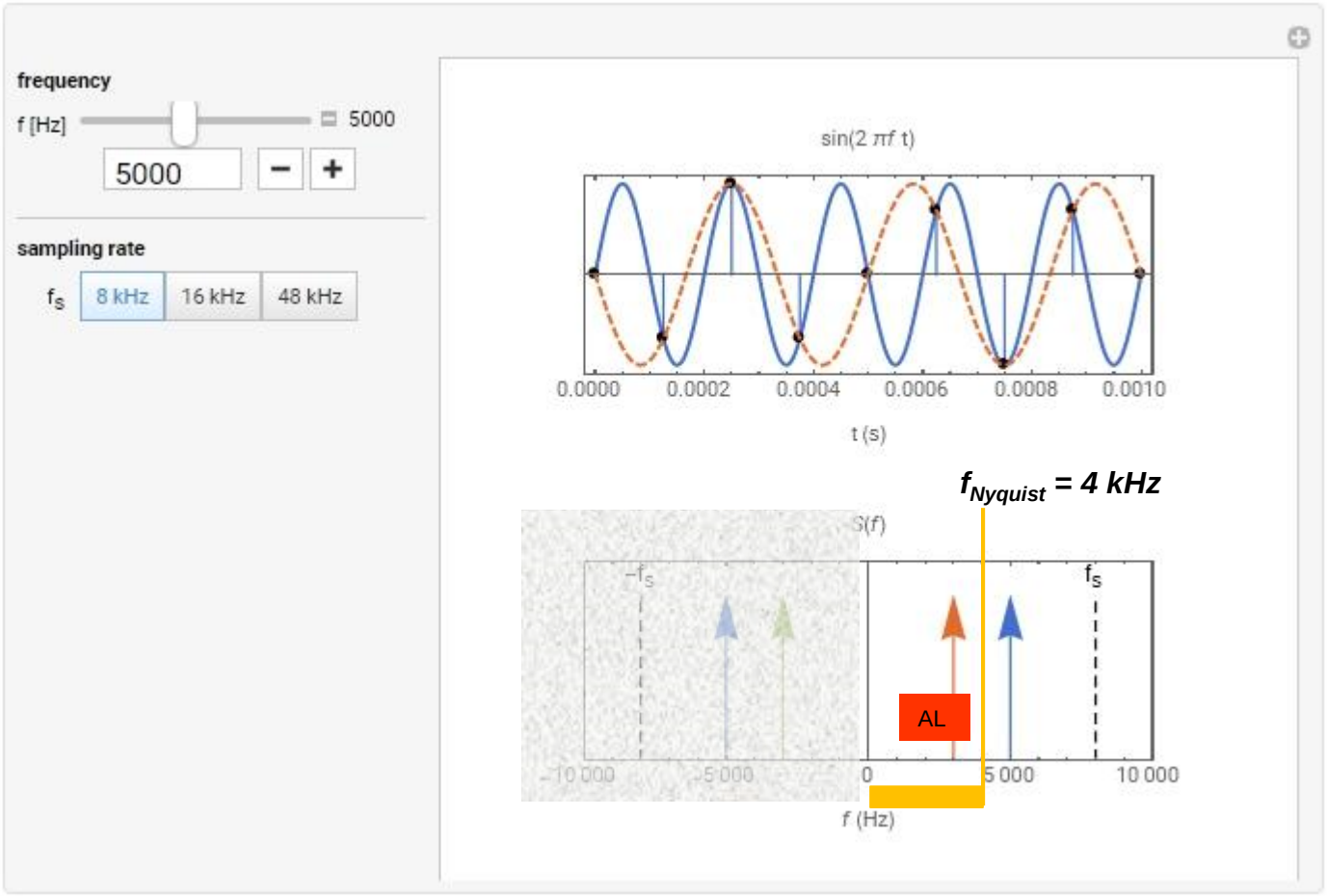
$f_{signal} = 1500$ Hz

$f_{signal} < f_{Nyquist}$

OK!

https://demonstrations.wolfram.com/SamplingTheorem/

Nyquist–Shannon sampling theorem example



$f_{sampling} = 8$ kHz
$f_{Nyquist} = 4$ kHz

$f_{signal} = 5000$ Hz

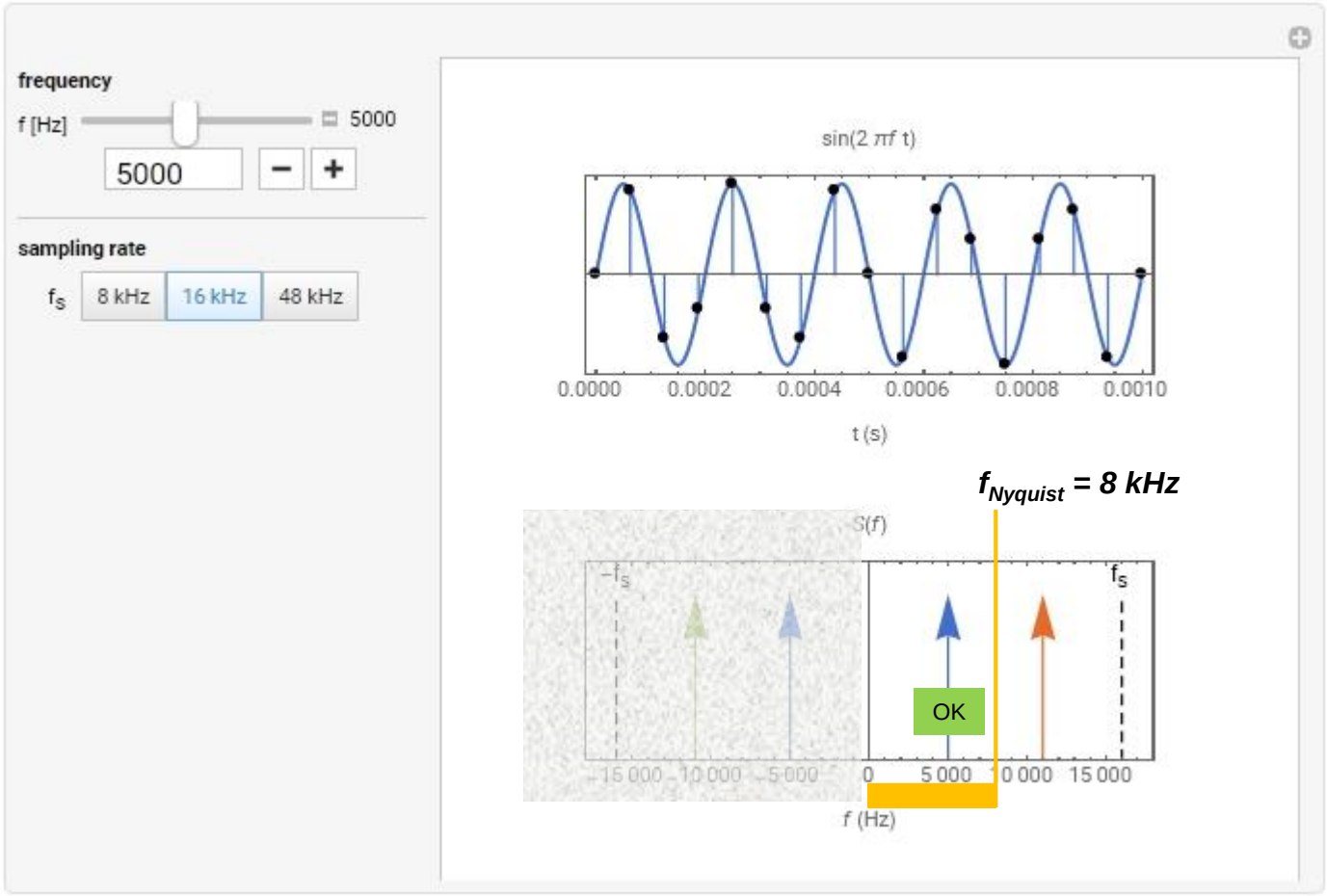
$f_{signal} > f_{Nyquist}$

ALIASING!

Apparent (wrong) frequency sampled: 3000 Hz

https://demonstrations.wolfram.com/SamplingTheorem/

Nyquist–Shannon sampling theorem example



$f_{sampling} = 16\ kHz$
$f_{Nyquist} = 8\ kHz$

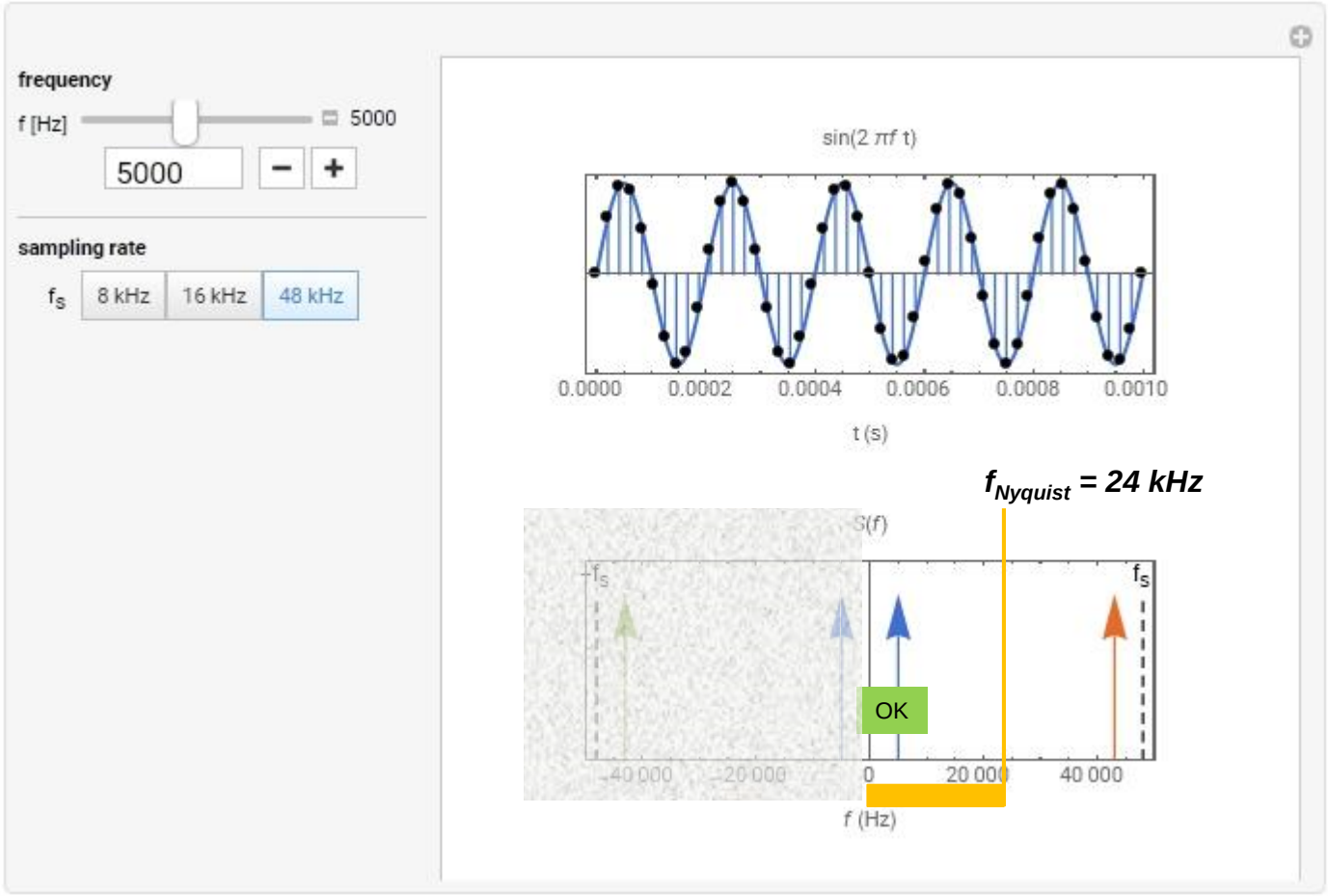
$f_{signal} = 5000\ Hz$

$f_{signal} < f_{Nyquist}$

OK!

https://demonstrations.wolfram.com/SamplingTheorem/

Nyquist–Shannon sampling theorem example



$f_{sampling} = 48$ kHz
$f_{Nyquist} = 24$ kHz

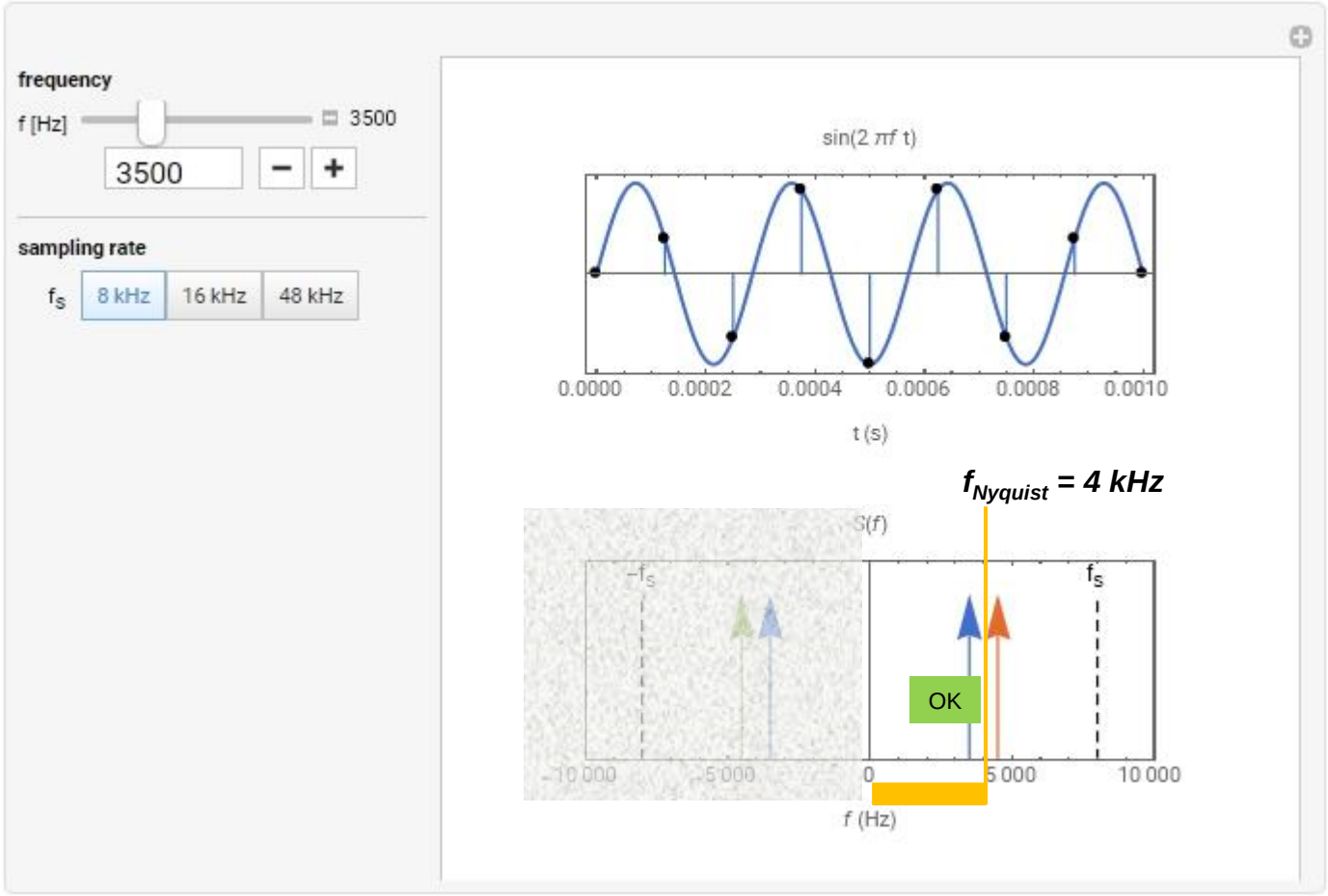
$f_{signal} = 5000$ Hz

$f_{signal} < f_{Nyquist}$

OK!

https://demonstrations.wolfram.com/SamplingTheorem/

Nyquist–Shannon sampling theorem example



$f_{sampling} = 8\ kHz$
$f_{Nyquist} = 4\ kHz$

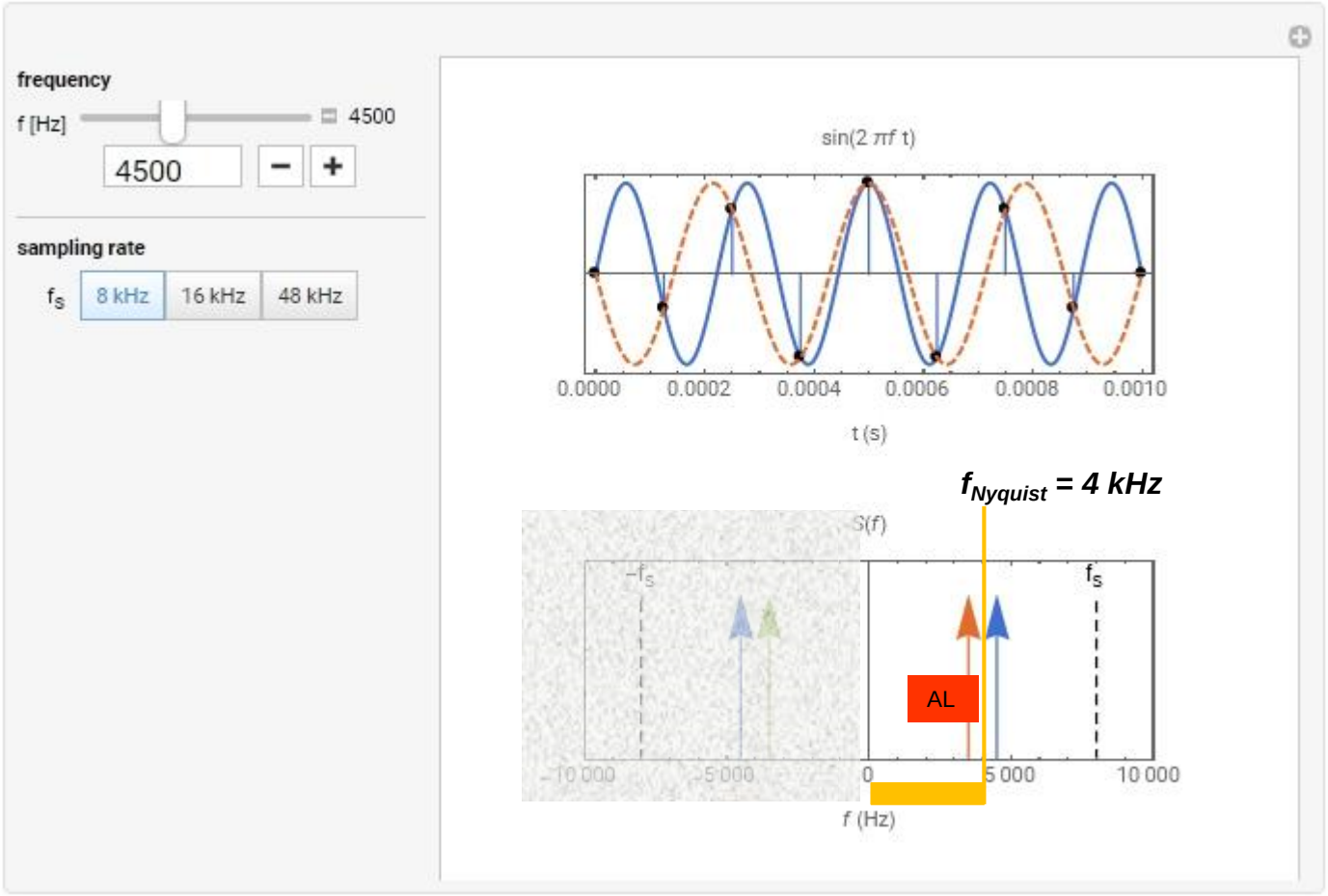
$f_{signal} = 3500\ Hz$

$f_{signal} < f_{Nyquist}$

OK!

https://demonstrations.wolfram.com/SamplingTheorem/

Nyquist–Shannon sampling theorem example



$f_{sampling} = 8$ kHz
$f_{Nyquist} = 4$ kHz

$f_{signal} = 4500$ Hz

$f_{signal} > f_{Nyquist}$

ALIASING!

Apparent (wrong) frequency sampled: 3500 Hz

https://demonstrations.wolfram.com/SamplingTheorem/

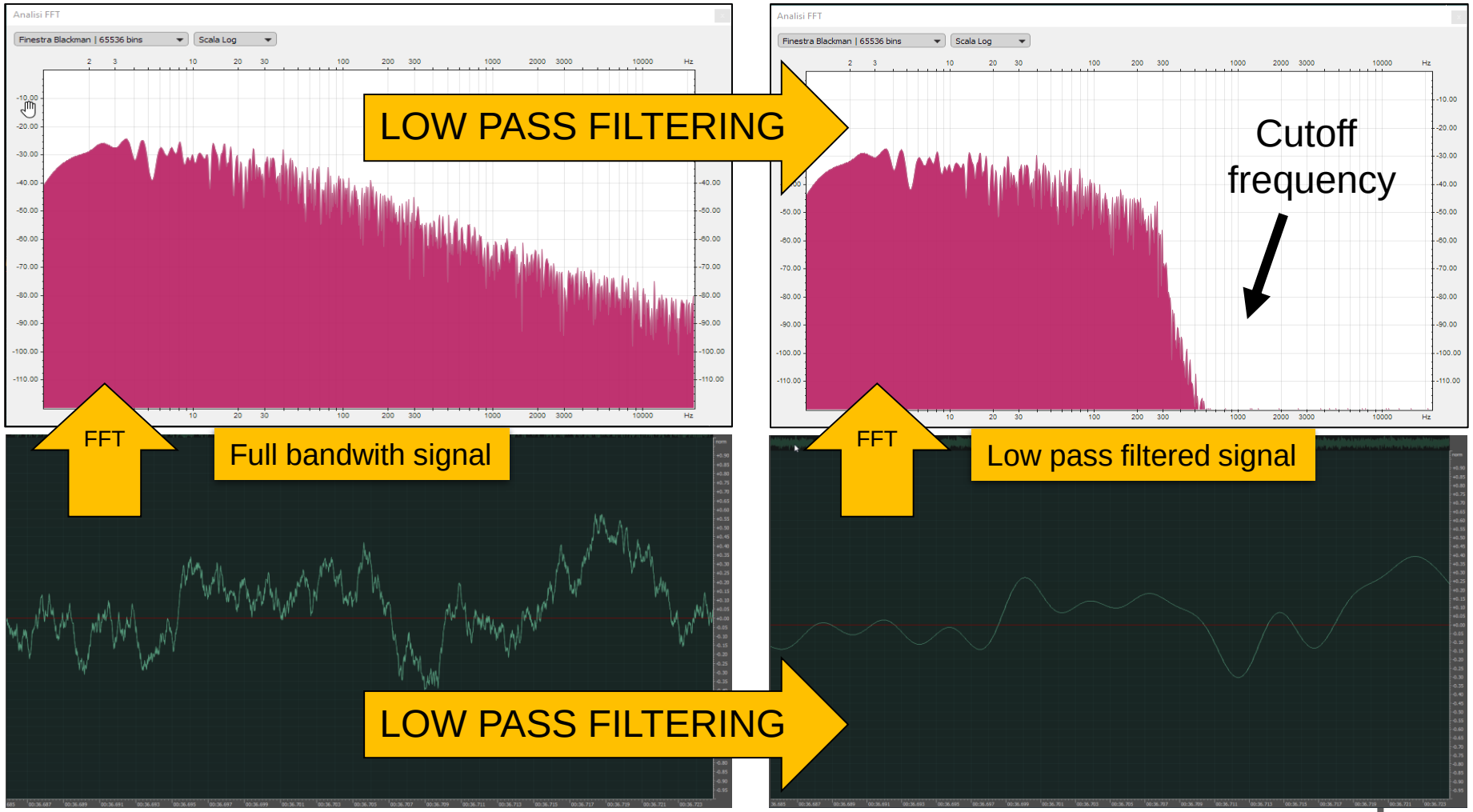
- Arduino is a “SLOW” device and, with normal programming, its ADCs can reach sampling frequencies around 9 kHz at most, so, in order to avoid aliasing, the sampled signals can have a bandwidth up to 4.5 kHz

- Other microcontrollers, such as Teensy, are much faster than the Arduino and can reach higher sample rates (48 kHz and above)

- If the data that Arduino is sampling varies very slowly or is stationary, there are no problems regarding sampling rate and aliasing

- If the signal in input to the ADC has a bandwidth greater than the Nyquist frequency, to avoid aliasing the input signal bandwidth must be reduced by means of a **low-pass filter**, with cutoff frequency equal or lower than the Nyquist frequency

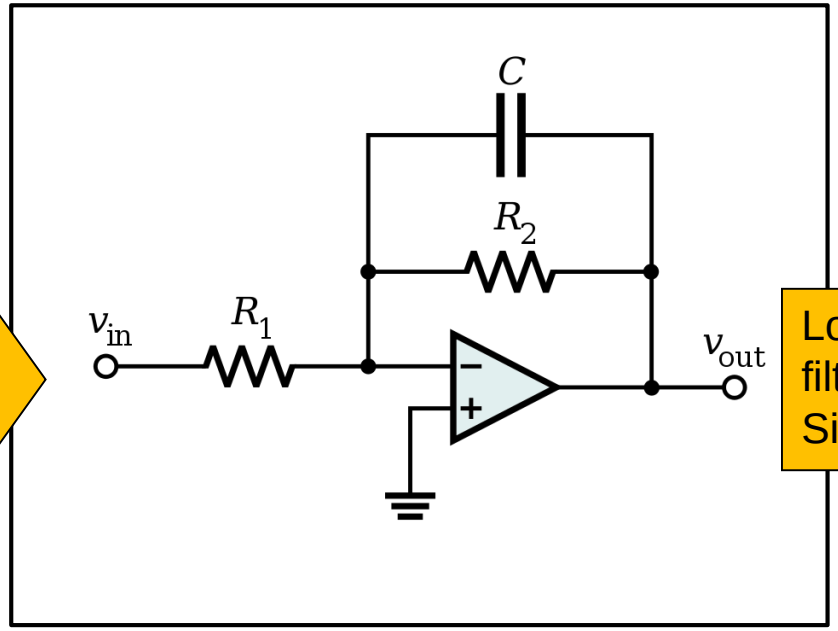
Low-pass filtering



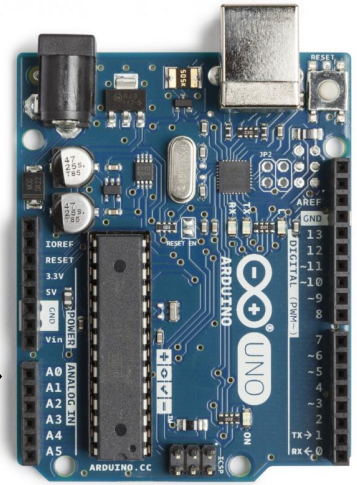
Low-pass filtering

Temperature sensor
(Thermocouple)

Signal



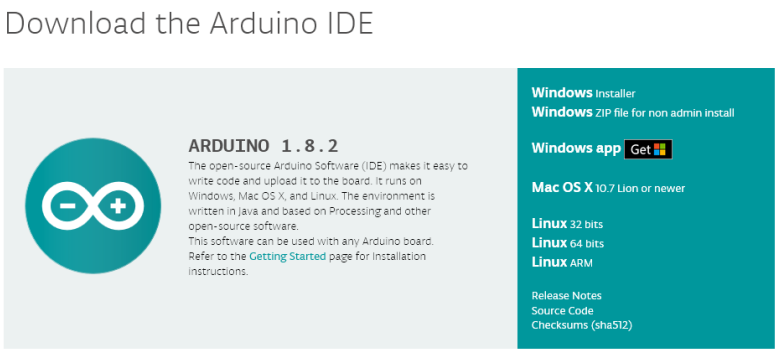
Low-pass filtered Signal



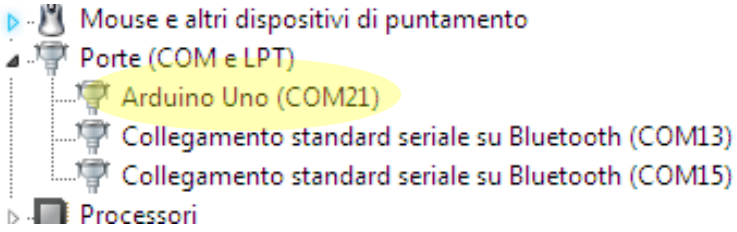
Example of low pass filter

The Integrated Development Environment (IDE) of Arduino

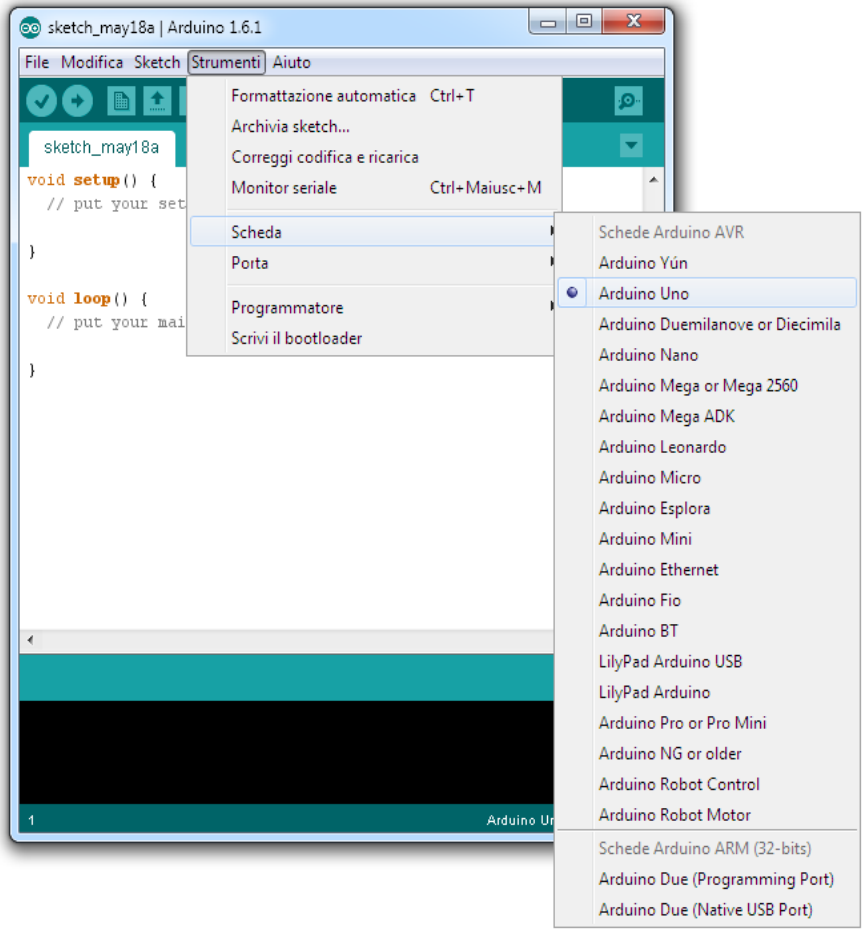
Arduino connects to the computer via USB. The IDE (Integrated development environment) is a simple text editor that allows to edit the user's source code, verify, compile and load it on the Arduino board. The IDE is cross-platform, it is available for Windows, OSX and Linux.

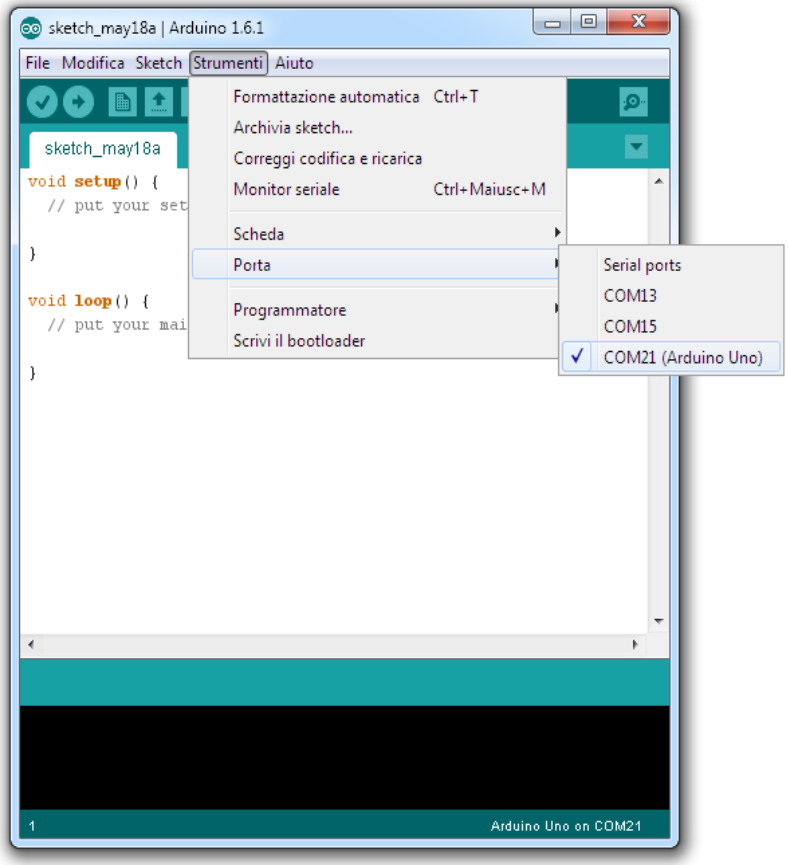


After installing the IDE, including its drivers, and connecting the Arduino board to the USB port of the- computer, it will be recognized as a serial port.



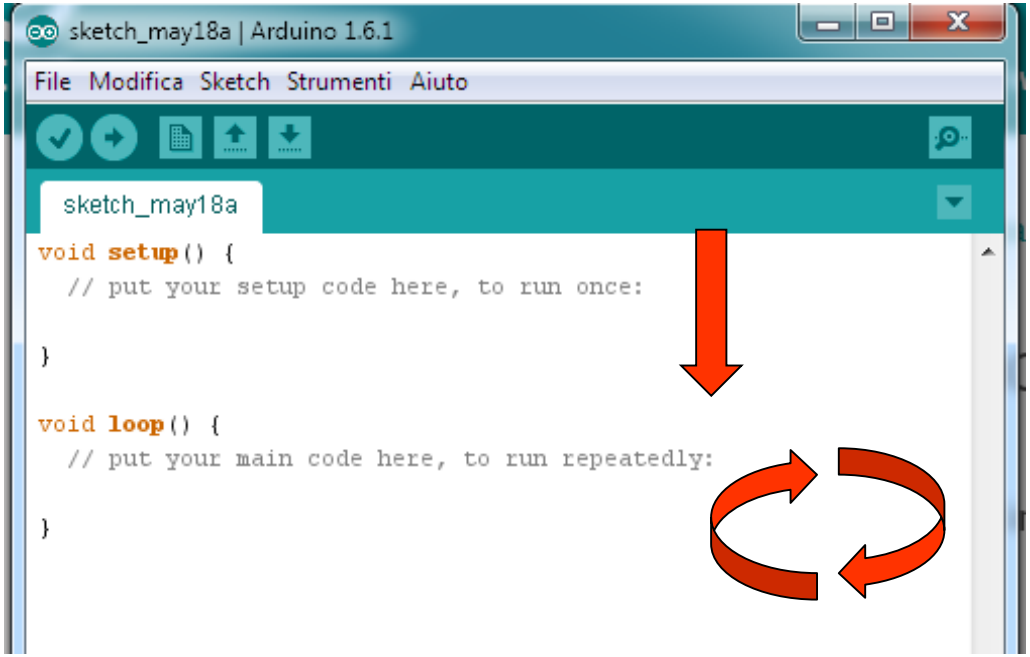
Now, in the IDE it is necessary to select the type of card in use and its (virtual) serial port. This operation must be performed only the first time the Arduino card is connected to the computer.





READY TO GO!!

The Arduino programming language



All user programs loadable on Arduino are formed by at least two parts (plus any others user-defined functions).

The "mandatory" parts are called:

setup() and **loop**()

- The **setup()** function executes the code enclosed in brackets { } only once, when starting or resetting the Arduino board. Here the initialization code of the program or of the peripherals connected to the board must be placed.
- The **loop()** function executes in infinite loop the part of the code enclosed in brackets { }. Here the main code of the program must be placed.
- Other functions, created by the user, can be optionally written and called, if required.

The “C” programming language syntax implemented in Arduino

The programming language reference guide is available on the official website
https://www.arduino.cc/en/Reference/HomePage

All Arduino programs include:

- **Structures**
- **Values** (memorized inside variables and constants)
- **Functions**

The basic structure of the program consists of the two functions **Setup** and **Loop**, but within these there may be other control structures such as *if... else* or *do... while* or *for*.

The C syntax requires that the curly brackets { } delimit the portion of code executed by the various control structures.

Control Structures

- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Further Syntax

- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)
- #define
- #include

Bitwise Operators

- & (bitwise and)
- | (bitwise or)
- ^ (bitwise xor)
- ~ (bitwise not)
- << (bitshift left)
- >> (bitshift right)

Arithmetic Operators

- = (assignment operator)
- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)

Boolean Operators

- && (and)
- || (or)
- ! (not)

Pointer Access Operators

- * dereference operator
- & reference operator

Comparison Operators

- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Compound Operators

- ++ (increment)
- -- (decrement)
- += (compound addition)
- -= (compound subtraction)
- *= (compound multiplication)
- /= (compound division)
- %= (compound modulo)
- &= (compound bitwise and)
- |= (compound bitwise or)

Data Types

- void
- boolean
- char
- unsigned char
- byte
- int
- unsigned int
- word
- long
- unsigned long
- short
- float
- double
- string - char array
- String - object
- array

Constants

- HIGH | LOW
- INPUT | OUTPUT | INPUT_PULLUP
- LED_BUILTIN
- true | false
- integer constants
- floating point constants

Conversion

- char()
- byte()
- int()
- word()
- long()
- float()

Variable Scope & Qualifiers

- variable scope
- static
- volatile
- const

Utilities

- sizeof()
- PROGMEM

Digital I/O

- pinMode()
- digitalWrite()
- digitalRead()

Analog I/O

- analogReference()
- analogRead()
- analogWrite() - *PWM*

Due & Zero only

- analogReadResolution()
- analogWriteResolution()

Random Numbers

- randomSeed()
- random()

Bits and Bytes

- lowByte()
- highByte()
- bitRead()
- bitWrite()
- bitSet()
- bitClear()
- bit()

Advanced I/O

- tone()
- noTone()
- shiftOut()
- shiftIn()
- pulseIn()

Time

- millis()
- micros()
- delay()
- delayMicroseconds()

Math

- min()
- max()
- abs()
- constrain()
- map()
- pow()
- sqrt()

Trigonometry

- sin()
- cos()
- tan()

Characters

- isAlphaNumeric()
- isAlpha()
- isAscii()
- isWhitespace()
- isControl()
- isDigit()
- isGraph()
- isLowerCase()
- isPrintable()
- isPunct()
- isSpace()
- isUpperCase()
- isHexadecimalDigit()

External Interrupts

- attachInterrupt()
- detachInterrupt()

Interrupts

- interrupts()
- noInterrupts()

Communication

- Serial
- Stream

USB (32u4 based boards and Due/Zero only)

- Keyboard
- Mouse

if / else

if/else allows greater control over the flow of code than the basic **if** statement, by allowing multiple tests to be grouped together. For example, an analog input could be tested and one action taken if the input was less than 500, and another action taken if the input was 500 or greater. The code would look like this:

```
if (pinFiveInput < 500)
{
  // action A
}
else
{
  // action B
}
```

else can proceed another **if** test, so that multiple, mutually exclusive tests can be run at the same time.

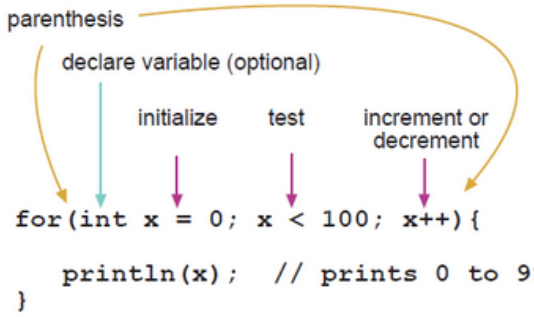
for statements

Description

The **for** statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The **for** statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the **for** loop header:

```
for (initialization; condition; increment) {

//statement(s);    }
```



The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

Example

```
// Dim an LED using a PWM pin
int PWMpin = 10; // LED in series with 470 ohm resistor on pin 10

void setup()
{
  // no setup needed
}

void loop()
{
   for (int i=0; i <= 255; i++){
      analogWrite(PWMpin, i);
      delay(10);
   }
}
```

```
for(int x = 2; x < 100; x = x * 1.5){
println(x);
}
```

Generates: 2,3,4,6,9,13,19,28,42,63,94

Another example, fade an LED up and down with one **for** loop:

```
void loop()
{
   int x = 1;
   for (int i = 0; i > -1; i = i + x){
      analogWrite(PWMpin, i);
      if (i == 255) x = -1;              // switch direction at peak
      delay(10);
   }
}
```

switch / case statements

Like **if** statements, **switch...case** controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The **break** keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

Example

```
switch (var) {
    case 1:
      //do something when var equals 1
      break;
    case 2:
      //do something when var equals 2
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
    break;
  }
```

while loops

Description

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the **while** loop will never exit. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax

```
while(expression){
  // statement(s)
}
```

Expression true: statement(s) is executed

Expression false: statement(s) is not executed and instructions subsequent to the *while* block are executed

Parameters

expression - a (boolean) C statement that evaluates to true or false

Example

```
var = 0;
while(var < 200){
  // do something repetitive 200 times
  var++;
}
```

do - while

The **do** loop works in the same manner as the **while** loop, with the exception that the condition is tested at the end of the loop, so the **do** loop will *always* run at least once.

```
do
{
    // statement block
} while (test condition);
```

Example

```
do
{
  delay(50);           // wait for sensors to stabilize
  x = readSensors();   // check the sensors

} while (x < 100);
```

break

break is used to exit from a **do**, **for**, or **while** loop, bypassing the normal loop condition. It is also used to exit from a **switch** statement.

Example

```
for (x = 0; x < 255; x ++)
{
    analogWrite(PWMpin, x);
    sens = analogRead(sensorPin);
    if (sens > threshold){      // bail out on sensor detect
       x = 0;
       break;
    }
    delay(50);
}
```

continue

The continue statement skips the rest of the current iteration of a loop (**do**, **for**, or **while**). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.

Example

```
for (x = 0; x < 255; x ++)
{
    if (x > 40 && x < 120){       // create jump in values
        continue;
    }

    analogWrite(PWMpin, x);
    delay(50);
}
```

return

Terminate a function and return a value from a function to the calling function, if desired.

Syntax:

return;

return value; // both forms are valid

Parameters

value: any variable or constant type

The return keyword is handy to test a section of code without having to "comment out" large sections of possibly buggy code.

```
void loop(){

// brilliant code idea to test here

return;

// the rest of a dysfunctional sketch here
// this code will never be executed
}
```

Examples:

A function to compare a sensor input to a threshold

```
int checkSensor(){
    if (analogRead(0) > 400) {
        return 1;
    else{
        return 0;
    }
}
```

goto: better not to use it…

Variable Scope

Variables in the C programming language, which Arduino uses, have a property called *scope*. This is in contrast to early versions of languages such as BASIC where every variable is a *global* variable.

A global variable is one that can be *seen* by every function in a program. Local variables are only visible to the function in which they are declared. In the Arduino environment, any variable declared outside of a function (e.g. setup(), loop(), etc.), is a global variable.

When programs start to get larger and more complex, local variables are a useful way to insure that only one function has access to its own variables. This prevents programming errors when one function inadvertently modifies variables used by another function.

It is also sometimes handy to declare and initialize a variable inside a *for* loop. This creates a variable that can only be accessed from inside the for-loop brackets.

Example:

```
int gPWMval;  // any function will see this variable

void setup()
{
  // ...
}

void loop()
{
  int i;    // "i" is only "visible" inside of "loop"
  float f;  // "f" is only "visible" inside of "loop"
  // ...

  for (int j = 0; j <100; j++){
  // variable j can only be accessed inside the for-loop brackets
  }

}
```

pinMode()

Description

Configures the specified pin to behave either as an input or an output. See the description of digital pins for details on the functionality of the pins.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

Syntax

pinMode(pin, mode)

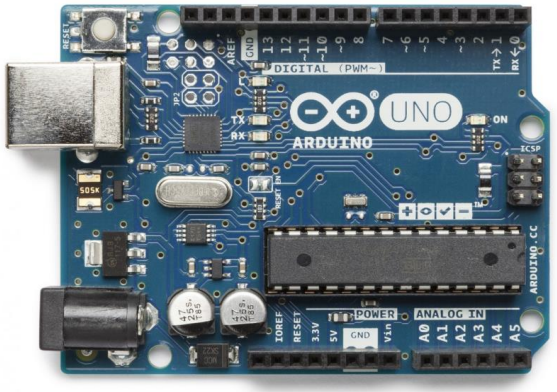
Parameters

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT, or INPUT_PULLUP. (see the digital pins page for a more complete description of the functionality.)

Returns

None



Atmega168 Pin Mapping

Arduino function				Arduino function
reset	(PCINT14/RESET) PC6	1	28 PC5 (ADC5/SCL/PCINT13)	analog input 5
digital pin 0 (RX)	(PCINT16/RXD) PD0	2	27 PC4 (ADC4/SDA/PCINT12)	analog input 4
digital pin 1 (TX)	(PCINT17/TXD) PD1	3	26 PC3 (ADC3/PCINT11)	analog input 3
digital pin 2	(PCINT18/INT0) PD2	4	25 PC2 (ADC2/PCINT10)	analog input 2
digital pin 3 (PWM)	(PCINT19/OC2B/INT1) PD3	5	24 PC1 (ADC1/PCINT9)	analog input 1
digital pin 4	(PCINT20/XCK/T0) PD4	6	23 PC0 (ADC0/PCINT8)	analog input 0
VCC	VCC	7	22 GND	GND
GND	GND	8	21 AREF	analog reference
crystal	(PCINT6/XTAL1/TOSC1) PB6	9	20 AVCC	VCC
crystal	(PCINT7/XTAL2/TOSC2) PB7	10	19 PB5 (SCK/PCINT5)	digital pin 13
digital pin 5 (PWM)	(PCINT21/OC0B/T1) PD5	11	18 PB4 (MISO/PCINT4)	digital pin 12
digital pin 6 (PWM)	(PCINT22/OC0A/AIN0) PD6	12	17 PB3 (MOSI/OC2A/PCINT3)	digital pin 11(PWM)
digital pin 7	(PCINT23/AIN1) PD7	13	16 PB2 (SS/OC1B/PCINT2)	digital pin 10 (PWM)
digital pin 8	(PCINT0/CLKO/ICP1) PB0	14	15 PB1 (OC1A/PCINT1)	digital pin 9 (PWM)

Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

digitalWrite()

Description

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor. See the digital pins tutorial for more information.

NOTE: If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.

Syntax	Parameters	Returns
digitalWrite(pin, value)	pin: the pin number value: HIGH or LOW	none

digitalRead()

Description

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax

digitalRead(pin)

Parameters

pin: the number of the digital pin you want to read (*int*)

Returns

HIGH or LOW

Note: To prevent that an input pin to which nothing is connected (e.g. an open switch) remain in an unknown or uncertain state, a “pull-up” or “pull-down” resistor must be used (this function is also available via software, see pinMode INPUT_PULLUP)

analogWrite()

Description

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to **analogWrite()**, the pin will generate a steady square wave of the specified duty cycle until the next call to **analogWrite()** (or a call to **digitalRead()** or **digitalWrite()** on the same pin). The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz. Pins 3 and 11 on the Leonardo also run at 980 Hz.

On most Arduino boards (those with the ATmega168 or ATmega328), this function works on pins 3, 5, 6, 9, 10, and 11. On the Arduino Mega, it works on pins 2 - 13 and 44 - 46. Older Arduino boards with an ATmega8 only support analogWrite() on pins 9, 10, and 11.

The Arduino Due supports analogWrite() on pins 2 through 13, plus pins DAC0 and DAC1. Unlike the PWM pins, DAC0 and DAC1 are Digital to Analog converters, and act as true analog outputs.

You do not need to call pinMode() to set the pin as an output before calling analogWrite().

The *analogWrite* function has nothing to do with the analog pins or the *analogRead* function.

Syntax

analogWrite(pin, value)

Parameters

pin: the pin to write to.

value: the duty cycle: between 0 (always off) and 255 (always on).

Returns

nothing

analogRead()

Description

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference().

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

analogRead(pin)

Parameters

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns

int (0 to 1023)

Note

If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

#include

#include is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

The main reference page for AVR C libraries (AVR is a reference to the Atmel chips on which the Arduino is based) is here.

Note that **#include**, similar to **#define**, has no semicolon terminator, and the compiler will yield cryptic error messages if you add one.

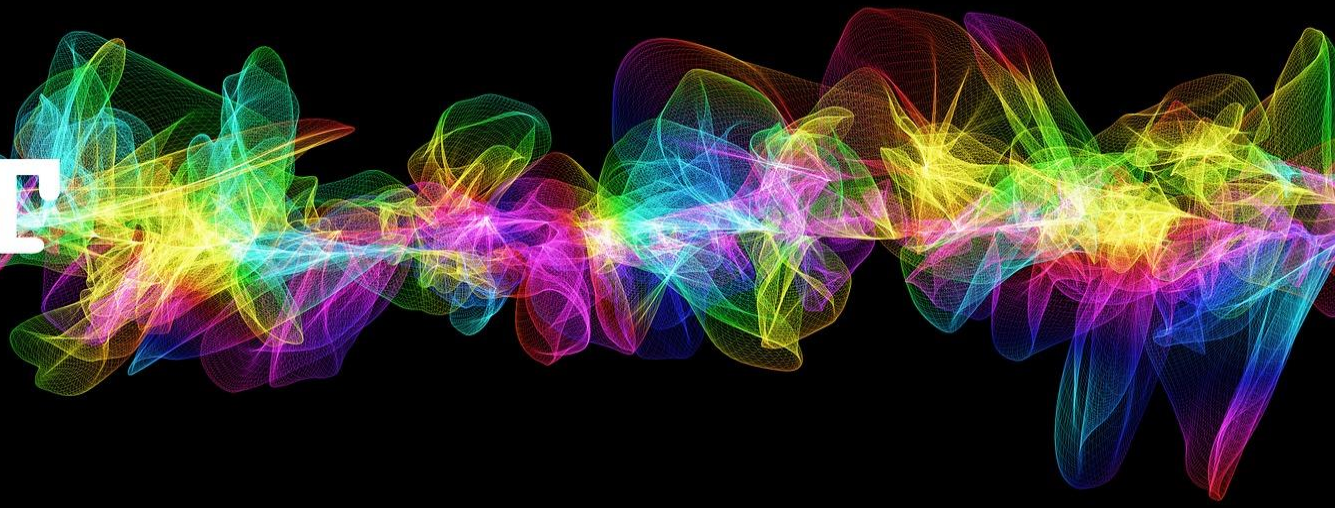
Example

This example includes a library that is used to put data into the program space *flash* instead of *ram*. This saves the ram space for dynamic memory needs and makes large lookup tables more practical.

```
#include <avr/pgmspace.h>

prog_uint16_t myConstants[] PROGMEM = {0, 21140, 702   , 9128,  0, 25764, 8456,
0,0,0,0,0,0,0,0,29810,8968,29762,29762,4500};
```

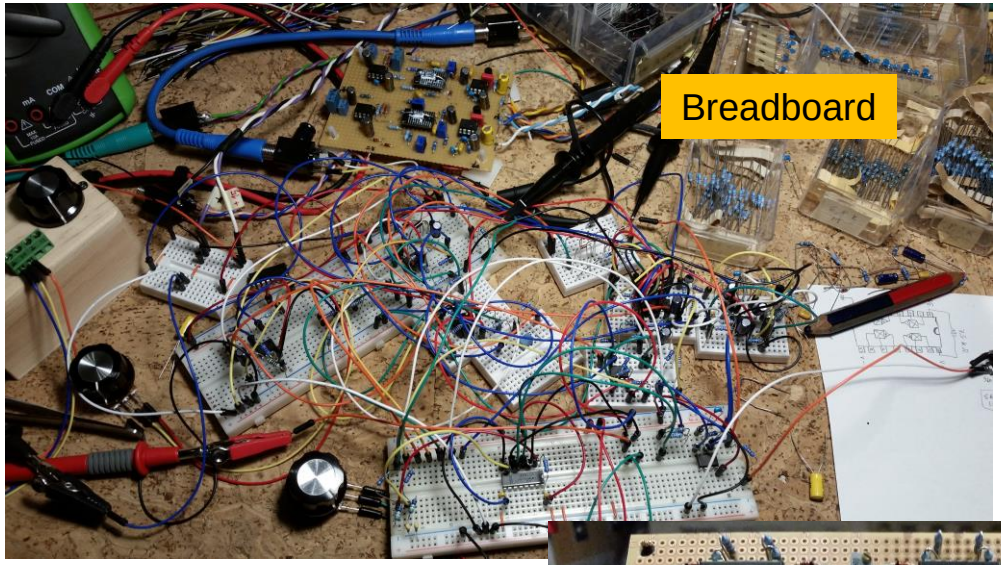
MEASUREMENTS WITH ARDUINO

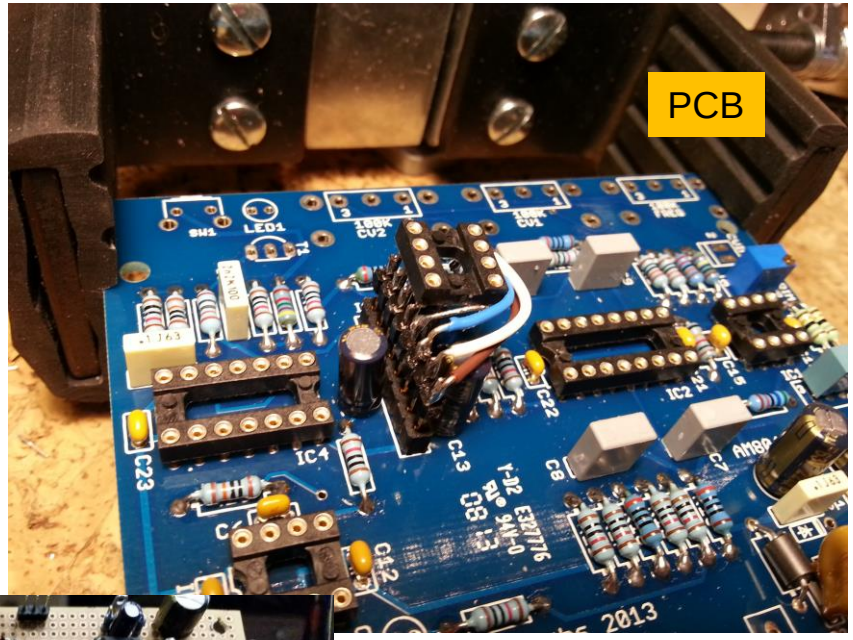
START

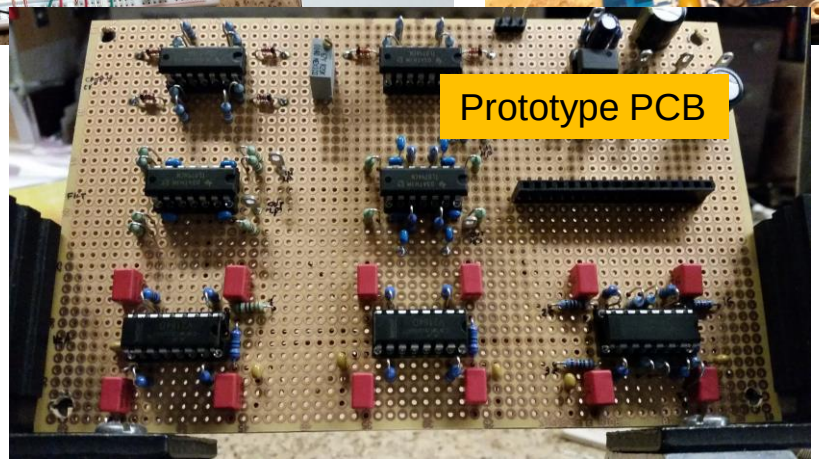


Let's start playing with the ARDUINO board!

Breadboards vs *Prototype Board* vs *PCB*







Reading the value of a potentiometer

- Potentiometer between mass and 5V
- Central pin potentiometer to input pin A0
- Central pin potentiometer to oscilloscope

AnalogReadSerial

```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5V and ground.

 This example code is in the public domain.
 */

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

// 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:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```

SETUP

LOOP

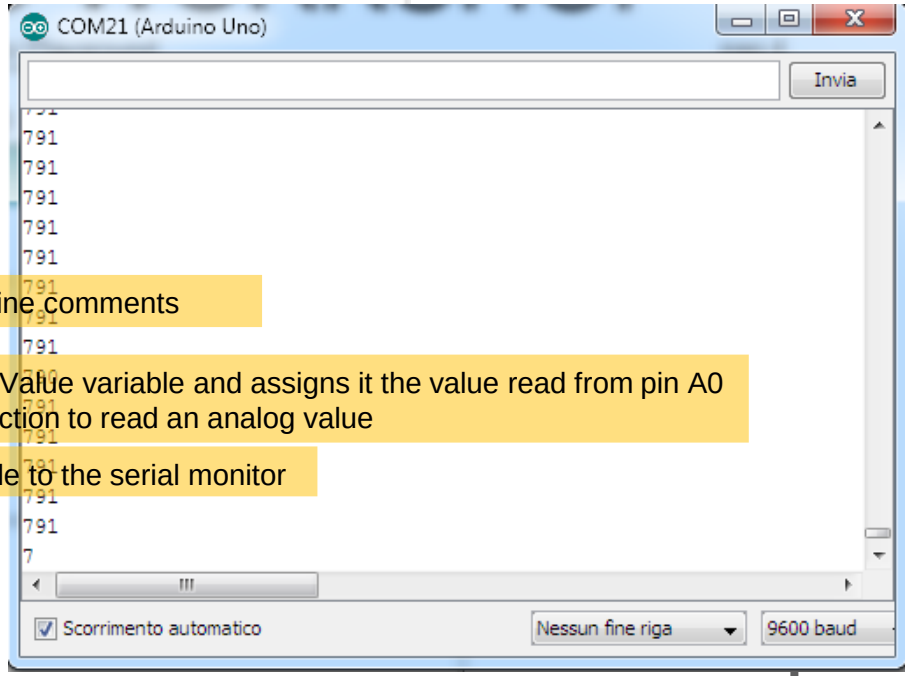
Comments on multiple lines

Initialize serial monitor communication

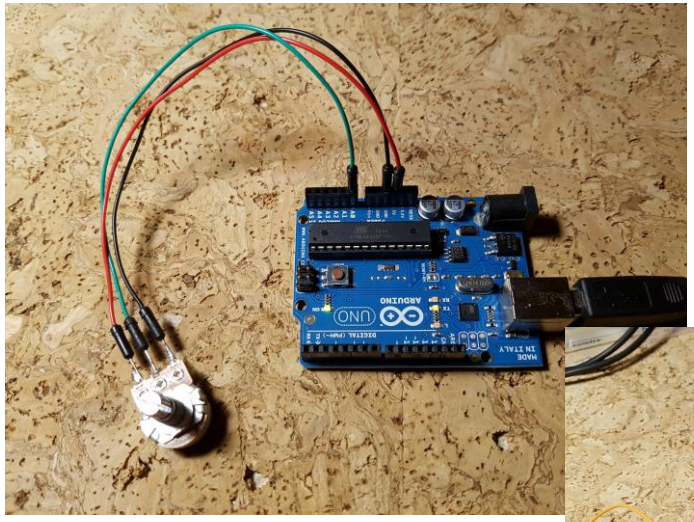
Single line comments

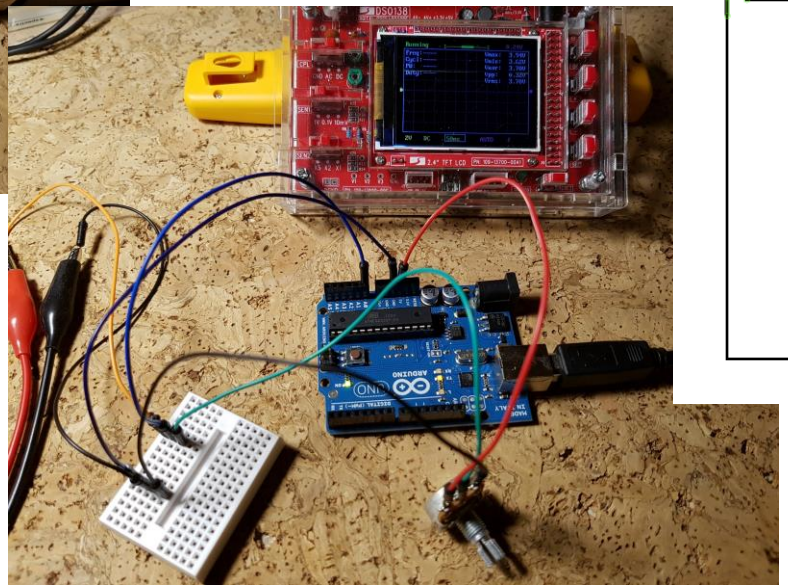
Initializes the sensorValue variable and assigns it the value read from pin A0
AnalogRead is a function to read an analog value

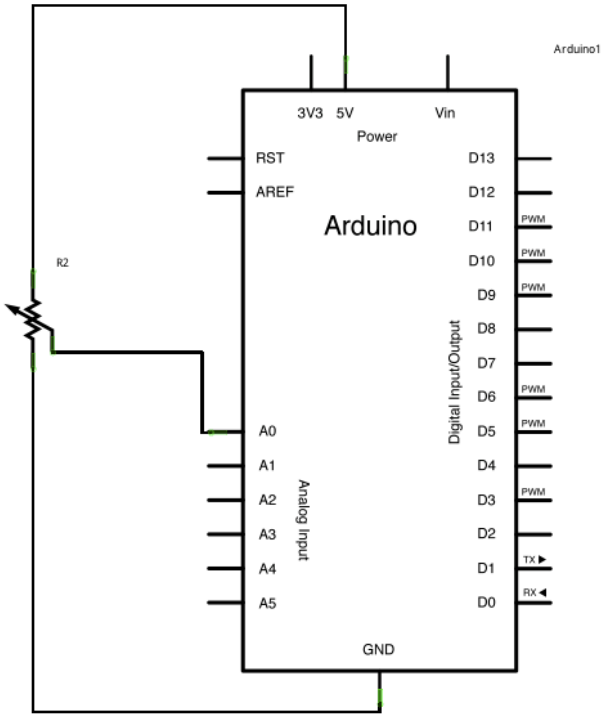
Prints the value of the variable to the serial monitor



Reading the value of a potentiometer







From the value of a potentiometer to the PWM output - Changing the brightness of an LED

AnalogInOutSerial

```
/*
  Analog input, analog output, serial output

 Reads an analog input pin, maps the result to a range from 0 to 255
 and uses the result to set the pulsewidth modulation (PWM) of an output pin.
 Also prints the results to the serial monitor.

 The circuit:
 * potentiometer connected to analog pin 0.
   Center pin of the potentiometer goes to the analog pin.
   side pins of the potentiometer go to +5V and ground
 * LED connected from digital pin 9 to ground

 created 29 Dec. 2008
 modified 9 Apr 2012
 by Tom Igoe

 This example code is in the public domain.

 */

// These constants won't change.  They're used to give names
// to the pins used:
const int analogInPin = A0;  // Analog input pin that the potentiometer is attached
const int analogOutPin = 9; // Analog output pin that the LED is attached to

int sensorValue = 0;        // value read from the pot
int outputValue = 0;        // value output to the PWM (analog out)

void setup() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
}
```

Writes the PWM value to the output pin: sets the pulse width

```
void loop() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);
  // map it to the range of the analog out:
  outputValue = map(sensorValue, 0, 1023, 0, 255);
  // change the analog out value:
  analogWrite(analogOutPin, outputValue);

  // print the results to the serial monitor:
  Serial.print("sensor = " );
  Serial.print(sensorValue);
  Serial.print("\t output = ");
  Serial.println(outputValue);

  // wait 2 milliseconds before the next loop
  // for the analog-to-digital converter to settle
  // after the last reading:
  delay(2);
}
```

Re-maps values from 0..1023 to 0..255

print and stay in line

\t → tab

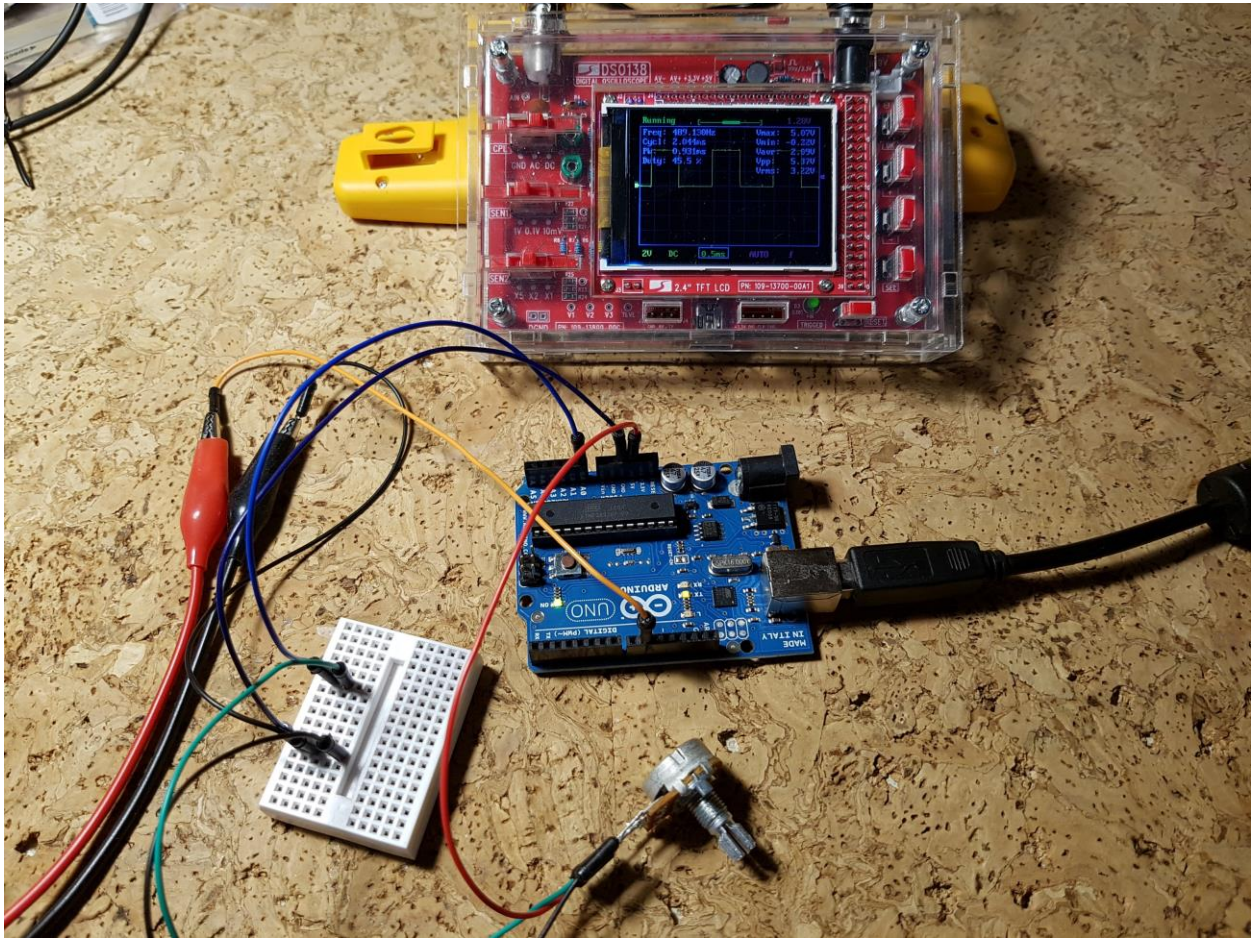
print and «new line»

COM21 (Arduino Uno)

```
sensor = 470    output = 117
sensor = 469    output = 116
sensor = 469    output = 116
sensor = 470    output = 117
sensor = 470    output
```

- Potentiometer between mass and 5V
- Central pin potentiometer to input pin A0
- Output pin 9 to a LED (with 1K resistor)
- Central pin potentiometer to oscilloscope

From the value of a potentiometer to the PWM output - Changing the brightness of a LED



Continuously variable PWM output

_03_Fade

```
/*
 Fade

 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 This example code is in the public domain.
 */

int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}
```

Increase or decrease brightness (PWM)

```
// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

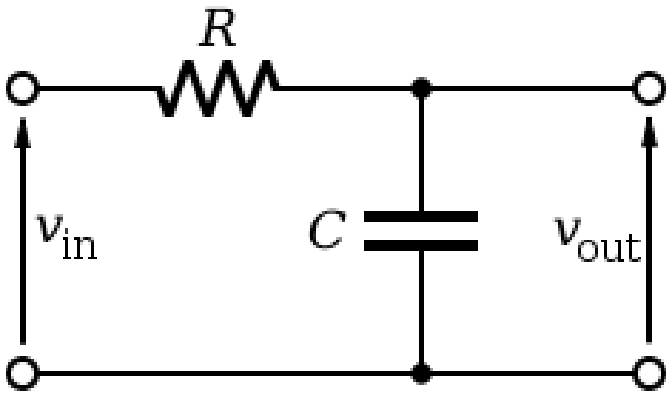
Change the brightness (PWM) to positive or negative

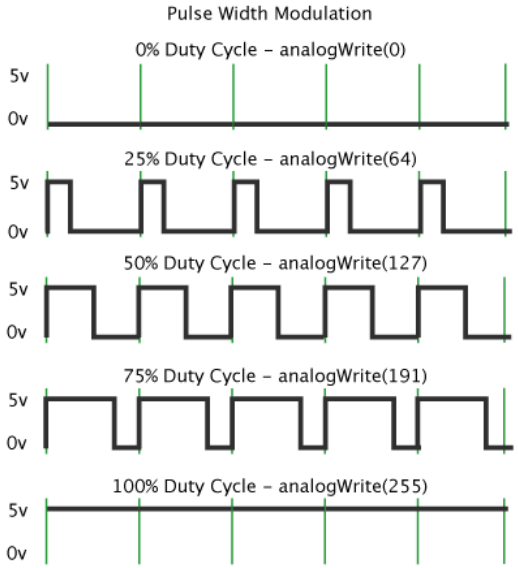
- Output pin 9 to a LED (with 1K resistor)
- Output pin 9 to oscilloscope

From the PWM signal to a continue voltage. «Poor man DAC»

In Arduino (except for the DUE model) there are no DACs on board, but only ADC inputs. Using the PWM outputs, however, it is possible to obtain a DC voltage. The frequency of the Arduino PWM modulator is of about 490 Hz, but it can be modified via some internal registers. Assuming, however, to leave the frequency at 490 Hz, a low-pass filter placed on the output pin allows to obtain a DC voltage, to be used for various uses, making up for the lack of a real DAC (we will use them soon), but with some limitations. A low pass filter of the first order is used, built in the simplest way, namely with a resistor and a capacitor, placed at the output of a pin with signal PWM. Arduino allows to vary the pulse width with a resolution of 8 bits (256 values), from 0% to 100%, i.e. 256 possible pulse widths are possible. A simulation of the effect of low pass filtering, to understand the limits of this solution, is available on this page :

http://sim.okawa-denshi.jp/en/PWMtool.php





MEASUREMENTS WITH ARDUINO

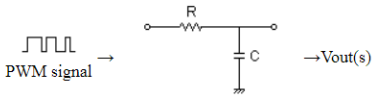
OKAWA Electric Design

Top > Tools > Filters > RC Low-pass Filter Design for PWM > Result

RC Low-pass Filter Design for PWM - Result -

Calculated peak-to-peak ripple voltage and settling time at a given PWM frequency and cut-off frequency or values of R and C.

RC Filter



Transfer Function:

$$G(s)=\frac{666.66666666667}{s+666.66666666667}$$

Cut-off frequency

fc = 106.1032953946[Hz]

Final Vout value of the step response (without a ripple)

f_{PWM}=490 Hz

Duty Step 0%→50 [%]

PWM signal voltage:

V_L = 0 [V] V_H = 5 [V]

R and C values of filter | Cut-off frequency

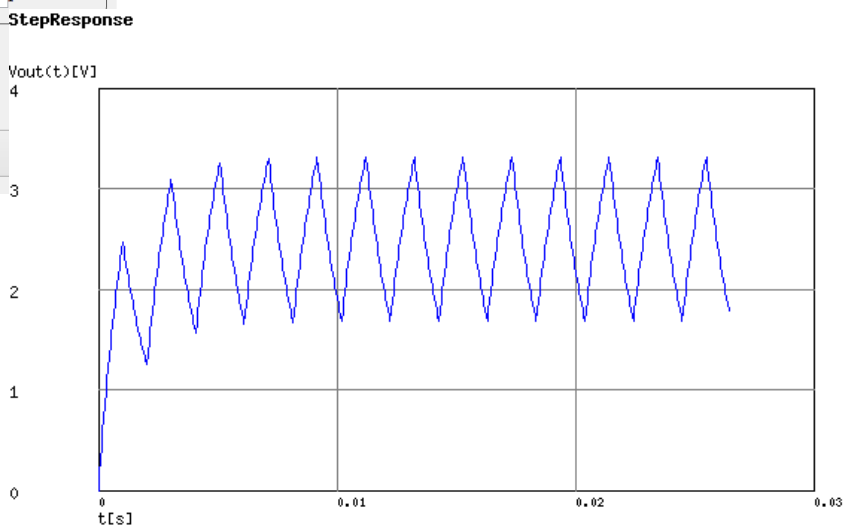
○ Cut-off frequency fc = [Hz]

◉ R and C values

R=15000 Ω C=0.1u F

p:pico, n:nano, u:micro, k:kilo, M:mega

Calculate



- Cutoff frequency about 100 Hz
- Fast to reach maximum average value
- Very high ripple on output

http://sim.okawa-denshi.jp/en/PWMtool.php

https://en.wikipedia.org/wiki/Low-pass_filter

(c)okawa-denshi.jp

MEASUREMENTS WITH ARDUINO

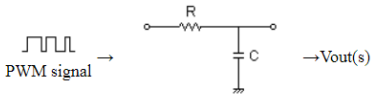
OKAWA Electric Design

Top > Tools > Filters > RC Low-pass Filter Design for PWM > Result

RC Low-pass Filter Design for PWM - Result -

Calculated peak-to-peak ripple voltage and settling time at a given PWM frequency and cut-off frequency or values of R and C.

RC Filter



Transfer Function:

$$G(s)=\frac{66.666666666667}{s+66.666666666667}$$

Cut-off frequency

fc = 10.61032953946[Hz]

Final Vout value of the step response (without a ripple)

f_{PWM}= 490 Hz

Duty Step 0%→ 50 [%]

PWM signal voltage:

V_L = 0 [V] V_H = 5 [V]

R and C values of filter | Cut-off frequency

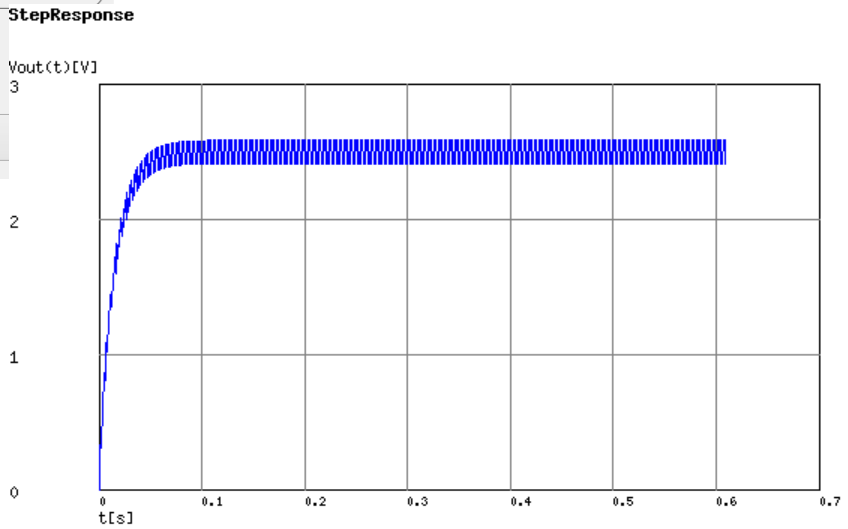
○ Cut-off frequency fc = [Hz]

◉ R and C values

R= 15000 Ω C= 1u F

p:pico, n:nano, u:micro, k:kilo, M:mega

Calculate



- Cutoff frequency about 10 Hz
- Less fast to reach maximum average value
- Still some ripple on output

http://sim.okawa-denshi.jp/en/PWMtool.php

https://en.wikipedia.org/wiki/Low-pass_filter

MEASUREMENTS WITH ARDUINO

From the PWM signal to a continue voltage. «Poor man DAC»

OKAWA Electric Design

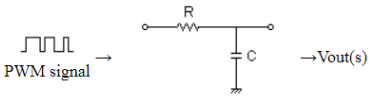
English

Top > Tools > Filters > RC Low-pass Filter Design for PWM > Result

RC Low-pass Filter Design for PWM - Result -

Calculated peak-to-peak ripple voltage and settling time at a given PWM frequency and cut-off frequency or values of R and C.

RC Filter



Transfer Function:

$$G(s)=\frac{6.6666666666667}{s+6.6666666666667}$$

Cut-off frequency

fc = 1.061032953946[Hz]

f_{PWM}= 490 Hz

Duty Step 0%→ 50 [%]

PWM signal voltage:

V_L = 0 [V] V_H = 5 [V]

R and C values of filter | Cut-off frequency

○ Cut-off frequency fc = [Hz]

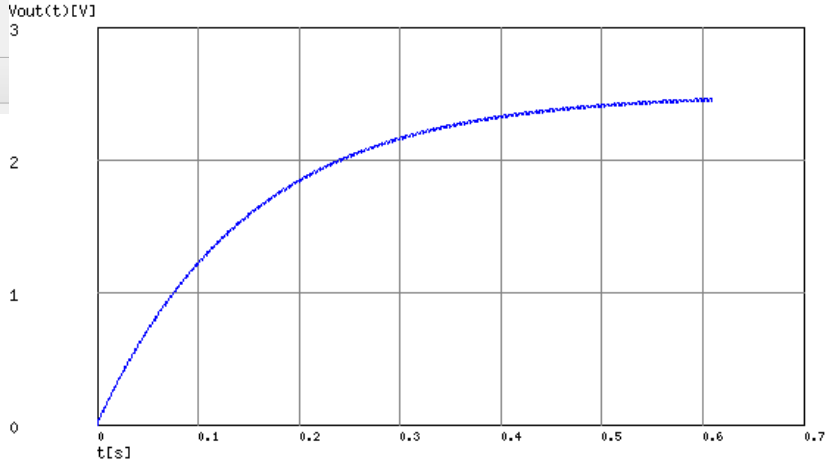
◉ R and C values

R= 15000 Ω C= 10u F

p:pico, n:nano, u:micro, k:kilo, M:mega

Calculate

StepResponse



- Cutoff frequency about 1 Hz
- Slow to reach maximum average value
- Very little ripple on output

http://sim.okawa-denshi.jp/en/PWMtool.php

https://en.wikipedia.org/wiki/Low-pass_filter

(c)okawa-denshi.jp

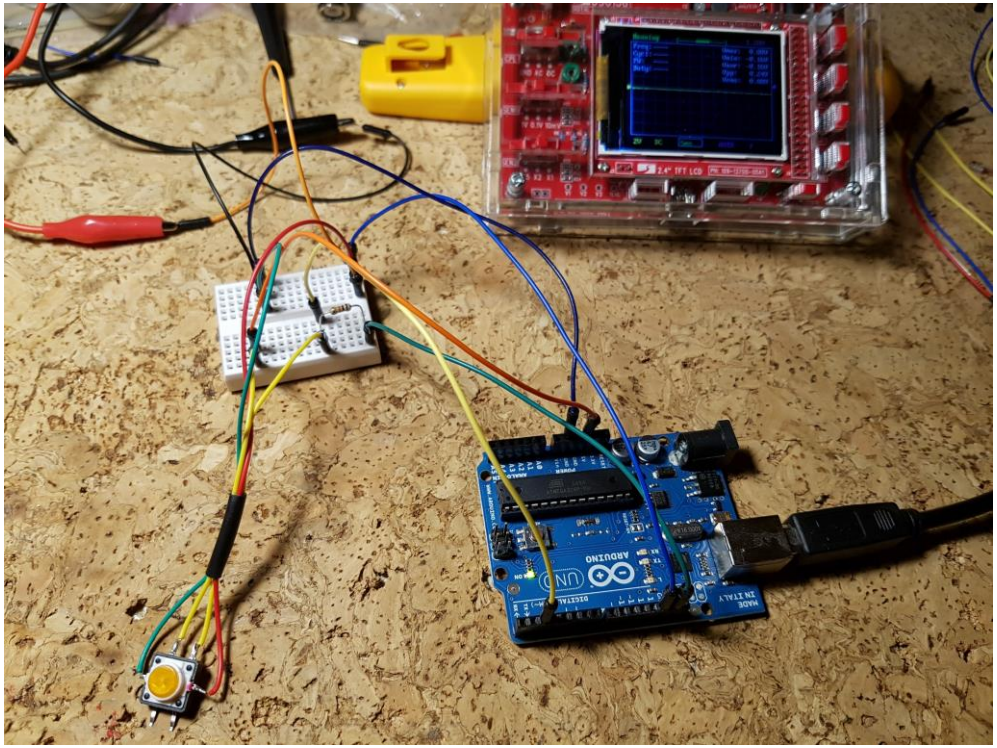
Using a button. Pull-up and pull-down resistors

- Button with built-in LED

- The LED is connected to pin 13

- The button connects pin 2 to the 5 V when pressed

- **A 10K resistance is connected between pin 2 and ground**



WHY?

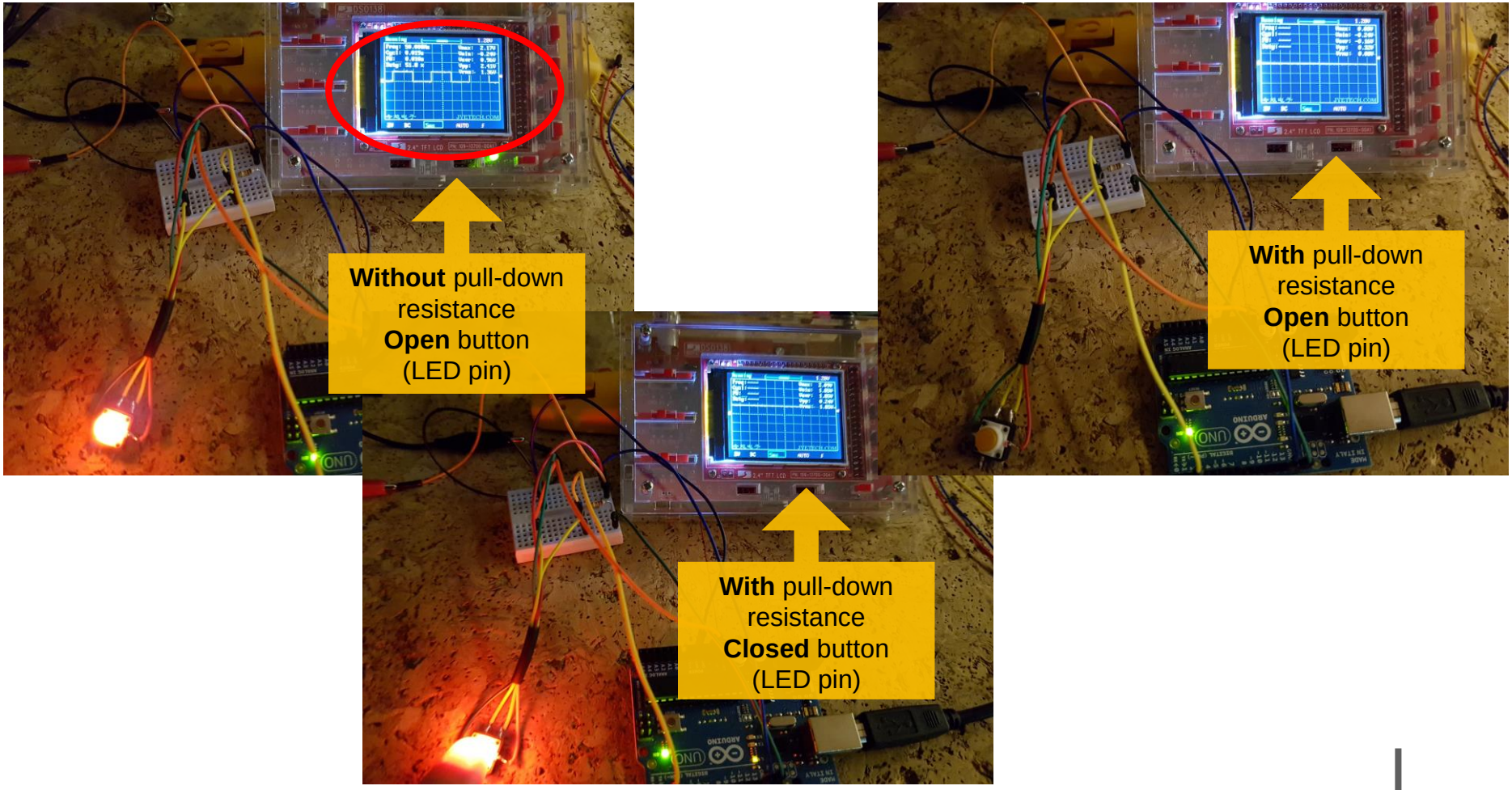
Using a button. Pull-up and pull-down resistors

_04_Button

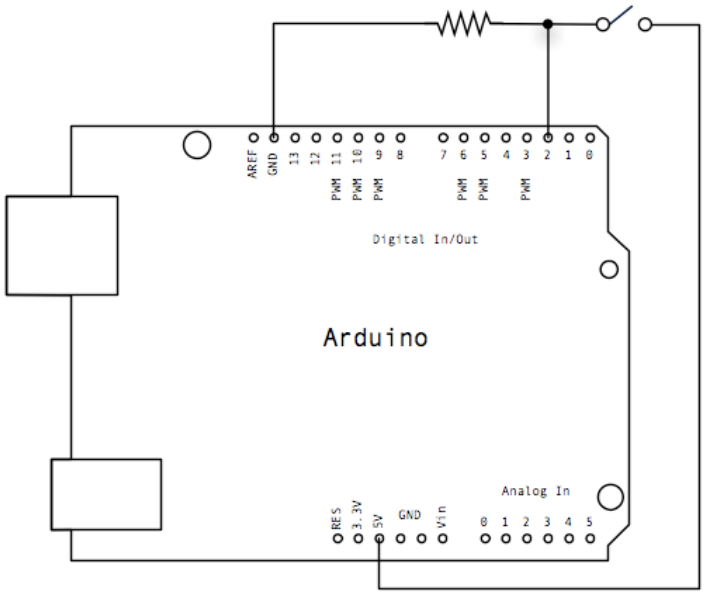
```
/*
  Button

 Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when pressing a pushbutton attached to pin 2.


 The circuit:
 * LED attached from pin 13 to ground
 * pushbutton attached to pin 2 from +5V
 * 10K resistor attached to pin 2 from ground

 * Note: on most Arduinos there is already an LED on the board
 attached to pin 13.


 created 2005
 by DojoDave <http://www.0j0.org>
 modified 30 Aug 2011
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/Button
 */

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status
```

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Using a button. Pull-up and pull-down resistors



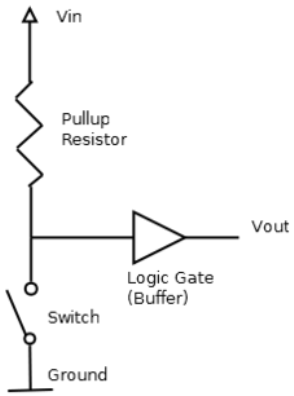
Using a button. Pull-up and pull-down resistors



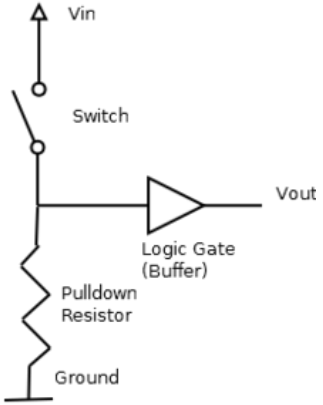
The digital input, connected to the button, requires the use of an additional resistor, connected to ground or to +5V, depending on the circuit. This because, in the absence of it, when the circuitry is open (i.e. if the button is not pressed) the input, high impedance, results in a state indefinite, picking up disturbances (e.g. 50 Hz mains). Therefore, the resistance is used to define a state "by default" at the pin input, that the button leads to 0 or +5V. The use of a resistor (e.g. 10 KOhm) and not of a direct connection ensures that there are no short circuits when the button is pressed. So, in the example circuit (pull-down resistor), when the push button is open at pin 2 there are 0V (ground), when the button is pressed at pin 2 there are +5V.

Pull-up and pull-down resistors

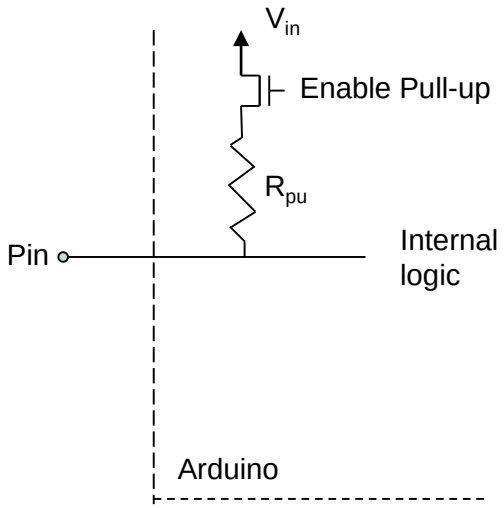
Pull-up



Pull-down



Internal Pull-up



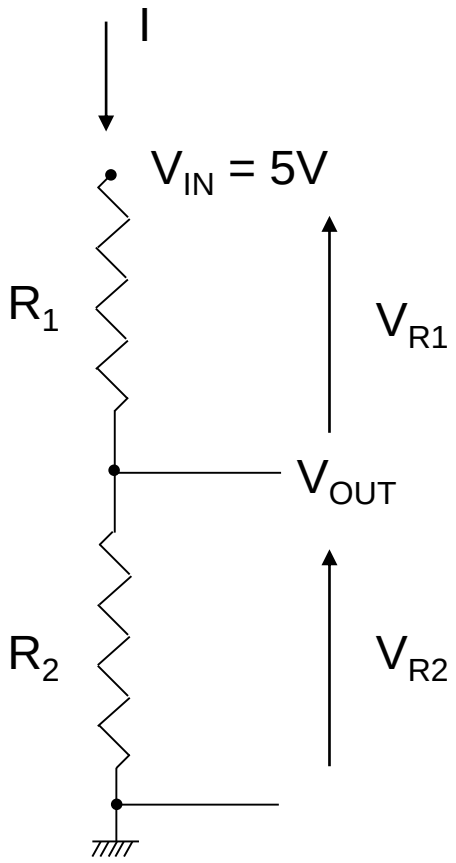
NOTE: as seen previously, the pinMode function has among its different *modes* the option INPUT or OUTPUT or **INPUT_PULLUP**.

It is possible to set up via software a pullup resistor (about 20K) already present inside the microcontroller. This operation is (electrically) identical to setting the **HIGH** logic level of the pin (when in use as output).

```
pinMode(pin, INPUT);
digitalWrite(pin, HIGH);
```

```
pinMode(pin, INPUT_PULLUP)
```

The voltage divider



$$V_{R1} = R_1 \cdot I$$
$$V_{R2} = R_2 \cdot I$$

$$V_{IN} = (R_1 + R_2) \cdot I$$

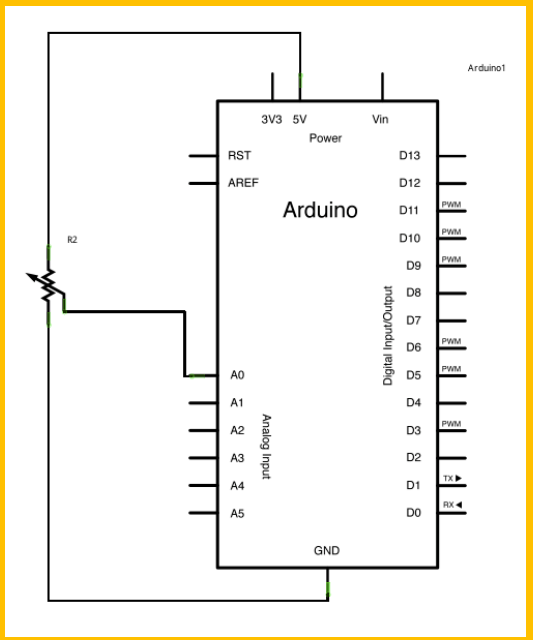
$$I = \frac{V_{IN}}{R_1 + R_2}$$

$$V_{R1} = V_{IN} \cdot \frac{R_1}{R_1 + R_2}$$

$$V_{R2} = V_{IN} \cdot \frac{R_2}{R_1 + R_2} = V_{OUT}$$

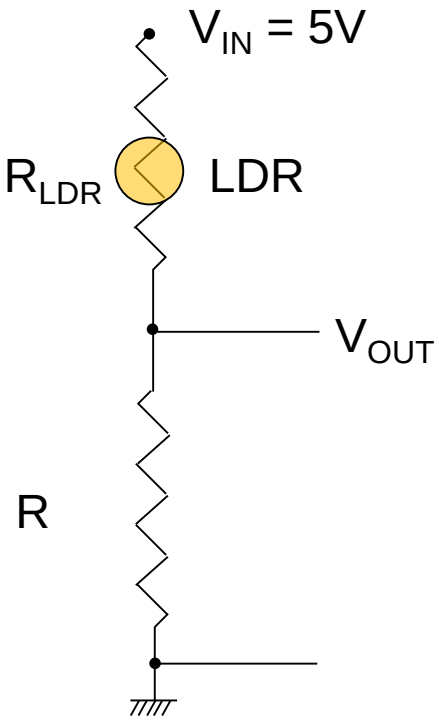
The potentiometer is a voltage divider:

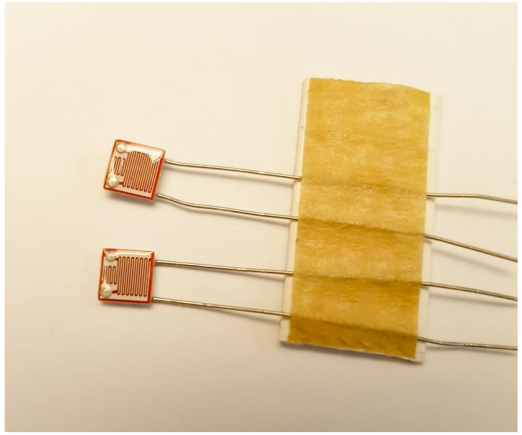
- $R_2 = 0 \rightarrow V_{R2} = 0$
- $R_1 = 0 \rightarrow V_{R2} = VIN$
- $R_1 = R_2 \rightarrow V_{R2} = V_{IN}/2$



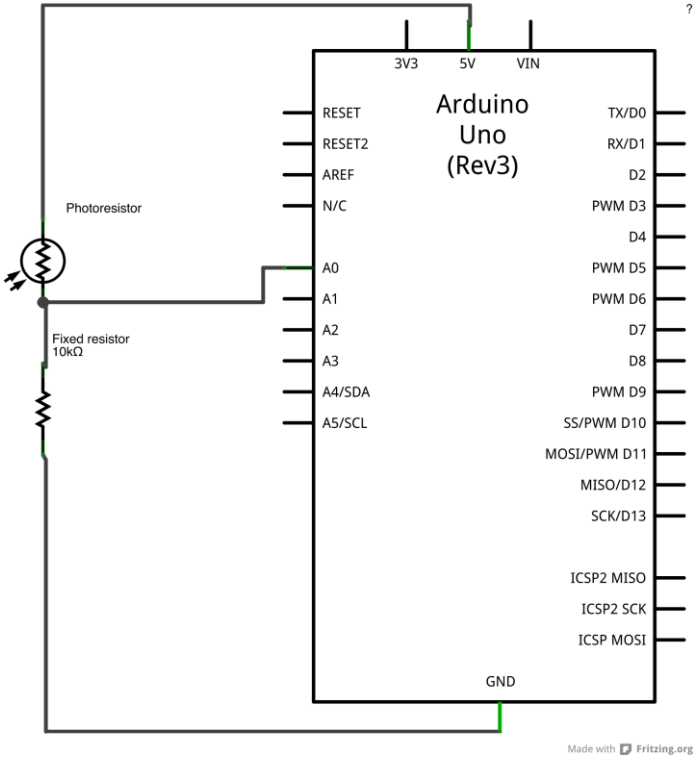
→ Pull-up and pull-down resistors

Other resistive sensors: the photocell





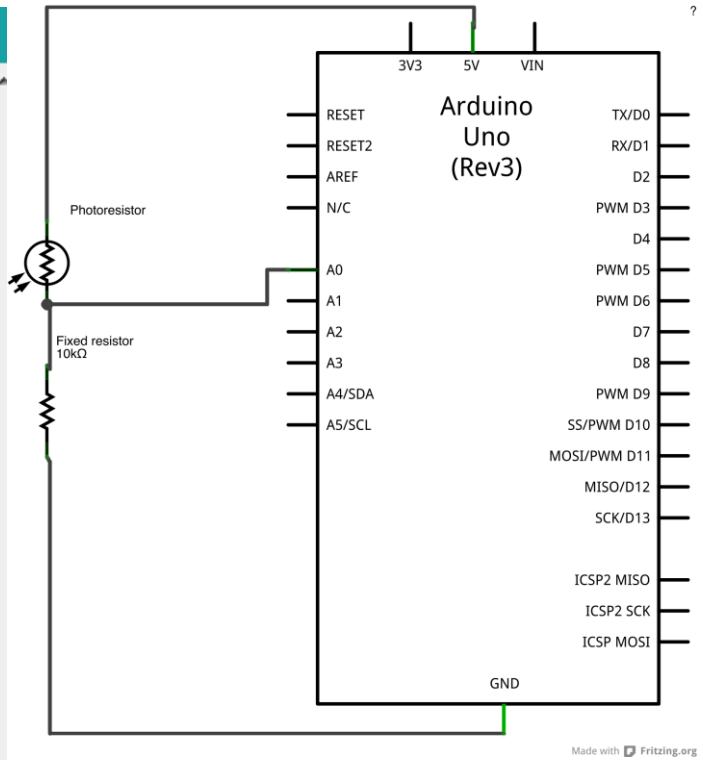
$$V_{OUT} = V_{IN} \cdot \frac{R}{R_{\mathrm{LDR}} + R}$$



Other resistive sensors: the photocell

_01_WS_AnalogReadSerial

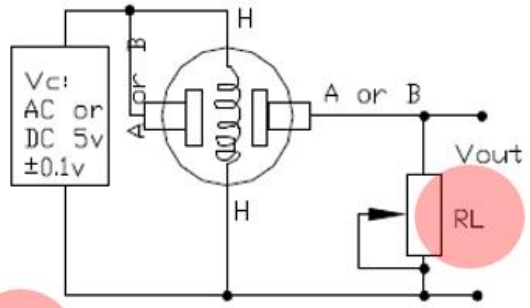
```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Attach the center pin of a potentiometer to pin A0, and the outside pins to +5

 This example code is in the public domain.
 */

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

// 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:
  Serial.println(sensorValue);
  delay(1);        // delay in between reads for stability
}
```



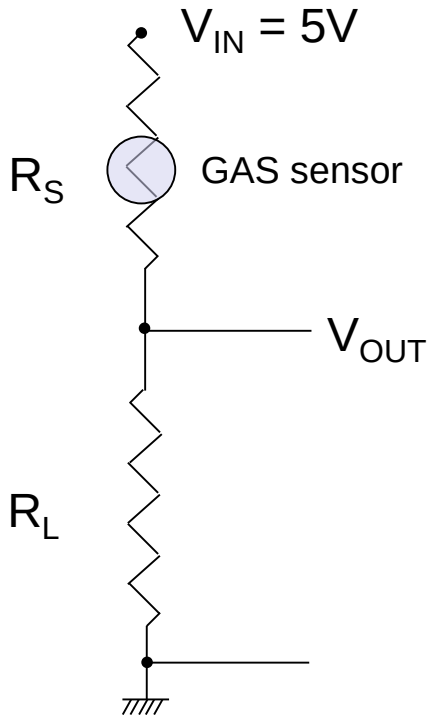
Try with both R = 10K and R = 1K

$$V_{OUT} = V_{IN} \cdot \frac{R}{R_{\mathrm{LDR}} + R}$$

Other resistive sensors: (analog) GAS sensor

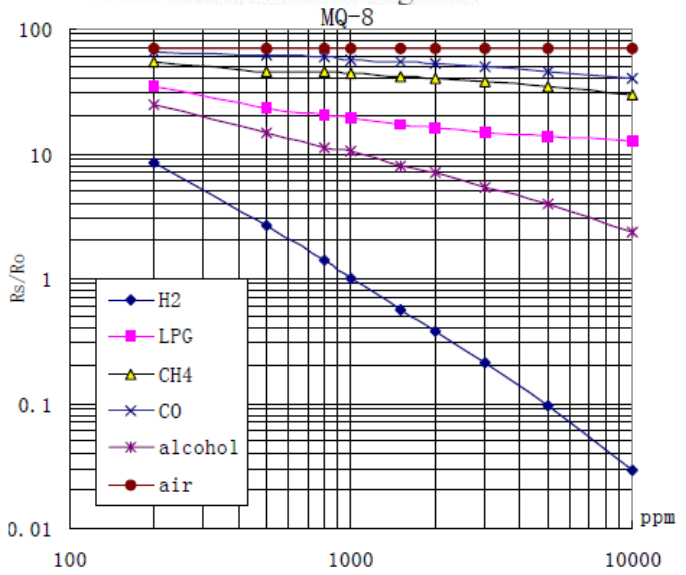


Ro: sensor resistance at 1000ppm of H_2 in the clean air.
Rs:sensor resistance at various concentrations of gases.



$$V_{OUT} = V_{IN} \cdot \frac{R_L}{R_S + R_L}$$

$$R_S = \frac{R_L \cdot (V_{IN} - V_{OUT})}{V_{OUT}}$$



Temperature sensors

- ANALOG SENSORS

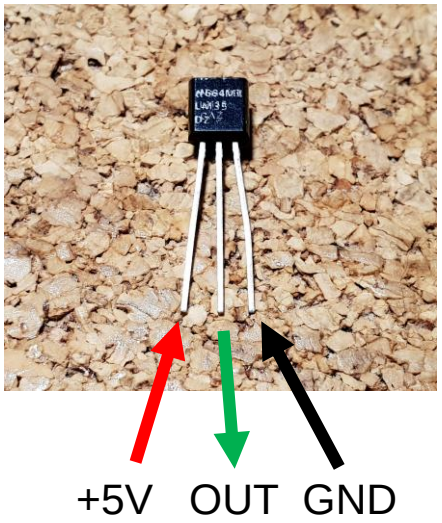
The measured thermodynamic variable is converted inside the sensor in the form of voltage, according to a variation law specified by the manufacturer. The Arduino ADC reads this voltage directly and, according to the manufacturer's specification, the measured parameter can be calculated directly in the Arduino sketch. For example the temperature sensor **LM35**:

PARAMETER	TEST CONDITIONS	LM35A			LM35CA			UNIT
		TYP	TESTED LIMIT(1)	DESIGN LIMIT(2)	TYP	TESTED LIMIT(1)	DESIGN LIMIT(2)	
Sensor gain (average slope)	$T_{MIN} \le T_A \le T_{MAX}$	10	9.9		10		9.9	mV/°C
	$-40°C \le T_J \le 125°C$	10	10.1		10		10.1	

- DIGITAL SENSORS

The measured parameters are transmitted to Arduino through a digital communication protocol (e.g. SPI or I^2C). The user does not need to know in detail the low level software instructions of the communication with the sensor because software libraries are provided by the sensor manufacturer (or by the user's community). Once the sensor libraries have been installed in the Arduino IDE, at first use, in the sketch it is sufficient to ***#include*** the library related to the sensor and the reading of the parameters is done usually just with a single software instruction.

LM 35 - ANALOG temperature sensor



It is a simple analog sensor, which provides an output voltage proportional to the measured temperature, varying **10 mV/°C**.

There are various models, for example LM35A has operating ranges between -55°C and 150°C and LM35D has operating ranges between 0°C and 100°C.

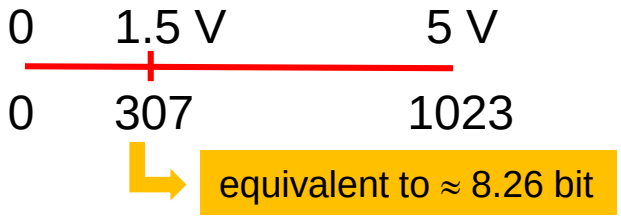
The 6 analog inputs of Arduino UNO have a resolution of 10 bits, that is 1024 possible values, measuring by default an input voltage variable between 0 and 5 V (outside these values the device is damaged).

$$\text{ADC INPUT } 0..5\text{ V} \Rightarrow \frac{5}{1024} = 0.00488\text{ V} = 4.88\text{ mV (each amplitude step)}$$

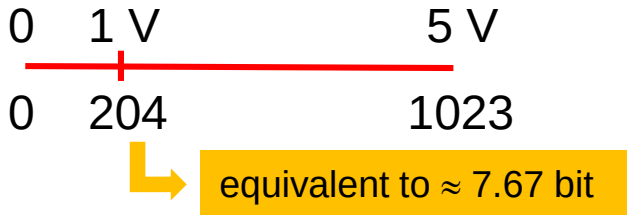
$$\frac{10\text{ mV}}{\text{°C}} \Rightarrow \text{measurement resolution} \simeq 0.48\text{ °C}$$

LM 35 - ANALOG temperature sensor

LM35A: max 150 °C ⇒ **max 1.5 V** at Arduino ADC input



LM35D: max 100 °C ⇒ **max 1 V** at Arduino ADC input



In both cases, there is a "waste" of bits, in other words of resolution, because the whole ADC measurement range (5V over 10 bit = 1024 steps) is not used.

tempC = aRead*0.488

Example of temperature measurement:

The Arduino program reads the value **43** on the ADC. Remembering that the read value can vary from 0 to 1023 and that each step is 4.88 mV (**0.48 °C**), the input voltage to the ADC is nominally **43*4.88 = 210 mV** (0.21 V), corresponding to **21 °C**.

LM 35 - ANALOG temperature sensor

In Arduino a “trick” can be used to increase the accuracy of the measurement, when the range of measured voltages is narrow. The voltage range of the ADC inputs can be set with the **analogReference** software instruction:

analogReference(**DEFAULT**); // The ADC range is the Arduino voltage (5V or 3.3V)
analogReference(**INTERNAL**); // The ADC range is 1.1 V
analogReference(**EXTERNAL**); // The ADC range is the voltage applied to EXT pin (≤ 5 V)

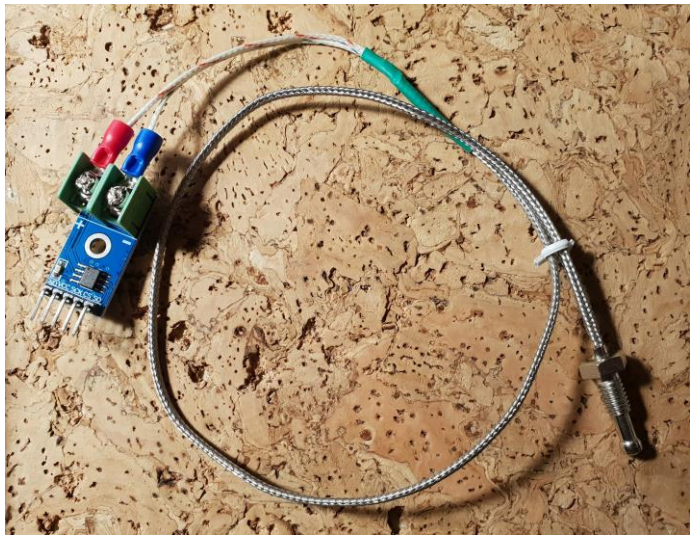
Setting to **INTERNAL**: $\text{ADC INPUT } 0 \,..\, 1.1\text{ V} \Rightarrow \frac{1.1}{1024} = 0.00107\text{ V} \cong 1.07\text{ mV}$

The resolution is now about **0.1 °C** instead of **0.48 °C**

tempC = aRead*0.10742

NOTE: the ADC input voltage must never exceed the value set with the analogReference instruction. Setting **INTERNAL**, the max measurable temperature with LM35 is 110 °C

Thermocouple + MAX6675 digital interface



A thermocouple can be used very easily with Arduino by using the **MAX6675** circuit, which contains a 12-bit ADC and automatically applies cold junction compensation. Type K thermocouples must be used and the temperature resolution is 0.25 °C. Communication takes place with Arduino using the SPI protocol.

Software instructions for the use:

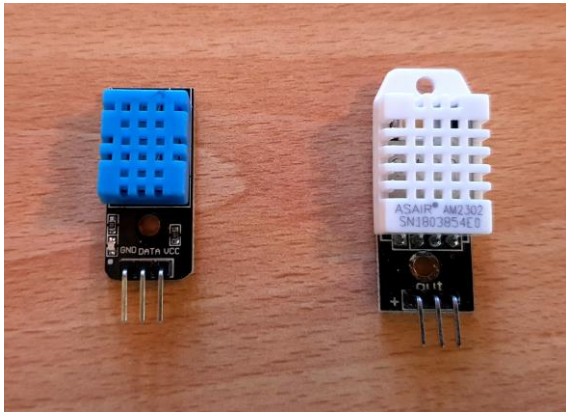
```
#include "max6675.h“
…
// Pin connections (SPI bus)
int thermoSO = 4;
int thermoCS = 5;
int thermoSCK = 6;
…
thermocouple.readCelsius()
```

// library to be included at beginning of the source code

// function to read the temperature from the MAX6675

Before first use, the MAX6675 library by Adafruit must be installed in the Arduino IDE

Temperature and Humidity sensors DHT11 and DHT22



These 2 low cost devices contain a temperature sensor and a relative humidity sensor and communicate with Arduino digitally, using a single wire.
DHT11 (blue): temperature range 0..50 °C, error on relative humidity 5%.
DHT22 (white): temperature range -40..80 °C, error on relative humidity 2%.

Software instructions for the use:

```
#include "DHT.h“        // library to be included at beginning of the source code
…
#define DHTPIN 2      // Arduino digital pin connected to the DHT sensor
#define DHTTYPE DHT11    // set DHT11 or DHT22
…
DHT dht1(DHTPIN, DHTTYPE);  // Initialize the sensor
float h;
float t;
…
dht1.begin();   //  This instruction in the setup function
…
h = dht1.readHumidity();  // read humidity and put in variable h
t = dht1.readTemperature(); // read temperature in celsius and put in variable t
```

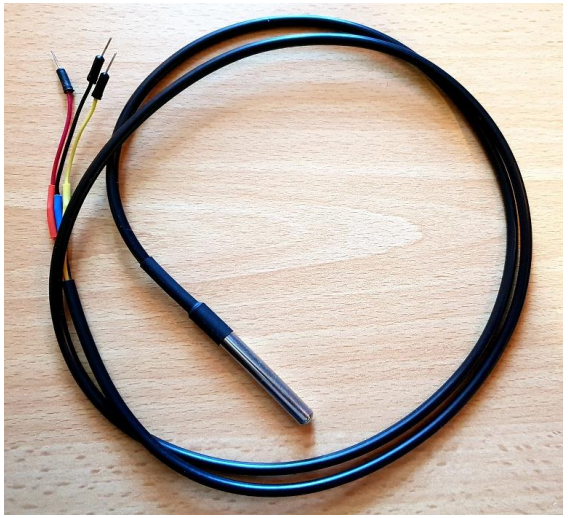
Two libraries must be installed in the Arduino IDE at first use:

DHT Sensor Library:
https://github.com/adafruit/DHT-sensor-library

Adafruit Unified Sensor Lib:
https://github.com/adafruit/Adafruit_Sensor

Temperature sensor DS18B20

The DS18B20 is a temperature sensor that communicates digitally with Arduino with a 1-wire interface (many sensors can be connected on the same bus). The temperature range covered varies from -55 to +125 °C with +/- 0.5 °C accuracy.



Put a 5k resistor between +5V (red wire) and data bus (yellow)

Software instructions for the use (example with 2 sensors):

```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2  // Pin bus sensors (all)
OneWire oneWire(ONE_WIRE_BUS); // Open communication
DallasTemperature sensors(&oneWire); // Setup
float Temp1;
float Temp2;
…
sensors.begin(); // Start sensors
…
sensors.requestTemperatures(); // Read temperatures from all sensors
Temp1 = sensors.getTempCByIndex(0);  // Temperature from sensor 1
Temp2 = sensors.getTempCByIndex(1);  // Temperature from sensor 2
```

Two libraries must be installed in the Arduino IDE at first use:

1-Wire bus:
http://www.pjrc.com/teensy/arduino_libraries/OneWire.zip

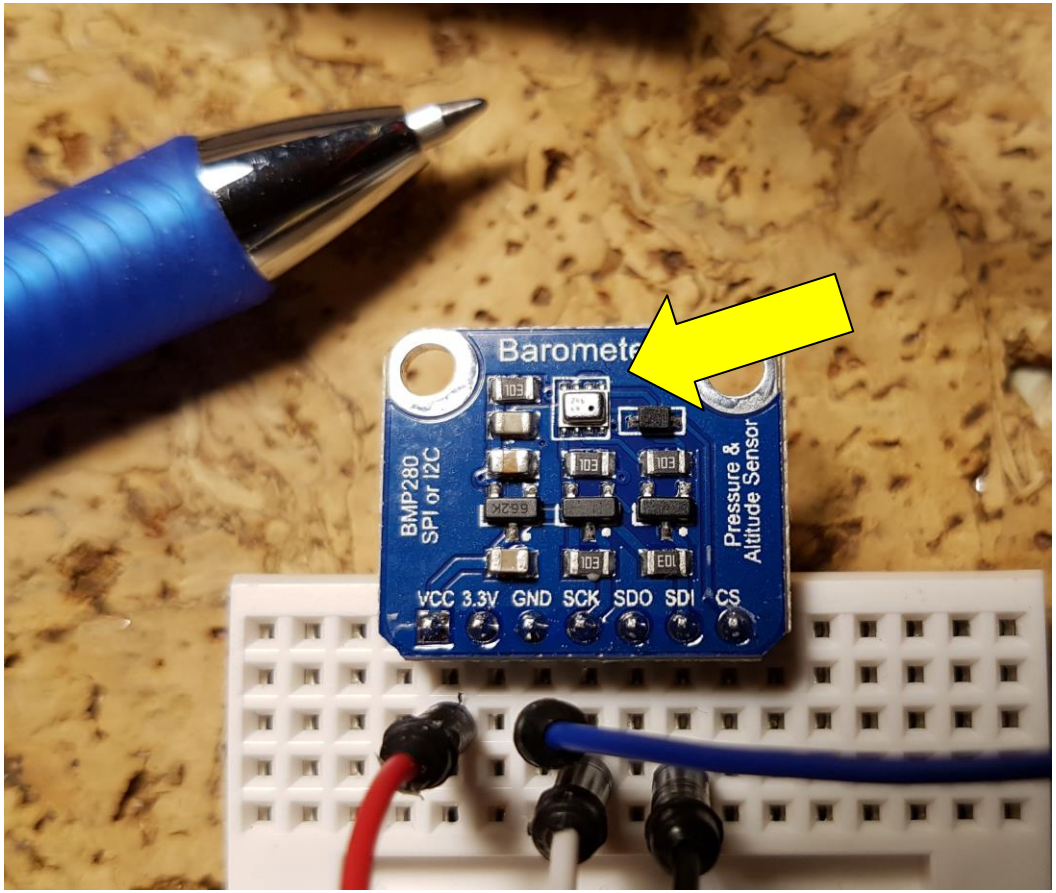
Dallas Temperature:
https://github.com/milesburton/Arduino-Temperature-Control-Library

Temperature and pressure sensor BMP280

BMP280. I²C and SPI interface

Pressure:
Range: 300-1100 hPa
Resolution: 0.16 Pa
Noise: 1.3 Pa

Temperature:
Range: -40 / +85 °C
Resolution: 0.01 °C



https://www.sunfounder.com/bmp280-barometric-pressure-temperature-altitude-sensor-module.html
https://www.bosch-sensortec.com/bst/products/all_products/bmp280

MEASUREMENTS WITH ARDUINO

_11_BMP280_TempPressAlt

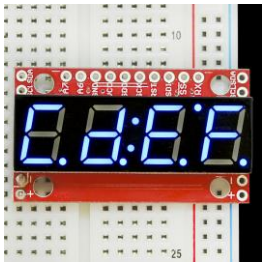
```
/***************************************************************************
  This is a library for the BMP280 humidity, temperature & pressure sensor

  Designed specifically to work with the Adafruit BMEP280 Breakout
  ----> http://www.adafruit.com/products/2651

  These sensors use I2C or SPI to communicate, 2 or 4 pins are required
  to interface.

  Adafruit invests time and resources providing this open source code,
  please support Adafruit andopen-source hardware by purchasing products
  from Adafruit!

  Written by Limor Fried & Kevin Townsend for Adafruit Industries.
  BSD license, all text above must be included in any redistribution
 ***************************************************************************/

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>

//Vin to 5V
//Gnd to Gnd
//SCK to SCL (21 on MEGA, A5 UNO)
//SDI to SDA (20 on MEGA, A4 UNO)

Adafruit_BMP280 bme; // I2C

void setup() {
  Serial.begin(9600);
  Serial.println(F("BMP280 test"));

  if (!bme.begin()) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
    while (1);
  }
}
```

```
void loop() {
    Serial.print(F("Temperature = "));
    Serial.print(bme.readTemperature());
    Serial.println(" *C");

    Serial.print(F("Pressure = "));
    Serial.print(bme.readPressure());
    Serial.println(" Pa");

    Serial.print(F("Approx altitude = "));
    Serial.print(bme.readAltitude(1013.25));
    Serial.println(" m");

    Serial.println();
    delay(1000);
}
```

COM16 (Arduino Mega or Mega 2560)

```
Pressure = 100952.34 Pa
Approx altitude = 31.18 m

Temperature = 26.11 *C
Pressure = 100952.83 Pa
Approx altitude = 31.03 m

Temperature = 26.11 *C
Pressure = 100956.31 Pa
Approx altitude = 30.96 m
```

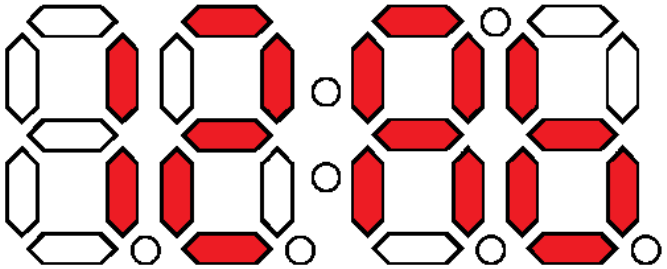
Serial 7-segment display



To avoid having to command every single segment and make it easier to use, this *shield* includes a microscopic Arduino that communicates with the main Arduino and receives commands on what to display. It can connect in 3 ways: **serial** (TTL), **SPI** serial or **I2C** serial.

Arduino Sample Snippet (Serial Mode): To make the display read *12Ab*., we can't be guaranteed that the cursor is at position 1. To ensure that it is, we can use the clear display command before sending our data.

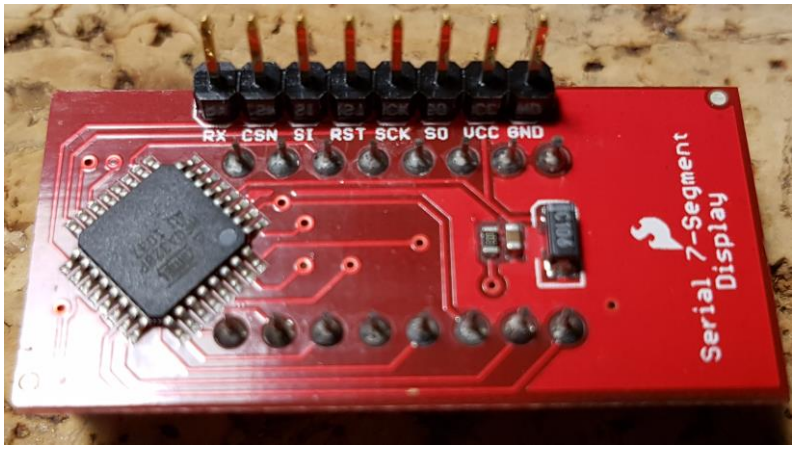
```
// ... after initializing Serial at the correct baud rate...
Serial.write(0x76);  // Clear display command, resets cursor
Serial.write(0x01);  // Hex value for 1, will display '1'
Serial.write('2');   // ASCII value for '2', will display '2'
Serial.write(0x0A);  // Hex value for 10, will display 'A'
Serial.write('B');   // ASCII value for 'B', will display 'b'
```



https://www.sparkfun.com/products/11442

Connection via **SPI**

Because we use the SPI library, you'll need to connect the Arduino's hardware SPI pins to the display.



Arduino Pin	Serial 7-Segment Display Pin
10 (CS)	SS (with a bar over it)
11 (MOSI)	SDI
13 (SCK)	SCK
5V	VCC
GND	GND

Connecting the displays *SDO* pin to MISO (12) on the Arduino is not required. Communication only goes one way - from Arduino (master) to display (slave).

SPI: Serial Peripheral Interface

4 wires bus: **MOSI** (Master Out Slave In), **MISO** (Master In Slave Out), **SCK** (Clock), **SS** (Slave Select, **SS1**, **SS2**, .., **SSn**)

https://www.sparkfun.com/products/11442

```
#include <SPI.h>

int csPin = 10; //You can use any IO pin but for this example we use 10

int cycles = 0;

void setup()
{
  pinMode(csPin, OUTPUT);
  digitalWrite(csPin, HIGH); //By default, don't be selecting OpenSegment

  Serial.begin(9600); //Start serial communication at 9600 for debug statements
  Serial.println("OpenSegment Example Code");

  SPI.begin(); //Start the SPI hardware
  SPI.setClockDivider(SPI_CLOCK_DIV64); //Slow down the master a bit

  //Send the reset command to the display - this forces the cursor to
  //return to the beginning of the display
  digitalWrite(csPin, LOW); //Drive the CS pin low to select OpenSegment
  SPI.transfer('v'); //Reset command
}
```

https://www.sparkfun.com/products/11442

```
void loop()
{
  cycles++; //Counting cycles! Yay!
  Serial.print("Cycle: ");
  Serial.println(cycles);

  spiSendValue(cycles); //Send the four characters to the display

  delay(1); //If we remove the slow debug statements, we need a very small delay to prevent flickering
}

//Given a number, spiSendValue chops up an integer into four values and sends them out over spi
void spiSendValue(int tempCycles)
{
  digitalWrite(csPin, LOW); //Drive the CS pin low to select OpenSegment

  SPI.transfer(tempCycles / 1000); //Send the left most digit
  tempCycles %= 1000; //Now remove the left most digit from the number we want to display
  SPI.transfer(tempCycles / 100);
  tempCycles %= 100;
  SPI.transfer(tempCycles / 10);
  tempCycles %= 10;
  SPI.transfer(tempCycles); //Send the right most digit

  digitalWrite(csPin, HIGH); //Release the CS pin to de-select OpenSegment
}
```

https://www.sparkfun.com/products/11442

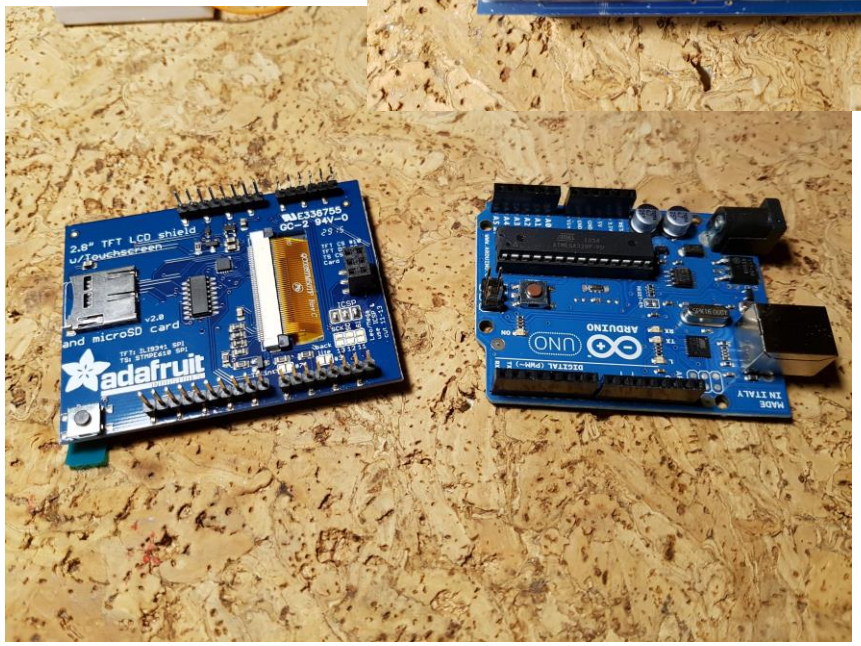
Color TFT graphic display (2.8”)

240 x 320 pixel, 16 or 18 bit color depth
Interfaced via SPI (TFT part) with Arduino:

SPI Clock: pin 13
SPI MISO: pin 12
SPI MOSI: pin 11
SPI CS (chip select): pin 10
SPI DS (data select): pin 9

The touch screen can be connected via I^2C and the microSD via SPI

Libraries to install:
Adafruit_ILI9341 and Adafruit_GFX



https://www.adafruit.com/product/1651

_06_TFT28Adafruit §

```
/***************************************************
  This is an example sketch for the Adafruit 2.2" SPI display.
  This library works with the Adafruit 2.2" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/1480

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/

#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9340.h"

// These are the pins used for the UNO
// for Due/Mega/Leonardo use the hardware SPI pins (which are different)
#define _sclk 13
#define _miso 12
#define _mosi 11
#define _cs 10
#define _dc 9
#define _rst 8
```

F(tells the compiler to store the string in the flash memory instead of internal RAM

SPI connection pin numbers

```
void setup() {
  Serial.begin(9600);
  while (!Serial);

  Serial.println("Adafruit 2.2\" SPI TFT Test!");

  tft.begin();

  Serial.println(F("Benchmark                Time (microseconds)"
  Serial.print(F("Screen fill              "));
  Serial.println(testFillScreen());
  delay(500);

  Serial.print(F("Text                     "));
  Serial.println(testText());
  delay(3000);

  Serial.print(F("Lines                    "));
  Serial.println(testLines(ILI9340_CYAN));
  delay(500);

  Serial.print(F("Horiz/Vert Lines         "));
  Serial.println(testFastLines(ILI9340_RED, ILI9340_BLUE));
  delay(500);

  Serial.print(F("Rectangles (outline)     "));
  Serial.println(testRects(ILI9340_GREEN));
  delay(500);

  Serial.print(F("Rectangles (filled)      "));
  Serial.println(testFilledRects(ILI9340_YELLOW, ILI9340_MAGENTA
```

https://www.sparkfun.com/products/11442

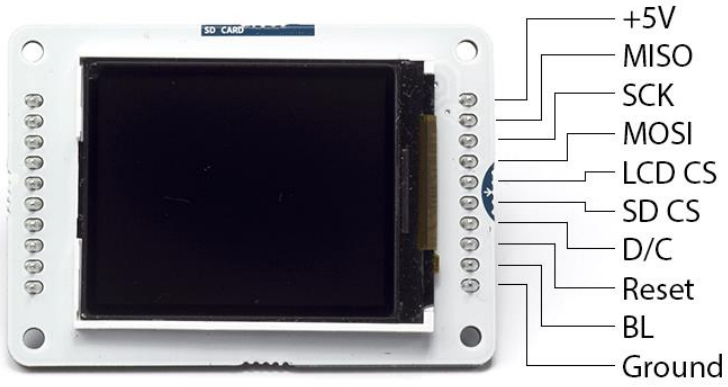
```
void loop(void) {
  for(uint8_t rotation=0; rotation<4; rotation++) {
    tft.setRotation(rotation);
    testText();
    delay(2000);
  }
}

unsigned long testText() {
  tft.fillScreen(ILI9340_BLACK);
  unsigned long start = micros();
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9340_WHITE);  tft.setTextSize(1);
  tft.println("Hello World!");
  tft.setTextColor(ILI9340_YELLOW); tft.setTextSize(2);
  tft.println(1234.56);
  tft.setTextColor(ILI9340_RED);    tft.setTextSize(3);
  tft.println(0xDEADBEEF, HEX);
  tft.println();
  tft.setTextColor(ILI9340_GREEN);
  tft.setTextSize(5);
  tft.println("Groop");
  tft.setTextSize(2);
  tft.println("I implore thee,");
  tft.setTextSize(1);
  tft.println("my foonting turlingdromes.");
  tft.println("And hooptiously drangle me");
  tft.println("with crinkly bindlewurdles,");
  tft.println("Or I will rend thee");
  tft.println("in the gobberwarts");
  tft.println("with my blurglecruncheon,");
  tft.println("see if I don't!");
  return micros() - start;
}
```

Functions to test graphic instructions

```
unsigned long testLines(uint16_t color) {
  unsigned long start, t;
  int           x1, y1, x2, y2,
                w = tft.width(),
                h = tft.height();

  tft.fillScreen(ILI9340_BLACK);

  x1 = y1 = 0;
  y2    = h - 1;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = w - 1;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);
  t     = micros() - start; // fillScreen doesn't count against ti

  tft.fillScreen(ILI9340_BLACK);

  x1    = w - 1;
  y1    = 0;
  y2    = h - 1;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = 0;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);
  t    += micros() - start;

  tft.fillScreen(ILI9340_BLACK);

  x1    = 0;
  y1    = h - 1;
  y2    = 0;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = w - 1;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);
  t    += micros() - start;
```

https://www.sparkfun.com/products/11442

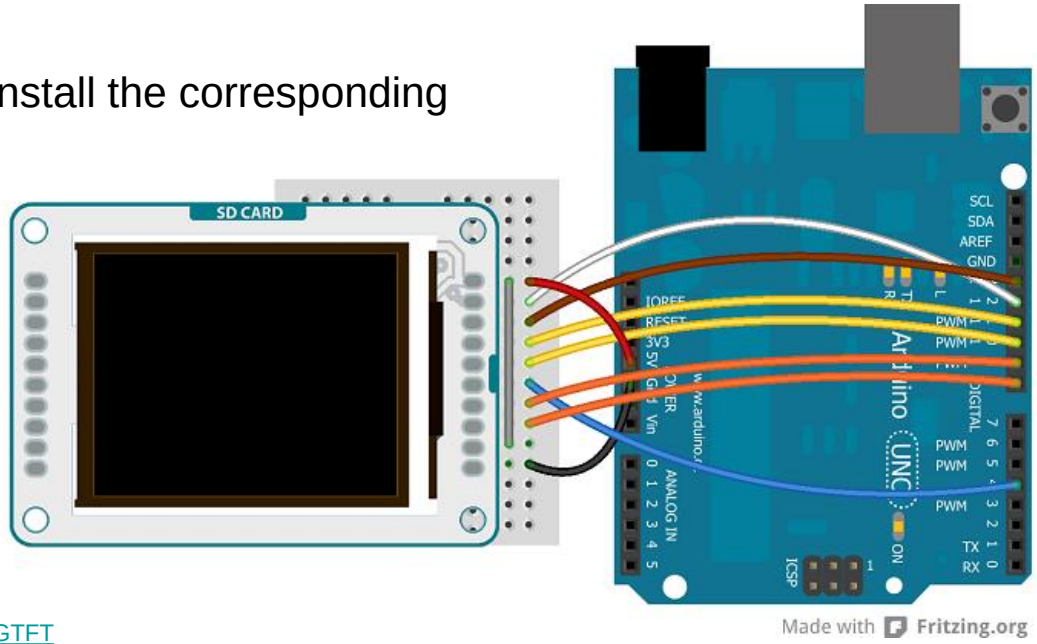
Connection of another TFT graphic display (1.8”)



160 x 128 pixel
Interfaced via SPI

Libraries to install:
Adafruit GFX and Adafruit ST7735

To use microSD, install the corresponding SD library



+5V:	+5V
MISO:	pin 12
SCK:	pin 13
MOSI:	pin 11
LCD CS:	pin 10
SD CS:	pin 4
D/C:	pin 9
RESET:	pin 8
BL:	+5V
GND:	GND

https://www.arduino.cc/en/Main/GTFT

_07_TFTPong_TFT18 §

```
This example for the Arduino screen reads the values
of 2 potentiometers to move a rectangular platform
on the x and y axes. The platform can intersect
with a ball causing it to bounce.

This example code is in the public domain.

Created by Tom Igoe December 2012
Modified 15 April 2013 by Scott Fitzgerald

http://arduino.cc/en/Tutorial/TFTPong

*/

#include <TFT.h>  // Arduino LCD library
#include <SPI.h>

// pin definition for the Uno
#define cs   10
#define dc   9
#define rst  8

TFT TFTscreen = TFT(cs, dc, rst);

// variables for the position of the ball and paddle
int paddleX = 0;
int paddleY = 0;
int oldPaddleX, oldPaddleY;
int ballDirectionX = 1;
int ballDirectionY = 1;

int ballSpeed = 10; // lower numbers are faster

int ballX, ballY, oldBallX, oldBallY;
```

```
void loop() {

  // save the width and height of the screen
  int myWidth = TFTscreen.width();
  int myHeight = TFTscreen.height();

  // map the paddle's location to the position of the potentiometers
  paddleX = map(analogRead(A0), 0, 1023, 0, myWidth) - 20 / 2;
  paddleY = map(analogRead(A1), 0, 1023, 0, myHeight) - 5 / 2;

  // set the fill color to black and erase the previous
  // position of the paddle if different from present
  TFTscreen.fill(0, 0, 0);

  if (oldPaddleX != paddleX || oldPaddleY != paddleY) {
    TFTscreen.rect(oldPaddleX, oldPaddleY, 20, 5);
  }

  // draw the paddle on screen, save the current position
  // as the previous.
  TFTscreen.fill(255, 255, 255);

  TFTscreen.rect(paddleX, paddleY, 20, 5);
  oldPaddleX = paddleX;
  oldPaddleY = paddleY;

  // update the ball's position and draw it on screen
  if (millis() % ballSpeed < 2) {
    moveBall();
  }
}
```

```
// this function determines the ball's position on
void moveBall() {
  // if the ball goes offscreen, reverse the direc
  if (ballX > TFTscreen.width() || ballX < 0) {
    ballDirectionX = -ballDirectionX;
  }
```

https://www.arduino.cc/en/Main/GTFT

_08_TFTGraph_TFT18

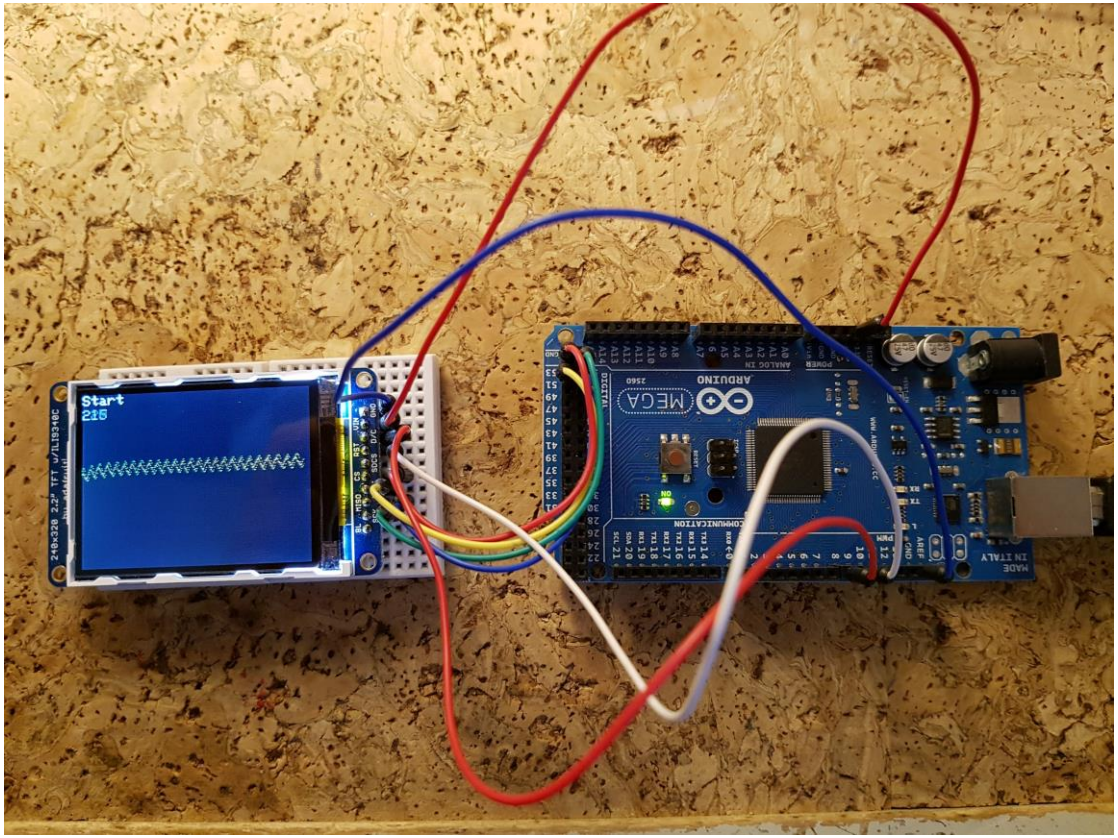
```
/*

 TFT Graph

 This example for an Arduino screen reads
 the value of an analog sensor on A0, and
 graphs the values on the screen.

 This example code is in the public domain.

 Created 15 April 2013 by Scott Fitzgerald

 http://arduino.cc/en/Tutorial/TFTGraph

 */

#include <TFT.h>  // Arduino LCD library
#include <SPI.h>

// pin definition for the Uno
#define cs   10
#define dc   9
#define rst  8

// pin definition for the Leonardo
// #define cs   7
// #define dc   0
// #define rst  1

TFT TFTscreen = TFT(cs, dc, rst);

// position of the line on screen
int xPos = 0;

void setup() {
  // initialize the serial port
  Serial.begin(9600);

  // initialize the display
  TFTscreen.begin();

  // clear the screen with a pretty color
  TFTscreen.background(250, 16, 200);
}

void loop() {
  // read the sensor and map it to the screen height
  int sensor = analogRead(A0);
  int drawHeight = map(sensor, 0, 1023, 0, TFTscreen.height());

  // print out the height to the serial monitor
  Serial.println(drawHeight);

  // draw a line in a nice color
  TFTscreen.stroke(250, 180, 10);
  TFTscreen.line(xPos, TFTscreen.height() - drawHeight,
                                    xPos, TFTscreen.height());

  // if the graph has reached the screen edge
  // erase the screen and start again
  if (xPos >= 160) {
    xPos = 0;
    TFTscreen.background(250, 16, 200);
  }
  else {
    // increment the horizontal position:
    xPos++;
  }

  delay(16);
}
```

https://www.arduino.cc/en/Main/GTFT

Connection of another TFT graphic display (2.2”) using Arduino MEGA2560

240 x 320 pixel, 16 bit color depth
Interfacing viaSPI to
Arduino MEGA2560:

SPI SCK (Clock): pin 52
SPI MISO: pin 50
SPI MOSI: pin 51
SPI CS (chip select): pin 53
SPI RST (reset): pin 9
SPI DC (data/command select): pin 8



Install libraries:
Adafruit_ILI9340 and Adafruit_GFX

https://www.adafruit.com/product/1480

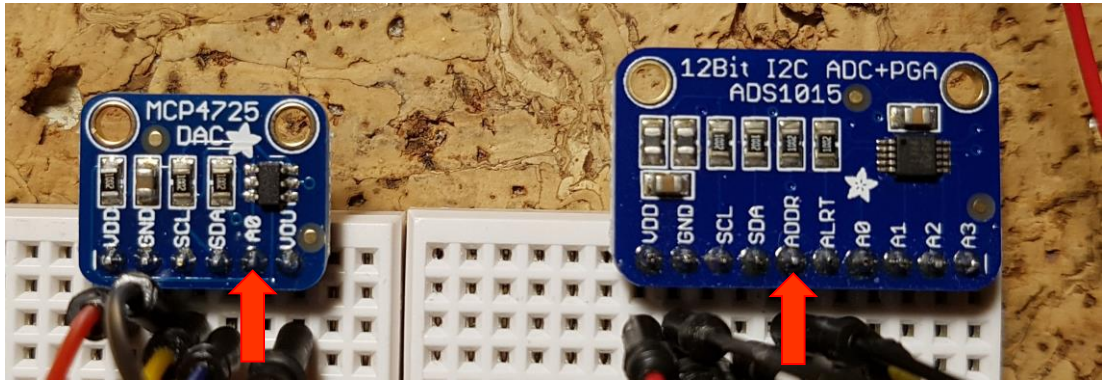
_09_MyScopeTFT22§

```
#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9340.h"

#if defined(__SAM3X8E__)
    #undef __FlashStringHelper::F(string_literal)
    #define F(string_literal) string_literal
#endif

// These are the pins used for the Mega
#define _sclk 52
#define _miso 50
#define _mosi 51
#define _cs 53
#define _rst 9
#define _dc 8

#define BLACK 0x0000
#define BLUE 0x001F
#define RED 0xF800
#define GREEN 0x07E0
#define CYAN 0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE 0xFFFF

Adafruit_ILI9340 tft = Adafruit_ILI9340(_cs, _dc, _rst);

// char array to print to the screen
char sensorPrintout[5];
int xPos = 1;
```

```
void setup() {
  // put your setup code here, to run once:

tft.begin();
tft.fillScreen(ILI9340_RED);
tft.fillScreen(ILI9340_BLACK);
tft.setRotation(1);
}

void loop() {
  // put your main code here, to run repeatedly:

 tft.setCursor(0, 0);
 tft.setTextColor(ILI9340_WHITE);  tft.setTextSize(2);
 tft.println("Start");

 String sensorVal1 = String(analogRead(A0));
 sensorVal1.toCharArray(sensorPrintout, 5);
 tft.setCursor(0, 20);
 tft.fillRect(0,20,50,28,ILI9340_BLACK);
 tft.println(sensorVal1);

// delay(100);

 tft.drawPixel(xPos, 50+analogRead(A0)/5,YELLOW); // draw a line across the screen
 tft.drawPixel(xPos, 50+analogRead(A1)/5,GREEN); // draw a line across the screen

  xPos = xPos + 1;
  if(xPos>=tft.width()) {
  xPos=0;
  tft.fillRect(0,50,tft.width(),tft.height(),ILI9340_BLACK);
 }
}
```

Oscilloscope of analog inputs A0 e A1

Let's add to the previous example an external DAC converter and an external ADC converter



Digital to Analog Converter

DAC **MCP4725** (12 bit)

https://learn.adafruit.com/mcp4725-12-bit-dac-tutorial

Analog to Digital Converter (x4)

ADC **ADS1015** (12 bit)

https://learn.adafruit.com/adafruit-4-channel-adc-breakouts

Both are connected to Arduino using **I²C** protocol, on different logic addresses.

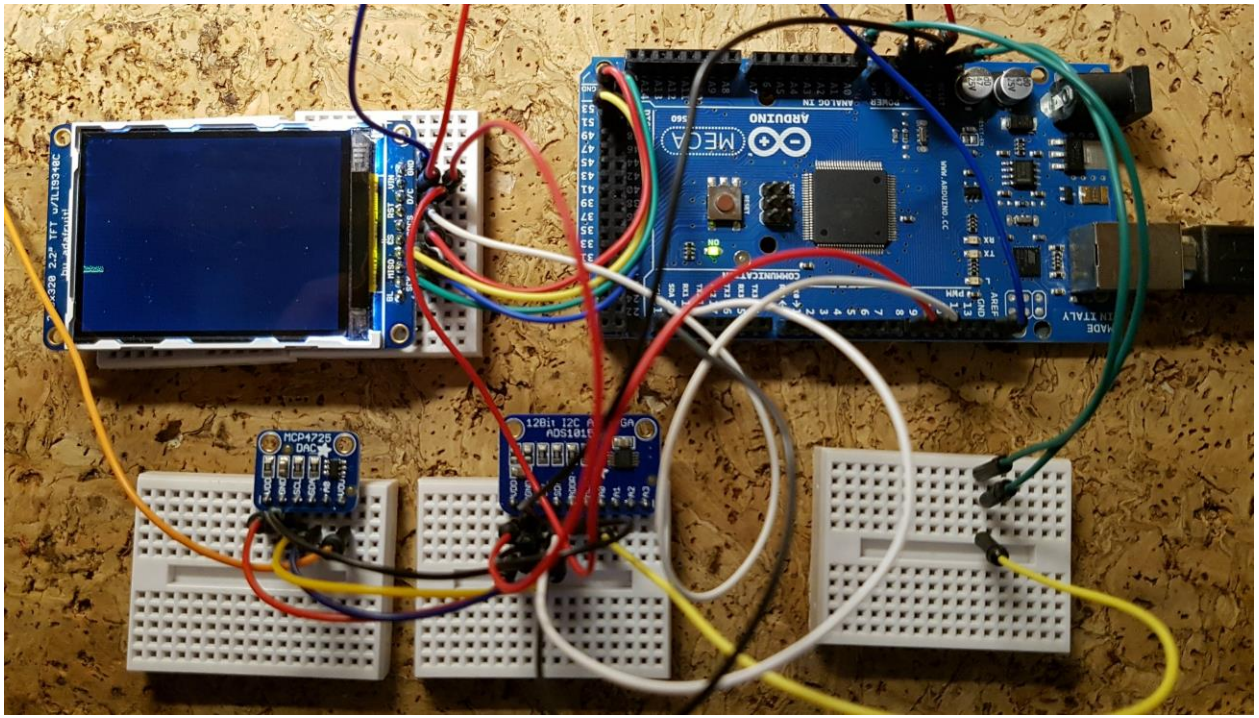
Using Arduino MEGA, the pins dedicated to **I²C** are 20 (SDA) and 21 (SCL); both devices are connected in parallel to this bus. The address is fixed in hardware mode on the respective card:

MCP4725: A0 unconnected → address 0x62; A0 connected to VDD → address 0x63

ADS1015: ADDR connected to GND → address 0x48; ADDR connected to VDD → address 0x49
ADDR connected to SDA → address 0x4A; ADDR connected to SCL → address 0x4B

HEXADECIMAL	↔	DECIMAL	↔	BINARY
0x4A	↔	74	↔	1001010

MEASUREMENTS WITH ARDUINO



We have now the TFT display connected on the SPI bus and the ADC and DAC connected both on the same I^{2}C bus but with different addresses. The loaded sketch reads the voltage on the analog input of Arduino on the pin A0 and draw it on the screen. In addition, the external ADC reads a voltage from its input A0, this value is drawn on the screen and the external DAC is set with the same voltage. The initial part of the sketch includes all the libraries necessary for the operation of the devices used. In the setup part these are initialized.

The **MCP4725** DAC address is set to 0x62 (its address selection pin is left disconnected), the **ADS1015** ADC address is set to 0x48 (its address selection pin is grounded). The SDAs of both go to pin 20 of the Arduino board and the SCLs to pin 21.

MEASUREMENTS WITH ARDUINO

_10_ADC_DAC_TFT22

```
#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9340.h"
#include <Wire.h>
#include <Adafruit_MCP4725.h>
#include <Adafruit_ADS1015.h>

Adafruit_MCP4725 dac;
Adafruit_ADS1015 ads1015(0x48);

// These are the pins used for the Mega
#define _sclk 52
#define _miso 50
#define _mosi 51
#define _cs 53
#define _rst 9
#define _dc 8

#define BLACK 0x0000
#define BLUE 0x001F
#define RED 0xF800
#define GREEN 0x07E0
#define CYAN 0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE 0xFFFF

Adafruit_ILI9340 tft = Adafruit_ILI9340(_cs, _dc, _rst);

// char array to print to the screen
char sensorPrintout[20];
int xPos = 1;
```

```
void setup() {
  // put your setup code here, to run once:

tft.begin();
tft.fillScreen(ILI9340_RED);
tft.fillScreen(ILI9340_BLACK);
tft.setRotation(1);
tft.setTextSize(1);

// DAC
dac.begin(0x62);
tft.println("DAC MCP4725 Started");

//    GAIN_TWOTHIRDS (for an input range of +/- 6.144V)
//    GAIN_ONE (for an input range of +/-4.096V)
//    GAIN_TWO (for an input range of +/-2.048V)
//    GAIN_FOUR (for an input range of +/-1.024V)
//    GAIN_EIGHT (for an input range of +/-0.512V)
//    GAIN_SIXTEEN (for an input range of +/-0.256V)

ads1015.begin();  // Initialize ads1115
ads1015.setGain(GAIN_ONE);
if (! ads1015.getGain()==GAIN_ONE)
      { tft.println("Error ADC ads1115"); }
      else { tft.println("ADC ADS1115 Started"); }

delay(1000);
tft.fillScreen(ILI9340_BLACK);
tft.println("Loop start");
delay(1000);
}
```

Setup TFT display

Setup DAC

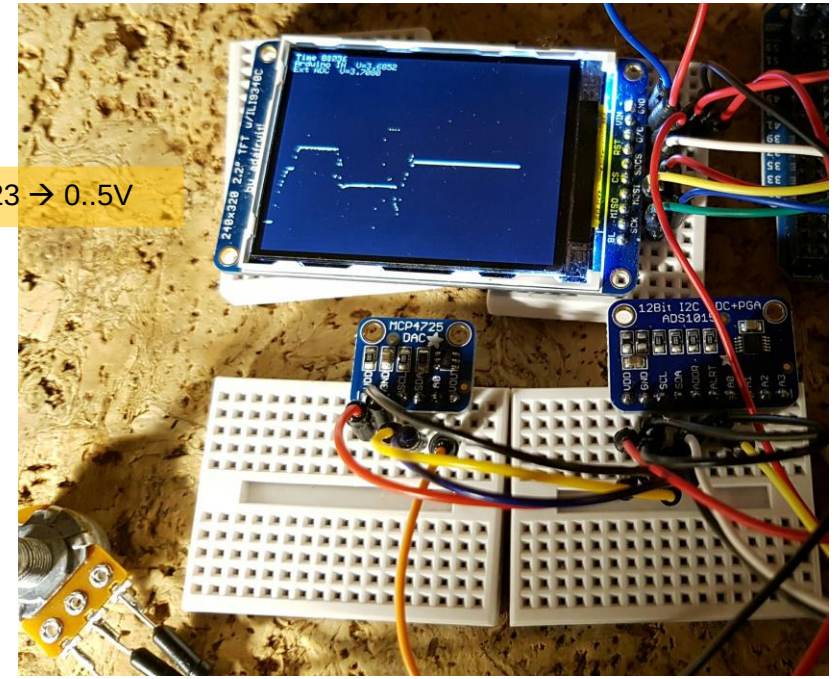
Setup ADC GAIN sets the range

MEASUREMENTS WITH ARDUINO


```
66  //-----------------------------------------------
67  //int16_t -32,768 to 32,767
68  int16_t adc0, adc1, adc2, adc3;
69  float VoltageExtADC,VoltageArdADC,VoltageExtDAC;
70  //-----------------------------------------------
71
72  void loop() {
73    // put your main code here, to run repeatedly:
74
75   tft.setCursor(0, 0);
76   tft.setTextColor(ILI9340_WHITE);
77
78   tft.print("Time ");
79   tft.println(millis());
80
81   VoltageArdADC=float(analogRead(A0))*5/1023; // 5V su 10 bit
82   String sensorV = String(VoltageArdADC,4); // Arduino ADC (10 bit)
83   sensorV.toCharArray(sensorPrintout, 20);
84   tft.print("Arduino IN  V=");
85   tft.println(sensorV);
86
87   tft.drawPixel(xPos, 240-VoltageArdADC*30,YELLOW); // plot ADC interno
88
89    xPos = xPos + 1;           // Avanzamento cursore plot
90    if(xPos>=tft.width()) {
91    xPos=0;
92    tft.fillRect(0,50,tft.width(),tft.height(),ILI9340_BLACK);
93   }
94
95  // ADC: 2047 per V=4.096 V
96   adc0 = ads1015.readADC_SingleEnded(0);
97
98  // DAC: 4095 per V=5 V;  2047 per V=2,5 V
99  // DAC: 4095 div 5 per V=1 V
100  dac.setVoltage(adc0*4.096/2.5, false); // V(DAC) = V(ADC)
101
102  VoltageExtADC=float(2*adc0)/1000; // per 1015
103  tft.drawPixel(xPos, 240-VoltageExtADC*30,WHITE); // plot ADC esterno
104
```

```
104
105   sensorV = String(VoltageExtADC,4); //(12 bit ADC)';
106   sensorV.toCharArray(sensorPrintout, 20);
107   tft.print("Ext ADC  V=");
108   tft.println(sensorV);
109   delay(200);
110   tft.fillRect(0,0,160,28,ILI9340_BLACK);
111 }
```

0..1023 → 0..5V



Reads ADC

Writes DAC

Max 4,096 V Input

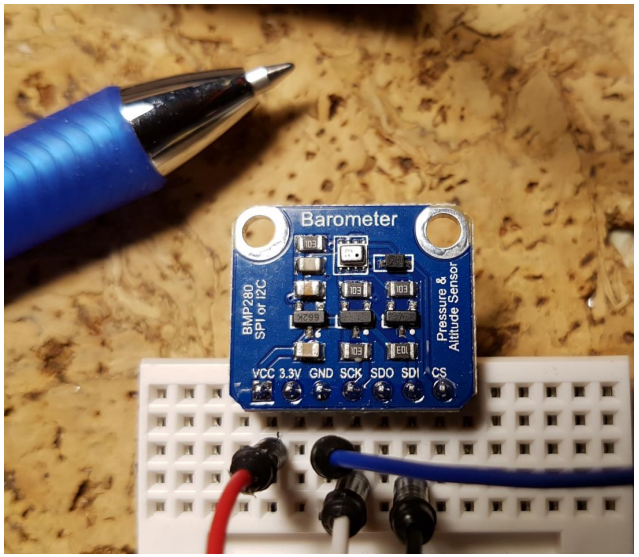
Potentiometer (divider) → external ADC
External ADC → internal DAC → internal ADC
Internal and external ADC plots are overlapped

Bidirectional data communication between Arduino and computer (via serial port)

http://www.lazarus-ide.org/

One of the many shields of Arduino allows to save data on microSD or other types of memories rewritable, but it can be useful to receive and process data in real time directly from Arduino. It is possible to do this in many ways, for example via WiFi or Ethernet shield or Bluetooth or GSM (the possibilities are endless). An option at no cost, without the use of additional shields consists in using the Arduino serial monitor port: on the computer runs a very simple ad-hoc software, which reads the data that Arduino sends to the serial port (in the form of text strings), converts into numerical values and uses them for processing or archiving on computers, in real time. Obviously also communication from computer to Arduino can be implemented, in a similar way. An example is now shown, made using the Lazarus compiler, i.e. free objects Pascal, free, open source (GPL/LGPL) and cross-platform (Windows, OSX, Linux).

MEASUREMENTS WITH ARDUINO



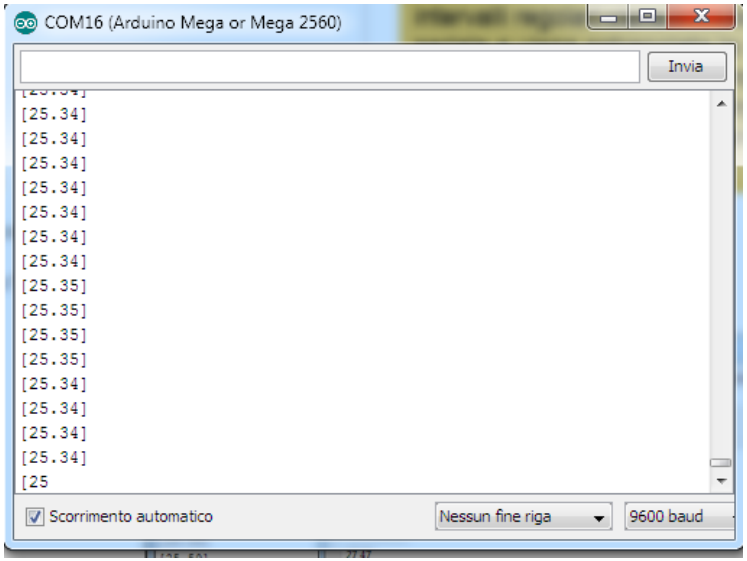
In this example the BMP280 is again used and the temperature value is sent to the computer through the serial port, in continuous way. To facilitate the interpretation of the data (which are sent in the form of a text string and in non-synchronized mode between Arduino and the computer), these will be formatted in the form [xx.yy] where xx.yy is the temperature, two decimal places. The computer program reads the serial data, extrapolates the string formed by the 7 characters [xx.yy], converts xx.yy to the form xx,yy (in Italy the numerical format provides the comma) and converts this string in numerical value, usable for processing or direct storage on the hard disk.
The sketch loaded on Arduino (MEGA) is a simplified version of the one previously seen for the BMP280 test.The program created with Lazarus to read the serial port includes the installation of the free 5dpoSerial library https://sourceforge.net/projects/sdpo-cl/files/ useful to manage the communication on serial port (virtual).

Both software sources are available for download.

MEASUREMENTS WITH ARDUINO

Modified version of the sketch from the previous example. On serial port only the temperature is written, adding the square brackets before and after the numerical value.

```
//Vin to 5V
//Gnd to Gnd
//SCK to SCL (21 on MEGA, A5 UNO)
//SDI to SDA (20 on MEGA, A4 UNO)

Adafruit_BMP280 bme; // I2C

void setup() {
  Serial.begin(9600);

  if (!bme.begin()) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
    while (1);
  }
}

String SerialRow;

void loop() {

 SerialRow = String()+ "["+bme.readTemperature()+"]";
 Serial.println(SerialRow);
 delay(10);

}
```

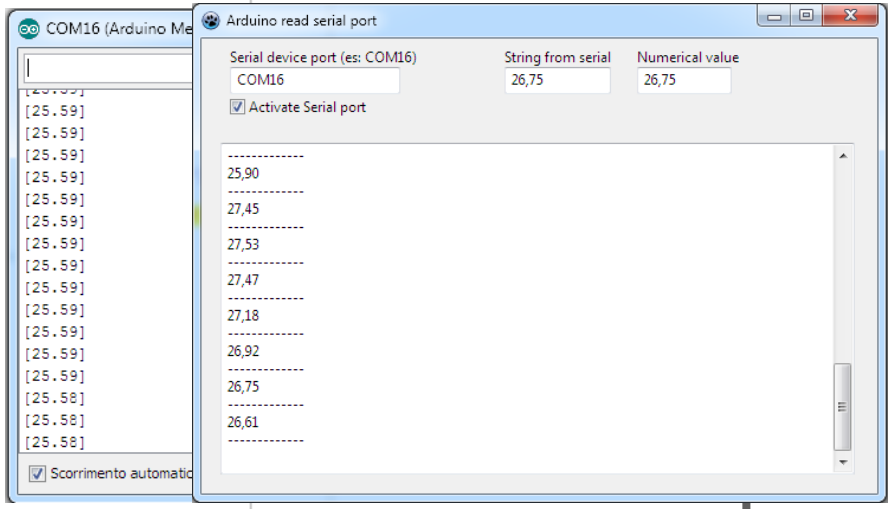


MEASUREMENTS WITH ARDUINO

Bidirectional data communication between Arduino and computer (via serial port)

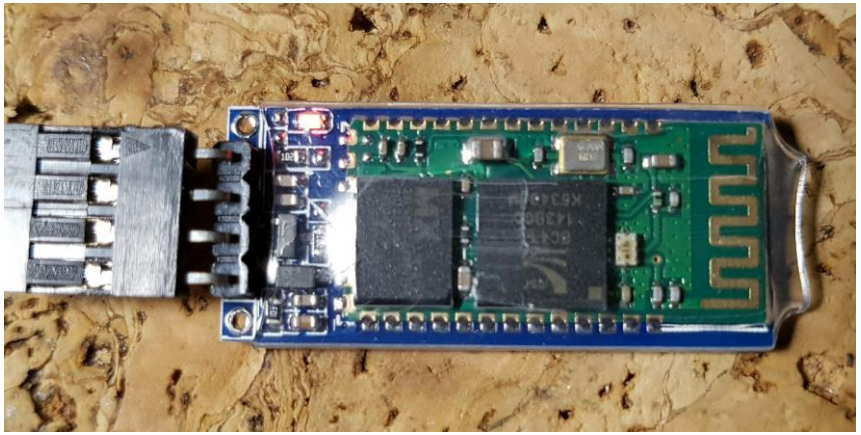

```
procedure TForm1.Timer1Timer(Sender: TObject);
var Data,First7chars,First5chars:String; NumericalValue:double; FlagError:boolean;
begin
  if not SdpoSerial1.Active then exit;

  Data:=SdpoSerial1.ReadData;

  if length(Data)>6 then
  begin
   First7chars:=copy(Data,1,7); // [24.57] expected

   if not ( (copy(First7chars,1,1)='[') or (copy(First7chars,7,1)=']') ) then exit;

   First5chars:=copy(First7chars,2,2)+','+copy(First7chars,5,2); // 24,57

   Memo1.Lines.Add(First5chars);
   Memo1.Lines.Add('-------------');
   Edit2.Text:=First5chars; // String
  end
   else exit;

  FlagError:=false;
  if length(First5chars)<>5 then exit;
  try NumericalValue:=StrToFLoat(First5chars);
    except On E:EConvertError do FlagError:=true; end;

  if not FlagError then Edit3.Text:=FloatToStr(NumericalValue)
                   else Edit3.Text:='N/A';

  sleep(150);
end;
```

Main routine of the Pascal program written with Lazarus. At regular intervals the serial port is read and the string containing the temperature value is extrapolated. The dot is converted into a comma and the string is converted into a numerical value.

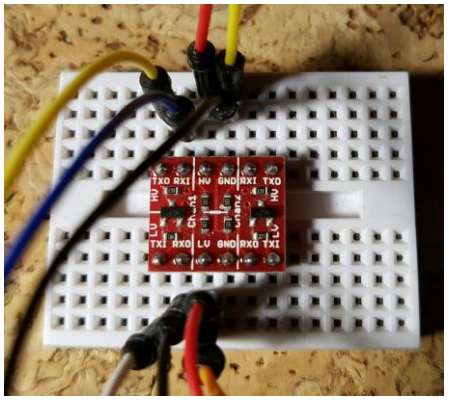


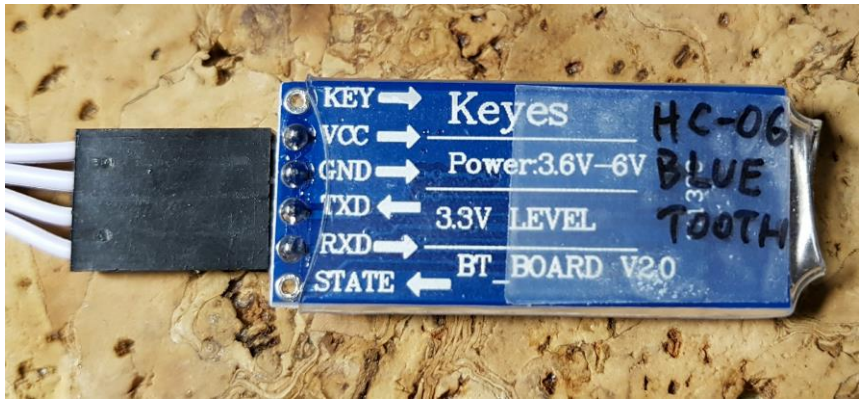
Data communication from smartphone to Arduino via Bluetooth

A shield called HC-06 is used, connected to Arduino MEGA. This card contains a transceiver Bluetooth and works at 3.3 V (even if it is indicated 3.6-6V). Communicates with Arduino through a serial port, so by a TX wire and an RX wire. These two signals follow the 3.3V CMOS standard so as not to damage the card a level shifter is required, to transform digital signals from 5V used on Arduino board (UNO or MEGA) to 3.3V and to transform the 3.3V signals from the shield to the 5V requested by Arduino.



https://www.sunfounder.com/bluetooth-transceiver-module-hc-06-rs232-4-pin-serial.html





Serial port 1 (of 4) of Arduino MEGA is used: pin 18 (TX1) and 19 (RX1). The TXD pin of the HC-06 card is therefore connected to the level shifter and then to Arduino's RX1 pin. The RXD pin of the HC-06 card is connected to the level shifter and after the conversion to the TX1 pin of Arduino.

The sketch uploaded on Arduino initializes the two serial ports used: **Serial** is the virtual serial port on computer that allows to display the serial monitor and **Serial1** is instead one of the 4 hardware serial ports of Arduino MEGA and in particular port 1, that uses pins 19 and 18.

In the main loop of the sketch, Arduino reads continuously what is received from the Serial1 port (HC-06) and copy it on the virtual serial port, to be displayed on the computer.

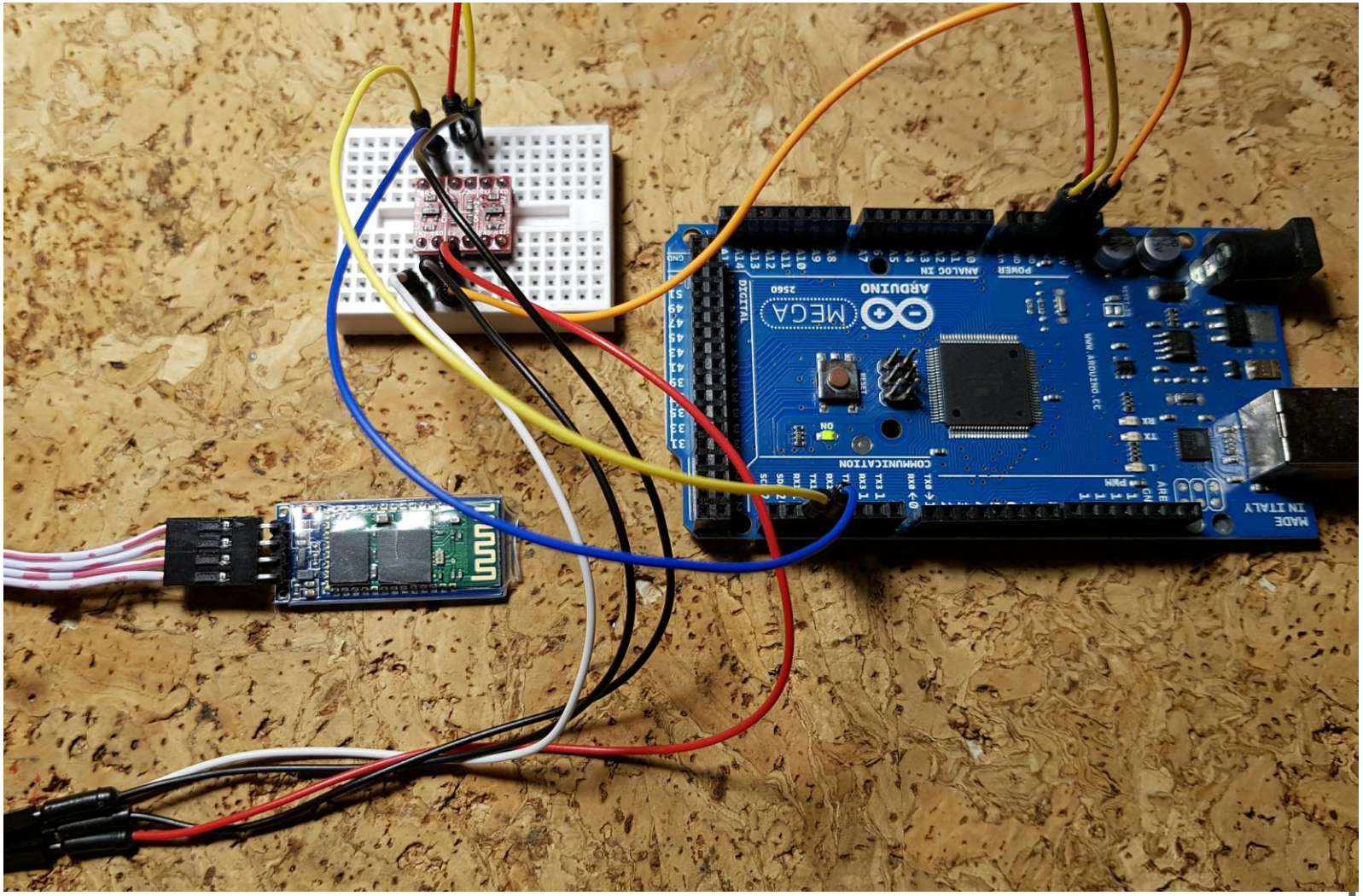
The strings are sent from the Bluetooth of a smartphone via a free app called ***Arduino Bluetooth Controller***. In this sketch the message received via Bluetooth is only displayed but the same operating scheme can be used to make Arduino perform remote actions (e.g. watering the lawn or turn on the house heating or turn on the light of a room), recognizing a certain command.

_13_HC06_Bluetooth

```
1  // Mega: Serial1, RX pin 19, TX pin 18
2  // su Android: "Arduino bluetooth controller", modalità Terminale
3
4  String message; //string that stores the incoming message
5
6  void setup()
7  {
8    Serial.begin(9600); //set baud rate (monitor su pc)
9    Serial1.begin(9600); //set baud rate (comunicazione con HC-06)
10 }
11
12 void loop()
13 {
14   while(Serial1.available())
15   {//while there is data available on the serial monitor
16     message+=char(Serial1.read());//store string from serial command
17   }
18   if(!Serial1.available())
19   {
20     if(message!="")
21     {//if data is available
22       Serial.println(message); //show the data
23       message=""; //clear the data
24     }
25   } else Serial.println("Serial 1 not available");
26   delay(1000); //delay
27 }
28
```

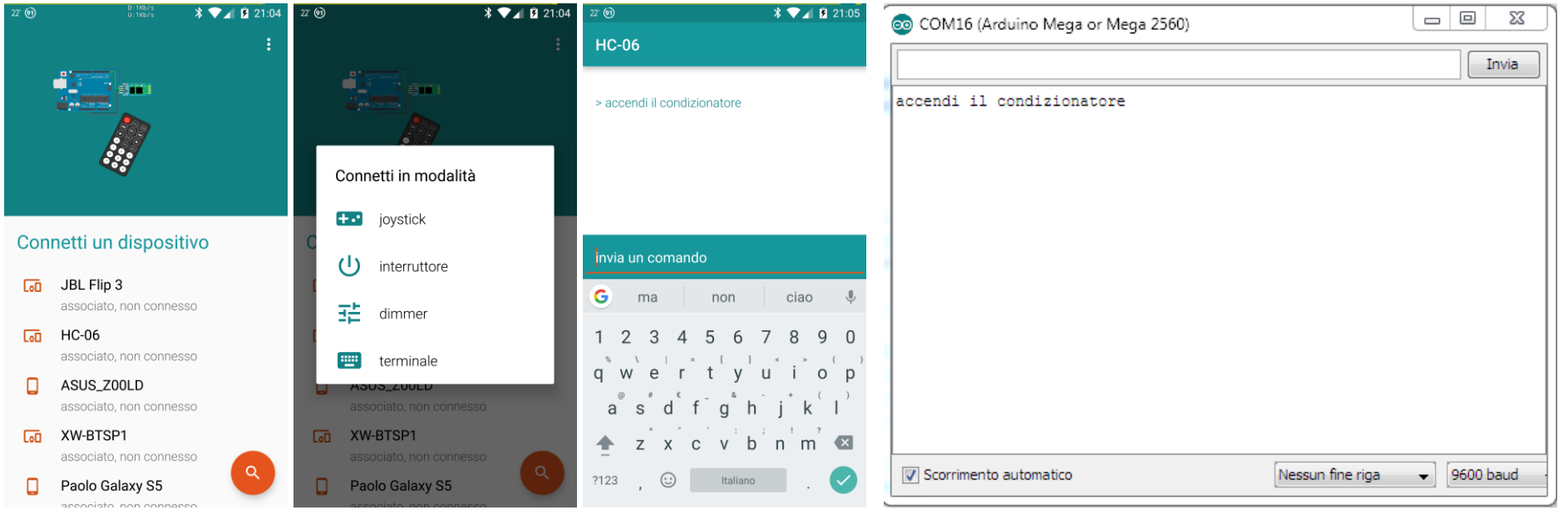
http://www.instructables.com/id/Add-bluetooth-to-your-Arduino-project-ArduinoHC-06/

MEASUREMENTS WITH ARDUINO



MEASUREMENTS WITH ARDUINO

Data communication from smartphone to Arduino via Bluetooth

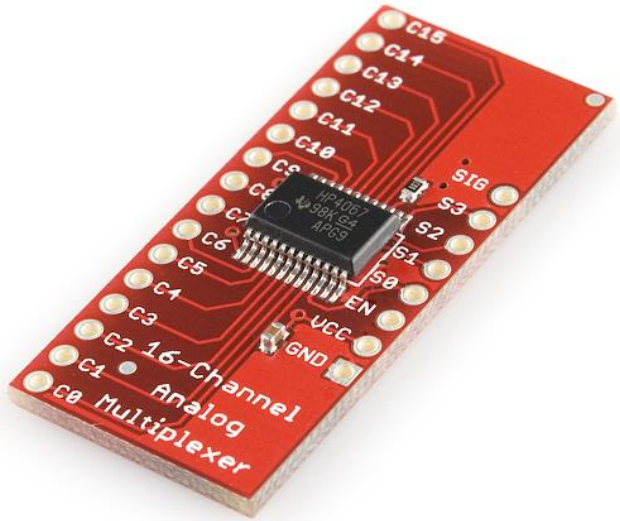


1. The HC-06 module will appear in the list of Bluetooth devices
2. With the Arduino Bluetooth Controller (Android) app, connect to HC-06 in "Terminal" mode
3. Once connected, the flashing LED of the HC-06 will remain steady on
4. In the Arduino IDE on the computer open the serial monitor
5. From the app terminal on the phone, type a sentence and send it via Bluetooth
6. In the serial monitor on the computer the sentence received from Arduino will appear

Using of a Multiplexer

It may sometimes be necessary to use a multiplexer, an electronic device that works in a similar way to a rotary selector, when for example it's required to connect sequentially a single ADC with several external analog sensors.

The shield used is a simple adapter of the **CD74HC4067** integrated circuit. It can be powered from 2V to 6V. Through the 4 digital inputs **S0..S3** can be selected which of the 16 pins **C0..C15** is connected to the (single) **SIG** pin (in bidirectional mode) via binary logic: the number in base 2 set to S0..S3 pins is converted in the decimal Cx number. The **EN** pin if connected at a HIGH logic level disables all connections (inverted logic). The use is very simple: 4 Arduino digital outputs are connected to the 4 selection pins S0..S3 and by using binary logic the desired electrical connection between SIG and C0..15 is established.

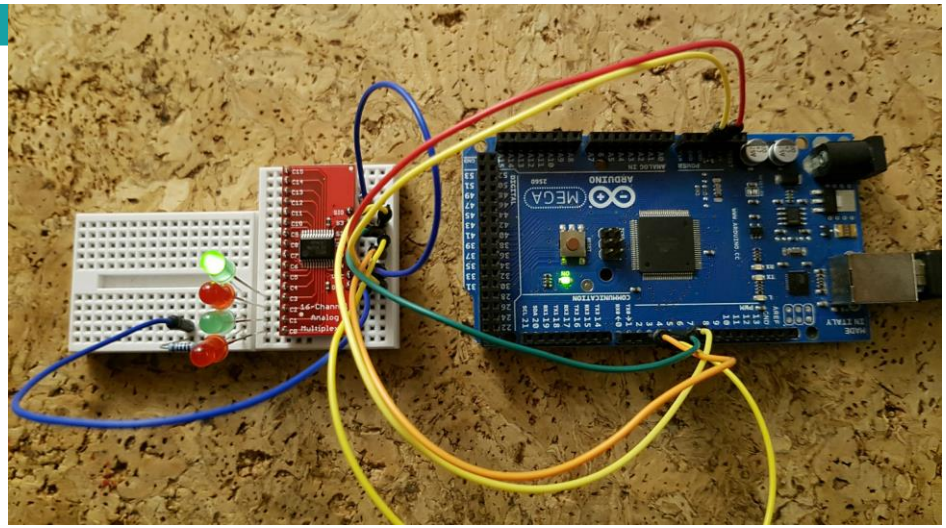


https://www.sparkfun.com/products/9056

MEASUREMENTS WITH ARDUINO

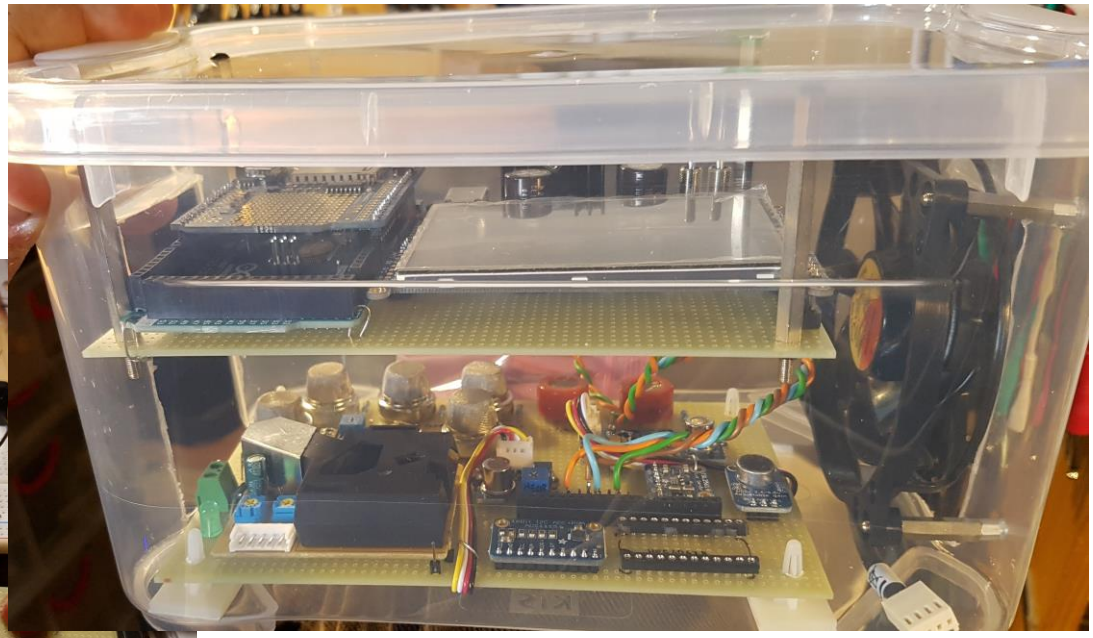
_14_Multiplexer §

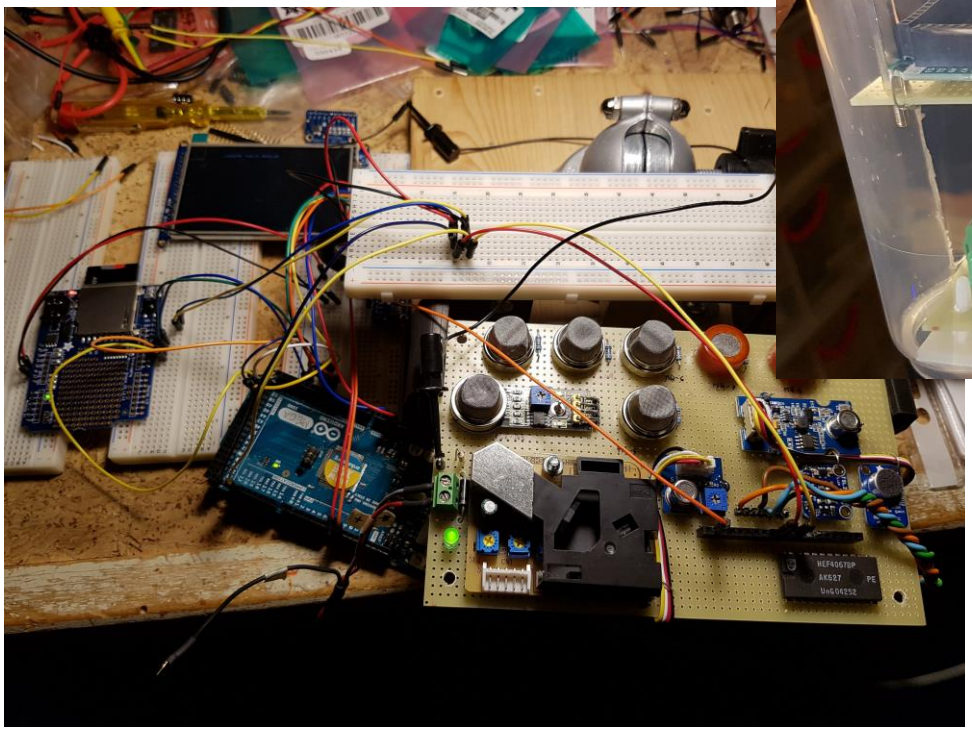
```
// address multiplexer
int A_zero =  2; // pin S0 a pin 2 di MEGA
int A_one =   3; // pin S1 a pin 3 di MEGA
int A_two =   4; // pin S2 a pin 4 di MEGA
int A_three = 5; // pin S3 a pin 5 di MEGA

void setup() {
// Multiplexer
  pinMode(A_zero, OUTPUT);      // sets the digital pin "A_zero" as output
  pinMode(A_one, OUTPUT);       // sets the digital pin "A_one" as output
  pinMode(A_two, OUTPUT);       // sets the digital pin "A_two" as output
  pinMode(A_three, OUTPUT);     // sets the digital pin "A_three" as output
}

void loop() {
// Select address 0000  =0
  digitalWrite(A_zero, LOW); digitalWrite(A_one,  LOW);
  digitalWrite(A_two,  LOW); digitalWrite(A_three,LOW);
  delay(1000);

// Select address 0001  =1
  digitalWrite(A_zero, HIGH); digitalWrite(A_one,  LOW);
  digitalWrite(A_two,  LOW);  digitalWrite(A_three,LOW);
  delay(1000);

// Select address 0010  =2
  digitalWrite(A_zero, LOW); digitalWrite(A_one, HIGH);
  digitalWrite(A_two, LOW);  digitalWrite(A_three, LOW);
  delay(1000);

// Select address 0011  =3
  digitalWrite(A_zero, HIGH); digitalWrite(A_one, HIGH);
  digitalWrite(A_two, LOW);   digitalWrite(A_three, LOW);
  delay(1000);
}
```

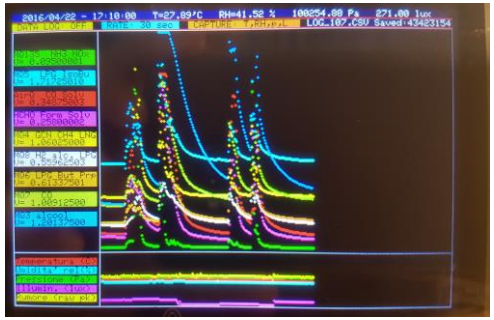


In this example, Arduino uses pins 2, 3, 4, 5 to control the S0..S3 multiplexer selector. 4 LEDs are connected (+) to the first 4 outputs of the multiplexer. The other pins of the 4 LEDs are all connected in common to a 1K resistor, connected to ground. The SIG multiplexer input is connected to 5V. The sketch enables the first 4 outputs in sequence, keeping them on for one second, then the 4 LEDs light up in sequence. Note that in the binary pin selection number, the rightmost digit is S0, the penultimate digit is S1, etc..

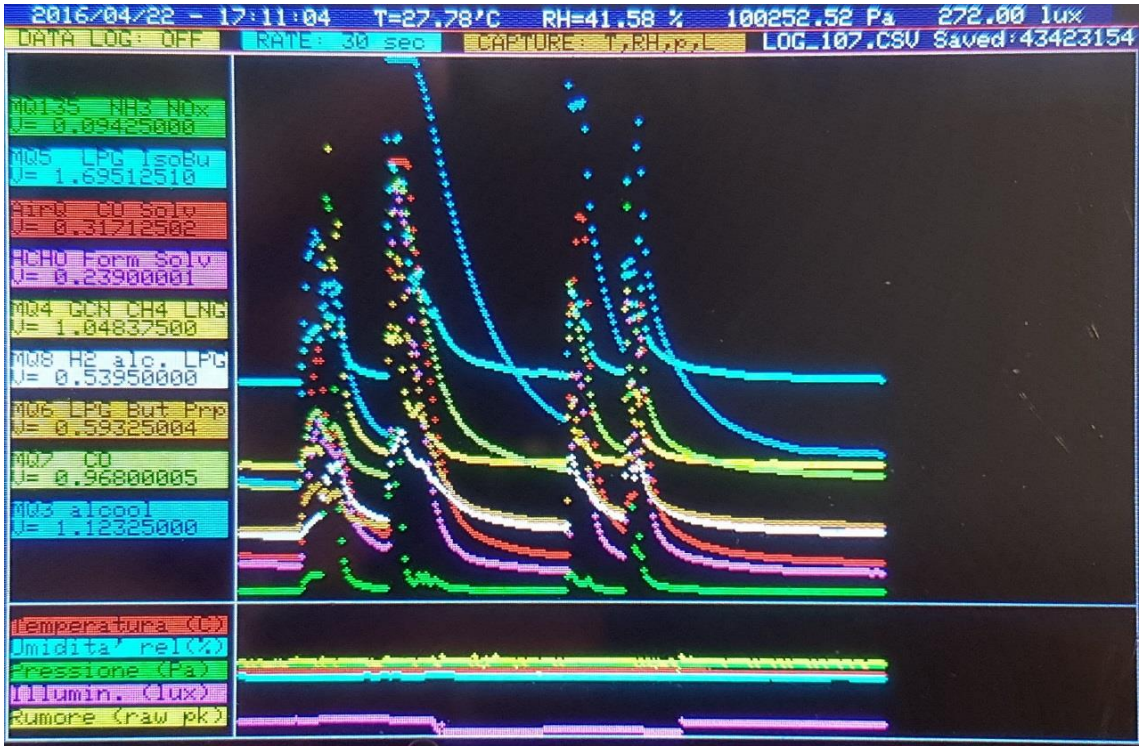
The "electronic nose" - Integrated multi-sensor platform based on Arduino





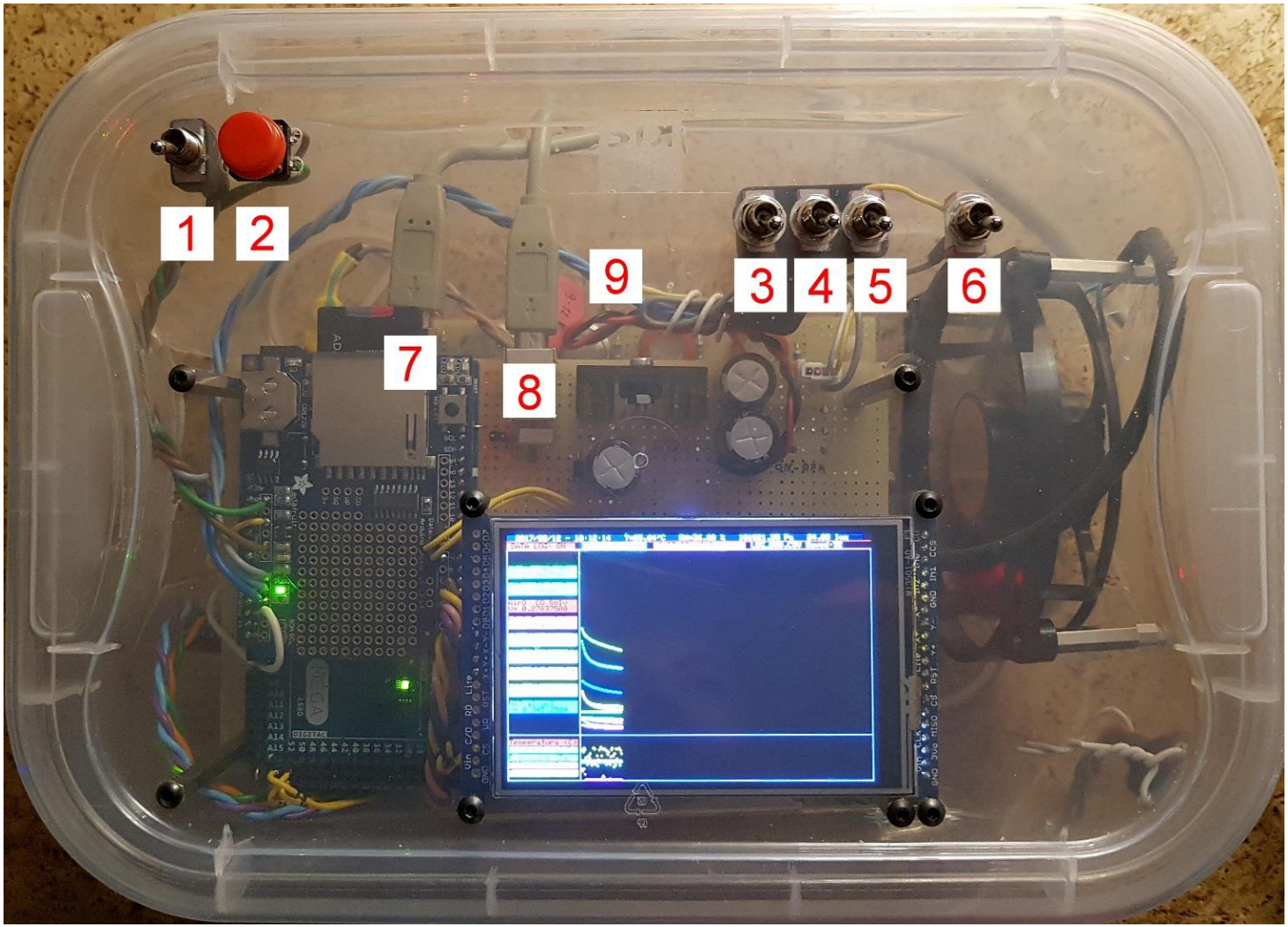


SENSORS USED:



TEMPERATURE (Celsius)
RELATIVE HUMIDITY (%)
PRESSURE (Pa)
Brightness (Lux)
MQ-3: Alcohol
MQ-4: CH4 methane, natural gas
MQ-5: LPG, natural gas
MQ-6: LPG, iso-butane, propane
MQ-7: CO
MQ-8: Hydrogen H2
MQ-135: Ammonia NH3, NOx, alcohol, benzene, fumes, CO2, etc.
AIR-Q (MP-503): Alcohol and fumes
HCHO (WSP2110): Organic gases, toluene, benzene, methanol

MEASUREMENTS WITH ARDUINO



The instrument can operate either through an external power supply (9V or 12V), to be connected to the red socket "9", or using a normal portable USB powerbank. In case of USB power supply, both sockets "7" and "8" must be connected to the powerbank (one powers Arduino and the other one powers the sensors board).

1: ON/OFF: (when using 9V or 12V power supply)

2: RESET button: to restart the measurement log on a new file (an incremental number filename is automatically assigned).

3: Data log ON/OFF: enables or disables data writing to file. Data writing can be enabled and disabled several times during the measurement, without the need to reset the instrument. Data will be appended to the current file when the option is enabled.

4: FAST / SLOW capture: Changes the data sampling period. When the mode is SLOW the numerical values of each sensor are written to the display, each cycle, when the mode is FAST only the graph is updated.

5: Select (a) sampling from all gas sensors plus Temperature, Relative Humidity, Pressure and Brightness, or (b) Temperature, Relative Humidity, Pressure and Brightness only. Only the saved data are shown in the graph.

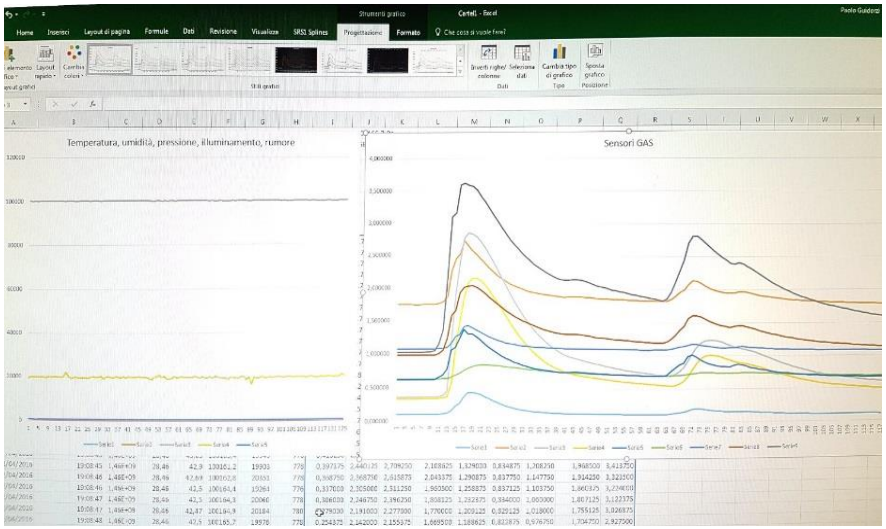
6: Turns ON or OFF the forced air circulation fan.

7: Arduino USB power supply and connection to PC (only for programming)

8: USB sensor power supply. Use 7 and 8 together to power from battery pack

9: Red power socket: 9V or 12V power supply

The measured data is saved on the SD card, in text format, which can be imported directly into Excel. When the software is switched on, it checks the correct functioning of the system components. In case "sensor error" is shown, the sensor card is probably missing power supply ("8") and is being powered via USB. In case of using socket "9" (not USB) both USB cables can be removed. The current size of the SD and the name of the log file are also shown when the power is turned on. Each time the power is turned on, the number associated with the data saving file name is increased by one, starting with the number of the last file saved previously. Empty the card to restart the numbering from scratch.



Note: the measured data of Temperature, Relative Humidity, Pressure and Brightness are represented by the correct value. The data from the gas sensors require calibration depending on the type of sensor and also on the current temperature and humidity, following the specifications in the datasheets of the individual sensors.

_15_Naso §

```
/*************************************************
DANTE 1.0
*************************************************/

#include <SPI.h>
#include "Adafruit_GFX.h"
#include "Adafruit_HX8357.h"
#include <Wire.h>
#include <Adafruit_ADS1015.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#include "RTClib.h"
#include <SD.h>
#include <Adafruit_TSL2561_U.h>

RTC_DS1307 rtc;

Adafruit_ADS1115 ads1115(0x48); // Construct an ads1115 at the default address: 0x48

// Pin del TFT
#define TFT_CS 9
#define TFT_DC 8
#define TFT_RST 7 // RST can be set to -1 if you tie it to Arduino's reset
Adafruit_HX8357 tft = Adafruit_HX8357(TFT_CS, TFT_DC, TFT_RST);

// SD
Sd2Card card;
SdVolume volume;
SdFile root;

#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme; // I2C

```

#include of libraries for the used digital sensors.

Initialization of sensors and devices (ADC, Graphic display,…).

```
34
35  // char array to print to the screen
36  char sensorPrintout[20];
37
38  float Voltage;
39  int16_t adc0,adc1;
40
41  #define black           0x0000      /*   0,   0,   0 */
42  #define navy            0x000F      /*   0,   0, 128 */
43  #define darkgreen       0x03E0      /*   0, 128,   0 */
44  #define darkcyan        0x03EF      /*   0, 128, 128 */
45  #define maroon          0x7800      /* 128,   0,   0 */
46  #define purple          0x780F      /* 128,   0, 128 */
47  #define olive           0x7BE0      /* 128, 128,   0 */
48  #define lightgrey       0xC618      /* 192, 192, 192 */
49  #define darkgrey        0x7BEF      /* 128, 128, 128 */
50  #define blue            0x051F      /*   0,   0, 255 */
51  #define green           0x07E0      /*   0, 255,   0 */
52  #define cyan            0x07FF      /*   0, 255, 255 */
53  #define red             0xF800      /* 255,   0,   0 */
54  #define magenta         0xF81F      /* 255,   0, 255 */
55  #define yellow          0xFFE0      /* 255, 255,   0 */
56  #define white           0xFFFF      /* 255, 255, 255 */
57  #define orange          0xFD20      /* 255, 165,   0 */
58  #define greenyellow     0xAFE5      /* 173, 255,  47 */
59  #define pink                        0xF81F
60
61  const int chipSelect = 10;
62  File logfile;
63
64  // address multiplexer
65  int A_zero =  2;
66  int A_one =   3;
67  int A_two =   4;
68  int A_three = 5;
69
70  Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_LOW, 12345);
```

```
71
72  int InitialDelays=200;
73  int scendigiu=18;
74
75  char filename[] = "LOG_000.CSV";
76
77  String sensorVal1,oldsensorVal1="";
78  String Orologio;
79  String Anno,Mese,Giorno,Ora,Minuto,Secondo,Unix;
80  int Switch0,Switch1,Switch2;
81
```

Definition of variables and constants

```
83  void setup() {
84
85    tft.begin(HX8357D);
86    tft.setRotation(1);
87    tft.fillScreen(black);
88    tft.setCursor(0, 0);
89
90    tft.setTextColor(greenyellow);
91    tft.setTextSize(1);
92    tft.println("DANTE - Electronic multisensorial acquisition device. Vr 1.0");
93    tft.println("Designed and engineered by Paolo Guidorzi");
94    tft.println("paolo.guidorzi@unibo.it");
95    tft.println();
96
97    tft.setTextColor(white);
98    tft.println("---- SYSTEM CHECK ----");
99
100   tft.println("OK TFT display!");
101   delay(InitialDelays);
102
103   String oldsensorVall="";
104
105 // Initialize adsl115 ADC
106     adsl115.begin();
107     adsl115.setGain(GAIN_ONE);
108
109   if (! adsl115.getGain()==GAIN_ONE) { tft.println("Error ADC adsl115"); }
110   else { tft.println("OK adsl115 (A/D Converter)!"); }
111   delay(InitialDelays);
112
113 // Sensore Luminosita'
114   /* Initialise the sensor */
115   if(!tsl.begin()) { tft.print("Could not find a valid TSL2561 sensor!");}
116   else { tft.println("OK TSL2561 (Light sensor)!");   }
117   delay(InitialDelays);
```

Setup: sensors and variables initialization

```
  /* You can also manually set the gain or enable auto-gain support */
  // tsl.setGain(TSL2561_GAIN_1X);      /* No gain ... use in bright light to avoid sensor saturation */
  // tsl.setGain(TSL2561_GAIN_16X);     /* 16x gain ... use in low light to boost sensitivity */
  tsl.enableAutoRange(true);            /* Auto-gain ... switches automatically between 1x and 16x */

  /* Changing the integration time gives you better sensor resolution (402ms = 16-bit data) */
  // tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);      /* fast but low resolution */
   tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS);  /* medium resolution and speed   */
  // tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);  /* 16-bit data but slowest conversions */

// RTC (orologio)
  if (! rtc.begin()) { tft.println("Couldn't find RTC!"); }
  else { tft.println("OK RTC (Real Time clock)!");   }
  delay(InitialDelays);

// REGOLA OROLOGIO - tenere disattivato se non per regolare
//rtc.adjust(DateTime(__DATE__, __TIME__));

// Sensore p, T, RH
  if (!bme.begin()) {
    tft.println("Could not find a valid BME280 sensor!");

    tft.println("");
    tft.setTextSize(2);
    tft.setTextColor(red,yellow);
    tft.println("                                    ");
    tft.println(" --- WARNING: SENSOR BOARD FAILURE --- ");
    tft.println("                                    ");
    delay(5000);
    tft.setTextSize(1);
    tft.setTextColor(white);
   }
  else { tft.println("OK BME280 (p, T, RH) sensor!"); }
  delay(InitialDelays);
```

Setup: sensors and variables initialization

```
167   int Switch_0= analogRead(A3);
168   tft.println(Switch_0);
169   tft.print("Unix time: ");
170   DateTime now = rtc.now();
171   tft.println(now.unixtime());
172
173 // SD
174   tft.print("\nInitializing SD card...");
175   pinMode(53, OUTPUT);
176   delay(InitialDelays);
177
178   if (!card.init(SPI_HALF_SPEED, 10, 11, 12, 13)) {
179     tft.println("initialization failed. Things to check:");
180     tft.println("* is a card is inserted?");
181   }
182   delay(InitialDelays);
183
184   // print the type of card
185   tft.print("\nCard type: ");
186   switch(card.type()) {
187     case SD_CARD_TYPE_SD1:
188       tft.println("SD1");
189       break;
190     case SD_CARD_TYPE_SD2:
191       tft.println("SD2");
192       break;
193     case SD_CARD_TYPE_SDHC:
194       tft.println("SDHC");
195       break;
196     default:
197       tft.println("Unknown");
198   }
199   delay(InitialDelays);
```

Setup: sensors and variables initialization

```
201  // Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
202  if (!volume.init(card)) {
203    tft.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");
204    tft.println("You cannot run the software without SD Card");
205    if (Switch_0>500) {while(1) { };}
206  }
207
208  // print the type and size of the first FAT-type volume
209  uint32_t volumesize;
210  tft.print("\nVolume type is FAT");
211  tft.println(volume.fatType(), DEC);
212  tft.println();
213
214  volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
215  volumesize *= volume.clusterCount();       // we'll have a lot of clusters
216  volumesize *= 512;                            // SD card blocks are always 512 bytes
217  volumesize /= 1024;
218  tft.print("Volume size (Mbytes): ");
219  volumesize /= 1024;
220  tft.println(volumesize);
221
222  root.openRoot(volume);
223
224  // list all files in the card with date and size
225  // root.ls(LS_R | LS_DATE | LS_SIZE);
226
227  // see if the card is present and can be initialized:
228  if (!SD.begin(10, 11, 12, 13)) {
229    tft.println("Card failed, or not present");
230    // don't do anything more:
231    //return;
232  }
233  tft.println("card initialized.");
```

Setup: sensors and variables initialization

```
235   // crea il nome file LOG_XYZ.CSV
236   for (uint8_t i = 0; i < 1000; i++) {
237     sprintf(filename, "LOG_%03d.CSV", i);
238     tft.print(".");
239     if (! SD.exists(filename)) {   // appena non esiste un file con numero XYZ, lo genera
240       // only open a new file if it doesn't exist
241       logfile = SD.open(filename, FILE_WRITE);
242       break;  // leave the loop!
243     }
244   }
245     tft.println("");
246     tft.print("filename: "); tft.println(filename);
247     delay(InitialDelays);
248
249   if (! logfile) {
250     tft.println("couldnt create file");
251   }
252
253   tft.print("Logging to: ");
254   tft.println(filename);
255   delay(1000);
256
257   Anno=String(now.year());
258   Mese=String(now.month()); if (Mese.toInt()<10)    { Mese="0"+Mese;}
259   Giorno=String(now.day()); if (Giorno.toInt()<10) { Giorno="0"+Giorno;}
260   Ora=String(now.hour()); if (Ora.toInt()<10) { Ora="0"+Ora;}
261   Minuto=String(now.minute()); if (Minuto.toInt()<10) { Minuto="0"+Minuto;}
262   Secondo=String(now.second()); if (Secondo.toInt()<10) { Secondo="0"+Secondo;}
263   Unix=String(now.unixtime());
264   Orologio=Anno+"/"+Mese+"/"+Giorno+" "+Ora+":"+Minuto+":"+Secondo+" "+Unix;
265   sensorVall = String()+ "  T= "+bme.readTemperature()+" 'C  RH= "+bme.readHumidity()+" %  "+bme.readPressure()+" Pa ";
266   Orologio=Orologio+sensorVall;
267
268   logfile.println(Orologio);
269   logfile.println("YYYY/MM/DD HH:MM:SS UNIXTIME## TT.00 RH.00 PRESSI.00 NOISE LUX.00 0.SENSOR01 0.SENSOR02 0.SENSOR03 0.SENSOR
270
271   logfile.flush();
```

Name assignment of the LOG file, with filename progressive numbering

Setup: creation of the LOG file and saving of the first line. The **"flush()"** command forces the writing, thus creating the file

```
// Multiplexer
  pinMode(A_zero, OUTPUT);      // sets the digital pin "A_zero" as output
  pinMode(A_one, OUTPUT);       // sets the digital pin "A_one" as output
  pinMode(A_two, OUTPUT);       // sets the digital pin "A_two" as output
  pinMode(A_three, OUTPUT);     // sets the digital pin "A_three" as output

  delay(3*InitialDelays);

  tft.setTextSize(2);
  tft.println("");
  tft.setTextColor(pink);
  tft.print("Starting acquisition");
  tft.setTextColor(white);
  delay(2*InitialDelays);
  tft.setTextSize(1);

// preparazione schermata di acquisizione
  tft.fillScreen(black);  // pulisce schermo
  tft.setTextWrap(false);

  tft.drawLine(0,8,480,8,red);
  tft.drawLine(0,18,480,18,red);

  tft.drawLine(0,19,480,19,lightgrey);
  tft.drawLine(0,260,480,260,lightgrey);
  tft.drawLine(95,19,95,319,lightgrey);
  tft.drawLine(0,19,0,319,lightgrey);
  tft.drawLine(0,319,479,319,lightgrey);
  tft.drawLine(478,19,478,319,lightgrey);

// Sensore 1
 tft.setCursor(2, scendigiu+22);
 tft.setTextColor(black,green);
 tft.print("MQ135  NH3 NOx ");
 tft.setCursor(2, scendigiu+22+8*1);
 tft.print("Val            ");
```

```
// Sensore 2
 tft.setCursor(2, scendigiu+6+22+8*2);
 tft.setTextColor(black,cyan);
 tft.print("MQ5  LPG IsoBu ");
 tft.setCursor(2, scendigiu+6+22+8*3);
 tft.print("Val            ");

// Sensore 3
 tft.setCursor(2, scendigiu+12+22+8*4);
 tft.setTextColor(black,red);
 tft.print("AirQ  CO Solv  ");
 tft.setCursor(2, scendigiu+12+22+8*5);
 tft.print("Val            ");

// Sensore 4
 tft.setCursor(2, scendigiu+18+22+8*6);
 tft.setTextColor(black,magenta);
 tft.print("HCHO Form Solv ");
 tft.setCursor(2, scendigiu+18+22+8*7);
 tft.print("Val            ");
```

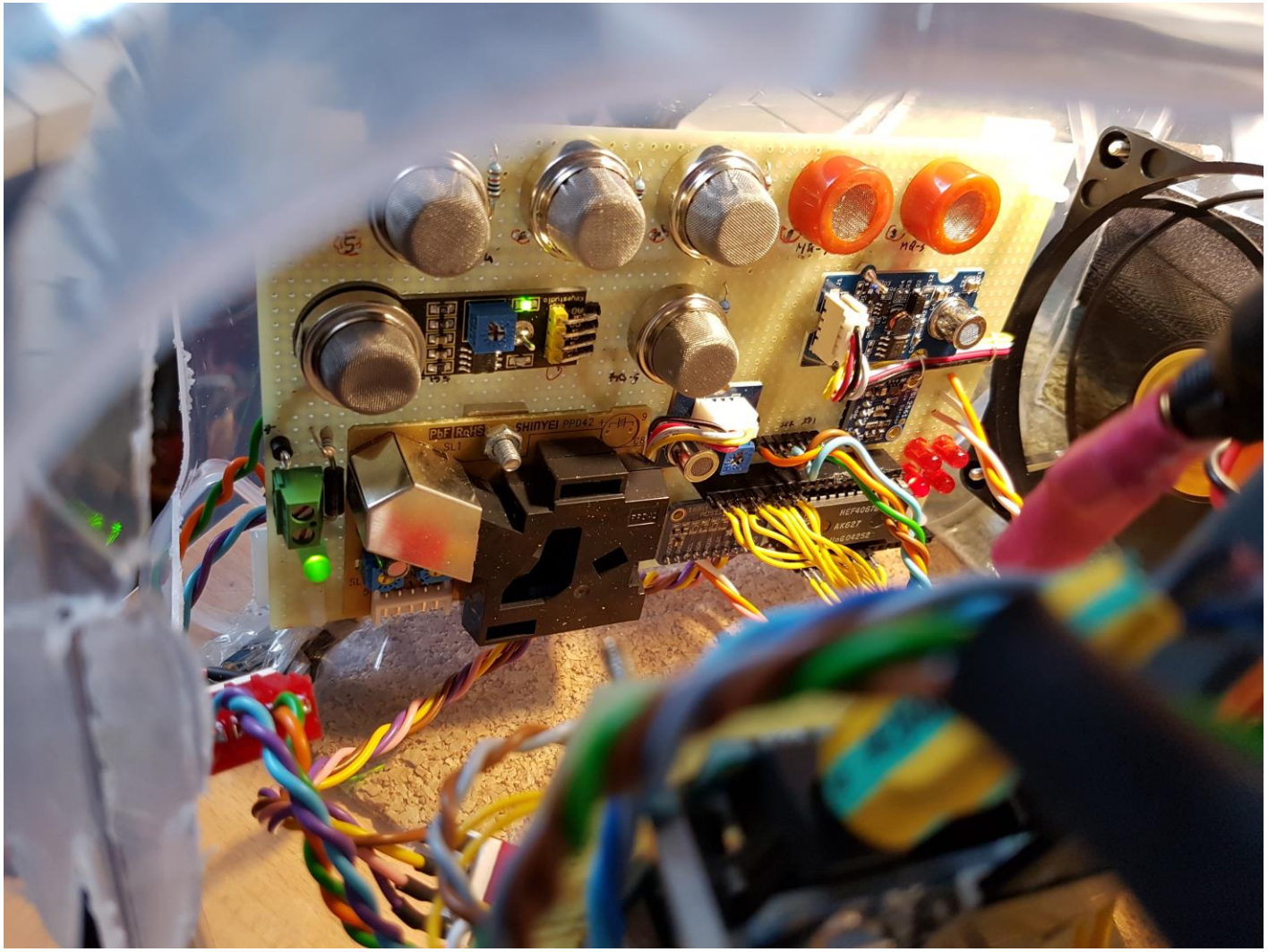
Creating the skeleton of the display graphic

```
// Sensore 5
 tft.setCursor(2, scendigiu+24+22+8*8);
 tft.setTextColor(black,yellow);
 tft.print("MQ4 GCN CH4 LNG");
 tft.setCursor(2, scendigiu+24+22+8*9);
 tft.print("Val            ");

// Sensore 6
 tft.setCursor(2, scendigiu+30+22+8*10);
 tft.setTextColor(black,white);
 tft.print("MQ8 H2 alc. LPG");
 tft.setCursor(2, scendigiu+30+22+8*11);
 tft.print("Val            ");

// Sensore 7
 tft.setCursor(2, scendigiu+36+22+8*12);
 tft.setTextColor(black,orange);
 tft.print("MQ6 LPG But Prp");
 tft.setCursor(2, scendigiu+36+22+8*13);
 tft.print("Val            ");

// Sensore 8
 tft.setCursor(2, scendigiu+42+22+8*14);
 tft.setTextColor(black,greenyellow);
 tft.print("MQ7  CO        ");
 tft.setCursor(2, scendigiu+42+22+8*15);
 tft.print("Val            ");

// Sensore 9
 tft.setCursor(2, scendigiu+48+22+8*16);
 tft.setTextColor(black,blue);
 tft.print("MQ3 alcool     ");
 tft.setCursor(2, scendigiu+48+22+8*17);
 tft.print("Val            ");
```

```
// Temperatura
 tft.setCursor(2, 100+22+8*18);
 tft.setTextColor(black,red);
 tft.print("Temperatura (C)");

// Umidita'
 tft.setCursor(2, 102+22+8*19);
 tft.setTextColor(black,cyan);
 tft.print("Umidita' rel(%)");

// Pressione
 tft.setCursor(2, 104+22+8*20);
 tft.setTextColor(black,green);
 tft.print("Pressione (Pa) ");

// Luminosita'
 tft.setCursor(2, 106+22+8*21);
 tft.setTextColor(black,magenta);
 tft.print("Illumin. (lux) ");

// Rumore
 tft.setCursor(2, 108+22+8*22);
 tft.setTextColor(black,yellow);
 tft.print("Rumore (raw pk)");
}
```

Creating the skeleton of the display graphic

END of SETUP sketch part


SHINYEI PPD42
AK6227

```
399 boolean datalogging,ratefastslow,ambientegas;
400 int16_t xPos = 96;
401 int16_t yPos;
402
403 boolean scrivivalori=true;
404 int16_t lumin;
405 int16_t pressio;
406 int16_t temper;
407 static float f_val = 123.6794;
408 static char outstr[15];
409
410 float sens1,sens2,sens3,sens4,sens5,sens6,sens7,sens8,sens9;
411 String sens1S,sens2S,sens3S,sens4S,sens5S,sens6S,sens7S,sens8S,sens9S;
412
413 int16_t DatiScrittiSuSD = 0;
414
415 void loop(void) {
416
417  sensors_event_t event;
418  tsl.getEvent(&event);
419
420  DateTime now = rtc.now();
421  tft.setCursor(0, 0);
422
423  Anno=String(now.year());
424  Mese=String(now.month()); if (Mese.toInt()<10)    { Mese="0"+Mese;}
425  Giorno=String(now.day()); if (Giorno.toInt()<10) { Giorno="0"+Giorno;}
426  Ora=String(now.hour()); if (Ora.toInt()<10) { Ora="0"+Ora;}
427  Minuto=String(now.minute()); if (Minuto.toInt()<10) { Minuto="0"+Minuto;}
428  Secondo=String(now.second()); if (Secondo.toInt()<10) { Secondo="0"+Secondo;}
429  Unix=String(now.unixtime());
430  Orologio="  "+Anno+"/"+Mese+"/"+Giorno+" - "+Ora+":"+Minuto+":"+Secondo;
```

Declaration of some variables and start of the LOOP part

```
sensorVall = String()+ "   T="+bme.readTemperature()+"'C   RH="+bme.readHumidity()+" %   "+bme.readPressure()+" Pa   ";
Orologio=Orologio+sensorVall;

if (event.light) {
 Orologio=Orologio+event.light+" "+"lux  ";
 lumin=event.light;
 };

if (scrivivalori==true) {
 tft.setTextSize(1);
 tft.setTextColor(white,navy);
 tft.print(Orologio);
 }

if ( bme.readTemperature()>120 || bme.readHumidity()==0)
{
 tft.setTextSize(2);
 tft.setTextColor(red,yellow);
 tft.setCursor(0, 100);
 tft.println("                                        ");
 tft.println(" --- WARNING: SENSOR BOARD FAILURE --- ");
 tft.println("                                        ");
 tft.setTextSize(1);
}

 xPos = xPos + 1;
 if(xPos>=tft.width()-2) {
 xPos=96;
 tft.fillRect(96,20,382,239,black);
 tft.fillRect(96,261,382,58,black);
}
```

Writing the first line in the display (temperature, humidity, pressure, time,...)

```
465 // Sensore 1
466 // Select address 0000  =0
467  if (ambientegas==true) {
468   digitalWrite(A_zero, LOW);digitalWrite(A_one,  LOW);digitalWrite(A_two,  LOW);digitalWrite(A_three,LOW);
469   adc0 = ads1115.readADC_SingleEnded(0);
470   dtostrf(adc0*4.096/32768,10, 8, outstr);
471   sens1S=outstr;
472   if (scrivivalori==true) {
473     tft.setTextColor(black,green);
474     tft.setCursor(2, scendigiu+22+8*1);
475     tft.print(String()+"V= "+outstr);
476     }
477   tft.fillCircle(xPos,260-adc0/137,1,green);
478  } else {sens1S="";}
479
480 // Sensore 2
481 // Select address 0001  =1
482  if (ambientegas==true) {
483   digitalWrite(A_zero, HIGH);digitalWrite(A_one,  LOW);digitalWrite(A_two,  LOW);digitalWrite(A_three,LOW);
484   adc0 = ads1115.readADC_SingleEnded(0);
485   dtostrf(adc0*4.096/32768,10, 8, outstr);
486   sens2S=outstr;
487   if (scrivivalori==true) {
488     tft.setTextColor(black,cyan);
489     tft.setCursor(2, scendigiu+6+22+8*3);
490     tft.print(String()+"V= "+outstr);
491     }
492   tft.fillCircle(xPos,260-adc0/137,1,cyan);
493  } else {sens2S="";}
```

Acquisition of data from the various gas sensors, writing the value on the display (if the option is enabled) and graphic plot

```
// Sensore 3
// Select address 0010  =2
 if (ambientegas==true) {
  digitalWrite(A_zero, LOW);digitalWrite(A_one, HIGH);digitalWrite(A_two, LOW);digitalWrite(A_three, LOW);
  adc0 = ads1115.readADC_SingleEnded(0);
  dtostrf(adc0*4.096/32768,10, 8, outstr);
  sens3S=outstr;
  if (scrivivalori==true) {
    tft.setTextColor(black,red);
    tft.setCursor(2, scendigiu+12+22+8*5);
    tft.print(String()+"V= "+outstr);
    }
  tft.fillCircle(xPos,260-adc0/137,1,red);
 } else {sens3S="";}

// Sensore 4
// Select address 0011  =3
 if (ambientegas==true) {
  digitalWrite(A_zero, HIGH);digitalWrite(A_one, HIGH);digitalWrite(A_two, LOW);digitalWrite(A_three, LOW);
  adc0 = ads1115.readADC_SingleEnded(0);
  dtostrf(adc0*4.096/32768,10, 8, outstr);
  sens4S=outstr;
  if (scrivivalori==true) {
    tft.setTextColor(black,magenta);
    tft.setCursor(2, scendigiu+18+22+8*7);
    tft.print(String()+"V= "+outstr);
    }
  tft.fillCircle(xPos,260-adc0/137,1,magenta);
 } else {sens4S="";}
```

Acquisition of data from the various gas sensors, writing the value on the display (if the option is enabled) and graphic plot

```
// Sensore 5
// Select address 0100  =4
 if (ambientegas==true) {
  digitalWrite(A_zero, LOW);digitalWrite(A_one, LOW);digitalWrite(A_two, HIGH);digitalWrite(A_three, LOW);
  adc0 = ads1115.readADC_SingleEnded(0);
  dtostrf(adc0*4.096/32768,10, 8, outstr);
  sens5S=outstr;
  if (scrivivalori==true) {
    tft.setTextColor(black,yellow);
    tft.setCursor(2, scendigiu+24+22+8*9);
    tft.print(String()+"V= "+outstr);
    }
  tft.fillCircle(xPos,260-adc0/137,1,yellow);
 } else {sens5S="";}

// Sensore 6
// Select address 0101 =5
 if (ambientegas==true) {
  digitalWrite(A_zero, HIGH);digitalWrite(A_one, LOW);digitalWrite(A_two, HIGH);digitalWrite(A_three, LOW);
  adc0 = ads1115.readADC_SingleEnded(0);
  dtostrf(adc0*4.096/32768,10, 8, outstr);
  sens6S=outstr;
  if (scrivivalori==true) {
    tft.setTextColor(black,white);
    tft.setCursor(2, scendigiu+30+22+8*11);
    tft.print(String()+"V= "+outstr);
    }
  tft.fillCircle(xPos,260-adc0/137,1,white);
 } else {sens6S="";}
```

Acquisition of data from the various gas sensors, writing the value on the display (if the option is enabled) and graphic plot

```
// Sensore 7
// Select address 0110 =6
 if (ambientegas==true) {
  digitalWrite(A_zero, LOW);digitalWrite(A_one, HIGH);digitalWrite(A_two, HIGH);digitalWrite(A_three, LOW);
  adc0 = ads1115.readADC_SingleEnded(0);
  dtostrf(adc0*4.096/32768,10, 8, outstr);
  sens7S=outstr;
  if (scrivivalori==true) {
    tft.setTextColor(black,orange);
    tft.setCursor(2, scendigiu+36+22+8*13);
    tft.print(String()+"V= "+outstr);
    }
  tft.fillCircle(xPos,260-adc0/137,1,orange);
 } else {sens7S="";}

// Sensore 8
// Select address 0111 =7
 if (ambientegas==true) {
  digitalWrite(A_zero, HIGH);digitalWrite(A_one, HIGH);digitalWrite(A_two, HIGH);digitalWrite(A_three, LOW);
  adc0 = ads1115.readADC_SingleEnded(0);
  dtostrf(adc0*4.096/32768,10, 8, outstr);
  sens8S=outstr;
  if (scrivivalori==true) {
    tft.setTextColor(black,greenyellow);
    tft.setCursor(2, scendigiu+42+22+8*15);
    tft.print(String()+"V= "+outstr);
    }
  tft.fillCircle(xPos,260-adc0/137,1,greenyellow);
 } else {sens8S="";}
```

Acquisition of data from the various gas sensors, writing the value on the display (if the option is enabled) and graphic plot

```
585 // Sensore 9
586 // Select address 1000 =8
587  if (ambientegas==true) {
588   digitalWrite(A_zero, LOW);digitalWrite(A_one, LOW);digitalWrite(A_two, LOW);digitalWrite(A_three, HIGH);
589   adc0 = ads1115.readADC_SingleEnded(0);
590   dtostrf(adc0*4.096/32768,10, 8, outstr);
591   sens9S=outstr;
592   if (scrivivalori==true) {
593     tft.setTextColor(black,blue);
594     tft.setCursor(2, scendigiu+48+22+8*17);
595     tft.print(String()+"V= "+outstr);
596     }
597   tft.fillCircle(xPos,260-adc0/137,1,blue);
598  } else {sens9S="";}
599
600 // End multiplexing
601   // Select address 0000  =0
602  digitalWrite(A_zero, LOW);digitalWrite(A_one,  LOW);digitalWrite(A_two,  LOW);digitalWrite(A_three,LOW);
603
604
605 // Temperatura
606   temper=bme.readTemperature();
607   if (temper<-15)  { temper=-15;}
608   if (temper>45) { temper=45;}
609   tft.fillCircle(xPos,318-temper,1,red);
610
611 // Umidita'
612   tft.fillCircle(xPos, 318-bme.readHumidity()/1.67 ,1,cyan);
```

Acquisition of data from the various gas sensors, writing the value on the display (if the option is enabled) and graphic plot

```
// Pressione
  pressio=bme.readPressure()-101325;
  if (pressio<1) { pressio=1;}
  if (pressio>58) { pressio=58;}
  tft.fillCircle(xPos,318-30-pressio/1,1,green);

// Luminosita'
 if (lumin>5800) {lumin=5800;}
  tft.fillCircle(xPos,318-lumin/50,1,magenta);

// Rumore
  adc1 = ads1115.readADC_SingleEnded(1);
  tft.fillCircle(xPos,320-adc1/580,1,yellow);


// legge interruttori
 Switch0= analogRead(A0);
 if(Switch0>512) {
  datalogging=true;
    tft.setCursor(0, 10);
    tft.setTextColor(black,red);
    tft.print(" DATA LOG: ON  ");
 }
 else
 {
  datalogging=false;
    tft.setCursor(0, 10);
    tft.setTextColor(black,yellow);
    tft.print(" DATA LOG: OFF ");
 }
```

```
Switch1= analogRead(A1);
if(Switch1>512) {
 ratefastslow=true;
 scrivivalori=true;

   tft.setCursor(100, 10);
   tft.setTextColor(black,cyan);
   tft.print(" SLOW/SHOW VL ");
}
else
{
 ratefastslow=false;
 scrivivalori=false;
   tft.setCursor(100, 10);
   tft.setTextColor(black,orange);
   tft.print(" FAST/PLOT OPT");
}
```

```
Switch2= analogRead(A2);
if(Switch2>512) {
 ambientegas=true;
   tft.setCursor(194, 10);
   tft.setTextColor(black,magenta);
   tft.print(" CAPTURE: GAS + Amb ");
}
else
{
 ambientegas=false;
   tft.setCursor(194, 10);
   tft.setTextColor(black,orange);
   tft.print(" CAPTURE: T,RH,p,L  ");
}
```

Graph of the values from the other sensors. Reading of the switches to enable the various options

```
678     tft.setCursor(322, 10);
679     tft.setTextColor(white,navy);
680     tft.print(String()+filename+" Saved:"+DatiScrittiSuSD);
681
682    if (datalogging==true) {
683     Orologio=Anno+"/"+Mese+"/"+Giorno+" "+Ora+":"+Minuto+":"+Secondo+" "+Unix;
684     sensorVall = String()+ " "+bme.readTemperature()+" "+bme.readHumidity()+" "+bme.readPressure()+" "+adc1+" ";
685     Orologio=Orologio+sensorVall;
686     Orologio=Orologio+event.light+" ";
687
688     logfile.print(Orologio);
689
690     if (ambientegas==true) {
691       logfile.println(sens1S+" "+sens2S+" "+sens3S+" "+sens4S+" "+sens5S+" "+sens6S+" "+sens7S+" "+sens8S+" "+sens9S );
692     } else {logfile.println("");}
693
694     DatiScrittiSuSD++;
695     logfile.flush();
696    }
697
698 }
```

Writing data to SD (if the option is enabled)

END OF PROGRAM


thank
you!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Ing. Paolo Guidorzi

Dipartimento di Ingegneria Industriale

paolo.guidorzi@unibo.it

http://acustica.ing.unibo.it/Staff/paolo/index.html