

# MICROCONTROLLERS & OPEN SOURCE HARDWARE



Jérémie GRISOLIA

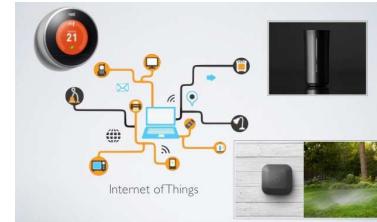
Département de Génie Physique – INSA TOULOUSE

05.61.55.96.58 – Bureau 138 (1er étage)

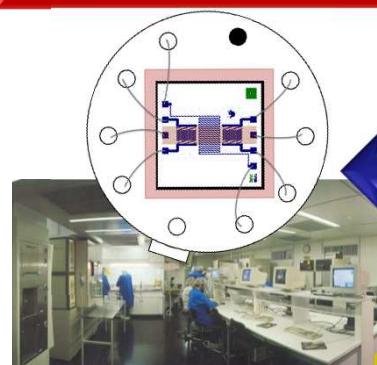
[jeremie.grisolia@insa-toulouse.fr](mailto:jeremie.grisolia@insa-toulouse.fr)

<http://moodle.insa-toulouse.fr/course/view.php?id=494>

## INTRODUCTION TO THE MEASUREMENT CHAIN AND SMART DEVICES



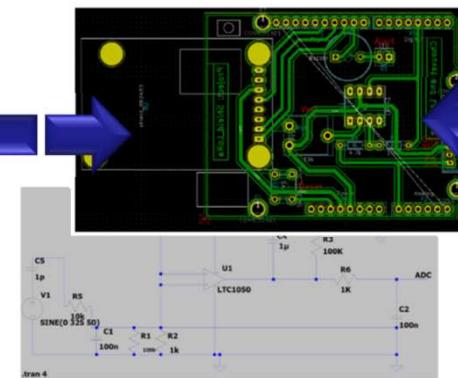
### SENSOR FABRICATION (CLEAN ROOM AIME)



### SENSORS AND DATASHEET



### PCB ELABORATION (KICAD) + ANALOG CIRCUIT



### MICROCONTROLLERS & OPEN SOURCE HARDWARE



GITHUB



# INTRODUCTION

**OBJECTIVES:** what are microcontrollers, their internal and architecture, their possibilities

- Set-up of an Arduino® ecosystem around Open-Source Hardware Platform, ESP32...

The Arduino® is a complete platform of electronic development to achieve lower cost high performance microcontroller-based applications.



This course provides all the necessary elements to the design and implementation of many applications with particularly:

- elements to create its own electronic circuits (simulation and schematic to the routing to pull its PCB)
- presentation of the language syntax of programming and development environment;
- Dozens of pattern interfaces with the most various devices (displays, motors, sensors, bus I2C, Nunchuk, touch-screen DS...).
- Several dozens of program examples for implementing these interfaces
- elements to create your own libraries
- elements to create your own human interface device
- elements of interfacing Arduino/Processing (programming interfaces Java), C#, ANDROID and microPython
- elements to create an IoT application

# SIMPLIFIED SUMMARY OF THE LECTURE

## I - MICROCONTROLLERS AND THEIR ARCHITECTURES:

## II - THE OPEN - SOURCE ARDUINO® PLATFORM

- II-1 What is an Arduino?
- II - 2 IDE development platform
- II-3 the addressable components: actuators and sensors?

## III - IMPLEMENTATION OF THE ARDUINO :

- III-1 input/output digital
- III-2 inputs/outputs analog
- III-3 applications digital & analog
- III-4 do the analog with digital debouncing
- III-5 hardware and software debouncing
- III-6 interrupts (hardware and software)
- III-7 communication series: (RS232) asynchronous & synchronous (I2C, SPI, one wire)
- III-8 create a library
- III-9 the shields & their creation

## IV - COMMUNICATION OF THE ARDUINO WITH OTHER PLATFORMS:

- IV - 1 PROCESSING => JAVA,
- IV - 2 ANDROID,
- IV - 3 MICRO-PYTHON,
- IV - 4 NODE-RED
- IV - 5 MICRO-PYTHON,
- IV - 6 IoT

## V - IMPLEMENTATION OF THE ESP32 WITH MICRO-PYTHON:

- V-1 ANALOG READ, OLED, LoRa, CAPACITIVE TOUCH, HALL EFFECT ON ESP32
- V-2 MQTT ON ESP32/ESP8266
- V-3 MQTT - Connect ESP32 to Node-RED

## VI - IMPLEMENTATION OF ARTIFICIAL INTELLIGENCE

- VI – 1 ARDUINO AND ARTIFICIAL INTELLIGENCE

## VII - ESSENTIAL REFERENCES

# PLAN DETAILLE DU COURS (AVEC LIENS HYPERTEXTES)

[1 - CONTEXT: OPEN SOURCE HARDWARE](#)

[2 - ARDUINO – PLATFORM](#)

[3 - GETTING STARTED WITH THE DEVELOPMENT ENVIRONMENT](#)

[4 – MICROCONTROLLERS](#)

[5 - WHAT FAMILY OF MICROCONTROLLERS](#)

[6 - AUTOPSY OF AN ARDUINO PLATFORM](#)

[7 DIGITALS INPUTS/OUTPUTS](#)

[7 - 1 TP1A](#)

[7 - 2 TP1B](#)

[7 - 3 TP1C](#)

[8 - ANALOG INPUTS](#)

[8 – 1 TP1E](#)

[8 - 2 TP2A](#)

[9 - ADRESSABLE DEVICES ?](#)

[10 - DIGITAL & ANALOG APPLICATIONS](#)

[10 - 1 TP2B](#)

[10 – 2 TP2C](#)

[11 - DO "ANALOG" WITH DIGITAL](#)

[11 – 1 TP3-A](#)

[11 – 2 TP3-B](#)

[12 – DEBOUNCING](#)

[13 – INTERRUPTS](#)

[14 - HARDWARE INTERRUPTS](#)

[14 - 1 TP3-F](#)

[15 - SOFTWARE INTERRUPTS](#)

[16 – TIMERS](#)

[17 - SERIAL COMMUNICATION](#)

[17-1 - ASYNCHRONOUS SERIAL COMMUNICATION](#)

[17-2 - SYNCHRONOUS SERIAL COMMUNICATION](#)

[17-3 - SYNCHRONOUS SERIAL COMMUNICATION: I2C](#)

[17 – 4 - TP4-A](#)

[18 - SYNCHRONOUS SERIAL COMMUNICATION: SPI](#)

[19 – LIBRAIRIES](#)

[20 - THE SHIELDS](#)

[21 - TO GO FURTHER: PROCESSING](#)

[21 - 1 - TP4-D](#)

[22 - PROCESSING FOR ANDROID](#)

[22 - 1 - TP5](#)

[23 - TO GO FURTHER: LIFA](#)

[24 - ARDUINO ET C#](#)

[25 - TO GO FURTHER: PYTHON](#)

[26 - OTHER INTERESTING TRACKS ?](#)

[27 - ARDUINO ET NODE-RED](#)

[28 - MICRO-PYTHON](#)

[29 - FLASHING MICRO-PYTHON TO ESP32](#)

[30 - GETTING STARTED WITH uPyCRAFT IDE](#)

[31 - MICROPYTHON PROGRAMMING BASICS](#)

[32 - MICROPYTHON MODULES](#)

[32 - 1 - TP ANALOG READ ON ESP32](#)

[32 - 2 - TP OLED ON ESP32](#)

[32 - 3 - TP LoRa ON ESP32](#)

[32 - 4 - TP CAPACITIVE TOUCH](#)

[32 - 5 - TP HALL EFFECT SENSOR](#)

[34 - HOW MQTT WORKS](#)

[35 - TP - MQTT ON ESP32/ESP8266](#)

[36 - MQTT - Connect ESP32 to Node-RED](#)

[37 - ARDUINO ET INTELLIGENCE ARTIFICIELLE](#)

[38 – THINGS NETWORK + NODERED](#)

[39 - RAPPEL SUCCESSIVE APPROXIMATION ADC](#)

[40 - COMMANDES LINUX UTILES](#)

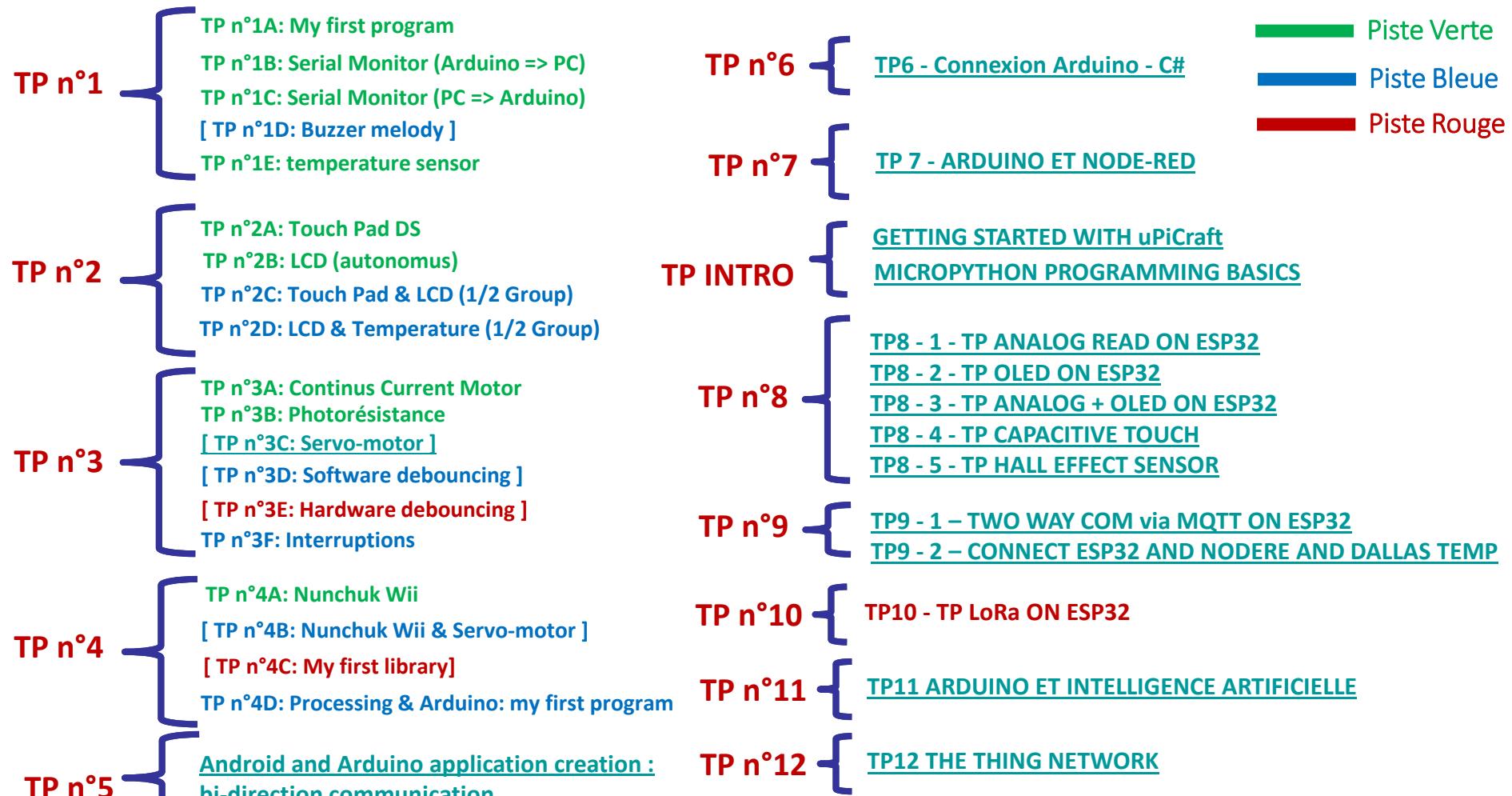
[41 – COMPARATIFS](#)

[42 - AND SECURITY ?](#)

[43 – REFERENCES](#)

[44 - QUIZZ](#)

# PRATICAL WORK



# PRATICAL WORK

TP  
Mini-Lab

**Construction of an Open Hardware solution:**

**Creating a shield to send data from a gas by LoRa sensor to The Thing Networks:**

**Link : <https://moodle.insa-toulouse.fr/course/view.php?id=494#section-19>**

**GREEN track :**

**I) Communication even LoRa pair**

**1 - communicate with the chip RN2483**

**2 - setting in place of a chip in rec mode reconstructed**

**3 - set place a chip in transmit mode**

**BLUE track :**

**II) creating a LoRaWAN application: 'Gas sensor connected' to accomplish this application we will use several components: Arduino Uno, RN2483, Buzzer, MQ, button gas sensor tappet, Resistance, Led**

**1) a quick prototype: purpose: network connection TTN, management of a disruption on the gas sensor indicating the presence of danger, sends the data over the network.**

**2) realize the shield: the realization of the shield will be done through the KiCAD software.**

**RED track :**

**III) manage the power consumption of the entire system 1) find a way to measure consumption**

**2) manage consumption**

**BLACK track :**

**Goal: achieve a dashboard with NodeRed to visualize the data of TTN!**

# I - CONTEXT: OPEN SOURCE HARDWARE

# ARDUINO AND OPEN-SOURCE HARDWARE : A PHILOSOPHY

## 1 - 'hardware' material is 'open source' :

The free hardware (OSHW - OpenSource Hardware) is a term that includes "tangible" products - machines, devices or all physical devices - whose plans have been made public in such a way that anyone can make them, edit, distribute and use (under the same terms as the license of the original work).

## 2 - The 'software' is 'open source':

we can use it, modify it and distribute it freely (under the same terms as the license of the original work).



Beware, in OSH, the hardware design (i.e. mechanical drawings, schematics, BOM, layout PCB, source code and layout of integrated circuits...), in addition to the software that drives the hardware are all required to approach free software (free and open-source software (FOSS))

*Most of the authors, regardless of their field of activity, and independently of their status of amateur or professional, have interest to foster an ecosystem where works can be distributed, re-used and derived in creative ways.*

*More easy is to reuse and derive works, the more our culture is enriched.*

**Major interest of open source work methods:**

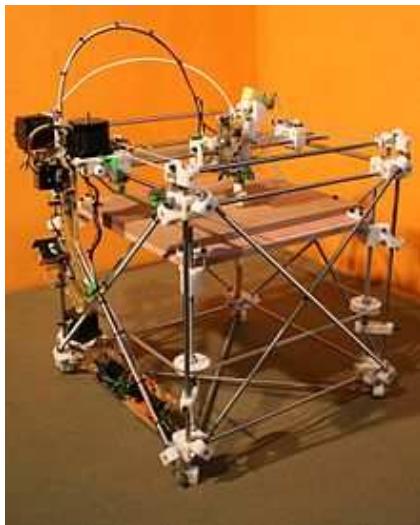
- principle of community development
- mutualisation of skills
- mutualization of problem solving.

<http://freedomdefined.org/OSHW/translations/fr>

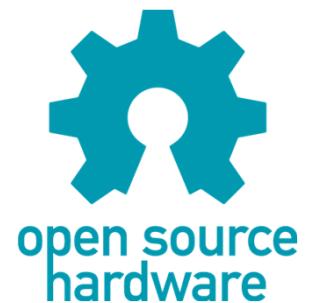
[http://en.wikipedia.org/wiki/Open\\_source\\_hardware](http://en.wikipedia.org/wiki/Open_source_hardware)

<http://freedomdefined.org/Definition/Fr>

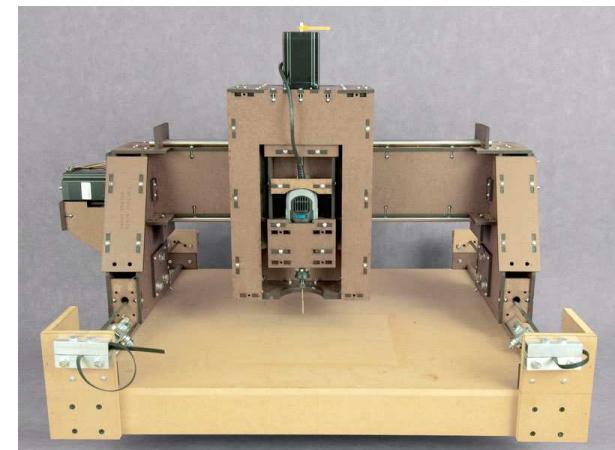
# OPEN-SOURCE HARDWARE EXAMPLES



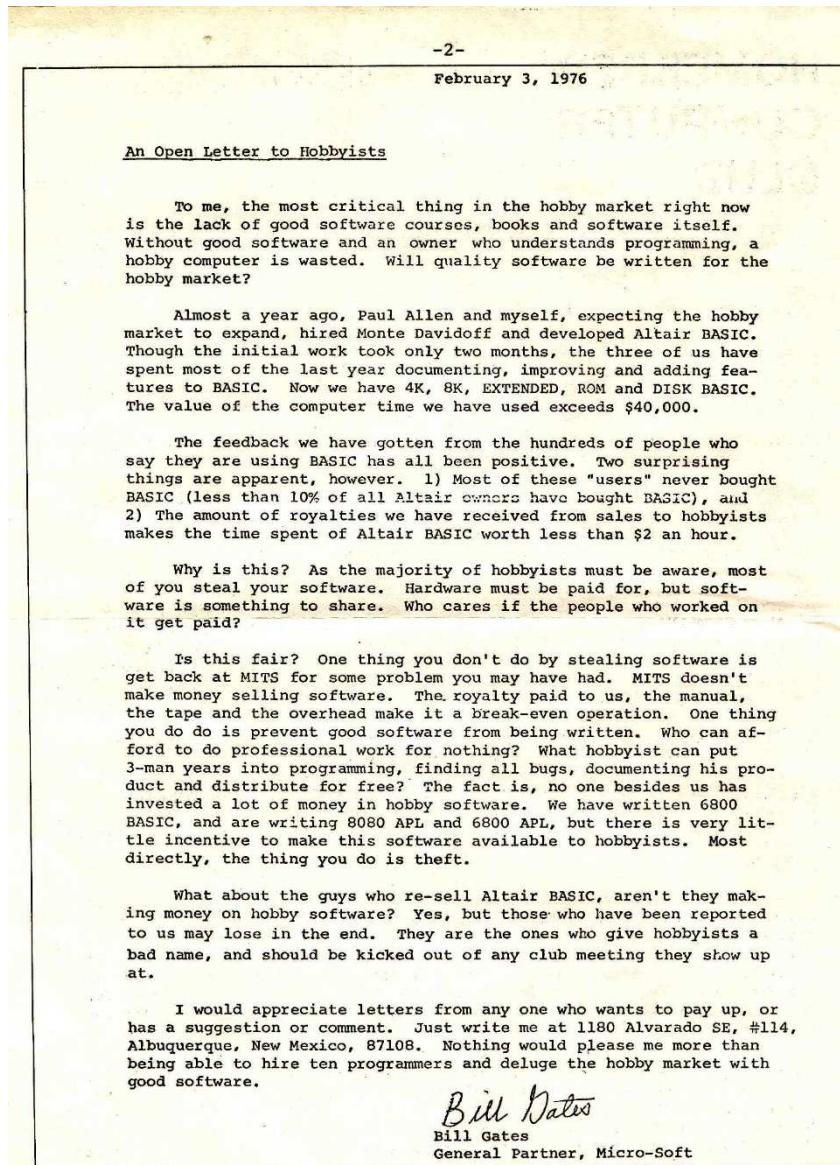
REPRAP



open source  
hardware



# AN OPEN LETTER TO HOBBYISTS by BILL GATES



[http://fr.wikipedia.org/wiki/An\\_Open\\_Letter\\_to\\_Hobbyists](http://fr.wikipedia.org/wiki/An_Open_Letter_to_Hobbyists)

# LOIS ET PRINCIPES SOUS-TENDANT LES LICENCES LIBRES

-Copyleft is the authorization given by the author of a work subject to copyright (artwork, text, computer or other program) to use, study, modify and copy his work, insofar as this authorization remains preserved.

=> the author denies that his work can evolve with a restriction of the right to copy (copyright).



-As a result, the contributor making a change (correction, addition, re-use, etc.) is forced to redistribute its own contributions with the same conditions of use as the original.

In other words, the creations made from items under copyleft inherit de facto this copyleft.

The best-known free license using copyleft is the GNU General Public License, but there are also other licenses, created specifically for certain areas very various (art, role play, journal, etc.), which may be considered as of "copyleft licenses".



The free Copyleft license is the best way to encourage sharing and encourage creativity. Intellectual creation must free themselves from the shackles of copyright, because the copyright only serves the industry, not the culture.

Culture can flourish in the sharing!

*If someone takes my work, improves it and makes something new, beautiful, great, useful, via Copyleft, then I'd be very happy and everyone will win.*

*In short, I think that the copy is not a brake, but a lever.*

<http://fr.wikipedia.org/wiki/Copyleft>

<http://artlibre.org/licence/lal>

[http://des-trucs-pour-changer-de-vie.blogspot.fr/2013/02/des-trucs-pour-changer-de-vie-devient.html#.UUw2Pjc\\_4tI](http://des-trucs-pour-changer-de-vie.blogspot.fr/2013/02/des-trucs-pour-changer-de-vie-devient.html#.UUw2Pjc_4tI)

## EXEMPLE DE LICENCE LIBRE: LICENCE CREATIVE COMMONS

Creative Commons licenses were created on the premise that intellectual property was fundamentally different from physical property, and the observation that the current copyright laws were an obstacle to the dissemination of the culture.



**GOAL:** To provide a legal tool that guarantees the protection of the rights of the author of an artistic work and the free movement of the cultural content of this work, in order to allow authors to contribute to a legacy of works available in the 'public domain' (term taken in the broad sense).

- Paternity [BY] (grant): the work can be freely used provided that attribute it to the author by citing its name.  

- No commercial use [NC] (Noncommercial): the owner of rights may authorize any use or instead restrict it to non-commercial use (commercial use remaining subject to approval).  

- No change [ND] (NoDerivs): the owner of rights may continue to reserve the faculty to make derivative works of type or instead beforehand authorize modifications, translations.  

- Sharing of initial conditions to the same [SA] (ShareAlike): the owner of the rights may authorize in advance changes; can overlap (SA) required for so-called derivative works to be offered to the public with the same freedoms (under the same Creative Commons options) as the original work.  


This License including although:

waiver - no matter which of the above conditions can be waived if you have the permission of the owner of rights.

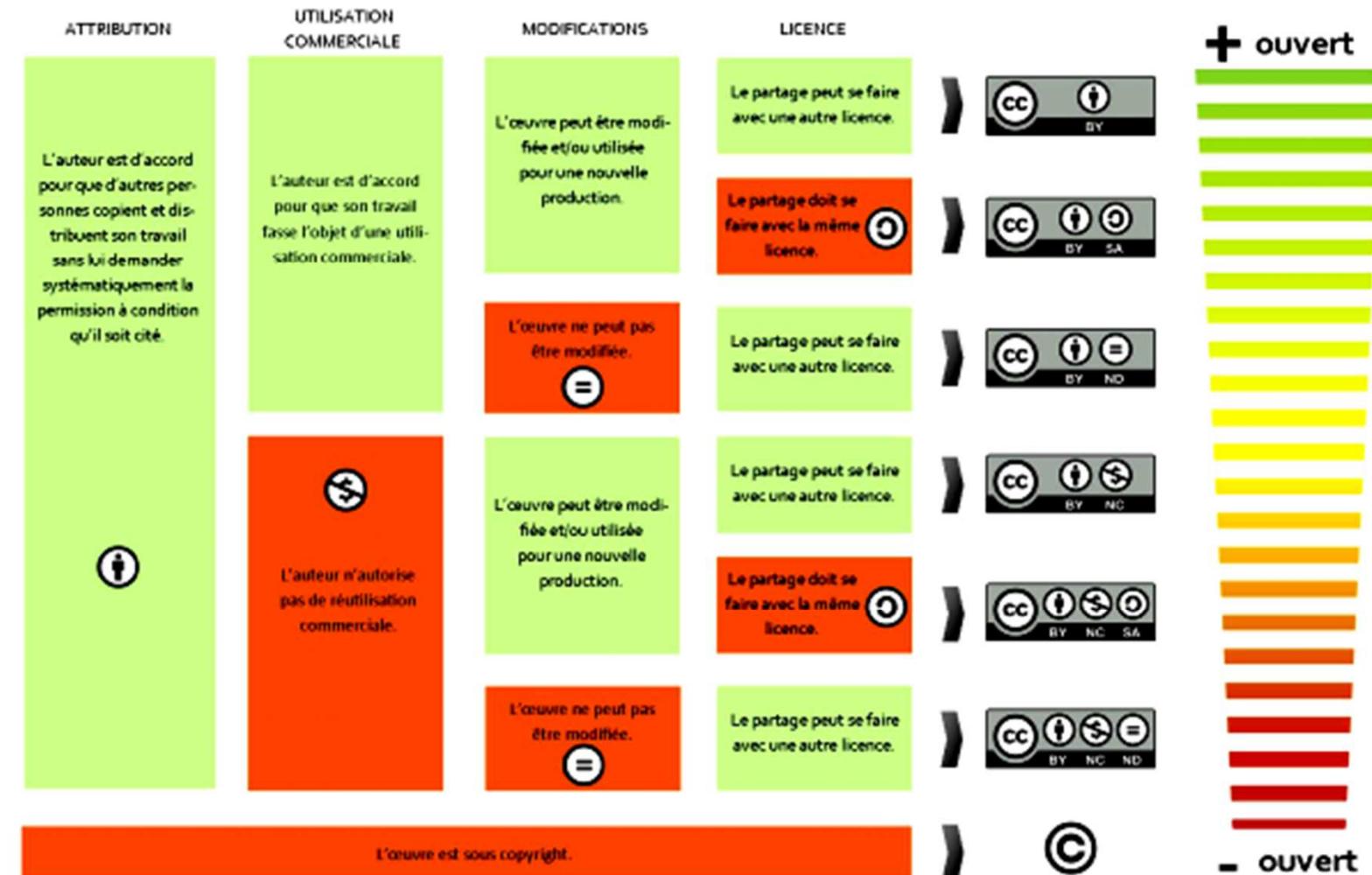
<http://creativecommons.org/licenses/by/3.0/deed.fr>

[http://fr.wikipedia.org/wiki/Licence\\_Creative\\_Commons](http://fr.wikipedia.org/wiki/Licence_Creative_Commons)

# CLASSIFICATION OF LICENCES

## Video explication licence CC

Comment attribuer ou utiliser une licence Creative Commons ?



<http://www.netpublic.fr/2015/04/comment-choisir-une-licence-creative-commons/>

<http://www.netpublic.fr/2012/12/creative-commons-video/>    <http://www.netpublic.fr/2012/03/creative-commons-dossier/>

F. Bordignon — Mars 2015



# CLASSIFICATION DES LICENCES

Désignation complète du contrat	Terme abrégé désignant la licence	Symboles désignant la licence				Type de licence
Paternité	CC-BY					<a href="#">Licence libre non copyleft</a>
Paternité Partage des conditions initiales à l'identique	CC-BY-SA					<a href="#">Licence libre copyleft</a>
Paternité Pas de modification	CC-BY-ND					<a href="#">Licence de libre diffusion</a>
Paternité Pas d'utilisation commerciale	CC-BY-NC					<a href="#">Licence de libre diffusion</a>
Paternité Pas d'utilisation commerciale Partage des conditions initiales à l'identique	CC-BY-NC-SA					<a href="#">Licence de libre diffusion</a>
Paternité Pas d'utilisation commerciale Pas de modification	CC-BY-NC-ND					<a href="#">Licence de libre diffusion</a>

### III - ARDUINO® PLATFORM

### III - ARDUINO® PLATFORM

# WHAT IS AN ARDUINO ?

A software

A hardware



```
ADXL3xx | Arduino 0022
File Edit Sketch Tools Help
ADXL3xx
/*
  * constants describe the pins. They won't change:
  */
int groundpin = 18;           // analog input pin 4 -- ground
int powerpin = 19;           // analog input pin 5 -- voltage
int xpin = A3;                // x-axis of the accelerometer
int ypin = A2;                // y-axis
int zpin = A1;                // z-axis (only on 3-axis models)

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

  // provide ground and power by using the analog inputs as normal digital pins. This makes it possible to directly connect the breakout board to the Arduino. If you use the normal 5V and GND pins on the Arduino, you can remove these lines.
  pinMode(groundpin, OUTPUT);
  pinMode(powerpin, OUTPUT);
  digitalWrite(groundpin, LOW);
  digitalWrite(powerpin, HIGH);

}

void loop()
{
  // read the sensor values:
  int x = analogRead(xpin);
  int y = analogRead(ypin);
  // print a tab between values:
  Serial.print("\t");
  Serial.print(analogRead(zpin));
  // print a tab between values:
  Serial.print("\t");
  Serial.println();
  // delay before next reading:
  delay(100);
}

Done compiling.

Binary sketch size: 2628 bytes (of a 32256 byte maximum)
1
```

The playground is a publicly-editable wiki about Arduino.

Manuals and Curriculum  
Board Setup and Configuration  
Development Tools  
Interfacing With Hardware

- Output
- Input

The Arduino Playground

Photo of an Arduino Board

Arduino Playground, a wiki where all the users of Arduino and benefit from their collective research.

to post and share your own code, circuit diagrams, tutorials, tips and tricks, and after all the hard work, to show off your can edit and add to the pages here.

round is a work in progress. We can use all the help you can!

The microcontroller card is programmed using the Arduino programming language (based on WIRING) and the Arduino development environment (based on the PROCESSING).

## WHY AN ARDUINO ?

There are many microcontrollers and many platforms based on microcontrollers available for programmed Electronics: Parallax Basic Stamp, Netmedia's BX - 24, Phidgets, MIT's Handyboard, and many others that offer the comparable features.

### So why Arduino?

- **A clear and simple programming environment:** environmental programming Arduino (= the Arduino software) is easy to use for beginners, while being flexible enough so that advanced users can benefit also.
- **Cheap:** Arduino cards are relatively expensive compared to other platforms. The less expensive versions of the Arduino module can be assembled by hand, and even the Arduino pre-assembled cost less than € 25 (microcontroller includes...)!
- **Cross-platform:** the Arduino software, written in Java, runs on Windows, Macintosh and Linux operating systems. Most microcontrollers systems are limited to Windows.
- **Important communication platform:**
  - completely 'stand-alone', or
  - he talks to other devices: 'it, Flash, Processing (Java), Pure data, MAX/MSP, Ruby, Python, .NET...

### -On the internet, we find:

- A community of users.
- Use guides.
- Examples.
- Support forums...

## WHY AN ARDUINO ?

### 1 - Software Open Source and extensible: => can use it and modify it freely.

The Arduino software and the Arduino language are published under open source, available license to be supplemented by experienced programmers.

The language can also be extended using C++ libraries, and people who want to understand the technical details can reconstruct the passage of the Arduino language in c for AVR Microcontroller on which it is based.

Similarly, you can add code to the AVR - C language directly in your Arduino programs if you want to.

### 2 - Open source hardware and extensible: => you can copy it, manufacture it and modify it freely.

Arduino cards are based on the Atmel ATMEGA8 microcontrollers, 168, 328, etc...

Which patterns of the modules are published under a Creative Commons (CC) license. Circuits experienced or relatively inexperienced designers can make their own version of Arduino, complementing them and improving them.

But be careful: a software or a hardware OPEN SOURCE is before all a software or a hardware whose codes or plans are accessible and modifiable by all.

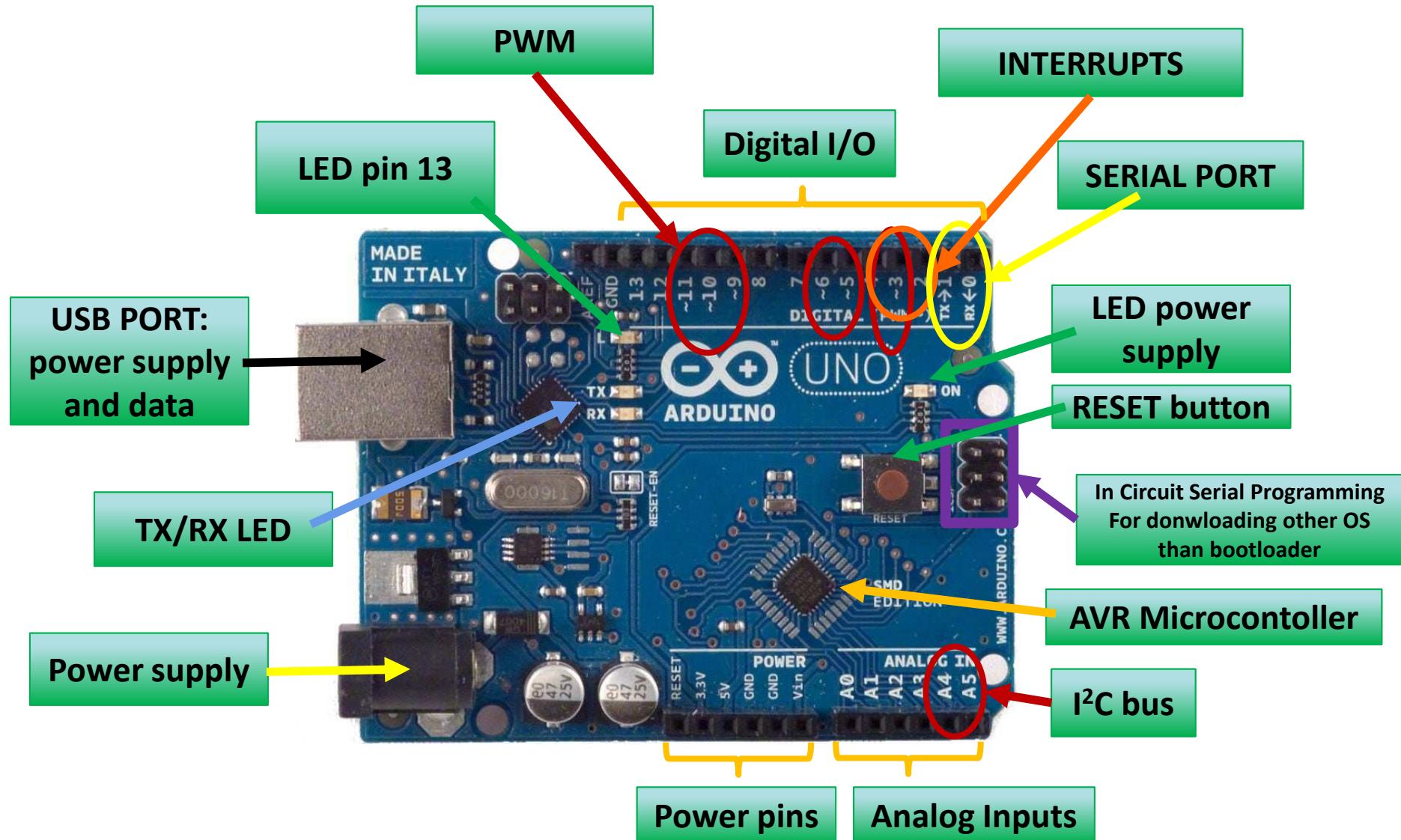
This sharing of knowledge does not preclude compensation!

The company Arduino is a commercial company, many companies work and thrive around the free (~ 100 M\$ total in 2010) but allows especially exchanges and circulation of knowledge...

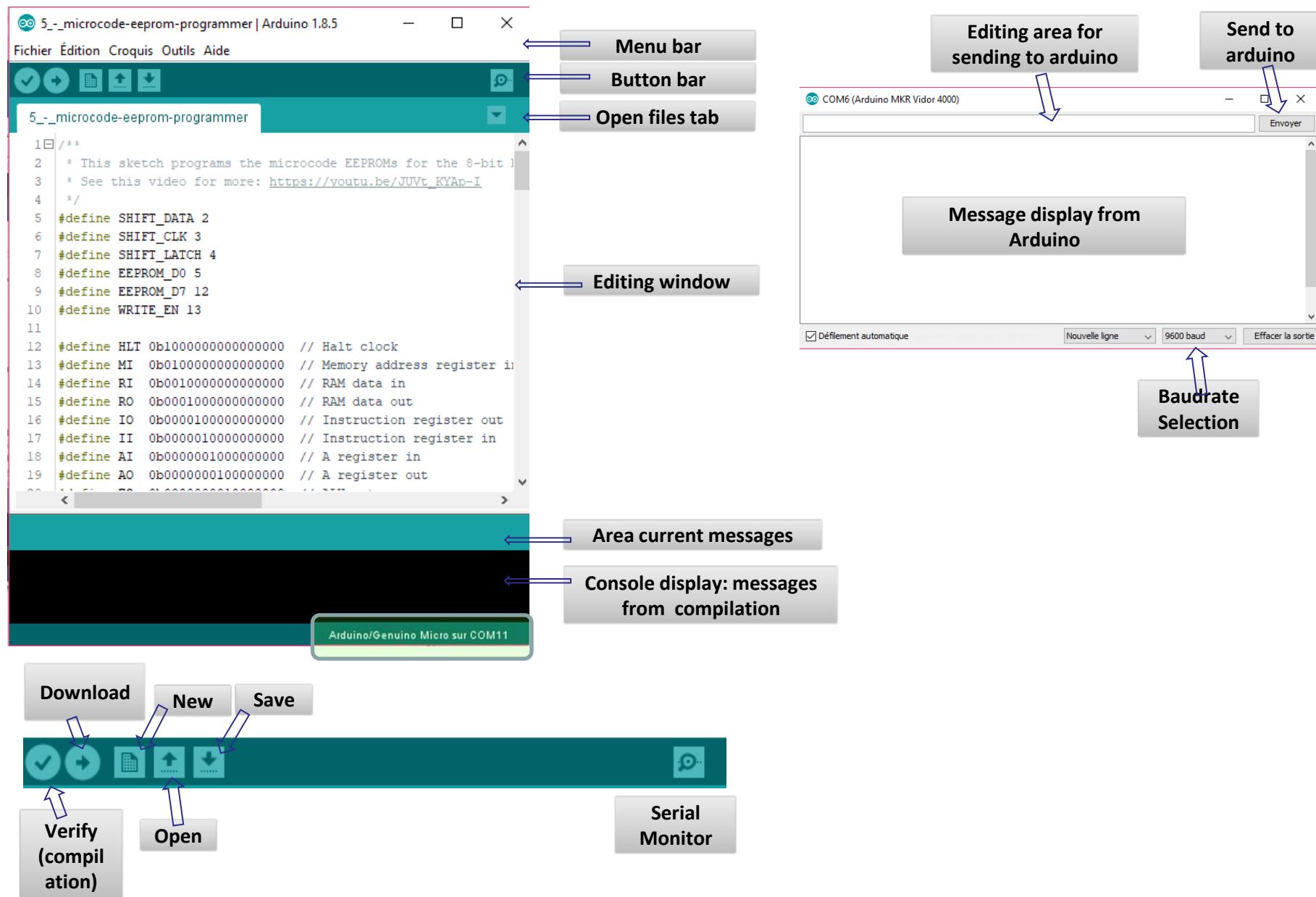
A period where the concerns about the sharing of knowledge are large, have a small ethical reflection on the merchandising of knowledge and his work seems beneficial...

# GETTING STARTED WITH THE DEVELOPMENT ENVIRONMENT

# AUTOPSY OF AN ARDUINO BOARD



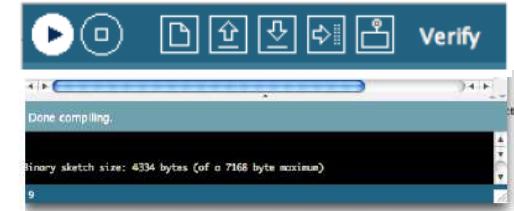
# IDE ARDUINO



# PRINCIPLE OF OPERATION OF THE ARDUINO AND ITS IDE

The Arduino to program in a high-level language ( $\neq$  machine language) mixing of C and C++ restricted and adapted to the possibilities of the board (language used by professionals).

1. Designing or opening an existing with the ARDUINO On software program



2. checks this program with the ARDUINO (compilation) software.

3. If errors are reported, you change / fix the program.



4. Load (upload) the program on the board,

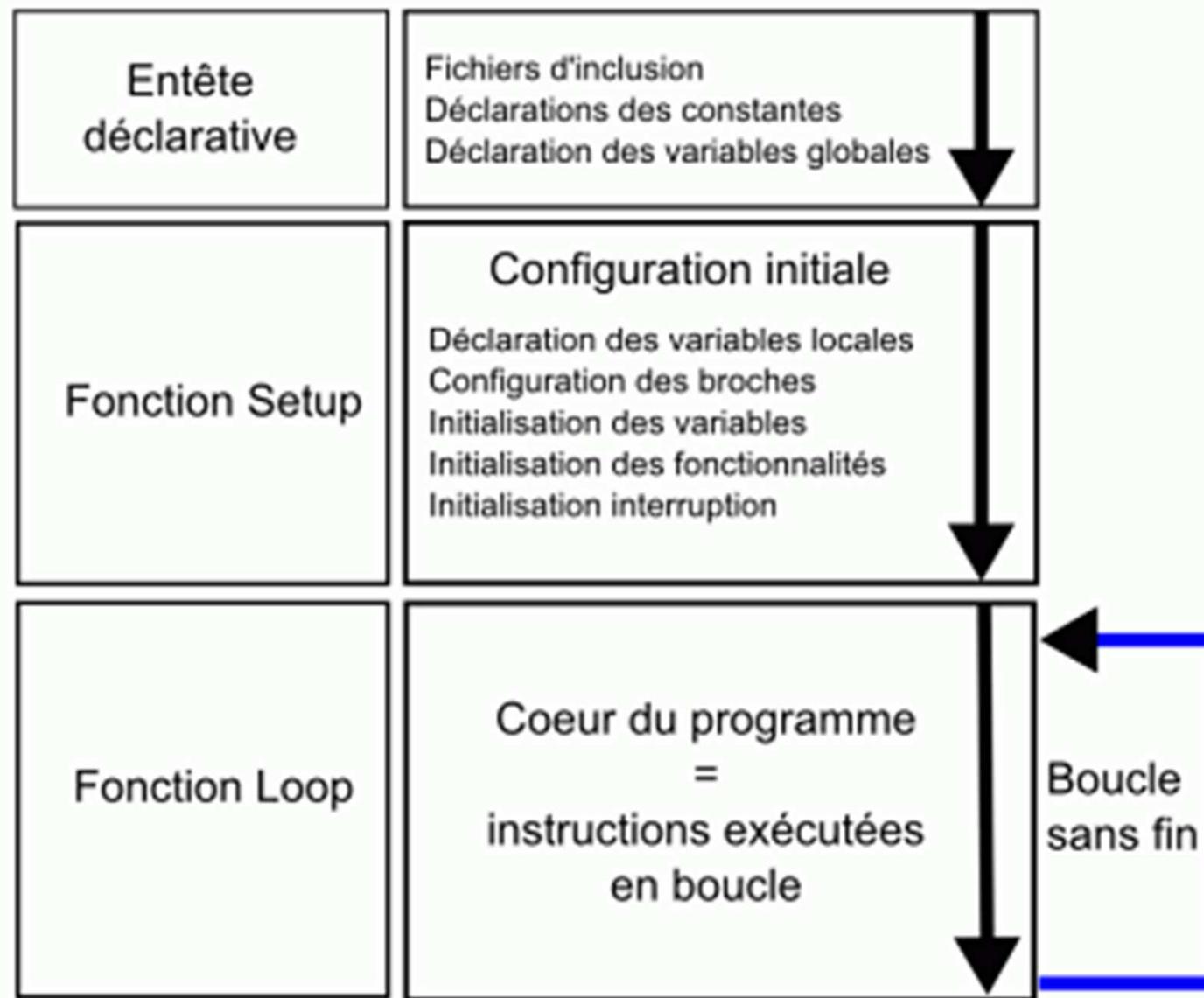
5. the electronic Assembly wiring.

6. The program execution is automatic after a few seconds if the arduino is connected (possibly you did a reset)

7. powered the card either through the USB port, or by a stand-alone power source (battery 9 volts for example)

8. Check if the installation works (and possibly a debugging)

# OPERATION OF THE IDE: STRUCTURE OF A PROGRAM



# OPERATION OF THE IDE: STRUCTURE OF A PROGRAM

## Commentaires

Toujours écrire des commentaires sur le programme: soit en multiligne, en écrivant entre des /\* \*/ , soit sur une ligne de code en se séparant du code avec //

(Syntaxe en marron, paramètres utilisateur en vert)

```
/* Ce programme fait clignoter une LED branchée sur la broche 13  
 * et fait également clignoter la diode de test de la carte  
 */
```

## Définition des variables:

Pour notre montage, on va utiliser une sortie numérique de la carte, qui est par exemple la 13 ème sortie numérique. Cette variable doit être définie et nommée ici: on lui donne un nom arbitraire BrocheLED . Le mot de la syntaxe est pour désigner un nombre entier est int

## Configuration des entrées-sorties void setup():

Les broches numériques de l'Arduino peuvent aussi bien être configurées en entrées numériques ou en sorties numériques. Ici on va configurer BrocheLED en sortie. pinMode ( nom, état) est une des quatre fonctions relatives aux entrées-sorties numériques.

```
void setup()  
{  
    pinMode(BrocheLED, OUTPUT); // configure BrocheLED comme une  
    // sortie  
}
```

## Programmation des interactions void loop():

Dans cette boucle, on définit les opérations à effectuer, dans l'ordre:

- digitalWrite ( nom, état) est une autre des quatre fonctions relatives aux entrées-sorties numériques.
- delay(temps en millisecondes) est la commande d'attente entre deux autres instruction
- Chaque ligne d'instruction est terminée par un point virgule
- Ne pas oublier les accolades, qui encadrent la boucle.

```
void loop()  
{  
    digitalWrite(BrocheLED, HIGH); // met la sortie num. à l'état haut (led  
    // allumée)  
    delay(3000); // attente de 3 secondes  
    digitalWrite(BrocheLED, LOW); // met la sortie num. à l'état bas (led  
    // éteinte)  
    delay(1000); // attente de 1 seconde  
}
```



TP to realize: download the program 'Blink' in the Arduino Uno

## III – 1 DIGITALS INPUTS/OUTPUTS

# DIGITAL INPUT/OUTPUT

The Arduino has 14 input/output digital D0 to D13.

In 'void setup', you must declare a PIN as an input or as an output by one of the following two statements:

```
pinMode (pin_name, INPUT); //input pin  
pinMode (pin_name, OUTPUT); //output pin
```

Output: it sends either 5V on pin, or 0V.

This corresponds to a '1' or a '0', to a level "above" or at a "low" level

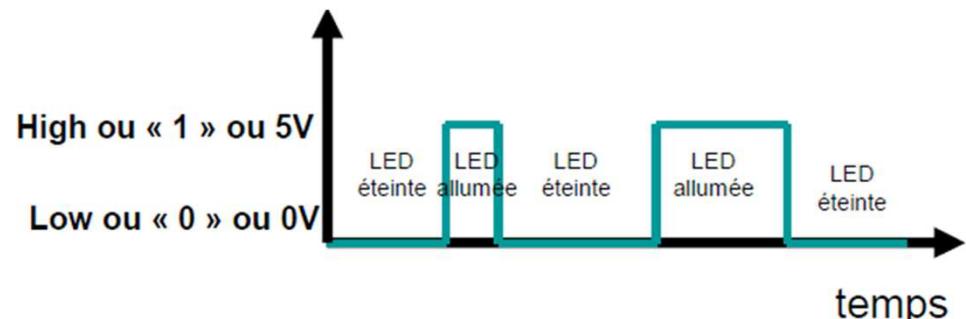
In the program this corresponds to the following instructions:

```
digitalWrite (pin_name, HIGH); //send 5V to pin is « 1 »  
digitalWrite (pin_name, LOW); //send 0V on pin is "0"
```

Input: the board can read either a high level ('1' or HIGH) is a low level ('0' or LOW)

In the program this corresponds to the following instructions:

```
digitalRead (pin_name);
```



**Signal numérique** : signal qui ne prend que deux états distinct comme 0V et 5V soit « 0 » et « 1 ».

# VALUE LOW AND HIGH ON ARDUINO?

For a remainder, we found here the voltage of the logical levels of entry for the Arduino (ATmega 328):

## 28.2 DC Characteristics

$T_A = -40^\circ\text{C}$  to  $85^\circ\text{C}$ ,  $V_{CC} = 1.8\text{V}$  to  $5.5\text{V}$  (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
$V_{IL}$	Input Low Voltage, except XTAL1 and $\overline{\text{RESET}}$ pin	$V_{CC} = 1.8\text{V} - 2.4\text{V}$ $V_{CC} = 2.4\text{V} - 5.5\text{V}$	-0.5 -0.5		$0.2V_{CC}^{(1)}$ $0.3V_{CC}^{(1)}$	V
$V_{IH}$	Input High Voltage, except XTAL1 and $\overline{\text{RESET}}$ pins	$V_{CC} = 1.8\text{V} - 2.4\text{V}$ $V_{CC} = 2.4\text{V} - 5.5\text{V}$		$0.7V_{CC}^{(2)}$ $0.6V_{CC}^{(2)}$	$V_{CC} + 0.5$ $V_{CC} + 0.5$	V

What gives for  $V_{CC} = 5\text{V}$ :

- logic low (LOW) in entry level:  $low = 0.3 \times 5\text{V} VCC = 0.3 = 1.5\text{ Volts}$ .

A voltage between -0.5 and less than 1.5V is considered to be a low level.

- Logic (HIGH) high level input:  $HIV = 0.6 \times VCC = 0.6 \times 5\text{V} = 3.0\text{ Volts}$ .

A tension above 3V and lower than 5.5V is considered to be a high level.

-The intermediate zone 1, 5V - 3V is indeterminate and must be avoided.

=> LOW and HIGH level have pretty tolerant values.

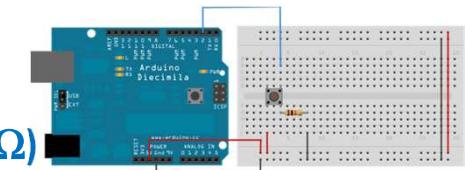
# DEFINITIONS OF DIGITAL PINS

3 possible definitions with `pinMode ()` instruction:

- `INPUT`,
- `INPUT_PULLUP` and
- `OUTPUT`

- Pins configured as INPUTS:

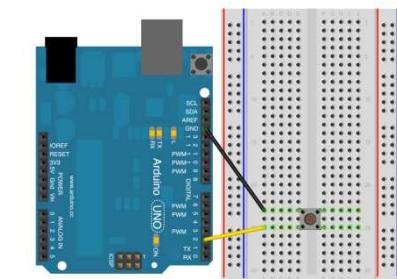
- `INPUT`: the Arduino (Atmega) pins are configured as input with `pinMode ()` in a State of high impedance (series resistance of  $100\text{ M}\Omega$ )
- Example the: `pinMode (inPin, INPUT);`



However, you probably want that spindle is referenced to the Earth, so 2 solutions:

- either put a pull-down resistor (resistance to Earth) in the external circuit or
- configure the pin entered as `INPUT_PULLUP`, which has a pull-up resistance

- `INPUT_PULLUP`: the Atmega on the Arduino chip has an internal pull-up resistance. Beware, since pull-up => this inverts the behavior of the pin, since HIGH means that the sensor is "off", and LOW means that the sensor is activated.
- Example: `pinMode (2, INPUT_PULLUP);`

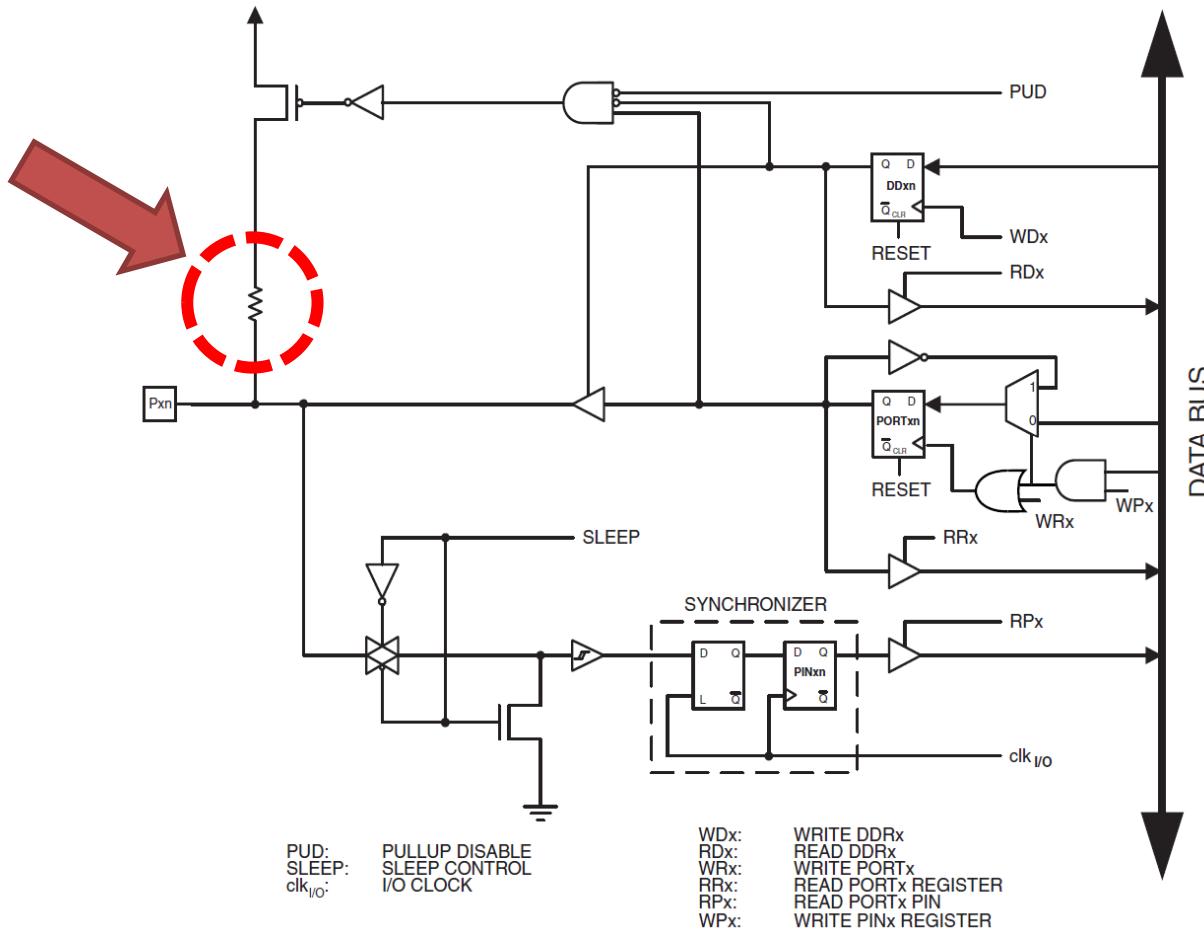


Pins configured as OUTPUTS:

- OUTPUT: pins configured as output with `pinMode ()` are supposed to be in a State of low impedance. This means that they can provide a significant amount of current to other circuits.  
Example: `pinMode (2, OUTPUT);`

# TO GO FURTHER

Figure 14-2. General Digital I/O<sup>(1)</sup>



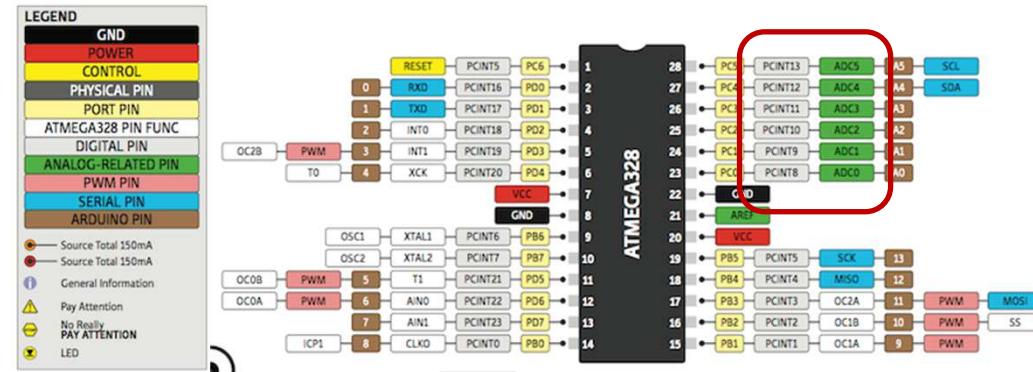
## IF YOU NEED OTHER DIGITAL PINS

The ATmega of the Arduino has 6 channels analog-digital (A/D) (8 in CMS version), for 10-bit resolution while the main function of the analog pins is to read sensors via analog INPUTS

They have also all features of general use of the input/output (GPIO)

⇒ the same pins digital (0-13).

⇒ if a user needs more input/output digital pins: he can use the analog pins as GPIO.



HOW DOES DO? Use the alias A0 (for the analog input 0), A1, etc...

For example, to use the pin A0 in digital output, the code looks like this:

`pinMode (A0, OUTPUT);`

`digitalWrite (A0, HIGH); //put pullup on the analog pin 0`

pull-up resistance is then enabled

Attention:

1 - the other analog pins also have a pull-up resistance, which works like the digital pins. They are activated by the command:

`digitalWrite (A0, HIGH); //pullup put on the analog pin 0 even if the PIN is declared as input.`

2. activation of a pullup will affect the values reported by the `analogRead()` function, used on the C PORT of analog pines (ATTENTION).

# TP1A

# MY FIRST PROGRAM: BLINKING LED

LEDs (light emitting diodes) are widely used nowadays.

Let's start with something very simple:  
turn on and turn off, several times => It is the  
'HELLO WORLD' of µ-controllers!

```
/* Blink
 *Turns on an LED on for one second, then off for one second, repeated
 *The circuit: LED connected from digital pin 13 to ground.
 */
```

```
int ledPin = 13; // LED connected to digital pin 13

// The setup() method runs once, when the sketch starts

void setup() {
    pinMode(ledPin, OUTPUT); // initialize the digital pin as an output:
}

// the loop() method runs over and over again, as long as the Arduino has
power

void loop()
{
    digitalWrite(ledPin, HIGH); // set the LED on
    delay(1000); // wait for a second
    digitalWrite(ledPin, LOW); // set the LED off
    delay(1000); // wait for a second
}
```



TLUR6400, TLUR6401

Vishay Semiconductors

Universal LED in Ø 5 mm Tinted Diffused Package



## PRODUCT GROUP AND PACKAGE DATA

- Product group: LED
- Package: 5 mm
- Product series: standard
- Angle of half intensity: ± 30°

## FEATURES

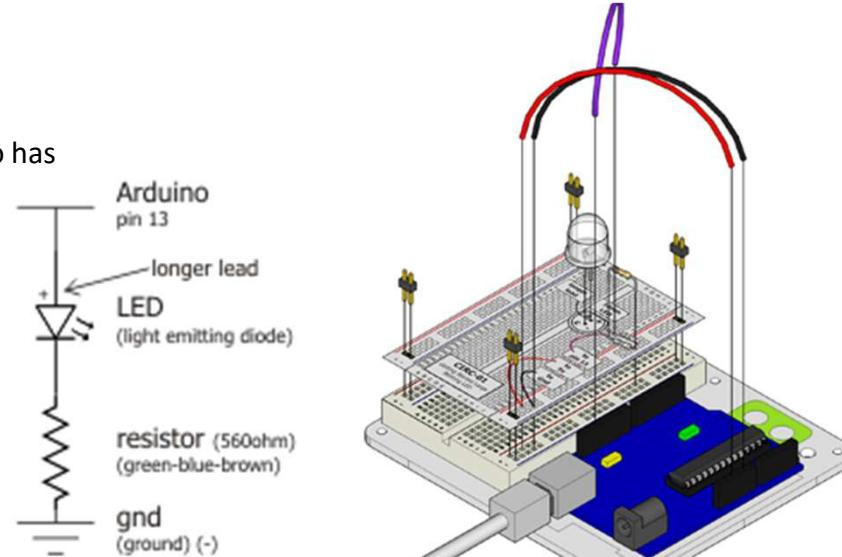
- For DC and pulse operation
- Luminous intensity categorized
- Standard T-1 1/4 package
- TLUR640, without stand-offs
- Material categorization:  
For definitions of compliance please see [www.vishay.com/doc?99912](http://www.vishay.com/doc?99912)



## APPLICATIONS

- General indicating and lighting purposes

PART	COLOR	LUMINOUS INTENSITY (mcd)			at If (mA)	WAVELENGTH (nm)			at If (mA)	FORWARD VOLTAGE (V)			at If (mA)	TECHNOLOGY
		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		MIN.	TYP.	MAX.		
TLUR6400	Red	4	15	-	10	-	630	-	10	-	2	3	20	GaAsP on GaAs
TLUR6401	Red	4	15	32	10	-	630	-	10	-	2	3	20	GaAsP on GaAs



<http://www.oomlout.com/a/products/ardx/circ-01>

<= BACK

# Behind the scene

# USEFUL INSTRUCTIONS: BLINK by ASM

## 2 — Ten (10) Asm Instructions used:

[AVR Instruction Set — Manual](#)

### Instruction : Cycle : Descriptions

**ldi** : 1 : *Load Immediate Into*— Loads an 8-bit constant directly to regs.16 to 31.

**cbi** : 1 : *Clear Bit In I/O Register*— Clears a specified bit in an I/O register.

**sbi** : 1 : *Set Bit in I/O Register*— Sets a specified bit in an I/O Register.

**out** : 1 : *Store Register to I/O Location*— Stores data from register Rr in the Register File to I/O Space (Ports, Timers, Configuration Registers, etc.).

**dec** : 1 : *Decrement*— Subtracts one from the contents of register Rd and places the result in the destination register Rd.

**adiw** : 2 : *Add Immediate to Word*— Adds an immediate value (0–63) to a register pair and places the result in the register pair.

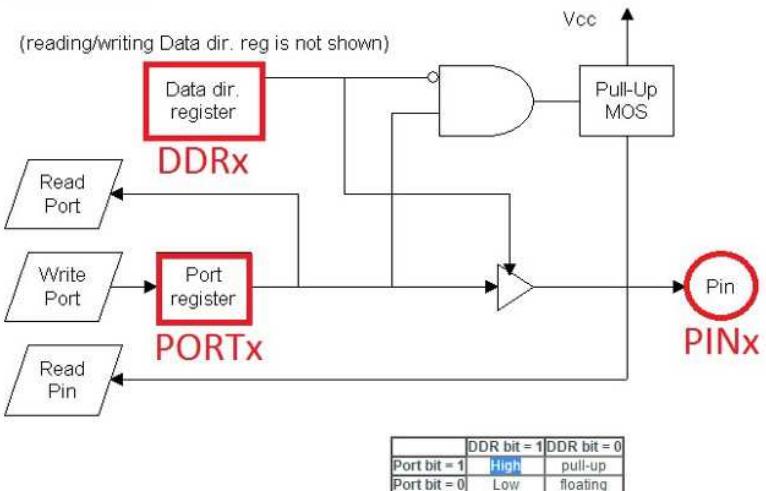
**brne** : 2 : *Branch if Not Equal*— Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is cleared.

**rcall** : 1 : *Relative Call to Subroutine*— Relative call to an address within PC

**ret** : 1 : *Return from Subroutine*— Returns from subroutine.

**rjmp** : 1 : *Relative Jump*— Relative jump to an address.

### AVR I/O Ports:



# BLINK by ASM

## Init your code:

Let's have look at probably the easiest program possible:

```
.ORG 0x000 ; the next instruction has to be written to add 0x0000  
            ; infinite loop  
START: ; this is a label "START"  
        rjmp START ; Relative JuMP to START
```

*When the instruction is executed, the cpu (ALU – Arithmetic Logic Unit) will jump to START. The jump will be repeated over and over, resulting in an infinite loop. Alright, that not a big deal...*

## Do the Math !

Assuming that our Arduino AVR is running at 16 MHz (16 Million clock cycles per second),

### **How long does all this take?**

$T = 1 / F$  then  $T = 1 / 16 \times 10^6 = 0,0000000625$  seconds, not precisely **0,06 us**

**That's pretty fast! We can not see at this frequency the blinking of the LED !!!**

### **Remark:**

We observed experimentally that human eye can't make out a difference in the picture frame if it appears for less than 16ms to 13ms (0,016-0,013s).

Hence purely based on this, we can say sampling frequency is 60Hz to 80Hz (about 0,06 & 0,08 MHz).

## Do the trick!

For example: let's take a LED 0,5s ON and 0,5s OFF: at  $F=16$  MHz, we need **8 000 0000 cycles** cycles  
(1 cycle = 0,0000000625 seconds at 16MHz => for 0,5s this makes  $0,5 / 0,0000000625s$ )

**How to achieve all this cycles if I can count only up to 255 (remember, we have an 8-bit chip)... magic?**

For this we will have to implement **two loops**: **inner and outloop**.

First registers can be used in pairs, allowing to work with values from 0 to 65 535. That's a **word (16 bits)**.

The following piece of code clears registers 24 and 25 and increments them in a loop until they overflow to zero again.

When that condition occurs, the loop doesn't go around again. Let's go ahead!

**clr r24** ; clr needs 1 cycle

**clr r25** ; clr needs 1 cycle

**DELAY\_05:** ; we need .5s delay

**adiw r24, 1** ; adiw needs 2 cycles and

**brne DELAY\_05** ; brne needs 2 cycles if the branch is done ; and 1 otherwise

The inner loop is treated like one BIG instruction needing **262145 clock cycles**

**Why ?**

Every time the registers don't overflow the loop takes **adiw(2) + brne(2) = 4 cycles**.

The loop takes 0xFFFF (65 535) times before the overflow occurs.

The next time the loop only needs 3 cycles, because no branch is done.

**This adds up to  $4 * 65\ 535$ (looping) + 3(overflow) + 2(clr) = 262 145 cycles.**

***This is still not enough:  $8\ 000\ 000 / 262\ 145 \sim 30.51$ .***

**=> The “outer” loop will be down-counting from 31 to zero using r16.**

# BLINK by ASM

ldi r16, 31

OUTER\_LOOP: ; outer loop label  
ldi r24, 0 ; clear register 24  
ldi r25, 0 ; clear register 25

DELAY\_05: ; the loop label  
adiw r24, 1 ; “add immediate to word”: r24:r25 are  
; incremented  
brne DELAY\_05  
dec r16 ; decrement r16  
brne OUTER\_LOOP ; load r16 with 8

The overall loop needs:  
262 145 (inner loop) + 1 (dec) + 2 (brne)  
= 262 148 \* 31 = 8 126 588 cycles.

⇒ this is more like what we want,  
**BUT** 126 588 cycles too long.

This is where the **fine-tuning** comes in : **we need to change the initial value of r24:r25**.

The outer loop is executed 31 times and includes the “big-inner-loop-instruction”.

We have to **subtract some cycles** from the inner loop: **126 588 / 31 = 4 083 cycles per inner loop**.

This is what the inner loop has to be shorter.

Every iteration of the inner loop takes 4 cycles (the last one takes 3 but that's not so important), so let's divide those **4 083 by 4 = 1 020.8 or 1 021 less iterations**.

**This is our new initialisation value for r24:r25! =>**

**ldi r24, low(1021)** ; load registers r24:r25 with 1021, our new ; init value  
**ldi r25, high(1021)**

=> The result is 8 000 000 clock cycles!

# BLINK by ASM

```
.ORG 0x0000          ; the next instruction has to be written to
                      ; address 0x0000
rjmp START           ; the reset vector: jump to "main"

START:ldi r16, low(RAMEND)      ; set up the stack
out SPL, r16
ldi r16, high(RAMEND)
out SPH, r16
ldi r16, 0xFF          ; load register 16 with 0xFF (all bits 1)
out DDRB, r16           ; write the value in r16 (0xFF) to Data ; Direction Register B

LOOP:
    sbi PortB, 5   ; switch off the LED
    rcall delay_05 ; wait for half a second
    cbi PortB, 5   ; switch it on
    rcall delay_05 ; wait for half a second
    rjmp LOOP     ; jump to loop

DELAY_05:             ; the subroutine:
    ldi r16, 31     ; load r16 with 31

OUTER_LOOP:          ; outer loop label
    ldi r24, low(1021)    ; load registers r24:r25 with 1021, our new ; init value
    ldi r25, high(1021)   ; the loop label

DELAY_LOOP:           ; "add immediate to word": r24:r25 are ; incremented
    adiw r24, 1          ; if no overflow ("branch if not equal"), go ; back to "delay_loop"

    brne DELAY_LOOP
    dec r16              ; decrement r16
    brne OUTER_LOOP      ; and loop if outer loop not finished
    ret                  ; return from subroutine
```

# FIBONACCI by ASM

## 1 - Exemple : Fichier Fibonacci

```
//FIBONACCI
//La suite des nombres de Fibonacci est la seule suite ayant les deux propriétés:
//Un+1 = Un + Un-1 et lim Un+1/Un = constante
//Le rapport entre deux nombres de Fibonacci consécutifs tend vers le nombre d'or. //lim Un+1/Un = Phi = (1+sqrt(5))/2 = 1.618034

int x, y , z;

void setup() {
    // put your setup code here, to run once:
x=0;
y=1;
Serial.begin(9600);//Affiche la suite des nombres
}

void loop() {
while (x<255) {
    Serial.println(x);//Affiche la suite des nombres
    z=y+x;
    x=y;
    y=z;
}
x=0; //Réinitialisation
}
```

## 2 – Version assemblée :

Récupérer le fichier .hex dans le répertoire de travail Arduino: C:\Users\Admin\AppData\Local\Temp\arduino\_build\_669485\Fibonacci.ino.hex"  
Pour s'afficher ce lien, il faut activer les options de compilations dans les préférences : **Afficher les résultats détaillés pendant la compilation**

### Fibonacci.ino.hex :

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

```
00000000 3A 31 30 30 30 30 30 30 30 30 43 39 34 33 35 30 :100000000C94350
00000010 30 30 43 39 34 35 44 30 30 30 43 39 34 35 44 30 00C945D000C945D0
00000020 30 30 43 39 34 35 44 30 30 32 34 0D 0A 3A 31 30 00C945D0024..:10
00000030 30 30 31 30 30 30 30 43 39 34 35 44 30 30 30 43 0010000C945D000C
00000040 39 34 35 44 30 30 30 43 39 34 35 44 30 30 30 43 945D000C945D000C
00000050 39 34 35 44 30 30 45 43 0D 0A 3A 31 30 30 30 32 945D00EC..:10002
00000060 30 30 30 30 43 39 34 35 44 30 30 30 43 39 34 35 0000C945D000C945
00000070 44 30 30 30 43 39 34 35 44 30 30 30 43 39 34 35 D000C945D000C945
...
...
```

# DISASSEMBLED FILE

## 2 - Utilisation AVRDisassembler : <https://github.com/christophediericx/AVRDisassembler>

- Télécharger la version x64 Windows
- Dans l'invite de commande CMD de windows :
- Exemple : **AVRDisassembler.exe -i Fibonacci.ino.hex**

D:\Mes\_Documents\Mes\_Fichiers\Logiciels\_Microsoft\Word\INSA\MOSH\Ajouts\_2018\AVRDisassembler-0.2.0-Windows-x64\AVRDisassembler-0.2.0-Windows-x64>AVRDisassembler.exe -i Fibonacci.ino.hex

Résultat :

```
0000: 0C-94-35-00 jmp 0x6a ; Jump
0004: 0C-94-5D-00 jmp 0xba ; Jump
0008: 0C-94-5D-00 jmp 0xba ; Jump
000C: 0C-94-5D-00 jmp 0xba ; Jump
0010: 0C-94-5D-00 jmp 0xba ; Jump
0014: 0C-94-5D-00 jmp 0xba ; Jump
0018: 0C-94-5D-00 jmp 0xba ; Jump
001C: 0C-94-5D-00 jmp 0xba ; Jump
0020: 0C-94-5D-00 jmp 0xba ; Jump
0024: 0C-94-5D-00 jmp 0xba ; Jump
0028: 0C-94-5D-00 jmp 0xba ; Jump
002C: 0C-94-5D-00 jmp 0xba ; Jump
0030: 0C-94-5D-00 jmp 0xba ; Jump
0034: 0C-94-5D-00 jmp 0xba ; Jump
0038: 0C-94-5D-00 jmp 0xba ; Jump
003C: 0C-94-5D-00 jmp 0xba ; Jump
0040: 0C-94-BC-01 jmp 0x378 ; Jump
0044: 0C-94-5D-00 jmp 0xba ; Jump
0048: 0C-94-2C-02 jmp 0x458 ; Jump
004C: 0C-94-06-02 jmp 0x40c ; Jump
0050: 0C-94-5D-00 jmp 0xba ; Jump
0054: 0C-94-5D-00 jmp 0xba ; Jump
0058: 0C-94-5D-00 jmp 0xba ; Jump
005C: 0C-94-5D-00 jmp 0xba ; Jump
0060: 0C-94-5D-00 jmp 0xba ; Jump
0064: 0C-94-5D-00 jmp 0xba ; Jump
0068: 2E-03 fmul r18, r22 ; Fractional Multiply Unsigned
006A: 11-24 clr r17 ; Clear Register
006C: 1F-BE out 0x3f, r1 ; Store Register to I/O Location
006E: CF-EF ldi r28, 0xff ; Load Immediate
0070: D8-E0 ldi r29, 0x08 ; Load Immediate
0072: DE-BF out 0x3e, r29 ; Store Register to I/O Location
0074: CD-BF out 0x3d, r28 ; Store Register to I/O Location
0076: 11-E0 ldi r17, 0x01 ; Load Immediate
0078: A0-E0 ldi r26, 0x00 ; Load Immediate
007A: B1-E0 ldi r27, 0x01 ; Load Immediate
007C: E4-E1 ldi r30, 0x14 ; Load Immediate
007E: F7-E0 ldi r31, 0x07 ; Load Immediate
0080: 02-C0 rjmp .+4 ; Relative Jump
0082: 05-90 lpm r0, Z+ ; Load Program Memory
0084: 0D-92 st X+, r0 ; Store Indirect From Register to Data Space
0086: A6-31 cpi r26, 0x16 ; Compare with Immediate
0088: B1-07 cpc r27, r17 ; Compare with Carry
008A: D9-F7 brbc 1, -10 ; Branch if Bit in SREG is Cleared
008C: 21-E0 ldi r18, 0x01 ; Load Immediate
008E: A6-E1 ldi r26, 0x16 ; Load Immediate
...
...
```

## REFERENCES

<https://medium.com/jungletronics/meeting-assembly-hello-world-arduino-blinking-code-330386652309>

<https://gist.github.com/mhitza/8a4608f4dfdec20d3879>

<https://onlinedisassembler.com/static/home/>

<https://www.instructables.com/id/Command-Line-Assembly-Language-Programming-for-Ard/>

<https://www.instructables.com/id/Command-Line-Assembly-Language-Programming-for-Ard-1/>

<https://www.instructables.com/id/Command-Line-Assembly-Language-Programming-for-Ard-2/>

<https://www.instructables.com/id/Command-Line-Assembly-Language-Programming-for-Ard-3/>

<https://www.instructables.com/id/Command-Line-Assembly-Language-Programming-for-Ard-4/>

## DEBUGGING WITH THE HELP OF THE SERIAL PORT (VIA USB)

Once your program has been downloaded in Arduino, the program operates independently. Everyone is happy if it works as expected... However, if this is not the case, start to understand what is happening.

Must be creative and develop assumptions... and I must admit that it is a tedious task because the board has no debugger or debugging message display!

Yet there is the serial port... so it is possible to send messages (message from debugging) since Arduino and reading them on the PC.

**Arduino serial port and USB Port :**

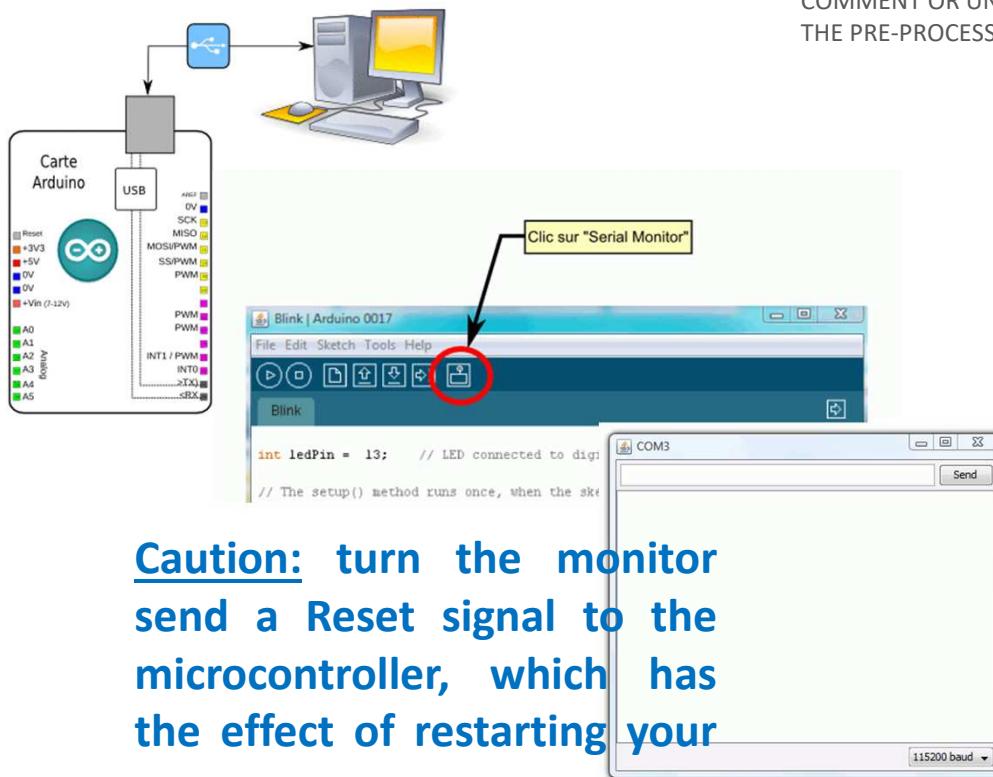
it is possible to communicate via the serial port of Arduino (what the microcontroller believes) but actually, all messages are returned via the USB (cf. FTDI) device to the PC (and its USB port).  
Just listen to the USB port.

**Caution: pins 0 and 1 for communication book series:**

during a serial, the Pin0 connection and 1 are automatically re-configured (pin0) RX & TX (pin1).  
=> it becomes impossible to use pins 0 & 1 for other purposes.

***Note: it is also this same connection USB is also used to power the board.***

# DISPLAYS TEXT ON THE SERIAL PORT => DEBUGGING



**Caution:** turn the monitor  
send a Reset signal to the  
microcontroller, which has  
the effect of restarting your  
program.

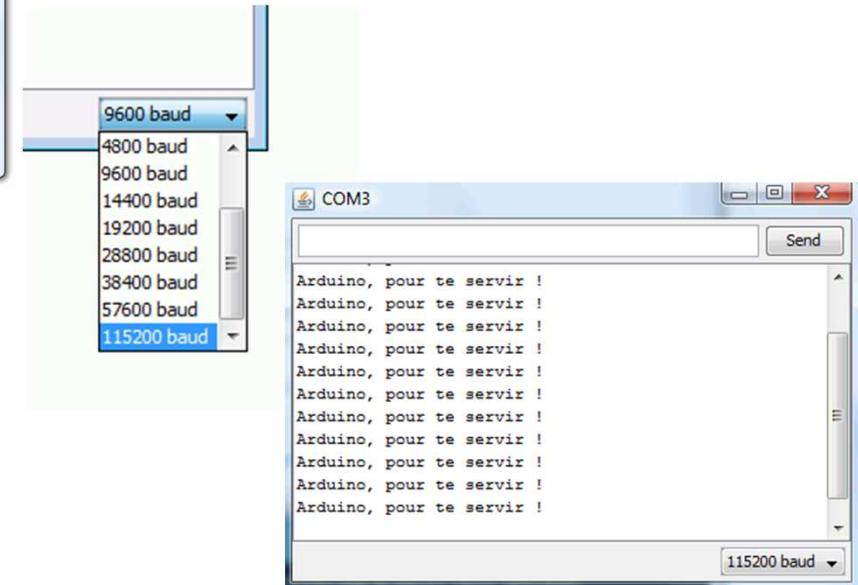
COMMENT OR UNCOMMENT  
THE PRE-PROCESSOR DIRECTIVES ➔

```
#define DEBUG

#ifndef DEBUG
    unsigned int mavar =255;
#endif

void setup() {
#ifndef DEBUG
    Serial.begin(9600); //Serial port init
#endif
}

void loop() {
#ifndef DEBUG
    int droite=digitalRead(CNY_D);
    Serial.print(" G:"); Serial.print(gauche);
    Serial.print(", BP:");
    Serial.println(digitalRead(BP));
#endif
} // fin de loop
```



# TP1B

# AFFICHE DU TEXTE SUR LE PORT SERIE => DEBUGGAGE

```
/* Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 * The circuit: LED connected from digital pin 13 to ground.
 * Note: On most Arduino boards, there is already an LED on the board
 * connected to pin 13, so you don't need any extra components for this example.
```

## + Displays the status of the LEDs in Serial monitor

\*Created 1 June 2005, By David Cuartielles  
<http://arduino.cc/en/Tutorial/Blink>

```
*/
```

```
int ledPin = 13; // LED connected to digital pin 13
```

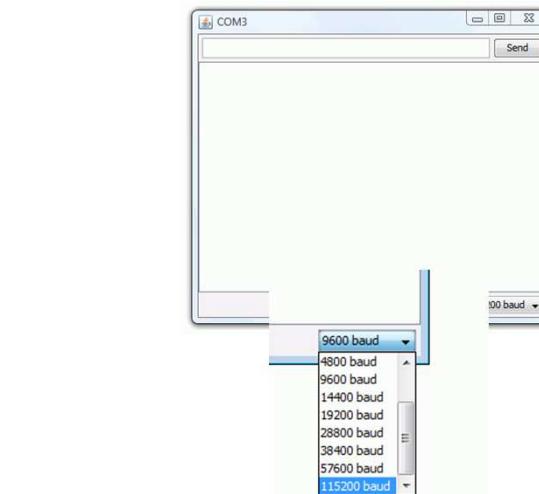
```
// The setup() method runs once, when the sketch starts
```

```
void setup() {
  // opens serial port, sets data rate to 9600 bps
  Serial.begin(9600);
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}
```

```
// the loop() method runs over and over again, as long as the Arduino has power
```

```
void loop()
{
  digitalWrite(ledPin, HIGH); // set the LED on
  Serial.print("LED Status: ON\n");
  delay(1000); // wait for a second

  digitalWrite(ledPin, LOW); // set the LED off
  Serial.print("LED Status: OFF\n");
  delay(1000); // wait for a second
}
```



# COMMANDER L'ARDUINO PAR LE PORT SERIE: EXEMPLE

## Exercice:

Faites un montage rudimentaire où on essaye de détecter la pression sur le bouton (Pin 2).

Un message de Debug est envoyé via le port série lorsque l'utilisateur enfonce ou relâche le bouton.

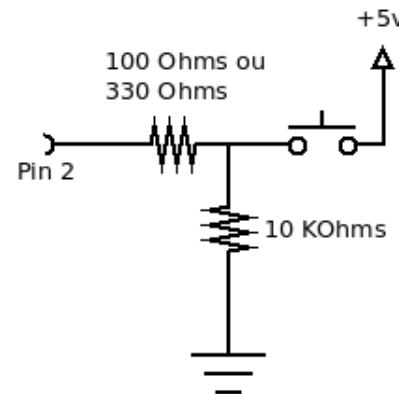
```
/* * Script montrant l'usage de la communication série
* pour débugger du code */
int pinBouton = 2;
int etatBouton;
int lecture1, lecture2;

void setup(){
pinMode( pinBouton, INPUT ); // Bouton d'entrée
etatBouton = digitalRead( pinBouton );
Serial.begin(9600); // Démarrage de la communication série
Serial.print("Démarrage du programme,\n");
Serial.print("Presser le bouton.\n");
}

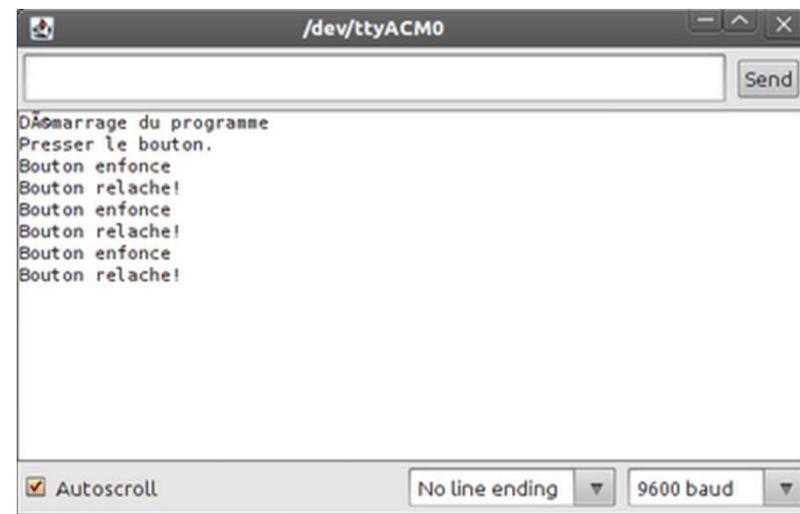
void loop(){
// --- Lecture du bouton avec déparasitage logiciel ---
// Si pas de parasite, lecture1 égal lecture2
lecture1 = digitalRead( pinBouton ); // première lecture du bouton
delay( 10 ); // attendre 10 milli-secondes
lecture2 = digitalRead( pinBouton ); // deuxième lecture du bouton
// changement d'état ?
if( (lecture1==lecture2) && (lecture1 != etatBouton) )
changeEtatBouton( lecture1 );

void changeEtatBouton( int nouvelEtat ){
if( nouvelEtat == LOW ) {
  Serial.print( "Bouton relache!\n" );
} else {
  Serial.print( "Bouton enfonce\n" );
}
etatBouton = nouvelEtat;
}
```

<http://arduino103.blogspot.com/2011/05/debugger-laide-du-port-serie-via-usb.html>



Matériel du kit:  
Bouton poussoir  
Résistances  
=> Pont diviseur



<= BACK



48

## SEND CHARACTERS TO THE SERIAL PORT => DEBUGGING

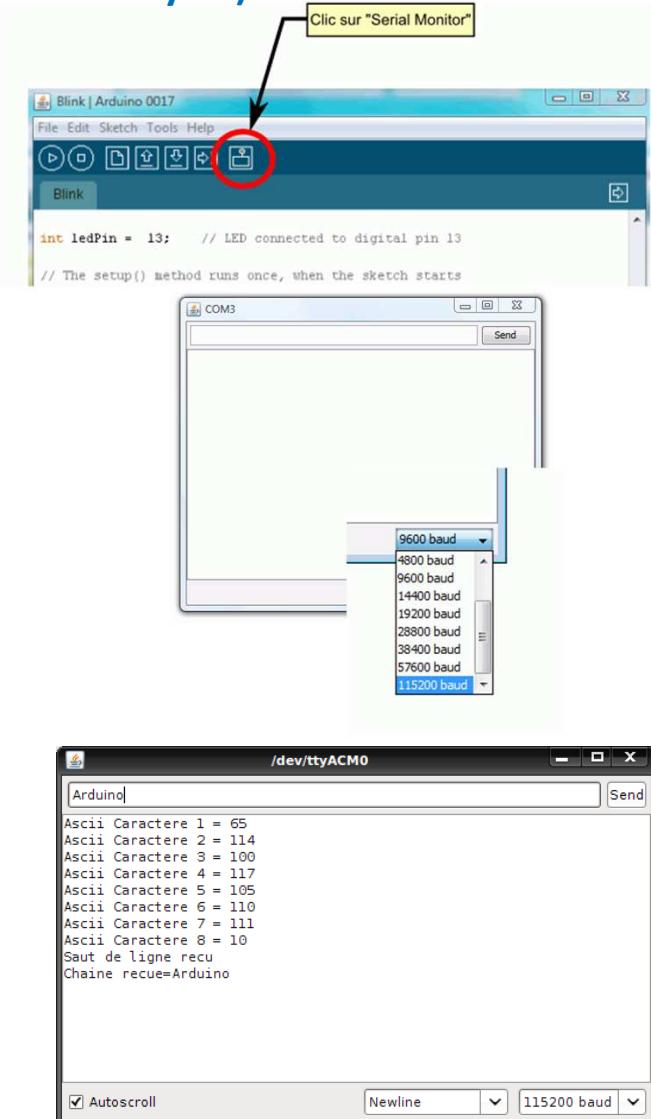
Get the number of bytes (characters) available to be read from the serial port. This is data that are already arrived and stored in the buffer of reception series (which holds 128 bytes).

```
int incomingByte = 0; // for incoming serial data
```

```
void setup() {  
Serial.begin(9600); // opens serial port, sets data rate to 9600 bps  
}
```

```
void loop() {  
  
// send data only when you receive data:  
if (Serial.available() > 0) {  
    // read the incoming byte:  
    incomingByte = Serial.read();  
  
    // say what you got:  
    Serial.print("I received: ");  
    Serial.println(incomingByte, DEC);  
}  
}
```

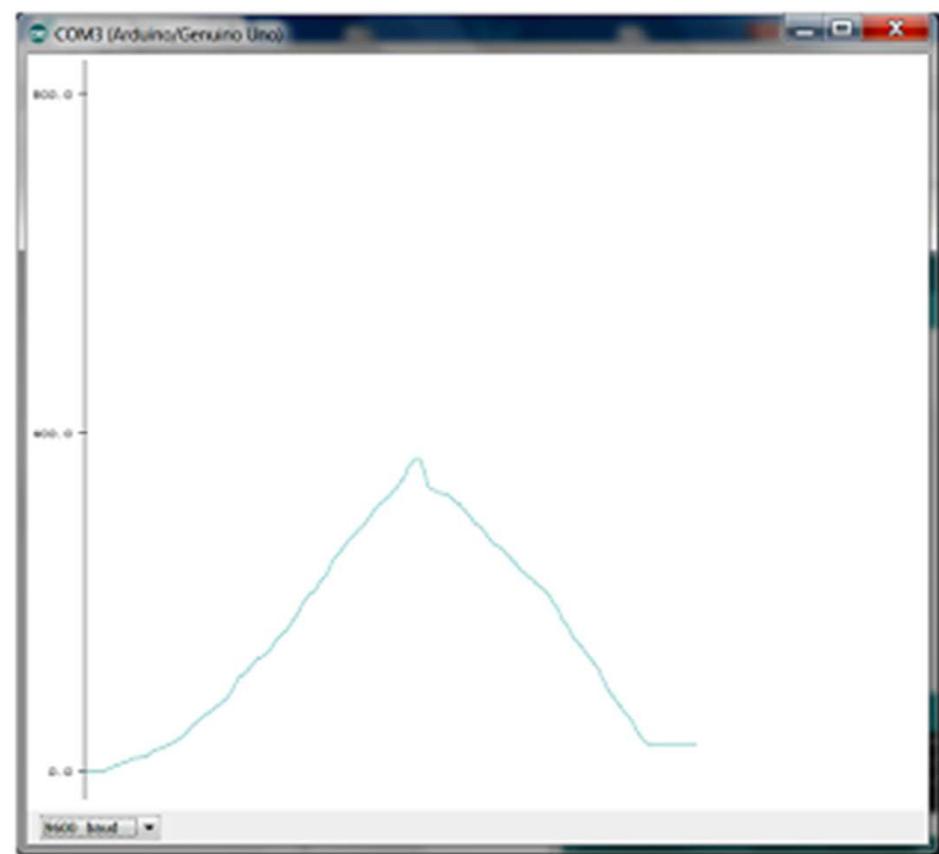
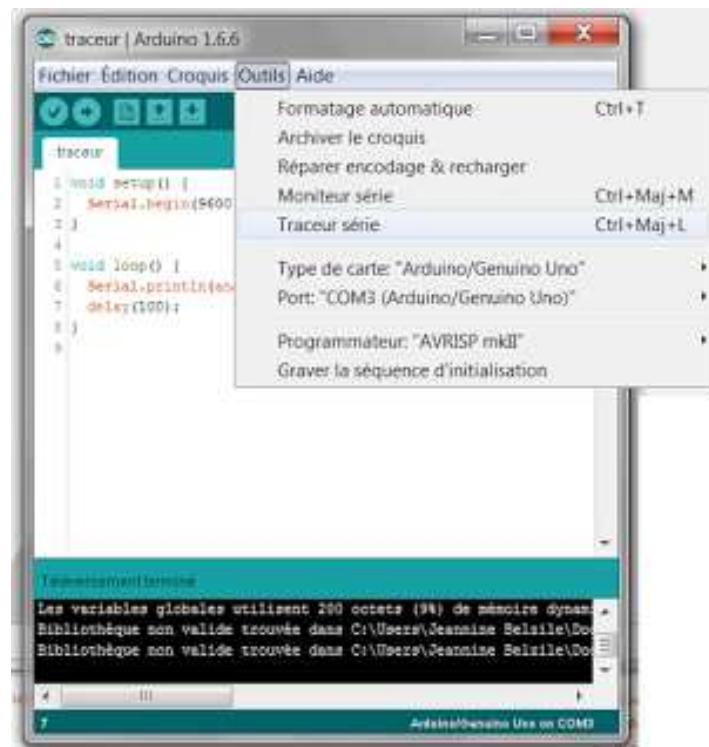
<http://www.arduino.cc/en/Serial/Available>



# SERIAL TRACER – (IDE version > 1.6.6)



```
1 void setup() {  
2   Serial.begin(9600);  
3 }  
4  
5 void loop() {  
6   Serial.println(analogRead(A0));  
7   delay(100);  
8 }  
9
```



The series plotter displays a fixed number of data. It is important to set your `delay()` in the loop.

# TP1C

# MANAGING THE ARDUINO FROM THE SERIAL PORT: EXAMPLE

Exercise: Pick up the Assembly of the first program and provoke the ignition or the extinction of the LED by sending the character 'y' and 'n' respectively.

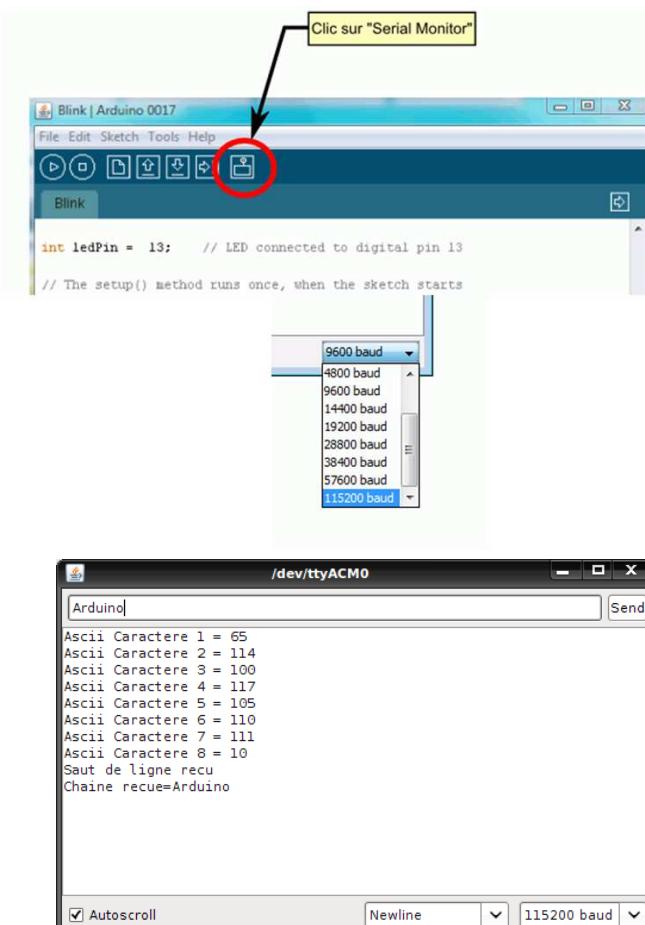
```
/* Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 * The circuit: LED connected from digital pin 13 to ground.
 * Created 1 June 2005, By David Cuartielles
 * http://arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13; // LED connected to digital pin 13
int incomingByte = 0; //serial char

// The setup() method runs once, when the sketch starts
void setup() {
    // opens serial port, sets data rate to 9600 bps
    Serial.begin(9600);
    // initialize the digital pin as an output:
    pinMode(ledPin, OUTPUT);
    digitalWrite(ledPin, LOW);
}

// the loop() method runs over and over again, as long as the Arduino has power
void loop()
{
    if (Serial.available() > 0) {
        // read the incoming byte:
        incomingByte = Serial.read();

        if (incomingByte == 'y') {
            digitalWrite(ledPin, HIGH); // set the LED ON
        }
        else if (incomingByte == 'n') {
            digitalWrite(ledPin, LOW); // set the LED off
        }
    }
    delay(250); // wait for a second
}
```



<= BACK

# AND IF THE SERIAL PORT IS OCCUPIED BY A DEVICE ?

How to debug? => "Software" series library : SoftwareSerial

*For example: GPS module*

- *Pin 0 module & Pin1 occupied for serial communication  
⇒ the Pin 2 & Pin3 (for example) is defined to display the data from the GPS on the Serial Monitor module*

```
/* Ce programme fait afficher sur le moniteur série de l'environnement de développement toutes les trames émises par le récepteur GPS */  
  
#include <SoftwareSerial.h>  
  
SoftwareSerial PortSerie(2,3); // Tx du GPS sur 2 de l'Arduino, Rx sur 3 mais non utilisé  
  
void setup()  
{  
    Serial.begin(115200); // Initialisation du port série matériel  
    PortSerie.begin(4800); // Initialisation du port série logiciel  
    Serial.println("Affichage des trames du GPS");  
    Serial.println();  
}  
  
void loop()  
{  
    while (PortSerie.available()) // Tant que des données GPS sont disponibles  
    {  
        char c = PortSerie.read(); // Lecture de ces données sur le port série logiciel  
        Serial.write(c); // Affichage sur le port série matériel  
    }  
}
```



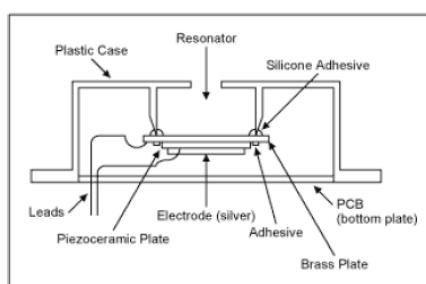
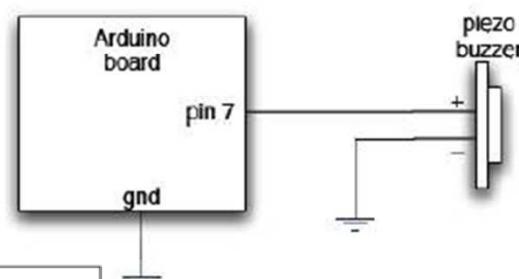
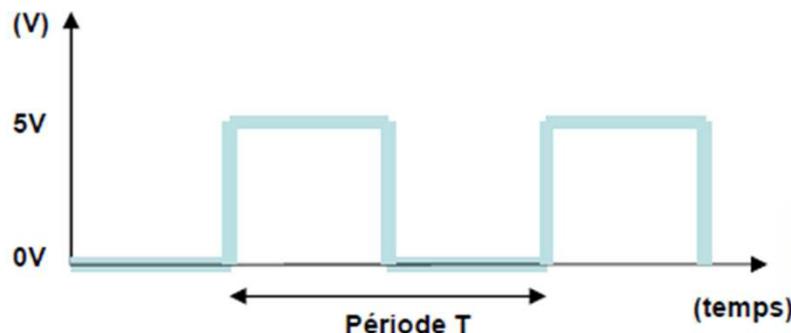
*Of course, the display on the Serial monitor must go slower than the acquisition of data from the module!*

# WIRE A BUZZER (PIEZO SPEAKER)

The buzzer cable on a digital output.

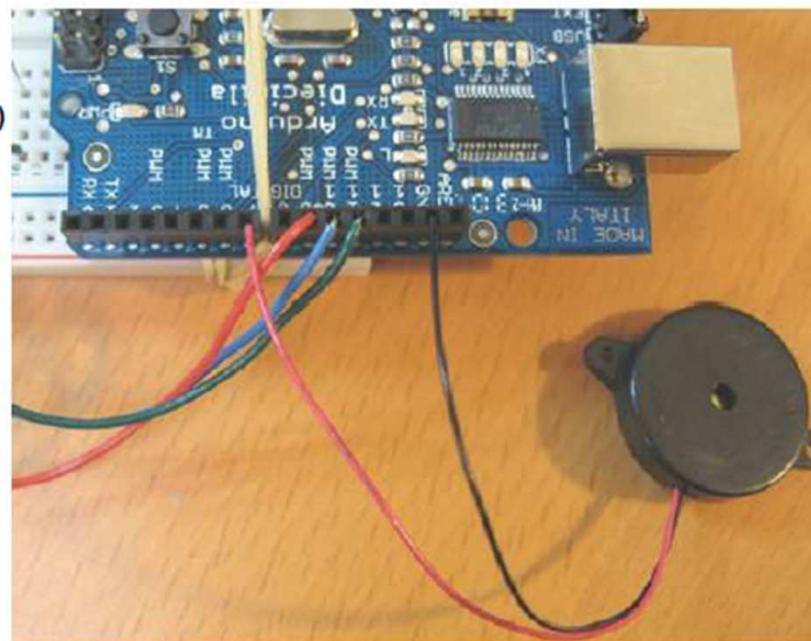
We then send it a periodic signal which is varying the frequency according to the note you wish to play:

example: A note is a signal of frequency f 440Hz



	0	1	2	3	4	5	6	7	8	9
do	32.703	65.406	130.81	261.63	523.25	1046.5	2093.	4186.	8372.	16744.
do#	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9	8869.8	17740.
ré	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6	9397.3	18795.
ré#	38.891	77.782	155.56	311.13	622.25	1244.5	2489.	4978.	9956.1	19912.
mi	41.203	82.407	164.81	329.63	659.26	1318.5	2637.	5274.	10548.	21096.
fa	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7	11175.	22351.
fa#	46.249	92.499	185.	369.99	739.99	1480.	2960.	5919.9	11840.	23680.
sol	48.999	97.999	196.	392.	783.99	1568.	3136.	6271.9	12544.	25088.
sol#	51.913	103.83	207.65	415.3	830.61	1661.2	3322.4	6644.9	13290.	26580.
la	55	110	220	440	880	1760	3520	7040	14080	28160
la#	58.27	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6	14917.	29834.
si	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1	15804.	31609.

$$f = 1/T ; T=1/f ; f : \text{fréquence} ; T : \text{période}.$$



# WIRE A BUZZER (PIEZO SPEAKER) - PROGRAM

/\* Melody (copyleft) 2005 D. Cuartielles for K3

\* This example uses a piezo speaker to play melodies. It sends a square wave of the appropriate frequency to the piezo, generating  
\* the corresponding tone. The calculation of the tones is made following the mathematical operation: timeHigh = period / 2 = 1 / (2 \* toneFrequency)

\* where the different tones are described as in the table:

\* note frequency period timeHigh

\* c 261 Hz 3830 1915

\* d 294 Hz 3400 1700

\* e 329 Hz 3038 1519

\* f 349 Hz 2864 1432

\* g 392 Hz 2550 1275

\* a 440 Hz 2272 1136

\* b 493 Hz 2028 1014

\* C 523 Hz 1912 956

\* http://www.arduino.cc/en/Tutorial/Melody

\*/

int speakerPin = 9;

int length = 15; // the number of notes

char notes[] = "ccggaagffeeddc "; // a space represents a rest

int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 4 };

int tempo = 300;

**void setup()**

pinMode(speakerPin, OUTPUT);

}

**void loop()**

for (int i = 0; i < length; i++) {  
 if (notes[i] == ' ') { delay(beats[i] \* tempo); // rest  
 } else { playNote(notes[i], beats[i] \* tempo);  
 }  
 delay(tempo / 2); // pause between notes  
}

}

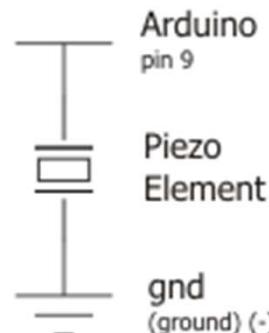
**void playTone(int tone, int duration)**

for (long i = 0; i < duration \* 1000L; i += tone \* 2) {  
 digitalWrite(speakerPin, HIGH); delayMicroseconds(tone);  
 digitalWrite(speakerPin, LOW); delayMicroseconds(tone);  
}

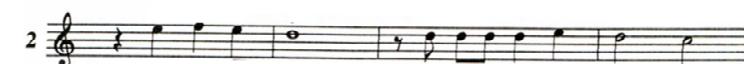
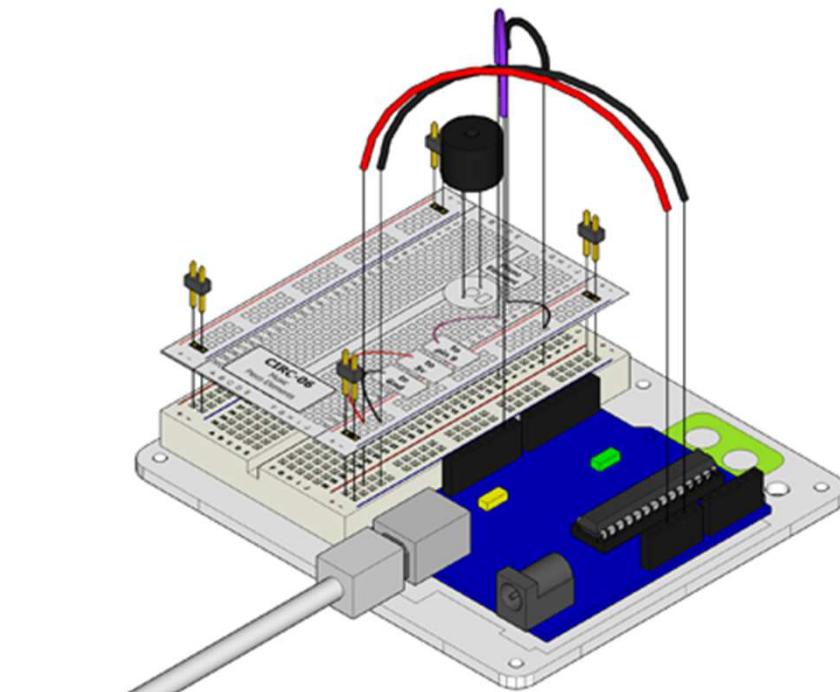
}

**void playNote(char note, int duration)**

char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };  
int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };  
// play the tone corresponding to the note name  
for (int i = 0; i < 8; i++) {  
 if (names[i] == note) {  
 playTone(tones[i], duration);  
 }  
}



<http://www.oomlout.com/a/products/ardx/circ-06>



<= RETOUR

### III – 2 ANALOG INPUTS

### III – 2 ANALOG INPUTS

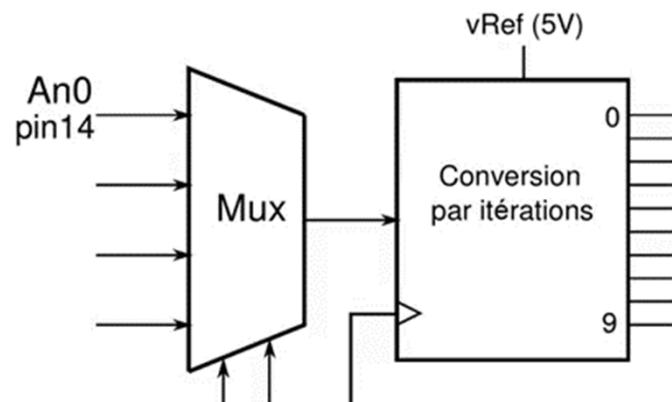
# ANALOG INPUTS

Unlike the digital signal that can take only two different States, an analog signal can take an infinite number of values, such as a voltage that varies progressively from 0V to 5V.

The Arduino works in digital, microcontroller includes only the '0' and '1'.

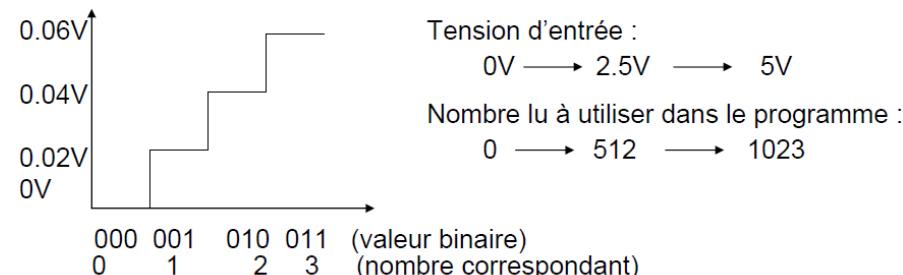
Analog inputs A0 to A5 are therefore equipped with analog-to-digital converters that convert a tension in a suite of '0' and '1' card did match a number of 0 to 1023 (encoding 10-bit)

You can retrieve the information from a sensor.



Valeur\_lue = analogRead(AN0);

```
int valeurLue ; byte valeurUtile;  
valeurLue = analogRead (AN0) ; //10 bits  
map (valeurLue, minA,maxA,minV,maxV) ;
```



AnalogRead(pin) => 10 bits => 1024 values

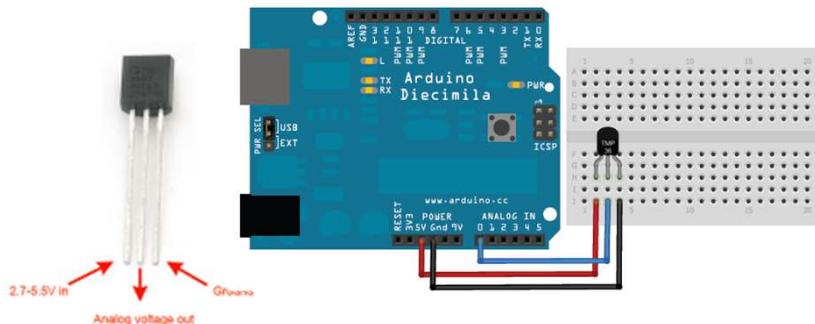
The accuracy depends on the wiring, which is difficult to guarantee the last 2 bits!

Conversion lasts 110µs (AT328), a flag says when it is completed

# TEMPERATURE SENSOR

The TMP36 is a sensor that is in the form of a small transistor.

Once powered (between 3 and 5V), he released an analog voltage directly proportional to the T.



## SPECIFICATIONS

$V_s = 2.7 \text{ V to } 5.5 \text{ V}$ ,  $-40^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$ , unless otherwise noted.

Table 1.

Parameter <sup>1</sup>	Symbol	Test Conditions/Comments	Min	Typ	Max	Unit
ACCURACY						
TMP35/TMP36/TMP37 (F Grade)		$T_A = 25^\circ\text{C}$		$\pm 1$	$\pm 2$	$^\circ\text{C}$
TMP35/TMP36/TMP37 (G Grade)		$T_A = 25^\circ\text{C}$		$\pm 1$	$\pm 3$	$^\circ\text{C}$
TMP35/TMP36/TMP37 (F Grade)		Over rated temperature		$\pm 2$	$\pm 3$	$^\circ\text{C}$
TMP35/TMP36/TMP37 (G Grade)		Over rated temperature		$\pm 2$	$\pm 4$	$^\circ\text{C}$
Scale Factor, TMP35		$10^\circ\text{C} \leq T_A \leq 125^\circ\text{C}$		10		$\text{mV}/^\circ\text{C}$
Scale Factor, TMP36		$-40^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$		10		$\text{mV}/^\circ\text{C}$
Scale Factor, TMP37		$5^\circ\text{C} \leq T_A \leq 85^\circ\text{C}$		20		$\text{mV}/^\circ\text{C}$
		$5^\circ\text{C} \leq T_A \leq 100^\circ\text{C}$		20		$\text{mV}/^\circ\text{C}$
		$3.0 \leq V_s \leq 5.5 \text{ V}$				
Load Regulation		$0 \mu\text{A} \leq I_L \leq 50 \mu\text{A}$	6	20		$\text{m}^\circ\text{C}/\mu\text{A}$
		$-40^\circ\text{C} \leq T_A \leq +105^\circ\text{C}$	25	60		$\text{m}^\circ\text{C}/\mu\text{A}$
		$-105^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$	30	100		$\text{m}^\circ\text{C}/\text{V}$
Power Supply Rejection Ratio	PSRR	$3.0 \leq V_s \leq 5.5 \text{ V}$	50			$\text{m}^\circ\text{C}/\text{V}$
Linearity			0.5			$^\circ\text{C}$
Long-Term Stability		$T_A = 150^\circ\text{C}$ for 1 kHz	0.4			$^\circ\text{C}$
SHUTDOWN						
Logic High Input Voltage	$V_{IH}$	$V_s = 2.7 \text{ V}$	1.8			V
Logic Low Input Voltage	$V_{IL}$	$V_s = 5.5 \text{ V}$		400		mV
OUTPUT						
TMP35 Output Voltage		$T_A = 25^\circ\text{C}$	250			mV
TMP36 Output Voltage		$T_A = 25^\circ\text{C}$	750			mV
TMP37 Output Voltage		$T_A = 25^\circ\text{C}$	500			mV
Output Voltage Range			100		2000	mV
Output Load Current	$I_L$		0	50		$\mu\text{A}$
Short-Circuit Current	$I_{SC}$			250		$\mu\text{A}$
Capacitive Load Driving	$C_L$	Note 2 No oscillations <sup>2</sup>	1000	10000	1	pF
Device Turn-On Time		Output within $\pm 1^\circ\text{C}$ , $100 \text{ k}\Omega  100 \text{ pF}$ load <sup>2</sup>	0.5	1		ms
POWER SUPPLY						
Supply Range	$V_s$		2.7	5.5		V
Supply Current	$I_{sv}(\text{ON})$	Unloaded		50		$\mu\text{A}$
Supply Current (Shutdown)	$I_{sv}(\text{OFF})$	Unloaded	0.01	0.5		$\mu\text{A}$

<sup>1</sup> Does not consider errors caused by self-heating.

<sup>2</sup> Guaranteed but not tested.

The main advantages of the TMP36 are:

- 1 - broad range of temperatures (from -40 to + 150 ° C).
- 2 - The output voltage is completely independent of the power of the TMP36.

The TMP36 feature:

Size: box TO-92 3-pin (similar to a transistor)

Output voltage: 0.1V (-40 ° C) to 2.0V (150 ° C) but accuracy decreases after 125 ° C

Scale factor: 10mV/° C

Supply voltage: 2.7V to 5.5V

Load current: 0.05 mA

Convert the analog voltage level:

Attention, the TMP36 allows to measure negative temperatures!

⇒ 0 °C is placed at an offset of 500 mV.

⇒ If  $V_{out} < 500 \text{ mV} \Rightarrow$  negative temperature.

⇒  $T (\text{ }^\circ\text{C}) = (V_{out} \text{ (mV)} - 500) / \text{scale factor} \Rightarrow$

⇒  $T (\text{ }^\circ\text{C}) = (V_{out} \text{ (mV)} - 500) / 10$

Example: If  $V_{out} = 1 \text{ Volts}$ ,  $T (\text{ }^\circ\text{C}) = (1000 - 500) / 10 = 50 \text{ }^\circ\text{C}$

Equivalent to the TMP36: LM35/TMP35 (measure in °C) or LM34/TMP34 (measure in ° F). Attention to the conversion formulae: the LM35 doesn't have the 500mV to offset

# CONSIDERATION OF ACCURACY: TEMPERATURE SENSOR

The analog input takes a value between 0 and 1023 for a voltage between 0 and 5V.

Measurement accuracy:  $5/2^{10}=5/1024=0.0048V$  ( $\sim 4.8\text{ mV}$ ). => precision of the analog input: about  $+/-0.5^\circ\text{C}$  (e.g. for the TMP36:  $4.8(\text{mV})/10(\text{mV}/^\circ\text{C})$ ).

## How to increase the precision of the analog input?

By the `analogReference (type)` function that configures the voltage used for analog input:

**DEFAULT:** default reference = 5V or 3.3V

**INTERNAL** analog: an internal reference 1.1V for ATmega168, ATmega328

**EXTERNAL:** the voltage used by the AREF pin (0 to 5V).

### EXTERNAL reference :

Supply 3.3V (from the Arduino) and using this voltage as reference (AREF pin) for analog readings, it improves the accuracy of reading: indeed, the analog input will evolve from 0 to 1023 for a voltage between 0 and ARef (3.3V ).

=> measurement accuracy:  $3.3/2^{10}=3/1024\sim 3.2\text{mV}$ ) => error is here reduced to  $+/-0.33^\circ\text{C}$ .

### INTERNAL reference :

By feeding into 1.1V, it improves the accuracy of reading:

⇒ indeed, the analog input will evolve from 0 to 1023 for a voltage between 0 and ARef (1.1V).

⇒ measurement accuracy:  $1.1/2^{10}=1.1/1024\sim 1.1\text{mV}$ )

⇒ error is here reduced to  $+/-0.1^\circ\text{C}$ .

⇒ In practice: either `analogReference (EXTERNAL)` or `analogReference (INTERNAL)` in `setup()`.

⇒ *Attention, do not forget to change the line of code calculating the conversion of `analogRead` in tension. The line of code goes from float voltage =  $V * 5.0$ ; to float voltage =  $V * 3.3$ ; or float voltage =  $V * 1.1$*

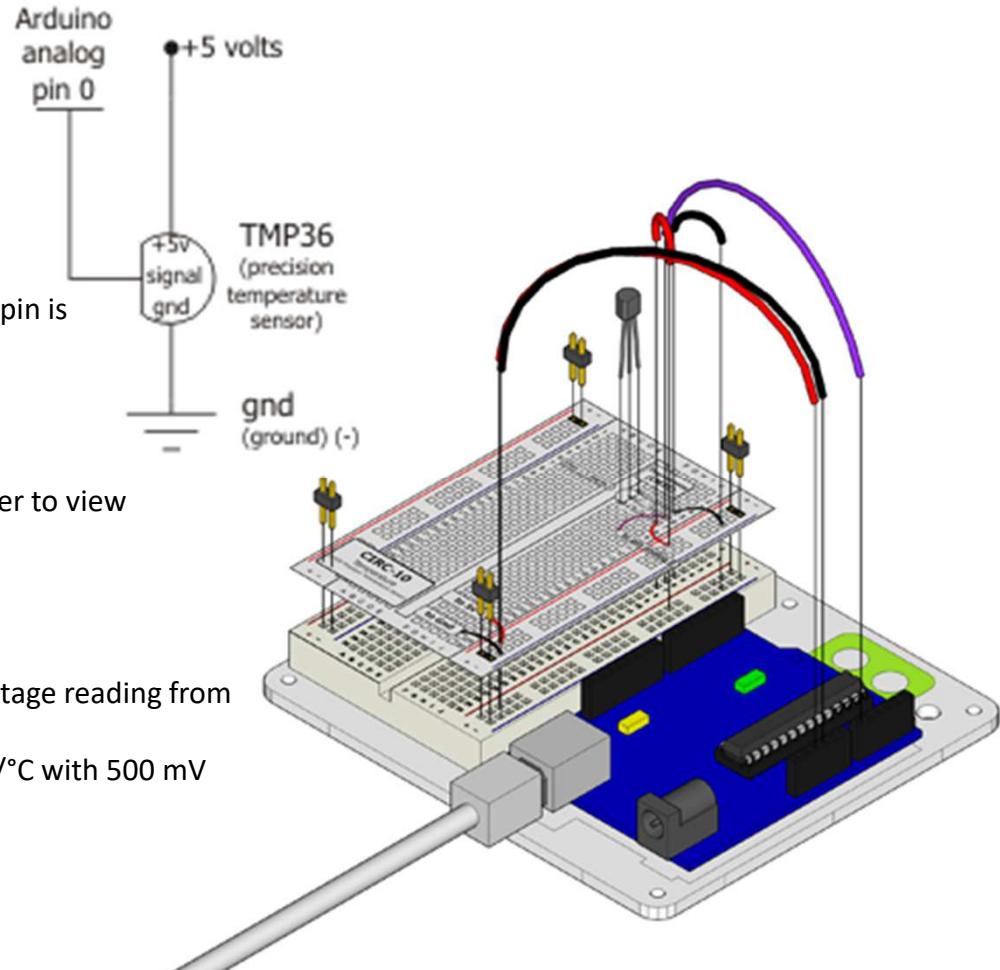
# TP1E

# CAPTURE OF THE TEMPERATURE: PROGRAM

```
/*
 * | Arduino Experimentation Kit Example Code
 * | CIRC-10 :: Temperature :: (TMP36 Temperature Sensor)
 * A simple program to output the current temperature to the IDE's
debug window
* For more details on this circuit: http://tinyurl.com/c89tvd
*/
int temperaturePin = 0; //the analog pin the TMP36's Vout (sense) pin is
connected to

void setup()
{
    Serial.begin(9600); //Start the serial connection with the computer to view
the result open the serial monitor
}
void loop()
{
float temperature = getVoltage(temperaturePin); //getting the voltage reading from
the temperature sensor
temperature = (temperature - 0.5) * 100; //converting from 10mV/°C with 500 mV
//offset to degrees ((voltage - 500mV)*100)
Serial.println(temperature); //printing the result
delay(1000); //waiting a second
}

/* getVoltage() – returns the voltage on the analog input defined by pin */
float getVoltage(int pin){
return (analogRead(pin) * .004882814);
//converting from a 0 to 1023 digital range
//to 0 to 5 volts (each 1 reading equals ~ 5 mV)
}
http://www.oomlout.com/a/products/ardx/circ-10
```



QUESTION: avec ce programme, le moniteur série affiche les valeurs de températures avec 2 digit après la virgule, EST-CE PERTINENT ?

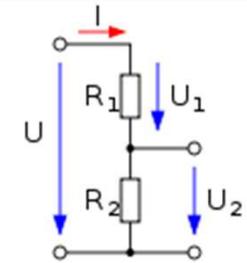
<= RETOUR

# VOLTAGE DIVIDER

The voltage divider is a simple electronic circuit that allows to divide an input voltage. For example, the circuit consists of two resistances in series:

A U voltage input on these two resistances and the U<sub>2</sub> output voltage is measured at the terminals of R<sub>2</sub>.

$$U_2 = U \frac{R_2}{R_1 + R_2}$$



**Application:** reading higher voltages if you want to read V > 5 volts (e.g. 0 to 24 volts), then you must use this voltage divider.

Example with a divider for U<sub>max</sub> = 24V :

U<sub>2max</sub> is fixed at 5V because of the Arduino, arbitrarily set R<sub>2</sub> = 1kΩ, the necessary value for R<sub>1</sub>: => R<sub>1</sub> = ((U/U<sub>2</sub>) \* R<sub>2</sub>) - R<sub>2</sub> = ((24/5) \* 1000) - 1000 = 3.8 kΩ thus, when the voltage U is 24 V (maximum), the maximum voltage on the analog input (U<sub>2</sub>) is 5V.

**DECODING of analog playback (reminder: 10bits resolution):**

What is the voltage at the entrance of the bridge, if analogRead() returns 436?

Method 1: the rule of three: analogRead() gives a value between 0 and 1023 (or 1024 values as possible) for a voltage between 0 and 24V =>  $(24V/2^{10}) * 436 = 10.22 V$

Méthod 2: use the above formula : each analogRead() interval gives  $5/2^{10} = 4.88mV \Rightarrow U_2 = 436 * 4.88 \times 10^{-3} = 2.12V \Rightarrow U = U_2 * (R_1 + R_2) / R_2 = 2.12 * (3800 + 1000) / 1000 = 10.22 V$

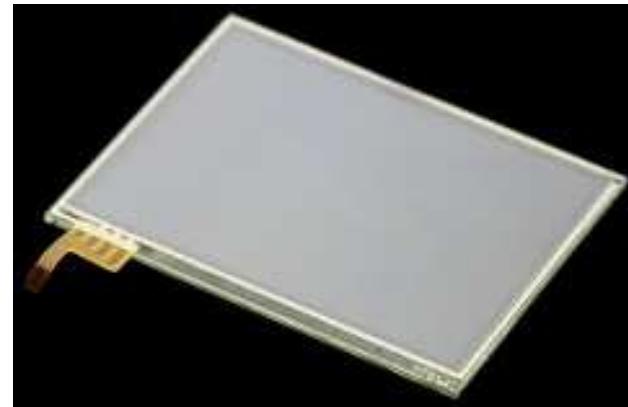
*Note: using a voltage divider, lost also precision in measurement since each interval represents a variation of voltage 24V U / 210 = 23.4 mV instead of 4.88mV.*

<= BACK

## TOUCH PAD DS

Small screens touch resistive 4-wire are now very cheap (< €10), because products in huge quantities for mobile phones, PDAs and video games (Nintendo DS)

Use bigger screens is also possible for less than € 150, because the popularity of netbooks with screens of 7 "and 10" did decrease their price. "



Although they are supplied with an electronic control unit and a USB interface, these screens are mainly in 4-wire resistive-online one Arduino can drive a tactile screen of 10 "(and more)

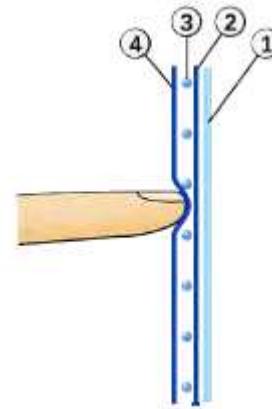
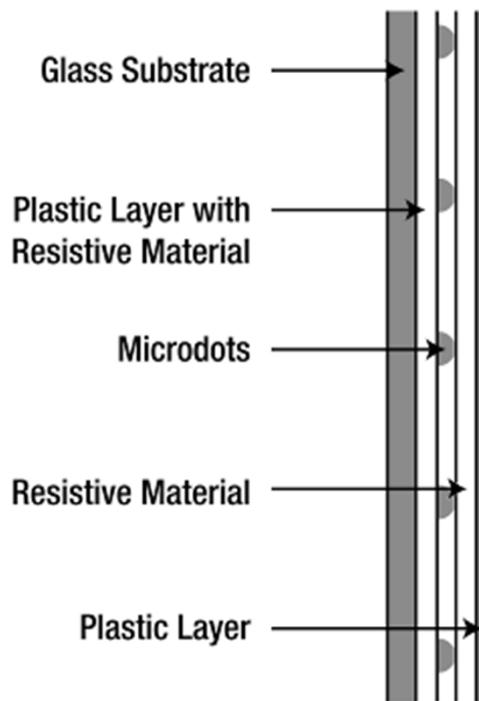
Note, however, that this "touch screen" is actually the glass transparent that fits on the front of an LCD display for example.

They are very good at creating control panels customized with the "buttons" printed on a sheet behind the touch screen. The Arduino capturing the coordinates (X, Y) and comparing them with the coordinates of the button pressed. Of course, your control panel might represent something, not only the buttons.

You could have a top slider to select the volume or temperature level by touching somewhere along a scale, or it could be a plan of a House so that you can control the lights in different rooms by touching the correct part of the ground plane...

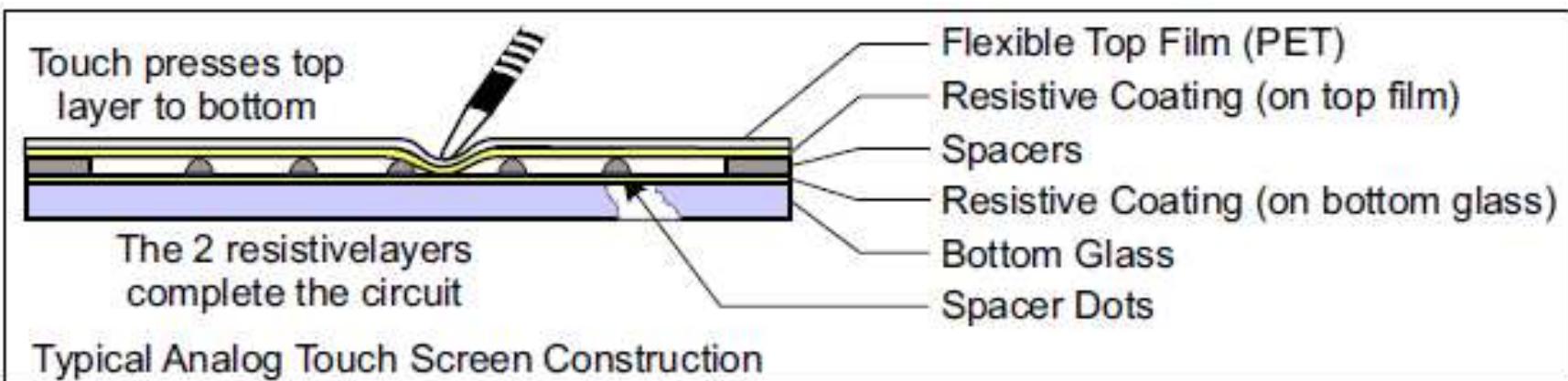
In what follows, we will implement a touch screen of the Nintendo DS on a sheet of paper to trigger actions.

# TOUCH PAD DS: HOW DOES IT WORKS



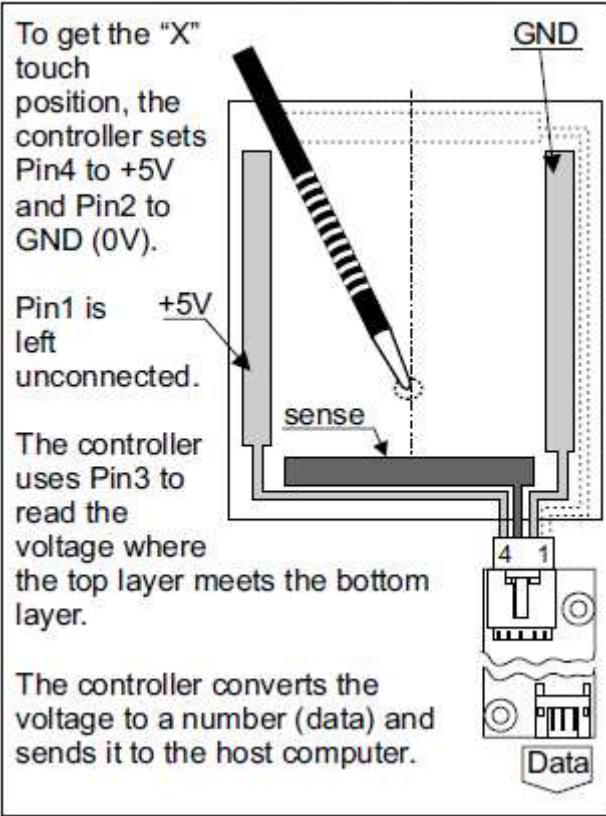
*How a touch screen works (courtesy of Mercury from Wikimedia Commons).*

- 1: Rigid layer.
- 2: Metal oxide layer.
- 3: Insulating dots.
- 4: Flexible layer with metal oxide film.

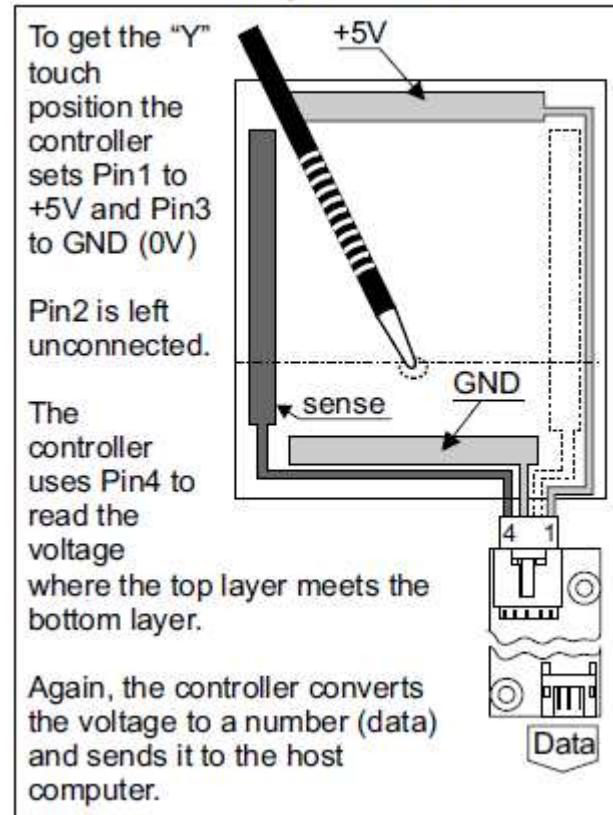


# TOUCH PAD DS: HOW DOES IT WORKS

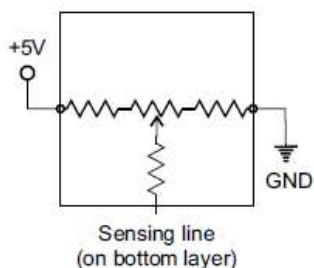
Capturing the "X" Touch



Capturing the "Y" Touch

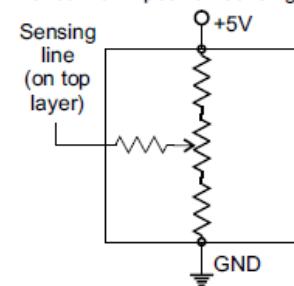


Circuit for X position sensing



ATMega168 Pin	Pin Assignment for X Position	Pin Assignment for Y Position
PC4	5V	ADC
PC3	floating	GND
PC2	ADC	5V
PC1	GND	floating

Circuit for Y position sensing



# TP2A

# TOUCH PAD DS TEST

```

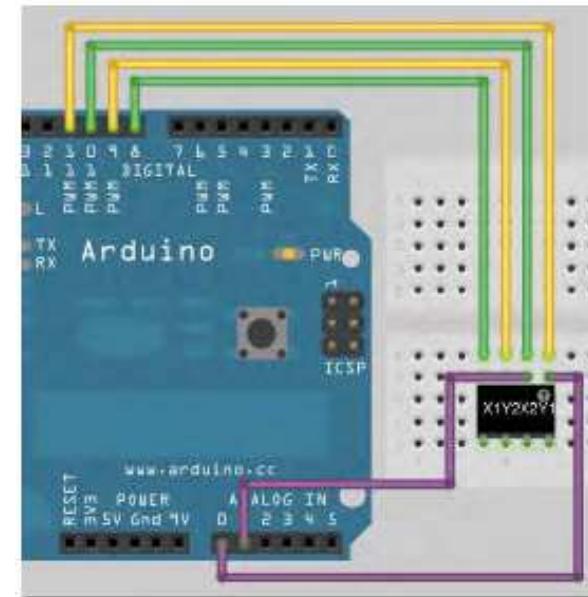
// Project 33 – TOUCH SCREEN
// Power connections
#define Left 8 // Left (X1) to digital pin 8
#define Bottom 9 // Bottom (Y2) to digital pin 9
#define Right 10 // Right (X2) to digital pin 10
#define Top 11 // Top (Y1) to digital pin 11
// Analog connections
#define topInput 0 // Top (Y1) to analog pin 0
#define rightInput 1 // Right (X2) to analog pin 1
int coordX = 0, coordY = 0;
void setup() {
  Serial.begin(38400);
}

void loop() {
  if (touch()) // If screen touched, print co-ordinates
  {
    Serial.print(coordX);
    Serial.print(" ");
    Serial.println(coordY);
    delay(250);
  }
} //fin du void

boolean touch() // return TRUE if touched, and set coordinates to touchX and touchY
{
  boolean touch = false;
  // get horizontal co-ordinates
  pinMode(Left, OUTPUT);
  digitalWrite(Left, LOW); // Set Left to Gnd
  pinMode(Right, OUTPUT); // Set right to +5v
  digitalWrite(Right, HIGH);
  pinMode(Top, INPUT); // Top and Bottom to high impedance
  pinMode(Bottom, INPUT);
  delay(3);
  coordX = analogRead(topInput);
  // get vertical co-ordinates
  pinMode(Bottom, OUTPUT); // set Bottom to Gnd
  digitalWrite(Bottom, LOW);
  pinMode(Top, OUTPUT); // set Top to +5v
  digitalWrite(Top, HIGH);
  pinMode(Right, INPUT); // left and right to high impedance
  pinMode(Left, INPUT);
  delay(3);
  coordY = analogRead(rightInput);
  // if co-ordinates read are less than 1000 and greater than 0 then the screen has been touched
  if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}
  return touch;
}

```

*Beginning Arduino Michael McRoberts APRESS*



Arduino	Breakout
Digital Pin 8	X1
Digital Pin 9	Y2
Digital Pin 10	X2
Digital Pin 11	Y1
Analog Pin 0	Y1
Analog Pin 1	X2

<= BACK



67

# ANOTHER EXAMPLE OF CODE FOR TOUCH PAD DS

```
/**  
 * Touch Control Panel  
 * Copyright 2009 Jonathan Oxer <jon@oxer.com.au>  
 * Copyright 2009 Hugh Blemings <hugh@blemings.org>  
  
 * • Reads touch coordinates on a Nintendo DS touch screen attached to an  
 * Arduino and compares them to defined hot zones representing buttons and sliders.  
 * If a touch occurs within a hot zone a matching event message is sent to the host via the serial port.  
 * Based on the ReadTouchscreen example included in the TouchScreen library.  
 * * www.practicalarduino.com/projects/touch-control-panel  
 */  
#include <TouchScreen.h>  
TouchScreen ts(3, 1, 0, 2);  
void setup()  
{  
    Serial.begin(38400);  
}  
void loop()  
{  
    int coords[2];  
    ts.read(coords);  
    Serial.print(coords[0]);  
    Serial.print(",");  
    Serial.print(coords[1]);  
  
    if((coords[0] > 696) && (coords[0] < 866) && (coords[1] > 546) && (coords[1] < 831)) { Serial.print(", Fan ON"); }  
    if((coords[0] > 696) && (coords[0] < 866) && (coords[1] > 208) && (coords[1] < 476)) { Serial.print(", Fan OFF"); }  
    if((coords[0] > 420) && (coords[0] < 577) && (coords[1] > 540) && (coords[1] < 866)) { Serial.print(", Drapes OPEN"); }  
    if((coords[0] > 420) && (coords[0] < 577) && (coords[1] > 208) && (coords[1] < 476)) { Serial.print(", Drapes CLOSE"); }  
    if((coords[0] > 139) && (coords[0] < 327) && (coords[1] > 208) && (coords[1] < 866)) { Serial.print(", Illumination"); }  
    Serial.print(constrain(map(coords[1], 318, 756, 0, 100), 0, 100));  
    Serial.print("%");  
}  
    Serial.println();  
    delay (100);  
}
```

Librairie TouchScreen: <https://github.com/practicalarduino/TouchScreen>

<https://github.com/practicalarduino/TouchControlPanel/blob/master/TouchControlPanel.pde>

<http://www.sparkfun.com/tutorials/139>

# TO GO FURTHER: ACCELERATING THE ANALOG READ()

Extracted from the doc2559 d'ATMEL datasheet :

"The ADC accuracy also depends on the ADC clock.

The recommended maximum ADC clock frequency is limited by the internal DAC in the conversion circuitry.

By default, the successive approximation circuitry requires an input clock frequency between 50kHz and 200kHz to get maximum resolution (10bits) (a normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry)

However, frequencies up to 1 MHz do not reduce the ADC resolution significantly.

*Operating the ADC with frequencies greater than 1 MHz is not characterized.*

For Arduino and f=16MHz Clock: since the ADC clock needs to be between 50 kHz & 200 kHz for 10 bit accuracy, we can only use the 128 prescaler and can achieve a 125 kHz ADC clock.

If a lower resolution than 10 bits is possible, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate.

## Tiré de la datasheet ATMEGA328P: doc8271 d'ATMEL

### 24.9.2 ADCSRA – ADC Control and Status Register A

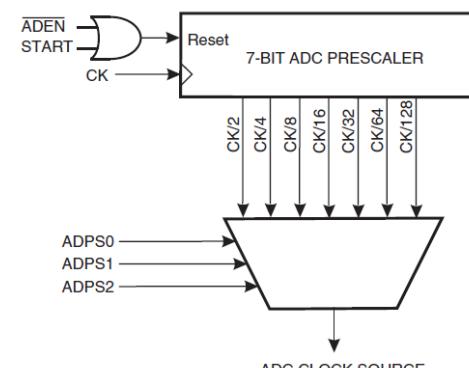
Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bits 2:0 – ADPS[2:0]: ADC Prescaler Select Bits

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 24-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor	f(MHz)/Horloge à 16MHz
0	0	0	2	
0	0	1	2	8
0	1	0	4	4
0	1	1	8	2
1	0	0	16	1
1	0	1	32	0,5
1	1	0	64	0,25
1	1	1	128	0,125



Prescaler from ARDUINO library (wiring.c, line 276.) to have 10 resolution bits because  $50\text{kHz} < (16\text{MHz}/128)=125\text{kHz} < 200\text{kHz}$

# TO GO FURTHER: ACCELERATING THE ANALOG READ()

```

sketch_mar23a | Arduino 1.0.2
Fichier Édition Croquis Outils Aide
sketch_mar23a FastAnalogRead §
//TITRE: FASTANALOGREAD
//QUE FAIT CE PROGRAMME:
// IL TESTE LA VITESSE DE LECTURE D'UNE ENTREE ANALOGIQUE
// EN MODIFIANT LES PRESCALERS
// This example code is in the public domain.
// J. Grisolia (2013)

// à définir pour initialiser et effacer les bits des registres
#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

int start;
int i,j;
int FASTADC;
int pin_analogique;
float somme_analogique;

//ADC PRESCALER BITS
// ATTENTION LE PRESCALER PAR DEFAUT EST LE 128, AUGMENTER LA VITESSE DE TRAITEMENT
// DE L'ANALOGREAD SIGNIFIE DIMINUER LE PRESCALER
int prescaler[] = {0, 2, 4, 8, 16, 32, 64, 128};

void setup() {
pin_analogique=0;
somme_analogique=0;

Serial.begin(9600);
Serial.println("Test de l'ADC avec différents PRESCALER mesuré sur 1000 appels de la fonction ANALOGREAD: ") ;

//Boucle sur les prescaler
}

```

```

for (j = 0 ; j < 8 ; j++) {
FASTADC = prescaler[j];
switch (FASTADC) {
case 0: // prescale à 0
cbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 2: // prescale à 2
cbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
case 4: // prescale à 4
cbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 8: // prescale à 8
cbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
case 16: // prescale à 16
sbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 32: // prescale à 32
sbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
case 64: // prescale à 64
sbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 128: // prescale à 128 - Analog Read par défaut => ~113ms
sbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
default: break;
};

start = millis();
for (i = 0 ; i < 1000 ; i++) {
// analogRead(pin_analogique) ;
somme_analogique= somme_analogique + analogRead(pin_analogique);
}//Fin du i

Serial.print("Prescaler ");
Serial.print(FASTADC);
Serial.print(": ");
Serial.print(millis() - start) ;
Serial.print(" msec, ");
Serial.print("T(c):"); //Mesure de température avec le TMP36
//Serial.println((analogRead(pin_analogique)*4.88-500)/10); //Affichage de la température
Serial.println(((somme_analogique/1000)*4.88-500)/10); //Affichage de la température
somme_analogique=0;

}//Fin du j

}//Fin du setup

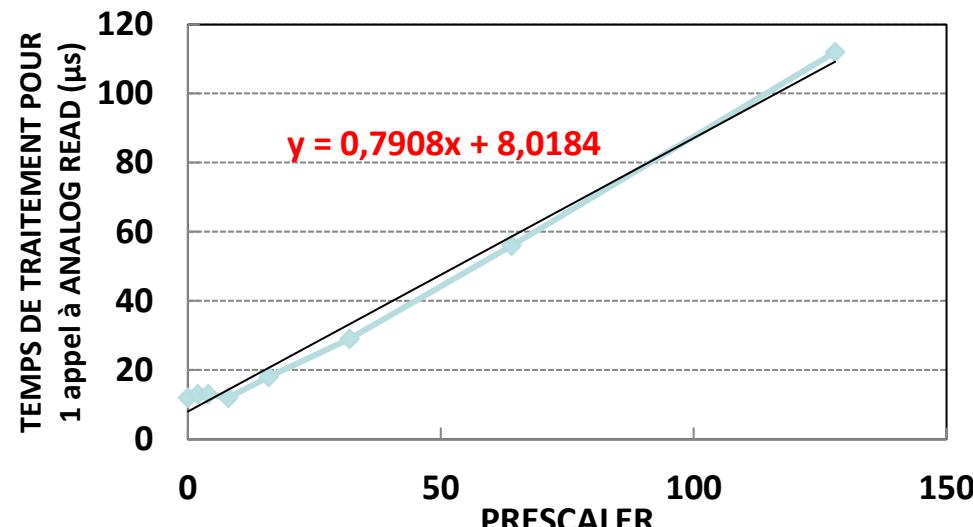
void loop() {
}

```

**cbi : clear bit in I/O register**  
**⇒ Put 0 to the designed bit**

**sbi : set bit in I/O register**  
**⇒ Put 1 to the designed bit**

# TO GO FURTHER: ACCELERATING THE ANALOG READ()



<http://arduino.cc/forum/index.php/topic,6549.0.html>  
<http://www.arduino.cc/en/Reference/PortManipulation>  
<http://playground.arduino.cc/Code/BitMath>  
<http://www.fiz-ix.com/2012/01/how-to-configure-arduino-timer-2-registers-to-drive-an-ultrasonic-transducer-with-a-square-wave/>  
<http://www.marulaberry.co.za/index.php/tutorials/code/arduino-adc/>

We double the frequency, and we get substantially the double samples/s between two successive measurement

Prescaler	Fréquence (MHz) - Horloge 16MHz	temps lecture (1000 appels ANALOGREAD) (ms)	temps lecture/appel (μs)	Température avec TMP36 sur analog read()	Echantillons par seconde (théorie: f/13 cycles)	Echantillons par seconde (expérience: 1/temps lecture)	Rapport fréquence (théorique)	Rapport fréquence (expérimental)
0		19	19	449,22	0	52632		
2	8	18	18	449,22	615385	55556		0,95
4	4	18	18	24,12	307692	55556	2	1,00
8	2	20	20	23,42	153846	50000	2	1,11
16	1	28	28	23,2	76923	35714	2	1,40
32	0,5	41	41	23,05	38462	24390	2	1,46
64	0,25	69	69	22,97	19231	14493	2	1,68
128	0,125	121	121	22,88	9615	8264	2	1,75

Conclusions: From 128 to 4 prescaler, we go from 121μs to 18μs by reading the analog function ANALOG READ(), either about 10 x faster, without losing "too" precision (it remains in the bars of the TMP36 errors), even if there is a slight offset! We can quietly go up at least until f = 1 MHz (16 prescaler)

Attention: avoid the prescaler 0 and 2 who give completely erroneous values and more who don't win in reading speed!

## III - 3 DIGITAL & ANALOG APPLICATIONS

## III - 3 DIGITAL & ANALOG APPLICATIONS

## LCD GENERALITIES

An LCD display is a cheap display device.

It is easy to interface with Arduino, because they contain their own microcontroller (black pad on the back) which manages the display logic.

This µcontroller is the Hitachi 44780 HD standard on the majority of viewers.



This standardization => a lot of µ-controller platform has a ready-made library for development.

It is also the case with Arduino with the LiquidCrystal library.

This makes it possible to control the display with a few lines of command.

The majority of the current displays (e.g., LCD 2 x 16) have two benefits:

1 - the display is able to use 4-bit or 8-bit data bus: the 4-bit version is advantageous because it allows to send data to the display in using only 4 pins instead of 8. It's as much pine recovered to control other devices

2 - order logic (and library) can just use only 2 pins instead of 3 (with R/W the mass order, if you do not wish to read the memory of the display (do NC make only display))

So the display is fully controllable with only 6 pins!

# HITACHI HD44780 (LCD-II) Block Diagram

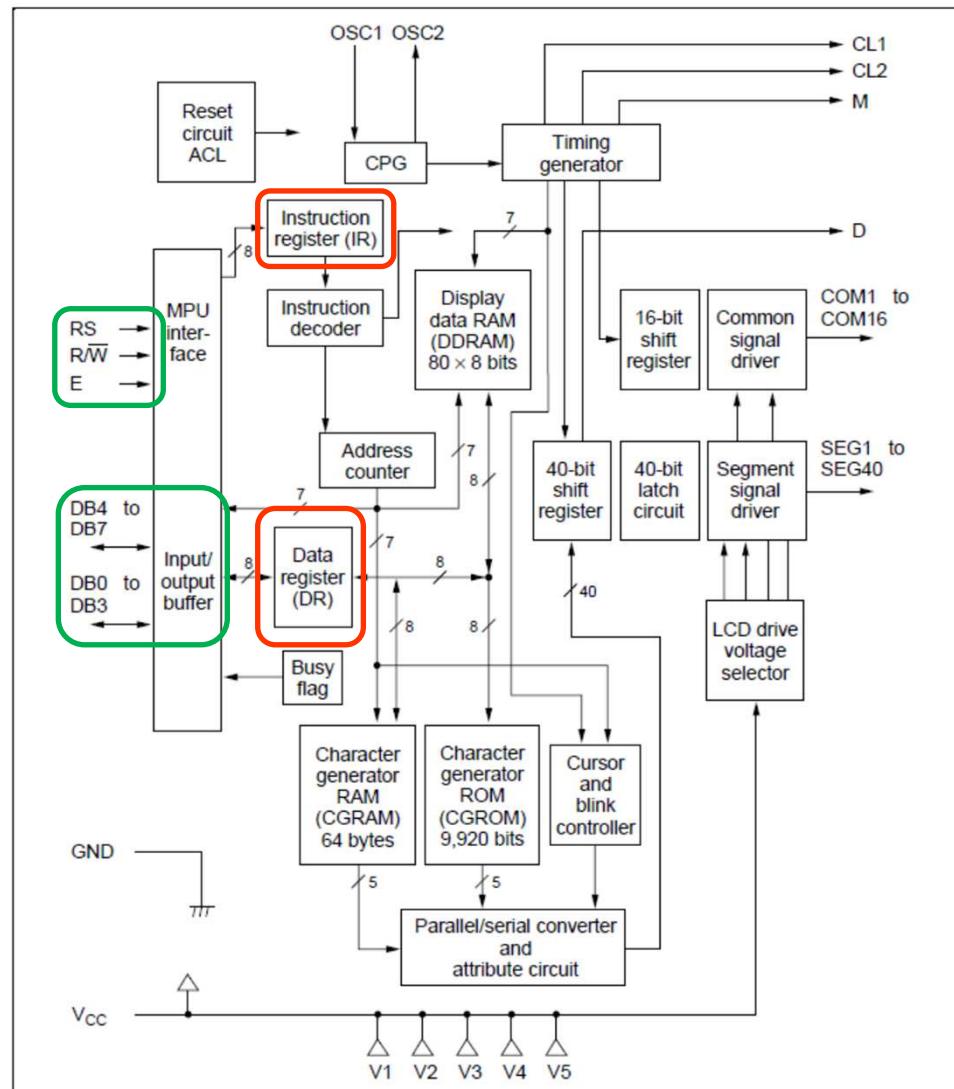
## HD44780U (LCD-II)

(Dot Matrix Liquid Crystal Display Controller/Driver)

### Pin Functions

Signal	No. of Lines	I/O	Device Interfaced with	Function								
RS	1	I	MPU	Selects registers. 0: Instruction register (for write) Busy flag: address counter (for read) 1: Data register (for write and read)								
R/W	1	I	MPU	Selects read or write. 0: Write 1: Read								
E	1	I	MPU	Starts data read/write.								
DB4 to DB7	4	I/O	MPU	Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag.								
DB0 to DB3	4	I/O	MPU	Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. These pins are not used during 4-bit operation								
Code												
Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when $f_{osc}$ or $f_{op}$ is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 $\mu$ s
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, 37 $\mu$ s cursor on/off (C), and blinking of cursor position character (B).	37 $\mu$ s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 $\mu$ s
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 $\mu$ s
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 $\mu$ s
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 $\mu$ s
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 $\mu$ s

HD44780U Block Diagram



## LCD GENERALITIES

Also data lines DB0-DB7 that one driver using 4 or 8 pins to digital, and that convey both the displayed information and commands sent to the display,

Three control lines are also necessary for its functioning :

1 - E or Enable valid display when the pin is high. It can then receive orders or characters to display via its rows of data, it is insensitive to their condition otherwise.

2 - R/W for Read/Write indicates if you want to write a data display (R/W to 0) or read its information (R/W to 1). When you want to save input/output, this line is frequently grounded blocking the display mode (main mode of use) writing

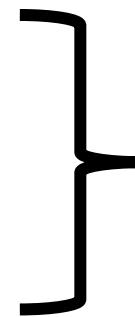
=> cannot read its State registers and his memory of characters.

3 - RS to Register Select indicates if the data lines carry information to display (RS at 1) or orders of management display (RS to 0). Various commands, like moving the slider from right to left or vice versa, with or without deletion of character, erasing the display... are sent on the lines of data to DB7 DB0 putting RS at 0

# LCD COMMAND

DIALOGUE with the display in the case of a writing:

- 1 - set to 0 for line R/W
- 2 - RS at the level desired depending on whether you want to send a character or an order
- 3 - positioning of the code of the character or command on DB0 to DB7 (8 bit mode) or DB4 to DB7 (mode 4 bits) :
- 4 - setting to 1 line E to send these information
- 5 - set to 0 on line E making it insensitive to the State of DB0 display to DB7



► **Sending data/command in 4 bits mode**  
It separates the strong 4 bits and the low 4 bits.  
The common steps are:  
Hide 4 LSB bits  
Send to LCD  
Send ENABLE signal  
Hide 4 MSB bits  
Send to LCD  
Send ENABLE signal

***REM: 4-bit mode saves pins but is slower than the 8-bit because we need to send the data in 2 times***

# LCD PINOUTS AND COMMAND

The pinout and specifications of the display are available in the respective datasheets, but in general, they are as follows (pins 1 and 16 are shown on the display).

Table 1: Display Control

Pin	Symbol	Description
1	V <sub>SS</sub>	Ground
2	V <sub>DD</sub>	Supply Voltage for Logic
3	V <sub>o</sub>	Supply Voltage for LCD (Contrast)
4	RS	Register Select
5	R/W	Read/Write
6	CE	Chip Enable
15	LED(+)	Anode of LED Backlight
16	LED(-)	Cathode of LED Backlight

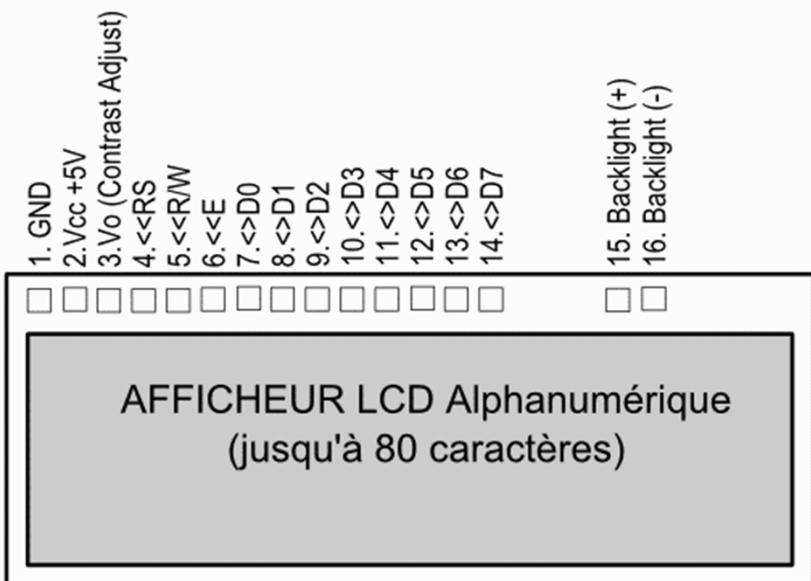


Table 2: Parallel Data

Pin	Symbol	Description
7	DB0	*Data bit 0
8	DB1	*Data bit 1
9	DB2	*Data bit 2
10	DB3	*Data bit 3
11	DB4	Data bit 4
12	DB5	Data bit 5
13	DB6	Data bit 6
14	DB7	Data bit 7

\*Note: Not used in 4-bit mode

Higher 4bit	Lower 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx	0000													
xxxx	0001													
xxxx	0010													
xxxx	0011													
xxxx	0100													
xxxx	0101													
xxxx	0110													
xxxx	0111													
xxxx	1000													
xxxx	1001													
xxxx	1010													
xxxx	1011													
xxxx	1100													
xxxx	1101													
xxxx	1110													
xxxx	1111													

## TP2B

# LCD: CODE EXAMPLE

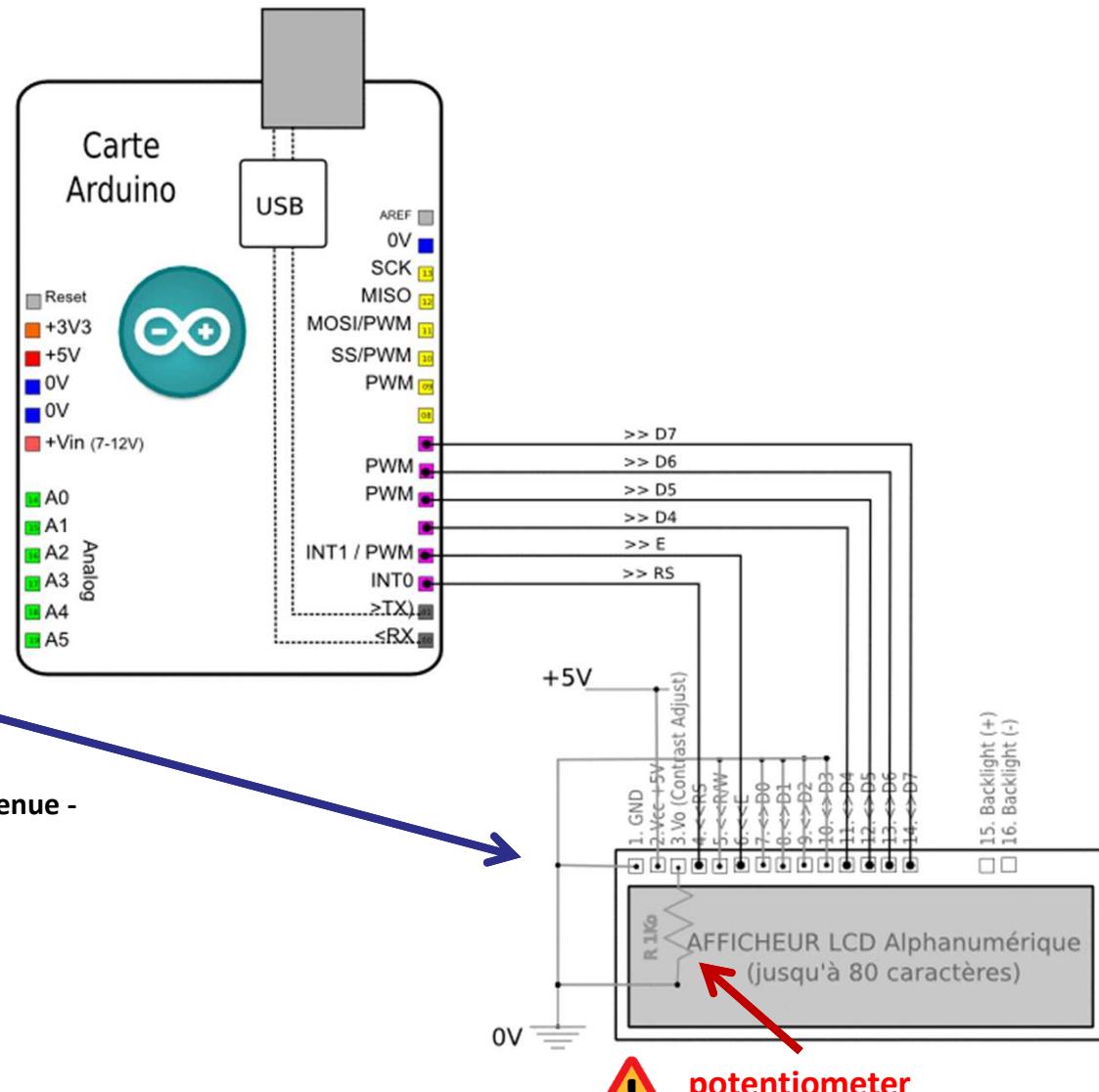
```

// Inclusion de la librairie
#include <LiquidCrystal.h> // Dimension de l'afficheur
const int numRows = 2;
const int numCols = 16; // Initialisation de la librairie
// avec le nbre de pins d'interface
// 4 bit de données dans notre cas.
/* Le montage: Afficheur LCD
* LCD RS - pin 2
* LCD Enable - pin 3
* LCD D4 - pin 4
* LCD D5 - pin 5
* LCD D6 - pin 6
* LCD D7 - pin 7
* LCD R/W - GND
* LCD Vo contrast- potentiomètre 10K (entre GND et +5V)
*/
LiquidCrystal lcd(2,3,4,5,6,7);

void setup()
{
Serial.begin(9600); // Bouton changer thème
lcd.begin(numCols,numRows); //--- Message de bienvenue -
lcd.print( "demo LCD" ); // placer curseur sur ligne 2
lcd.setCursor(0,1); // col, row
lcd.print( "Hello!" ); // clignotement curseur plein
lcd.blink();
}

void loop()
{
}

```



<= BACK



79

# LCD: ANOTHER EXAMPLE OF CODE

```
// --- Programme Arduino --- par X. HINAULT - 01/2010
// --- Que fait ce programme ? ---
/* Affiche des messages texte sur l'afficheur LCD*/
// Fonctionnalités utilisées ---
// Utilise un afficheur LCD 4x20 en mode 4 bits
// Circuit à réaliser ---
// Connexion du LCD sur les broches de la carte Arduino
// Connecter broche RS du LCD sur la broche 2
// Connecter broche E du LCD sur la broche 3
// Connecter broche D4 du LCD sur la broche 4
// Connecter broche D5 du LCD sur la broche 5
// Connecter broche D6 du LCD sur la broche 6
// Connecter broche D7 du LCD sur la broche 7

//***** Entête déclarative *****
// A ce niveau sont déclarées les librairies, les constantes, les variables...
// --- Inclusion des librairies utilisées ---
#include <LiquidCrystal.h> // Inclusion de la librairie pour afficheur LCD
// --- Déclaration des constantes ---
// --- constantes des broches ---
const int RS=2; //declaration constante de broche
const int E=3; //declaration constante de broche
const int D4=4; //declaration constante de broche
const int D5=5; //declaration constante de broche
const int D6=6; //declaration constante de broche
const int D7=7; //declaration constante de broche

// --- Déclaration des variables globales ---
// --- Initialisation des fonctionnalités utilisées ---
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); // initialisation LCD en mode 4 bits

//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et 1 seule fois, au démarrage du programme
void setup() { // début de la fonction setup()

    // --- ici instructions à exécuter au démarrage ---
    lcd.begin(20,4); // Initialise le LCD avec 20 colonnes x 4 lignes
    delay(10); // pause rapide pour laisser temps initialisation
    // Test du LCD
    lcd.print("LCD OK"); // affiche la chaîne texte - message de test
    delay(2000); // pause de 2 secondes
    lcd.clear(); // efface écran et met le curseur en haut à gauche
    delay(10); // pour laisser temps effacer écran

} // fin de la fonction setup()
// *****
```

```
***** FONCTION LOOP = Boucle sans fin = cœur du programme *****
// la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous tension

void loop(){ // début de la fonction loop()

    lcd.print("Arduino..."); // affiche la chaîne texte - message de test
    delay(2000); // pause de 2 secondes
    lcd.setCursor(9, 1); // 10ème col - 2ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
    lcd.print("...pour te"); // affiche la chaîne texte - message de test
    delay(2000); // pause de 2 secondes
    lcd.setCursor(4, 2); // 5ème col - 3ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
    lcd.print("...servir"); // affiche la chaîne texte - message de test
    delay(2000); // pause de 2 secondes
    lcd.setCursor(12, 3); // 13ème col - 4ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
    lcd.print("...amigo"); // affiche la chaîne texte - message de test
    delay(2000); // pause de 2 secondes
    lcd.clear(); // efface écran et met le curseur en haut à gauche
    delay(10); // pour laisser temps effacer écran

    // --- ici instructions à exécuter par le programme principal ---
} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
// ***** Fin programme *****
```

```
----- memo LCD -----
// LiquidCrystal(rs, enable, d4, d5, d6, d7); // initialisation 4 bits
// lcd.begin(cols, rows); // initialisation nombre colonne/ligne
//
// lcd.clear(); // efface écran et met le curseur en haut à gauche
// lcd.home(); // repositionne le curseur en haut et à gauche SANS effacer écran
//
// lcd.setCursor(col, row); // positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
// lcd.print("texte"); // affiche la chaîne texte
//
// lcd.cursor(); // affiche la ligne de base du curseur
// lcd.noCursor(); // cache le curseur
// lcd.blink(); // fait clignoter le curseur
// lcd.noBlink(); // stoppe le clignotement du curseur
// lcd.noDisplay(); // éteint le LCD sans modifier affichage
// lcd.display(); // rallume le LCD sans modif affichage
//
// lcd.scrollDisplayLeft(); // décale l'affichage d'une colonne vers la gauche
// lcd.scrollDisplayRight(); // décale l'affichage d'une colonne vers la droite
// lcd.autoscroll(); // les nouveaux caractères poussent les caractères déjà affichés
// noAutoscroll(); // stoppe le mode autoscroll
```

# LCD + TEMPERATURE SENSOR TMP36

```
/* Mesure de la température à l'aide d'un TMP36 et Affichage sur LCD: MOP-AL162A-BBTW: LCD 2x16
Le montage: Afficheur LCD
* LCD RS - pin 2
* LCD Enable - pin 3
* LCD D4 - pin 4
* LCD D5 - pin 5
* LCD D6 - pin 6
* LCD D7 - pin 7
* LCD R/W - GND
* LCD Vo contrast- potentiomètre 10K (entre Gnd et +5V)
Senseur de température:
* TMP36 Analog Output - Pin A0 (analogique)

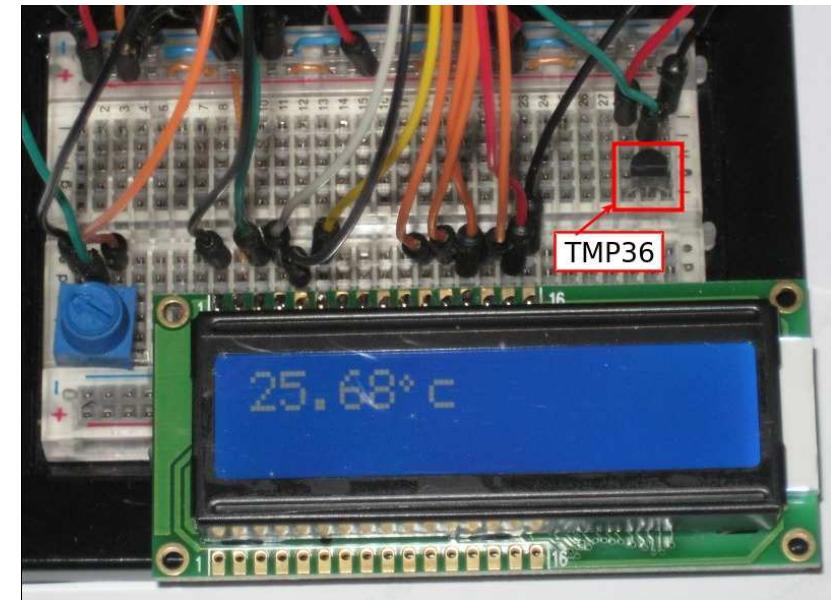
// include the library code:
#include <LiquidCrystal.h>
int tempSensorPin = 0; // Pin analogique pour lecture de la tension de sortie du TMP36 (Vout).
// Resolution: 10 mV / degree celsius avec une offset de 500 mv. // Definition du caractere °
// initialize the library with the numbers of the interface pins
byte degrees[8] = { B00000, B01000, B10100, B01000, B00000, B00000, B00000, B00000, };
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

void setup(){
lcd.begin(16, 2); // set up the LCD's number of columns and rows:
lcd.clear(); //efface l'écran
lcd.createChar(0, degrees ); } // initialiser le caractère ° dans le LCD
int lastTemp = -100; // mémorise la dernière température affichée

void loop(){

float temp = lectureTemp(); // rafraichit le LCD que si la // température a varié sensiblement
if( abs(temp-lastTemp)<0.20) return; lastTemp = temp; // Afficher la valeur en évitant le // lcd.clear(), pour éviter l'effet de // scintillement.
lcd.setCursor(0,0);
lcd.print( temp );
lcd.write( 0 ); // affiche le signe degré
lcd.print( "c" ); // Efface les derniers caractères si // la température chute subitement
lcd.print( " " ); // ne pas rafraîchir trop souvent
delay(800); }

//Description: // Lecture de la température sur la pin A0 // //Returns: // La température en degré Celcius. //
float lectureTemp(){ // Lecture de la valeur sur l'entrée analogique // Retourne une valeur entre 0->1024 pour 0->5v
int valeur = analogRead(tempSensorPin); // Converti la lecture en tension
float tension = valeur * 5.0; tension /= 1024.0; // Convertir la tension (mv) en température
float temperature = ((tension * 1000) - 500) / 10;
return temperature; }
```



Source: <http://arduino103.blogspot.com/search/label/sensor>

LCD character editor: <http://icontexto.com/charactercreator/>

<= BACK



81

# TP2C

# TOUCH SCREEN & LCD

```

// Project 34
#include <LiquidCrystal.h>
LiquidCrystal lcd(2, 3, 4, 5, 6, 7); // create an lcd object
//and assign the pins
// Power connections
#define Left 8 // Left (X1) to digital pin 8
#define Bottom 9 // Bottom (Y2) to digital pin 9
#define Right 10 // Right (X2) to digital pin 10
#define Top 11 // Top (Y1) to digital pin 11
// Analog connections
#define topInput 0 // Top (Y1) to analog pin 0
#define rightInput 1 // Right (X2) to analog pin 1
int coordX = 0, coordY = 0;
char buffer[16];
void setup()
{
lcd.begin(16, 2); // Set the display to 16 columns and 2 rows
lcd.clear();
}
void loop()
{
if (touch())
{
if ((coordX<110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
if ((coordX<110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
if ((coordX<110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
delay(250);
}
}

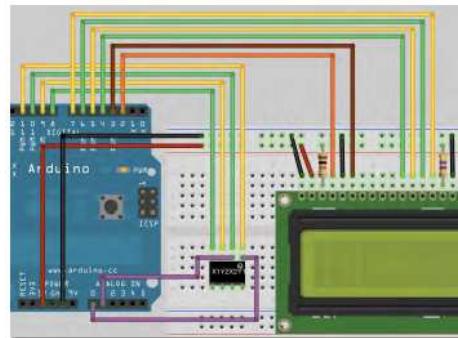
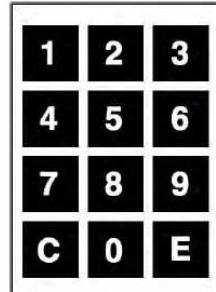
```

Arduino	Other	Matrix
Digital 2		Enable
Digital 3		RS (Register Select)
Digital 4		DB4 (Data Pin 4)
Digital 5		DB5 (Data Pin 5)
Digital 6		DB6 (Data Pin 6)
Digital 7		DB7 (Data Pin 7)
Gnd		Vss (GND)
Gnd		R/W (Read/Write)
+5v		Vdd
	+5v via resistor	Vo (Contrast)
	+5v via resistor	A/Vee (Power for LED)
	Gnd	Gnd for LED

```

// return TRUE if touched, and set coordinates to touchX and touchY
boolean touch()
{
boolean touch = false;
// get horizontal co-ordinates
pinMode(Left, OUTPUT);
digitalWrite(Left, LOW); // Set Left to Gnd
pinMode(Right, OUTPUT); // Set right to +5v
digitalWrite(Right, HIGH);
pinMode(Top, INPUT); // Top and Bottom to high impedance
pinMode(Bottom, INPUT);
delay(3); // short delay
coordX = analogRead(topInput);
// get vertical co-ordinates
pinMode(Bottom, OUTPUT); // set Bottom to Gnd
digitalWrite(Bottom, LOW);
pinMode(Top, OUTPUT); // set Top to +5v
digitalWrite(Top, HIGH);
pinMode(Right, INPUT); // left and right to high impedance
pinMode(Left, INPUT);
delay(3); // short delay
coordY = analogRead(rightInput);
// if co-ordinates read are less than 1000 and greater than 0 then the
screen has been touched
if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch
= true;}
return touch;
}
void scrollLCD() {
for (int scrollNum=0; scrollNum<16; scrollNum++) {
lcd.scrollDisplayLeft();
delay(100);
}
lcd.clear();
}

```



<= BACK



83

### III - 4 DO "ANALOG" WITH DIGITAL

III - 4 DO "ANALOG" WITH  
DIGITAL

# PULSE WIDTH MODULATION (PWM)

The Arduino lack of a DIGITAL/ANALOG converter in the strict sense of the term.

=> Impossible to have a DC voltage which is the image of a digital data on one of the connectors.

On the other hand, they are capable of generating signals with pulse width modulation (PWM in English: pulse width modulation) that fill the failings of the analog control.

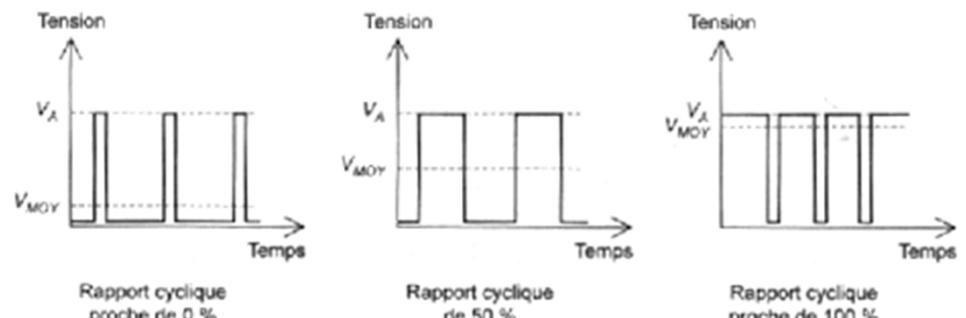
But elegant solution because we're working transistors in commutation (blocked and saturated regime; cf course features A semiconductor 3 A IMACS) => its dissipation is theoretically zero, since either the I or the U are 0 alternately.

The limitations of the analog control: if we are to control the brightness of an LED or the speed of a DC motor, apply a voltage higher or lower (analog control) does imply a getting the desired results-online brightness low for a led is absolutely not guaranteed (because it is a diode (rectifier)). Similarly, the slower starting the engine not provided (because of the inertia).

Duty cycle:

duty cycle = the ratio between the duration of the phenomenon over a period and this same period ( $\alpha = \text{ton}/T$ )

The frequency of the PWM signal is of about ~ 490 Hertz (be a duty cycle of +/- 2 ms) and ~ 1 kHz according to PWM pins.



⇒ Duty cycle 100% => signal high. Duty cycle 0% => signal down  
⇒ all intermediate values are of course permitted

⇒ Even if the signal is not an analog voltage, it can become so if we apply a filter low pass (which keeps only the average of the signal;  $V_{\text{MOY}} = V_A * R_{\text{cyc}}$ )

# PULSE WIDTH MODULATION (PWM)

## Activation mode PWM:

Attention, there is no special instruction to activate the PWM mode and do not initialize the pin as Output before using the `analogWrite()` statement.

Then we select a pin PWM (3, 5, 6, 9, 10, 11) instead of an analog pin.

## *BEWARE: Limit of 5 & 6 in PWM pins:*

*Attention: following a technical constraint (sharing the function millis and delays), PWM control on pins 5 and 6 is inconsistent for the values 0 to 10 analogWrite.*

*As a result, a value of 0 on pins 5 and 6 does not necessarily a LOW output!*

## Calculation Note:

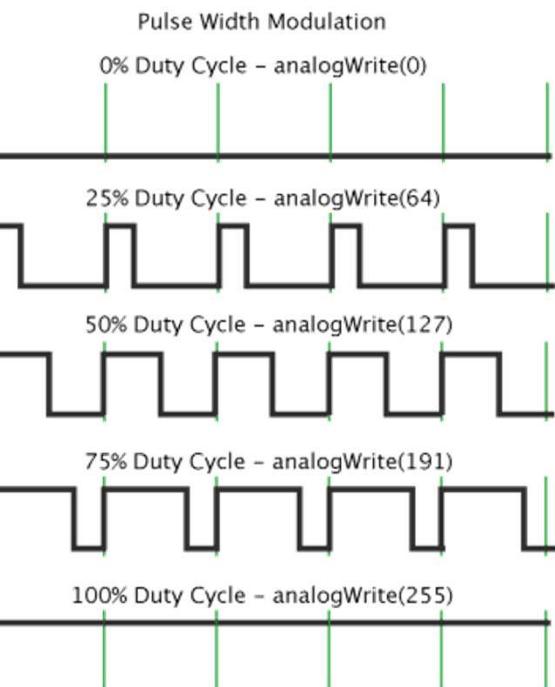
the value passed to `analogWrite()` indicates what proportion of the "duty cycle" in which the output is activated (see chart).

## As AnalogWrite value can vary from 0 to 255:

- A duty cycle of 50% of the time will give: 50% of 255 or 255 \* 0.5 = 127.5 (so the statement `analogWrite 127`).
- Or more generally: a duty cycle x % will give a ((255) n / 100/x)).

## OR vice versa:

If one has a value `analogWrite() = n`,  
the duty cycle will be (n/255)\*100% of the period.



Refinement: low pass filter=> <http://arduino-info.wikispaces.com/Analog-Output>

## « TRUE » ANALOG OUTPUT!

With "the trick" PWM signals, we can get a quasi analog, but how if I want a completely analog signal from a digital signal?

Solution: put a RC filter at the output of the PWM

1 - if the PWM signal is 1, the capacitor takes over.

2 - If the PWM signal goes to 0, the capacitor discharges.  
and so on... involving a variation of voltage at the terminals of the capacitor that is similar to this:

What's new compared to the square wave?

Nothing more, the average value of the blue signal is the same as the red signal.

It should be noted that in this case, the time of charge/discharge of the capacitor is chosen equal to a half of the signal

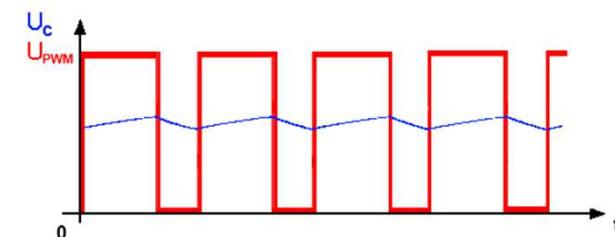
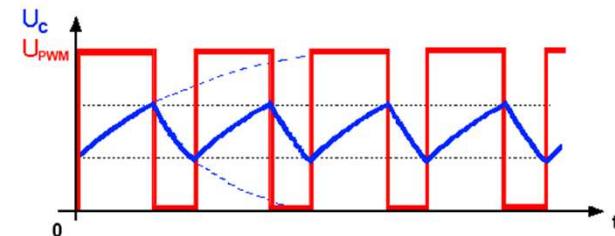
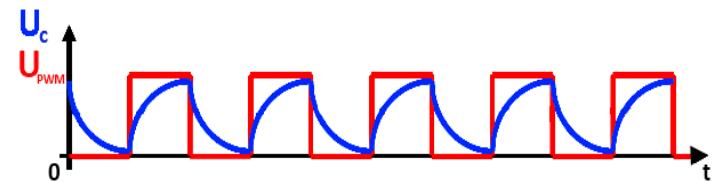
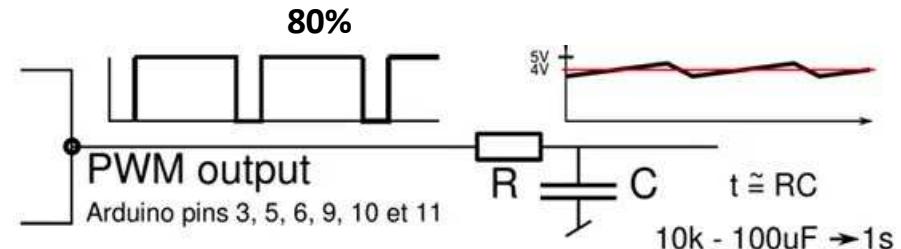
What will happen if we choose a smaller or larger load/discharge time? For example a largest RC:

The voltage at the terminals of the capacitor does not reach the + 5V and 0V as in the previous timeline. The couple RC being larger than previously, the capacitor takes longer to load and as the signal "go faster" capacitor, it cannot be load/unload completely.

What will happen if we continue to increase the constant RC?

This signal approaches in addition to the average value of the PWM signal.

Have we won the game?



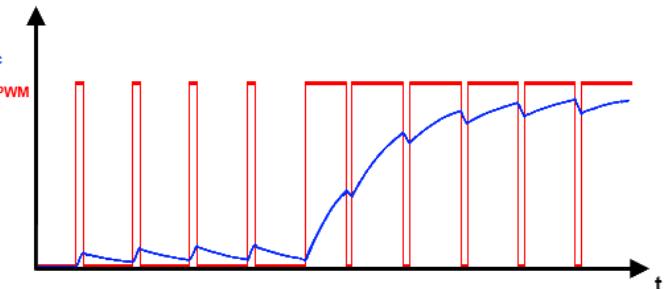
## « TRUE » ANALOG OUTPUT !

What to do if you want to have a beautiful right?

Further increase the time constant RC?

Yes but be careful, take precautions, because problems can occur:

=> issue of time of stabilization between two levels (different cyclical reports)



The more you increase the load with RC, more the capacitor will take some time to stabilize at the desired level:

If we want to create an analogue signal which varies fairly quickly-online problem.

*Which can sometimes save you it is persistence of vision of the eye for an LED application or that an engine that has inertia!*

And then, we can conversely reduce the RC because changing tier will be faster, but the CPU voltage will tend to follow the signal: is the first timeline that we saw earlier.

How to properly calibrate the constant RC? Find the right balance!

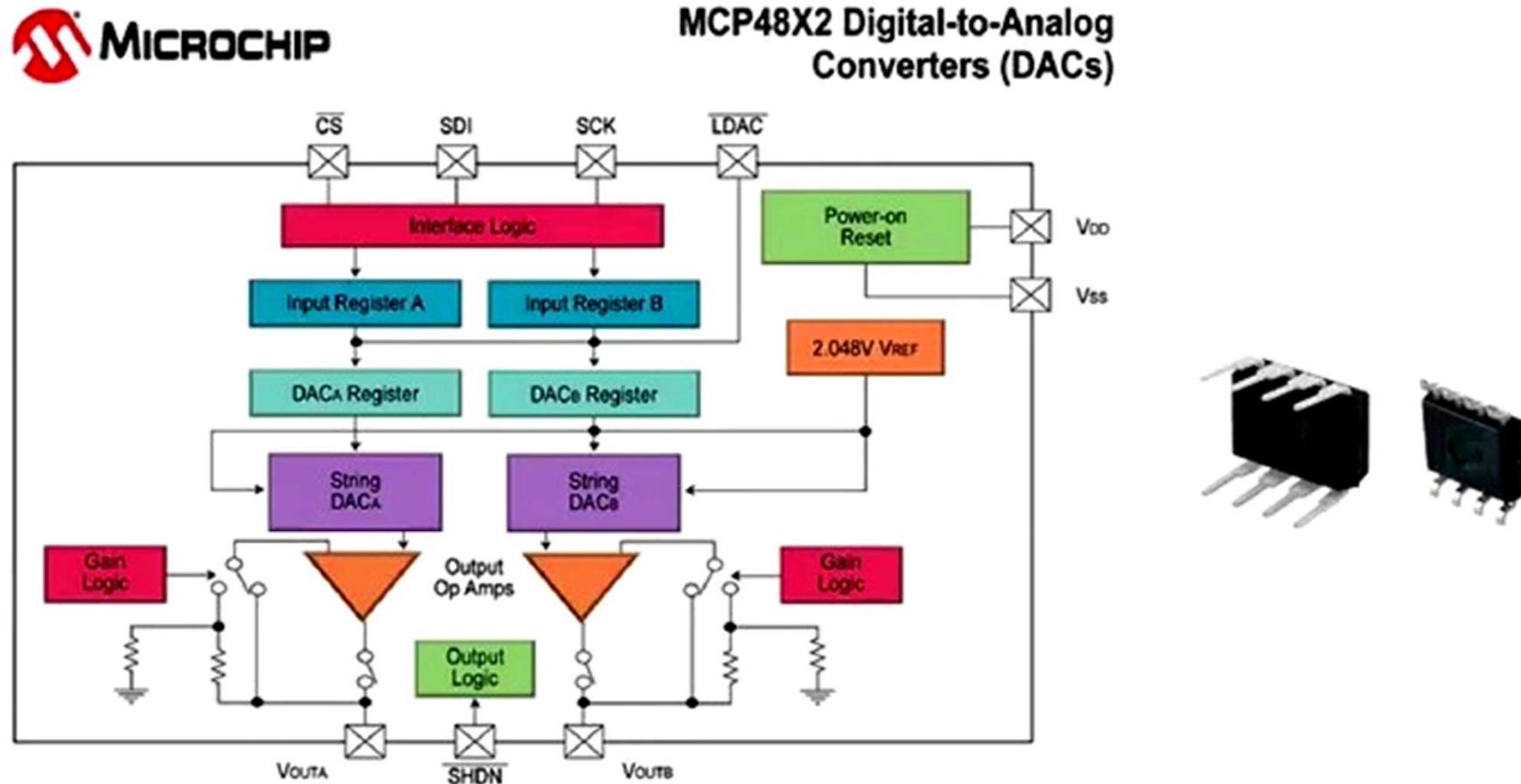
-If a signal that is the closest possible value average: constant great RC.

-If instead we want a signal that is the fastest and the average value is an approximation, then as the low time constant.

-If you want a fast signal and the closest possible value average, you can also change the frequency of the PWM

## TO GO FURTHER

Example of 'real' DIGITAL-to-ANALOG converter:



## CHANGE THE FREQUENCY PWM - 1

The PWM value is set to 8/16 bit when calling `analogWrite()`: `analogWrite(myPWMPin, 128);`

This value is then compared to the value of one of the registers, counters **OCRnX** (8 - bit or 16 - bit):

1 - when the meter is less than the value of PWM: the PIN output high.

2 - When the count is greater than the PWM value, the output pin low.

With the 8-bit timer, the example above generates a square signal because the pin is the same amount of time at the top (of 0 to 127) and then at the bottom (from 128 to 255) level.

With the timer 16 bit => it count of  $2^{16} = 65536$

**On the ATMega:** There are three types of registers timer/counter: **TCCR0B**, **TCCR1B** and **TCCR2B**.

1 - Each timer has two Output Compare Register **OCRnA** & **OCRnB**-online 3timers\*2 OCR = 6-channel PWM

2 - each timer has a prescaler used to generate a frequency = frequency of clock/prescaler

This prescaler is defined by 3bits stored in the 3 low bits of the register of timer/counter: **CS02**, **CS01**, **CS00**

With only three registers (defining the 3 different prescalers), six PWM pins can only be grouped into three pairs, each pair having its own prescaler and thus its own frequency:

Pin 5 and 6: **TCCR0B** (timer0/8bits), Pin 9 and 10: **TCCR1B** (timer1 /) 16 bit), Pin 3 and 11: **TCCR2B** (timer2 / 8bits)

By default on the Arduino Diecimila: Pins 5 and 6: 1 kHz and Pins 9, 10, 11 and 3: 500 Hz (prescaler 64)

=> the frequency of the PWM signal is determined by the value of the prescaler, clock speed, and the resolution of the timer (255 or 65536 for the timer 8-bit or 16-bit respectively).

## CHANGE THE PWM FREQUENCY – 2 – FAST VS PHASE CORRECT PWM

There are two modes of operation for the PWM timers:

- 1 - Fast PWM and
- 2 - Phase-correct PWM.

These modes work almost in the same way, except that once arrived at the TOP, down to the BOTTOM for the correct Phase PWM, is applying the same method of comparison. Advantage: increase the resolution, at the expense of the maximum frequency, which is divided by two.

### FAST PWM :

The following code fragment sets up fast PWM on pins 3 and 11 (Timer 2), using OCR2A as the top value for the timer.

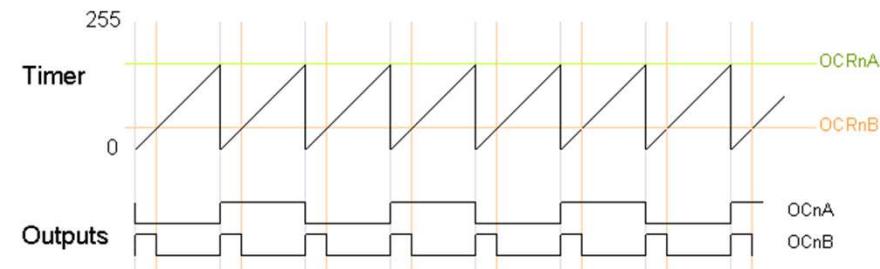
The waveform generation mode bits WGM are set to 111 for fast PWM with OCRA controlling the top limit.

The OCR2A top limit is arbitrarily set to 180, and the OCR2B compare register is arbitrarily set to 50. OCR2A's mode is set to "Toggle on Compare Match" by setting the COM2A bits to 01.

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(WGM22) | _BV(CS22); OCR2A = 180; OCR2B = 50;
```

On the Arduino Duemilanove, these values yield:

- Output A frequency:  $16 \text{ MHz} / 64 / (180+1) / 2 = 690.6\text{Hz}$
- Output A duty cycle: 50%
- Output B frequency:  $16 \text{ MHz} / 64 / (180+1) = 1381.2\text{Hz}$
- Output B duty cycle:  $(50+1) / (180+1) = 28.2\%$



### PHASE CORRECT PWM :

The following code fragment sets up phase-correct PWM on pins 3 and 11 (Timer 2), using OCR2A as the top value for the timer.

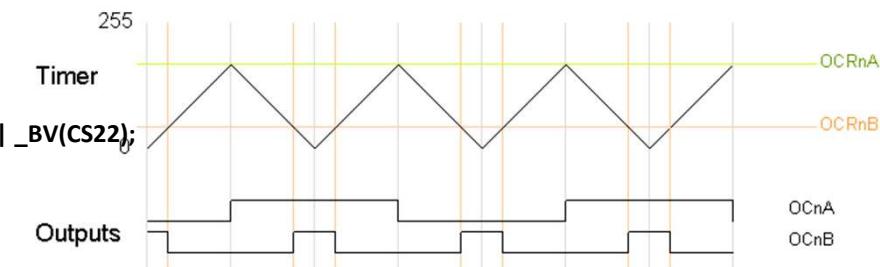
The waveform generation mode bits WGM are set to 101 for phase-correct PWM with OCRA controlling the top limit.

The OCR2A top limit is arbitrarily set to 180, and the OCR2B compare register is arbitrarily set to 50. OCR2A's mode is set to "Toggle on Compare Match" by setting the COM2A bits to 01.

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM20); TCCR2B = _BV(WGM22) | _BV(CS22);
OCR2A = 180; OCR2B = 50;
```

On the Arduino Duemilanove, these values yield:

- Output A frequency:  $16 \text{ MHz} / 64 / 180 / 2 / 2 = 347.2\text{Hz}$
- Output A duty cycle: 50%
- Output B frequency:  $16 \text{ MHz} / 64 / 180 / 2 = 694.4\text{Hz}$
- Output B duty cycle:  $50 / 180 = 27.8\%$



<http://www.righto.com/2009/07/secrets-of-arduino-pwm.html>

## CHANGE THE PWM FREQUENCY - 2

TCCRnA and B registers have different bits:

- Waveform Generation Mode bits – WGM
- Clock Select bits – CS
- Compare Match Output bits – COMnA & COMnB

Table 18-8. Waveform Generation Mode Bit Description

Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on <sup>(1)(2)</sup>
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	—	—	—
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	—	—	—
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

In the part setup () to your Arduino code:

1 - Select the mode (Fast PWM...) in the relevant TCCRnA registers

### 18.11.1 TCCR2A – Timer/Counter Control Register A

Bit (0xB0)	7	6	5	4	3	2	1	0	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

2 - set or clear the bit CS22 CS21, CS20 TCCRnB records relevant to set the prescaler.

### 17.11.2 TCCR2B – Timer/Counter Control Register B

Bit (0xB1)	7	6	5	4	3	2	1	0	TCCR2B
Read/Write	W	W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The frequencies can be then defined as follows according to the Timer that is used:

8-bit

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

16-bit

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

The counter counts from BOTTOM to TOP then restarts from BOTTOM.

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

<http://www.marulaberry.co.za/index.php/tutorials/code/pulse-width-modulation/>

# CHANGE THE PWM FREQUENCY- 3

EXAMPLE: if you want to configure the pins 5 and 6 to generate a PWM signal to the highest frequency possible, i.e. 64kHz-online TCCR0B = xxxxx001 (0-8-bit timer)

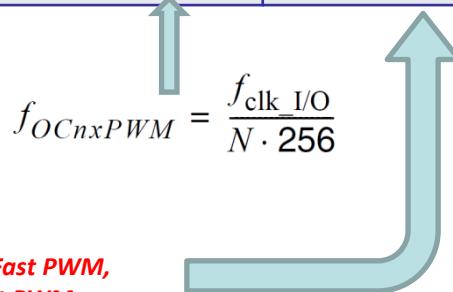
CS02	CS01	CS00	DESCRIPTION	Fréquence (pins 5 et 6 (OC0A and OC0B): TIMER0 (8BIT PWM))	Fréquence [(pins 9, 10, (OC1A, OC1B) TIMER1 (16BIT PWM)) ET [pins 3, 11 (OC2A, OC2B): TIMER2 (8BIT PWM)]]
0	0	0	<b>Timer/Counter Disabled</b>		
0	0	1	<b>No Prescaling</b>	<b>64kHz</b>	<b>32kHz</b>
0	1	0	<b>Clock / 8</b>	<b>8kHz</b>	<b>4kHz</b>
0	1	1	<b>Clock / 64</b>	<b>1kHz (défaut)</b>	<b>500Hz (défaut)</b>
1	0	0	<b>Clock / 256</b>	<b>250Hz</b>	<b>125Hz</b>
1	0	1	<b>Clock / 1024</b>	<b>62.5Hz</b>	<b>31.25Hz</b>

**Example:**

**Clock frequency f=16MHz**

**Prescaler à 64**

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$



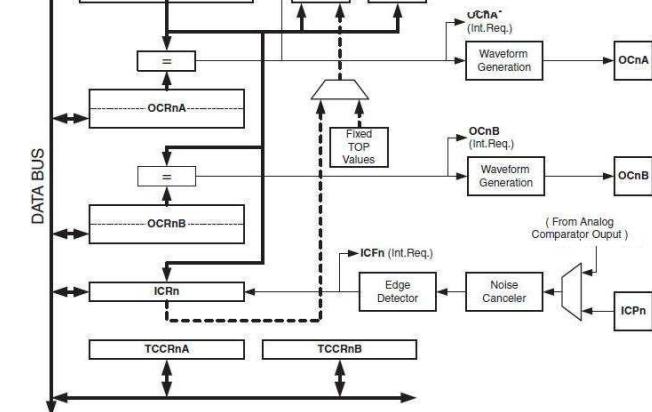
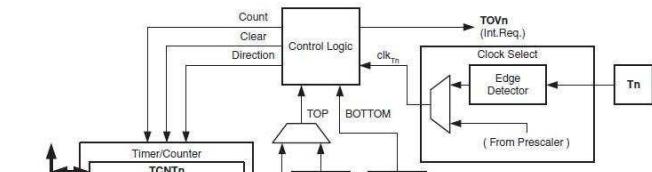
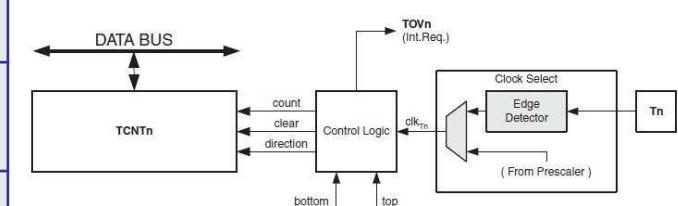
Beware, by default in wiring.c: Timer 0 is initialized to Fast PWM, While Timer 1 and Timer 2 is initialized to Phase Correct PWM.

TCCR1B	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	

Timer/Counter Control Register 1 B

<http://www.righto.com/2009/07/secrets-of-arduino-pwm.html>

TCCR0B	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
<i>Timer/Counter Control Register 0 B</i>								



## CHANGE THE PWM FREQUENCY - 4

### FIRST METHOD:

```
//First, clear prescaler bits  
int prescalerVal = 0x07; // => prescalerVal = binaire "00000111"  
TCCR0B &= ~prescalerVal; //registre TCCR0B = ET logique entre TCCR0B et reverse (tilde) de prescalerVal "1111000"  
//Then, put good bits values to prescaler  
int prescalerVal = 1; //put prescalerVal to binary "00000001"  
TCCR0B |= prescalerVal; // OR logic between previous value of TCCR0B "1111000" and binary "00000001"  
⇒ Result "11111001" => CS02, CS01 has been cleared and CS00 activated !  
⇒ No prescaling => max frequency
```

### SECOND METHOD:

```
//First, clear prescaler bits  
// do 1<<2 => « 00000001 » => « 00000100 »  
// ~ « 00000100 » = « 11111011 »  
// TCCR0B ET « 11111011 »  
TCCR0B &= ~(1<<CS02);  
//idem for CS01, // do 1<<1 => « 00000001 » => « 00000010 »  
TCCR0B &= ~(1<<CS01);  
//put CS00 to good value  
TCCR0B |= (1<<CS00); //OR with TCCR0B.
```

<http://www.atmel.com/Images/doc2505.pdf>

<http://www.avrfreaks.net/>

<https://sites.google.com/site/qeewiki/books/avr-guide/timers-on-the-atmega328>

<http://www.marulaberry.co.za/index.php/tutorials/code/pulse-width-modulation/>

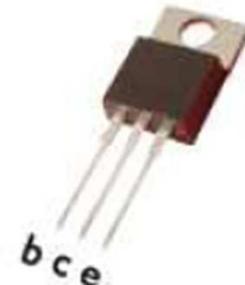
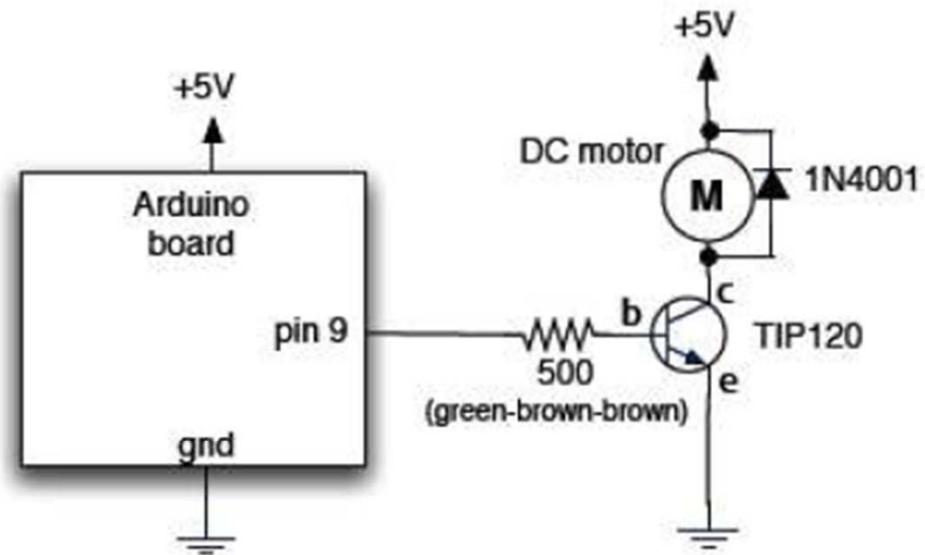
## APPLICATION OF PWM: WIRE A DC MOTOR

A transistor is used to drive the motor.

It sends a signal that the duty cycle varies according to the desired speed.

The so-called led "free wheel" allows to evacuate the current created when the engine slowed down so that it is no longer powered.

The supply voltage of the motor can be different (of course)



# TP3-A

# APPLICATION DU PWM: CABLER UN MOTEUR A COURANT CONTINU

```
/* Contrôle d'un moteur DC - en utilisant une sortie PWM.  
* Attention, les pin PWM 5 et 6 ont une limitation. */  
int pinMoteur = 9; // pin contrôlant le moteur (PWM pin)  
  
void setup(){  
    //En PWM il ne faut pas assigner la pin en output  
}  
void loop(){  
    MoteurVitessePWM();  
    MoteurAccelerationPWM();  
}  
  
// Contrôle de la vitesse du moteur en contrôlant le % du cycle de service PWM  
// sur la pin 9 (de 0 à 255 pour le % du cycle de service ).  
//  
void MoteurVitessePWM(){  
    int vitesse1 = int(255) / 3; // High 33% du cycle  
    int vitesse2 = int(255) / 2; // High 50% du cycle  
    int vitesse3 = 2 * 255 / 3;  
    int vitesse4 = (int)255; // High 100% du cycle  
    analogWrite( pinMoteur, vitesse1 ); delay( 1000 );  
    analogWrite( pinMoteur, vitesse2 ); delay( 1000 );  
    analogWrite( pinMoteur, vitesse3 ); delay( 1000 );  
    analogWrite( pinMoteur, vitesse4 ); delay( 1000 );  
    analogWrite( pinMoteur, vitesse3 ); delay( 1000 );  
    analogWrite( pinMoteur, vitesse2 ); delay( 1000 );  
    analogWrite( pinMoteur, vitesse1 ); delay( 1000 );  
    analogWrite( pinMoteur, LOW );  
}  
  
// Contrôle plus fin de l'accélération du moteur via PWM.  
// NB: Selon la qualité du moteur, celui-ci peut avoir du mal à décoller  
// lorsque le % du cycle de service est assez bas.  
// Un option est d'envoyer une impulsion pour démarrer/décoller le moteur.  
void MoteurAccelerationPWM(){  
    // Impulsion de démarrage (75%)  
    //analogWrite( pinMoteur, 191 );  
    //delay(50); // Acceleration  
    for( int i = 30; i<= 255; i++ ){  
        analogWrite( pinMoteur, i );  
        delay(50); // delay pour avoir un progression  
    }  
    // pause de 2 secondes a plein régime  
    delay( 2000 );  
    // Deceleration  
    for( int i = 255; i>=0; i-- ){  
        analogWrite( pinMoteur, i );  
        delay(50); // delay pour avoir un progression  
    }  
}
```

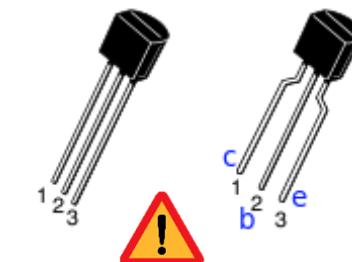
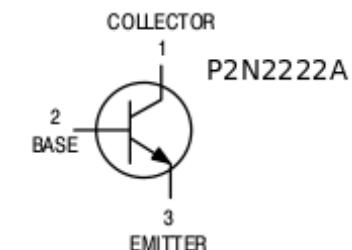
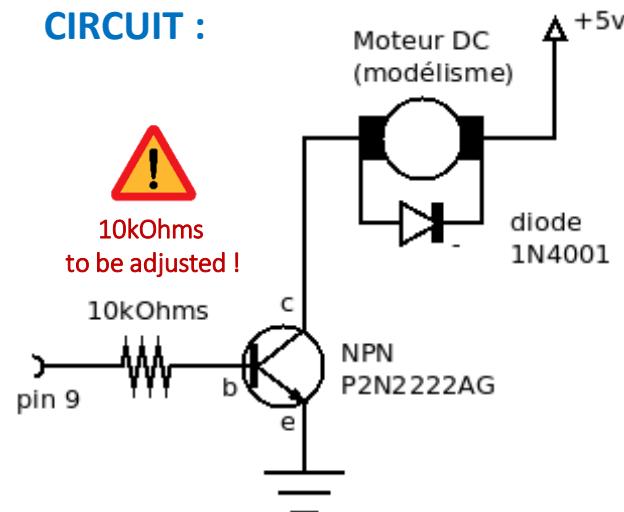
<http://arduino103.blogspot.com/2011/05/controle-moteur-dc-via-transistor-pwm.html>

## Control DC motor via transistor (PWM):

as we have seen, the analog speed control is not really appropriate because we indeed have to overcome the inertia at startup.

With the same mounting, it is possible to fine-tune the speed with the PWM control:

### CIRCUIT :



Beware of transistor different pinout from manufacturer  
=> Always check the datasheet



QUESTION: pourquoi doit-on mettre une diode de roue libre en parallèle du moteur ?

<= BACK



97

## TP3-B

# APPLICATION OF PWM PINS: EXAMPLE WITH PHOTO-RESISTANCE

If the man can control a potentiometer, how to make the environment that controls (e.g. light).

We will take a photo-resistance (its resistance is dependent on the light received). However the Arduino cannot directly measure resistance (it does detect a voltage) we have implemented a voltage divider.

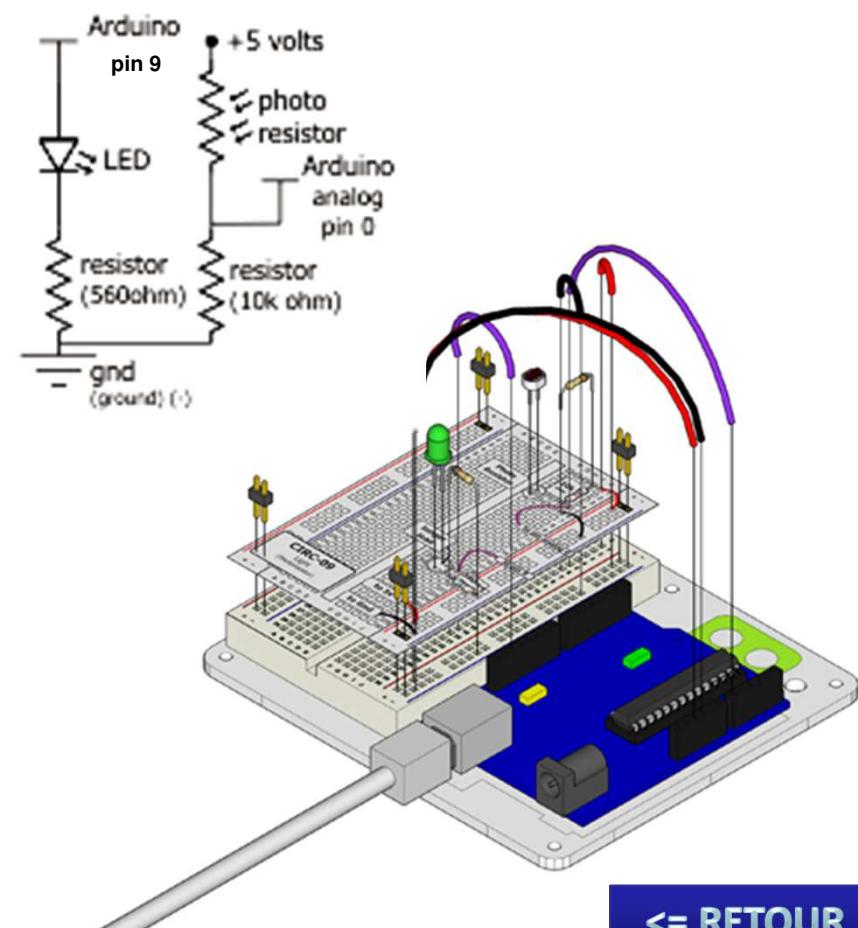
The measurement is done on an analog input.

The strength of the photoresistor decreases when the light increases

```
/* A simple program that will change the intensity of a LED based on the
 * amount of light incident on the photo resistor. */
//PhotoResistor Pin
int lightPin = 0; //the analog pin the photoresistor is connected to the
//photoresistor is not calibrated to any units so this is simply a raw sensor value
//(relative light)
int ledPin = 9; //the pin the LED is connected to
//we are controlling brightness so, we use one of the PWM (pulse
//width modulation pins)
void setup()
{
    Serial.begin(9600);
    pinMode(ledPin, OUTPUT); //sets the led pin to output
}

void loop()
{
    int lightLevel = analogRead(lightPin); //Read the lightlevel
    Serial.print(lightLevel); //Affiche le lightlevel sur le moniteur série
    //ajuster les valeurs en fonction du lightlevel: adjust 180 to 610 to span 0 to 255
    lightLevel = map(lightLevel, 250, 700, 0, 255);
    Serial.print(","); Serial.println(lightLevel);
    delay(50);
    lightLevel = constrain(lightLevel, 0, 255); //make sure the value is between 0 et 255
    analogWrite(ledPin, lightLevel); //write the value
}
```

<http://www.oomlout.com/a/products/ardx/circ-09>



<= RETOUR

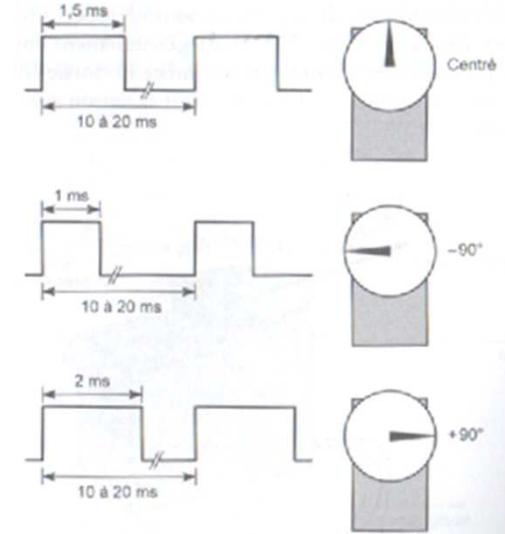
# SERVO-MOTOR

A servo motor is a type of electric motor. It is a device used to control the direction of a controlled car model. On a servo-motor, the angle can vary from a fixed angle between 0 and 180 degrees depending on the signal being sent.



A servo-motor includes:

- A (continuous) electric motor, usually small enough.
- Gear reducer output of the engine (for less speed and more torque or force).
- A sensor type "potentiometer" connected to the output (resistance which varies depending on the angle, which allows to measure the angle of rotation on the output shaft).
- A servo electronic control the position/rotation of the output shaft to hold it in the correct position.



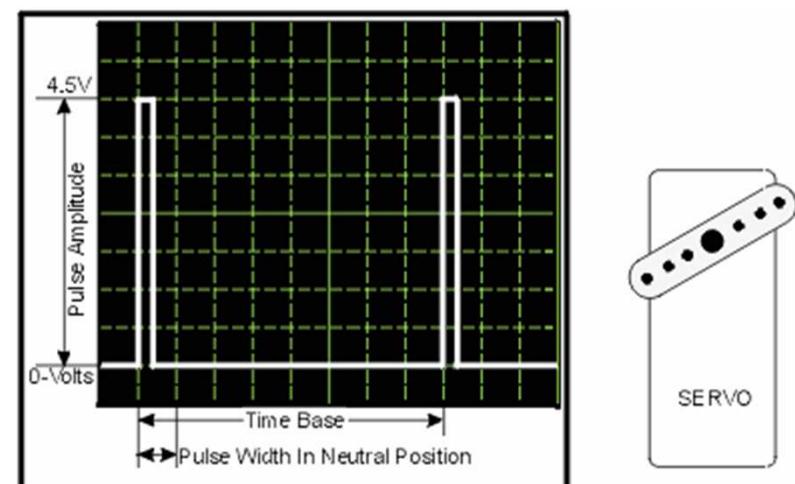
To manage a servo-motor?

Just send a pulse whose duration will determine the angle of the servo-motor.

This pulse time is some ms should be repeated at regular intervals (every 10 ms to 20ms) to maintain the position.

Standard values:

1ms = - 90 ° | 1.5 ms = 0 ° (centered position): 2 ms = + 90 °



The graph below shows the correspondence between the pulse length and angle of the servo-motor.

# SERVOMOTOR WIRING

A servo motor is connected with only 3 wires:  
the mass, the + 5V and the command pulse.

The black wire is connected to the 0V or GND.

The Red wire is connected to the 5V.

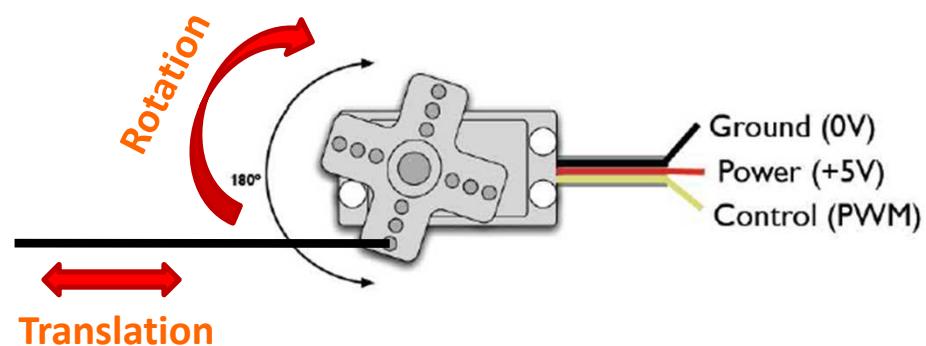
The yellow wire or white is connected to an output Digital PWM

One sends a signal that we do vary according to the direction  
and the desired position.

Wiring the actuator on an output Digital PWM: D11, D3, D10,  
D9, D6, D5.

Program or even impulse drive is still possible  
but it would be tedious,

Arduino owns the Servo.h library, which  
allows you to easily take control.



This sub-program "servo.h" must be included in the program, it is easy to order.

# TP3-C

# SERVO MOTOR

//File > Examples > Library-Servo > Sweep (example from the great arduino.cc site  
//check it out for other ideas)

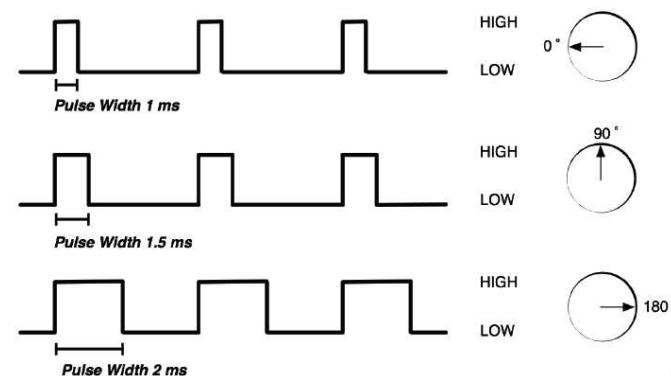
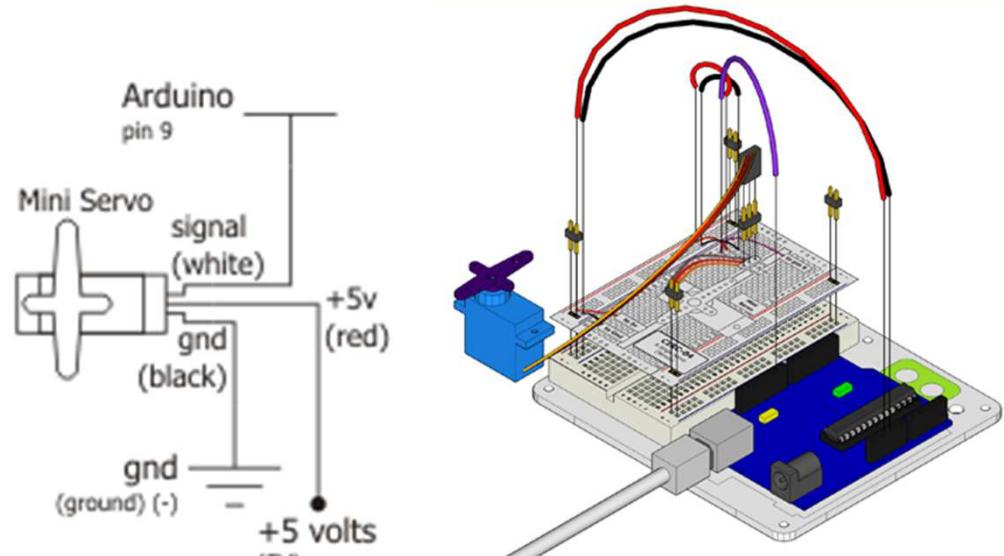
```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
// a maximum of eight servo objects can be created

int pos = 0; // variable to store the servo position

void setup()
{
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
  {
    // in steps of 1 degree
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }

  for(pos = 180; pos>=1; pos-=1) // goes from 180 degrees to 0 degrees
  {
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
}
```



Attach(): allows to define and attach a  
Detach() servo connection: obviously the opposite  
Write(): main function of the servo control, it allows to generate the pulse train, the setting is the angular position of 0 to 180 ° (not the width 90 ° pulse) is the resting position or median  
Read(): allows to know the position of the servo by returning the value of the last call to the Write()

<http://www.oomlout.com/a/products/ardx/circ-04>

<= BACK



103

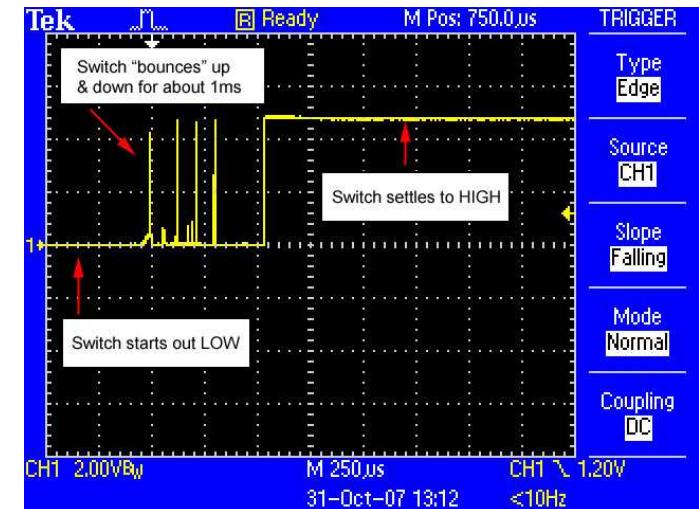
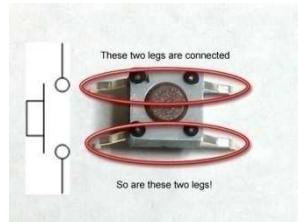
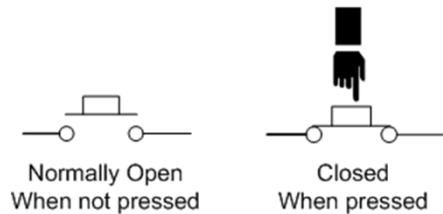
## III-5 DEBOUNCING

## III-5 DEBOUNCING

# BUTTON DEBOUNCING

When you press/release button, there is often the appearance of parasites for a few ms.

This parasites appear only during the times where is pushed / release the push button.



If you count the number of pressures (to make a counter), these parasites just disrupt the proper functioning of the software.

The problem is hardware... and parasites are added pressures "ghosts" .

There are several ways to correct the problem:

- software (Software debouncing) => introduce a software delay in the program
- hardware (Hardware debouncing) => use an ability of deworming (Hardware debouncing) (and possibly a trigger) Inverter Schmidt (Jeremy Blum Tutorial)

See the article of IkaLogic.com explaining how to debounce circuits (way software and hardware) or article from LadyADA <http://www.ladyada.net/learn/arduino/lesson5.html>

# SOFTWARE DEBOUNCING

The transitory phenomenon is only during a few ms, would suffice to two successive readings of the entry (after a delay of a few ms) and make sure that it is not a transitional phase. Reading the State of 10ms later entry would be the same as 10 ms earlier:  
*anyone arriving to press and release a button in less than 10 milliseconds!*

/\* Debounce: each time the input pin goes from LOW to HIGH (e.g. because of a push-button press), the output pin is toggled from LOW to HIGH or HIGH to LOW. There's a minimum delay between toggles to debounce the circuit (i.e. to ignore noise). The circuit: \* LED attached from pin 13 to ground | \* pushbutton attached from pin 2 to +5V | \* 10K resistor attached from pin 2 to ground  
Created 21 November 2006, by David A. Mellis, modified 3 Jul 2009, by Limor Fried and J. Grisolia et B. Detroussel Oct 2011 \*/

// 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
```

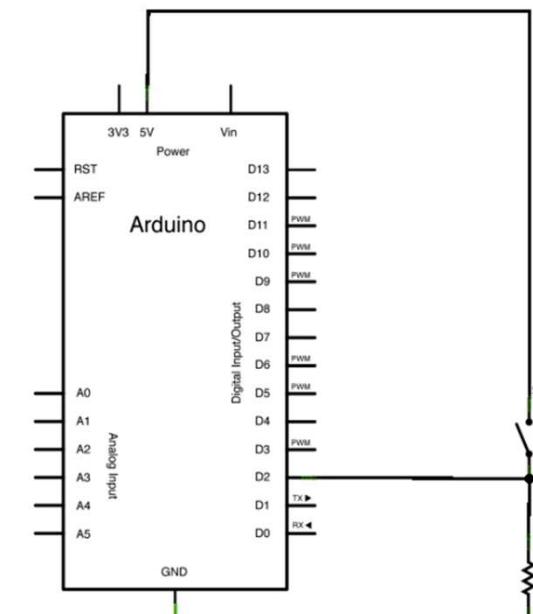
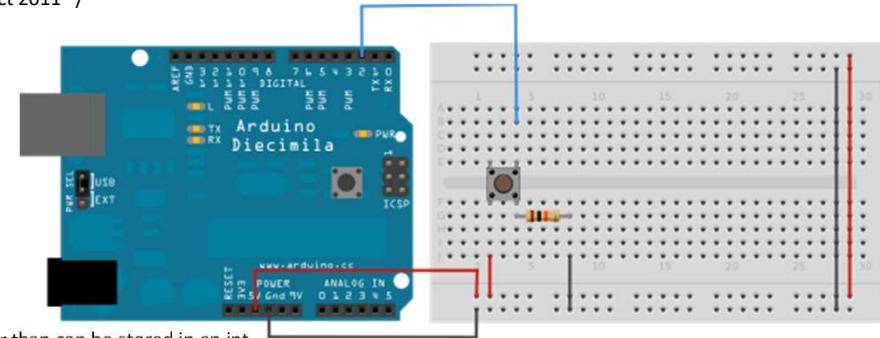
// Variables will change:

```
int ledState = HIGH; // the current state of the output pin
int buttonState; // the current reading from the input pin
int lastButtonState = LOW; // the previous reading from the input pin
int compteur_filtre; //compteur filtré
int compteur_non_filtre; //compteur non filtré
```

// the following variables are long's because the time, measured in milliseconds, will quickly become a bigger number than can be stored in an int.

```
long lastDebounceTime = 0; // the last time the output pin was toggled
long debounceDelay = 50; // the debounce time; increase if the output flickers
```

```
void setup() {
  Serial.begin (9600);
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}
void loop() {
  // read the state of the switch into a local variable:
  int reading = digitalRead(buttonPin);
  // check to see if you just pressed the button (i.e. the input went from LOW to HIGH),
  //and you've waited long enough since the last press to ignore any noise: If the switch changed, due to noise or pressing:
  if (reading != lastButtonState) {
    lastDebounceTime = millis(); // reset the debouncing timer
    if (reading == 1) { compteur_non_filtre = compteur_non_filtre + 1; }
  }
  if ((millis() - lastDebounceTime) > debounceDelay)
    // whatever the reading is at, it's been there for longer than the debounce delay, so take it as the actual current state:
    if ((reading == 1) && (reading != buttonState)) { compteur_filtre = compteur_filtre + 1; }
    buttonState = reading;
  }
  digitalWrite(ledPin, buttonState); // set the LED using the state of the button
  lastButtonState = reading; // save the reading. Next time through the loop, it'll be the lastButtonState
  Serial.print("Compteurs:");
  Serial.print(compteur_non_filtre);
  Serial.print(", ");
  Serial.println(compteur_filtre);
}
```



<http://www.arduino.cc/en/Tutorial/Debounce>

<= RETOUR



106

# HARDWARE DEBOUNCING

Debouncing is done using a capacitor that can absorb the noise pulses. But alone it is not sufficient. In the case of a pull-up montage, there is a capacity of  $100\text{nF}$  (in series with a resistor of  $10\text{k}\Omega$ ) that is a time constant  $\tau = \text{RC} = 1\text{ms}$

## When you press the button:

the capacitor is instantly discharged (bypassed by the button) and the input goes LOW. If there are parasites (peaks to + 5 volts) at the button, they will be absorbed by the capacitor.

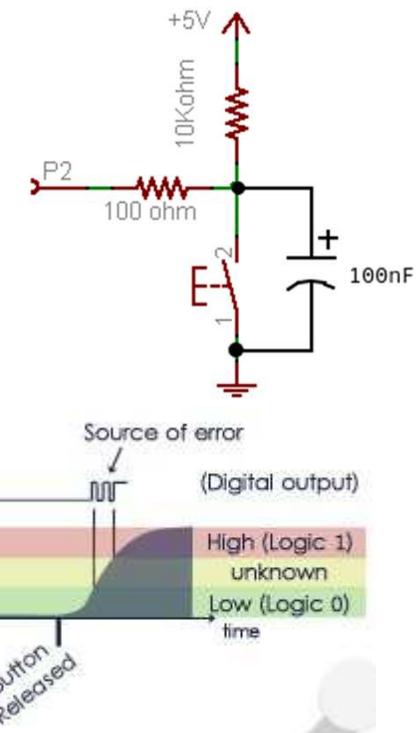
## When you release the button:

the voltage is applied on the capacitor and it will load fast enough. However, the load is not instantaneous but gradual (depends on  $\tau$ ) => the 5 volts are applied gradually to the entry =>

The load curve spends some time in the zone of uncertainty where the microcontroller don't know if it's still a 0 or a 1 logic. The noise is always at the point where the button is released (but in a different form).

To avoid this new type of interference, you must insert a Schmitt trigger on the entry.

Thus, as long as the voltage is above the minimum logic 1, the trigger does not change from State to 1... and vice versa as long as the voltage will not be passed under the minimum threshold of the trigger (0 logic), the trigger don't pass again to 0 (cf slide  $V_{IL}$  and  $V_{IH}$ ).



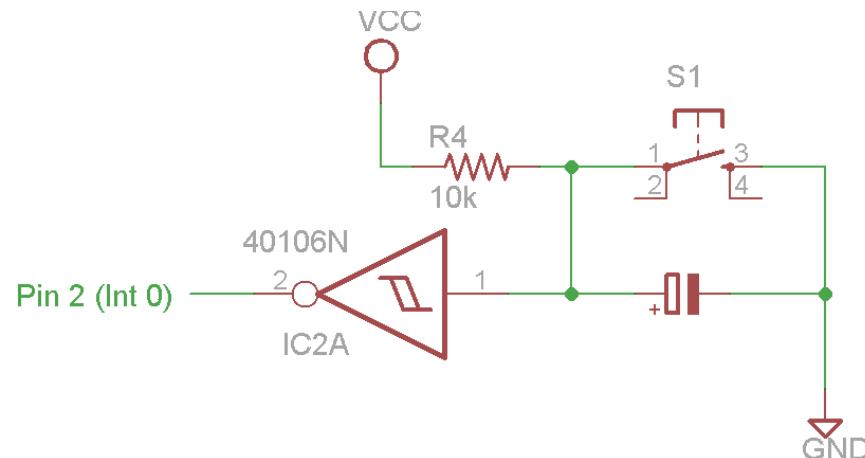
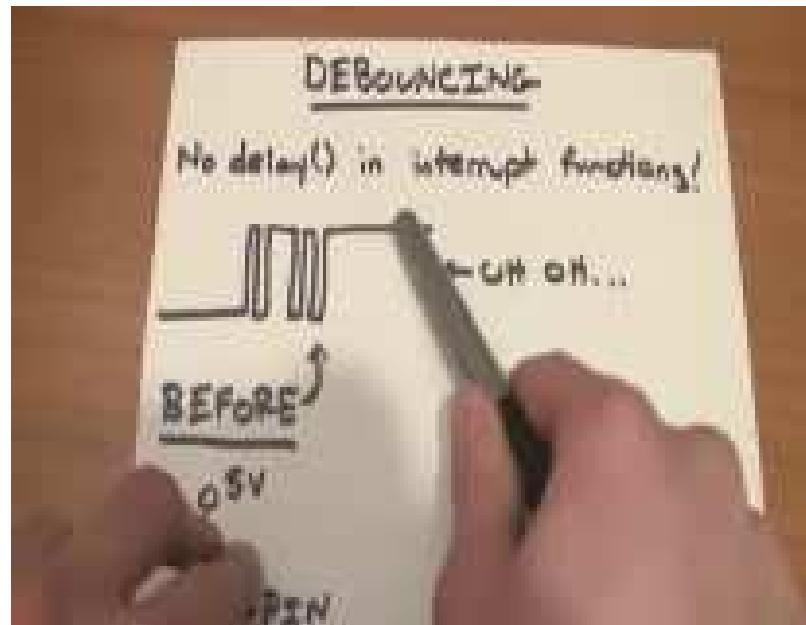
# HARDWARE DEBOUNCING

//Program by Jeremy Blum www.jeremyblum.com  
//Debounced Switch Input via hardware

```
int switchPin = 2;  
int ledPin = 13;  
boolean lastButton = LOW;  
boolean ledOn = false;  
void setup()  
{  
    pinMode(switchPin, INPUT);  
    pinMode(ledPin, OUTPUT);  
    Serial.begin(9600);  
}  
void loop()  
{  
    Serial.println(digitalRead(switchPin));  
    if (digitalRead(switchPin) == HIGH && lastButton == LOW)  
    {  
        ledOn = !ledOn;  
        lastButton = HIGH;  
    }  
    else  
    {  
        //lastButton = LOW;  
        lastButton = digitalRead(switchPin);  
    }  
  
    digitalWrite(ledPin, ledOn);  
}
```

## PART LIST:

- Inverting Schmitt Trigger: <http://us.element-14.com/stmicroelectronics/m74hc14b1r/ic-hex-inverter-schmitt-trigger/dp/89K0862>
- 10k ohm Resistor: <http://us.element-14.com/multicomp/mccfr0w4j0103a50/resistor-carbon-film-10kohm-250mw/dp/58K5002>
- 10uF Capacitor: <http://us.element-14.com/multicomp/mcgpr100v106m6-3x11/capacitor-alum-elect-10uf-100v/dp/70K9661>



<http://jeremyblum.com/2011/03/07/arduino-tutorial-10-interrupts-and-hardware-debouncing/>

<= RETOUR



108

## III – 6 INTERRUPTS

### III – 6 - 1 INTERRUPTS

# INTERRUPTS

- DEFINITION:** an interruption, as its name implies, interrupts a running program (main program) to run a subroutine in the µcontroller through the interrupt service routine (ISR).
- It ends with a return from interrupt instruction that allows the microcontroller to return to the main program where he had left it.

An interruption can occur in two ways:

- [ 1 - material Interruption: she is asked by the hardware to perform a processing (read a key on the keyboard for example). In this case, it triggers an IRQ (Interrupt ReQuest) with a number unique to the port to which the equipment is connected.
- 2 - Software interrupt: she is requested by the software running.

**Why do?** Interruptions are a very powerful way to monitor an external event. The use of interrupts releases including the microcontroller to do other things until an expected event does not occur.

For example, Suppose a program in which a push button must cause a change:

- Classical programming: we put a test button in the program, without being sure that it will not support for example very brief.
- Programming interruption: there is more need to test, because support of the button will generate the call to subprogram

Another example :

- other sensors have a dynamic interface similar role, such as a sensor of sounds trying to detect a noise or a sensor infra-red (phototransistor) trying to detect an obstacle

To go further: [http://www.iutc3.unicaen.fr/~fougep/assembleur/les\\_chapitres\\_du\\_cours/interruptions.html](http://www.iutc3.unicaen.fr/~fougep/assembleur/les_chapitres_du_cours/interruptions.html)  
[http://www.technologuepro.com/microprocesseur/chap5\\_microprocesseur.htm](http://www.technologuepro.com/microprocesseur/chap5_microprocesseur.htm)

### III – 6 - 1 HARDWARE INTERRUPTS

# INTERRUPTS WITH ARDUINO

## attachInterrupt (interrupt, fonction, mode):

Specifies the function to call when an external interruption occurs.

Most of the Arduino have 2 external interruptions: n ° 0 on digital Pin 2 and no. 1 on digital Pin 3 except, the Arduino Mega, who has four more: n ° 2, 3, 4, 5 on Pin 21, 20, 19, 18

Parameters:

- interruption: the interrupt (int type)
- function : the function to call when the interrupt occurs; the function should receive no parameters and returns nothing. This function is also called an interrupt service routine (or ISR).
- mode: defines the way the external interruption must be taken into account. Four constants have valid predefined values:
  - LOW: for triggering the interruption when the PIN is at the bottom
  - CHANGE level: for triggering the interruption when the PIN changes state low/high
  - RISING: for release of the interruption when the PIN changes from bottom to top (rising edge)
  - FALLING: for release of the interruption when the PIN goes from the top state to the down state (falling edge)

Example: attachInterrupt(0, blink, CHANGE); //attach external interrupt n°0 (Pin2) to fonction blink

Attention:

- the inside of the function attached to the interruption, the delay function does not work and does not increment the value returned by millis. Why?
- Data series received during execution of the function is lost.
- You must declare all variables used in the function attached to the interruption "volatile": allows to make the variable accessible from all contexts, including in interruptions.

# BEHIND THE SCENE...

## 12.4 Interrupt Vectors in ATmega328 and ATmega328P

Vecteurs appelés par l'interruption:

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address <sup>(1)</sup>	Source	Interrupt Definition
1	0x0000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x00E	TIMER2 COMPA	Timer/Counter2 Compare Match A

### 13.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0
(0x69)	-	-	-	-	ISC11	ISC10	ISC01	ISC00
ReadWrite	R	R	R	R	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EICRA

Table 13-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

### 13.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
0x1D (0x3D)	-	-	-	-	-	-	INT1	INT0
ReadWrite	R	R	R	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EIMSK

### 13.2.3 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
0x1C (0x3C)	-	-	-	-	-	-	INTF1	INTF0
ReadWrite	R	R	R	R	R	R	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

EIFR

## External interrupt

```
// on a initialisé un poussoir et une led
//Setup - INT1
EICRA = 1<<ISC11 ; // falling edge
EIMSK = 1<<INT1 ;
sei() ; Set global interrupt flag (I) in SREG (status register)
```

```
// ISR(INT1_vect) {
    LedToggle ;
}
```

Tell the microprocessor in the correct address (vector)

## Arduino attachInterrupt

```
void setup() {
    pinMode ...
    attachInterrupt (pin/no, FaireQqch, LOW);
}
```

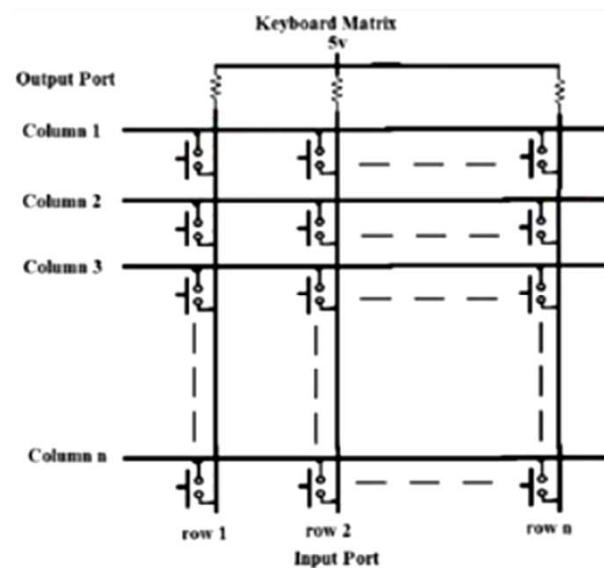
```
void FaireQqch () {
    LedToggle ;
}
```

Arduino version more simple, but of course more slowly because they are so procedures that test...

EIFR

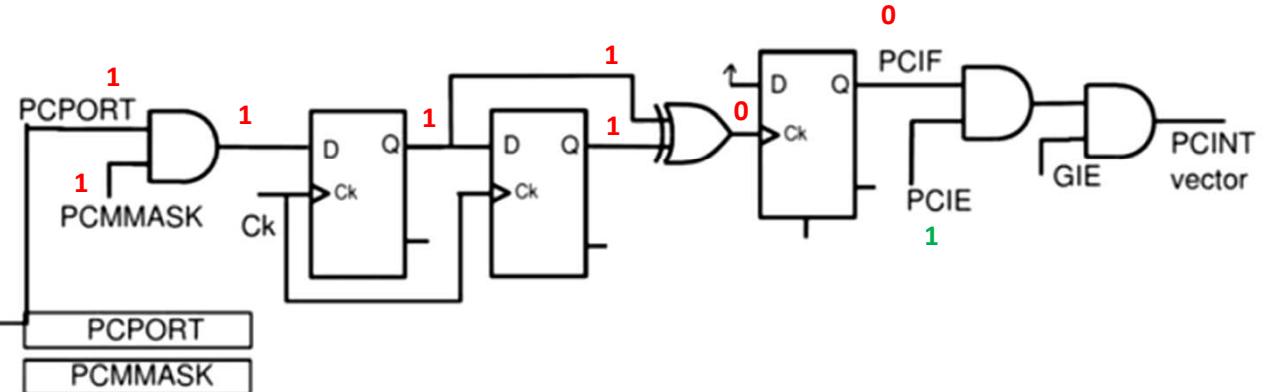
<http://www.atmel.com/webdoc/avrdoc/avrasm/avrasm.htm>

# TO GO FURTHER...

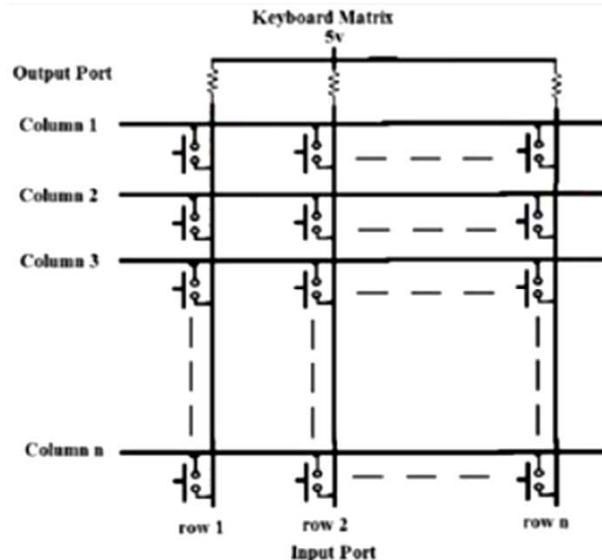


With pull-up: logic state 1 without pushing

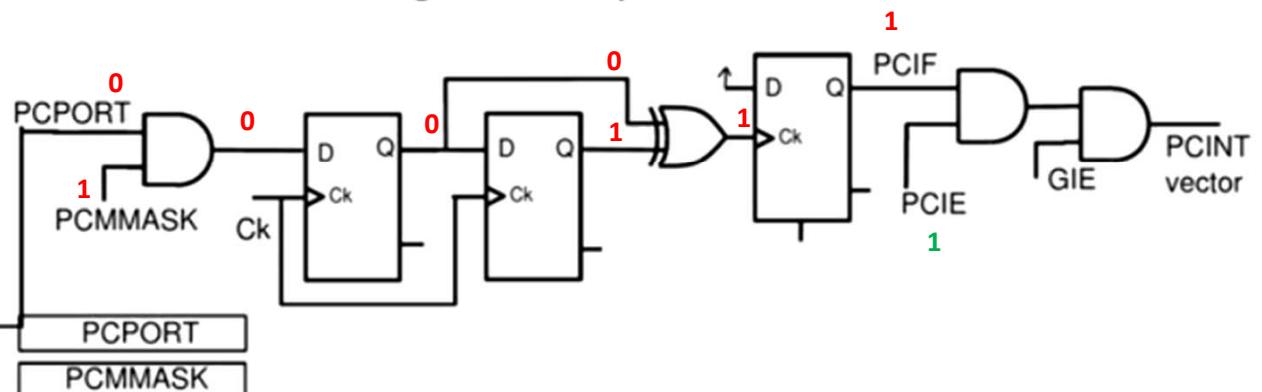
## Pin change interrupt



Pushing : 1 => 0



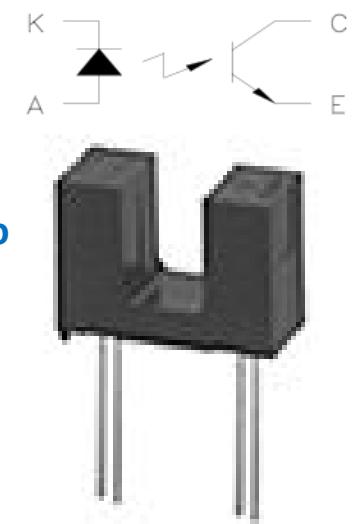
## Pin change interrupt



# INTERRUPTS PROGRAM

An opto-coupler (or optocoupler) in fork combines in a single plastic case:

- 1 - a LED which emits infra-red and
- 2 - a phototransistor that detects the light emitted by the LED light.



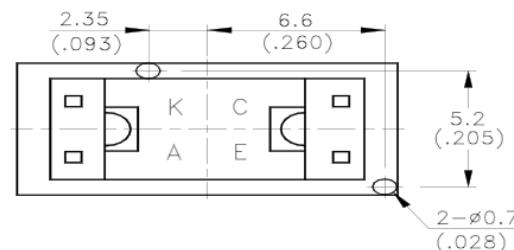
The two components are head to head and separated by a slot in which it is possible to pass an encoder wheel or rotating object any.

The whole forms a miniature photoelectric barrier.

This component has 4 pins:

- 2 first rated pins A and K correspond to the pins of the LED
- 2 other rated pins C and E for the collector and the transmitter of the phototransistor. The base of the transistor is here a "photobase" receiving photons from the LED instead to receive a current

*Example: optocoupler in fork: Liton LTH 307-01*



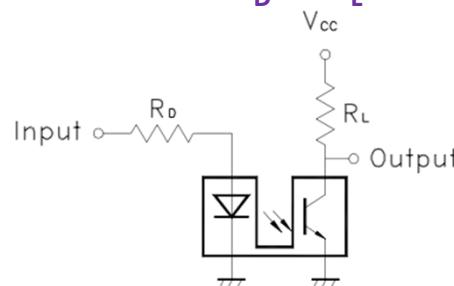
# HOW DOES IT WORKS ?

By setting the value of the resistors used cleverly, you get with  $V_{CC} = 5V$ :

- { 0V (or low) in out of the phototransistor when it will be lighted
- 5V (or high) the phototransistor output when it will not be lit.

=> It will be possible to detect the passage of an object in the slot.

How to choose  $R_D$  et  $R_L$  ?



For resistance  $R_D$  to the LED :

LED Forward voltage  $\sim 1,1$  V (cf datasheet)

=> for a  $I_{LED} = 15mA$  (under  $V_{CC}=5V$ ):  $R_D = U/I = (5-1,1)/0.015 = 260 \Omega$

=>  $270\Omega$  standardized value

For resistance  $R_L$  to photo-transistor, goal is :

1 - 0V on  $R_L$  (i.e. Output to 5V) When the phototransistor is not lit  
and

2 -  $\sim 5V$  on  $R_L$  (i.e. Output to 0V, more precisely  $\sim 1V$ ) when lit.

- If  $I_{LED} = 0mA$ :

$R_L$  has no significance when the phototransistor is not lit because

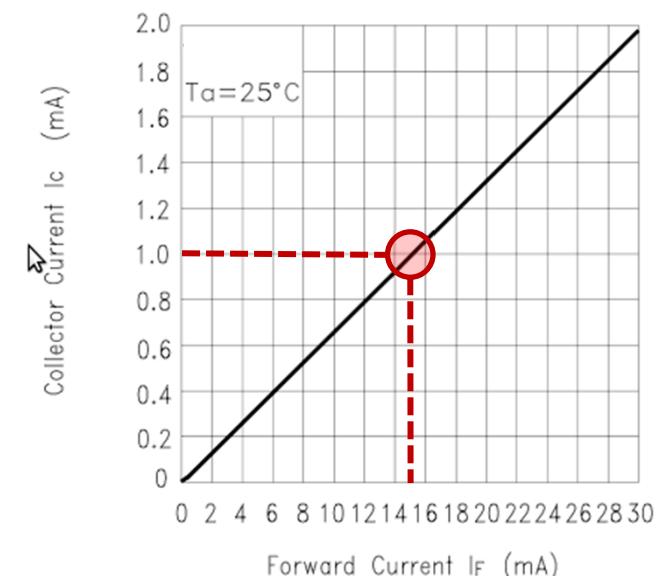
$I_C = 0mA$  and then  $U_L = R_L \times I_C = R_L \times 0 = 0 V$

- If  $I_{LED} = 15mA$ : =>  $I_C = 1mA$  et  $U_{CE} \sim 1V$  (typical of a PN junction).

For 5V on  $R_L$  with  $I_R = 1mA$  equals :  $R_L = (5V-1V)/0.001 = 4 k\Omega \Rightarrow 4,7 k\Omega$   
standardized value

We can take :  $R_D = 270 \Omega$  and  $R_L = 4.7 k\Omega$

Fig.3 Collector Current vs.  
Forward Voltage



# INTERRUPTS PROGRAM

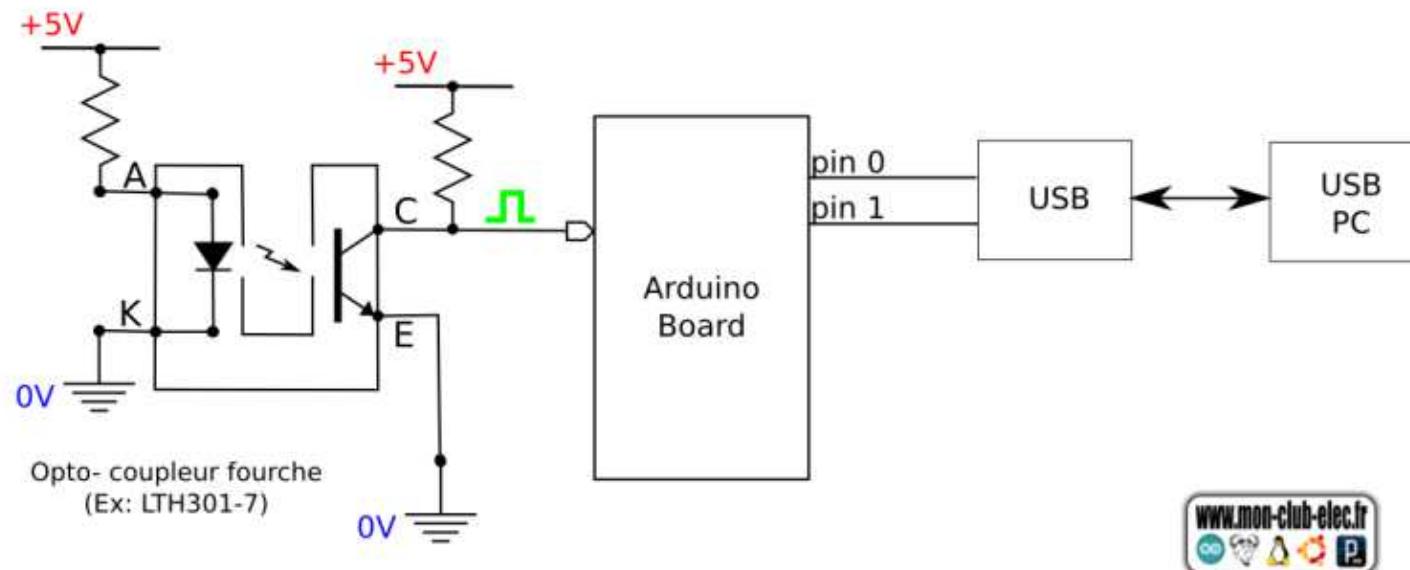
Realize the following program:

the program counts the pulses from an opto-Coupler in fork using the external interruption n ° 0 on pin 2 of the Arduino board.

The result is displayed in the Terminal series of the Arduino software.

This program uses the following features:

- using the connection set to the PC (pins 0 and 1 (via USB cable))
- uses the external interrupt 0 (pin 2)



The interrupt handler should be called whenever a rising edge is detected on the PIN:

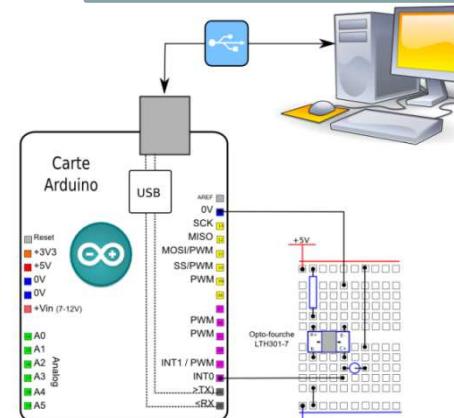
- the count variable is incremented
- the result is displayed in the Terminal of the Arduino software window.

## TP3-F

# INTERRUPTS PROGRAM

```
/* Ce programme compte les impulsions en provenance d'un opto-coupleur en fourche  
en utilisant l'interruption externe n°0 sur la broche 2 de la carte Arduino.  
Le résultat est affiché dans le Terminal Série du logiciel Arduino. */  
// --- Fonctionnalités utilisées ---  
// Utilise la connexion série vers le PC  
// Utilise l'interruption externe 0 (broche 2)  
// Utilise optocoupleur infra-rouge en fourche, type LTH301-7,  
  
// --- Déclaration des variables globales ---  
volatile int comptageImpulsion=0; // variable accessible dans la routine interruption externe 0  
  
void setup() {  
  
Serial.begin(115200); // initialise connexion série à 115200 bauds  
attachInterrupt(0, gestionINT0, RISING); // attache l'interruption externe n°0 à la fonction  
gestionINT0() // mode déclenchement possibles = LOW, CHANGE, RISING, FALLING  
}  
  
void loop(){ // debut de la fonction loop()  
// tout se passe dans la fonction de gestion de l'interruption externe  
} // fin de la fonction loop()  
  
// ----- fonction de gestion l'interruption externe n°0 (broche 2) -----  
// cette fonction est appelée à chaque fois que l'interruption a lieu selon le mode configuré  
(LOW, //CHANGE, RISING, FALLING)
```

```
void gestionINT0() // la fonction appelée par l'interruption externe n°0  
comptageImpulsion=comptageImpulsion+1; // Incrémente la variable de comptage  
// ATTENTION : delay() et millis() non dispo ici - données série perdues  
Serial.print("Nombre impulsions = "); //--- affiche le nombre d'impulsions sur le port série  
Serial.println(comptageImpulsion);  
}
```



Pour aller plus loin, interfaçage avec Processing: [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.OutilsProcessingProgGraphOscilloSimple](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.OutilsProcessingProgGraphOscilloSimple)

### III – 6 - 1 SOFTWARE INTERRUPTS

# SOFTWARE INTERRUPT WITH TIMER 2

How to make 'software' interrupts at regular intervals: library **MSTimer2**

This program generates:

- an interruption all the seconds using the Timer 2 (8 - bit)
- and briefly turns a LED

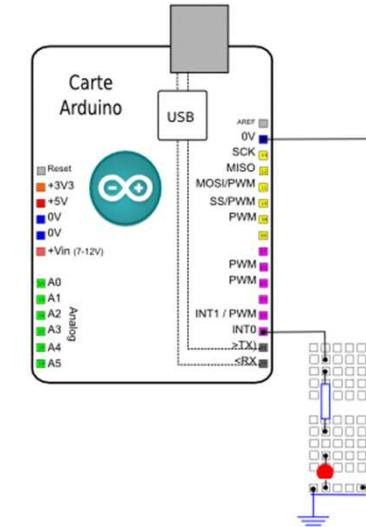
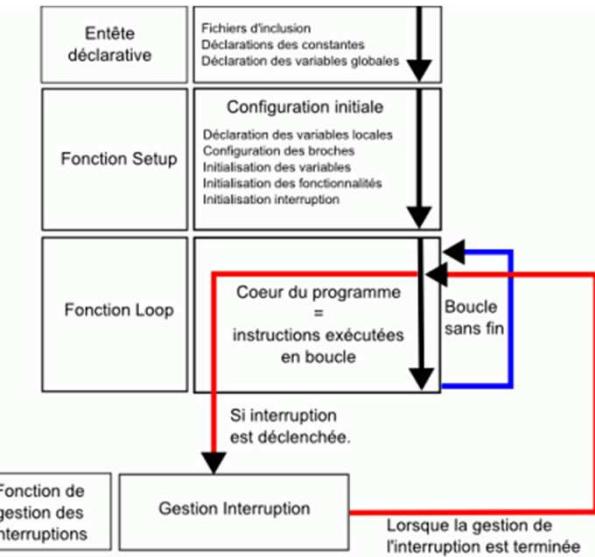
// --- Programme Arduino --- par X. HINAULT - Le 24/02/2010 www.mon-club-elec.fr

```
#include <MsTimer2.h> // inclusion de la librairie Timer2
const int LED=2; //declaration constante de broche

void setup() { // debut de la fonction setup()
pinMode(LED, OUTPUT); //met la broche en sortie
// initialisation interruption Timer 2
MsTimer2::set(1000, InterruptTimer2); // periode 1000ms
MsTimer2::start(); // active Timer 2
} // fin de la fonction setup()

void loop(){ // debut de la fonction loop()
}

***** Autres Fonctions du programme *****
void InterruptTimer2() { // beginning of the function interruption Timer2
digitalWrite(LED, HIGH);
delayMicroseconds(10000); // delayMicroseconds doesn't block the interrupts
digitalWrite(LED, LOW);
}
```



Exemple: <http://blog.blinkenlight.net/experiments/removing-flicker/heartbeat/>

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoInitiationInterruptionsTemporisationTimer2UneSeconde](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoInitiationInterruptionsTemporisationTimer2UneSeconde)

## III – 6 INTERRUPTS

### III – 6 - 2 TIMERS

# TIMERS

As a programmer Arduino, you used for timers and disruptions without knowing, because all the hardware low-level is hidden by the Arduino IDE.

Many functions use the timers: `delay()`, `millis()` `analogWrite()` for PWM, `tone()`.

Even the Servo library uses timers and interrupts.

**WHAT DOES A TIMER?** A timer, or to be more precise a timer/counter, is a piece of hardware of the ATmega that behaves like a clock in order to:

-measure temporal events, to count, to repeat, to compare.

The ATmega 328 of the Arduino UNO has three timers, called `timer0`, `timer1` and `timer2` while the ATmega1280 and ATMEGA2560 of the Arduino Mega series has 6 (2 \* 8bits and 4 \* 16 bit):

-Timer0: (8-bit timer) used by `delay()`, `millis()` and `micros()`. If you change this timer() records that might influence these functions => ATTENTION

-Timer1: (16-bit timer) used by the bookstore Servo on the Arduino Uno (`timer5` on the Arduino Mega).

-Timer2: (8-bit timer) as the timer0. Used by `tone()` in the Arduino Uno.

-Timer3, Timer4 Timer5: (16 bit timers) only available on the Arduino Mega.

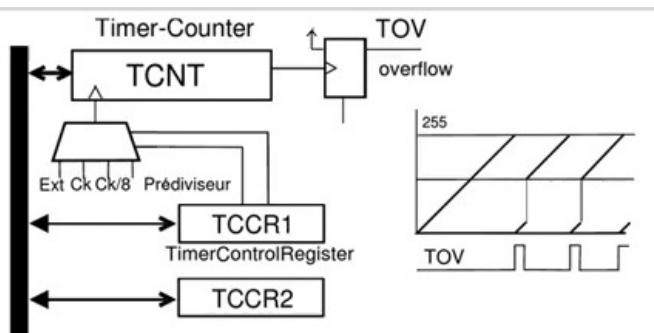
*Example: The `tone()` function uses the timer2-online you can not use the PWM on pins 3 and 11*

*The big difference between 8 and 16-bit timers is of course the clock resolution: 256 and 65536 values respectively for the 8 and 16-bit*

All timers are dependent on the system clock of your Arduino (e.g. 16 MHz for the UNO) but some can operate at different frequencies: example the Arduino Pro at only 8 MHz  
=> so be careful when you write your own functions of Timers.

# TIMERS

Timers are programmed by special registers, use the prescalers and has several modes of operation that we see now:

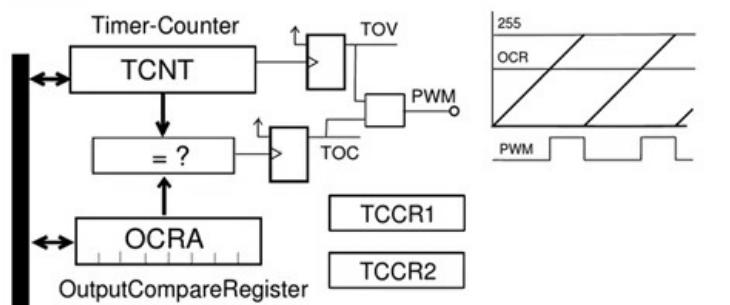


- 1 - Frequency of the timer-counter set by prescaler
- 2 - initialize registers,
- 3 - wait for the activation of TOV (Timer Overflow interrupt flag) (when the counter gets to the supplied value)
- 4 - a decision: e.g. you can't start from 0, but another value => less time to reach 255.
- 5 - Return to zero TOV
- 6 - reset TCNT to the desired time  
=> measure times

Intermediate comparison with one or two values gives a lot of flexibility

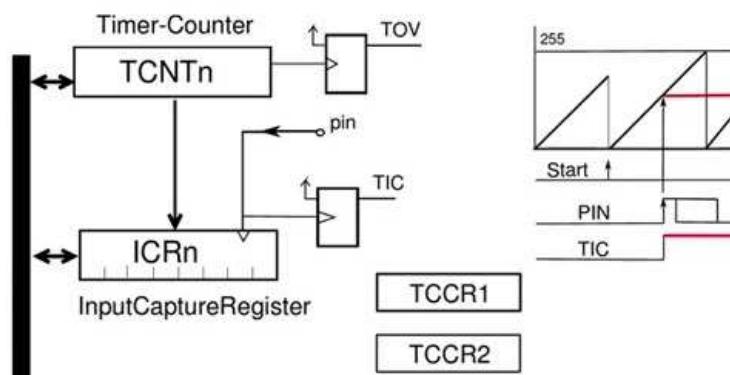
⇒ PWM

In the logical block: we start at the activation of TOC (Timer Overflow Interrupt) and go to the activation of TOV



Activation of the pin stores the State of the TCNTn counter in the ICRn registry => we'll know when we had a transition on the pin

When the transfer is done => activate TIC flag  
=> hardware interrupt flag



# TIMERS REGISTERS

**TIMERS registers: you can change the behavior of timers via timers registers. Timers the most important registers are the following:**

- **TCCRxy - Counter/Timer Control Register.**  
where you can set the prescalers and other things...

**-TCNTx - Timer/Counter Register.**

The value of the real clock is stored here.

**- OCRx - Output Compare Register**

**- ICRx - Input Capture Register (only 16bit timers).**

**- TIMSKx - Timer/Counter Interrupt Mask Register.**  
To toggle the clock interrupts.

**- TIFRx - Timer/Counter Interrupt Flag Register.**

Indicates an interruption of clock on hold.

**Remarks:**

**-the x suffix represents the timer number (0... 2)**

**-the y suffix represents the letter of the output (A, B, C):**

**e.g. TIMSK1 (mask of the timer1 interrupt) or OCR2A (A register of OCR from timer 2).**

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W

18.11.3 TCNT2 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0
(0xB2)	TCNT2[7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

16.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0
(0x89)	OCR1A[15:8]				OCR1A[7:0]			
(0x88)	OCR1A[15:8]				OCR1A[7:0]			

16.11.7 ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0
(0x87)	ICR1[15:8]				ICR1[7:0]			
(0x86)	ICR1[15:8]				ICR1[7:0]			

15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0
Read/Write	R	R	R	R	R	R/W	R/W	R/W

16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W

18.11.7 TIFR2 – Timer/Counter2 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
0x17 (0x37)	–	–	–	–	–	OCF2B	OCF2A	TOV2
Read/Write	R	R	R	R	R	R/W	R/W	R/W

**Ref: Atmel 8-bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash: pages 132, 134, 160...**

# FREQUENCIES AND MODES OF TIMERS

Selection of the frequency of the timer and the clock cycle:

$$\frac{f_{clk\_I/O}}{N.256} \quad \frac{f_{clk\_I/O}}{N.65536}$$

Different clock sources can be selected independently for each timer

Example : trigger value if you want to change the State of a pin at a frequency of  $f = 2$  Hz with the timer1 (16-bits), a prescaler of 256 and 16 MHz system clock frequency:

**Counter value = (16MHz/(256\*65536))/2Hz=31250**

If the result of the timer/counter compared to its maximum is OK (Yes  $31250 < 65536 == ""$ ) => can use the timer1 with 8bits timer:  $31250 > 256$

⇒ could not

If failure in one of the two cases, solution: choose a larger prescaler (=> slow down the clock).

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Table 16-4. Waveform Generation Mode Bit Description<sup>(1)</sup>

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX



# INTERRUPTS

To an interrupt pending is able to call ISR, be that:

-the interrupts are enabled. They are enabled/disabled by the `Interrupt()` / `noInterrupts()` functions. By default, they are enabled in the Arduino firmware.

Attention, the `attachInterrupt()` and `detachInterrupt()` functions are used for external interruptions on the pins (i.e. hardware INTERRUPTS).

-The mask of interrupts to be activated. It is enabled/disabled in defining/clear the bits of the register of the interrupt mask (`TIMSKx`).

*In practice: when an interruption occurs, a 'flag' is set in the interrupt register (TIFRx). This interruption is then automatically cleared when entering the ISR or even clearing manually the bit in the registry that contains the flag.*

Examples of interrupts:

1 - Timer Overflow: fires when that timer reaches its value limited. When this happens, the 'Timer Overflow bit' `TOVx` is put into registry "timer interrupt flag register" `TIFRx`. When the "timer overflow interrupt enable bit" `TOIEx` is set in the register "interrupt mask register" `TIMSKx`, the "timer overflow interrupt service routine" ISR (`TIMERx_OVF_vect`) is called.

2 - Output Compare Match: when an event Output Compare Match occurs, an `OCFxy` flag is set in the 'interrupt flag register' register `TIFRx`. When the "output compare interrupt enable bit" `OCIExy` is put into registry "interrupt mask register `TIMSKx`", the "output compare match interrupt service" ISR (`TIMERx_COMPy_vect`) is called.

3 - PWM: using records `OCRxA`, `OCRxB`.....

# EXAMPLE OF SOFTWARE INTERRUPT

## BLINKING A LED WITH TIMER/COUNTER COMPARE MATCH INTERRUPT:

This example uses the timer1 (16 - bit) in CTC mode and "compare match interrupt" to cause a LED to blink. The timer is set to a frequency of 2 Hz. The LED is now activated in the interrupt service routine.

Data: clock frequency system = 16MHz, Prescaler at 256

Same example but with TIMER OVERFLOW:

=> Counter =  $65536 - (16\text{MHz}/(256*65536))/2\text{Hz} = 34286$

```
#define ledPin 13

void setup()
{
    pinMode(ledPin, OUTPUT);

    // initialize timer1
    noInterrupts();           // disable all interrupts
    TCCR1A = 0;
    TCCR1B = 0;

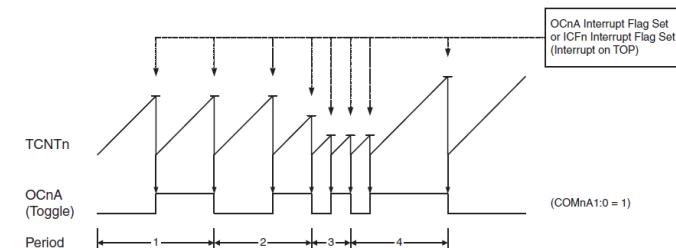
    TCNT1 = 34286;           // preload timer 65536-16MHz/256/2Hz
    TCCR1B |= (1 << CS12);  // 256 prescaler
    TIMSK1 |= (1 << TOIE1); // enable timer overflow interrupt
    interrupts();             // enable all interrupts
}

ISR(TIMER1_OVF_vect)        // interrupt service routine that wraps
                            // a user defined function supplied by attachInterrupt
{
    TCNT1 = 34286;           // preload timer
    digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}

void loop()
{
    // your program here...
}
```

<http://letsmakerobots.com/node/28278>

Figure 16-6. CTC Mode, Timing Diagram



=> Counter =  $(16\text{MHz}/(256*65536))/2\text{Hz} = 31250$

```
#define ledPin 13

void setup()
{
    pinMode(ledPin, OUTPUT);

    // initialize timer1
    noInterrupts();           // disable all interrupts
    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1 = 0;

    OCR1A = 31250;           // compare match register 16MHz/256/2Hz
    TCCR1B |= (1 << WGM12); // CTC mode
    TCCR1B |= (1 << CS12);  // 256 prescaler
    TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
    interrupts();             // enable all interrupts
}

ISR(TIMER1_COMPA_vect)      // timer compare interrupt service routine
{
    digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // toggle LED pin
}

void loop()
{
    // your program here...
}
```

## TOWARDS REAL-TIME OS: DUIN-OS

With the craze for the Arduino, it was normal that a version of a RTOS appears: DuinOS

DuinOS is a system real time operating with a preemptive multitasking system for Arduino and Atmel AVR. It is developed by RobotGroup (from Argentina) and is based on FreeRTOS

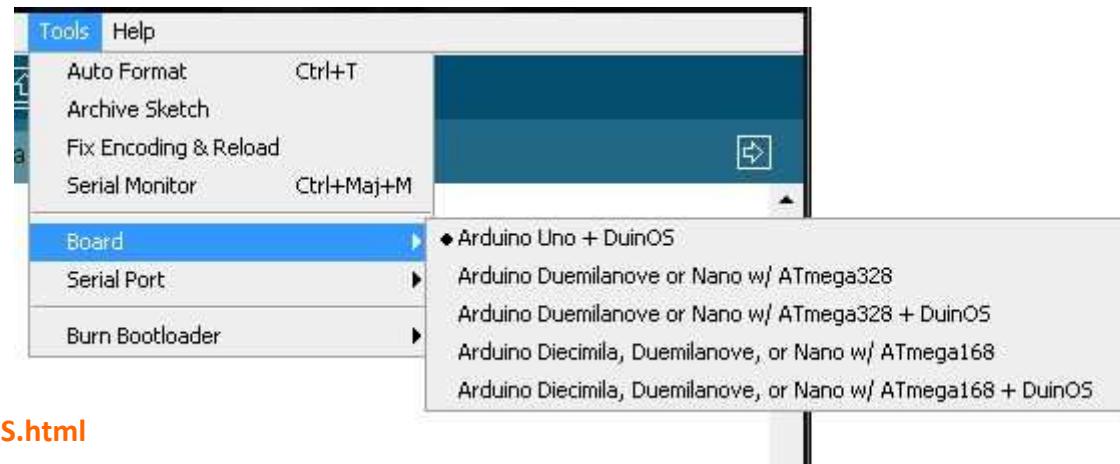
FreeRTOS is real time for embedded systems (on more than 30 microcontrollers) very popular operating system. It is distributed under the GPL with an optional exception.

The exception permits the code for users to remain owner of the closed source, while maintaining the kernel itself as open source, which facilitates the use of FreeRTOS in proprietary applications. [2]

I suggest you to install DuinOS and write your first program multi-tasking under this RTOS.

After the installation of the OS:

once you start the Arduino environment (with the arduino.exe file located at the root), you see no major change: but go to the "tools" menu and then the submenu "boards" and you will see the maps in version DuinOS:



<http://www.freertos.org/>

<https://github.com/carlosdelfino/DuinOS>

<http://www.pobot.org/Premiers-pas-avec-DuinOS.html>

<http://code.google.com/p/duinos/>

# FIRST EXAMPLE WITH DUIN-OS

Code to do simply flashing the two leds in parallel:

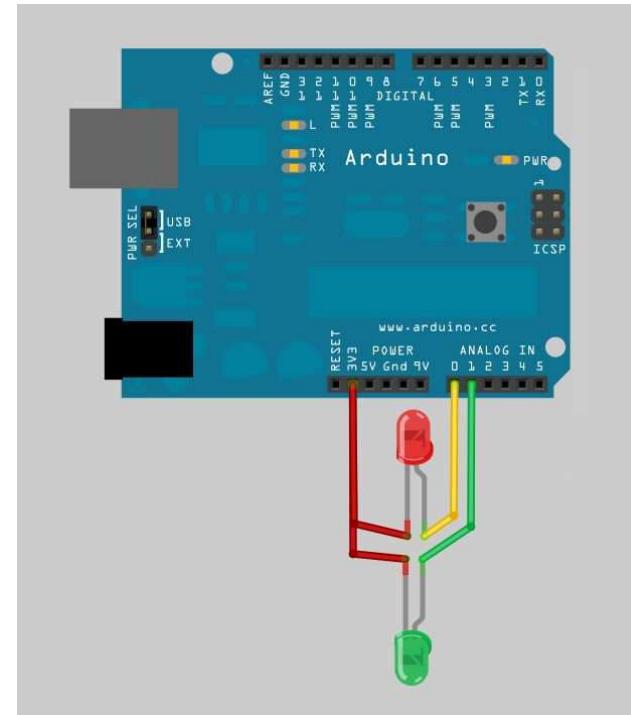
Schema of connection for quick test:

connect two leds directly on the Arduino between:

- 1 - the 3.3 volts (the lowest voltage available, and it's already too) and
- 2 - a controllable leg by a "digital" output

Reminder: "analog 0" and "analog 1" can be used simply everything or nothing, under the name of "digital 14" and "digital 15".

Can put one pull-up?

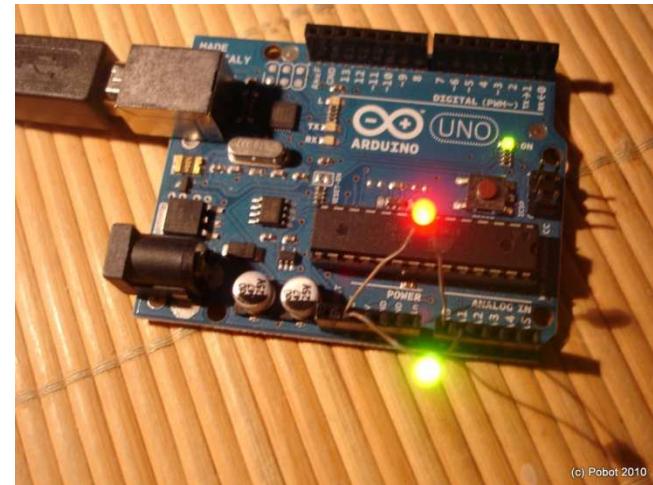


If you compile using a card that does not mention "DuinOS", you will see an error from the statement "taskLoop" since the compiler expects a function declaration, but there is no type before.  
error: expected constructor, destructor, or type conversion before '(...)

# FIRST EXAMPLE WITH DUN-OS

```
****  
 * Exemple simple d'utilisation de DuinOS  
 * basé sur l'exemple fourni dans la v0.2 "MoreComplexBlinking"  
 * Il s'agit tout simplement de faire clignoter deux leds en parallèle à des rythmes différents.  
 *  
 * Based on the original Blink code by David Cuartielles  
 *  
 */  
  
// on place les leds entre le 3,3 volts et les pattes analog 0 et 1 soit digital 14 et 15  
int ledPinRed = 14;  
int ledPinGreen = 15;  
  
// Déclaration des deux tâches parallèles  
  
taskLoop(redLED)  
{  
    digitalWrite(ledPinRed,HIGH);  
    delay(200);  
    digitalWrite(ledPinRed,LOW);  
    delay(200);  
}  
  
taskLoop(greenLED)  
{  
    digitalWrite(ledPinGreen, HIGH);  
    delay(500);  
    digitalWrite(ledPinGreen, LOW); // set the LED off  
    delay(500);  
}  
  
// Fonctions habituelles  
  
void setup()  
{  
    // Initialisation des deux sorties pour les leds  
    pinMode(ledPinRed, OUTPUT);  
    pinMode(ledPinGreen, OUTPUT);  
  
    // Création des deux tâches  
    createTaskLoop(redLED, NORMAL_PRIORITY);  
    createTaskLoop(greenLED, NORMAL_PRIORITY);  
}
```

```
void loop()  
{  
    // Deux tâches en parallèle pendant 2 secondes  
    resumeTask(greenLED);  
    resumeTask(redLED);  
    delay(2000);  
    // Puis seulement la verte pendant 2 secondes  
    suspendTask(redLED);  
    delay(2000);  
    // Puis seulement la rouge  
    resumeTask(redLED);  
    suspendTask(greenLED);  
    delay(2000);  
    // et on recommence (loop)  
}
```



(c) Pobot 2010

*Of course, the CPU can process only one order at a time, so the real time is obtained by putting an algorithm that distributes the time CPU turn (intelligently).*

*At least you know that you have an environment that works with DuinOS, and you can calmly begin to discover the multi-task real-time programming.*

## IV-1 POUR ALLER PLUS LOIN:

### LES PONTS-H

# COMMANDE DE MOTEURS DC

## COMMENT COMMANDER UN MOTEUR À COURANT CONTINU SACHANT QUE :

**Problème 1:** que l'on veut faire varier sa vitesse. Or, pour démarrer, un moteur à besoin d'une tension minimale et sa fréquence de rotation n'est pas réellement proportionnelle à la tension d'alimentation.

**Problème 2:** le moteur doit pouvoir tourner dans les deux sens et fonctionner dans les 4 quadrants, c'est-à-dire, soit en moteur (M), soit en génératrice (G) (freinage du moteur),

**Problème 3:** le courant absorbé est beaucoup plus important que ce que peut fournir un microcontrôleur.

Les solutions suivantes peuvent être envisagées (ce ne sont sûrement pas les seules)

### 1°) *Résolution du problème 1 :*

Il suffit de fournir au moteur une tension qui reste constante : la tension maximale et en même temps, cette tension ne sera appliquée que par très courtes périodes de temps => utilisation de la PWM  
En ajustant la durée de ces périodes, on arrive à faire varier la vitesse du moteur qui devient proportionnelle à la longueur des périodes de temps passées à la tension maximale par rapport au temps passé sans application de tension (tension nulle).

Problème, avec une PWM on ne peut contrôler un moteur DC que dans un seul sens .

Il serait possible d'utiliser un relai pour inverser la connexion (polarisation) du moteur mais cette option serait extrêmement gourmande en énergie.

### 2°) *Résolution du problème 2: => le pont H.*

[http://wiki.mdl29.net/doku.php?id=elec:moteur\\_a\\_courant\\_continu](http://wiki.mdl29.net/doku.php?id=elec:moteur_a_courant_continu)

# COMMANDÉ DE MOTEURS DC

## 2°) Résolution du problème 2 et 3 : PONT H

1 - Pour inverser le sens de rotation du moteur il suffit d'inverser la polarité de la tension à ses bornes. Le montage à utiliser est très simple : c'est un montage à 4 interrupteurs piloter 2 par 2 en alternance, d'où le nom pont en H.

Montés de telle façon, le courant peut passer soit dans un sens, soit dans l'autre dans la charge

**Fonctionnement :**

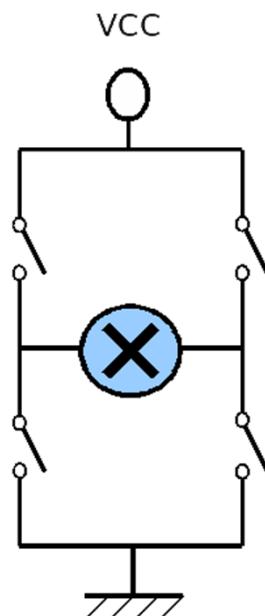
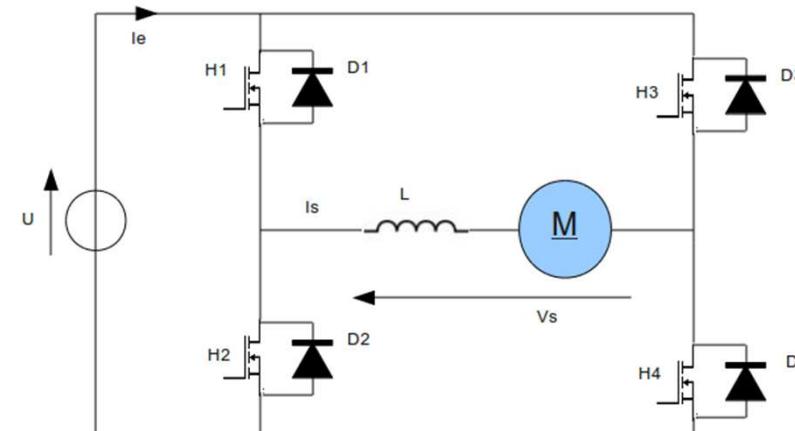
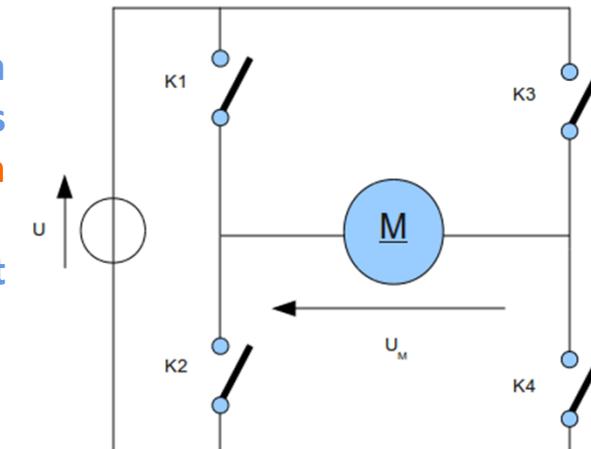
K1 et K4 fermés, K2 et K3 ouverts :  $U_M = U$

K1 et K4 ouverts, K2 et K3 fermés :  $U_M = -U$

2 - Les interrupteurs doivent pouvoir être commandés à l'ouverture et à la fermeture, et amplifier le courant fournit par le micro-contrôleur : on utilise des transistors en commutation , généralement des MOSFET.

Montage :

$H_1, H_2, H_3$  et  $H_4$  sont des interrupteurs pilotés à l'ouverture et à la fermeture (ce peut être des transistors ou des thyristors). L'inductance permet de lisser le courant.

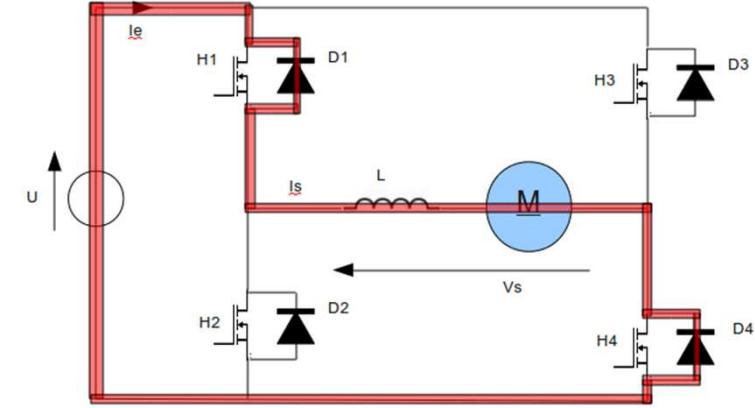
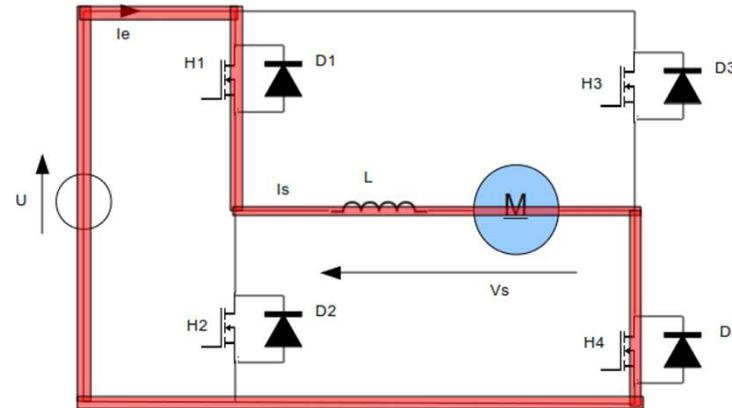


# COMMANDE DE MOTEURS DC

## Fonctionnement dans les 4 quadrants :

1)  $H_1$  et  $H_4$  fermés,  $H_2$  et  $H_3$  ouverts  $\Rightarrow V_s = U$  (le moteur tourne dans un sens)

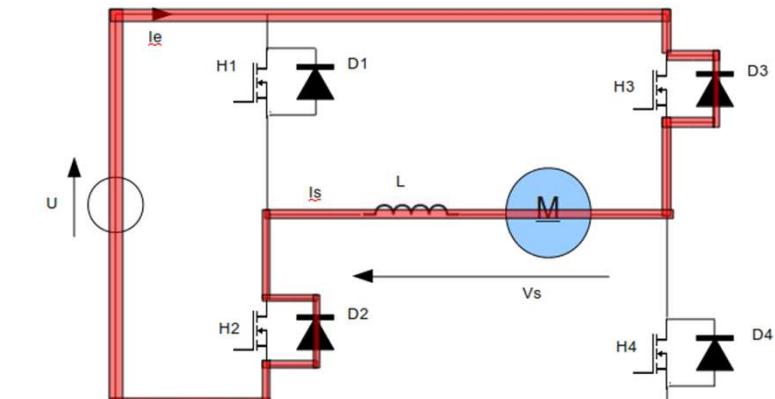
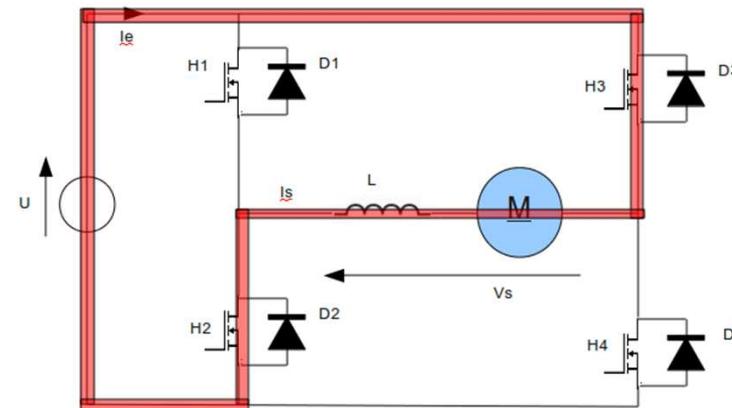
*Si  $I_s > 0$  alors le courant passe par  $H_1$  et  $H_4$  (quadrant Q<sub>1</sub>)      Si  $I_s < 0$  alors le courant passe par  $D_1$  et  $D_4$  : (quadrant Q<sub>2</sub>)*



2)  $H_1$  et  $H_4$  sont ouverts,  $H_2$  et  $H_3$  sont fermés  $\Rightarrow V_s = -U$  (le moteur tourne dans l'autre sens)

*Si  $I_s < 0$  alors le courant passe par  $H_2$  et  $H_3$  (quadrant Q<sub>3</sub>)*

*Si  $I_s > 0$  alors le courant passe par  $D_2$  et  $D_3$  (quadrant Q<sub>4</sub>)*



Les diodes sont appelées diodes de récupération, elles permettent la circulation du courant lorsque l'interrupteur est commandé et que le courant est dans le sens opposé à celui de l'interrupteur. Cette phase est appelée phase de récupération, elle correspond au freinage du moteur (qui fonctionne à ce moment précis en génératrice) appelé « freinage par récupération ».

# COMMANDÉ DE MOTEURS DC

Les L293NE et SN754410 sont des exemples basiques de pont-H:

1 - Ils disposent de deux ponts:

- l'un sur le côté gauche de la puce et
- l'autre sur la droite,

2 - Ils peuvent commander 2 moteurs.

3 - Ils peuvent piloter jusqu'à 1A de courant

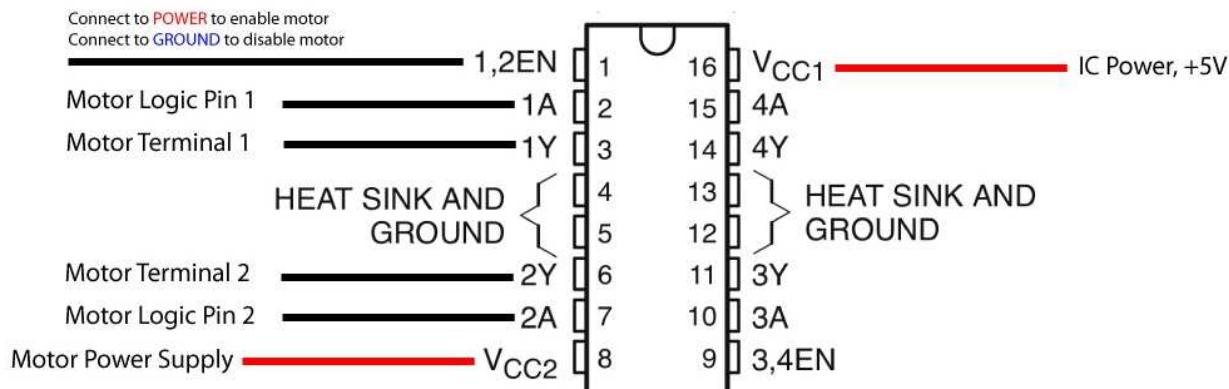
4 - et fonctionnent entre 4.5V et 36V.



EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

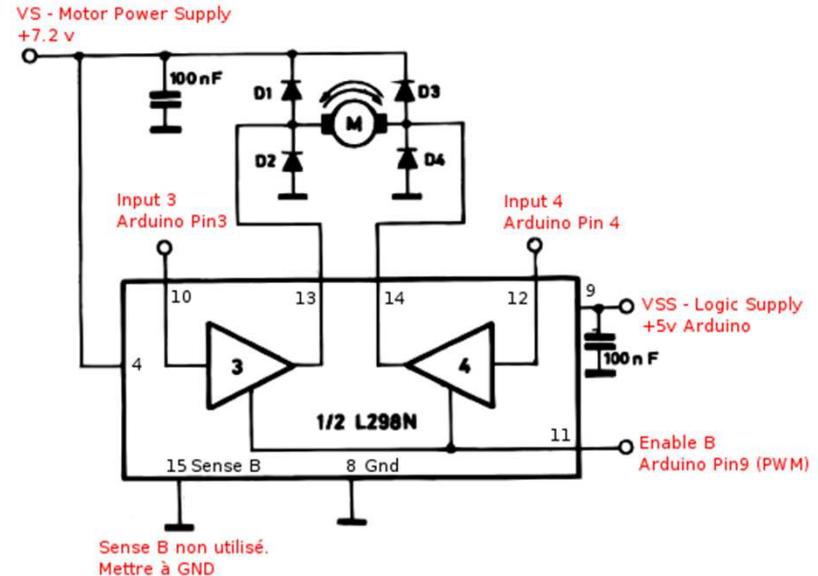
## L293NE or SN754410



# COMMANDE DE MOTEURS DC – QUELQUES RECOMMANDATIONS

1 - Nécessite absolument 4 diodes en roue libre pour protéger les sorties (Output) du pont-H.

Utiliser des diodes de type "fast" ([BYV27-100-TAP: Diode Ultrafast, 2A, 100V. Farnell #1469371](#)).



2 - MOSFET Vs Bipolaires => Anticiper la chute de tension

Les transistors bipolaires commutés en saturation présentent une légère chute de tension ( $V_{CE\_SAT}$ ). Et comme le courant qui actionne le moteur passe par deux transistors, la chute de tension intervient donc deux fois!

Exemple L298N:

$V_{CEsat}(L)$	Sink Saturation Voltage	$I_L = 1A$ (5) $I_L = 2A$ (5)	0.85 1.2 1.7	1.6 2.3	V V
$V_{CEsat}$	Total Drop	$I_L = 1A$ (5) $I_L = 2A$ (5)	1.80	3.2 4.9	V V

Chute de tension  $V_{CE\_SAT}$  totale (total drop) de 1.8V (typique), 3.2V à  $I_L=1A$ , 4.9V à  $I_L=2A$ .

*Si cela semble beaucoup, c'est aussi un avantage car cela permet d'utiliser un accu de 7.2V directement avec un moteur 5V.*

# COMMANDE DE MOTEURS DC

```
int switchPin = 2;      // switch input
int motor1Pin1 = 3;     // pin 2 on L293D
int motor1Pin2 = 4;     // pin 7 on L293D
int enablePin = 9;      // pin 1 on L293D

void setup() {
    // set the switch as an input:
    pinMode(switchPin, INPUT);

    // set all the other pins you're using as outputs:
    pinMode(motor1Pin1, OUTPUT);
    pinMode(motor1Pin2, OUTPUT);
    pinMode(enablePin, OUTPUT);

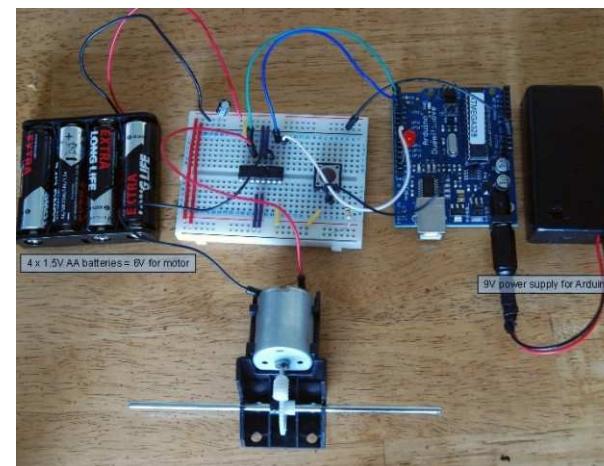
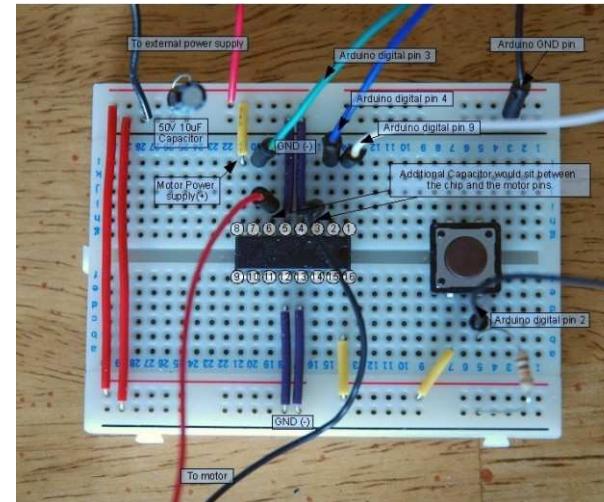
    // set enablePin high so that motor can turn on:
    digitalWrite(enablePin, HIGH);
}

void loop() {
    // if the switch is high, motor will turn on one direction:
    if (digitalRead(switchPin) == HIGH) {
        digitalWrite(motor1Pin1, LOW);    // set pin 2 on L293D low
        digitalWrite(motor1Pin2, HIGH);   // set pin 7 on L293D high
    }
    // if the switch is low, motor will turn in the opposite direction:
    else {
        digitalWrite(motor1Pin1, HIGH);  // set pin 2 on L293D high
        digitalWrite(motor1Pin2, LOW);   // set pin 7 on L293D low
    }
}
```

<http://luckylarry.co.uk/arduino-projects/control-a-dc-motor-with-arduino-and-l293d-chip/>

[http://wiki.mdl29.net/doku.php?id=elec:moteur\\_a\\_courant\\_continu](http://wiki.mdl29.net/doku.php?id=elec:moteur_a_courant_continu)  
<http://itp.nyu.edu/physcomp/Labs/DCMotorControl>

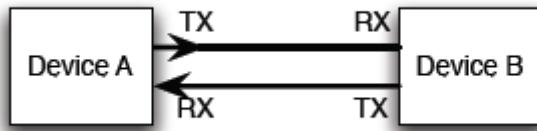
<http://arduino103.blogspot.com/2011/06/pont-h-transistor-pour-controler-un.html>



# III-7 SERIAL COMMUNICATION: ASYNCHRONOUS & SYNCHRONOUS

# SERIAL COMMUNICATIONS

## Asynchronous communication

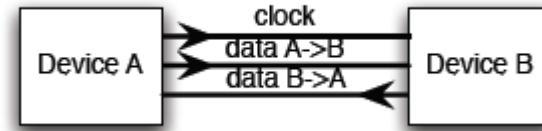


Asynchronous, because no clock.  
The data is represented with HIGH/LOW at certain times

Wires separated for  
the transmission and reception

=> both devices need to  
communicate at the same rate

## Synchronous Communication



Synchronous, because clock.  
The data is represented with HIGH/LOW  
when the clock changes

A single wire of clock and data for each of  
the directions

=> no need to have two devices  
communicating at the same rate, but one of  
the two is the MASTER

What is the best?

It depends on your application:

- the asynchronous are good when there are only two devices and they are pre-configured to talk at the same speed.
- Synchronous are usually better in speed

## ASYNCHRONOUS SERIAL COMMUNICATION : UART...

ASYNCHRONOUS SERIAL  
COMMUNICATION : UART...

# SERIAL INTERFACE – UART

The RS-232 is a bus of devices designed in 1962.

A RS-232 connection works:

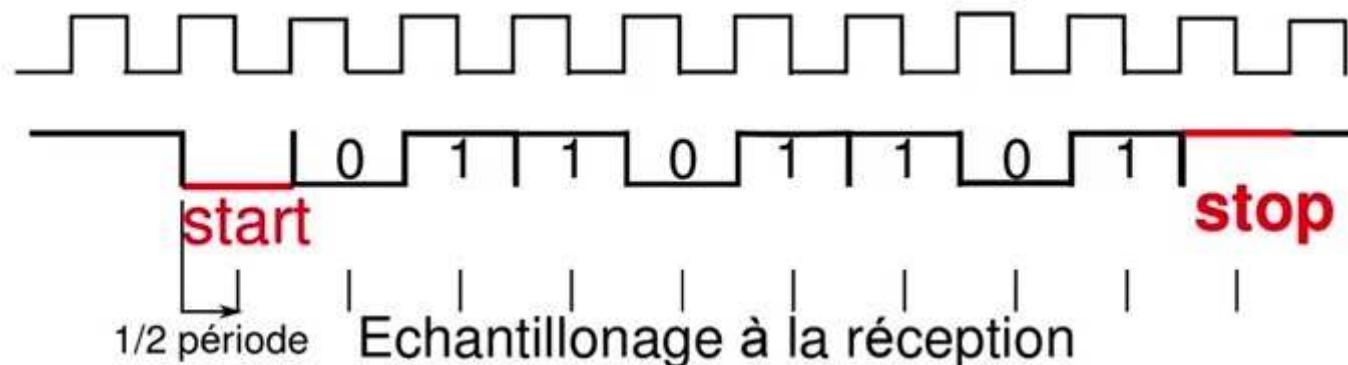
- peer-to-peer; There is neither slave nor master defined at the level of the Protocol
- full duplex; the two peers can communicate at the same time.

Unlike SPI and I2C links, it is possible to use it on several tens of meters, at the expense of the flow (2400 to 115200 b/s and even more under certain conditions).

Main principle :

It transmits on line 8 bits:

- 1 - preceded by a start bit (set to 0 over a period) and
- 2 - followed by a bit of stop 'silence' (set to 1 on a period)



Typical values: 8 bits, 1 stop bit, no parity, 9600 bps

Possible values: 300,... 9600, 19200,...115200 bits/s

### SYNCHRONOUS SERIAL COMMUNICATION : I<sup>2</sup>C, SPI, ONE WIRE, ...

## SYNCHRONOUS SERIAL COMMUNICATION : I<sup>2</sup>C

# I2C BUS

The I2C bus is characterized by a link in series produced using 2-wire mode. It's the Philips company which created the concept in the early 1980s. Its success is linked to its simplicity.

Here is the type of an I2C-bus architecture, we notice that several components (128 Max (because address coding 7bits)) are "grafted" on the same bus, the data passes through the lines:

-SDA: data signal, generated by the master or the slave.

-SCL: clock signal generated by the master.

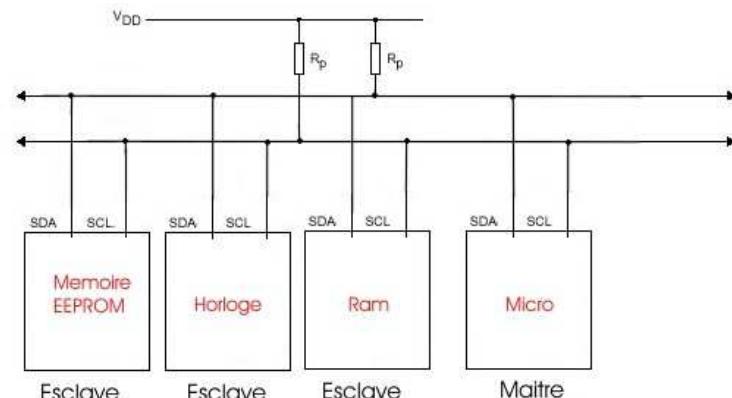


Schéma d'interface matérielle des circuits compatibles I<sup>2</sup>C

## Communication on the bus is orchestrated in the following way:

- the master sends on the bus address of the component with which it wishes to communicate,
- each of the slaves having a fixed address, the slave who recognizes responds in turn by a signal of confirmation,
- then the master continues the communication... procedure: (read/write)
- in all cases, transactions will be confirmed by an ACK (acknowledge)

## Electrical characteristics:

**Bus voltage: 5V DC**

**Maximum operating frequency: 400kHz (variable according to the components connected to the bus, see datasheet)**

**Logical States: -HIGH: 3 to 5 Volts, -LOW: 0 to 1.5 Volts**

**Maximum eligible on the bus: 400 pf**

**Rise of signals time: less than 1 μs**

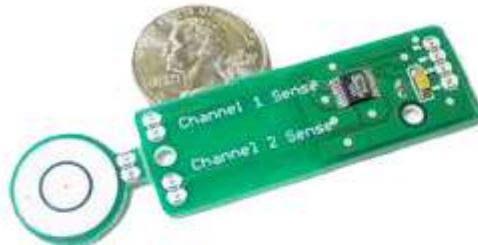
**Stability time for consideration of a signal: greater than 5 μs (general rule)**

**Pull-up Resistance : to calculate the impedance of the Bus (practical values observed from 1.5 K to 3.5 K)**

**Distance from community communication: some tens of centimeters, and several meters with a buffer.**

# MANY I2C COMPONENTS

Examples of components using the I2C: memories, converters, analog/digital and digital / analog clocks real-time, integrated radio or TV receivers frequency synthesizers...



touch sensor



non-volatile  
memory



compass

*Boussole digitale  
indiquant le nord*



fm transmitter



LCD display

And many others  
(gyros, keyboards, motors,...)

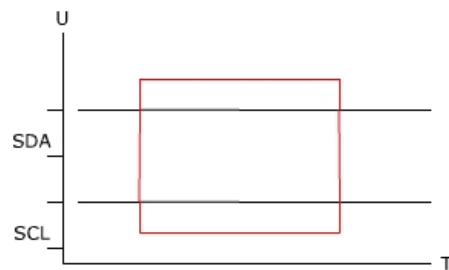


temperature &  
humidity sensor

# THE BASICS OF THE PROTOCOL

## 1 - The REST CONDITION :

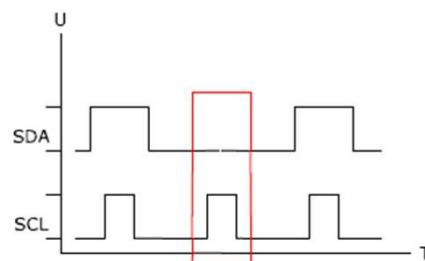
Condition in which no components communicate, bus lines are in the HIGH state:



## 3 – Acknowledgment :

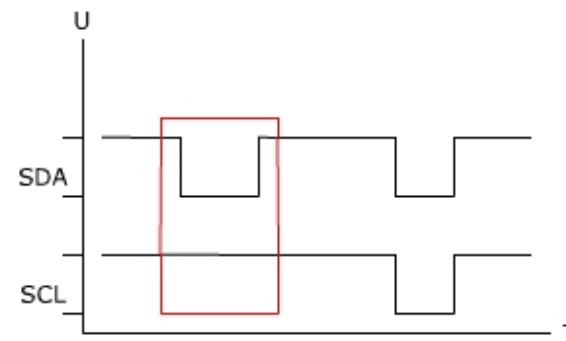
once the data is transmitted, an acquittal is operated by the person who receives the data.  
This procedure is performed in the end frame transmission, namely the ninth impulse.

Keeping receiver SDA line low during the rising of clock:



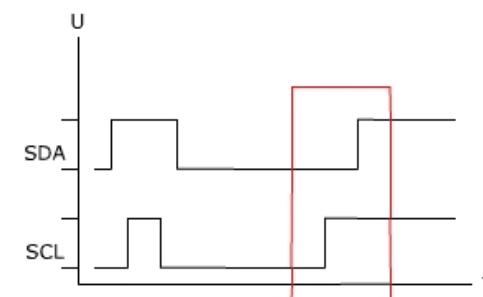
## 2 - The starting CONDITION:

the master force line SDA low to "wake up" the slaves when SCL is high:



## 4 - The stop CONDITION:

SDA and SCL lines are low, and then when SCL goes at the upper level, the master releases the SDA line at the high level. Now the bus is regarded as available:



# ARCHITECTURE OF A FRAME

## THE FRAME:

- 1 - must wake up the bus with a start condition,
- 2 - then come to interpolate data frames. The significant bit being sent first. The slave who received information perform the master to let her know that it successfully received the frame of data, otherwise, is the master to start the transactions since the start.
- 3 - to finish the Stop signal will end any communication.

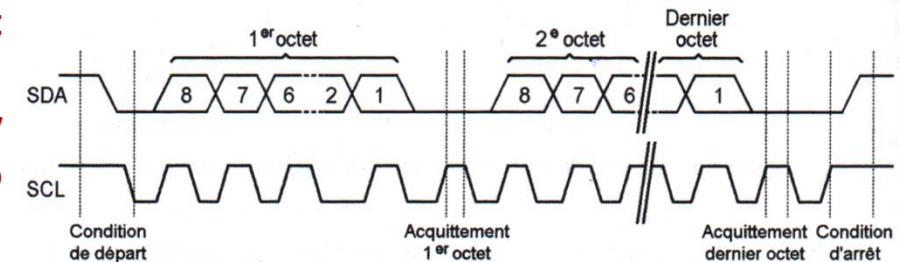
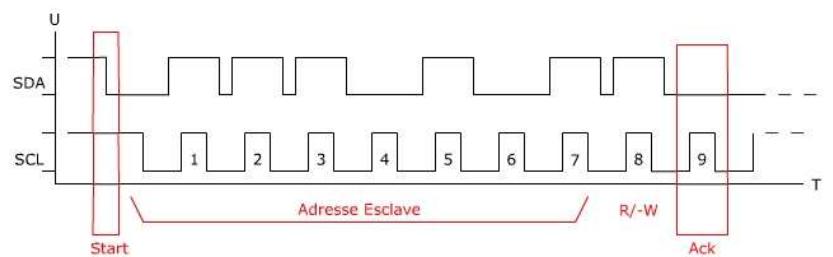


Figure 6.8 – Chronogramme d'un échange complet sur le bus I2C.

## THE ADDRESSING: to communicate with the ≠ components connected to the bus:

- 1 - after you give a start condition, the master shows with what slave it wishes to connect, for this he will send on the address of the slave composed of 7-bit,
- 2 – then a last bit showing the transfer direction: reading (1: slave-> master) or writing (0: master-> slave).



The slave recognize his address will validate with an acknowledgment...

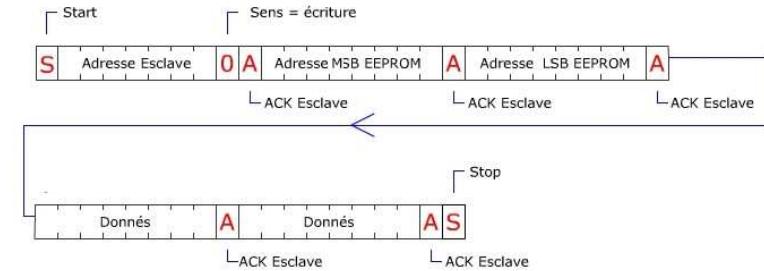
Wire function library : <http://www.arduino.cc/en/Reference/Wire>

# COMMUNICATION BY BYTE

## COMMUNICATION by byte: a complete Exchange on the bus between master and slave

### 1 - Sequence of writing data to a memory EEPROM I2C:

- Condition departure
- Emission of the slave address, i.e. the EEPROM here
- Acknowledgment of the slave
- Emission by the master of the memory address in which he wishes to enter the future data, subsequent acknowledgment of the slave
- Transmission of the data to write, acknowledgment on each receipt package
- Stop condition



We notice that the master, after emission of the address of data storage, didn't need to interrupt the communication at each byte and write again to the pointing address. I2C components are generally to 'auto-loading', that is to say that their address pointer automatically increments and this for the 8-byte value.

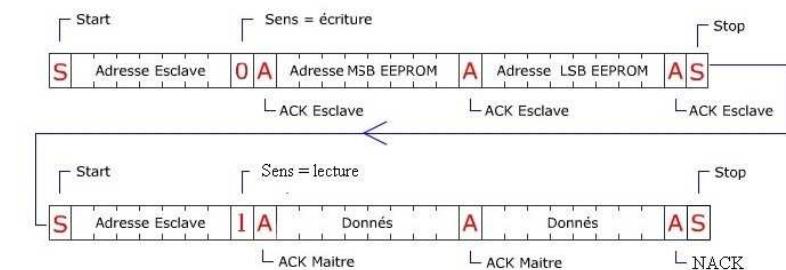
### 2 - Sequence of reading data in a memory EEPROM I2C:

#### 1 - Start Condition

- Emission of the slave address, i.e. the EEPROM, with a write request. What for? just to point the internal memory register on the chosen address, otherwise, reading would be the value contained in this pointer, which can give you in return a no good values.
- Acknowledgment of the slave and stop condition (Restart is possible)

#### 2 - Start condition

- Show the slave address, ie. the EEPROM, with a read request this time. The component confirms the order and the data is sent. Playback can be done by block, in this case, confirm each data read by an ACK,
- then before the stop condition, sent a NACK.
- Stop condition



## REMARKS: SLAVE ADDRESSES

On the I<sup>2</sup>C bus, an address is assigned at each component, but what do if there had several times the same TYPES of components present on the BUS?

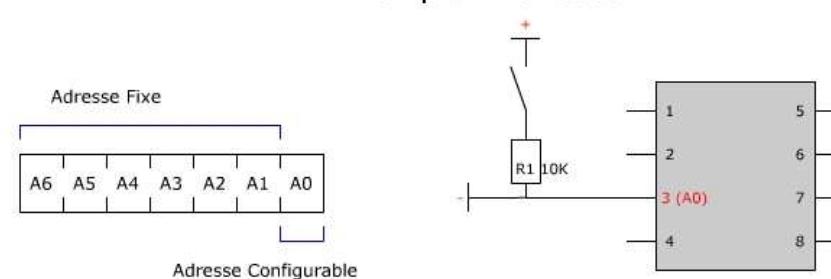
To generalize, every family circuit consists of an "Silicon" address, permanent and unchangeable, then other part entirely configurable totally accessible by external way, via the terminals of the integrated circuit:

In the example below, the PCF8583 (clock-calendar + RAM) address consists of 6 fixed bits and 1 editable, ie. the ability to connect 2 same circuits on the bus.

Table 8. SDA Write Mode Bit Format

S	0	1	0	1	1	0	AD0	0	A
Slave Address Byte									

Exemple : PCF8583



I<sup>2</sup>C popular component and the composition of their fixed addresses:

Type	Description	Constant adress
PCD 3311	Générateur / Décodeur DTMF	010010x
PCF 8570	RAM Statique 2 Ko (256 x 8)	1010xxx
PCF 8573	Horloge/Calendrier temps réel	11010xx
PCF 8574	Extension de port parallèle 8 Bits	0100xxx
PCF 8552	EEprom 2 Ko (256 x 8)	1010xxx
PCF 8583	Horloge/Calendrier temps réel + 240 Octets de RAM	101000x
PCF 8591	Convertisseur A/N N/A 8 Bits	1001xxx
SAA 1064	Drivers d'affichage à leds 4 Digits	01110xx
M24C256	EEprom 128 Ko (32k x 8)	1010xxx

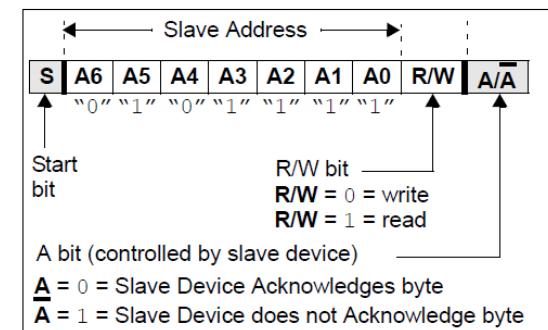


FIGURE 5-9: Slave Address Bits in the I<sup>2</sup>C Control Byte.

We notice that some circuits have the same address, but this is only of components of similar functions as the memories...

## PROGRAM EXAMPLE

Join the bus I2C  
(as master)



```
#include <Wire.h>

void setup() {
    Wire.begin();          // join i2c bus (address optional for master)
    Serial.begin(9600);    // start serial for output
}

void loop() {
    Wire.requestFrom(2, 6); // request 6 bytes from slave device #2

    while(Wire.available()) { // slave may send less than requested
        char c = Wire.receive(); // receive a byte as character
        Serial.print(c);       // print the character
    }

    delay(500);
}
```

Request the data  
from the device



Retrieve the data



# EXEMPLE D'UTILISATION DU BUS I2C

interface Control of I2C digital potentiometer (AD5171) :

*Reminder: I2C Pin on the Arduino : SDA: Analog In 4, SCK: Analog In 5*

Table 8. SDA Write Mode Bit Format

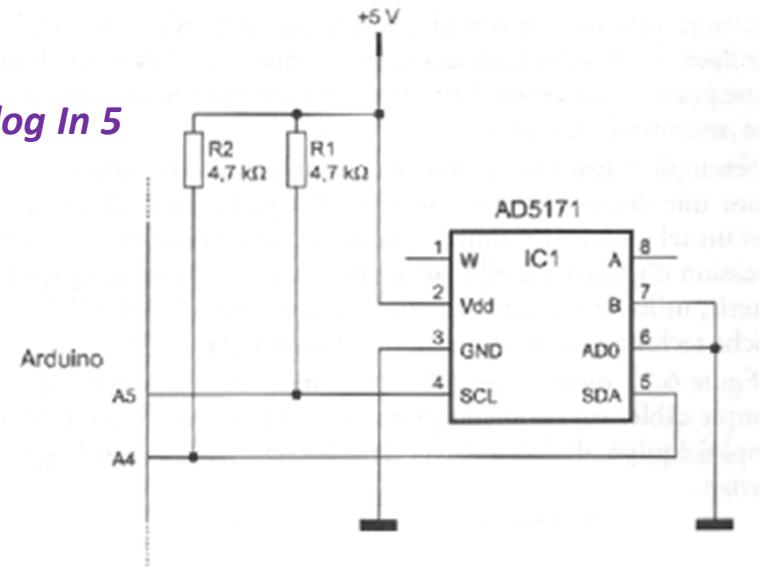
S	0	1	0	1	1	0	AD0	0	A
Slave Address Byte									
	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$		
	0	1	0	1	1	0	0		

Decimal adress 44 ( $2^5 + 2^3 + 2^1 = 44$ ) => 0x2C (Hexadecimal)

```
#include <Wire.h>
byte position = 0;
```

```
void setup ()
{
    Wire.begin(); // join i2c bus, Arduino en Maître
}
void loop ()
{
    Wire.beginTransmission (0x2C); // L'AD5171 est à l'adresse 0x2C
    Wire.send (0x40); // Envoi de l'instruction
    Wire.send (position); // Envoi de la position
    Wire.endTransmission (); // Fin de l'échange
    Position++;
    if (position == 64) //si le maximum (64 positions pour AD5171) est atteint
    {
        Position=0;
    }
    Delays (500);
}

I2C Device Address Bit. Allows a maximum of two AD5171s to be addressed.
```



FUNCTIONAL BLOCK DIAGRAM

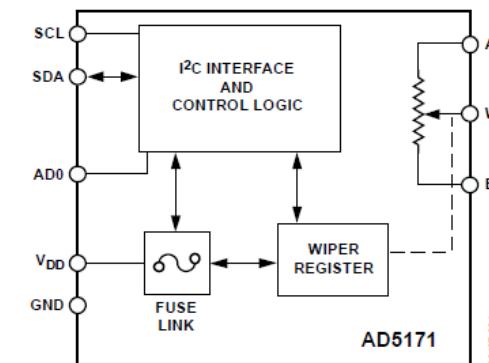


Figure 1.

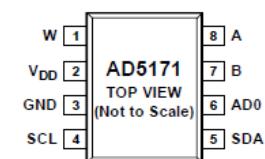


Figure 2. Pin Configuration

# I<sup>2</sup>C APPLICATION EXAMPLE : Real Time Clock: DS1307

The DS1307 which is a I<sup>2</sup>C controlled real-time clock.

I<sup>2</sup>C adress (7bits) of the DS1307 : 1101000

Figure 4. Data Write—Slave Receiver Mode

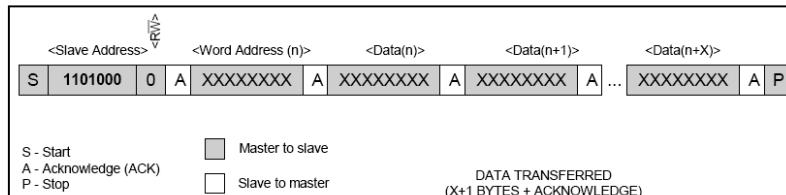
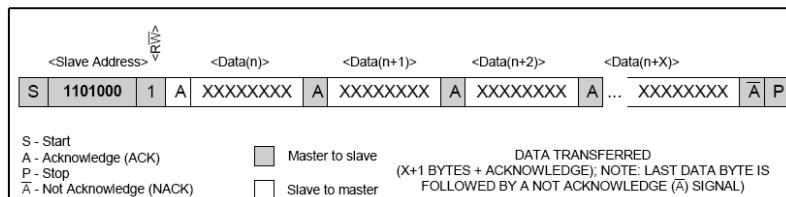


Figure 5. Data Read—Slave Transmitter Mode



The addresses of registers:

register RTC 00h to 07h

registry RAM 08h to 3Fh

Table 2. Timekeeper Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds			Seconds	00-59	
01H	0	10 Minutes			Minutes			Minutes	00-59	
02H	0	12	10	Hour	Hours			Hours	1-12 +AM/PM 00-23	
		24	PM/ AM							
03H	0	0	0	0	0	DAY	Day		01-07	
04H	0	0	10 Date		Date			Date	01-31	
05H	0	0	0	10	Month		Month	Month	01-12	
06H	10 Year			Year			Year	Year	00-99	
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH								RAM 56 x 8	00H-FFH	

0 = Always reads back as 0.

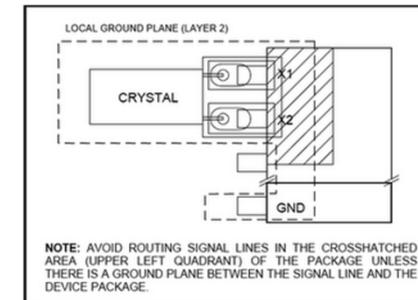
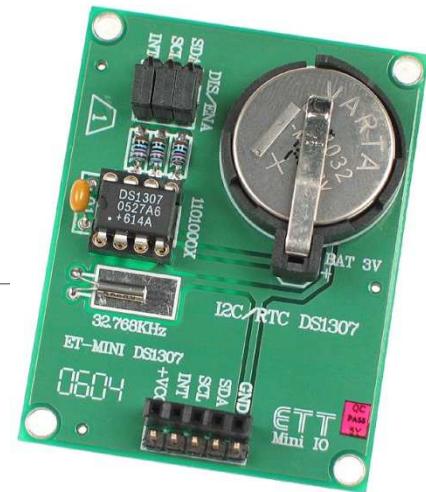
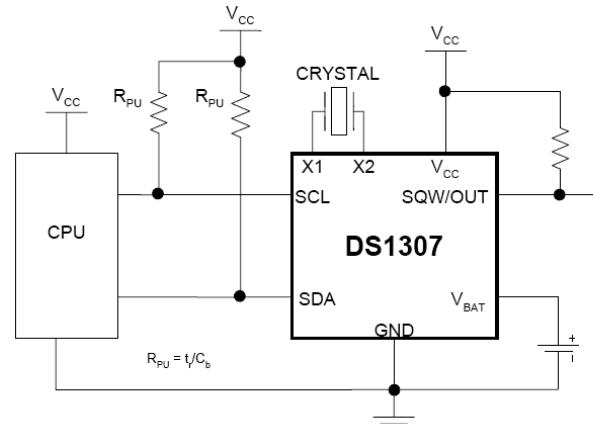
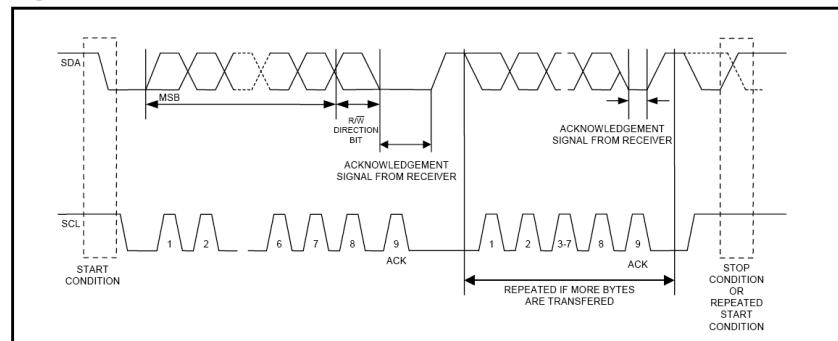


Figure 3. Data Transfer on I<sup>2</sup>C Serial Bus



# I<sup>2</sup>C APPLICATION EXAMPLE : Real Time Clock: DS1307

Each bit in the different registers can only be 0 or 1.

How to represent the contents of the registry in hexadecimal?

Answer: You must first determine the binary representation and then convert it to hexadecimal.

Take 3th register 02H (HOURS) and an hour example to represent 12:34 pm :

Reading from left to right :

- Bit 7 is unused => 0.
- Bit 6 determine the hour display format (12- ou 24-hour): let's choose 0 for example for the 12-hour
- Bit 5 (when bit 6 is 0) represent indicator AM/PM: let's choose 1 for example for PM.
- Bit 4 represents the most left bit of the hour, i.e. the 1 in 12:34 pm: let's choose 1 for example
- Bits 3 to 0 represent binary conversion of 2 from 12 i.e. is 0010.

Hence, the 3rd register allows to represent 12 pm by writing the binary 00110010

Converted into hexadecimal i.e. 0x32, sent to register 02H.

To code minutes, we must proceed the same way with 2<sup>nd</sup> register 01H!!!

Table 2. Timekeeper Registers

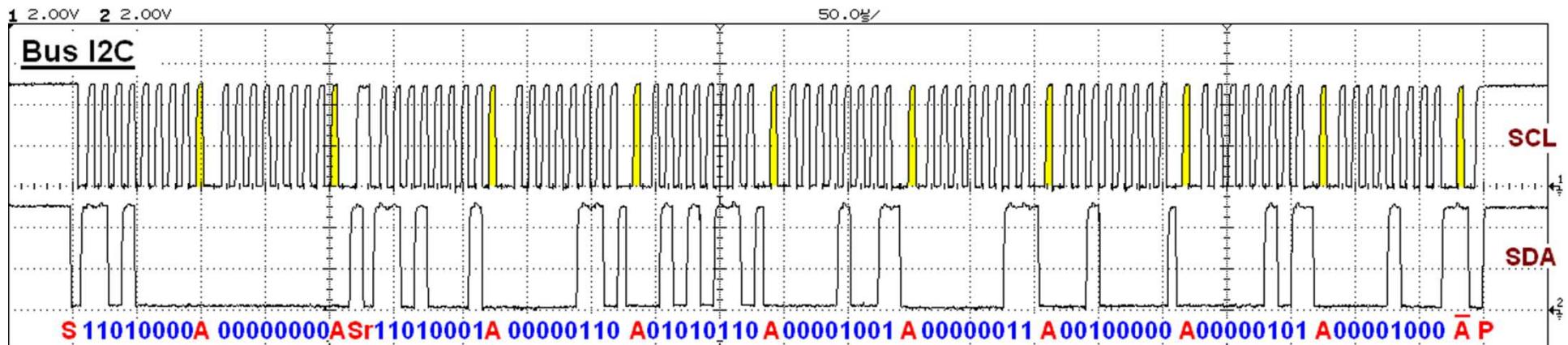
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00H	CH		10 Seconds			Seconds			Seconds	00–59
01H	0		10 Minutes			Minutes			Minutes	00–59
02H	0	12	10 Hour	10 Hour		Hours			Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03H	0	0	0	0	0	DAY			Day	01–07
04H	0	0	10 Date			Date			Date	01–31
05H	0	0	0	10 Month		Month			Month	01–12
06H		10 Year				Year			Year	00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH									RAM 56 x 8	00H–FFH

0 = Always reads back as 0.

# I<sup>2</sup>C APPLICATION EXAMPLE : Real Time Clock: DS1307

Example of use: plug the SQW/OUT output signal (PIN 7) at 1 Hz frequency to one of the interrupt pins of the Arduino:

- that causes an interruption every 1000,000 ms.
- the Arduino reads the time and date in the DS1307, communication is via the I<sup>2</sup>C bus:



In this chronogram, it is 9 hours 56 minutes 6 seconds, on Tuesday 20 May 2008.

<http://www.glacialwanderer.com/hobbyrobotics/?p=12>

[http://combustory.com/wiki/index.php/RTC1307 - Real\\_Time\\_Clock](http://combustory.com/wiki/index.php/RTC1307 - Real_Time_Clock)

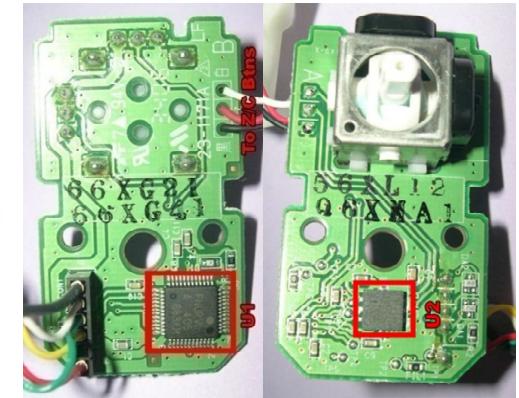
<http://tronixstuff.wordpress.com/2010/05/20/getting-started-with-arduino-%E2%80%93-chapter-seven/>

<http://tronixstuff.wordpress.com/2010/10/07/add-a-real-time-clock-to-the-freetronics-twentyten/>

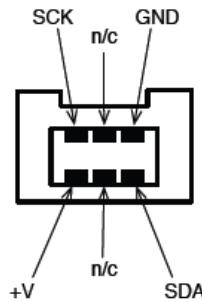
# Nintendo Wii Nunchuck

The Wii Nunchuk has:

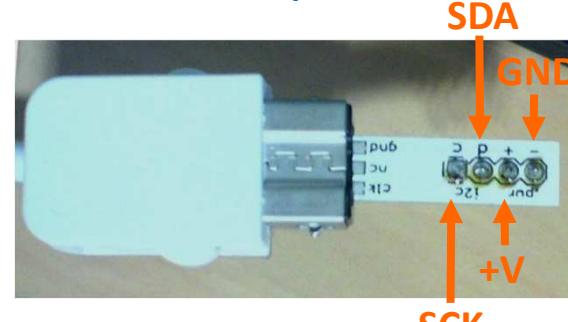
- 1 - standard interface I2C
- 2 - a 3 accelerometer axis resolution 10bits.
- 3 - a 2 axis analog joystick with a 8 bit A/D converter
- 4- 2 buttons



Nunchuk PinOUT

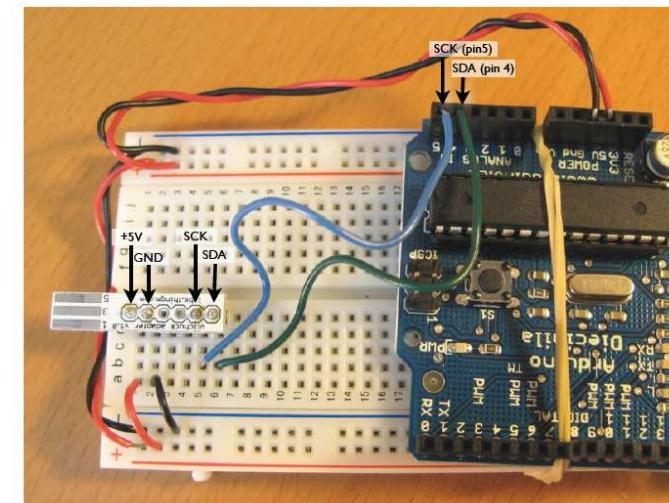


PinOUT adapter



Chip listing:

- U1: Unknown manufacturer - "FNURVL" "A 405" "633KM" (custom processing chip? - TQFP48)  
U2: ST Microelectronics LIS3L02AL 3-axis accelerometer - "1806" "3L02AE" "615" "MLT" (accelerometer - LGA-8)



## Program: Nunchuk Print

read data every 100ms, and display:

- the joystick Position (x, y)
- accelerometer (x, y, z)
- buttons Z, C

# POUR ALLER PLUS LOIN

The Arduino must send a HandShake to communicate with the Nunchuck.  
So, send first 2 bytes "0x40, 0x00".

Then send only 1 byte "0x00" whenever you request data from the Nunchuck.

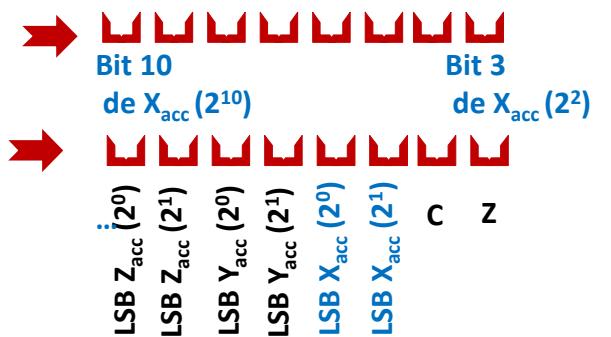
These data will be returned to you by packs of 6 bytes.

Byte	Description	Values of sample Nunchuk
1	X-axis value of the analog stick	Min(Full Left):0x1E / Medium(Center):0x7E / Max(Full Right):0xE1
2	Y-axis value of the analog stick	Min(Full Down):0x1D / Medium(Center):0x7B / Max(Full Right):0xDF
3	X-axis acceleration value	Min(at 1G):0x48 / Medium(at 1G):0x7D / Max(at 1G):0xB0
4	Y-axis acceleration value	Min(at 1G):0x46 / Medium(at 1G):0x7A / Max(at 1G):0xAF
5	Z-axis acceleration value	Min(at 1G):0x4A / Medium(at 1G):0x7E / Max(at 1G):0xB1
6	Button state (Bits 0/1) / acceleration LSB	Bit 0: "Z"-Button (0 = pressed, 1 = released) / Bit 1: "C" button (0 = pressed, 1 = released) / Bits 2-3: X acceleration LSB / Bits 4-5: Y acceleration LSB / Bits 6-7: Z acceleration LSB

Beware:

X acceleration values are on 10 bits !

8 bits from n°3 + 2 bits from n°6



Data byte receive							
Joystick X							
Joystick Y							
Accelerometer X (bit 9 to bit 2 for 10-bit resolution)							
Accelerometer Y (bit 9 to bit 2 for 10-bit resolution)							
Accelerometer Z (bit 9 to bit 2 for 10-bit resolution)							
Accel. Z bit 1	Accel. Z bit 0	Accel. Y bit 1	Accel. Y bit 0	Accel. X bit 1	Accel. X bit 0	C-button	Z-button

Octet 1 : Joystick - Axis X : a byte from 0 to 255

Octet 2 : Joystick - Axis Y : a byte from 0 to 255

Octet 3 : Accelerometer - Axe X : a byte for 8 MSB over the possible 10 bits, if we only use this byte we'll only have values from 0 to 255

Octet 4 : Accelerometer - Axe Y : a byte for 8 MSB over the possible 10 bits, if we only use this byte we'll only have values from 0 to 255

Octet 5 : Accelerometer - Axe Z : a byte for 8 MSB over the possible 10 bits, if we only use this byte we'll only have values from 0 to 255

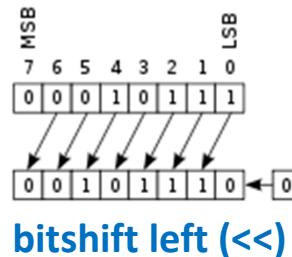
Octet 6 : from MSB to LSB : 2 bits for Z axis accelerometer, concatenated with previous 8 bits => obtaining 10 bits résolution, ie. Values from 0 to 1023.

2 bits for Y axis accelerometer, 2 bits for X axis accelerometer. 1 bit for C button: 0:pushed, 1:release. 1 bit for Z button: 0:pushed, 1:release.

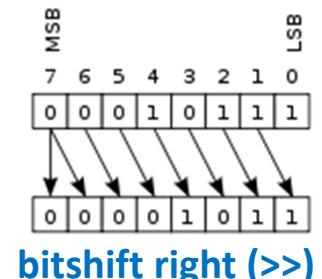
# bitshift left (<<), bitshift right (>>) OPERATORS

There are two C/C++ bit shift operators:

1 - the left shift operator << (bitshift left) who causes the shift to the left of the bits.



2 - right shift operator >> (bitshift right) that causes the right bit shift.



Syntax: variable << number\_of\_bits ou variable >> number\_of\_bits

Examples:

int a = 5; // binary: 000000000000101 (*over 16 bits*)

int b = a << 3; // binary: 0000000000101000 => 40 (decimal) ( $5 \times 2^3 = 5 \times 8 = 40$ )

int c = b >> 3; // binary: 000000000000101 => we return to 5

Remark 1 : When you offset a number to the left, left bit is literally lost!

Example:

int a = 5; // binary: 000000000000101

int b = a << 14; // binary: 0100000000000000 – the first 1 from 101 is lost!

Remark 2 : bitshift left of a n number is multiplication by  $2^n$

$1 << 0 == 1 = 2^0$

$1 << 1 == 2 = 2^1$

$1 << 2 == 4 = 2^2$

...

$1 << 8 == 256 = 2^8$

$1 << 10 == 1024 = 2^{10}$

...

In contrary, bitshift right of a n number is division by  $2^n$

# Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^)

Bitwise operator (AND, OR, XOR) perform their calculations on each bit position.

Bitwise AND (&):

If the two input bits are 1, the output is 1, otherwise 0.

The most common uses of the AND is to select bits by "masking."

Bitwise OR (|):

The OR binary of two bits is 1 if one OR the other input bits is 1, or 0.

Bitwise XOR (^): This operator is very similar to the OR operator (|), only it puts a bit to 0 when the two input bits are 1:

0 0 1 1 operand1

0 1 0 1 operand2

-----

0 0 0 1 (operand1 & operand2) - returned result

0 0 1 1 operand1

0 1 0 1 operand2

-----

0 1 1 1 (operand1 | operand2) - returned result

0 0 1 1 operand1

0 1 0 1 operand2

-----

0 1 1 0 (operand1 ^ operand2) - returned result

## APPLICATIONS: decoding of the information of the Nunchuk

```
// byte outbuf[5] contains bits for z and c buttons and it also contains the least significant bits for the
accelerometer data
// so we have to check each bit of byte outbuf[5]
if ((outbuf[5] >> 0) & 1) {    z_button = 1;    }
if ((outbuf[5] >> 1) & 1) {    c_button = 1;    }
if ((outbuf[5] >> 2) & 1) {    accel_x_axis += 2;    }
```

## TP4-A

# THE WII NUNCHUK CONTROL

```
#include <Wire.h>
#include <string.h>
#undef int
#include <stdio.h>

uint8_t outbuf[6]; // array to store arduino output
int cnt = 0;
int ledPin = 13;

void
setup ()
{
beginSerial (19200);
Serial.print ("Finished setup\n");
Wire.begin (); // join i2c bus as master
nunchuck_init (); // send the initialization handshake
}

void
nunchuck_init ()
{
Wire.beginTransmission (0x52); // transmit to device 0x52
Wire.send (0x40); // sends memory address
Wire.send (0x00); // sends sent a zero.
Wire.endTransmission (); // stop transmitting
}

void
send_zero ()
{
Wire.beginTransmission (0x52); // transmit to device 0x52
Wire.send (0x00); // sends one byte
Wire.endTransmission (); // stop transmitting
}

Void loop ()
{
Wire.requestFrom (0x52, 6); // request data from nunchuck
while (Wire.available ())
{
outbuf[cnt] = nunchuk_decode_byte (Wire.receive ()) // receive byte as an integer
digitalWrite (ledPin, HIGH); // sets the LED on
cnt++;
}
}
```

```
// If we received the 6 bytes, then go print them
if (cnt >= 5)
{
print ();
}
cnt = 0;
send_zero (); // send the request for next bytes
delay (100);
}

// Print the input data we have received
// accel data is 10 bits long
// so we read 8 bits, then we have to add
// on the last 2 bits. That is why I
// multiply them by 2 * 2
void
print ()
{
int joy_x_axis = outbuf[0];
int joy_y_axis = outbuf[1];
int accel_x_axis = outbuf[2] * 2 * 2;
int accel_y_axis = outbuf[3] * 2 * 2;
int accel_z_axis = outbuf[4] * 2 * 2;

int z_button = 0;
int c_button = 0;

// byte outbuf[5] contains bits for z and c buttons
// it also contains the least significant bits for the
accelerometer data
// so we have to check each bit of byte outbuf[5]
if ((outbuf[5] >> 0) & 1)
{
z_button = 1;
}
if ((outbuf[5] >> 1) & 1)
{
c_button = 1;
}

if ((outbuf[5] >> 2) & 1)
{
accel_x_axis += 2;
}
if ((outbuf[5] >> 3) & 1)
{
accel_x_axis += 1;
}}
```

<http://www.windmeadow.com/node/42>

```
if ((outbuf[5] >> 4) & 1)
{
accel_y_axis += 2;
}

if ((outbuf[5] >> 5) & 1)
{
accel_y_axis += 1;
}

if ((outbuf[5] >> 6) & 1)
{
accel_z_axis += 2;
}
if ((outbuf[5] >> 7) & 1)
{
accel_z_axis += 1;
}

Serial.print (joy_x_axis, DEC);
Serial.print ("\t");
Serial.print (joy_y_axis, DEC);
Serial.print ("\t");
Serial.print (accel_x_axis, DEC);
Serial.print ("\t");
Serial.print (accel_y_axis, DEC);
Serial.print ("\t");
Serial.print (accel_z_axis, DEC);
Serial.print ("\t");
Serial.print (z_button, DEC);
Serial.print ("\t");
Serial.print (c_button, DEC);
Serial.print ("\t");
Serial.print ("\r\n");
}

// Encode data to format that most wiimote drivers except
// only needed if you use one of the regular wiimote drivers
char
nunchuk_decode_byte (char x)
{
x = (x ^ 0x17) + 0x17;
return x;
}
```

<= RETOUR

<http://todbot.com/blog/bionicarduino/>

# Nintendo Wii Nunchuck & SERVO (Library version)

```
/* PROGRAMME PRINCIPAL
*/
#include <math.h>
#include <Servo.h>
#include "Wire.h"
#include "WiiChuck.h"
#define MAXANGLE 90
#define MINANGLE -90

Servo myservo; // create servo object to control a servo
int potpin = 0; // analog pin used to connect the potentiometer

WiiChuck chuck = WiiChuck();

int angleStart, currentAngle;
int tillerStart = 0;
double angle;

void setup()
{
    //nunchuck_init();
    Serial.begin(115200);
    chuck.begin();
    chuck.update();
    //chuck.calibrateJoy();
    // attaches the servo on pin 9 to the servo object
    myservo.attach(9);
}

void loop()
{
    delay(20);
    chuck.update();
    // sets the servo position according to the scaled value
    myservo.write(chuck.readPitch());
    Serial.print(chuck.readRoll());
    Serial.print(", ");
    Serial.print(chuck.readPitch());
    Serial.print(", ");

    Serial.print((int)chuck.readAccelX());
    Serial.print(", ");
    Serial.print((int)chuck.readAccelY());
    Serial.print(", ");
    Serial.print((int)chuck.readAccelZ());
    Serial.println();
}
```

Insertion of the library: WiiChuck.h  
(cf next slide)

# Nintendo Wii Nunchuck & SERVO (Library version)

```

/*
 * Nunchuck -- Use a Wii Nunchuck
 * Tim Hirzel http://www.growdown.com
 *
notes on Wii Nunchuck Behavior.
This library provides an improved derivation of rotation angles from the nunchuck accelerometer
data. The biggest difference over existing libraries (that I know of ) is the full 360 degrees of Roll data
from teh combination of the x and z axis accelerometer data using the math library atan2. It is
accurate with 360 degrees of roll (rotation around axis coming out of the c button, the front of the
wii), and about 180 degrees of pitch (rotation about the axis coming out of the side of the wii).
(read more below)
In terms of mapping the wii position to angles, its important to note that while the Nunchuck
sense Pitch, and Roll, it does not sense Yaw, or the compass direction. This creates an important
disparity where the nunchuck only works within one hemisphere. At a result, when the pitch
values are less than about 10, and greater than about 170, the Roll data gets very unstable,
essentially, the roll data flips over 180 degrees very quickly. To understand this property better,
rotate the wii around the axis of the joystick. You see the sensor data stays constant (with noise).
Because of this, it cant know the difference between arriving upside via 180 degree Roll, or 180
degree pitch. It just assumes its always 180 roll.
*/
/* This file is an adaptation of the code by these authors:
 * Tod E. Kurt, http://todbot.com/blog/
 * The Wii Nunchuck reading code is taken from Windmeadow Labs
 * http://www.windmeadow.com/node/42
 */
#ifndef WiiChuck_h
#define WiiChuck_h

#include "WProgram.h"
#include <Wire.h>
#include <math.h>

// these may need to be adjusted for each nunchuck for calibration
#define ZEROX 510
#define ZERØY 490
#define ZEROZ 460
#define RADIUS 210 // probably pretty universal

#define DEFAULT_ZERO_JOY_X 124
#define DEFAULT_ZERO_JOY_Y 132

class WiiChuck {
private:
    byte cnt;
    uint8_t status[6];// array to store wiichuck output
    byte averageCounter;
    //int accelArray[3][AVERAGE_N]; // X,Y,Z
    int i;
    int total;
    uint8_t zeroJoyX; // these are about where mine are
    uint8_t zeroJoyY; // use calibrateJoy when the stick is at zero to correct
    int lastJoyX;
    int lastJoyY;
    int angles[3];

    boolean lastZ, lastC;
public:
    byte joyX;
    byte joyY;
    boolean buttonZ;
    boolean buttonC;
    void begin()
    {
        Wire.begin();
        cnt = 0;
        averageCounter = 0;
        // instead of the common 0x40 -> 0x00 initialization, we use 0xF0 -> 0x55 followed
        //by 0xFB -> 0x00, this lets us use 3rd party nunchucks (like cheap $4 ebay ones)
        // while still letting us use official ones. Only side effect is that we no longer need
        //to decode bytes in _nunchuk_decode_byte
        // see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1264805255

        Wire.beginTransmission(0x52);// device address
        Wire.send(0xF0);
        Wire.send(0x55);
        Wire.endTransmission();
        delay(1);
        Wire.beginTransmission(0x52);
        Wire.send(0xFB);
        Wire.send(0x00);
        Wire.endTransmission();
        update();
        for (i = 0; i<3;i++) {
            angles[i] = 0;
        }
        zeroJoyX = DEFAULT_ZERO_JOY_X;
        zeroJoyY = DEFAULT_ZERO_JOY_Y;
    }

    void calibrateJoy() {
        zeroJoyX = joyX;
        zeroJoyY = joyY;
    }

    void update() {
        Wire.requestFrom (0x52, 6); // request data from nunchuck
        while (Wire.available ()) { // receive byte as an integer
            status[cnt] = _nunchuk_decode_byte (Wire.receive()); //
            cnt++;
        }
        if (cnt > 5) {
            lastZ = buttonZ;
            lastC = buttonC;
            lastJoyX = readJoyX();
            lastJoyY = readJoyY();
            //averageCounter++;
            //if (averageCounter >= AVERAGE_N)
            //    averageCounter = 0;
            cnt = 0;
            joyX = (status[0]);
            joyY = (status[1]);
            for (i = 0; i < 3; i++)
                //accelArray[i][averageCounter] = ((int)status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2) ) >> ((i+1)*2)));
                angles[i] = (status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2) ) >> ((i+1)*2)));
                //accelYArray[averageCounter] = ((int)status[3] << 2) + ((status[5] & B00110000) >> 4);
                //accelZArray[averageCounter] = ((int)status[4] << 2) + ((status[5] & B11000000) >> 6);
                buttonZ = !(status[5] & B00000001);
                buttonC = !(status[5] & B00000010) >> 1;
                _send_zero(); // send the request for next bytes
        }
    }

    // UNCOMMENT FOR DEBUGGING
    //byte * getStatus() {
    //    return status;
    //}
    float readAccelX() {
        // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
        return (float)angles[0] - ZERØX;
    }
    float readAccelY() {
        // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
        return (float)angles[1] - ZERØY;
    }
    float readAccelZ() {
        // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
        return (float)angles[2] - ZERØZ;
    }
    boolean zPressed() {
        return (buttonZ && ! lastZ);
    }
    boolean cPressed() {
        return (buttonC && ! lastC);
    }
    // for using the joystick like a directional button
    boolean rightJoy(int thresh=60) {
        return (readJoyX() > thresh and lastJoyX <= thresh);
    }
    // for using the joystick like a directional button
    boolean leftJoy(int thresh=60) {
        return (readJoyX() < -thresh and lastJoyX >= -thresh);
    }
    int readJoyX() {
        return (int) joyX - zeroJoyX;
    }
    int readJoyY() {
        return (int) joyY - zeroJoyY;
    }
    // R, the radius, generally hovers around 210 (at least it does with mine)
    // int R() {
    //    return sqrt(readAccelX() * readAccelX() +readAccelY() * readAccelY() +
    //readAccelZ() * readAccelZ());
    //}
    // returns roll degrees
    int readRoll() {
        return (int)(atan2(readAccelX(),readAccelZ()) / M_PI * 180.0);
    }
    // returns pitch in degrees
    int readPitch() {
        return (int)(acos(readAccelY()/RADIUS)/ M_PI * 180.0); // optionally
swap 'RADIUS' for 'R'
    }
private:
    byte _nunchuk_decode_byte (byte x)
    {
        //decode is only necessary with certain initializations
        //x = (x ^ 0x17) + 0x17;
        return x;
    }
    void _send_zero()
    {
        Wire.beginTransmission (0x52); // transmit to device 0x52
        Wire.send (0x00); // sends one byte
        Wire.endTransmission (); // stop transmitting
    };
#endif

```

## SYNCHRONOUS SERIAL COMMUNICATION : SPI

## THE BUS SPI - GENERAL INFORMATION

The I2C bus is not alone, he is followed by other series synchronous circuits type SPI or One Wire. These last two names, however, are less well standardized than the I2C bus.

### 1 - SPI "Serial Peripheral Interface": (or Microwire (for National Semiconductor) or bus serial 3-wire)

- Standard established by Motorola and taken up by different brands
  - Synchronous serial data bus, using 4-wire (3 fils + 1 Masse)
  - Connection of type master-slave, allowing the connection of several circuits => dedicated for inter-composants communication, see inter-cartes, within the same system
  - Full Duplex (the data can travel in both directions at the same time) - as opposed to (for example), the bus I2C (2 son + mass) which cannot do.
  - Master-slave - a master only possible on the bus several slaves can coexist on a bus
  - The selection of the recipient is made by a dedicated line chip select (CS).

The SPI bus contains 4 logical signals:

- **SCLK** — Horloge (because SERIAL SYNCHRONOUS BUS) (*generated by MASTER*)

- **MOSI** — Master Output, Slave Input (*generated by MASTER*)

- **MISO** — Master Input, Slave Output (*généré par l'esclave*)

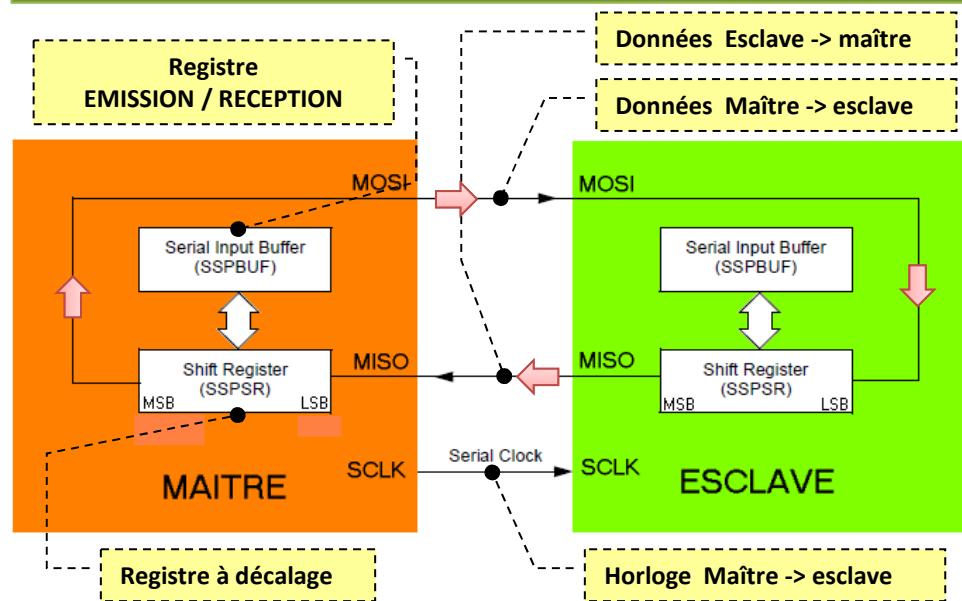
- **SS** — Slave Select, Slave active at LOW state (*generated by MASTER*)

If high level, the component ignores what is happening on the bus => several peripheral components that share the 3 lines !



Beware, you can encounter other names:  
SDI, DI, SI — Serial Data IN ⇔ MISO  
SDO, DO, SO — Serial Data OUT ⇔ MOSI  
nCS, CS, nSS, STE ⇔ SS

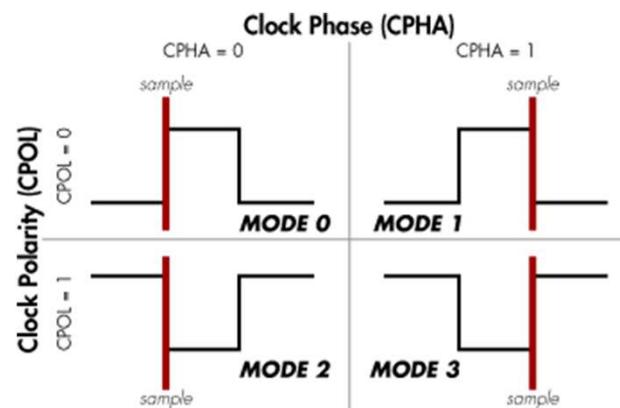
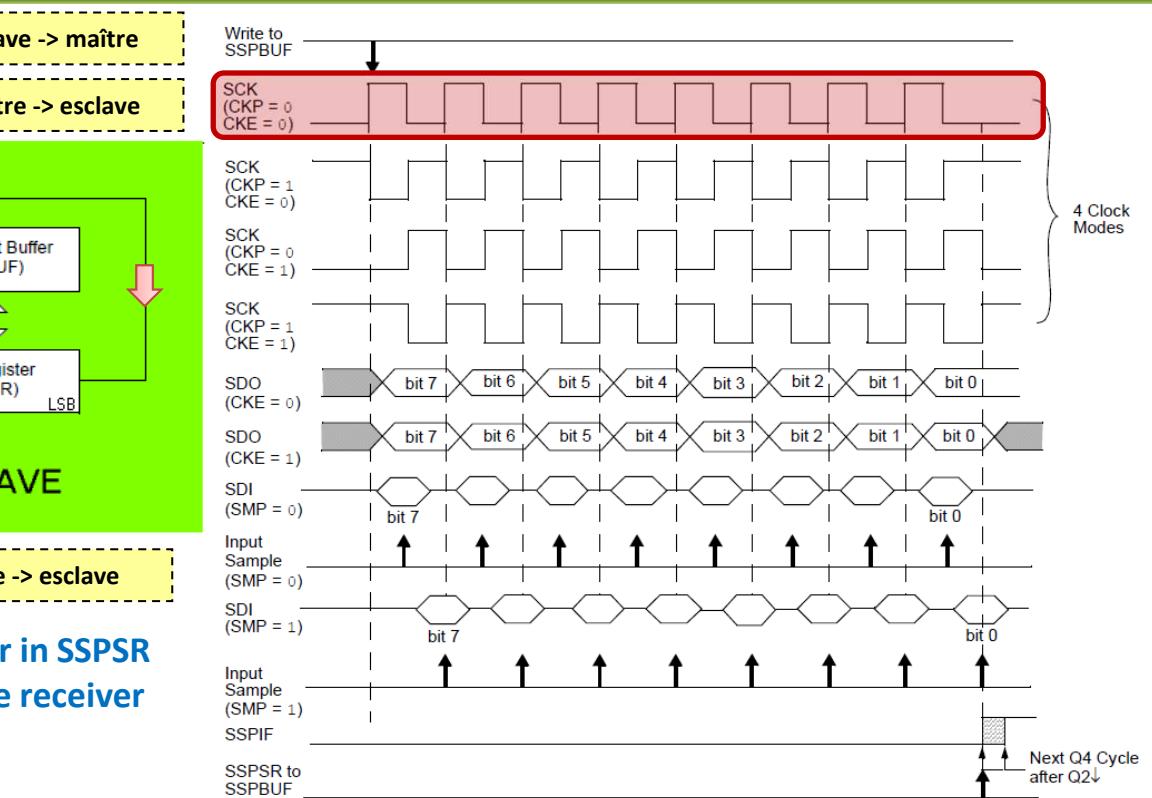
# THE SPI - CHRONOGRAM GENERAL BUS



1. the transmitter copy SSPBUF of the master in SSPSR
2. the output data are sent in SSPSR from the receiver to the clock pulses
3. receiver copy SSPSR in its SSPBUF

Three settings:

- Clock speed (set by the master).
- The polarity of the clock (CPOL (Clock polarity) parameter): determines the logical level of the SCK line at rest.
- The phase of the clock (setting CPHA (Clock phase)): determines the front on which the data is changed and the front on which the data will be read.

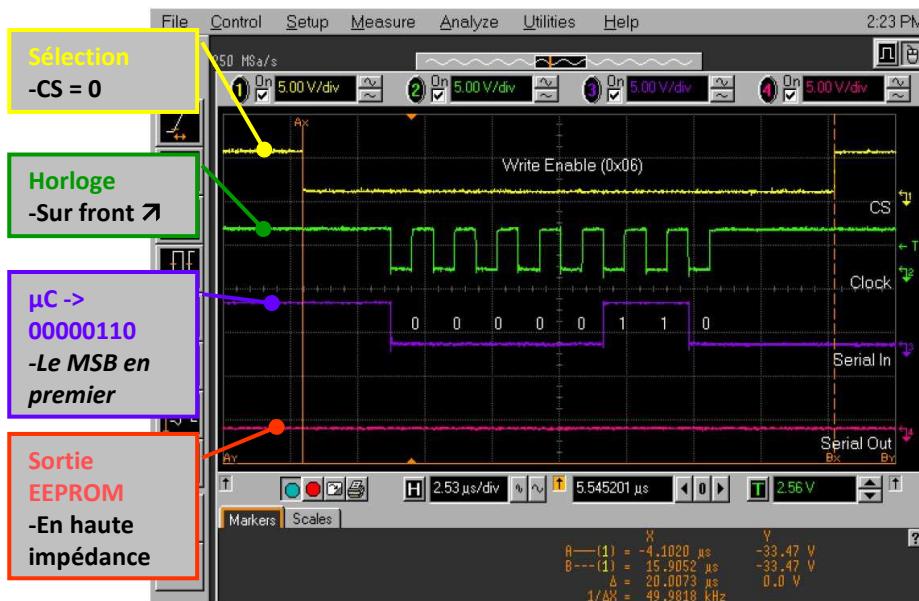


CPOL and CPHA have 2 possible states :  
4 possible configurations

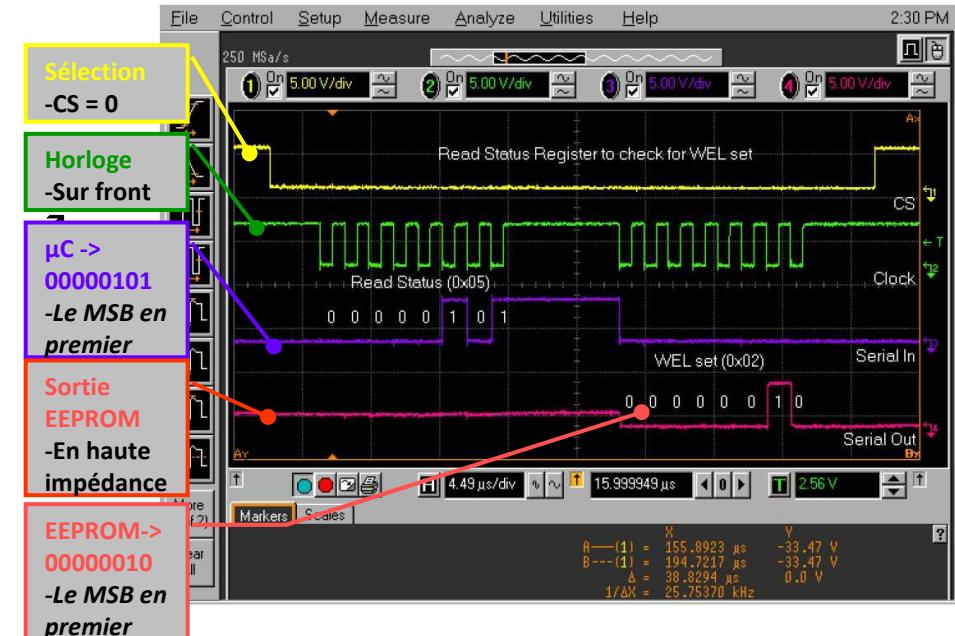
SPI-mode	CPOL (CKP)	CPHA (CKE)
0	0	0
1	0	1
2	1	0
3	1	1

(MASTER AND SLAVE MUST HAVE THE SAME PARAMETERS)

# THE SPI - CHRONOGRAM GENERAL BUS



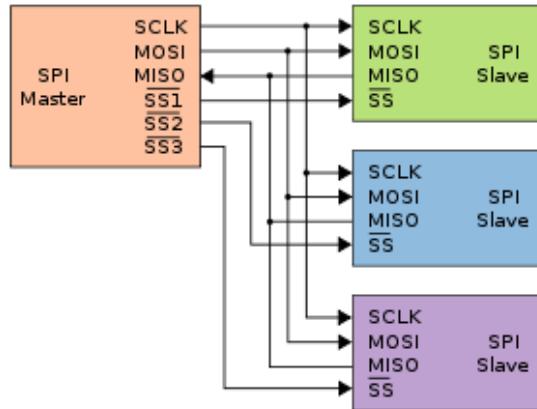
Example: in an EEPROM write permission



Example: the registry of an EEPROM reading

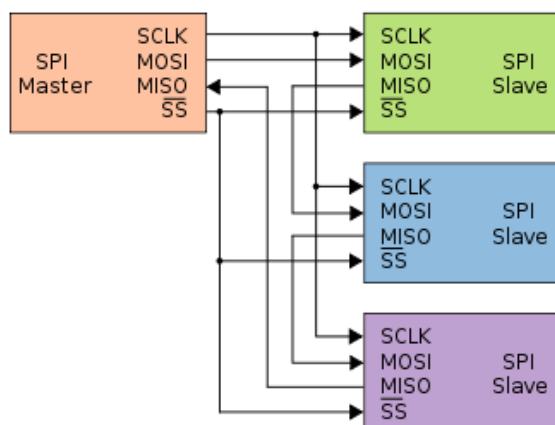
# THE BUS SPI - FEATURES

Several slaves but only one active slave at the same time:



Several slaves (daisy chain): simultaneous selection

Some SPI components are designed to be chained together, simplifying the connections between components, reducing the number of lines SS.



BUS SPEED :

In master mode, the transmission speed is selected by 3 bits of the register SPCON (Serial Peripheral CONtrol register): SPR1, SPR2 and SPR0.

The clock frequency is chosen among 7 frequencies obtained by division of the operating frequency of the microcontroller.

SPR2 : SPR1 : SPR0	SPI frequency	Example $F_{\text{transmission}}$ (quartz at $F_{\mu c}=20\text{MHz}$ )
000	$F_{\mu c}/2$	10
001	$F_{\mu c}/4$	5
010	$F_{\mu c}/8$	2.5
011	$F_{\mu c}/16$	1.25
100	$F_{\mu c}/32$	0.625
101	$F_{\mu c}/64$	0.3125
110	$F_{\mu c}/128$	0.156

# CONTRÔLE D'UN POTENTIOMETRE DIGITAL AVEC LE BUS SPI

The management of the SPI bus is easy because the Arduino contains a library named SPI.h

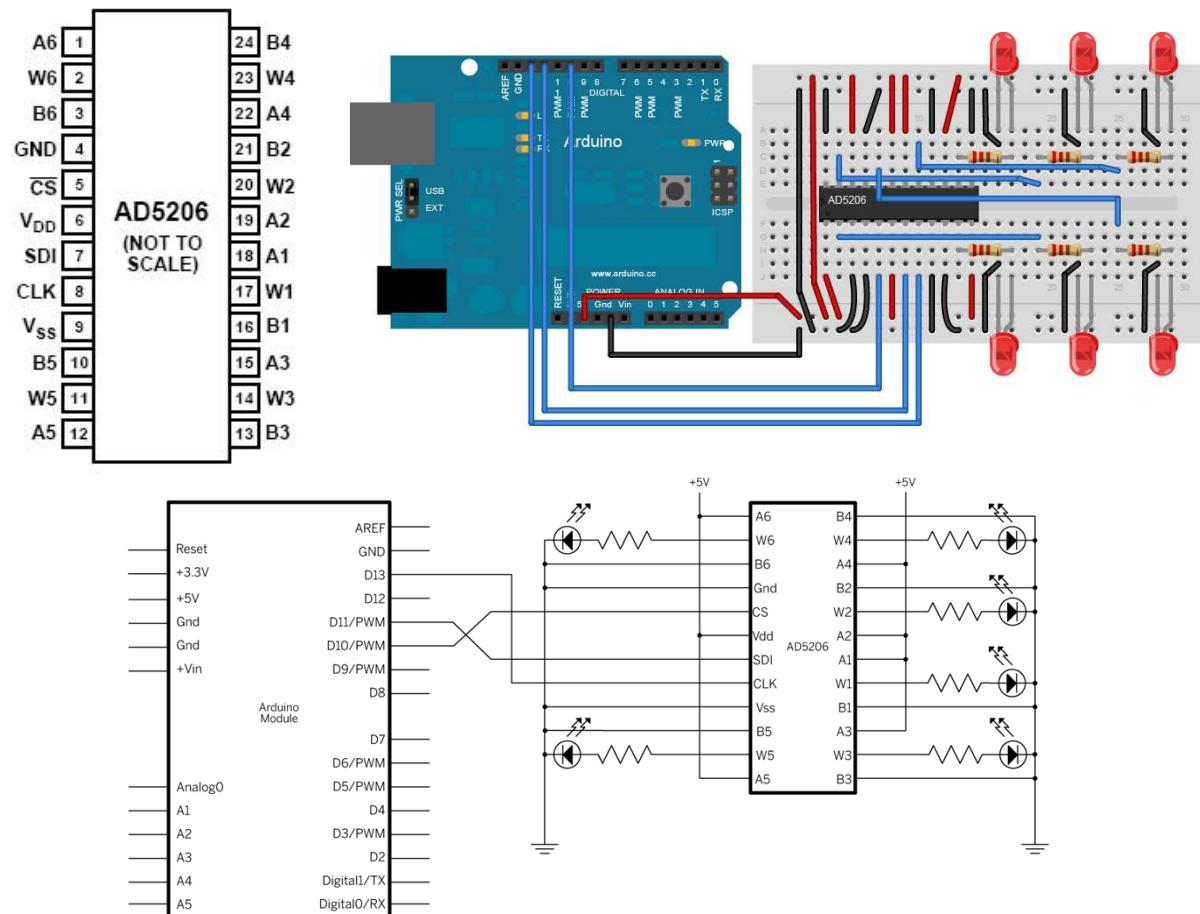
<http://www.arduino.cc/en/Reference/SPI>

<http://tronixstuff.wordpress.com/2011/05/13/tutorial-arduino-and-the-spi-bus/>

## AD5206 Digital Potentiometer

AD5206 PIN FUNCTION DESCRIPTIONS

Pin No.	Name	Description
1	A6	A Terminal RDAC #6.
2	W6	Wiper RDAC #6, addr = $101_2$ .
3	B6	B Terminal RDAC #6.
4	GND	Ground.
5	CS	Chip Select Input, Active Low. When CS returns high, data in the serial input register is decoded based on the address bits and loaded into the target RDAC latch.
6	V <sub>DD</sub>	Positive power supply, specified for operation at both +3 V or +5 V. (Sum of  V <sub>DD</sub>   +  V <sub>SS</sub>   < 5.5 V.)
7	SDI	Serial Data Input. MSB First.
8	CLK	Serial Clock Input, positive edge triggered.
9	V <sub>SS</sub>	Negative Power Supply, specified for operation at both 0 V or -2.7 V. (Sum of  V <sub>DD</sub>   +  V <sub>SS</sub>   < 5.5 V.)
10	B5	B Terminal RDAC #5.
11	W5	Wiper RDAC #5, addr = $100_2$ .
12	A5	A Terminal RDAC #5.
13	B3	B Terminal RDAC #3.
14	W3	Wiper RDAC #3, addr = $010_2$ .
15	A3	A Terminal RDAC #3.
16	B1	B Terminal RDAC #1.
17	W1	Wiper RDAC #1, addr = $000_2$ .
18	A1	A Terminal RDAC #1.
19	A2	A Terminal RDAC #2.
20	W2	Wiper RDAC #2, addr = $001_2$ .
21	B2	B Terminal RDAC #2.
22	A4	A Terminal RDAC #4.
23	W4	Wiper RDAC #4, addr = $011_2$ .
24	B4	B Terminal RDAC #4.



<http://arduino.cc/en/Tutorial/SPIDigitalPot>

## Functions

[begin\(\)](#)  
[end\(\)](#)  
[setBitOrder\(\)](#)  
[setClockDivider\(\)](#)  
[setDataMode\(\)](#)  
[transfer\(\)](#)

# A SPI BUS CONTROL FOR THE DIGITAL POTENTIOMETER

## /\* Digital Pot Control

This example controls an Analog Devices AD5206 digital potentiometer. The AD5206 has 6 potentiometer channels. Each channel's pins are labeled

A - connect this to voltage

W - this is the pot's wiper, which changes when you set it

B - connect this to ground.

The AD5206 is SPI-compatible, and to command it, you send two bytes, one with the channel number (0 - 5) and one with the resistance value for the channel (0 - 255).

The circuit:

- \* All A pins of AD5206 connected to +5V
- \* All B pins of AD5206 connected to ground
- \* An LED and a 220-ohm resistor in series connected from each W pin to ground
- \* CS - to digital pin 10 (SS pin)
- \* SDI - to digital pin 11 (MOSI pin)
- \* CLK - to digital pin 13 (SCK pin)

created 10 Aug 2010 by Tom Igoe

Thanks to Heather Dewey-Hagborg for the original tutorial, 2005  
\*/

<http://arduino.cc/en/Tutorial/SPIDigitalPot>

```
// include the SPI library:  
#include <SPI.h>  
// set pin 10 as the slave select for the digital pot:  
const int slaveSelectPin = 10;  
  
void setup() {  
    // set the slaveSelectPin as an output:  
    pinMode(slaveSelectPin, OUTPUT);  
    // initialize SPI:  
    SPI.begin()();  
}  
  
void loop() {  
    // go through the six channels of the digital pot:  
    for (int channel = 0; channel < 6; channel++) {  
        // change the resistance on this channel from min to max:  
        for (int level = 0; level < 255; level++) {  
            digitalPotWrite(channel, level);  
            delay(10);  
        }  
        // wait a second at the top:  
        delay(100);  
        // change the resistance on this channel from max to min:  
        for (int level = 0; level < 255; level++) {  
            digitalPotWrite(channel, 255 - level);  
            delay(10);  
        }  
    }  
}  
  
int digitalPotWrite(int address, int value) {  
    // take the SS pin low to select the chip:  
    digitalWrite(slaveSelectPin, LOW);  
    // send in the address and value via SPI:  
    SPI.transfer(address);  
    SPI.transfer(value);  
    // take the SS pin high to de-select the chip:  
    digitalWrite(slaveSelectPin, HIGH);  
}
```

## THE SPI APPLICATION: TOWARDS THE FPGA

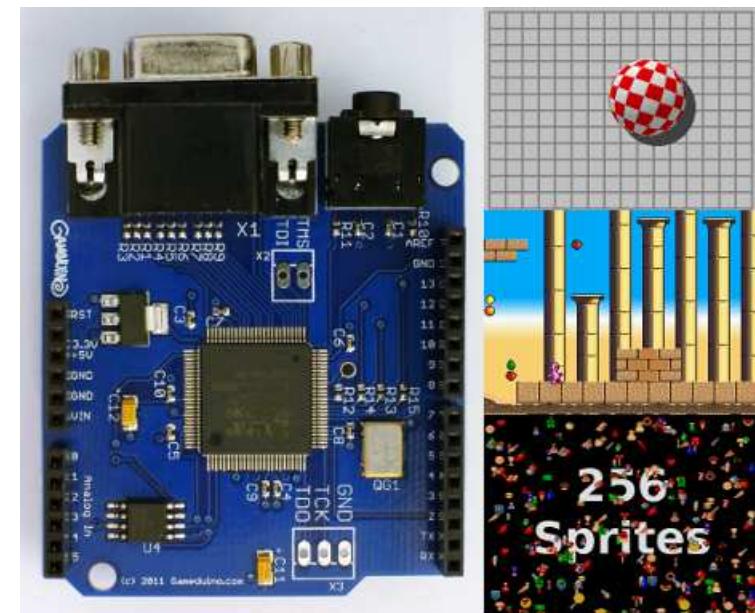
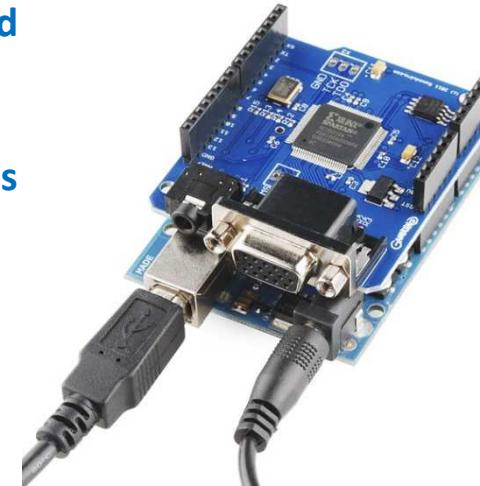
Gameduino is an Arduino with a controller graphics and sound shield type FPGA Xilinx.

His control is done by the SPI bus.

The code is used to send orders to the card to manage records toggle, memory and graphics functions.

The data is then managed by the Xilinx FPGA that allows:

- video output is 400x300 pixels in 512 colors
- all color processed internally at 15-bit precision
- compatible with any standard VGA monitor (800x600 @ 72Hz)
- background graphics
  - 512x512 pixel character background
  - 256 characters, each with independent 4 color palette
  - pixel-smooth X-Y wraparound scroll
- foreground graphics
  - each sprite is 16x16 pixels with per-pixel transparency
  - each sprite can use 256, 16 or 4 colors
  - four-way rotate and flip
  - pixel-perfect sprite collision detection
- audio output is a stereo 12-bit frequency synthesizer
- 64 independent voices 10-8000 Hz
- per-voice sine wave or white noise
- sample playback channel



<http://excamera.com/sphinx/gameduino/>

# THE SPI APPLICATION: TOWARDS THE FPGA

Application Note: Spartan-3A FPGAs



XAPP974 (v1.1.3) March 24, 2009

## Indirect Programming of SPI Serial Flash PROMs with Spartan-3A FPGAs

Author: Jameel Hussein

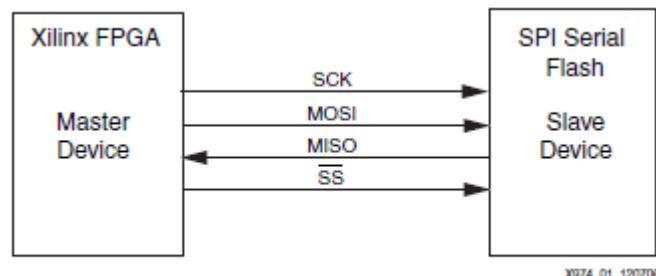
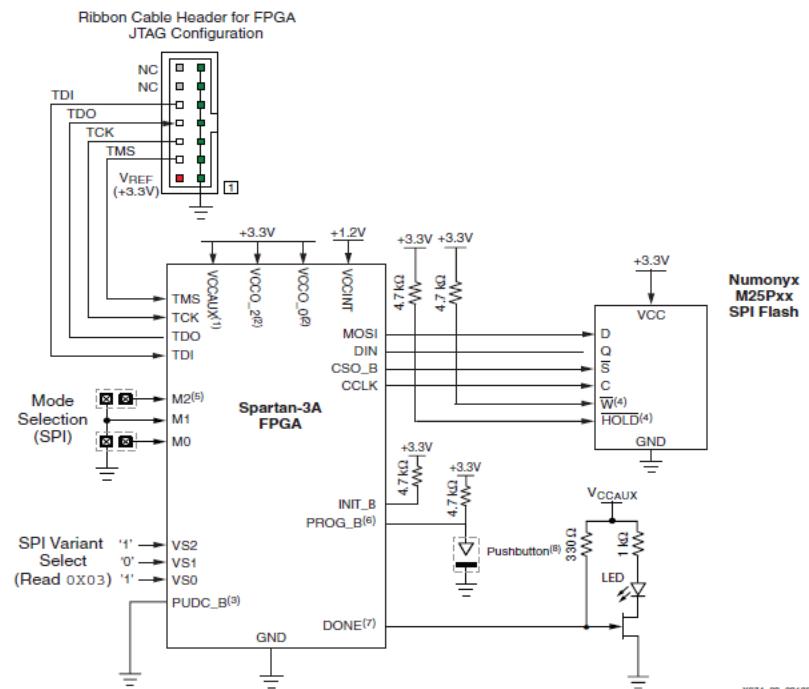


Figure 1: Basic Block Diagram for SPI Configuration Mode



[http://www.xilinx.com/support/documentation/application\\_notes/xapp974.pdf](http://www.xilinx.com/support/documentation/application_notes/xapp974.pdf)

# THE SPI APPLICATION: TOWARDS THE FPGA

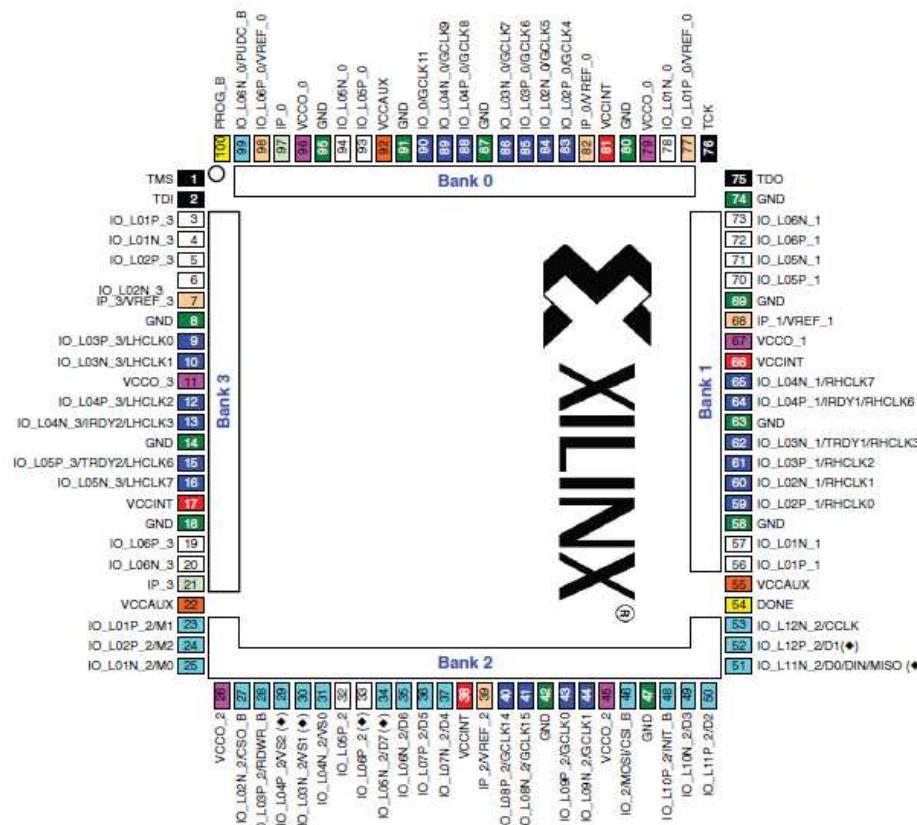
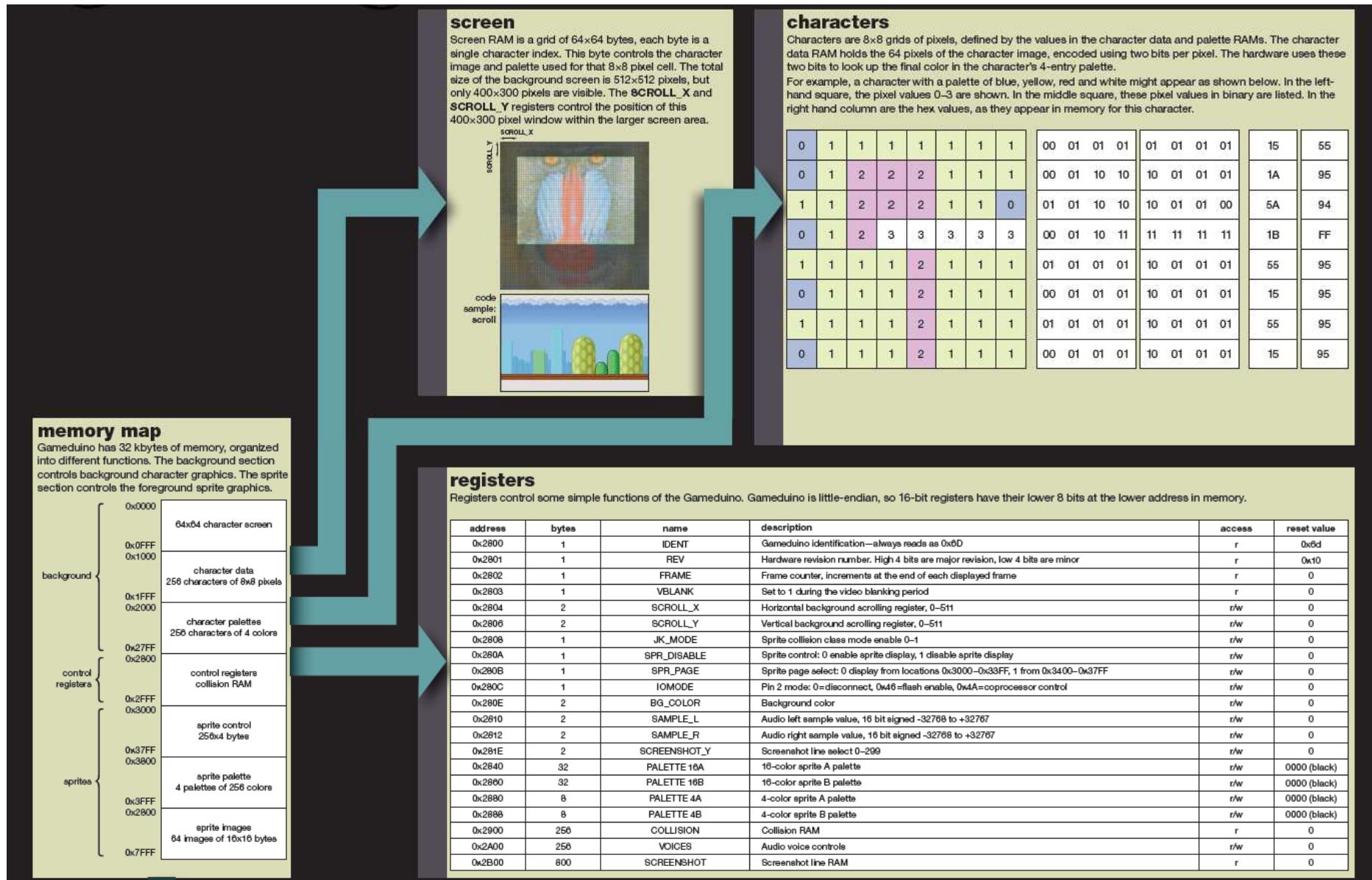


Figure 17: VQ100 Package Footprint - XC3S50A (Top View)

17	I/O: Unrestricted, general-purpose user I/O	20	DUAL: Configuration pins, then possible user I/O	6	VREF: User I/O or input voltage reference for bank
2	INPUT: Unrestricted, general-purpose input pin	23	CLK: User I/O, input, or global buffer input	6	VCCO: Output voltage supply for bank
2	CONFIG: Dedicated configuration pins	4	JTAG: Dedicated JTAG port pins	4	VCCINT: Internal core supply voltage (+1.2V)
0	N.C.: Not connected	13	GND: Ground	3	VCCAUX: Auxiliary supply voltage

[http://www.xilinx.com/support/documentation/data\\_sheets/ds529.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf)

# GAMEDUINO REFERENCE



# GAMEDUINO REFERENCE

**sprite control**

Gameduino has 256 hardware sprites: 16x16 pixel images that can appear anywhere on the screen. Sprites are drawn from back-to-front, so higher-numbered sprites cover up lower-numbered ones. Each sprite's appearance is controlled by a 32-bit word:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
C	IMAGE	Y		PAL	ROT	X																											

XX coordinate, 0-511. 0 is the left edge of the screen, 399 is the right edge.  
 YY coordinate, 0-511. 0 is the top edge of the screen, 299 is the bottom edge.  
 IMAGE Spriteimageselect, 0-63. Selects which source image the sprite uses.  
 PAL Sprites palette select, 4 bits. Controls how pixel data is turned into color.  
 ROT Sprite rotate, 3 bits. Rotates a sprite by quarter-turns.  
 C Collision class, 0 is J, 1 is K.  
 To hide a sprite park it off the screen by setting its Y coordinate to 400.  
 To make a large sprite draw several sprites together in a grid pattern. For example, four 16x16 sprites can be arranged to make a single 32x32 sprite.  
 To spin a sprite use the ROT field to rotate it, and use extra animation frames for finer rotations.  
 To animate a sprite you can
 

- change the IMAGE field
- change the PAL field to do simple color animation
- change the ROT field to apply flips and rotates of 90, 180 and 270 degrees
- load new data to the source image.

**sprite palette select**

Each pixel of the sprite image is fetched and looked up in a sprite palette. This palette is a list of colors. Gameduino gives you several palette options: a 256-color palette, a 16-color palette and a 4-color palette. Why not always use the 256-color palette? Because using the smaller palette options lets you squeeze more images into memory. 256 bytes of sprite image RAM can hold one 16x16 sprite image in 256-color mode, two images in 4-bit mode (with a 16 color palette), or four images in 2-bit mode (4 color palette).

3	2	1	0
0	0	P	
0	1	N	AB
1	N	AB	

8-bit mode: Each byte indexes into 256-color palette, P  
 4-bit mode: The high (N=1) or low (N=0) 4 bits from each byte index into 16-color palette A (AB=0) or B (AB=1)  
 2-bit mode: Two bits from each byte (N=3 is highest, N=0 is lowest) index into 4-color palette A (AB=0) or B (AB=1)

code sample: palettes.

**sprite rotate**

Each sprite has a 3-bit ROT field that applies a simple rotation and flip to the sprite image.

Y flip	X flip	XY swap
--------	--------	---------

Y flip flip the image top-to-bottom  
 X flip flip the image left-to-right  
 XY swap flip the image diagonally

By using these in combination, the sprite image can be rotated:

ROT	Y flip	X flip	XY swap	results
0	0	0	0	R
1	0	0	1	¤
2	0	1	0	Я
3	0	1	1	¤
4	1	0	0	Б
5	1	0	1	Б
6	1	1	0	Б
7	1	1	1	¤

code sample: rotation

**sprite collision detection**

The Gameduino has a special memory area that you can use to detect when sprites overlap. As it draws the image, Gameduino keeps track of which pixels cover others, and writes the results into the collision RAM.

Each byte in the collision RAM corresponds with the same numbered sprite. If the sprite does not cover another then the byte's value is 0xFF. But if the sprite covers any part of another sprite, then the value is the number of the other sprite.

For example, if sprites 00-03 are arranged like this:

address	value	meaning
0x2900	FF	sprite 00 did not cover up any other sprites
0x2901	00	sprite 01 covered up some pixels from sprite 00
0x2902	FF	sprite 02 did not cover up any other sprites
0x2903	02	sprite 03 covered some pixels from sprite 02

code sample: collision

**sprite collision class**

In a game you might have the following rules:

- when the player touches an enemy bomb, the player dies
- when a player's missile touches an enemy, that enemy dies.

Here is a typical in-game situation:

Notice that bomb 04 covers both bomb 03 and player 00. In this situation we're much more interested that bomb 04 is covering player 00. For this reason, the hardware has a mode JK\_MODE where it ignores "friendly" collisions. In this mode, each sprite belongs a collision class J or K. Collision notifications only happen when a J sprite covers a K sprite, or when a K sprite covers up a J sprite.

address	value	meaning
0x2900	FF	sprite 00 no collision
0x2901	FF	sprite 01 no collision
0x2902	01	sprite 02 covers some pixels from sprite 01
0x2903	FF	sprite 03 no collision
0x2904	00	sprite 04 covers some pixels from sprite 00

code sample: jkcollision

**colors**

Gameduino stores colors in an ARGB1555 format:

A	R	G	B
---	---	---	---

Each color field red (R) green (G) and blue (B) has a range 0-31.

Gameduino is little-endian, so a color stored in two bytes stored starting at address is:

address { G <sub>1</sub> G <sub>0</sub> B <sub>1</sub> B <sub>0</sub> }	address+1 { A R <sub>1</sub> R <sub>0</sub> B <sub>1</sub> B <sub>0</sub> }
---	---

The A bit controls transparency. When A=1 the pixel is transparent and the other fields are ignored.

For sprites, transparent pixels show through the background layer. For the background layer, transparent pixels show BG\_COLOR.

code sample: bgcolor

# SPRITES



In early video games, 'hardware' sprites were a method for managing bitmaps pretending to be part of a single image on a screen.

Computer, chip	Year	Sprites on screen	Sprites on line	Max. texels on line	Texture width	Texture height	Colors	Hardware zoom	Rotation	Background	Collision detection	Transparency	Source
Amiga, Denise	1985	Display list	8	?	16	Arbitrary	3, 15	Vertical by display list	No	2 bitmap layers	Yes	Color key	
Amiga (AGA), Lisa	1992	Display list	8	?	16, 32, 64	Arbitrary	3, 15	Vertical by display list	No	2 bitmap layers	Yes	Color key	
Amstrad Plus, Asic	1990	Display list run by CPU	16 min.	?	16	16	15	1, 2, 4x vertical, 1, 2, 4x horizontal	No	Bitmap layer	No	Color key	[7]
Atari 2600, TIA	1977	Multipplied by CPU	9 (with triplication)	51 (with triplication)	1, 8	262	1	1, 2, 4, 8x horizontal	Horizontal mirroring	1 bitmap layer	Yes	Color key	[8]
Atari 8-bit, GTIA/ANTIC	1979	Display list	8	40	2, 8	128, 256	1,3	1, 2x vertical, 1, 2, 4x horizontal	No	1 tile or bitmap layer	Yes	Color key	[9]
C64, VIC-II	1982	Display list run by CPU	8	96, 192	12, 24	21	1, 3	1, 2x integer	No	1 tile or bitmap layer	Yes	Color key	[10]
Game Boy	1989	40	10	80	8	8, 16	3	No	No	1 tile layer	No	Color key	[11]
GBA	2001	128	128	1210	8, 16, 32, 64	8, 16, 32, 64	15, 255	Yes affine	Yes affine	4 layers, 2 layers, and 1 affine layer, 2 affine layers	No	Color key, blending	[12]
Gameduino	2011	256	96	1,536	16	16	255	No	Yes	1 tile layer	Yes	Color key	[13]
NES, RP200x	1983	64	8	64	8	8, 16	3	No	Horizontal and vertical mirroring	1 tile layer	Partial	Color key	[15]
Neo Geo	1990	384	96	1536	16	Up to 512	15	sprite shrinking	No	No	No	Color key	.
Out Run, dedicated hardware	1986	128	32	?	8	8	?	Yes anisotropic	No	3 tile layers	?	Alpha	[16][17]
PC Engine, HuC6270A	1987	64	16	256	16, 32	16, 32, 64	15	No	No	1 tile layer	Yes	Color key	
Sega Master System Sega Game Gear	1985	64	8	64	8	8, 16	15	1, 2x integer	No	1 tile layer	Yes	Color key	[18]
Mega Drive	1988	80	20	320	8, 16, 24, 32	8, 16, 24, 32	15	No	No	2 tile layers	Yes	Color key	[19]
Sharp X68000	1987	128	32	?	16	16	15	No	No	?	?	Color key	
SNES	1990	128	34	272	8, 16, 32, 64	8, 16, 32, 64	15	Background only	Background only	3 tile layers or 1 affine mapped tile layer	Yes	Color key, averaging	
Texas Instruments TMS9918	1979	32	4	64	8, 16	8, 16	1	1, 2x integer	No	1 tile layer	Partial	Color key	[20]
Yamaha V9938	1986	32	8	128	8, 16	8, 16	1, 3, 7, 15 per line	1, 2x integer	No	1 tile or bitmap layer	Partial	Color key	
Yamaha V9958	1988	32	8	128	8, 16	8, 16	1, 3, 7, 15 per line	1, 2x integer	No	1 tile or bitmap layer	Partial	Color key	
Computer, chip	Year	Sprites on screen	Sprites on line	Max. texels on line	Texture width	Texture height	Colors	Hardware zoom	Rotation	Background	Collision detection	Transparency	Source

[http://en.wikipedia.org/wiki/Sprite\\_%28computer\\_graphics%29#Hardware\\_sprites](http://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29#Hardware_sprites)

## BUS SPI PROS & CONS



### CONS:

- Needs more legs of a case than the I<sup>2</sup>C or a UART which use only two.
- no address Possible, but a line of selection by slave mode not chained. -
- The Protocol has no acknowledgement. The master can talk in a vacuum without knowing.
- There may be only one master on the bus.
- Use on short distances unlike protocols RS-232, RS-485 or CAN bus



### PROS:

- Full duplex Communication
- enough throughput compared to I<sup>2</sup>C
- flexibility of the number of bits to transmit
- simplicity of the hardware interface
- no need referee because no possible collision
- slaves use the clock of the master and therefore did not need precision oscillator
- sharing a common bus for the clock, MISO and MOSI between slave devices
- have no need a unique ID unlike the I<sup>2</sup>C, the GPIB and SCSI

## **SYNCHRONOUS SERIAL COMMUNICATION : ONE-WIRE**

## OTHER PROTOCOLS: ONE WIRE...

### 2. ONE WIRE created by Dallas (now called company MAXIM)

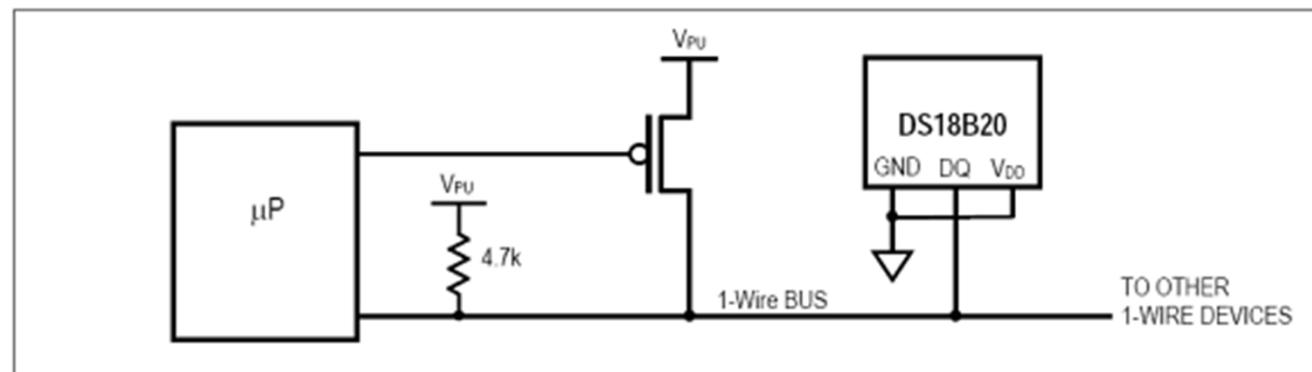
"One Wire" technology is a serial communication bus allowing the addressing and the use of several sensors in communication series on a single digital PIN.

Data bus synchronous series using only a single thread (+ 1 mass of potential reference).

Allows you to transmit data in both directions, each component can be transmitter and receiver at all moment.

- But it also allows to feed (parasite power mode) for the less greedy!
- The management of the bus ONE WIRE is easy because the Arduino contains a third-party library named: OneWire.h (author: Jim Studt)

Figure 4. Supplying the Parasite-Powered DS18B20 During Temperature Conversions



[http://www.milesburton.com/?title=Dallas\\_Temperature\\_Control\\_Library](http://www.milesburton.com/?title=Dallas_Temperature_Control_Library)

<http://www.arduino.cc/playground/Learning/OneWire>

# SYNCHRONOUS SERIAL COMMUNICATION : A COMPARISON

# COMPARISON OF PROTOCOLS: I2C/SPI/ONE WIRE

I2C	Transmission Speed	SPI
Although there are variations of the I2C rising above 1 MHz, the vast majority of implementations use generally 100 or 400 kHz	For the SPI it is possible to find some components beyond 20 Mbits	

I2C	Topology	SPI
It's a real protocol that allows the interconnection of multiple boxes in different configurations: slave /master, master/Multiple slaves, Multiple masters/multiple slaves	Generally point-to-point, you can connect several slave but must then have additional lines. A single master which generates the clock.	

I2C	Consumption	SPI
the collector configuration / drain opened on the 2 lines of transmission (SDA + SCL), relatively high consumption	TTL/CMOS signal => <b>low consumption</b>	

## Pros / Cons

- ✓ Whether to interconnect several boxes and speed is not a problem, prefer the I2C because it is a Protocol (which is not the case of the SPI)
- ✓ If you want to speed the SPI is far ahead...
- ✓ Software implementation on I/O : it is much easier (and it takes less resources) to the SPI by software on I/O pins of the I2C due to the state machine.
- ✓ Implementation: the I2C is more complicated to implement (just see the number of questions on the I2C in forums...).
- ✓ The interconnection of several boxes is also more delicate with the I2C because it takes into account the impedances of each of the boxes to calculate resistances of reminders.

## Bus One Wire :

The One Wire technology is comparable to the I2C technology (and better?).

# COMPARISON OF PROTOCOLS: I<sup>2</sup>C/SPI/ONE WIRE

## Bus I<sup>2</sup>C - Inter Integrated Circuit:

Mainly developed for home automation and home electronics, this Protocol is very popular and used by many components.

The I<sup>2</sup>C Protocol requires only a 3-wire to operate and can be implemented on any microcontroller.  
Supports several masters (so collision)

## Bus SPI - Serial Peripheral Interface:

The SPI bus is Full Duplex and based on master-slave communication.

The SPI bus supports multiple slaves but one can communicate both with the master (so selection logic!)

## Bus One Wire :

La technologie One Wire est une technologie comparable à la technologie I<sup>2</sup>C (en mieux ?).

## EXAMPLE OF APPLICATION OF THE SERIAL COMMUNICATION



to Wi-Fi

to Ethernet



to graphic LCD

to 8-servo controller

Lantronix Wi-Port and  
Lantronix Xport  
<http://lantronix.com/>

Seetron Serial Graphic display and  
Mini SSC  
<http://www.seetron.com/slcds.htm>  
<http://www.seetron.com/ssc.htm>



to Roomba

“Hacking Roomba”,  
out in a few weeks,  
<http://hackingroomba.com/>

### III – 8 LIBRAIRIES

## III – 8-1 LIBRAIRIE EXAMPLES

# EEPROM LIBRARY

The ATMEGA of Arduino has an EEPROM: memory whose values are retained when the map is off:

- 512 bytes on ATmega168,
- 1024 bytes (1 Ko) on ATmega328,
- 4096 bytes (4 Ko) on ATmega8,

The EEPROM library allows you to read and write these bytes:

## Reading example

```
#include <EEPROM.h>
int a = 0;
int value;

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

void loop()
{
  value = EEPROM.read(a);
  Serial.print(a);Serial.print("\t");
  Serial.print(value);
  Serial.println();
  a = a + 1;
  if (a == 512)
    a = 0;
  delay(500);
}
```

## Syntax

```
EEPROM.read(address)
```

## Writing example

```
#include <EEPROM.h>

void setup()
{
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, i);
}
void loop() {

/* EEPROM Clear: Sets all of the bytes of the EEPROM to 0.*/
#include <EEPROM.h>
void setup()
{
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++) EEPROM.write(i, 0);
}
void loop() {
```

*Note: a writing EEPROM takes 3.3 ms. EEPROM memory has a specified life of 100,000 write/erase cycles. Be careful in the way we use it.*

## III – 8 - 2 CREATE A LIBRAIRY

## TO GO FURTHER – GOOD PRACTICES - 1

When you start to make big projects, it becomes useful or even essential to (very) well organize his code:  
separate your code into different files in order to have separate logical entities from each other.

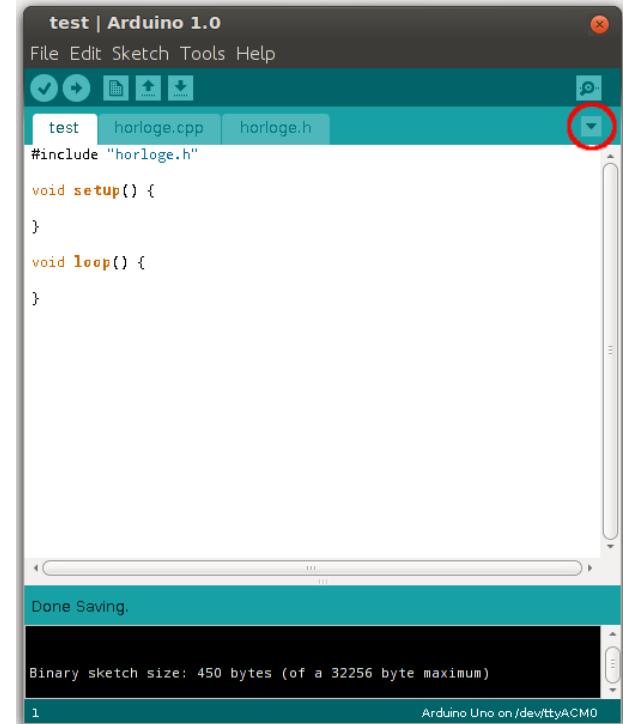
Generally, we do files by logic units.

In practice: to create a new file in the Arduino IDE, simply click on the small arrow at the top of the editing of the code space and then click on "New tab" as highlighted in the screenshot below:

Attention, whenever you want to create a new code file we will not create one but two!

**1 - The file. h:** the first file will have the extension .h meaning header. This file will regroup the prototypes of the functions as well as the definitions of structures or classes but we'll cover that later.

**Prototype of a function:** sets the name of the function, what goes inside (parameters) and what comes out (the return variable). So your main program will have an idea of how your function works outside.



Example: Let's imagine a file for RTC clock managing : horloge.h

```
char getHeure(); char getMinute(); char getSeconde();
char getJour(); char getMois();
char getAnnee(); void setHeure(char val);
void setMinute(char val); void setSeconde(char val);
void setJour(char val); void setMois(char val);
void setAnnee(char val); void afficherDate();
void afficherHeure(); void afficherDateHeure();
```

As you can see, with these definitions we can know what is supposed to do the service through its name and the type of variable it handles input and output.

## TO GO FURTHER – GOOD PRACTICES - 2

### 2 - The second file .cpp:

it is used to implement the functions defined in the .h, i.e. write the content of your functions,

**First step: include file prototypes via the #include preprocessor command:**

For example:

#include "horloge.h" // horloge.h for our example

**Attention: this line should be the first of your .cpp file and it does not; at the end.**

### 3 - Link our files to the main program:

1 - Please make sure that your .h and .cpp files are in the same folder as your .ino

2 - include the at the top of your file's .h file: #include "horloge.h"

### Functions code:

```
/* fichier horloge.cpp */  
#include "horloge.h"  
char getHeure() { Serial.println("getHeure"); return 0; }  
char getMinute() { Serial.println("getHeure"); return 0; }  
char getSeconde() { Serial.println("getHeure"); return 0; }  
char getJour() { Serial.println("getHeure"); return 0; }  
char getMois() { Serial.println("getHeure"); return 0; }  
char getAnnee() { Serial.println("getHeure"); return 0; }  
void setHeure(char val) { Serial.print("setHeure : "); Serial.println(val, DEC); }  
void setMinute(char val) { Serial.print("setMinute : "); Serial.println(val, DEC); } void  
setSeconde(char val) { Serial.print("setSeconde : "); Serial.println(val, DEC); } void  
setJour(char val) { Serial.print("setJour : "); Serial.println(val, DEC); }  
void setMois(char val) { Serial.print("setMois : "); Serial.println(val, DEC); }  
void setAnnee(char val) { Serial.print("setAnnee : "); Serial.println(val, DEC); }  
void afficherDate() { Serial.println("afficherDate"); }  
void afficherHeure() { Serial.println("afficherHeure"); }  
void afficherDateHeure() { Serial.println("afficherDateHeure"); }
```



### 4 - Use:

you can now make simple calls to your custom functions in the program (setup(), loop() functions...!).

**Now, when you compile, the compiler will fetch the file pointed by the include, compile it, then link it in your main program.**

Remark1: it can happen that the connection with the symbols/library Arduino does not correctly. In this case, add the following at the beginning of your .h or .cpp include: #include "Arduino.h"

Remark2: with this technique: you can code in C++ to create classes and thus push the Organization even further!

# CREATE AN ARDUINO LIBRARY - 1

- Creation of custom libraries to simplify the reuse of code.
- Many libraries are already present and integrated to the Arduino software.
- They allow you to program faster and more simply your systems.

We will see how to create a library through a concrete example: the Morse library.

Example from the official tutorial of creation of library:<http://arduino.cc/en/Hacking/LibraryTutorial>

***Caution: Library is a false friend. The French translation is Bibliothèque not library.***

***Let's look at the code: the parts of the code that we will put in our new library are clearly  
pinMode (pin, OUTPUT)  
and dot() and dash() functions.***

***Now let's see how to create a library, e.g. Morse.***

Code C

```
int pin = 13;

void setup()
{
    pinMode(pin, OUTPUT);
}

void loop()
{
    // SOS
    // 3x POINT 3x TIRET 3xPOINT
    dot(); dot(); dot();
    dash(); dash(); dash();
    dot(); dot(); dot();
    delay(3000);
}

void dot() // point
{
    digitalWrite(pin, HIGH);
    delay(250);
    digitalWrite(pin, LOW);
    delay(250);
}

void dash() // tiret
{
    digitalWrite(pin, HIGH);
    delay(1000);
    digitalWrite(pin, LOW);
    delay(250);
}
```

## CREATE AN ARDUINO LIBRARY - 2

**STRUCTURE of a library:** Arduino libraries are composed of at least 2 files:

- a header file ending by .h (which contains the definitions of the functions available)
- a source file ending with .cpp (which contains the implementation of the code, i.e. the code inside the functions that have been defined in the header file).

**Header file (.h):** in our case Morse.h

Contains all of the declarations of the methods of the Morse class that will represent the library.

Before the declaration of the class, you must ensure that the file of head will not be victim of infinite inclusion.

So as usual in C, we add a protection with preprocessor directives (e.g. #ifndef...).

To use the constants or functions as a part of the Arduino core, add an include before the declaration of the class =>

We also add a private property (private): \_pin which will contain the number of LEDs to use pin.

In this way, during initialization of the Morse object, we can both put pin in OUTPUT mode and store it for not having to give it back when it will make a call to another method.

So now we can tackle the class declaration. What gives the final =>

```
Code : C
#ifndef Morse_h
// si Morse_h n'est pas défini

#define Morse_h
// On le définit

// Code de la classe ICI

#endif // Fin si

Code : C
#include "WProgram.h"

Code : C
#ifndef Morse_h
#define Morse_h

#include "WProgram.h"

class Morse
{
public:
    Morse(int pin);
    void dot();
    void dash();
private:
    int _pin;
};

#endif
```

# CREATE AN ARDUINO LIBRARY - 3

The source (.cpp) file: in our case Morse.cpp.

The source file contains the implementation of the methods described in the header file.

So there was this code =>

Install the library:

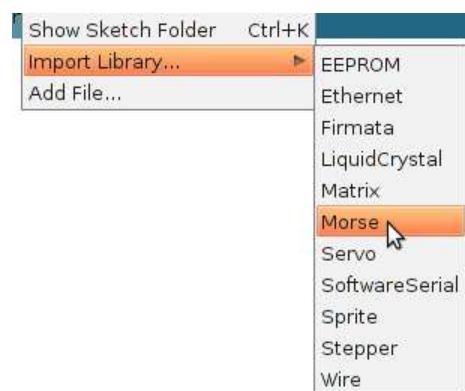
Arduino in the software folder is a subfolder named "libraries" where each subfolder represents a library.

To add ours, we create a folder on the behalf of our library:

e.g. "Morse" folder

and put the two files Morse.h and Morse.cpp.

Use the library in the IDE: from now on you can see the library in the IDE in the Sketch menu > Import Library if you restart it:



Code : C

```
/*
Morse.cpp - Bibliothèque pour code morse avec une DEL.

Récupéré du site officiel d'Arduino.
http://arduino.cc/en/Hacking/LibraryTutorial
Créé David A. Mellis, November 2, 2007.
Le code est du domaine public.

*/
#include "WProgram.h"
#include "Morse.h"

Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin;
}

void Morse::dot()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

## CREATE AN ARDUINO LIBRARY - 4

Click on the 'MORSE' menu item, which will add the necessary code to use your library in Arduino:

```
#include <Morse.h>
```

### FINAL RESULT:

Thanks to the library, you can now simplify our initial code. This allows us to have this code =>

-this code, of course, has the same effect as the code of the start.

The creation of the library is now ended.

Abuse of this feature to simplify your code.

Creating libraries ensures the logic of your code and decreases the time of future development.

In addition, you can share libraries to benefit the community !

### Code : C

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
    morse.dot(); morse.dot(); morse.dot();
    morse.dash(); morse.dash(); morse.dash();
    morse.dot(); morse.dot(); morse.dot();
    delay(3000);
}
```



# Arduino Style Guide for Writing Libraries

This is a style guide to writing library APIs in an Arduino style. Some of these run counter to professional programming practice. We're aware of that, but it's what's made it possible for so many beginners to get started with Arduino easily. So please code with these principles in mind. If you have suggestions on how to make Arduino libraries clearer for that core audience, please jump in the discussion. This is a work in progress.

**Be kind to the end user.** Assume you are writing an API for an intelligent person who has not programmed before. Come up with a clear mental model of the concept you're working with, and the terms and functions you will use.

**Match your API to the underlying capabilities.** You don't want to expose implementation details to the user but you also don't want an API that suggests an inaccurate mental model of the possibilities. For example, if there are only a few possible options for a particular setting, don't use a function that takes an int, as it implies you can use any value you want.

**Organize your public functions around the data and functionality that the user wants.** Quite often, the command set for a particular electronic module is overly complicated for the most common uses, or can be re-organized around higher level functionality. Think about what the average person thinks the thing does, and try to organise your API functions around that. Adafruit's [BMP085 library](#) is a good example. The `readPressure()` command performs all the necessary steps to get the final pressure. The library wraps this commonly executed series of functions into a high-level single command which returns the value the user's looking for in a format she expects. It abstracts away not only the low-level I2C commands, but also the mid-level temperature and pressure calculations, while still offering those mid-level functions as public functions for those who want them.

**Don't assume knowledge of pointers.** Beginning users of C find this the biggest roadblock, and get very confused by & and \*, so whenever you can avoid having them hanging out in the API, do so. One way is to pass by reference using array notation rather than \* notation, for example.

```
void printArray( char* array);
```

can be replaced by

```
void printArray(char[] array);
```

Though there are some libraries where we pass pointers by using structures like const chars, avoid anything that requires the user to pass them. For example, rather than:

```
foo.readAccel(&x, &y, &z);
```

use something like this:

```
xAxis = adxl.readX();
yAxis = adxl.readY();
zAxis = adxl.readZ();
```

<https://www.arduino.cc/en/Reference/APIStyleGuide>

## III-9 THE SHIELDS & THEIR CREATION

## III-9 THE SHIELDS & THEIR CREATION

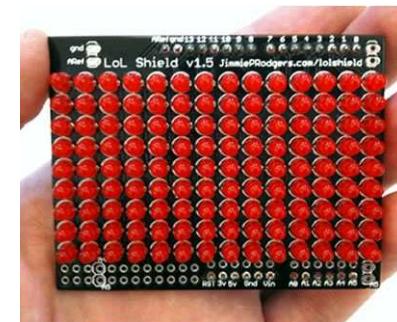
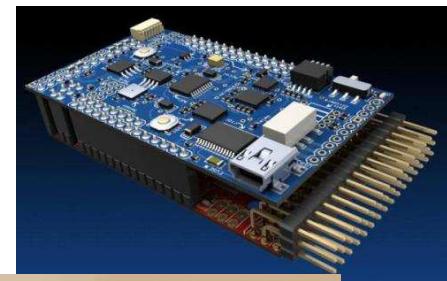
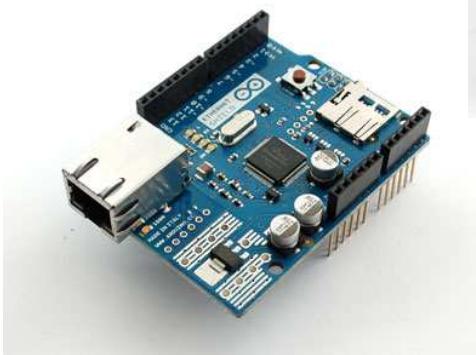
## INTERFACE BOARD or SHIELDS

One of the major interests of the Arduino is there today on the market, a variety of interface cards (~ 300), called Shields, able to cover most of the needs of an application.

These shields neighboring dimensions of the Arduino to plug directly on the peripheral connectors of this last. They follow the same philosophy as Arduino: easy to set up and inexpensive to produce.

There are many shields between 15 and 30€ that add interesting features:

- Ethernet
- GPS
- Wireless transmissions (Xbee, LoRa...),
- interfaces graphic LCD color, monochrome, or text,
- keyboard
- Game shield
- arrays of leds...

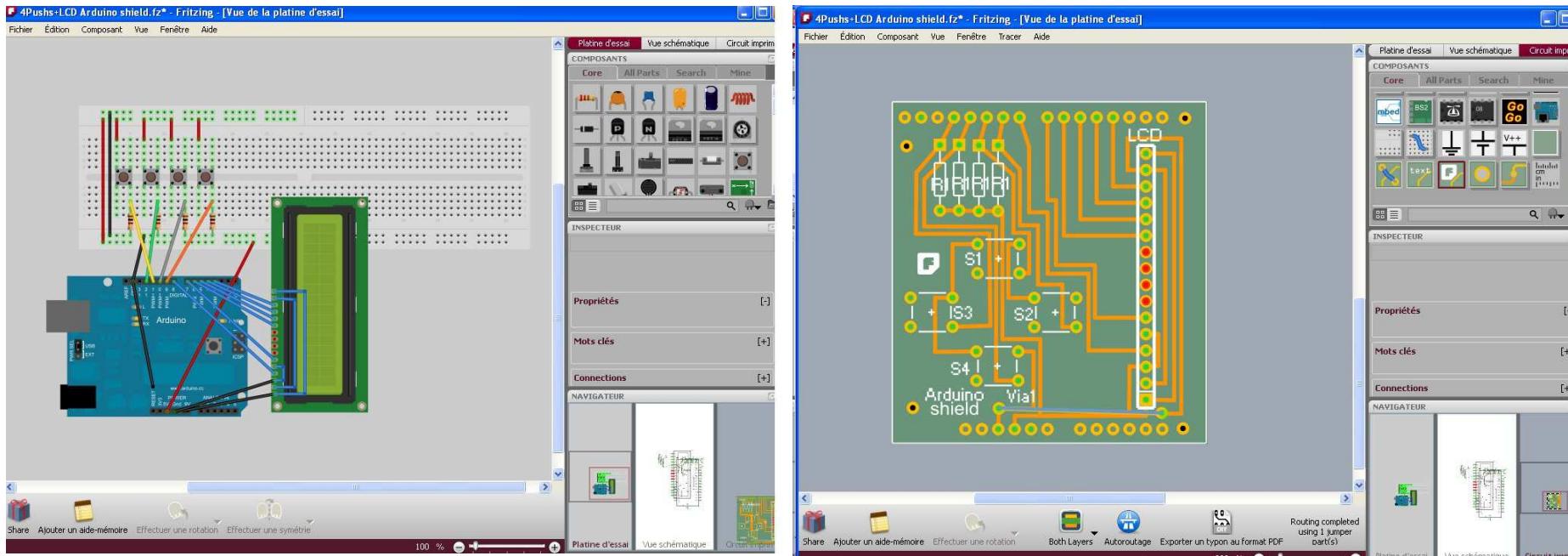


<http://shieldlist.org/>

# EXAMPLE

Example: 4 buttons LCD

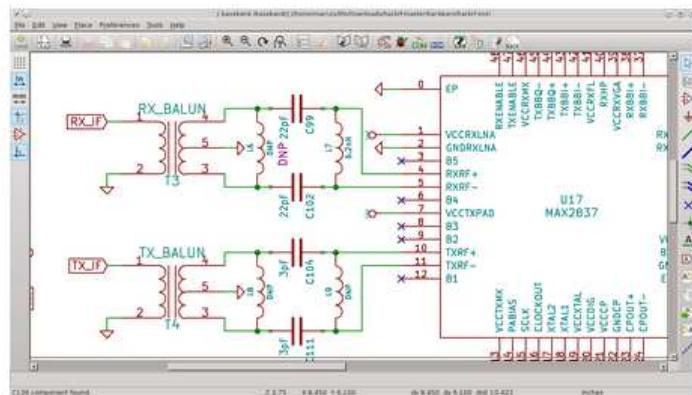
<http://fritzing.org/projects/arduino-lcd-4-push-buttons/>



*Fritzing can draw its way graphic, schematic, or electronic circuit*

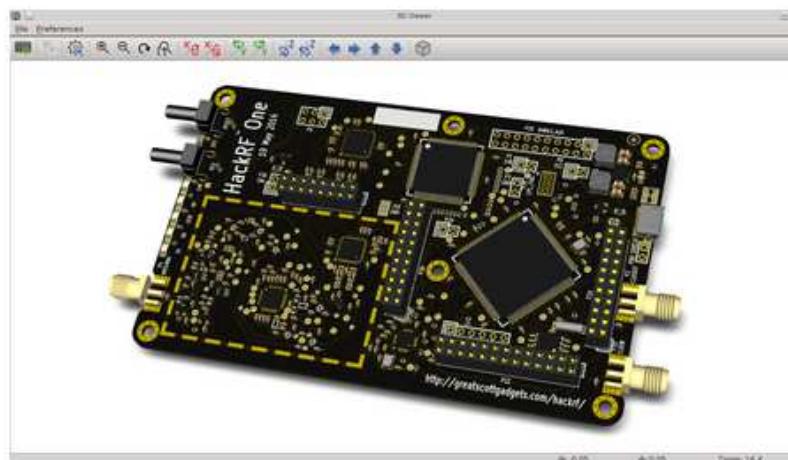
## OPEN-SOURCE SOFTWARE:

- ExpressPCB: <http://www.expresspcb.com/>
- Fritzing: <http://fritzing.org/>
- KiCAD: [http://www.lis.inpg.fr/realise\\_au\\_lis/kicad/](http://www.lis.inpg.fr/realise_au_lis/kicad/)
- TCI:<http://b.urbani.free.fr/pagetci/tci.htm>



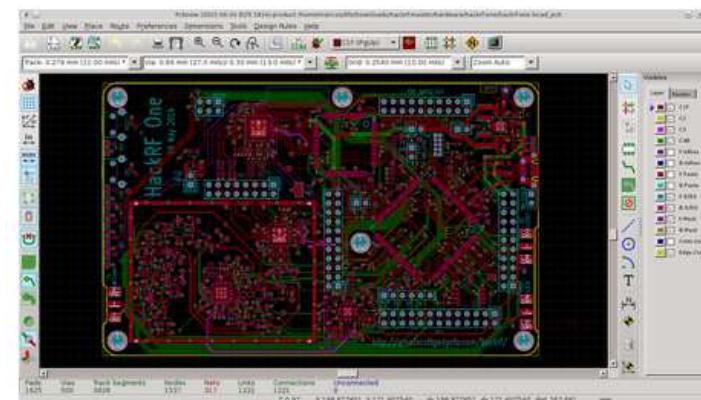
## PCB Layout

Make professional PCB layouts with up to 32 copper layers. KiCad now has a push and shove router which is capable of routing differential pairs and interactively tuning trace lengths.



## Schematic Capture

With the schematic editor you can create your design without limit; there are no paywalls to unlock features. An official library for schematic symbols and a built-in schematic symbol editor help you get started quickly with your designs.

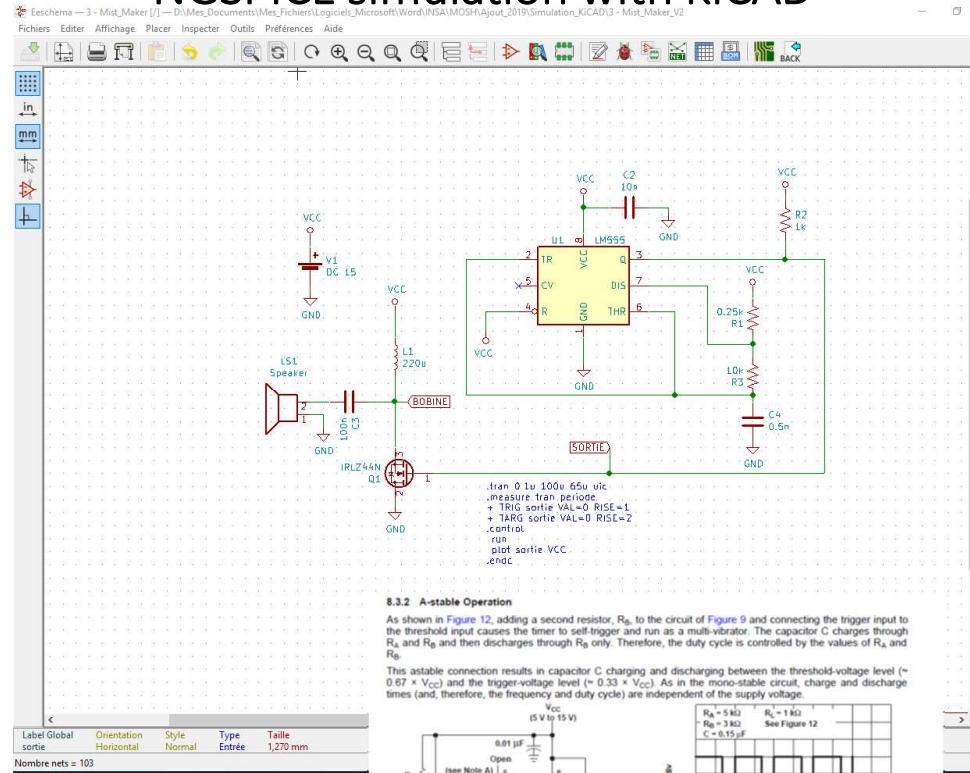


## 3D Viewer

KiCad includes a 3D viewer which you can use to inspect your design in an interactive canvas. You can rotate and pan around to inspect details that are difficult to inspect on a 2D view. Multiple rendering options allow you to modify the aesthetic appearance of the board or to hide and show features for easier inspection.

**< RETOUR**

## NGSPICE simulation with KiCAD



### 8.3.2 A-stable Operation

As shown in Figure 9, adding a second resistor,  $R_6$ , to the circuit of Figure 9 and connecting the trigger input to the threshold input causes the timer to self-trigger and run as a multi-vibrator. The capacitor C charges through  $R_4$  and  $R_6$  and then discharges through  $R_6$  only. Therefore, the duty cycle is controlled by the values of  $R_4$  and  $R_6$ .

This astable connection results in capacitor C charging and discharging between the threshold-voltage level ( $\approx 0.67 \times V_{CC}$ ) and the trigger-voltage level ( $\approx 0.33 \times V_{CC}$ ). As in the mono-stable circuit, charge and discharge times (and, therefore, the frequency and duty cycle) are independent of the supply voltage.

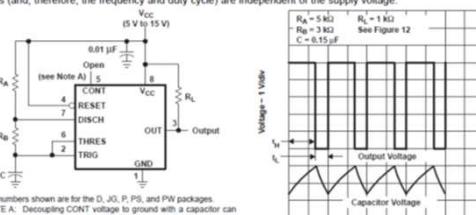


Figure 12. Circuit for Astable Operation

Figure 12 shows typical waveforms generated during astable operation. The output high-level duration  $t_H$  and low-level duration  $t_L$  can be calculated as follows:

$$t_H = 0.693(R_A + R_B)C \quad (1)$$

$$t_L = 0.693(R_B)C \quad (2)$$

Other useful relationships are shown below:

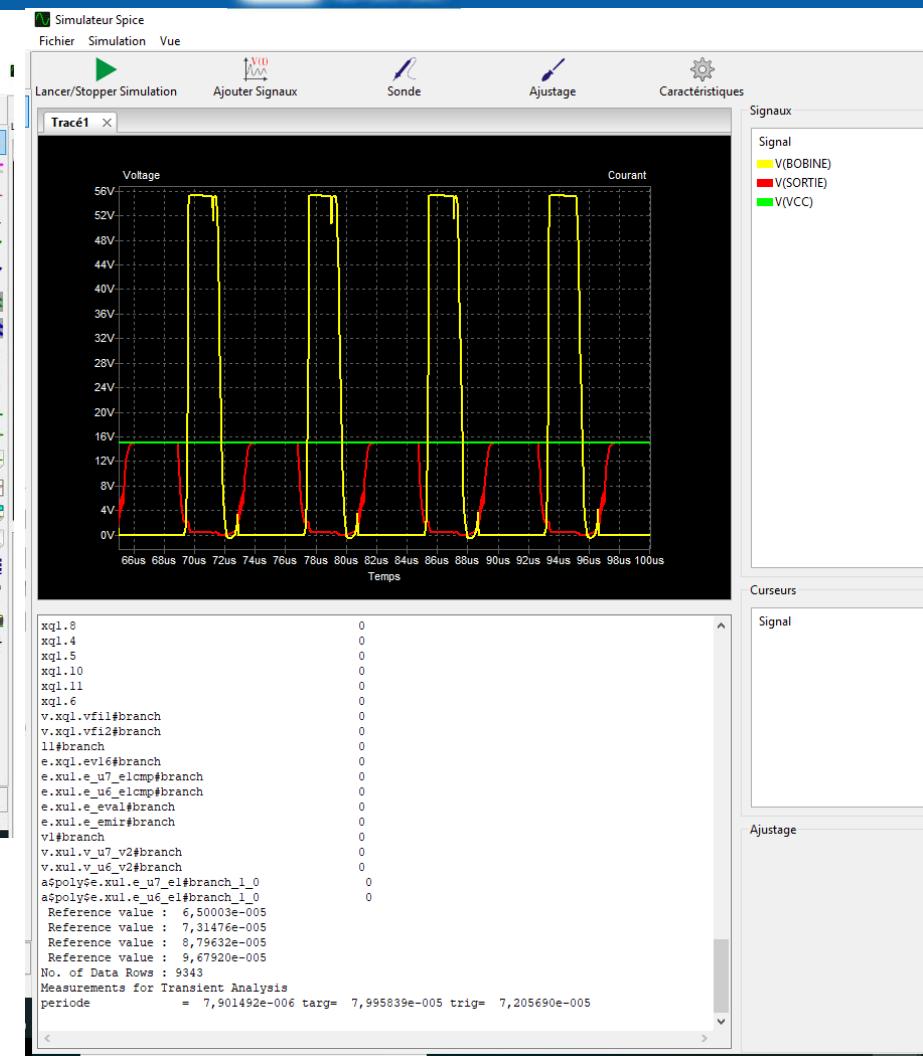
$$\text{period} = t_H + t_L = 0.693(R_A + 2R_B)C \quad (3)$$

$$\text{frequency} = \frac{1.44}{(R_A + 2R_B)C} \quad (4)$$

$$\text{Output driver duty cycle} = \frac{t_H}{t_H + t_L} = \frac{R_A}{R_A + 2R_B} \quad (5)$$

$$\text{Output waveform duty cycle} = \frac{t_H}{t_H + t_L} = 1 - \frac{R_B}{R_A + 2R_B} \quad (6)$$

$$\text{Low-to-high ratio} = \frac{t_H}{t_L} = \frac{R_A}{R_A + R_B} \quad (7)$$



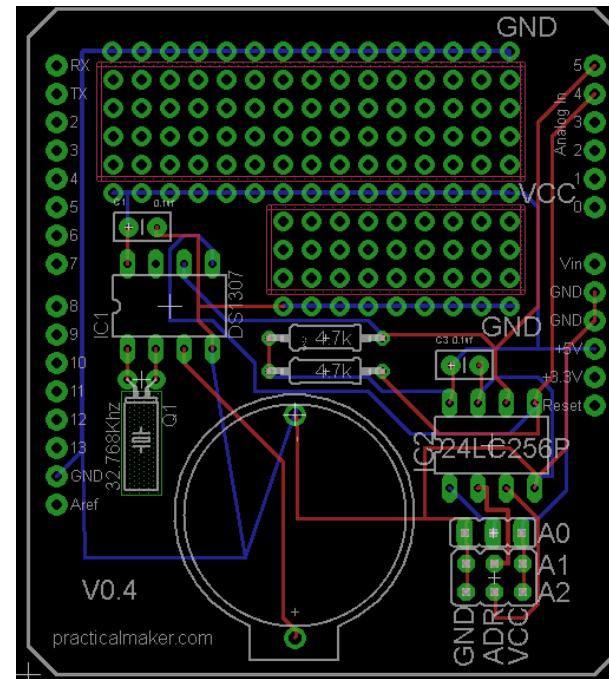
# PRECAUTIONS: SHIELD STANDARDISATION

Design a shield requires a standardization that will make the whole system more modular and open up new fields of opportunity for the Arduino.

The idea is simple: when designing a shield, some logic must be respected

The following are guidelines for the assignment of pins in the design of a shield.

Pin	Reserved For
Analog 0	User Selectable Input
Analog 1	User Selectable Input
Analog 2	User Selectable Input
Analog 3	User Selectable Input
Analog 4	I2C
Analog 5	I2C
Digital 0	RX
Digital 1	TX
Digital 2	OneWire
Digital 3	PWM / Digital I/O / I2C Interrupt
Digital 4	CS Select
Digital 5	PWM / Digital I/O
Digital 6	PWM / Digital I/O
Digital 7	CS Select
Digital 8	CS Select
Digital 9	PWM / Digital I/O
Digital 10	PWM / Digital I/O
Digital 11	SPI
Digital 12	SPI
Digital 13	SPI



## CREATE YOUR SHIELDS :

<http://aaroneiche.com/2010/06/24/a-beginners-guide-to-making-an-arduino-shield-pcb/>

<http://www.krisbarrett.com/2008/09/03/make-a-custom-arduino-shield/>

## SHIELDS STANDARDISATION :

<http://www.practicalarduino.com/news/id/661>

<http://www.practicalmaker.com/blog/arduino-shield-design-standards>

## IV-1 TO GO FURTHER : PROCESSING

IV-1 TO GO FURTHER :  
PROCESSING

# PROCESSING



"PROCESSING" makes the JAVA programming as fun and easy as the AVR with the Arduino programming

- It is used to connect the Arduino to the PC
- It is completely OPEN-SOURCE like the Arduino

Processing sketches are very similar to the Arduino

- `setup()` – sketch establishment
- `draw()` - like `loop()` in Arduino (called permanently)
- other features are based on libraries

A screenshot of the Processing 1.5.1 software interface. The window title is "Bounce | Processing 1.5.1". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has standard icons for play, stop, and save. The sketch name "Bounce" is selected in the dropdown. The code area contains the following Java-like pseudocode:

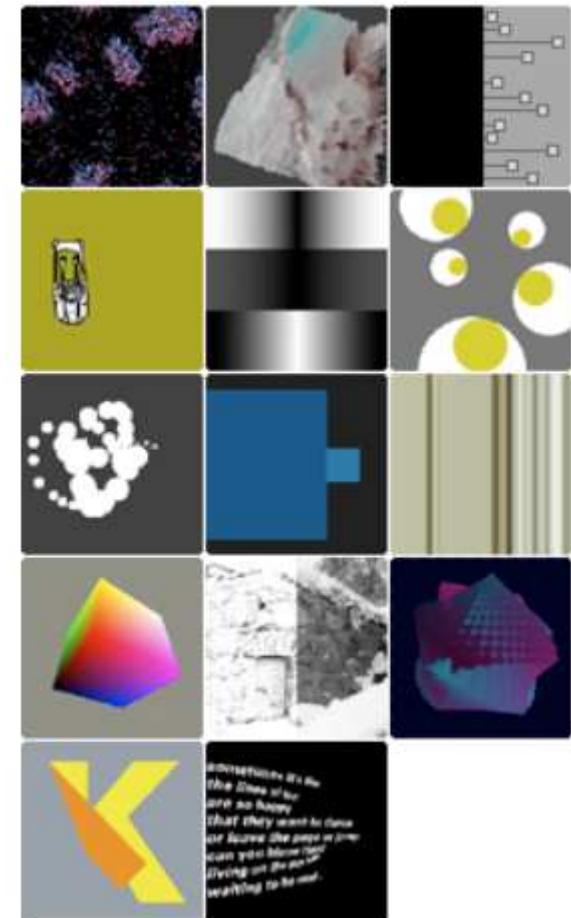
```
background(102);

// Update the position of the shape
xpos = xpos + ( xspeed * xdirection );
ypos = ypos + ( yspeed * ydirection );

// Test to see if the shape exceeds the boundaries of the screen
// If it does, reverse its direction by multiplying by -1
if (xpos > width-size || xpos < 0) {
    xdirection *= -1;
}
if (ypos > height-size || ypos < 0) {
    ydirection *= -1;
}

// Draw the shape
ellipse(xpos+size/2, ypos+size/2, size, size);
```

The status bar at the bottom shows the number 37.



<http://processing.org/learning/gettingstarted/>

[http://wiki.processing.org/w/Main\\_Page](http://wiki.processing.org/w/Main_Page)

[http://www.ecole-art-aix.fr/rubrique.php?id\\_rubrique=81](http://www.ecole-art-aix.fr/rubrique.php?id_rubrique=81)

# PROCESSING

Download and Install PROCESSING:

<http://processing.org/download/>

```
/** Bounce.
 * When the shape hits the edge of the window, it reverses its direction.
 */
int size = 60; // Width of the shape
float xpos, ypos; // Starting position of shape
float xspeed = 2.8; // Speed of the shape
float yspeed = 2.2; // Speed of the shape
int xdirection = 1; // Left or Right
int ydirection = 1; // Top to Bottom
void setup()
{
    size(640, 200);
    noStroke();
    frameRate(30);
    smooth();
    // Set the starting position of the shape
    xpos = width/2;
    ypos = height/2;
}

void draw()
{
    background(102);

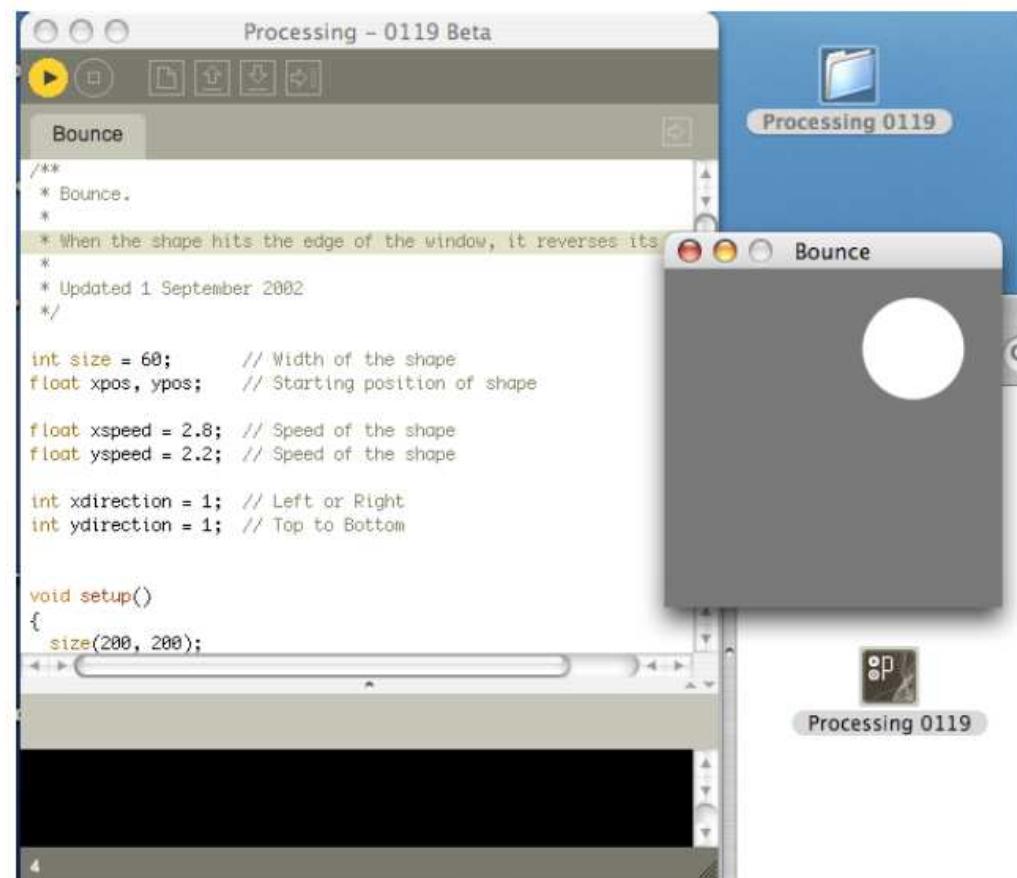
    // Update the position of the shape
    xpos = xpos + ( xspeed * xdirection );
    ypos = ypos + ( yspeed * ydirection );

    // Test to see if the shape exceeds the boundaries of the screen
    // If it does, reverse its direction by multiplying by -1
    if (xpos > width-size || xpos < 0) {
        xdirection *= -1;
    }
    if (ypos > height-size || ypos < 0) {
        ydirection *= -1;
    }
    // Draw the shape
    ellipse(xpos+size/2, ypos+size/2, size, size);
}
```

1 – Load TOPICS/MOTION/BOUNCE example

2 – Press RUN

=> you just created a JAVA applet



# PROCESSING & ARDUINO: SERIAL COMMUNICATION

- Processing and Arduino talking both the "series"
- but, one program possible by serial port:  
thus, close the Arduino monitor series when you connect via Processing, and vice versa.

- Processing has a library "Series" to talk to Arduino:

`port = new Serial(this,"mon_nom_port",19200)`

`port.read(), port.write(), port.available(), etc.`

`serialEvent() { }`

4 steps:

1 - load the library

2. Set the name of the port:

`portname`

3. Open port: `port = new Serial (...);`

4. read/write on port

```
import processing.serial.*;

String portname = "/dev/tty.usbserial-A4001qa8"; // or "COM8"
Serial port; // Create object from Serial class

int val=100; // Data received from the serial port, with an initial value

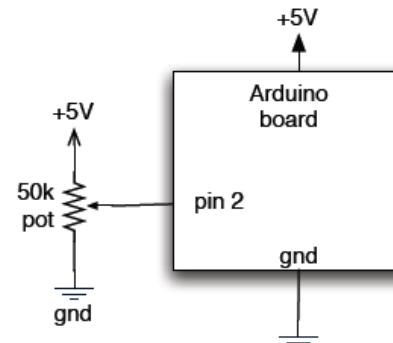
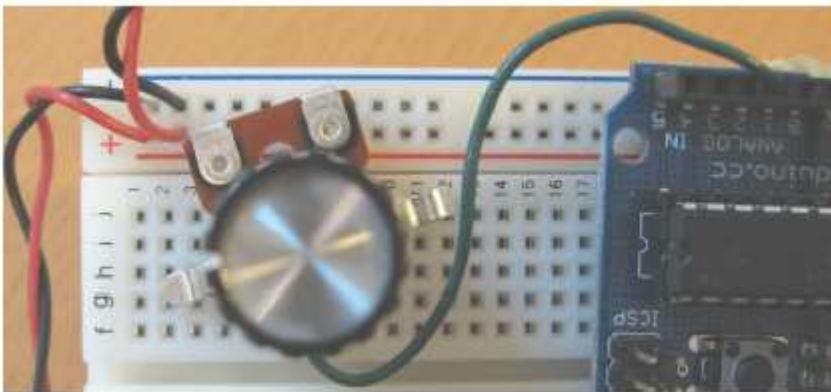
void setup()
{
    // Open the port the board is connected to
    port = new Serial(this, portname, 19200);
}

void draw()
{
    if (port.available() > 0) { // If data is available,
        val = port.read(); // read it and store it in val
    }
}
```

Make sure you put the good name of serial port!

# ARDUINO SPEAKS PROCESSING

Create an Arduino program that reads the value of a potentiometer and sends it to Processing in the form of a number between 0 and 255:



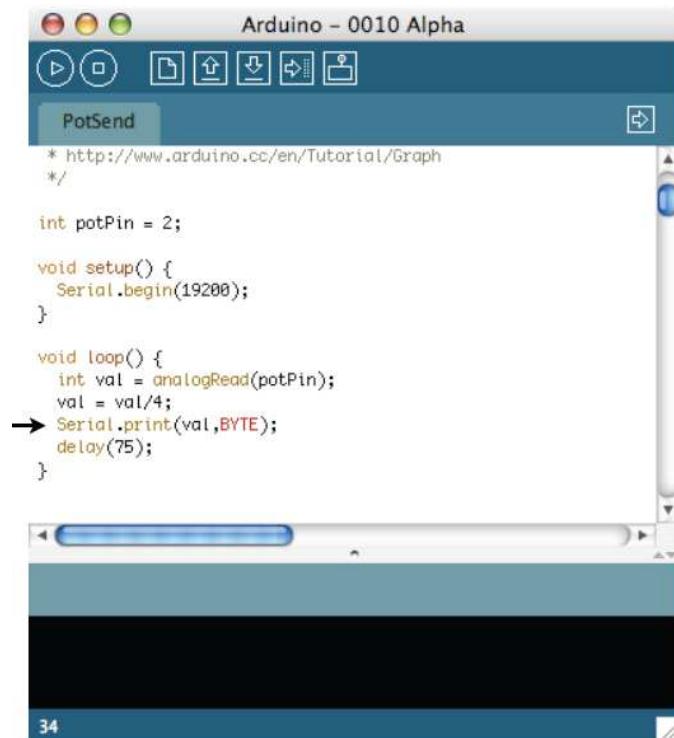
## Arduino Sketch:

```
/* PotSend
 * Send analog values from pots or similar over the serial port
 * Also see: http://www.arduino.cc/en/Tutorial/Graph
 */
int potPin = 2;

void setup() {
  Serial.begin(19200);
}

void loop() {
  int val = analogRead(potPin);
  val = val/4;
  Serial.print(val,BYTE);
  delay(100);
}
```

Note: do not send the value as ASCII text, but rather as a binary number (the BYTE are easier to handle in Processing)



34

## TP4-D

# ARDUINO & PROCESSING: PONG

```

/** Collision (Pong).
 * Move the mouse up and down to move the
paddle.
 * Modified to use Serial port by Tod E. Kurt, 2007
 * Updated 13 January 2003 by K Pfeiffer
 */

import processing.serial.*;

String portname = "/dev/tty.usbserial-A4001qa8";
// or "COM8"
Serial port; // Create object from Serial class

// Global variables for the ball
float ball_x;
float ball_y;
float ball_dir = 1;
float ball_size = 5; // Radius
float dy = 0; // Direction

// Global variables for the paddle
int paddle_width = 5;
int paddle_height = 20;
int paddle_pos; // new position
int paddle_ppos; // last position
int dist_wall = 15;

void setup()
{
    size(255, 255);
    rectMode(CENTER_RADIUS);
    ellipseMode(CENTER_RADIUS);
    noStroke();
    smooth();
    ball_y = height/2;
    ball_x = 1;

    // Open the port that the board is connected to
    // and use the same speed (19200 bps)
    port = new Serial(this, portname, 19200);
}

void draw()
{
    background(51);
    ball_x += ball_dir * 1.0;
    ball_y += dy;
    if(ball_x > width+ball_size) {
        ball_x = -width/2 - ball_size;
        ball_y = random(0, height);
        dy = 0;
    }
    if(port.available() > 0) { // If data is available,
        paddle_ppos = paddle_pos; // save old position
        paddle_pos = port.read(); // read it and store it as the new pos
        paddle_pos = paddle_pos *2; //pour avoir les valeurs de 0 à 255
        println(paddle_pos);
    }
    // Constrain paddle to screen
    float paddle_y = constrain(paddle_pos, paddle_height, height-paddle_height);

    // Test to see if the ball is touching the paddle
    float py = width-dist_wall-paddle_width-ball_size;
    if(ball_x == py
        && ball_y > paddle_y - paddle_height - ball_size
        && ball_y < paddle_y + paddle_height + ball_size) {
        ball_dir *= -1;
        if(paddle_pos != paddle_ppos) {
            dy = (paddle_pos - paddle_ppos)/2.0;
            if(dy > 5) { dy = 5; }
            if(dy < -5) { dy = -5; }
        }
    }
    // If ball hits paddle or back wall, reverse direction
    if(ball_x < ball_size && ball_dir == -1) {
        ball_dir *= -1;
    }
    // If the ball is touching top or bottom edge, reverse direction
    if(ball_y > height-ball_size) {
        dy = dy * -1;
    }
    if(ball_y < ball_size) {
        dy = dy * -1;
    }
}

```

<http://moodle.insa-toulouse.fr/course/view.php?id=494>

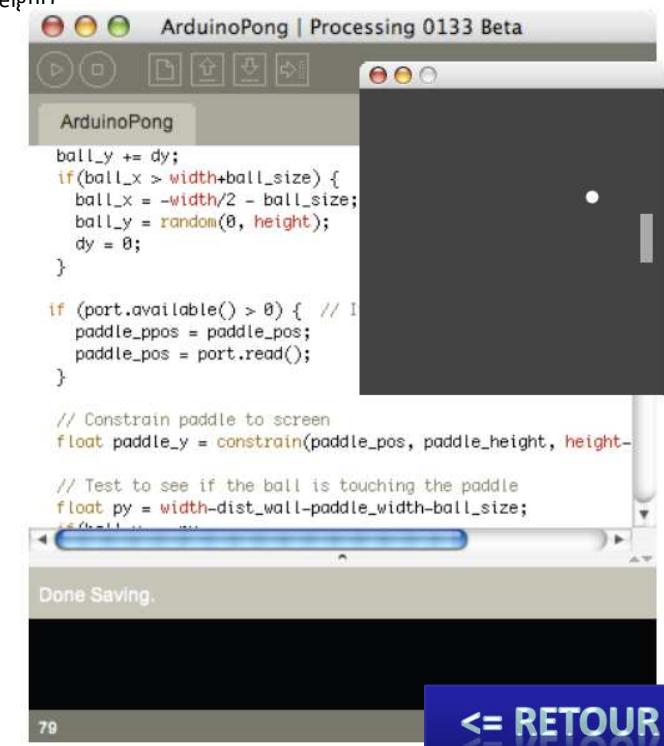
```

    // Draw ball
    fill(255);
    ellipse(ball_x, ball_y, ball_size, ball_size);

    // Draw the paddle
    fill(153);
    rect(width-dist_wall, paddle_y, paddle_width, paddle_height);
}


```

**The basic game of pong**  
**the potentiometer controls the position**  
**of the racket**  
**=> add a potentiometer and a few lines of**  
**code and you will have a two-player game**



# ARDUINO & PROCESSING: Wii Nunchuk & CUBE 3D (EXAMPLE 2)

Sketch ARDUINO:

```
#include <math.h>

#include "Wire.h"
#include "WiiChuck.h"
//#include "nunchuck_funcs.h"

#define MAXANGLE 90
#define MINANGLE -90

WiiChuck chuck = WiiChuck();
int angleStart, currentAngle;
int tillerStart = 0;
double angle;

void setup() {
  //nunchuck_init();
  Serial.begin(115200);
  chuck.begin();
  chuck.update();
  //chuck.calibrateJoy();
}
```

```
void loop() {
  delay(20);
  chuck.update();

  Serial.print(chuck.readRoll());
  Serial.print(", ");
  Serial.print(chuck.readPitch());
  Serial.print(", ");

  Serial.print((int)chuck.readAccelX());
  Serial.print(", ");
  Serial.print((int)chuck.readAccelY());
  Serial.print(", ");

  Serial.print((int)chuck.readAccelZ());
  Serial.println();
}
```

<http://arduino.cc/playground/Main/WiiChuckClass>

# ARDUINO & PROCESSING: Wii Nunchuk & CUBE 3D (EXAMPLE 2)

## Call WiiChuck.h:

/\* Nunchuck -- Use a Wii Nunchuck \* Tim Hirzel http://www.growdown.com \*  
notes on Wii Nunchuck Behavior.  
This library provides an improved derivation of rotation angles from the nunchuck  
accelerometer data. The biggest difference over existing libraries (that I know of) is the  
full 360 degrees of Roll data  
from the combination of the x and z axis accelerometer data using the math library  
atan2.  
It is accurate with 360 degrees of roll (rotation around axis coming out of the c button,  
the front of the wii), and about 180 degrees of pitch (rotation about the axis coming out  
of the side of the wii). (read more below)

In terms of mapping the wii position to angles, its important to note that while the  
Nunchuck  
sense Pitch, and Roll, it does not sense Yaw, or the compass direction. This creates an  
important  
disparity where the nunchuck only works within one hemisphere. At a result, when the  
pitch values are less than about 10, and greater than about 170, the Roll data gets very  
unstable. essentially, the roll data flips over 180 degrees very quickly. To understand  
this property better, rotate the wii around the axis of the joystick. You see the sensor  
data stays constant (with noise). Because of this, it cant know the difference between  
arriving upside via 180 degree Roll, or 180 degree pitch. It just assumes its always 180  
roll.\*

\* This file is an adaptation of the code by these authors:  
\* Tod E. Kurt, http://todbot.com/blog/  
\* The Wii Nunchuck reading code is taken from Windmeadow Labs  
\* http://www.windmeadow.com/node/42

```
/*
#ifndef WiiChuck_h
#define WiiChuck_h
#include "WProgram.h"
#include <Wire.h>
#include <math.h>
```

// these may need to be adjusted for each nunchuck for calibration  
#define ZEROX 510  
#define ZEROY 490  
#define ZEROZ 460  
#define RADIUS 210 // probably pretty universal

#define DEFAULT\_ZERO\_JOY\_X 124  
#define DEFAULT\_ZERO\_JOY\_Y 132

```
class WiiChuck {
private:
    byte cnt;
    uint8_t status[6]; // array to store wiichuck output
    byte averageCounter;
    //int accelArray[3][AVERAGE_N]; // X,Y,Z
    int i;
    int total;
    uint8_t zeroJoyX; // these are about where mine are
    uint8_t zeroJoyY; // use calibrateJoy when the stick is at zero to correct
    int lastJoyX;
    int lastJoyY;
    int angles[3];
```

```
    boolean lastZ, lastC;
public:
    byte joyX;
    byte joyY;
    boolean buttonZ;
    boolean buttonC;
```

```
void begin()
{
    Wire.begin();
    cnt = 0;
    averageCounter = 0;
    // instead of the common 0x40 -> 0x00 initialization, we
    // use 0xF0 -> 0x55 followed by 0xFB -> 0x00.
    // this lets us use 3rd party nunchucks (like cheap $4 ebay ones)
    // while still letting us use official ones.
    // only side effect is that we no longer need to decode bytes in _nunchuk_decode_byte
    // see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1264805255
    //
    Wire.beginTransmission(0x52); // device address
    Wire.send(0xF0);
    Wire.send(0x55);
    Wire.endTransmission();
    delay(1);
    Wire.beginTransmission(0x52);
    Wire.send(0xFB);
    Wire.endTransmission();
    update();
    for (i = 0; i<3;i++) {
        angles[i] = 0;
    }
    zeroJoyX = DEFAULT_ZERO_JOY_X;
    zeroJoyY = DEFAULT_ZERO_JOY_Y;
}

void calibrateJoy()
{
    zeroJoyX = joyX;
    zeroJoyY = joyY;
}

void update()
{
    Wire.requestFrom(0x52, 6); // request data from nunchuck
    while (Wire.available ()) {
        // receive byte as an integer
        status[cnt] = _nunchuk_decode_byte (Wire.receive()); //
        cnt++;
    }
    if (cnt > 5) {
        lastZ = buttonZ;
        lastC = buttonC;
        lastJoyX = readJoyX();
        lastJoyY = readJoyY();
        //averageCounter++;
        //if (averageCounter >= AVERAGE_N)
        //    averageCounter = 0;
        cnt = 0;
        joyX = (status[0]);
        joyY = (status[1]);
        for (i = 0; i < 3; i++)
            //accelArray[i][averageCounter] = ((int)status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2) ) >> ((i+1)*2));
            angles[i] = (status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2) ) >> ((i+1)*2));
            //accelYArray[averageCounter] = ((int)status[3] << 2) + ((status[5] & B00110000) >> 4);
            //accelZArray[averageCounter] = ((int)status[4] << 2) + ((status[5] & B1100000) >> 6);
            buttonZ = !( status[5] & B00000001);
            buttonC = !(status[5] & B00000010) >> 1;
            _send_zero(); // send the request for next bytes
    }
}

// UNCOMMENT FOR DEBUGGING
//byte * getStatus() {
//    return status; //}
}

float readAccelX() {
    // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
    return (float)angles[0] - ZEROY; //}
float readAccelY() {
    // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
    return (float)angles[1] - ZEROY; //}
float readAccelZ() {
    // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
    return (float)angles[2] - ZEROZ; //}

boolean zPressed() {
    return (buttonZ && ! lastZ); //}

boolean cPressed() {
    return (buttonC && ! lastC); //}

// for using the joystick like a directional button
boolean rightJoy(int thresh=60) {
    return (readJoyX() > thresh and lastJoyX <= thresh); //}

// for using the joystick like a directional button
boolean leftJoy(int thresh=60) {
    return (readJoyX() < -thresh and lastJoyX >= -thresh); //}

int readJoyX() {
    return (int)joyX - zeroJoyX; //}

int readJoyY() {
    return (int)joyY - zeroJoyY; //}

// R, the radius, generally hovers around 210 (at least it does with mine)
// int R() {
//    return sqrt(readAccelX() * readAccelX() +readAccelY() * readAccelY() + readAccelZ() * readAccelZ()); //}

// returns roll degrees
int readRoll() {
    return (int)(atan2(readAccelX(),readAccelZ()) / M_PI * 180.0); //}

// returns pitch in degrees
int readPitch() {
    return (int)(acos(readAccelY() / RADIUS) / M_PI * 180.0); // optionally swap 'RADIUS' for 'R()'

}

private:
    byte _nunchuk_decode_byte (byte x)
    {
        //decode is only necessary with certain initializations
        //x = (x ^ 0x17) + 0x17;
        return x; //}
}

void _send_zero()
{
    Wire.beginTransmission (0x52); // transmit to device 0x52
    Wire.send (0x00); // sends one byte
    Wire.endTransmission (); // stop transmitting
}

};

#endif
```

# ARDUINO & PROCESSING: Wii Nunchuk & CUBE 3D (EXAMPLE 2)

## Sketch PROCESSING:

```
/*
 * Wii controlled
 * RGB Cube.
 * The three primary colors of the additive color model are red, green, and blue.
 * This RGB color cube displays smooth transitions between these colors.
 */

float xmag, ymag = 0;
float newXmag, newYmag = 0;
int sensorCount = 5; // number of values to expect

import processing.serial.*;
Serial myPort; // The serial port
int BAUDRATE = 115200;
char DELIM = ','; // the delimiter for parsing incoming data

void setup()
{
    size(200, 200, P3D);
    noStroke();
    colorMode(RGB, 1);
    // Affiche la liste des ports série
    println(Serial.list());
    //myPort = new Serial(this, Serial.list()[0], BAUDRATE);
    //affecte la bonne valeur au tableau Serial.list
    myPort = new Serial(this, Serial.list()[1], BAUDRATE);
    // clear the serial buffer:
    myPort.clear();
}
float x, z;

void draw()
{
    background(0.5, 0.5, 0.45);
    pushMatrix();
    translate(width/2, height/2, -30);
    newXmag = mouseX/float(width) * TWO_PI;
    newYmag = mouseY/float(height) * TWO_PI;
    float diff = xmag-newXmag;
    if (abs(diff) > 0.01) {
        xmag -= diff/4.0;
    }

    diff = ymag-newYmag;
    if (abs(diff) > 0.01) {
        ymag -= diff/4.0;
    }
}
```

```
// if ((sensorValues[1] > 15) && (sensorValues[1] < 165)) {
//   z = sensorValues[0] / 180 * PI;
//   x = sensorValues[1] / 180 * PI;
// }

rotateZ(z);
rotateX(x);
scale(50);
beginShape(QUADS);

fill(0, 1, 1);
vertex(-1, 1, 1);
fill(1, 1, 1);
vertex(1, 1, 1);
fill(1, 0, 1);
vertex(1, -1, 1);
fill(0, 0, 1);
vertex(-1, -1, 1);

fill(1, 1, 1);
vertex(1, 1, 1);
fill(1, 0, 0);
vertex(1, -1, 1);
fill(1, 0, 0);
vertex(1, -1, -1);
fill(1, 0, 1);
vertex(1, -1, 1);

fill(1, 0, 0);
vertex(1, 1, -1);
fill(0, 1, 0);
vertex(-1, 1, -1);
fill(0, 0, 0);
vertex(-1, -1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);

fill(0, 1, 0);
vertex(-1, 1, -1);
fill(0, 1, 1);
vertex(-1, 1, 1);
fill(0, 0, 1);
vertex(-1, -1, 1);
fill(0, 0, 0);
vertex(-1, -1, -1);

fill(0, 1, 0);
vertex(-1, 1, -1);
fill(1, 1, 0);
vertex(1, 1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);

fill(0, 0, 0);
vertex(-1, -1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);
fill(1, 0, 1);
vertex(1, 1, -1);
fill(0, 0, 1);
vertex(-1, -1, 1);

fill(0, 0, 0);
vertex(-1, -1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);
fill(1, 0, 1);
vertex(1, 1, -1);
fill(0, 0, 1);
vertex(-1, -1, 1);

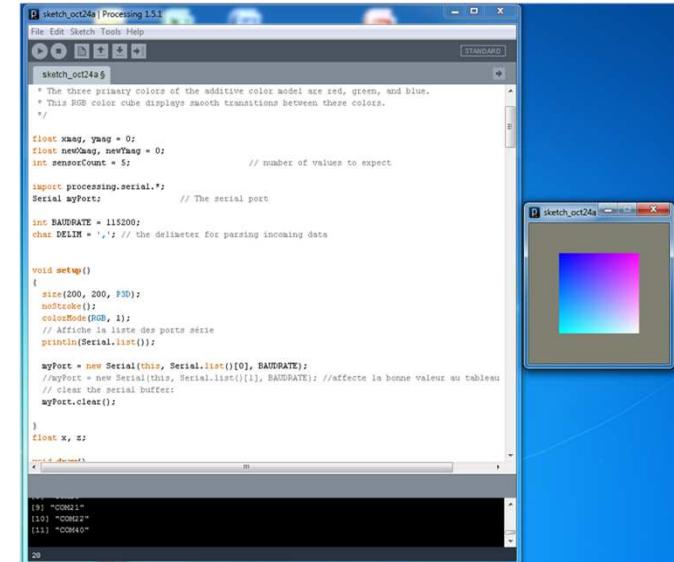
endShape();

popMatrix();
}

float[] sensorValues = new float[sensorCount]; // array to hold the incoming values

void serialEvent(Serial myPort) {
    // read incoming data until you get a newline:
    String serialString = myPort.readStringUntil('\n');
    // if the read data is a real string, parse it:

    if (serialString != null) {
        //println(serialString);
        //println(serialString.charAt(serialString.length()-3));
        //println(serialString.charAt(serialString.length()-2));
        // split it into substrings on the DELIM character:
        String[] numbers = split(serialString, DELIM);
        // convert each subastring into an int
        if (numbers.length == sensorCount) {
            for (int i = 0; i < numbers.length; i++) {
                // make sure you're only reading as many numbers as
                // you can fit in the array:
                if (i <= sensorCount) {
                    // trim off any whitespace from the substring:
                    numbers[i] = trim(numbers[i]);
                    sensorValues[i] = float(numbers[i]);
                }
            }
            // Things we don't handle in particular can get output to the text window
            print(serialString);
        }
    }
}
```

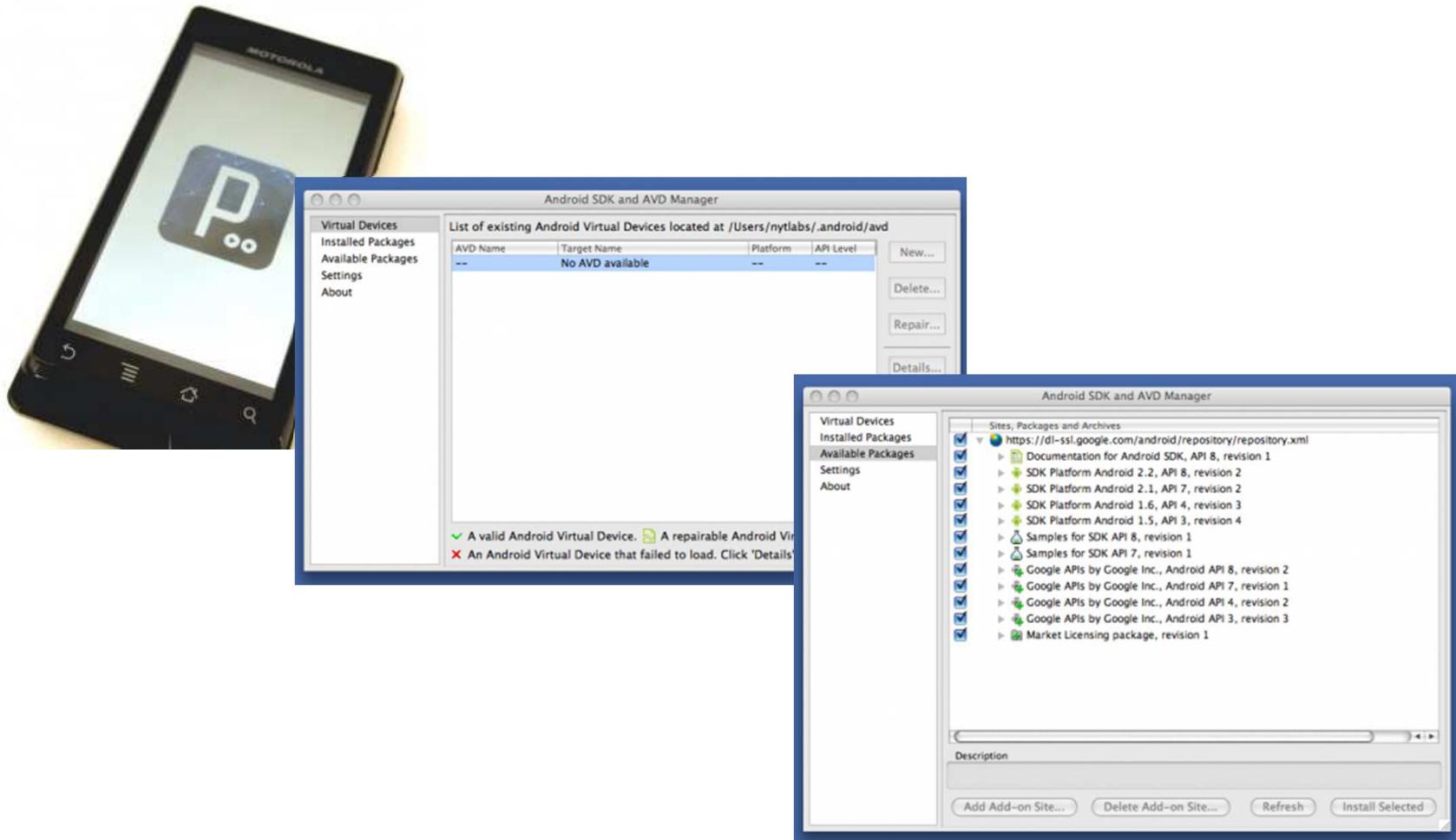


<http://arduino.cc/playground/Main/WiiChuckClass>

## IV-2 TO GO FURTHER: PROCESSING FOR ANDROID

# PROCESSING FOR ANDROID

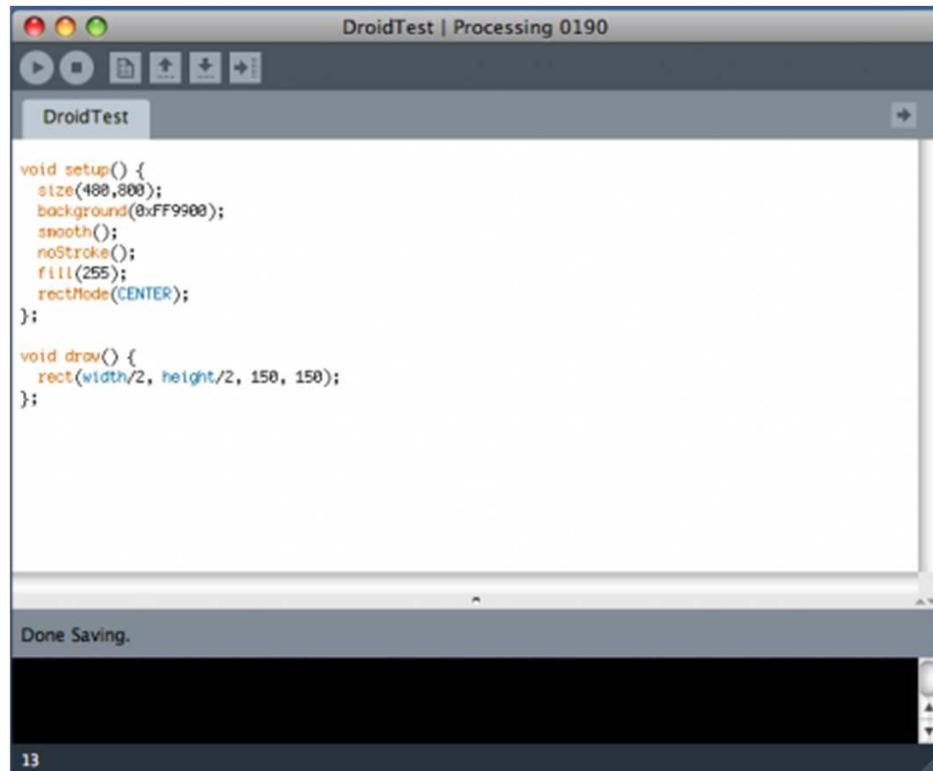
You can now use PROCESSING > V2.0 to quickly create applications for mobile devices that use the Android operating system.



Note: you don't need to have an Android device to make these applications, using the software emulator.

<http://processing.org/tutorials/android/>

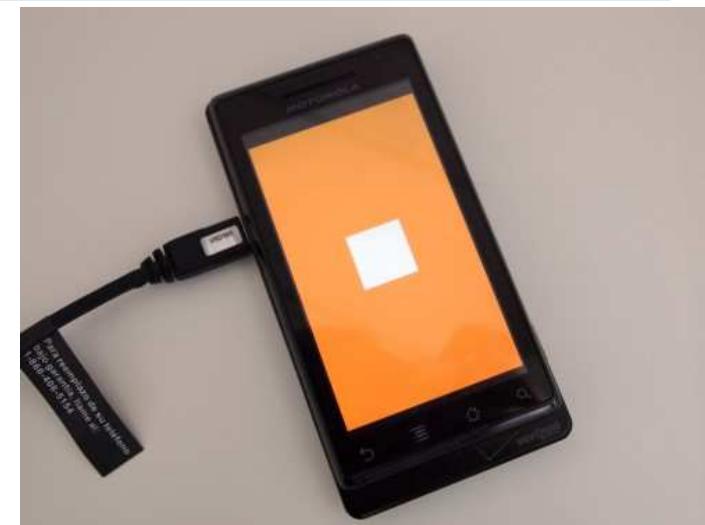
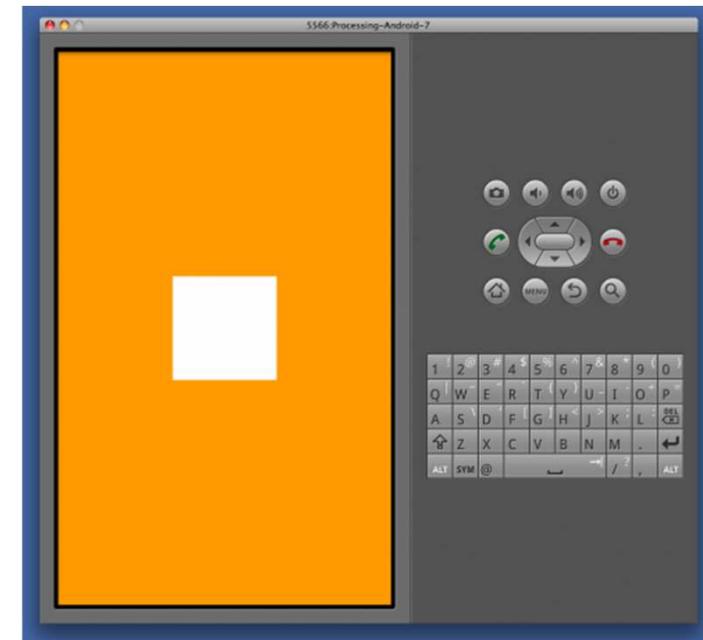
# PROCESSING FOR ANDROID: « HELLO WORLD »



DroidTest | Processing 0190

```
void setup() {  
    size(480,800);  
    background(0xFF9900);  
    smooth();  
    noStroke();  
    fill(255);  
    rectMode(CENTER);  
}  
  
void draw() {  
    rect(width/2, height/2, 150, 150);  
}  
  
Done Saving.
```

13



# PROCESSING FOR ANDROID: A BIT OF INTERACTIVITY

/\* World's simplest Android App!  
[blprnt@blprnt.com](mailto:blprnt@blprnt.com) Sept 25, 2010 \*/

```
//Build a container to hold the current rotation of the box
float boxRotation = 0;

void setup() {
    //Set the size of the screen (this is not really necessary in Android mode, but we'll do it anyway)
    size(480,800);
    //Turn on smoothing to make everything pretty.
    smooth();
    //Set the fill and stroke colour for the box and circle
    fill(255);
    stroke(255);
    //Tell the rectangles to draw from the center point (the default is the TL corner)
    rectMode(CENTER);
};

void draw() {
    //Set the background colour, which gets more red
    //as we move our finger down the screen.
    background(mouseY * (255.0/800), 100, 0);
    //Change our box rotation depending on how our finger has moved right-to-left
    boxRotation += (float) (pmouseX - mouseX)/100;

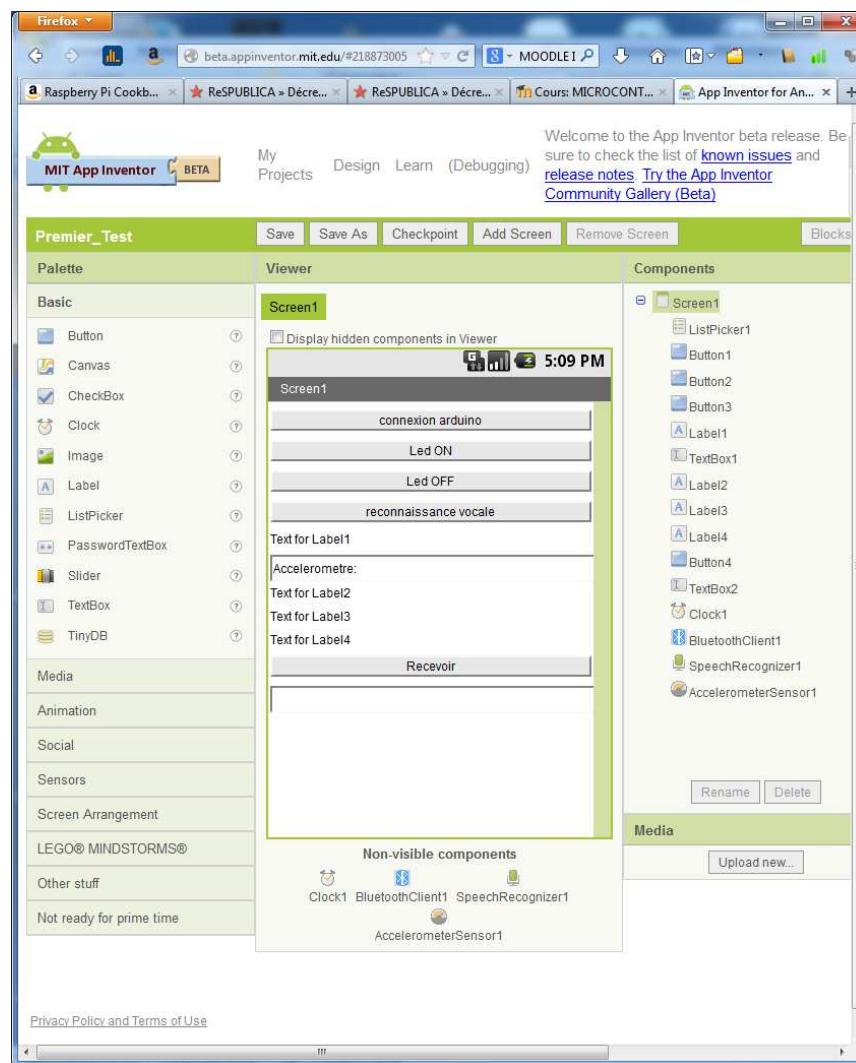
    //Draw the ball-and-stick
    line(width/2, height/2, mouseX, mouseY);
    ellipse(mouseX, mouseY, 40, 40);

    //Draw the box
    pushMatrix();
    translate(width/2, height/2);
    rotate(boxRotation);
    rect(0,0, 150, 150);
    popMatrix();
};
}
```



# TP5

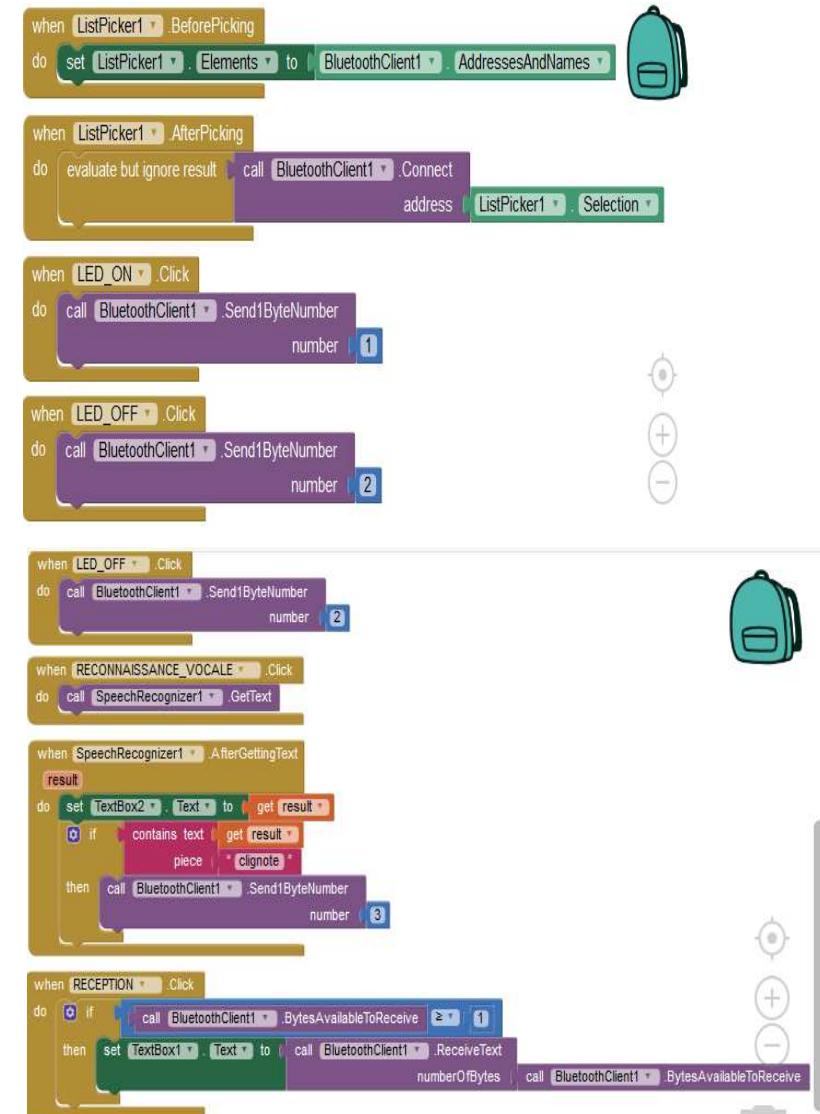
# OTHER WAY : MIT APP INVENTOR



<http://beta.appinventor.mit.edu/>

<http://appinventor.mit.edu/>

<http://ai2.appinventor.mit.edu/#5399440480272384>

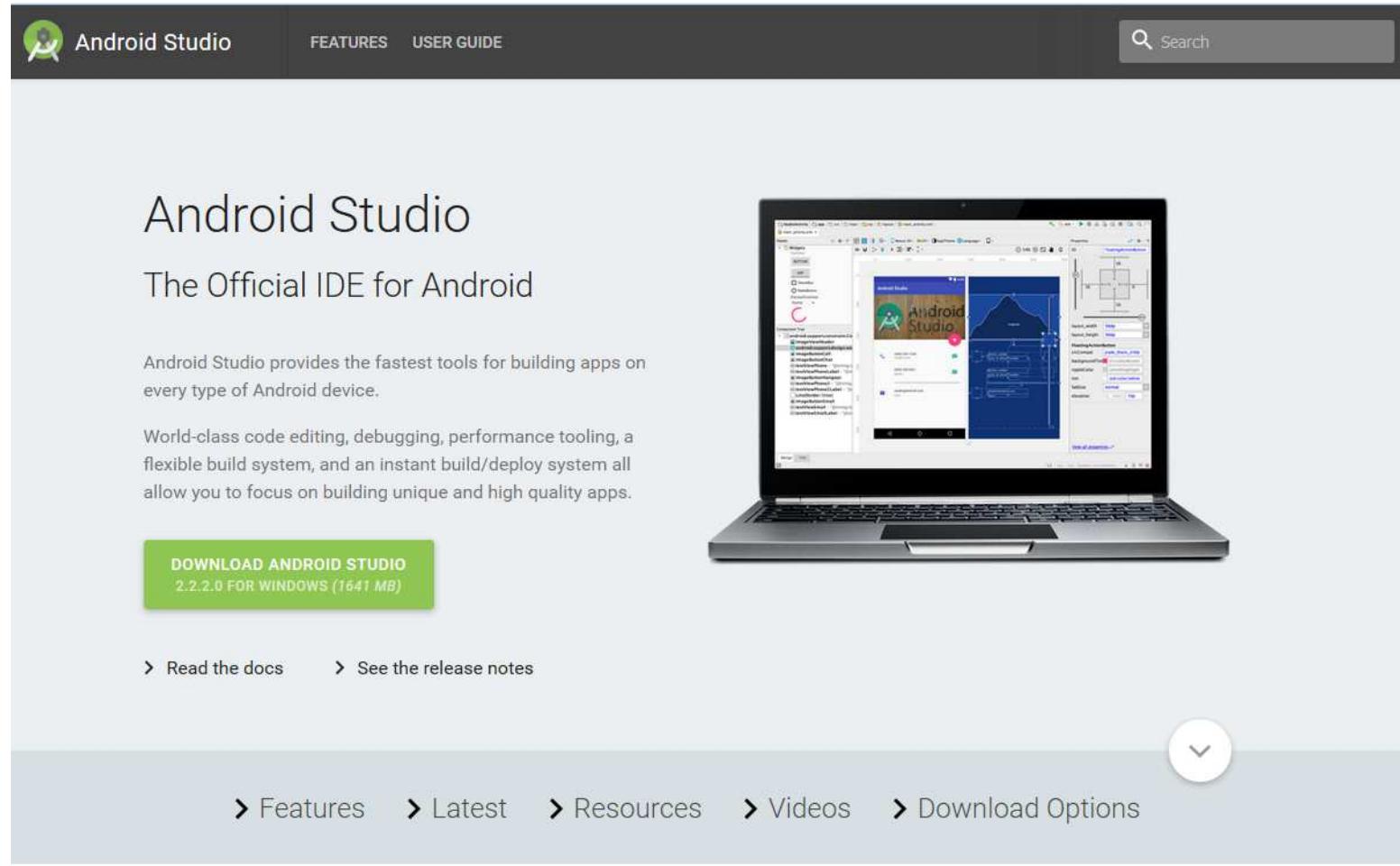


<= RETOUR



216

## OTHER WAY : ANDROID STUDIO



The screenshot shows the official website for Android Studio. At the top, there's a dark header with the Android Studio logo, a search bar, and links for "FEATURES" and "USER GUIDE". Below the header, the main title "Android Studio" is displayed in large letters, followed by the subtitle "The Official IDE for Android". A brief description states: "Android Studio provides the fastest tools for building apps on every type of Android device." Another paragraph highlights: "World-class code editing, debugging, performance tooling, a flexible build system, and an instant build/deploy system all allow you to focus on building unique and high quality apps." A prominent green button labeled "DOWNLOAD ANDROID STUDIO 2.2.0 FOR WINDOWS (1641 MB)" is visible. Below the download button, there are links to "Read the docs" and "See the release notes". At the bottom of the page, a navigation bar includes links for "Features", "Latest", "Resources", "Videos", and "Download Options".

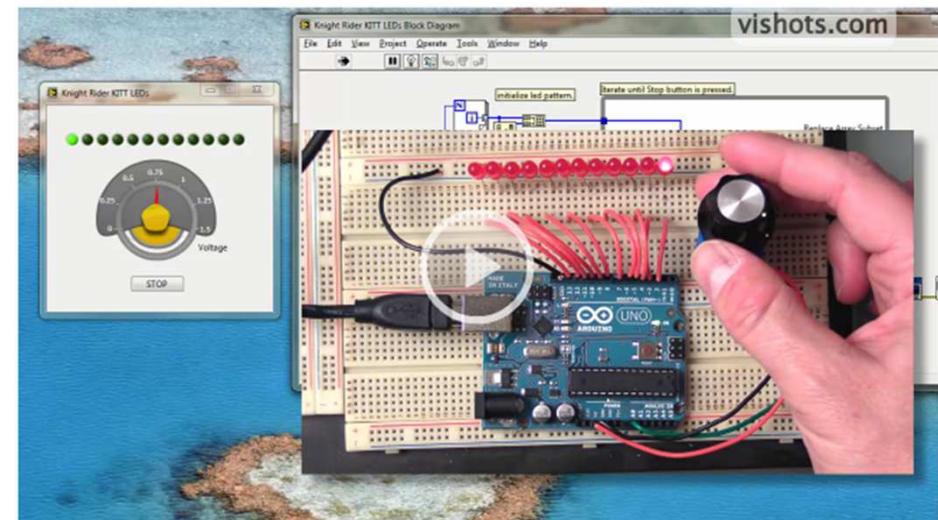
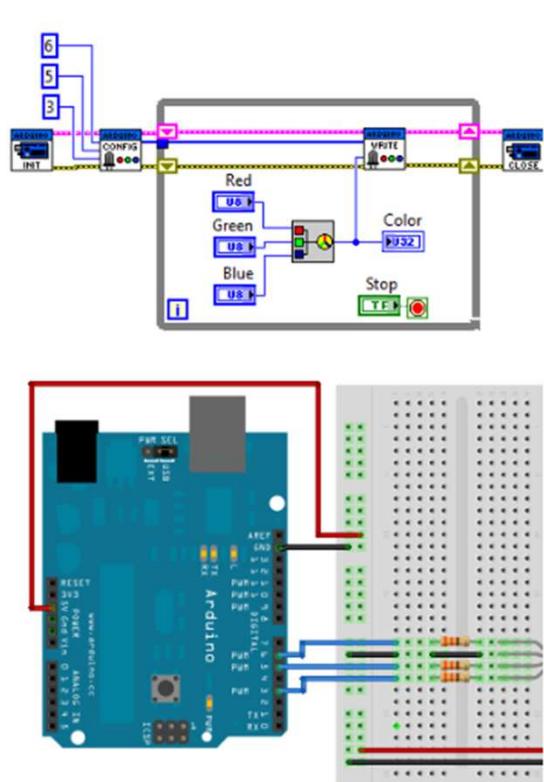
<https://developer.android.com/studio/index.html>

## IV-1 TO GO FURTHER : PROCESSING

## IV-2 TO GO FURTHER : LIFA

# LIFA - LabVIEW Interface for Arduino

LabVIEW for Arduino (LIFA) Toolkit interface allows developers to acquire data from the Arduino microcontroller and treat in the LabVIEW graphical programming environment and it's free! ;-)



Top 5 reasons that makes you more productive with LabVIEW when using Arduino:?

- 1 - Interact with your system via a graphical user interface.
- 2 - Optimize your design with intuitive graphical programming process.
- 3 - Improve your experience of debugging with interactive tools.
- 4 - Functions of simple implementation for complex tasks.
- 5 - Open API allows complete customization - customize your programs according to your application.

<https://decibel.ni.com/content/groups/labview-interface-for-arduino>  
<http://vishots.com/getting-started-with-the-labview-interface-for-arduino/>  
<http://innovelectronique.fr/2012/05/23/aduino-et-lifa-episode-2/>  
<http://innovelectronique.fr/2012/05/04/arduino-et-lifa-labview-interface-for-arduino/>

# LabVIEW Interface for Arduino - LIFA

<https://decibel.ni.com/content/groups/labview-interface-for-arduino?view=overview#/?tagSet=undefined>

The screenshot shows a Microsoft Internet Explorer browser window displaying the National Instruments (NI) Decibel community page for the 'LabVIEW Interface for Arduino' group. The URL in the address bar is <https://decibel.ni.com/content/groups/labview-interface-for-arduino?view=overview#/?tagSet=undefined>. The page title is 'LabVIEW Interface for Arduino - LIFA'. The main content area shows a list of documents and discussions. On the right side, there are sections for 'Actions', 'Documents populaires', and 'Meilleurs participants'.

**Actions:**

- Ajouter
- Fil du groupe

**Documents populaires:**

- LabVIEW Interface for Arduino Setup Procedure
- Ready to get started?
- LabVIEW Interface for Arduino FAQ
- Arduino Example: Analog Read
- Arduino Example: RGB LED

**Meilleurs participants:**

- Sammie\_K
- Benz
- Klaus\_F
- Mechanic
- hfh118
- Rymane

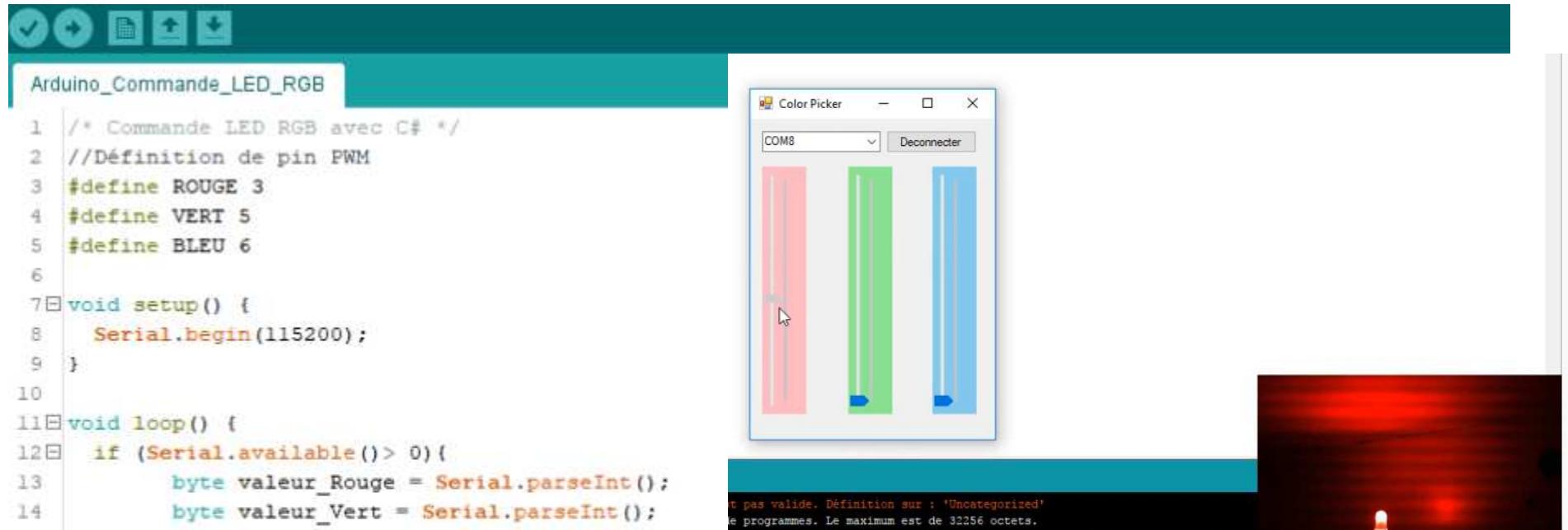
**Document List:**

Auteur	Sujet	Visualiser	Classification	Dernier modifié
Peter_R	Arduino Example: Simple LED	81	☆☆☆☆☆	29 avr. 2011 10:35 par Peter_R
rgparr	ITG3200	834	☆☆☆☆☆	23 avr. 2011 07:21 par rgparr
jasonD	Ball and Paddle Game	1 128	★★★★★	9 avr. 2011 10:11 par jasonD
Sammie_K	LabVIEW Interface for Arduino Packets	2 083	☆☆☆☆☆	27 juil. 2011 12:03 par Sammie_K
Nick_425	Arduino Push-Button: WAV Audio Sampler	1 353	☆☆☆☆☆	12 juil. 2011 13:49 par Nick_425
Sammie_K	LabVIEW Interface for Arduino FAQ	5 054	★★★★★	9 juil. 2011 13:42 par Sammie_K
Sammie_K	LabVIEW Interface for Arduino Setup Procedure	17 538	★★★★★	9 juil. 2011 13:27 par Sammie_K
Sammie_K	Arduino BlinkM Example	1 715	☆☆☆☆☆	23 mai 2011 14:20 par Sammie_K
	Brady, tu as fini?	6 897	☆☆☆☆☆	17 mai 2011 15:21

# ARDUINO ET C# CRÉER UNE APPLICATION

# TP6

## TP6 - Connexion Arduino - C#



The screenshot shows a C# application interface. On the left, a code editor displays an Arduino sketch named "Arduino\_Commande\_LED\_RGB". The code reads RGB values from the Serial port and controls three PWM pins (ROUGE, VERT, BLEU) connected to an Arduino. A color picker window titled "Color Picker" is open, showing three vertical sliders for red, green, and blue. The right side of the screen shows a live video feed of a red LED glowing.

```
1 /* Commande LED RGB avec C# */
2 //Définition de pin PWM
3 #define ROUGE 3
4 #define VERT 5
5 #define BLEU 6
6
7 void setup() {
8     Serial.begin(115200);
9 }
10
11 void loop() {
12     if (Serial.available() > 0){
13         byte valeur_Rouge = Serial.parseInt();
14         byte valeur_Vert = Serial.parseInt();
15         byte valeur_Bleu = Serial.parseInt();
16         Serial.println(String(valeur_Rouge) + "/" + String(valeur_Vert) + "/" + String(valeur_Bleu));
17
18         while (Serial.available() > 0) Serial.read();
19         analogWrite(ROUGE, valeur_Rouge); // PWM sur la PIN de la LED ROUGE
20         analogWrite(VERT, valeur_Vert); // PWM sur la PIN de la LED VERTE
21         analogWrite(BLEU, valeur_Bleu); // PWM sur la PIN de la LED BLEU
22     }
23 }
24
25 //parseInt() returns the first valid (long) integer number from the serial buffer.
26 //Characters that are not integers (or the minus sign) are skipped.
27 //In particular:
28 // Initial characters that are not digits or a minus sign, are skipped;
29 // Parsing stops when no characters have been read for a configurable time-out value, or a non-digit is read;
30 // If no valid digits were read when the time-out (see Serial.setTimeout()) occurs, 0 is returned;
```

# TP6 - Connexion Arduino - C#

## Avec Microsoft VISUAL STUDIO:

Créer une application qui envoie par port série 3 entiers entre 0 et 255 à 3 trackbars et pilote, côté Arduino, une LED RGB

### PISTE VERTE:

1 – Créer une application Windows Form

2 – Créer:    - 1 bouton (nommé “CONNEXION PORT SERIE”) +  
              - 3 trackbars (à mettre en vertical, valeur min 0 max 255,  
                   ENABLE FALSE, couleur de fond R, G, B)

3 – A l’ouverture du formulaire Form1 les 3 trackbars doivent être  
Désactivés et leurs valeurs à 0

4 – Ajouter un serialPort (double clic)  
      => serialPort1 (configurer sur COMxxx :  
            le port com de l’arduino, 9600...)

5 – Quand on clique sur le bouton CONNEXION PORT SERIE  
      => il ouvre le port défini précédemment si il était fermé ou le ferme si il était ouvert  
      - Si on est connecté => le bouton doit indiquer “SE DÉCONNECTER”  
      - Si on est déconnecté => le bouton doit indiquer “SE CONNECTER”  
      - Les trackbars s’activent

6 – Dès qu’un trackbar change de valeur, on envoie la valeur de TOUS les trackbars sur le port Série:  
      objet serialPort1 et méthode WriteLine(...).

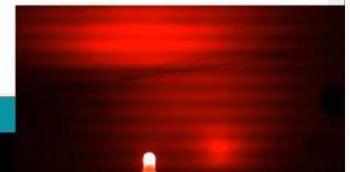
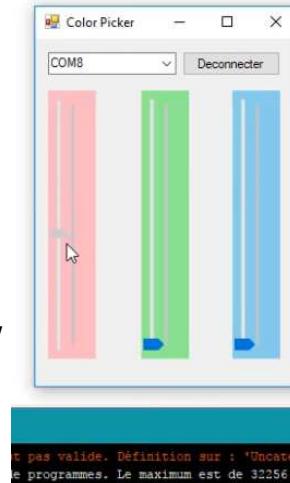
7 – Quand on clique sur “SE DÉCONNECTER”, remettre tous les trackbar à 0

### PISTE BLEU:

8 – Mettre une liste déroulante pour choisir le port Série:  
      => COMBO BOX, double cliquer sur FORM => FORM\_LOAD  
            Charger la classe série **IO.Ports** (using System.IO.Ports),  
            puis créer un tableau de Strings **ports** récupérant les PortNames  
            des ports Série avec la méthode **GetPortNames()**  
            Puis ajouter tous les éléments du tableau dans la **comboBox1**

```
String[] ports = SerialPort.GetPortNames();
foreach (var port in ports) {
    comboBox1.Items.Add(port);
}

serialPort1.PortName = comboBox1.SelectedItem.ToString()
```



## SOLUTION PISTE VERTE : C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
//using System.IO.Ports; //Pour récupérer les ports séries

namespace Test_Forms_02022020
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen) {
                serialPort1.Close();
                button1.Text = "Se CONNECTER";
                trackBar1.Enabled = false;
                trackBar2.Enabled = false;
                trackBar3.Enabled = false;
                trackBar1.Value = 0;
                trackBar2.Value = 0;
                trackBar3.Value = 0;
            }
            else {
                button1.Text = "Se DECONNECTER";
                serialPort1.PortName = "COM2";
                serialPort1.Open();
                trackBar1.Enabled = true;
                trackBar2.Enabled = true;
                trackBar3.Enabled = true;
            }
        }

        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            serialPort1.WriteLine(trackBar1.Value + "/" + trackBar2.Value + "/" + trackBar3.Value);
        }

        private void trackBar2_Scroll(object sender, EventArgs e)
        {
            serialPort1.WriteLine(trackBar1.Value + "/" + trackBar2.Value + "/" + trackBar3.Value);
        }

        private void trackBar3_Scroll(object sender, EventArgs e)
        {
            serialPort1.WriteLine(trackBar1.Value + "/" + trackBar2.Value + "/" + trackBar3.Value);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //Initialisation des trackballs
            trackBar1.Enabled = false;
            trackBar2.Enabled = false;
            trackBar3.Enabled = false;
            trackBar1.Value = 0; trackBar2.Value = 0; trackBar3.Value = 0;
        }
    }
}
```

# SOLUTION PISTE BLEUE: C#

```

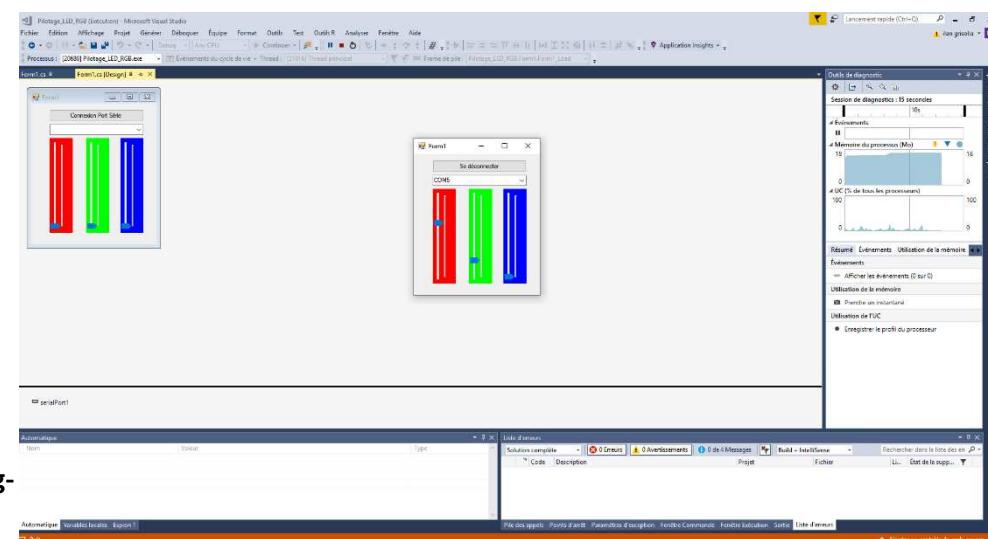
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10 using System.IO.Ports; //pour récupérer les ports séries
11
12 namespace Pilotage_LED_RGB
13 {
14     public partial class Form1 : Form
15     {
16         public Form1()
17         {
18             InitializeComponent();
19         }
20
21         private void button1_Click(object sender, EventArgs e)
22         {
23             if (serialPort1.IsOpen) {
24                 serialPort1.Close();
25                 button1.Text = "Se connecter";
26                 trackBar1.Enabled = false;
27                 trackBar2.Enabled = false;
28                 trackBar3.Enabled = false;
29                 trackBar1.Value = 0;
30                 trackBar2.Value = 0;
31                 trackBar3.Value = 0;
32             }
33             else {
34                 serialPort1.PortName = comboBox1.SelectedItem.ToString();
35                 serialPort1.Open();
36                 button1.Text = "Se déconnecter";
37                 trackBar1.Enabled = true;
38                 trackBar2.Enabled = true;
39                 trackBar3.Enabled = true;
40             }
41         }
42     }

```

```

43     private void trackBar1_Scroll(object sender, EventArgs e)
44     {
45         serialPort1.WriteLine(trackBar1.Value + "/" + trackBar2.Value + "/" +
46                             trackBar3.Value);
47     }
48
49     private void trackBar2_Scroll(object sender, EventArgs e)
50     {
51         serialPort1.WriteLine(trackBar1.Value + "/" + trackBar2.Value + "/" +
52                             trackBar3.Value);
53     }
54
55     private void trackBar3_Scroll(object sender, EventArgs e)
56     {
57         serialPort1.WriteLine(trackBar1.Value + "/" + trackBar2.Value + "/" +
58                             trackBar3.Value);
59     }
60
61     private void Form1_Load(object sender, EventArgs e)
62     {
63         String[] ports = SerialPort.GetPortNames();
64         foreach (var port in ports)
65         {
66             comboBox1.Items.Add(port);
67         }
68     }

```



## Générer un exécutable:

<https://docs.microsoft.com/fr-fr/visualstudio/ide/walkthrough-building-an-application?view=vs-2019>

### IV-3 TO GO FURTHER: PYTHON

# INTERFACE ARDUINO – PYTHON – AJAX - BDD

<https://www.python.org/>

The screenshot shows the Python.org homepage. At the top, there's a navigation bar with tabs for Python, PSF, Docs (which is highlighted), PyPI, Jobs, and Community. Below the navigation is the Python logo and a search bar. The main content area features a code snippet in a terminal window:

```
# Python 3: Simple arithmetic
>>> fruits = ['Banana', 'Apple', 'Lime']
0.5 loud_fruits = [fruit.upper() for fruit in
fruits]* 3
8>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']ion returns a float
5.666666666666667
#>list #d3the #enumerat#f#option
5>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

To the right of the code, there's a section titled "Intuitive Interpretation" with the following text:

Calculations are simple with Python, and expression syntax is straightforward: the operators `+`, `*`, and `/` work as can be expected; parentheses `( )` can be used for grouping. [More about simple math functions in Python 3.](#)

Below the code and interpretation, there's a navigation bar with links 1, 2, 3, 4, and 5. Further down, a large call-to-action box contains the text:

Python is a programming language that lets you work quickly  
and integrate systems more effectively. [»» Learn More](#)

## Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

[Start with our Beginner's Guide](#)

## Download

Python source code and installers are available for download for all versions! Not sure which version to use? Check here.

[Latest: Python 3.5.2 - Python 2.7.12](#)

## Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

[docs.python.org](#)

## Jobs

Looking for work or have a related position that you're hire for? Our [relaunched community-run job board](#) place to go.

[jobs.python.org](#)

# INTERFACE ARDUINO – PYTHON – AJAX - BDD

CODE: MANAGEMENT OF THE ROTARY ENCODER AND SENDING THE STRING BY ARDUINO VIA ITS SERIAL PORT:

```
Test_Encoder_Rotateur_LongPress_Debounce_PourProgramme_Python | Arduino 1.6.9
Fichier Édition Croquis Outils Aide
Test_Encoder_Rotateur_LongPress_Debounce_PourProgramme_Python

#define PIN_ENC1 2
#define PIN_ENC2 3
#define PIN_EBTN 4
#define PIN_LED1 13
static boolean moving=false;
volatile int encValue = 0;
int encValue_max = 3;
int encValue_min = 0;
unsigned int encValueTracking = 1;
boolean enc1 = false;
boolean enc2 = false;
// Here I'm messing around with button press durations - we could go to town here!
//enum pressDuration (shortPress, normalPress, longPress, veryLongPress );
//long presses[] = {150, 300, 800, 1400 };
//char* pressText[]={ "short press", "normal press", "long press", "very loooooooooong press"};
//char* pressText[]={"normal press", "long press", "very loooooooooong press"};
//#define DEBUG

enum pressDuration {normalPress, longPress, veryLongPress };
long presses[] = {300, 800, 1400 };
char* pressText[]={"normal press", "long press", "very loooooooooong press"};
//#define DEBUG

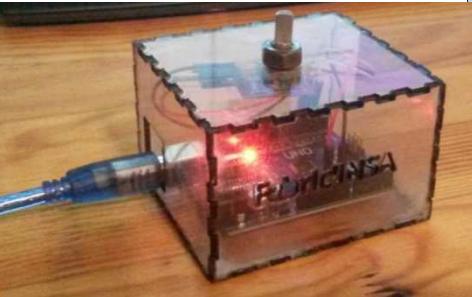
void setup()
{
    pinMode(PIN_ENC1, INPUT);
    pinMode(PIN_ENC2, INPUT);
    pinMode(PIN_EBTN, INPUT);
    pinMode(PIN_LED1, OUTPUT);
    digitalWrite(PIN_ENC1, HIGH);
    digitalWrite(PIN_ENC2, HIGH);
    digitalWrite(PIN_EBTN, HIGH);
    Serial.begin(9600);
    //menuIntro();
    attachInterrupt(0, intrEncChange1, CHANGE);
    attachInterrupt(1, intrEncChange2, CHANGE);
}

void intrEncChange1()
{
    if(moving)
        delay(1);
    if(digitalRead(PIN_ENC1) == enc1)
        return;
    enc1 = !enc1;
    if(enc1 && !enc2)
        encValue += 1;
}

void intrEncChange2()
{
    if(moving)
        delay(1);
    if(digitalRead(PIN_ENC2) == enc2)
        return;
    enc2 = !enc2;
    if(enc2 && !enc1)
        encValue -= 1;
}

void loop()
{
    static unsigned long btnHeld = 0;
    moving = true;
    if(encValueTracking != encValue){
        //Serial.print("encValue: ");
        if (encValue < encValue_min){
            encValue = encValue_max;
        }else {
            if (encValue > encValue_max){
                encValue = encValue_min;
            }
        }
        //fin du else

        Serial.println(encValue);
        //Serial.println(encValue, DEC);
        //Serial.println(encValue, BIN);
        encValueTracking = encValue;
    }
}
```



```
Test_Encoder_Rotateur_LongPress_Debounce_PourProgramme_Python

moving = false;

}

void intrEncChange2()
{
    if(moving)
        delay(1);
    if(digitalRead(PIN_ENC2) == enc2)
        return;
    enc2 = !enc2;
    if(enc2 && !enc1)
        encValue -= 1;

    moving = false;
}

void loop()
{
    static unsigned long btnHeld = 0;
    moving = true;
    if(encValueTracking != encValue){
        //Serial.print("encValue: ");
        if (encValue < encValue_min){
            encValue = encValue_max;
        }else {
            if (encValue > encValue_max){
                encValue = encValue_min;
            }
        }
        //fin du else

        Serial.println(encValue);
        //Serial.println(encValue, DEC);
        //Serial.println(encValue, BIN);
        encValueTracking = encValue;
    }
}
```

```
Test_Encoder_Rotateur_LongPress_Debounce_PourProgramme_Python

// Upon button press...
if((digitalRead(PIN_EBTN) == LOW) && !btnHeld){
    btnHeld = millis();
    digitalWrite(PIN_LED1, HIGH);
#endif DEBUG
    Serial.print("pressed selecting: ");
#endif DEBUG
    Serial.print("selected:");
    Serial.println(encValue);

}

// Upon button release...
if((digitalRead(PIN_EBTN) == HIGH) && btnHeld){
    long t = millis();
    t -= btnHeld;
    digitalWrite(PIN_LED1, LOW);
    int dur = veryLongPress;
    for(int i = 0; i<= veryLongPress; i++){
        if(t > presses[i])
            continue;
        dur = i;
        delay(100); //delay pour debounce
        break;
    }

#endif DEBUG
    Serial.print("released selecting: ");
    Serial.print(encValue, DEC);
    Serial.print(" (after ");
    Serial.print(t, DEC);
    Serial.print(" ms = ");
    Serial.print(pressText[dur]);
    Serial.println(")");
}

btnHeld = 0;
encValue=encValue_min; //reset de la valeur du compteur
//Serial.println(encValue);

}

void menuIntro()
{
    Serial.println("Encoder Menu - blah, blah, blah");
}
```

# INTERFACE ARDUINO – PYTHON – AJAX - BDD

PYTHON CODE TO READ FROM THE SERIAL PORT ON PC:

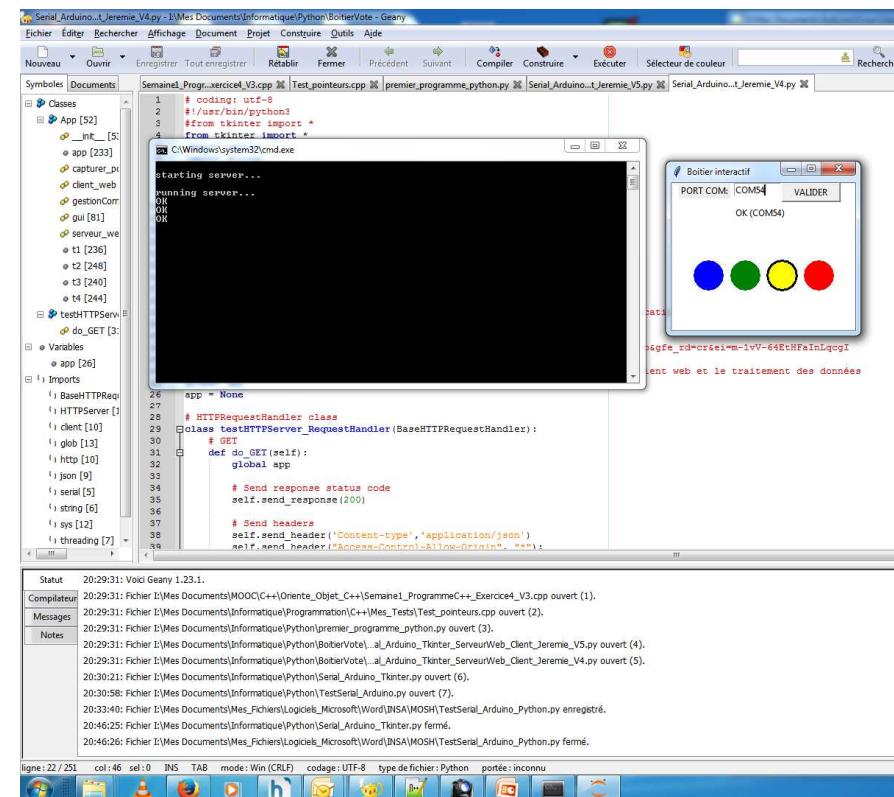
```

1 import serial
2 import string
3
4 ser = serial.Serial("COM54", 9600, timeout=1)
5 print (ser)
6
7 while 1:
8     donnee = str (ser.readline ())
9     tab = donnee.split ("') # Je découpe la chaîne en utilisant l'apostrophe ("').
10    #print (tab)
11    if len (tab) == 3: # Ca doit retourner 3 éléments: [0]:"selected:b", [1]:"XX\r\n", et [2]:"".
12        v = tab [1] # On récupère le deuxième élément.
13        v = v.strip ("\r\n") # On enlève les retours chariots.
14        #print (v)
15    try:
16        if v.find("selected:") == 0:
17            v = v.replace("selected:", "")
18            #print ("La valeur sélectionnée est:", int(v))
19            print ("La valeur sélectionnée est:", int (v))
20        else :
21            v = int (v) # On convertit la valeur en numérique.
22    except:
23        v = None # Si ça échoue, on met NULL (None en Python).
24    if v is not None: # Si v a bien été trouvé
25        #print ("La valeur lue est:", v)
26        print (v)
27    else:
28        print ("ERREUR !")

```

PYTHON CODE:

- LAUNCHING THE WEB SERVER
- - WEB CLIENT
- - TKINTER INTERFACE GRAPH



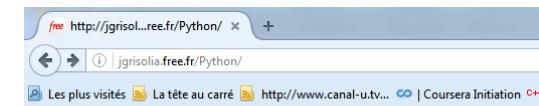
# INTERFACE ARDUINO – PYTHON – AJAX - BDD

**CODE AJAX data recovery sent to JSON from PYHTON**  
**SCRIPT: Technical AJAX - XMLHttpRequest send data in**  
**JSON format**

```

44 <script>
45 function maj()
46 {
47     var xhr = new XMLHttpRequest();
48     xhr.onreadystatechange = function() {
49         if (xhr.readyState == 4)
50         {
51             if (xhr.status == 200)
52             {
53                 /*
54                  readyState :
55                  Holds the status of the XMLHttpRequest. Changes from 0 to 4:
56                  0: request not initialized
57                  1: server connection established
58                  2: request received
59                  3: processing request
60                  4: request finished and response is ready
61                  status :
62                  200: "OK"
63                  404: Page not found
64                  */
65                 //console.log (xhr.responseText);
66                 var data = JSON.parse(xhr.responseText);
67                 document.getElementById("valeur").innerHTML = data.valeur;
68                 document.getElementById("select").innerHTML = data.select;
69                 /*document.getElementById("time").innerHTML = data.time;*/
70
71                 /*MISE A JOUR DES CARRÉS DE COULEURS*/
72                 for (var i = 0; i < 4; i++)
73                 {
74                     var oid = "val" + i
75                     var obj = document.getElementById(oid)
76                     if (obj != null)
77                     {
78                         if (i == data.valeur)
79                         {
80                             obj.style.border = "5px solid"
81                             obj.style.borderColor = "black"
82                         }
83                         else
84                         {
85                             obj.style.border = "5px solid"
86                             obj.style.borderColor = "transparent"
87                         }
88                     }
89                 }
90             }
91             setTimeout(maj, 100);
92         };
93     };
94     //xhr.open("GET", "http://localhost:8081", true); <!--envoie des données du local host vers
95     xhr.open("GET", "getval.php", true);<!--envoie des données de getval.php vers le serveur
96     xhr.send();
97 };
</script>
```

**WEB PAGE to**  
**display data with**  
**dynamic update (~**  
**100ms)**



AFFICHEZ LE CLASSEMENT

AFFICHEZ LES VOTES

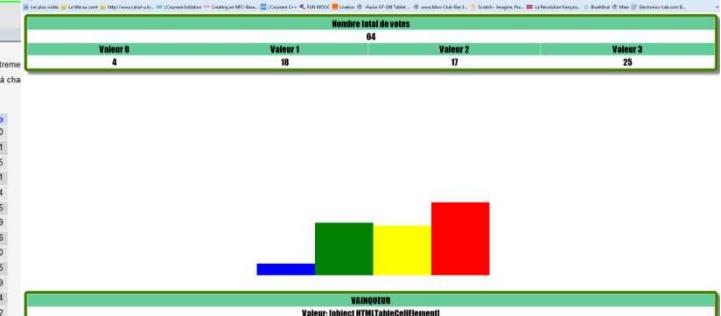
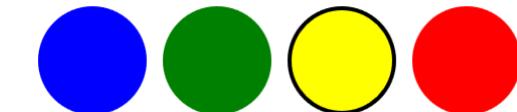
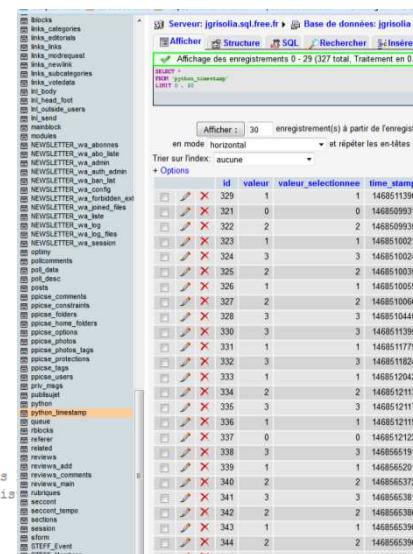
Valeur  
en cours

Valeur  
sélectionnée

2

0

BDD DE STOCKAGE  
DES DONNEES



**DISPLAY DYNAMICS OF VOTING**  
**DATA WEB PAGE:**

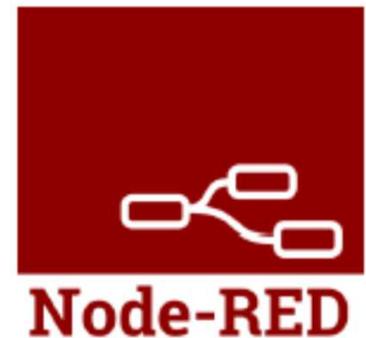
# ARDUINO ET NODE-RED

# What's Node-RED?

Node-RED is a powerful open source visual wiring tool for building Internet of Things (IoT) applications.

Node-RED uses visual programming that allows you to connect **code blocks**, **known as nodes**, together to perform a task.

The nodes when wired together are called flows.

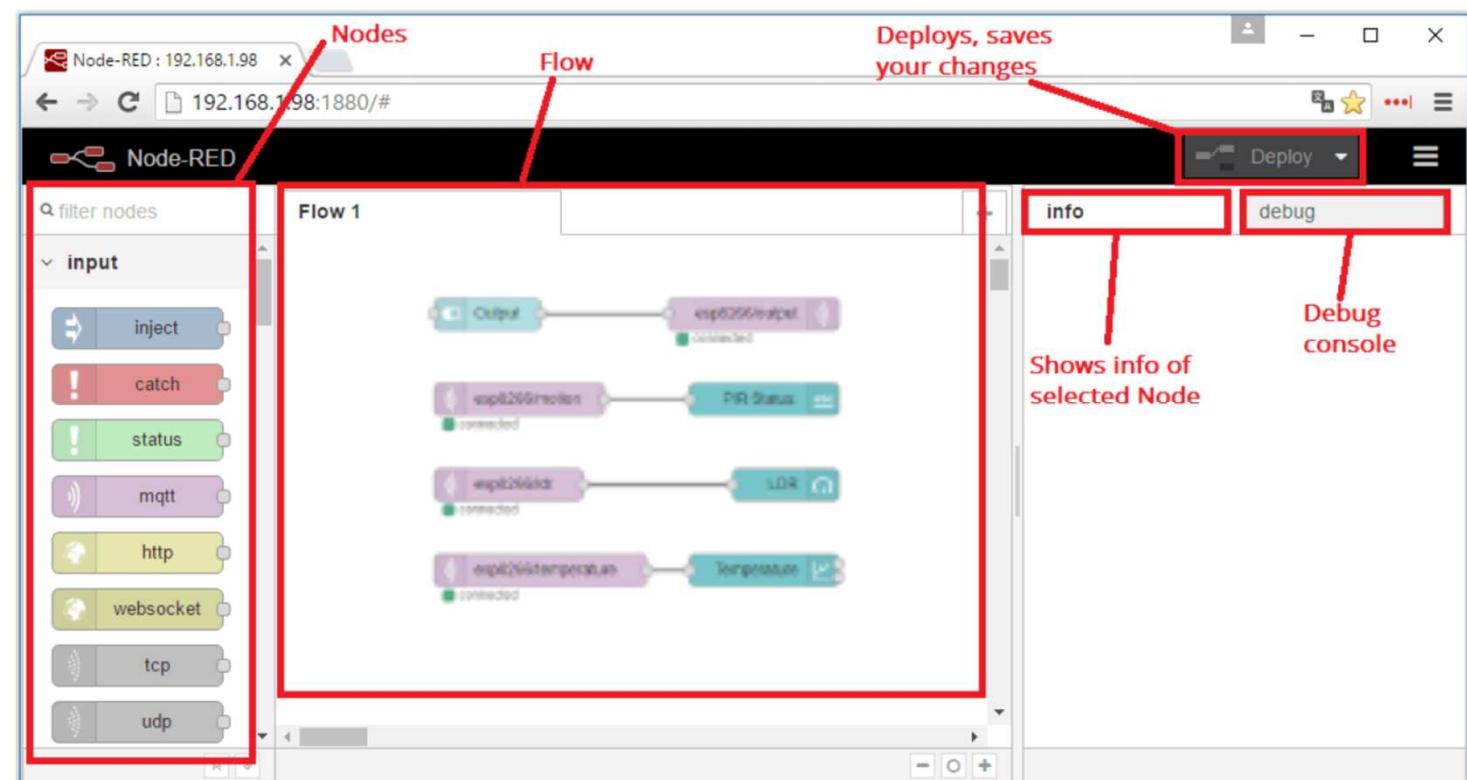


On the left-side, you can see a list with a bunch of blocks.

**These blocks are called nodes** and they are grouped by their functionality.

If you select a node, **you can see how it works in the info tab**.

**In the center, you have the Flow.**  
**The Flow is where you place the nodes.**



# Why Node-RED is a great solution?

## Why Node-RED is a great solution?

- Node-RED is open source and developed by IBM and runs perfectly with the Raspberry Pi.
- Node-RED allows you to prototype a complex home automation system quickly, giving you more time to spend on designing and making cool stuff.

## What can you do with Node-RED?

Node-RED makes it easy to:

- Access your Raspberry Pi GPIOs;
- Establish an MQTT connection with other boards (ESP32, ESP8266, Arduino, etc.);
- Create a responsive graphical user interface for your projects with Node-RED Dashboard;
- Communicate with third-party services (IFTTT.com, Adafruit.io, Thing Speak, and others);
- Retrieve data from the web (weather forecast, stock prices, emails, etc...);
- Create time triggered events;
- Store and retrieve data from a database;
- And much more...

Here's a resource with some [flow examples](#) and nodes for Node-RED.

# Installing Node-RED

Installing Node-RED in your Raspberry Pi is quick and easy. It just takes a few commands. Having a Terminal window of your Raspberry Pi open, enter the next command to install Node-RED:

```
pi@raspberry:~ $  
bash <(curl -sL https://raw.githubusercontent.com/nodered/  
raspbian-deb-package/master/resources/update-nodejs-andnodered)
```

The installation should be completed after a couple of minutes.

## Autostart Node-RED on boot

To run Node-RED automatically when the Pi boots up, you need to enter the following command:

```
pi@raspberry:~ $ sudo systemctl enable nodered.service
```

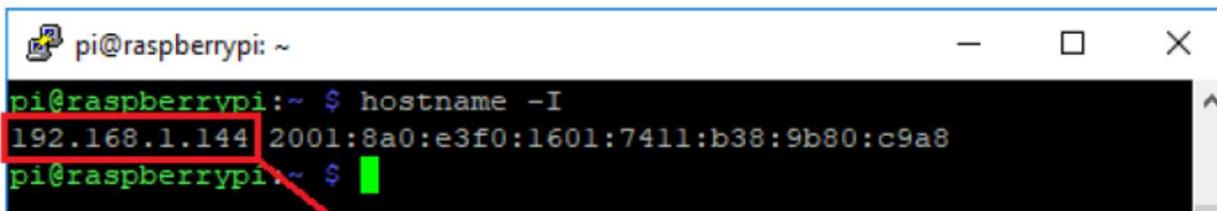
Now, restart your Raspberry Pi, so the auto-start takes effect:

```
pi@raspberry:~ $ sudo reboot
```

## Raspberry Pi IP Address

Retrieving your Raspberry Pi IP address, type next command in the terminal window:

```
pi@raspberry:~ $ hostname -I
```



```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ hostname -I  
192.168.1.144 2001:8a0:e3f0:1601:7411:b38:9b80:c9a8  
pi@raspberrypi:~ $
```

In this case, the Raspberry Pi IP address is 192.168.1.144 (save yours, because you'll need it in the next Unit).

# TESTING THE INSTALLATION

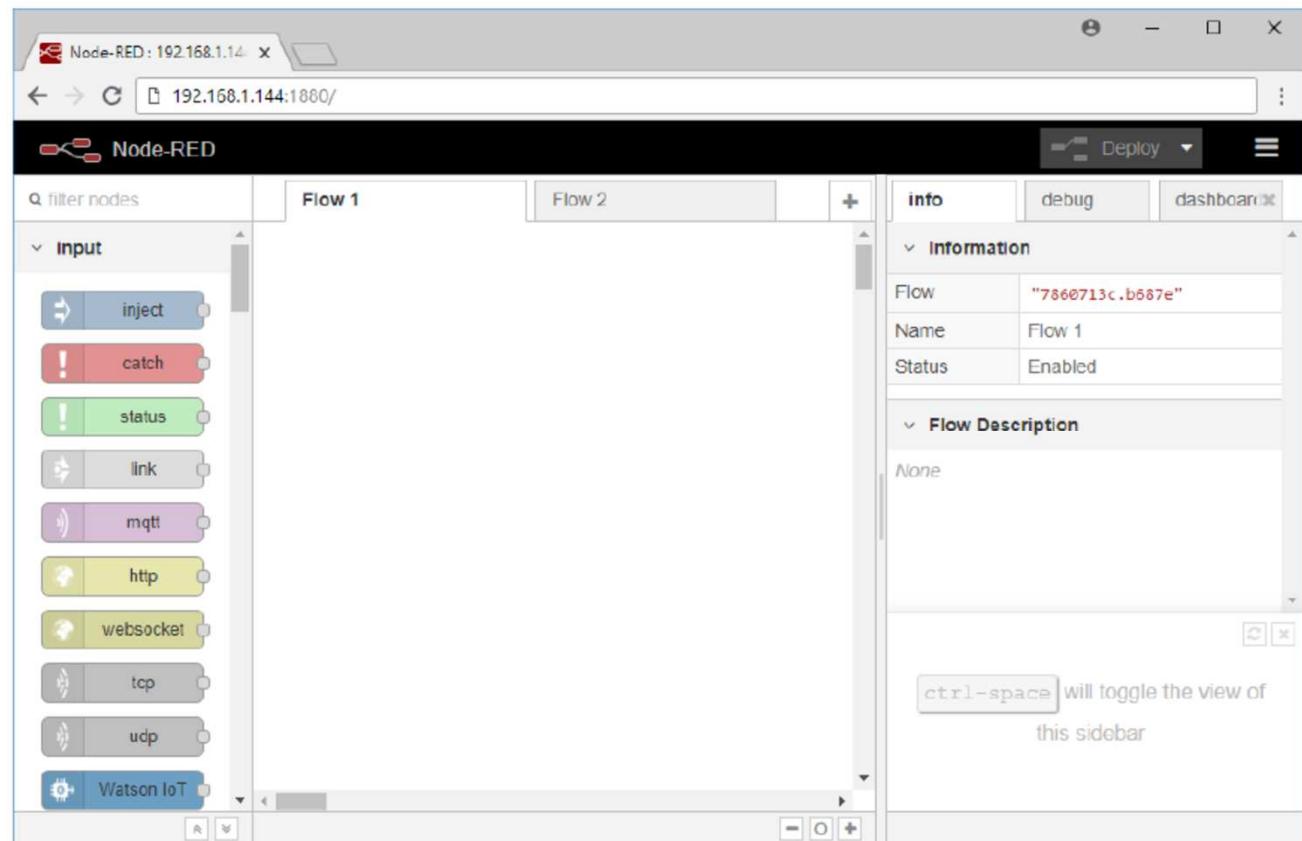
When your Pi is back on, you can test the installation by entering the IP address of your Pi in a web browser followed by the 1880 port number:

`http://YOUR_RPi_IP_ADDRESS:1880`

In my case is:

`http://192.168.1.144:1880`

A page as shown in  
the next figure  
should load:



# Installing Node-RED Dashboard

Node-RED Dashboard is a set of nodes that provides tools to create a user interface with buttons, charts, text, and other widgets.

To learn more about Node-RED Dashboard you can check the following links:

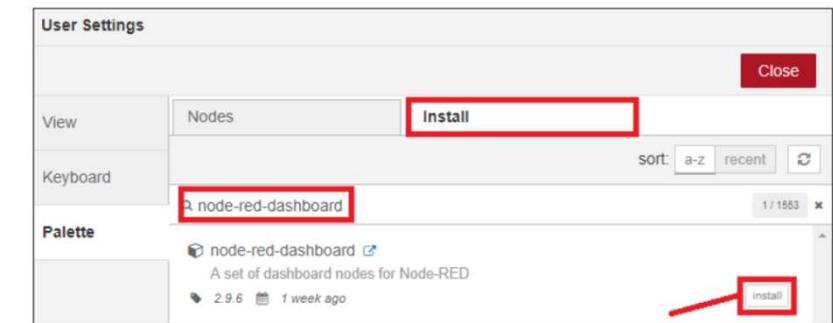
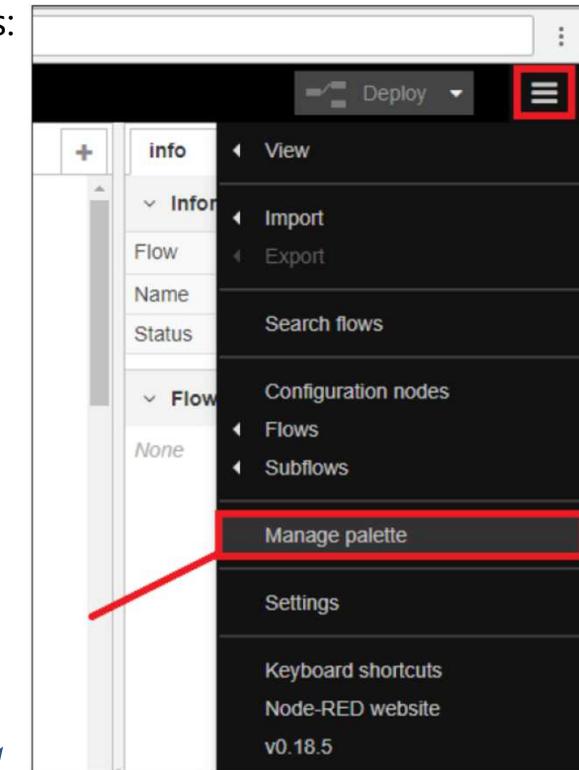
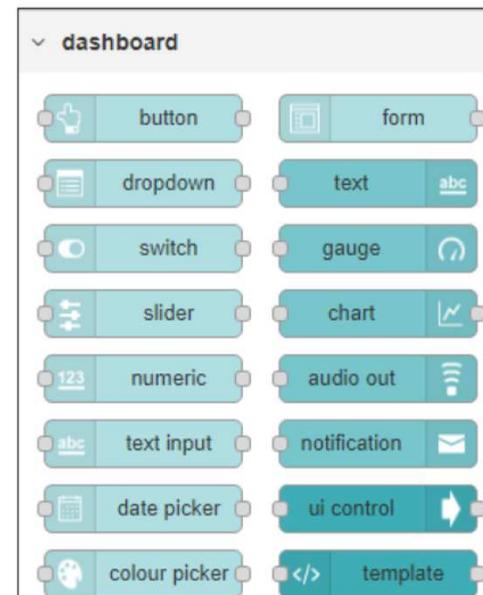
- Node-RED site: <http://flows.nodered.org/node/node-red-dashboard>
- GitHub: <https://github.com/node-red/node-red-dashboard>

## Installing Node-RED Dashboard

To install the Node-RED Dashboard, go to the “**Settings**” menu in the top-right corner and open the “**Manage palette**” menu:

A new window opens with the User Settings. Open the “**Install**” tab, search for “**node-red-dashboard**”, and press the “**install**” button:

Close the menu and a **new set of nodes should appear in your left window with the dashboard nodes**:



# TP7: ARDUINO AND NODE-RED

# TP7 - ARDUINO & NODE-RED

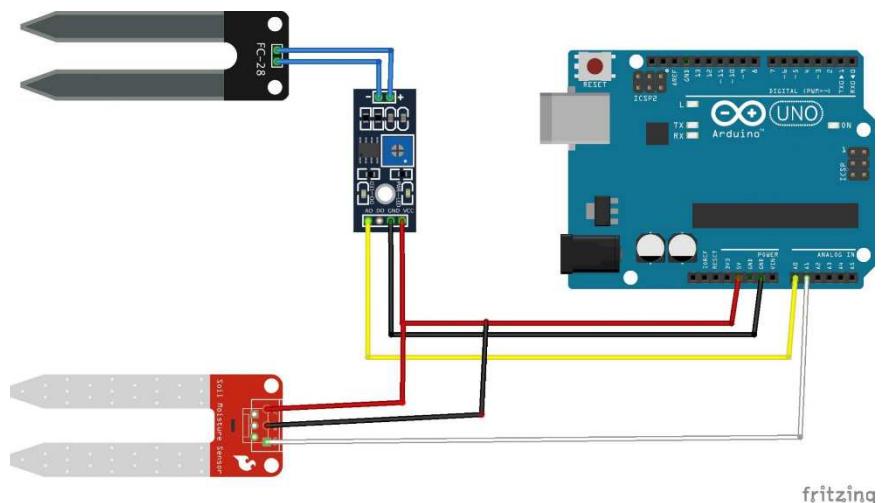
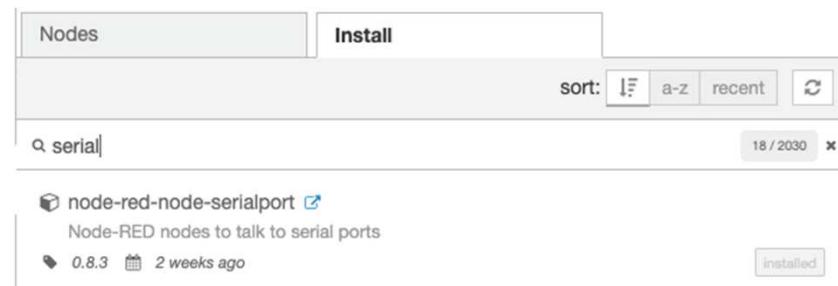
## Installer le Node Serial

Depuis la version 0.20, il est nécessaire d'installer le node Serial qui permet de communiquer via le port série. Comme précédemment, faites une recherche sur le mot clé Serial et installez le package officiel **node-red-node-serialport**.

On va simplement lire à intervalle régulier (ici toutes les 5s, 5000ms), la valeur du signal analogique (de 0 à 1023) pour chaque sonde d'humidité du sol. Ensuite on envoi la sortie sur le moniteur série de l'IDE Arduino.



```
Arduino_To_NodeRed_Humidity | Arduino 1.8.5
Fichier Édition Croquis Outils Aide
Arduino_To_NodeRed_Humidity
1 #define wait 5000
2
3 void setup() {
4     Serial.begin(115200);
5 }
6 void loop() {
7     int a4 = analogRead(A4); //Capteur1
8     int a5 = analogRead(A5); //Capteur2
9     Serial.print("{\"Capteur1\":");
10    Serial.print(a5);
11    Serial.print(", \"Capteur2\":");
12    Serial.print(a4);
13    Serial.println("}");
14    delay(wait);
15 }
```



Ce qui donne maintenant sur le moniteur série

**{"Capteur1":250, " Capteur2":301}**

**=> FORMAT JSON (clé = valeur)**

# FORMAT JSON

## Formater les données en JSON

On pourrait envoyer directement les mesures sur le port série de l'Arduino avec un séparateur, par exemple un caractère spécial (|, -, #...) mais cette stratégie implique que l'on sache précisément la position de chaque données.

- Aucun problème avec une ou deux données, ça devient plus compliqué lorsqu'il y en à une dizaine.
- Autre problème, la conversion de chaîne de caractères qui est un éternel problème en informatique.

Pour éviter tous ce problème, nous allons mettre en forme les données et indiquer à chaque fois à quoi elle correspond.

## Pour cela, nous allons utiliser le format JSON.

L'avantage, c'est qu'il est supporté par tous les langages modernes.

**C'est même la structure de données par défaut du javascript, langage sur lequel repose Node-RED.**

**Le JSON est une mise en forme structurée des données de type clé = valeur.**

Chaque ligne de données est séparée par une virgule (sans la dernière ligne).

Une valeur peut être:

- une chaîne de caractères (une image sera une chaîne),
- un nombre (entier ou décimal),
- un tableau (chaîne, nombre),
- une structure (qui contiendra elle-même des données sous la forme clé = valeur).

Voici un exemple. Tout d'abord en ligne, c'est ce que le code Arduino va générer

```
{"sonde1":22.1,"sonde2":64.1,"unites":{"sonde1":"°C","sonde2":"%"}}
```

# FORMAT JSON

Pour vérifier votre code, je vous conseille d'utiliser le site [jsonlint.com](http://jsonlint.com) qui est gratuit.

Pour de gros projet, je vous conseille d'utiliser la librairie ArduinoJSON.

Elle permet de stocker les données au format JSON dans la mémoire de l'Arduino.

C'est très pratique pour :

- extraire des données,
- des réglages,
- enregistrer un historique dans la mémoire SPIFFS ou une carte SD.

Ici, on va faire plus simple et directement construire une chaîne de caractères et l'envoyer sur le port série ce qui donne le code suivant

```
{  
    "sonde1":22.1,  
    "sonde2":64.1,  
    "unites": {  
        "sonde1": "°C",  
        "sonde2": "%" }  
}
```

# NODE-RED

1

Se connecter à l'Arduino à l'aide du Node Serial (dans la palette Input, puisqu'on veut lire les mesures).

Glissez-déposer le node sur la page blanche et faites un double clic sur le Node Serial pour ouvrir le panneau de configuration.



4

Cherchez le Node Debug



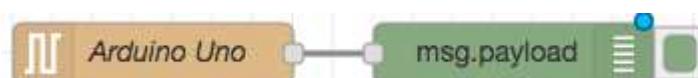
2

Cliquez sur le crayon pour ajouter une nouvelle connexion. Utilisez la loupe pour lister les ports COM. **Ici l'Arduino est connecté sur un Raspberry Pi 3, donc le chemin vers l'Arduino est au format Linux.** Sur Windows, ce sera un port COMx.



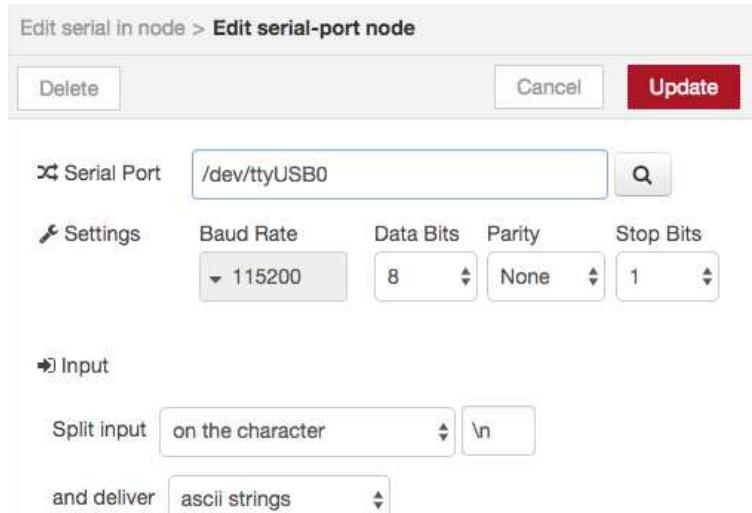
5

Reliez le Node Serial au Node debug. Pour cela, placez la souris sur le carré qui symbolise la sortie de Node Serial, un fil orange apparaît. Allez l'accrocher à l'entrée du node debug. Vous venez de créer votre premier flow.



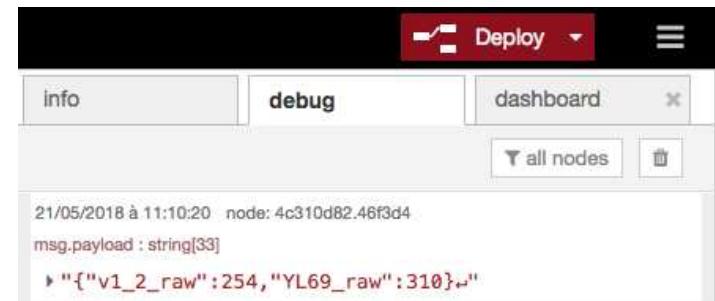
3

Dans le code Arduino, la ligne de code `Serial.begin(115200)` permet d'initialiser la vitesse à 115200 bauds, indiquez la vitesse sous Baud Rate. Enregistrez, c'est tout ce qu'il y à faire



6

Déployez le flow et cliquant sur Deploy et ouvrez l'onglet debug pour visualiser les données qui arrivent de l'Arduino. Attendez quelques secondes en fonction de la temporisation programmée dans le code Arduino.



# EXTRAIRE LES DONNÉES DU JSON VENANT DE L'ARDUINO

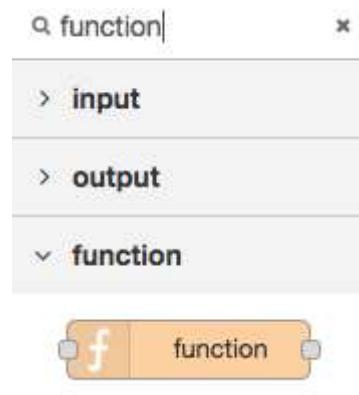
1

Pour le moment, on récupère une chaîne de caractère sur le port série de l'Arduino, on va la convertir en un objet JSON exploitable par Node-RED à l'aide du node JSON. Placez le sur le flow et reliez-le au port série.



2

On va maintenant extraire chaque mesure avec un peu de code javascript. Placez un node Function et reliez le à la sortie de node JSON



3

Ouvrez la fonction et collez ce code javascript.

```
msg.topic = "Capteur 1";
msg.payload =
msg.payload.Capteur1;
return msg;
```

Node-RED transfère des messages (**msg**) au format JSON entre chaque Node (noeud du programme).

Les données se trouvent dans la clé **payload**.

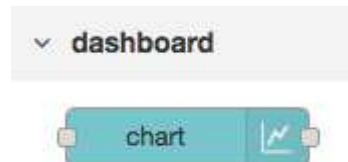
La première ligne extrait la mesure du capteur 1 (ou Capteur 2) et écrase le payload actuel.

La fonction retourne le message (msg) actualisé. Il ne renvoie plus que la mesure du premier capteur

*Faites la même chose pour le second capteur en ajoutant un second flow*

# VISUALISER LES MESURES SUR UN GRAPHIQUE

- 1 Cherchez le node chart et placer le sur le flow



- 2

Reliez les des fonctions au node chart. Ouvrez le panneau de configuration du graphique. Cliquez d'abord sur le crayon pour créer un groupe.

Edit chart node > Edit dashboard group node

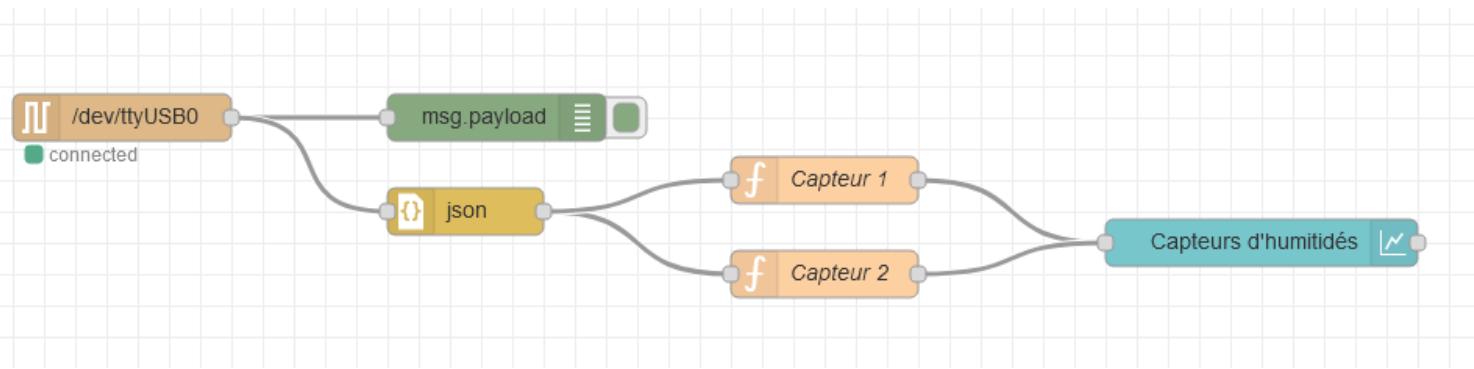
Delete	Cancel	Update
Name	Capteurs d'humidité - Moisture sensors	
Tab	Home	
Width	9	
<input checked="" type="checkbox"/> Display group name		
<input type="checkbox"/> Allow group to be collapsed		

- 3 Ensuite vous pouvez modifier certains paramètres :
- **Group** : sélectionnez le groupe que vous venez de créer
  - **Size** : taille
  - **Label** : libellé
  - **X-axis** : nombre de points ou période de temps
  - **Name** : le nom qui apparaît sur le flow

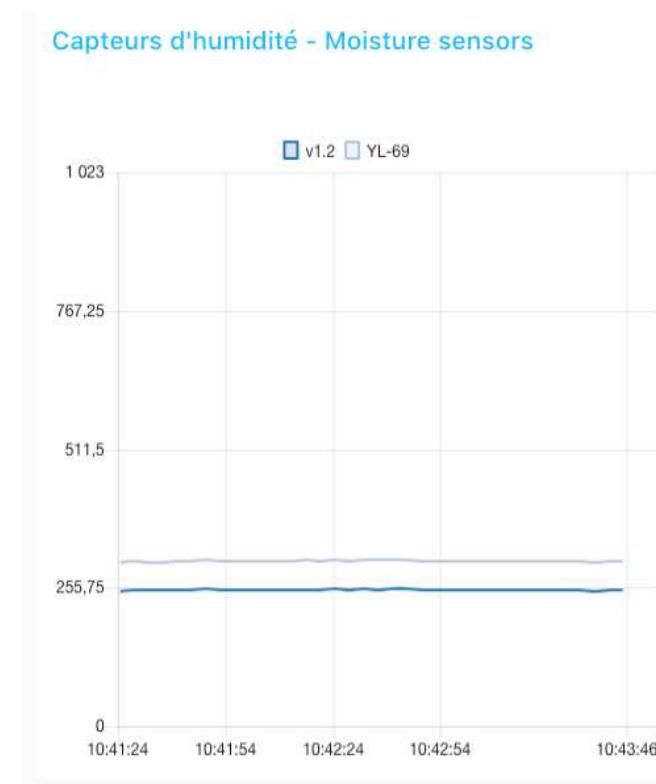
Edit chart node

Delete	Cancel	Done
node properties		
Group	Capteurs d'humidité - Moisture sensors	
Size	9 x 10	
Label	Arduino YL-69 + v1.2 data	
Type	Line chart	
X-axis	last 60 minutes OR 1000 points	
X-axis Label	HH:mm:ss	
Y-axis	min 0 max 1023	
Legend	Show Interpolate bezier	
Series Colours	Blue, Light Blue, Orange Green, Light Green, Red Pink, Purple, Light Purple	

# VISUALISER LES MESURES SUR UN GRAPHIQUE



Enregistrez, déployez et allez à l'adresse **localhost:1880/ui** (ou IP:1880/ui) pour visualiser les mesures de vos capteurs connectés à l'Arduino



# ENREGISTRER LES MESURES DANS UN FICHIER CSV

Exploiter vos données sur un tableau en les enregistrant au format csv (fichier texte dont le séparateur de données est un point-virgule). => On ajoute une nouvelle fonction qui renvoie une ligne dont chaque données est séparée par un point virgule. Voici l'ordre des colonnes :

- date au format année/mois/jour (supporté par tous les tableurs)
- heure au format hh:mm:ss
- valeur de la sonde Capteur1
- valeur de la sonde Capteur2

Cherchez le node file et placez un node input sur le flow

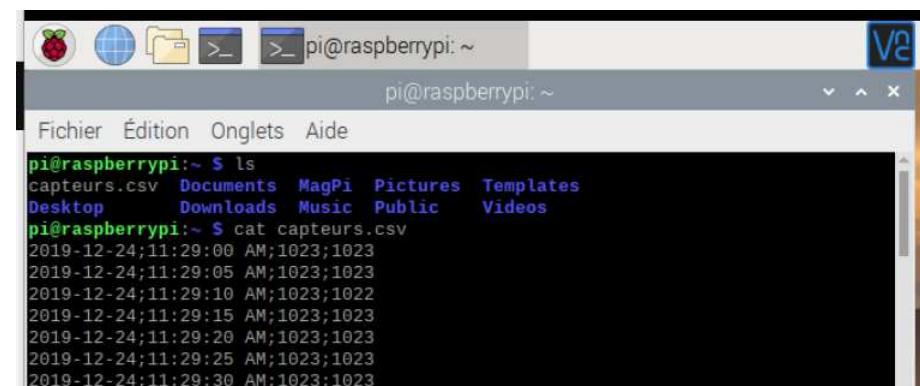
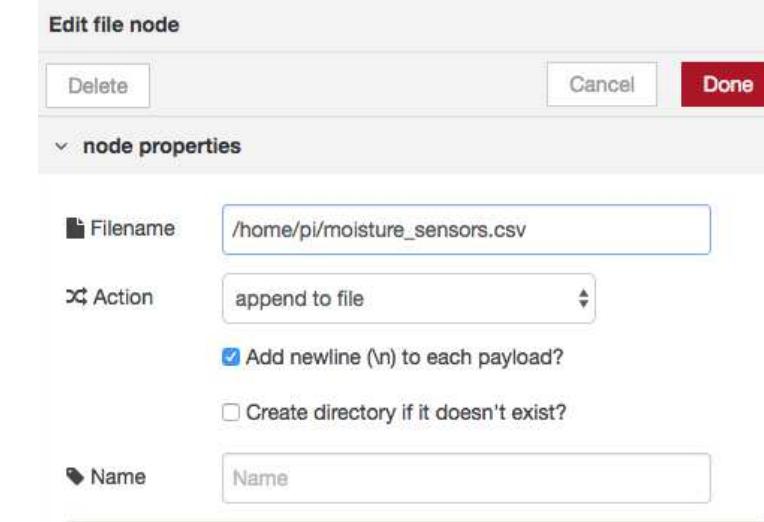


Sur le panneau de configuration, indiquez le chemin de destination. Par défaut le fichier est enregistré dans le répertoire de Node-RED. Cochez **Add newline to each payload** pour ajouter une nouvelle ligne au fichier à chaque nouvel enregistrement

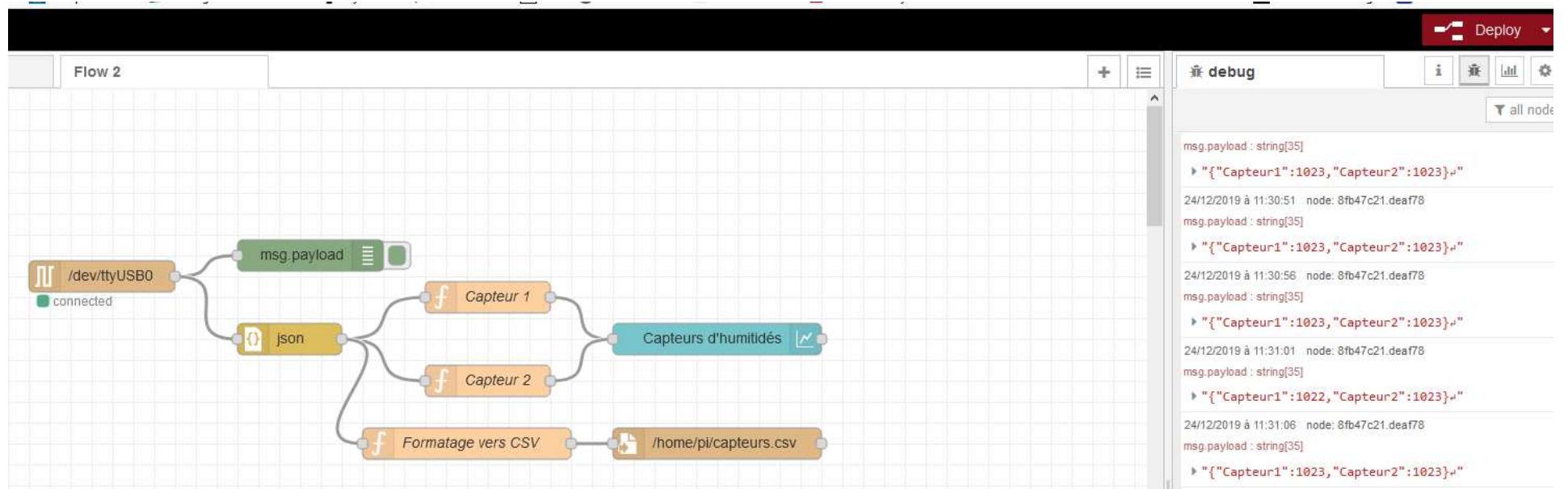
Déployez, après quelques minutes, voici à quoi va ressembler le fichier CSV

```
var date = new Date().toLocaleDateString();
var time = new Date().toLocaleTimeString();

var output = date + ";" + time + ";" + msg.payload.Capteur1 + ";" +
msg.payload.Capteur2;
msg.payload = output;
return msg;
```



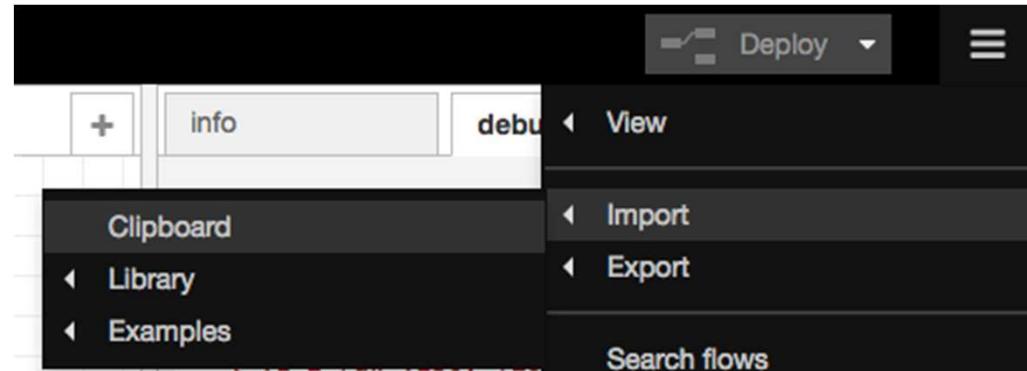
# RESULTAT FINAL



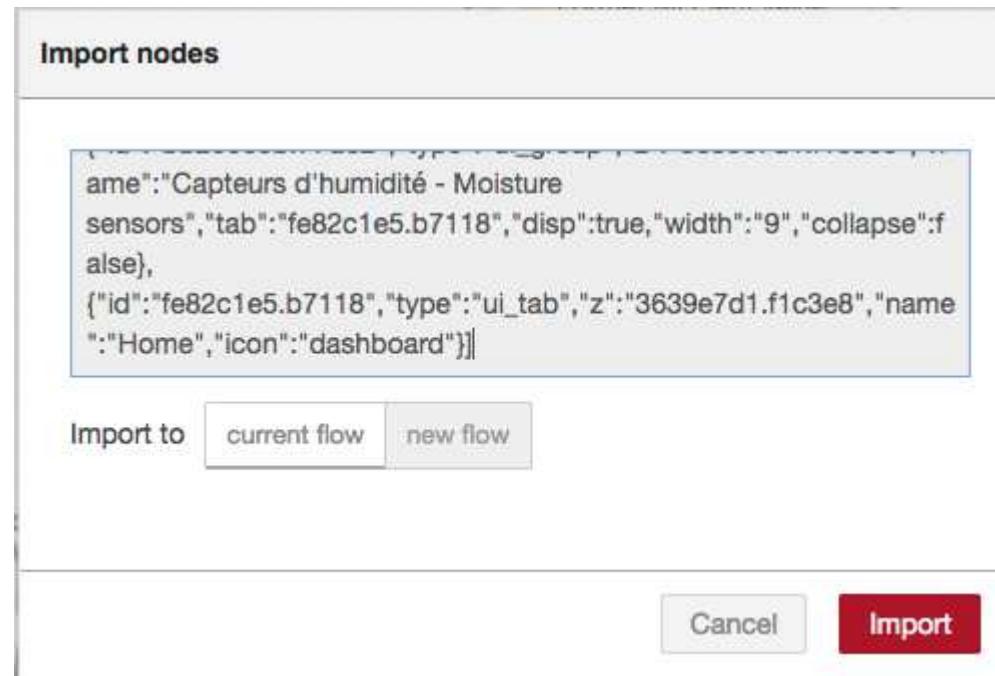
```
[{"id": "62e97db.8731284", "type": "tab", "label": "Flow 2", "disabled": false, "info": ""}, {"id": "54ab6674.d42c38", "type": "serial", "in": "z", "z": "62e97db.8731284", "name": "", "serial": "aa3b554b.8d2e5", "x": 270, "y": 180, "wires": [{"z": "8fb47c21.deaf78", "y": 3839e942.ff2806}], {"id": "8fb47c21.deaf78", "type": "debug", "z": "62e97db.8731284", "name": "", "active": true, "tostidebar": true, "console": false, "tostatus": false, "complete": false, "x": 470, "y": 160, "wires": []}, {"id": "3839e942.ff2806", "type": "json", "z": "62e97db.8731284", "name": "", "property": "payload", "action": "", "pretty": false, "x": 450, "y": 240, "wires": [{"z": "6f21aec7.17f468", "y": 599ec37e.beae6c}, {"z": "dd2233da.8b38b8"}]}, {"id": "6f21aec7.17f468", "type": "function", "z": "62e97db.8731284", "name": "Capteur 2", "func": "msg.topic = \"Capteur 2\";\\nmsg.payload = msg.payload.Capteur2;\\nreturn msg;", "outputs": 1, "noerr": 0, "x": 640, "y": 280, "wires": [{"z": "26b897fb.2568e8"}]}, {"id": "599ec37e.beae6c", "type": "function", "z": "62e97db.8731284", "name": "Capteur 1", "func": "msg.topic = \"Capteur 1\";\\nmsg.payload = msg.payload.Capteur1;\\nreturn msg;", "outputs": 1, "noerr": 0, "x": 640, "y": 200, "wires": [{"z": "26b897fb.2568e8"}]}, {"id": "26b897fb.2568e8", "type": "ui_chart", "z": "62e97db.8731284", "name": "", "group": "3de6bd76.a0e5ba", "order": 2, "width": "8", "height": "6", "label": "Capteurs d'humidités", "chartType": "line", "legend": "true", "xformat": "HH:mm:ss", "interpolate": "bezier", "nodata": "", "dot": true, "ymin": "0", "ymax": "1023", "removeOlder": "60", "removeOlderPoints": "", "removeOlderUnit": "60", "cutout": 0, "useOneColor": false, "colors": ["#1f77b4", "#aec7e8", "#ff7f0e", "#2ca02c", "#98df8a", "#d62728", "#ff9896", "#9467bd", "#c5b0d5"], "useOldStyle": false, "outputs": 1, "x": 860, "y": 340, "wires": [{"z": "1457badc.cfef6d"}]}, {"id": "1457badc.cfef6d", "type": "file", "z": "62e97db.8731284", "name": "", "filename": "/home/pi/capteurs.csv", "appendnewline": true, "createDir": false, "overwriteFile": "false", "encoding": "none", "x": 860, "y": 340, "wires": []}, {"id": "dd2233da.8b38b8", "type": "function", "z": "62e97db.8731284", "name": "Formatage vers CSV", "func": "var date = new Date().toLocaleDateString();\\nvar time = new Date().toLocaleTimeString();\\n\\nvar output = date + \";\" + time + \";\" + msg.payload.Capteur1 + \";\" + msg.payload.Capteur2;\\nmsg.payload = output;\\nreturn msg;\\n", "outputs": 1, "noerr": 0, "x": 620, "y": 340, "wires": [{"z": "1457badc.cfef6d"}]}, {"id": "aa3b554b.8d2e5", "type": "serial-port", "z": "", "serialport": "/dev/ttyUSB0", "serialbaud": "115200", ".databits": "8", "parity": "none", "stopbits": "1", "waitfor": "", "dtr": "none", "rts": "none", "cts": "none", "dsr": "none", "newline": "\n", "bin": "false", "out": "char", "addchar": "", "responsetimeout": "10000"}, {"id": "3de6bd76.a0e5ba", "type": "ui_group", "z": "", "name": "Main", "tab": "a15fdf1b.c86f1", "disp": true, "width": "8", "collapse": false}, {"id": "a15fdf1b.c86f1", "type": "ui_tab", "z": "", "name": "Home", "icon": "dashboard", "disabled": false, "hidden": false}]
```

## IMPORTER/EXPORTER UN FLOW

Copiez le code précédent. Ouvrez le menu de NodeRED puis Import -> Clipboard



Collez le code et choisissez où vous voulez le coller puis Import





# MicroPython on ESP32

## What is MicroPython?

MicroPython is a re-implementation of Python 3 targeted for microcontrollers and embedded systems. MicroPython is very similar with regular Python.

*So, if you already know how to program in Python, you also know how to program in MicroPython.*



## Python vs MicroPython

Apart from a few exceptions, the language features of Python are also available in MicroPython. The biggest difference, is that MicroPython was designed to work under constrained conditions.



Because of that, MicroPython does not come with the full standard library.

**It only includes a small subset of the Python standard library.**

**However, it does include modules to access low-level hardware**

**- this means that there are libraries to easily access and interact with the GPIOs.**

Additionally, devices with Wi-Fi capabilities like the ESP8266 and ESP32 include modules to support network connections.

# MicroPython on ESP32

## Why MicroPython?

Python is one of the most widely used, simple and easy-to-learn programming languages around. So, the emergence of MicroPython makes it extremely easy and simple to program digital electronics.

**MicroPython's goal is to make programming digital electronics as simple as possible**, so it can be used by anyone.

Currently, MicroPython is used by hobbyists, researchers, teachers, educators, and even in commercial products.

The code for blinking a LED on a ESP32 or ESP8266 is as simple as follows:

```
from machine import Pin  
from time import sleep  
  
led = Pin(2, Pin.OUT)  
  
while True:  
    led.value(not led.value())  
    sleep(0.5)
```



One great feature of MicroPython is that **it comes with an interactive REPL (Read-Evaluate-Print Loop).** The REPL allows you to connect to a board and execute code quickly without the need to compile or upload code

# MicroPython on ESP32

## MicroPython - Boards support

MicroPython runs on many different devices and boards, such as:



- [ESP32](#)
- [ESP8266](#)
- PyBoard
- Micro:Bit
- [Teensy 3.X](#)
- WiPy - Pycom
- Adafruit Circuit Playground Express
- Other ESP32/ESP8266 based boards

REM: ESP32 is the successor of the ESP8266, and at the moment, not all features are available in MicroPython to take the most out of the ESP32 – it's still an ongoing project.

***However, it's very usable and you can make a lot of projects with it.***

**ESP32 and ESP8266 boards are similar**, and you won't feel almost any difference programming them using MicroPython. This means that anything you write for the ESP8266 should also run with no changes or minimal changes on the ESP32 (**mainly changing the pin assignment**).

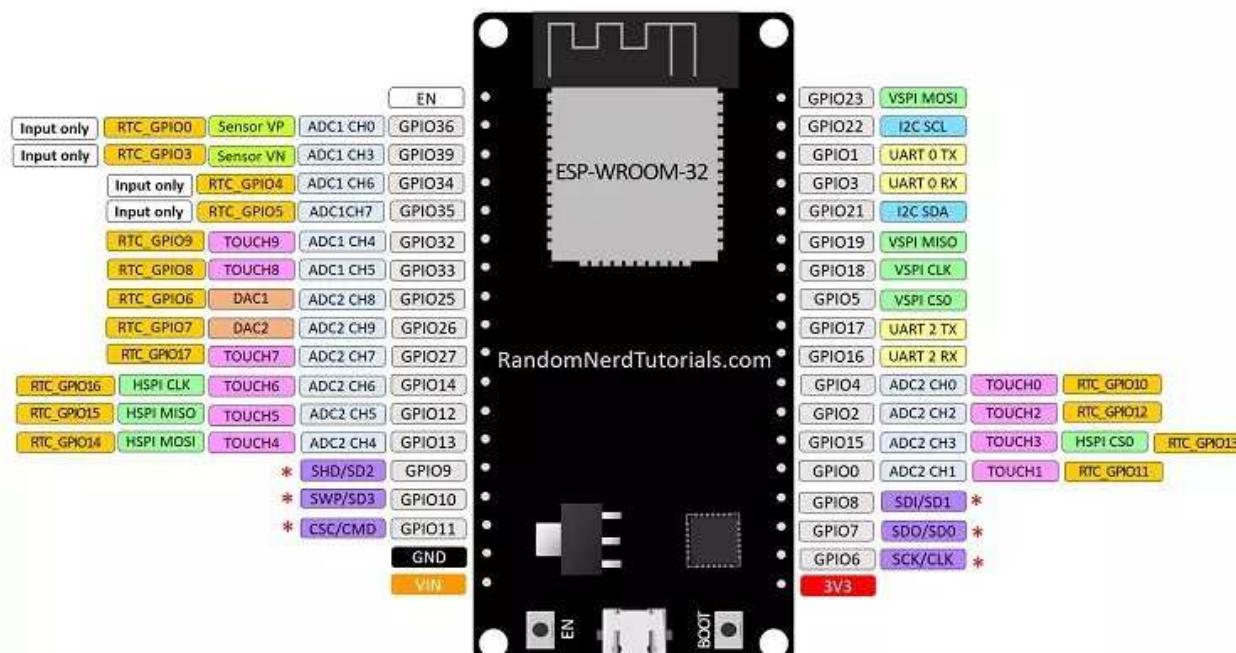
# ESP32 - Pinouts

<https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>



# ESP32 DEV KIT V1 - Pinouts

## ESP32 DEVKIT V1 – DOIT version with 36 GPIOs



GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fail if pulled high
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		Input only
39	OK		input only

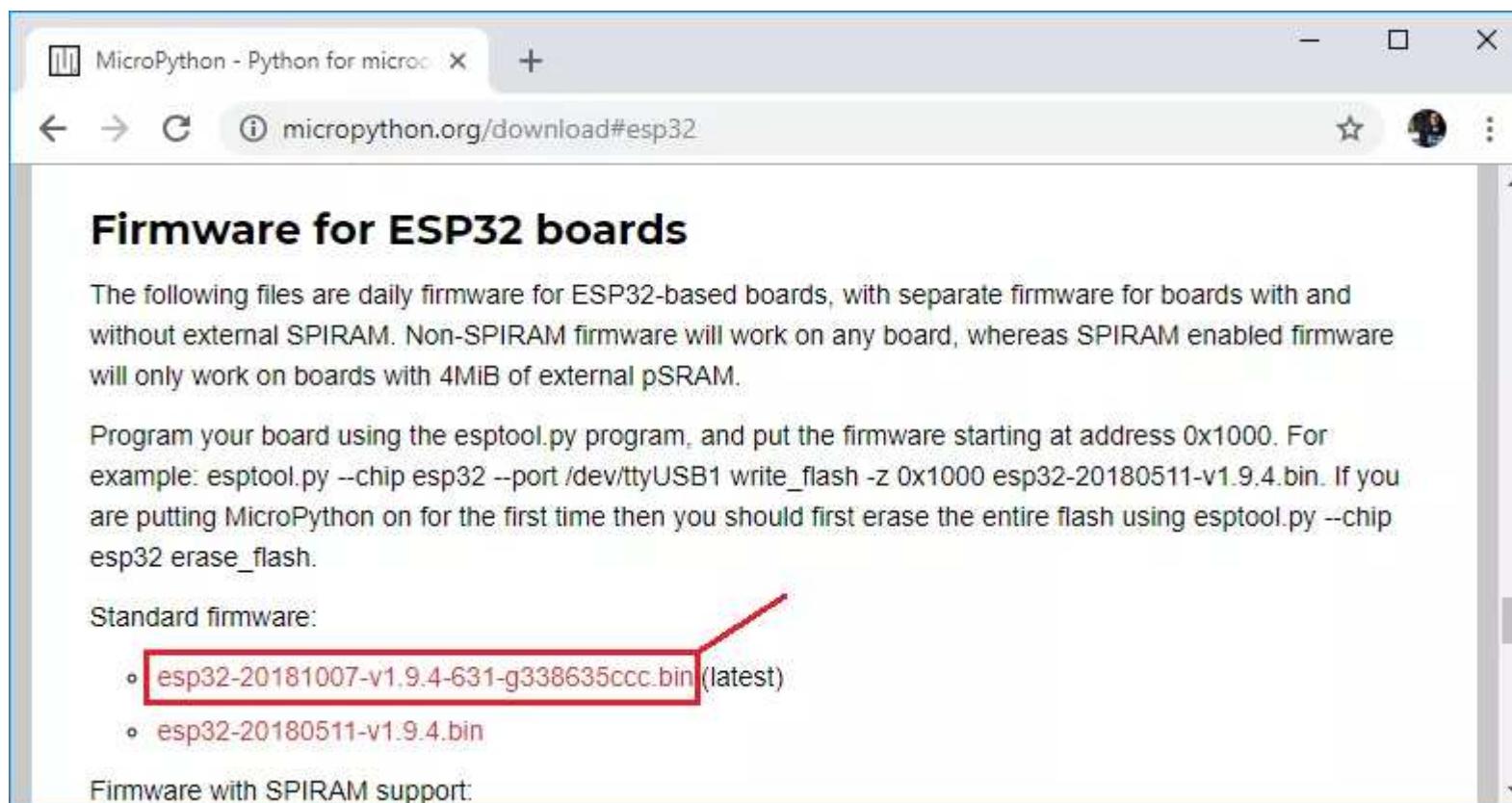
# FLASHING MICROPYTHON TO ESP32 and ESP8266



# MicroPython on ESP32

## [ESP32] Downloading and Flashing the MicroPython Firmware on ESP32:

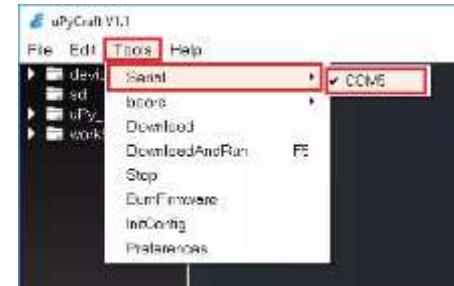
To download the latest version of MicroPython firmware for the ESP32, go to the [MicroPython Downloads page](#) and scroll all the way down to the ESP32 section. You should see a similar web page (see figure below) with the latest link to download the ESP32 .bin file - for example: **esp32-20181007-v1.9.4-631-g338635ccc.bin.**



# MicroPython on ESP32

## Selecting Serial Port

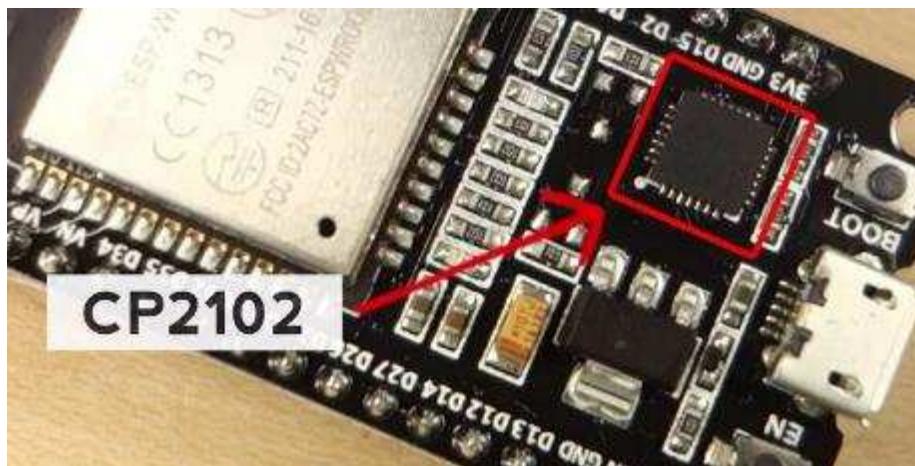
Go to **Tools** > **Serial** and select your ESP32 COM port (in our case it's COM5).



The [ESP32 DEVKIT V1 DOIT](#) board uses the **CP2102** chip.



You can download the CP2102 drivers on the [Silicon Labs](#) website.



SILICON LABS

About Products Solutions Community & Support

Products Development Tools Software USB to UART Bridge VCP Drivers

CP210x USB to UART Bridge VCP Drivers

The CP210x USB to UART Bridge Virtual COM Port (VCP) drivers are required for serial operation of a virtual COM Port to facilitate serial communication with CP210x products. The CP210x can also interface to a host using its USB-to-serial driver. These drivers are static, example detailed in application note AN1017: The Serial Communications Guide for the CP210x, download an example code.

AN1017: The Serial Communications Guide for the CP210x

Download Software

The CP210x Mini-Modem Driver and Runtime DLL have been updated and must be used with v1.0 and later of the CP210x Windows VCP Driver. Applications using software controls (flex232v1011497.zip, AN22221Lap124H12557.zip) (if you are using a 3rd party driver or need support you can download and read Application Notes Software)

Regulatory software and driver package download links and support information

Download for Windows 10 Universal (v10.1.1)

Platform	Software	Release Notes
Windows 10 Universal	Driver CP210x	Download CP210x Driver

# MicroPython on ESP32

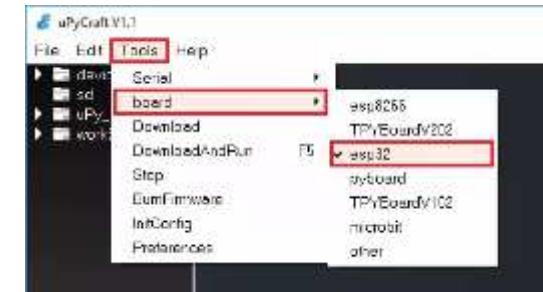
After they are installed, restart the uPyCraft IDE and you should see the COM port in the **Tools** menu.

2. If you have the drivers installed, but you can't see your device, double-check that you're using a USB cable with data wires.

USB cables from powerbanks often don't have data wires (they are charge only). So, your computer will never establish a serial communication with your ESP32. Using a proper USB cable should solve your problem.

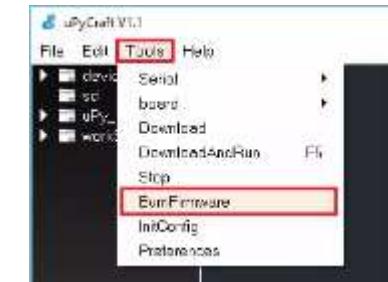
## Selecting the Board

Go to **Tools** > **Board**. For this tutorial, we assume that you're using the ESP32, so make sure you select the “**esp32**” option:



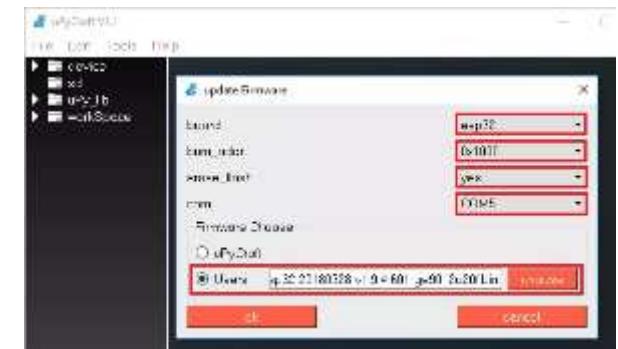
## Flashing/Uploading MicroPython Firmware

Finally, go to **Tools** > **BurnFirmware** menu to flash your ESP32 with MicroPython.



## Select all these options to flash the ESP32 board:

- board: **esp32**
- burn\_addr: **0x1000**
- erase\_flash: **yes**
- com: **COMX** (*in our case it's COM5*)
- Firmware: Select “**Users**” and choose the **ESP32xxx.bin** file downloaded earlier



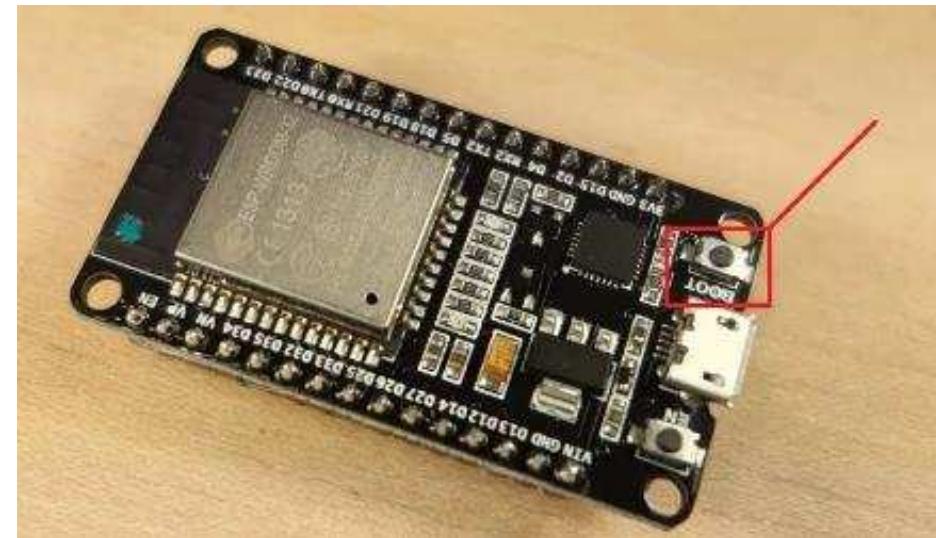
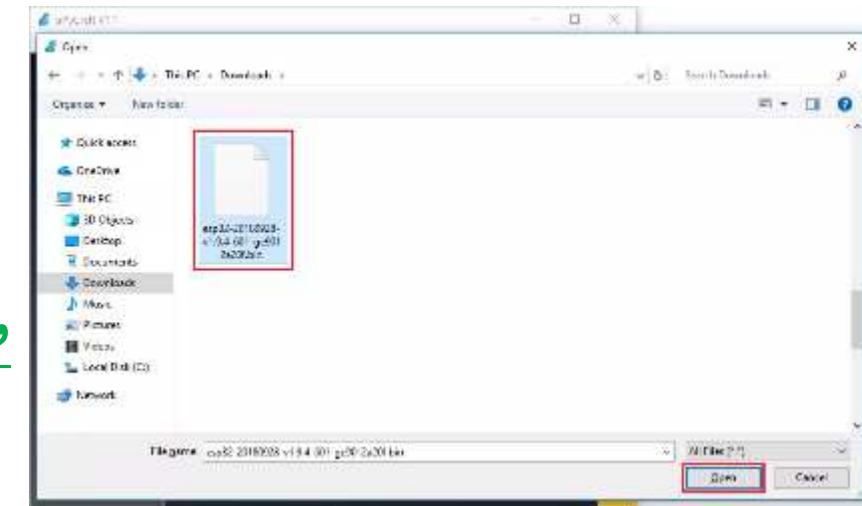
# MicroPython on ESP32

After pressing the “Choose” button, navigate to your Downloads folder and select the ESP32 .bin file:

Having all the settings selected,  
**hold-down the “BOOT/FLASH”  
button in your ESP32 board:**



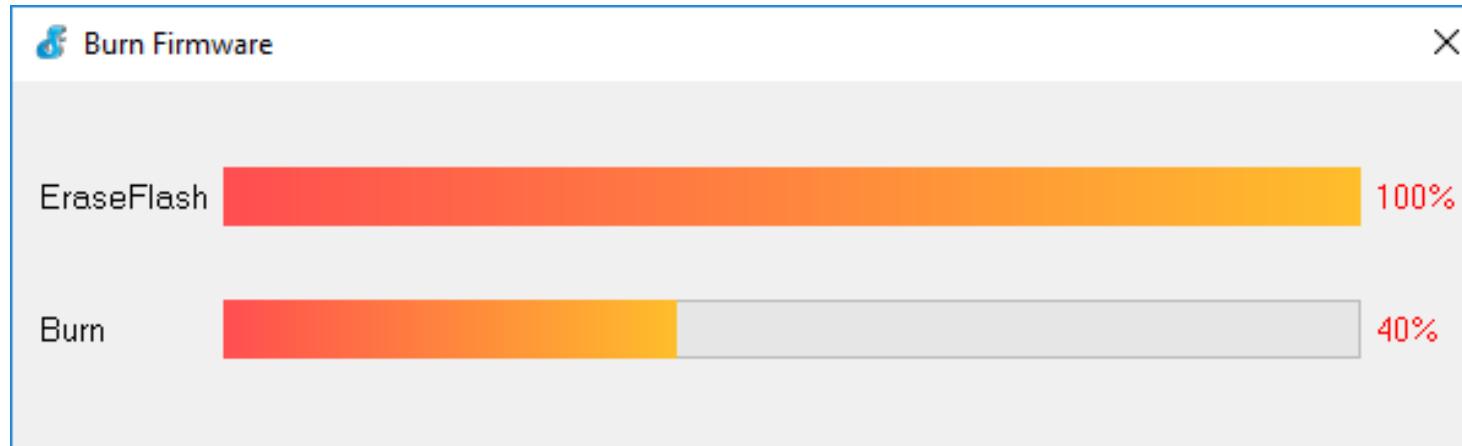
**While holding down the “BOOT/FLASH”, click  
the “ok” button in the burn firmware window:**



# MicroPython on ESP32

When the “**EraseFlash**” process begins, you can release the “**BOOT/FLASH**” button.

After a few seconds, the firmware will be flashed into your ESP32 board.



**Note:** if the “**EraseFlash**” bar doesn’t move and you see an error message saying “**erase false.**”, **it means that your ESP32 wasn’t in flashing mode:**

=> You need to repeat all the steps described earlier and hold the “**BOOT/FLASH**” button again to ensure that your ESP32 goes into flashing mode.

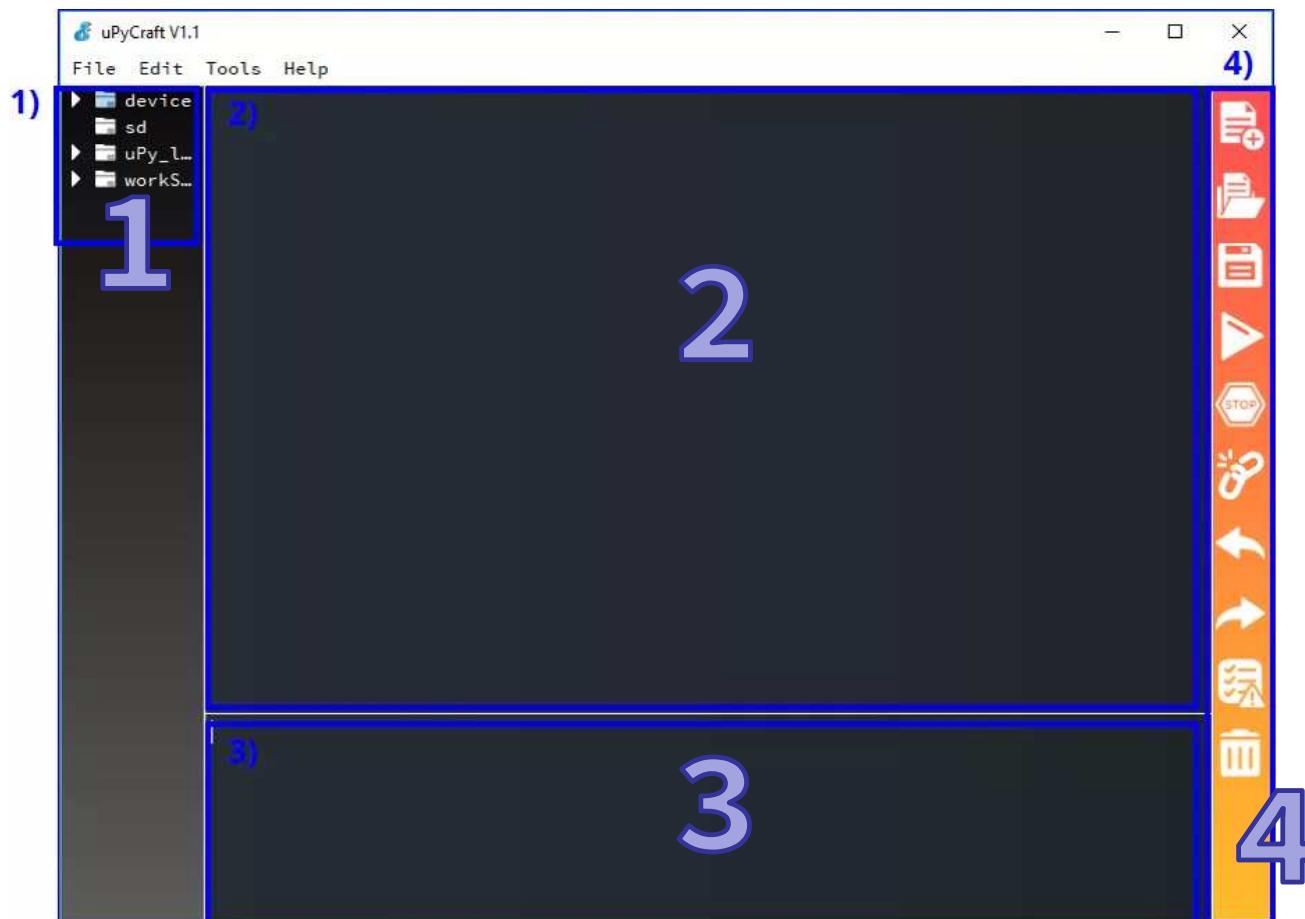


# **GETTING STARTED WITH uPyCraft IDE**

# uPyCraft IDE

There are many ways to program your ESP board with MicroPython.  
We've chosen uPyCraft IDE because it is simple and intuitive to use and works great with the ESP boards.

uPyCraft IDE software, so that you can start programming the ESP32/ESP8266 with MicroPython.



1. Folder and files
2. Editor
3. MicroPython Shell/Terminal
4. Tools

## uPyCraft sections

Several folders and files:

**1 - The device** folder shows the files that are currently stored on your ESP board.  
*If you have your ESP32 or ESP8266 connected via serial to uPyCraft IDE, when you expand the device folder, all files stored should load.*

**By default**, you should only have a **boot.py** file.

To run your main code, it is recommended to create a **main.py** file.

- boot.py**: runs when the device starts and sets up several configuration options;
- main.py**: this is the main script that contains your code. It is executed immediately after the **boot.py**.

**2 - The sd** folder is meant to access files stored on SD cards

- *this is only works with boards like the PyBoard that come with an SD card slot.*

**3 - The uPy\_lib** shows the built-in IDE library files.

**4 - Finally, the workspace** is a directory to save your files.

*These files are saved in your computer in a directory defined by you.*

This is a specially useful to keep all your files organized at hand.

*When using uPyCraft for the first time, to select your working directory, click the **workspace** folder. A new window pops up for you to chose your **workspace** path.*

*Create a new folder or select an existing folder to be your working directory.*

# uPyCraft sections

## 2. Editor

The Editor section is where you write your code and edit your .py files.

**You can open more than one file, and the Editor will open a new tab for each file.**

## 3. MicroPython Shell/terminal

On the MicroPython Shell, you can type commands to be executed immediately by your ESP board without the need to upload new files.

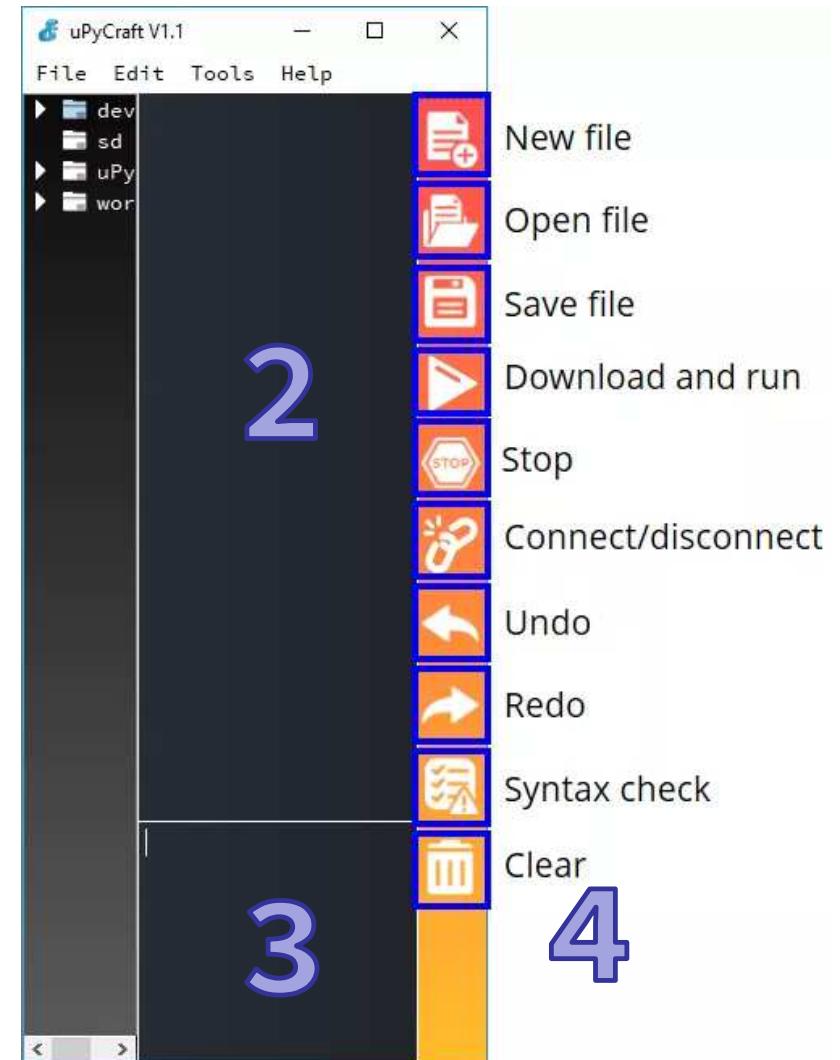
*The terminal also provides information about the state of an executing program, shows errors related with upload, syntax errors, prints messages, etc...*

## 4. Tools

The icons placed at the rightmost side allow you to quickly perform tasks.

**Each button is labeled in the figure below:**

- **New file:** creates a new file on the Editor;
- **Open file:** open a file from your computer;
- **Save file:** saves a file;
- **Download and run:** upload the code to your board and execute the code;
- **Stop:** stop the execution of the code – it's the same as entering CRTL+C on the Shell to stop all scripts from running;
- **Connect/Disconnect:** connect or disconnect to your board via Serial. You must select the serial port first in **Tools > Serial**;
- **Undo:** undo last change in the code Editor;
- **Redo:** redo last change in the code Editor;
- **Syntax check:** checks the syntax of your code;
- **Clear:** clear the Shell/terminal window messages.



# RUNNING YOUR FIRST SCRIPT

## Establishing a communication with the board

After :

- 1 - having the MicroPython firmware installed on your board and
- 2 - having the board connected to your computer through an USB cable,

follow the next steps:

1. Go to **Tools** > **Board** and select the board you're using.
2. Go to **Tools** > **Port** and select the com port your ESP is connected to.
3. Press the **Connect** button to establish a serial communication with your board.



Connect/disconnect

4. The **>>>** should appear in the Shell window after a successful connection with your board.

You can type the **print** command to test if it's working:

```
>>> print('Hello')
Hello
>>>
```

A screenshot of a terminal window with a dark background. It shows the text 'Hello' printed in white. The text 'Hello' is enclosed in a red rectangular box, while the other text ('>>>') is not. A blue arrow points from the code input area to this terminal window.

## CREATING THE *main.py* FILE ON YOUR BOARD

1. Press the “**New file**” button to create a new file.



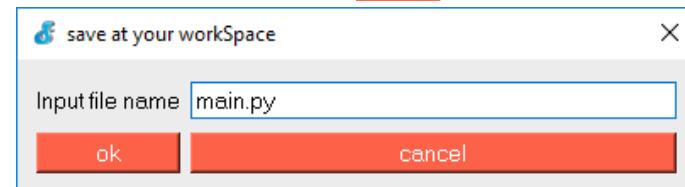
New file

2. Press the “**Save file**” button to save the file in your computer.

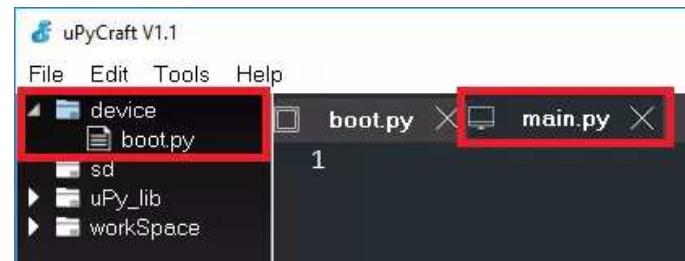


Save file

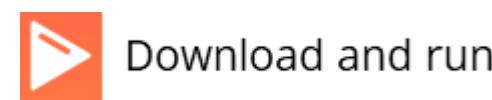
3. A new window opens, **name your file *main.py*** and save it in your PC:



4. After that, you should see the following in your uPyCraft IDE (the *boot.py* file in your device and a new tab with the *main.py* file):



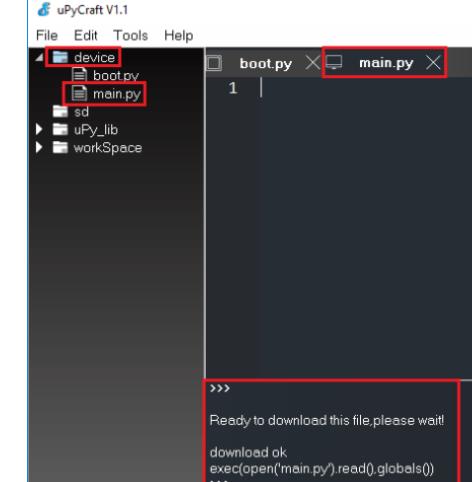
5. Click the “**Download and run**” button to upload the file to your ESP board:



Download and run

6. The device directory should now load the *main.py* file.

**Your ESP has the file *main.py* stored.**



# UPLOADING THE BLINK LED SCRIPT

1. Copy the following code to the Editor on the *main.py* file:

2. Press the “**Stop**” button to stop any script from running in your board:



Stop

3. Click the “**Download and Run button**” to upload the script to the ESP32 or ESP8266:



Download and run

4. You should see a message saying “download ok” in the Shell window:

```
from machine import Pin  
  
from time import sleep  
  
  
led = Pin(2, Pin.OUT)  
  
  
  
  
while True:  
    led.value(not led.value())  
    sleep(0.5)
```

The screenshot shows the uPyCraft V1.1 IDE interface. On the left is a file tree with 'device', 'boot.py', 'main.py', 'sd', 'uPy\_lib', and 'workSpace'. The 'main.py' tab is selected, displaying the following code:

```
1 from machine import Pin  
2 import time  
3 led = Pin(2, Pin.OUT)  
4 while True:  
5     led.value(not led.value())  
6     time.sleep(0.5)
```

In the bottom right pane, there is a terminal window with the message "Ready to download this file,please wait!". Below it, a red box highlights the text "download ok". To the right of the terminal is a vertical toolbar with various icons: a plus sign, a file, a play button, a stop button, a refresh, a back arrow, a forward arrow, a save, and a trash can.

# TESTING THE SCRIPT

To run the script that was just uploaded to your board, you need to follow these steps:

1. Press the “Stop” button:



Stop

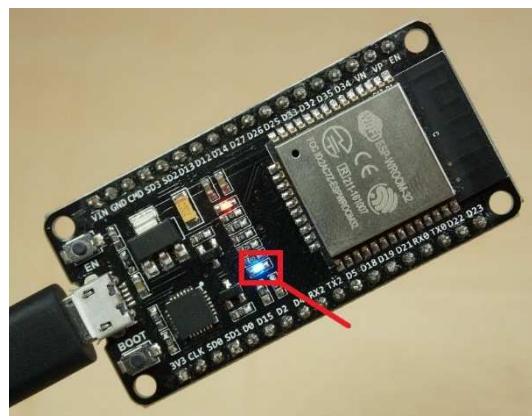
2. Press the on-board ESP32/ESP8266 EN (ENABLE) or RST (RESET) button to restart your board and run the script from the start:



If you're using an ESP32, your Terminal messages should look something as shown in the following figure after a EN/RST button press:

Your ESP32 or ESP8266 on-board **LED should be blinking every 500 milliseconds**.

Here's where the ESP32's on-board LED is located:



```
boot.py main.py
from machine import Pin
import time
led = Pin(2, Pin.OUT)
while True:
    led.value(not led.value())
    time.sleep(0.5)

Ready to download this file, please wait!
download ok
exec(open('main.py').read(),globals())
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<string>", line 8, in <module>
KeyboardInterrupt
>>> ets Jun 8 2016 00:00:00

rst0x1 (POWERON_P
configip: 0, SPIWP:0,
clk_drv:0x0,q_drv:0x0
mode:DIO, clock div:2
load:0x3ff0018,len:4
load:0x3ff001c,len:47
load:0x40078000,len:7
load:0x40080400,len:5512
entry:0x4008114c
[0:32m] (389) cpu_start: Pro cpu up: [0m
[0:32m] (389) cpu_start: Single core mode [0m
[0:32m] (389) heap_init: Initializing. RAM available for dynamic allocation: [0m
[0:32m] (393) heap_init: At 3FFAE80 len 00001920 (6 KiB): DRAM [0m
[0:32m] (399) heap_init: At 3FFC4F48 len 0001B0B8 (108 KiB): DRAM [0m
[0:32m] (405) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/I/RAM [0m
[0:32m] (412) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/I/RAM [0m
```

# TROUBLESHOOTING TIPS

Some common problems and error messages occur with uPyCraft IDE.

**Usually restarting your ESP with the on-board EN/RST button fixes your problem.**

Or pressing the uPyCraft IDE “**Stop**” button and repeating your desired action.

In case it doesn't work for you, see the common errors and discover how to solve them

**Error #1:** You get the following message:

```
>>> Select Serial Port could not open port 'COM4': FileNotFoundError(2, 'The  
system cannot find the file specified.', None, 2)
```

**Unplug, and plug back your ESP board.** Then, double-check that you've selected the right serial port in the **Tools > Serial** menu.

Then, click the “**Connect/disconnect**” button to establish a serial communication.  
You should now be able to upload a new script or re-run new code.

This error might also mean that you have your serial port being used in another program (like a serial terminal or in the Arduino IDE).

Finally, restart the uPyCraft IDE – try to select the serial port in the **Tools > Serial** menu.

**Error #2:** Trouble uploading a new script. already in download mode, please wait.

Press the “**STOP**” button in uPyCraft IDE (1 or 2 times) to make sure any code that was running stops. After that, **press the “Download and run” button to upload the new script to your ESP board.**

# TROUBLESHOOTING TIPS

## Error #3: After uploading a new script, if you see the following messages:

```
>>> Ready to download this file, please wait!
...
download ok
os.listdir('.')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'os' isn't defined
```

```
>>> Ready to download this file, please
      wait!
...
download ok
os.listdir('.')
OSError: [Errno 98]
```

It means the new file was uploaded to your board successfully. You can notice that it printed the “**download ok**” message. **Press the ESP on-board “EN/RST” button to restart your board and re-run the new uploaded script from the beginning**

## Error #4: Problem restarting your ESP board, running a new script or opening the serial port:

```
>>> Brownout detector was triggered
```

The “Brownout detector was triggered” error message means that there’s **some sort of hardware problem**. It’s often related to one of the following issues: **Poor quality USB cable; USB cable is too long; Board with some defect (bad solder joints); Bad computer USB port; Or not enough power provided by the computer USB port.**

**Solution:** try a different shorter USB cable (with data wires), try a different computer USB port or use a USB hub with an external power supply.

**Important:** if you keep having constant problems or weird error messages, we recommend re-flashing your ESP board with the latest version of MicroPython firmware.

## Error #5: When I try to open a serial communication with the ESP32/ESP8266 in uPyCraft IDE, sometimes it prompts the “Burn Firmware” window asking to re-flash the MicroPython firmware.

**Basically, it's happening when you're running a script in your board, sometimes it's busy running that script and performing the tasks. So, you need to try opening the COM port multiple times or restart the ESP to catch it available to establish the serial communication with uPyCraft IDE.**

If you’re running a script that uses Wi-Fi, deep sleep, or it’s doing multiple tasks, I recommend trying 3 or 4 times to establish the communication. If you can’t, I recommend re-flash the ESP with MicroPython firmware.

# MICROPYTHON PROGRAMMING BASICS

# MICROPYTHON PROGRAMMING BASICS

Programming in MicroPython is the same as Python. Just keep in mind that MicroPython is used for constrained devices. So, you must keep your code as simple as possible.

## Mathematical Operators

Operator	Mathematical Operation	
+	Addition	<code>&gt;&gt;&gt; 2+2*9-3</code>
-	Subtraction	17
*	Multiplication	<code>&gt;&gt;&gt; 28594/2312</code>
/	Division	12.36765
//	Division, discarding the decimal point	<code>&gt;&gt;&gt; 214522236/7.5</code>
%	Remainder after division	2.860297e+07

```
>>> 2+2*9-3  
17  
>>> 28594/2312  
12.36765  
>>> 214522236/7.5  
2.860297e+07  
>>> 23//2  
11  
>>> 25%3  
1
```

## Relational Operators

You can make comparisons using relational operators, these compare the values on either sides and show the relation between them.

Operator	Description	
==	Equal to	<code>&gt;&gt;&gt; 2 == 3</code>
!=	Not equal to	False
>	Greater than	<code>&gt;&gt;&gt; 4 == 4</code>
<	Less than	True
>=	Greater than or equal to	<code>&gt;&gt;&gt; 3 &gt; 2</code>
<=	Less than or equal to	True

```
>>> 2 == 3  
False  
>>> 4 == 4  
True  
>>> 3 > 2  
True  
>>> 489808234 != 2223  
True  
>>> 4.5 >= 4.5  
True
```

# MICROPYTHON - VARIABLES

**Assigning Values to Variables :**

**In Python you don't need to declare what type each variable is.**

*If you're used to program your boards using Arduino IDE, you know that you need to declare the type of a variable when creating a new variable.*

*There isn't such thing in Python.*

Variables are simply a storage placeholder for values: number or text.

**To assign a value to a variable you use the equal sign (=), with the variable name on the left and the value on the right.**

For example, to create a variable to hold the GPIO number where an LED is connected to, you can simply type the following:

```
led_pin = 23
```

In the Arduino IDE, you would have something like:

```
const int led_pin =23;
```

As you can see, Python is much simpler than programming in C (in Arduino IDE).

**Note: the names you give variables can't have spaces and are case sensitive, so `led_pin` is different from `LED_PIN` or `Led_Pin`.**

# MICROPYTHON – DATA TYPES

**Data Types:** Variables can store several types of values, not just whole numbers. That's where *data types* come in. A data type is a classification of a value that tells what operations can be done with the value and how it should be stored. The following table shows the data types we'll use most often in our projects.

Data type	Description
<b>int (Int)</b>	<b>Integer (whole number)</b>
<b>float (Float)</b>	<b>Number with a decimal point</b>
<b>str (String)</b>	<b>Set of characters between quotation marks</b>
<b>bool (Boolean)</b>	<b>True or False</b>

*Let's create variables with different data types:*

```
>>> a = 6  
>>> b = 95.32  
>>> c = 'Hello World!'  
>>> d = True
```

*This tells that a is an int (integer). Experiment with the other variables and you should get*

- The first value assigned to *a*, is an **integer**, which is a whole number.
- The *b* variable contains a **float** value, which is a number with a decimal.
- The third value, ‘Hello World!’, is a **string**, **which is a series of characters**.  
*Rem: a string must be put inside single ('Hello World!') or double quotation ("Hello World!") marks.*
- Finally, *d* is a **Boolean**, which is a type that can only take either **True or False**.

There is a function to check the data type of a variable: the **type()** function. This function accepts as argument the variable you want to check the data type.

```
type(variable)
```

# MicroPython – Print() function

## print() Function

The *print()* function prints the message between parentheses into the Shell. This is specially useful in our projects to debug the code and keep track of what's going on.

For example:

```
>>> print('LED is on') LED is on
```

## Comments

Comments in Python start with the hash character (#) and continue to the end of the line. A comment is useful to add “notes” in your program or to tell anyone who reads the program what the script does. This doesn't add any functionality to your program.

For example:

```
# This is just a comment
```

Because in MicroPython we are working under constrained conditions, there are occasions in which you should avoid adding comments to save space on the ESP memory.

# MicroPython – Conditional Statements

## Conditional Statements

To write useful programs, you'll probably need to perform different actions depending on whether a certain condition is True or False.

We're talking about *conditional statements*.

They have the following structure:

```
if <expr1>:  
    <statement1>  
elif <expr2>:  
    <statement2>  
elif <expr3>:  
    <statement3>  
(...)  
else:  
    <statementn>
```

**<expr> is a Boolean expression** and it can be either **True** or **False**.

If it is True, the <statement> right after it is executed.

The <statement> should be **indented** so that Python knows what statement belongs to each expression.

The **elif statement** stands for **else if** and runs only if the first if condition is not True.

The **else statement** only runs if none of the other expressions are True.

There's no limit to the number of elif statements in a program. It's also not necessary to include an else clause, but if there is one, it must come at the end.

In the Arduino IDE, we use {} curly brackets to define code blocks.

With MicroPython, we use indentation.

Additionally, you need to use a colon : after each expression. Contrary to the Arduino IDE, the expression doesn't need to be inside parentheses.

**Important: Python's standard indentation is 4 spaces. In MicroPython indentation should be only 2 spaces to fit more code into the microcontroller memory.**

# MicroPython – While and For loops

Loops allow you to execute a block of code multiple times for as long as a condition is met. **There are two kinds of loops: while and for loops.**

For example, you can print all numbers from 1 to 10 with a **while loop**:

```
number = 1
while number <= 10:
    print(number)
    number = number + 1
```

The code that belongs to the while loop, **indicated by the indentation**, is executed as long as the value in the *number* variable is less than or equal to ( $\leq$ ) 10. In every loop, the current *number* is printed and then 1 is added to it.

You can also print numbers from 1 to 10 using a **for loop**, like this:

```
number = 1
for number in range(1, 11):
    print(number)
```

The **for loop is executed as long as the value in the *number* variable is within the range of 1 and 11.**

The **range()** function automatically assigns the next value to the *number* variable, until 1 below the final number you specify.

Note: - you should use a for loop when you want to repeat a block of code a certain number of times.

- you use a while loop when you want to repeat code until a certain condition is no longer met. In some situations, you can use either one, oftentimes one is more suitable than the others.

**Similar to the conditional statements, the for and while Boolean expressions should have a colon : right after them, and the expressions to be executed should be indented.**

# MicroPython – User-defined Functions

To define a new function, you use the word **def** followed by the name you want to give the function and a set of parentheses (and arguments inside, if necessary).

After the parentheses you add a colon : and then tell the function what instructions to perform.

The statements should be indented with 2 spaces (in MicroPython), for example:

```
def my_function(<arg1>, <arg2>, ...):
    <statement>
    ...
    return
```

For example, a function that converts the temperature in Celsius to Fahrenheit could be the following:

```
def celsius_to_fahrenheit(temp_celsius):
    temp_fahrenheit = temp_celsius * (9/5) + 32
    return temp_fahrenheit
```

The *celsius\_to\_fahrenheit()* function accepts as argument a temperature in Celsius (*temp\_celsius*). Then, it does the calculation to convert the temperature. Finally, it returns the temperature in Fahrenheit (*temp\_fahrenheit*).

**Note: functions don't necessarily need to return something. They could just perform some work without the need to return anything.**

# MicroPython – Class

## Classes and Objects

Python is an Object-Oriented Programming (OOP) language.

There are two important concepts you need to understand about OOP:

- **classes and objects.**

A class is a blueprint for objects. It defines a set of attributes (data and functions) that characterize an object.

The functions inside of a class are called methods.

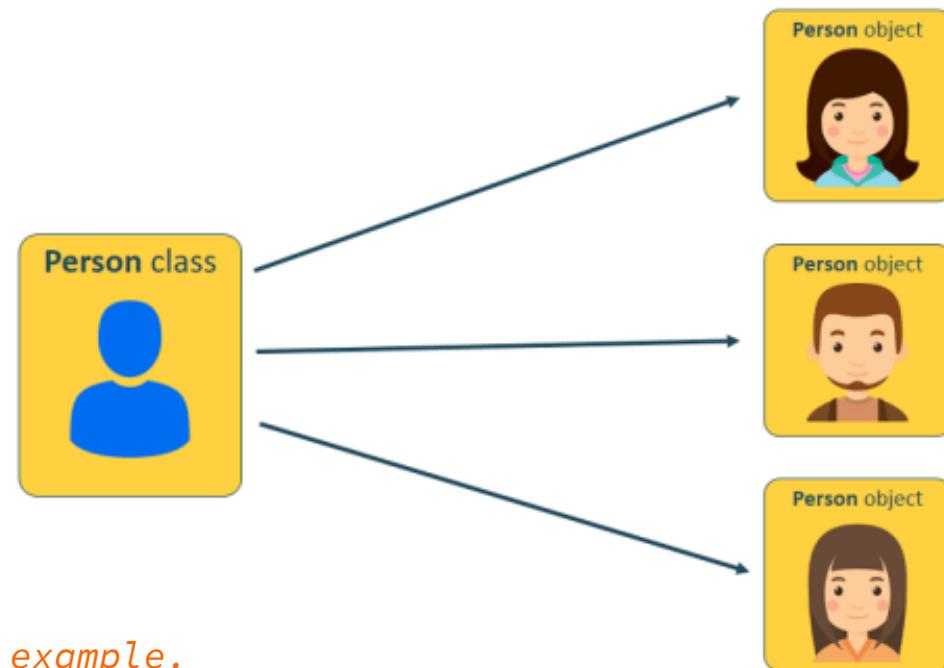
Classes are defined by the keyword **class** followed by the name of the class.

For example:

```
class MyClass:  
    (...)
```

### Note:

- by convention, classes' names in Python should be CapWords. However, you can give whatever name you want.
- An object is an instance of a class. It's simply a collection of data and methods into a single entity. Through the object, you can use all functionalities of its class.



Confused? Let's take a look at a simple example.

If we would like to define several persons in a Python program using the same attributes, we can think of the term person as a class.

We may want to define a person using attributes like name, age, country, etc.

# MicroPython – Class

So, we can create a class called *Person*.

Our class will have the following attributes: *name*, *age*, and *country*.

You can add as many attributes as you want. We'll also create a function (**method**) that prints a description of the person based on its attributes:

```
class Person:  
    name = ""  
    age = 0  
    country = ""  
    def description(self):  
        print("%s is %d years old and from %s." %(self.name, self.age, self.country))
```

As you can see, we define a new class by using the keyword **class**, followed by the name we want to give to the class.

Inside the **Person class**, we define several variables to hold values.

By default the *name* and *country* are empty strings, and the *age is 0*.

Then, we also define a function (**method**) that prints all the variables values into the Shell.

All functions inside a class should have the *self* parameter as argument and other arguments, if needed.

The *self* parameter refers to the object itself.

It is used to access variables that belong to the class.

For example, to access the *name* variable inside the class, we should use *self.name*.

Now that we've create a class, we can create as many *Person* objects as we want by using that class. The *Person* object will have a *name*, *age*, and *country*.

We'll also be able to print its description using the *description()* method.

For example, to create a new *Person* object called *person1*:

```
>>> person1 = Person()
```

# MicroPython – Class

## Set the object's properties

To set the *name*, *age*, and *country* of the *person1* object. You can do it as follows:

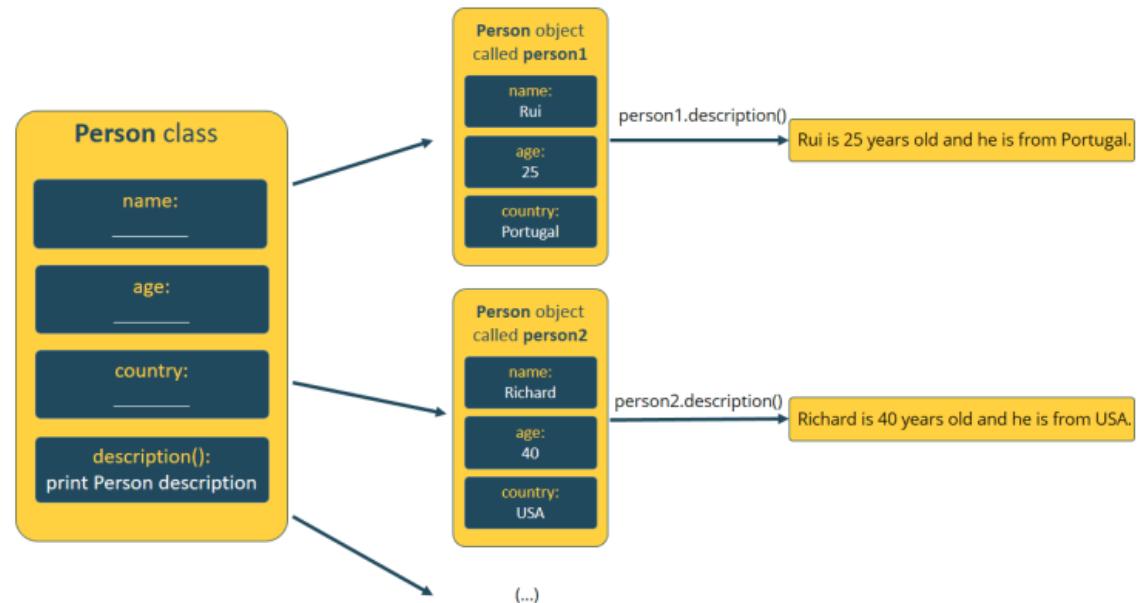
```
>>> person1.name = "Rui"  
>>> person1.age = 25  
>>> person1.country = "Portugal"
```

## Calling methods

Later in your code you can use the created *description()* method on any *Person* object. To call the *description()* method on the *person1* object:

```
>>> person1.description()  
Rui is 25 years old and he is from Portugal.
```

You should now understand that you can create as many objects as you want using the same class and you are able to use the available methods with all the objects of that class.



# MicroPython – Class – Constructor methods

Instead of having to define a class, and then set the object's properties, which can be time consuming, **you can use the constructor method inside your class.**

**The constructor method is used to initiate data as soon as an object of a class is instantiated.**

The constructor method is also known as **`__init__`** method.

**Using the `__init__` method, the `Person()` class looks as follows:**

```
class Person():
    def __init__(self, name, age, country):
        self.name = name
        self.age = age
        self.country = country
    def description(self):
        print("%s is %d years old and he is from %s." %(self.name,
self.age, self.country))
```

Then, to instantiate a `Person` object with the same attributes we've defined earlier, we just need to do the following:

```
>>> person1 = Person("Rui", 25, "Portugal")
```

If you call the `description()` on the `person1` object, you'll get the same result:

```
>>> person1.description()
Rui is 25 years old and he is from Portugal.
```

# MicroPython – Modules

A module is a file that contains a set of classes and functions you can use in your code – you can also call it library.

To access the classes and functions inside that code, you just need to import that module into your code. You can create your own modules, or use already created modules from the standard Python library.

When it comes to MicroPython, it only comes with a small subset of the standard Python library, but it does come with a set of modules to control GPIOs, make networks connections and much more.

Importing modules/libraries is as simple as using:

```
import module_name
```

For example, to import the **machine** library that contains classes to control GPIOs, type the following:

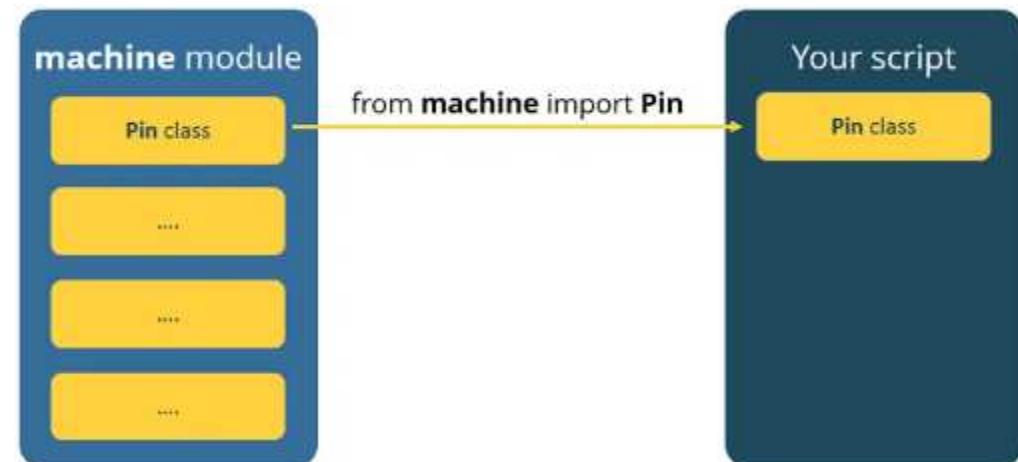
```
import machine
```

In most programs you won't need all the classes from one module.

You may just want to import a single class.

For example, to import only the *Pin* class from the *machine* module:

```
from machine import pins
```



## MicroPython – Wrapping Up

We've just scratched the surface of Python basics that also apply to MicroPython.

If you are used to program electronics using the Arduino IDE, you'll find that MicroPython has a much simpler and user-friendly syntax.

**Let's just summarize some of the main differences between your Arduino sketches and programs in MicroPython:**

- You don't use semicolon ; at the end of a statement
- After **Boolean** expressions in conditional statements and loops, you use a colon:
- To define code blocks use **indentation instead of curly brackets {}**
- **When creating a variable, you don't need to define which data type it is – you don't need to declare a variable**
- **Indentation in MicroPython is 2 spaces (instead of 4 in Python)**

# MICROPYTHON MODULES

# MICROPYTHON MODULES

## Python standard libraries and micro-libraries

- `Builtin functions and exceptions`
- `cmath` – mathematical functions for complex numbers
- `gc` – control the garbage collector
- `math` – mathematical functions
- `sys` – system specific functions
- `uarray` – arrays of numeric data
- `ubinascii` – binary/ASCII conversions
- `ucollections` – collection and container types
- `uerrno` – system error codes
- `uhashlib` – hashing algorithms
- `uheapq` – heap queue algorithm
- `uio` – input/output streams
- `ujson` – JSON encoding and decoding
- `uos` – basic “operating system” services
- `ure` – simple regular expressions
- `uselect` – wait for events on a set of streams
- `usocket` – socket module
- `ussl` – SSL/TLS module
- `ustruct` – pack and unpack primitive data types
- `utime` – time related functions
- `uzlib` – zlib decompression
- `_thread` – multithreading support

<http://docs.micropython.org/en/latest/library/machine.html>

<http://docs.micropython.org/en/latest/library/index.html#micropython-specific-libraries>

## MicroPython-specific libraries

- `btree` – simple BTree database
- `framebuf` – Frame buffer manipulation
- `machine` – functions related to the hardware
- `micropython` – access and control MicroPython internals
- `network` – network configuration
- `ubluetooth` – low-level Bluetooth
- `ucryptolib` – cryptographic ciphers
- `uctypes` – access binary data in a structured way

## Libraries specific to the ESP8266 and ESP32

The following libraries are specific to the ESP8266 and ESP32.

- `esp` – functions related to the ESP8266 and ESP32
  - Functions
- `esp32` – functionality specific to the ESP32
  - Functions
  - Flash partitions
  - RMT
  - The Ultra-Low-Power co-processor
  - Constants

## Libraries specific to the WiPy

The following libraries and classes are specific to the WiPy.

- `wipy` – WiPy specific features
  - Functions
- class ADCWiPy – analog to digital conversion
  - Constructors
  - Methods
- class ADCChannel – read analog values from internal or external sources
- class TimerWiPy – control hardware timers
  - Constructors
  - Methods
- class TimerChannel – setup a channel for a timer
  - Methods
  - Constants

# MICROPYTHON MODULES: machine

## machine – functions related to the hardware

The `machine` module contains specific functions related to the hardware on a particular board. Most functions in this module allow to achieve direct and unrestricted access to and control of hardware blocks on a system (like CPU, timers, buses, etc.). Used incorrectly, this can lead to malfunction, lockups, crashes of your board, and in extreme cases, hardware damage.

A note of callbacks used by functions and class methods of `machine` module: all these callbacks should be considered as executing in an interrupt context. This is true for both physical devices with IDs  $\geq 0$  and "virtual" devices with negative IDs like -1 (these "virtual" devices are still thin shims on top of real hardware and real hardware interrupts). See [Writing interrupt handlers](#).

### Reset related functions

#### `machine.reset()`

Resets the device in a manner similar to pushing the external RESET button.

#### `machine.reset_cause()`

Get the reset cause. See [constants](#) for the possible return values.

### Interrupt related functions

#### `machine.disable_irq()`

Disable interrupt requests. Returns the previous IRQ state which should be considered an opaque value. This return value should be passed to the `enable_irq()` function to restore interrupts to their original state, before `disable_irq()` was called.

#### `machine.enable_irq(state)`

## Classes

### [•class Pin – control I/O pins](#)

### [•class Signal – control and sense external I/O devices](#)

### [•class ADC – analog to digital conversion](#)

### [•class UART – duplex serial communication bus](#)

### [•class SPI – a Serial Peripheral Interface bus protocol \(master side\)](#)

### [•class I2C – a two-wire serial protocol](#)

### [•class RTC – real time clock](#)

### [•class Timer – control hardware timers](#)

### [•class WDT – watchdog timer](#)

### [•class SD – secure digital memory card \(cc3200 port only\)](#)

### [•class SDCard – secure digital memory card](#)

<http://docs.micropython.org/en/latest/library/machine.Pin.html>

## class Pin – control I/O pins

A pin object is used to control I/O pins (also known as GPIO - general-purpose input/output). Pin objects are commonly associated with a physical pin that can drive an output voltage and read input voltages. The pin class has methods to set the mode of the pin (IN, OUT, etc) and methods to get and set the digital logic level. For analog control of a pin, see the `ADC` class.

A pin object is constructed by using an identifier which unambiguously specifies a certain I/O pin. The allowed forms of the identifier and the physical pin that the identifier maps to are port-specific. Possibilities for the identifier are an integer, a string or a tuple with port and pin number.

### Usage Model:

```
from machine import Pin

# create an output pin on pin #0
p0 = Pin(0, Pin.OUT)

# set the value low then high
p0.value(0)
p0.value(1)

# create an input pin on pin #2, with a pull up resistor
p2 = Pin(2, Pin.IN, Pin.PULL_UP)

# read and print the pin value
print(p2.value())

# reconfigure pin #0 in input mode
p0.mode(p0.IN)

# configure an irq callback
p0.irq(lambda p:print(p))
```

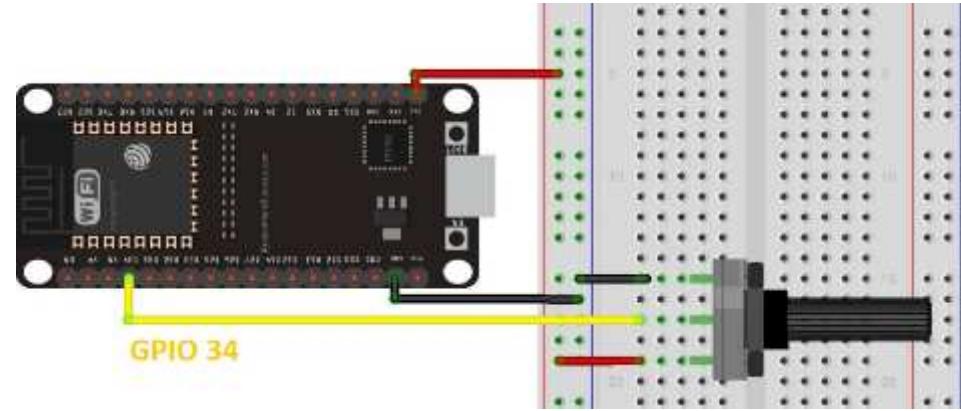
<http://docs.micropython.org/en/latest/esp32/quickref.html>

## TP8 - 1 – ANALOG READ ON ESP32

# AnalogRead with MicroPython

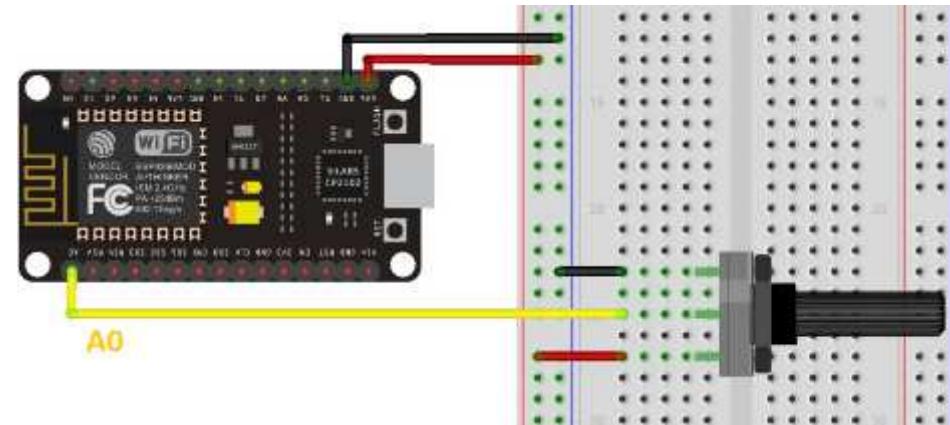
The following script for the ESP32 reads analog values from **GPIO 34**

```
from machine import Pin, ADC  
  
from time import sleep  
  
  
pot = ADC(Pin(34))  
pot.atten(ADC.ATTN_11DB)      #Full range: 3.3v  
  
  
while True:  
    pot_value = pot.read()  
    print(pot_value)  
    sleep(0.1)
```



The ESP8266 supports analog reading only on the **A0** pin.

```
from machine import Pin, ADC  
  
from time import sleep  
  
  
pot = ADC(0)  
  
  
while True:  
    pot_value = pot.read()  
    print(pot_value)  
    sleep(0.1)
```



Pinout Reference: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>

<https://randomnerdtutorials.com/esp32-esp8266-analog-readings-micropython/>

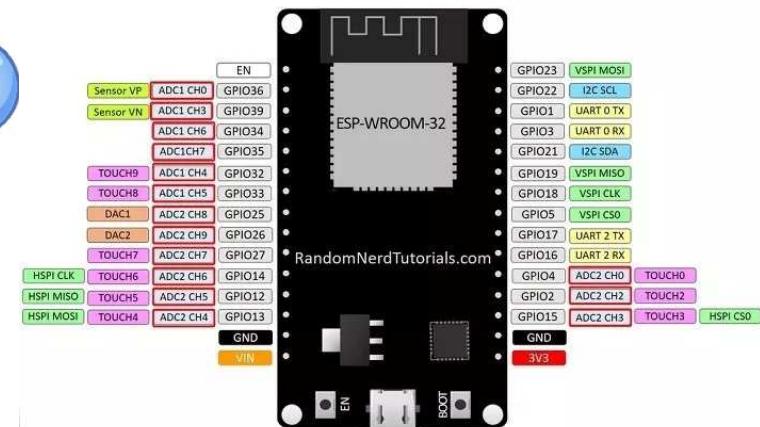
# AnalogRead with MicroPython

## Analog Readings - ESP32

There are several pins on the ESP32 that can act as analog pins - these are called ADC pins. All the following GPIOs can act as ADC pins: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36, and 39.



## ESP32 DEVKIT V1 - DOIT



**ESP32 ADC pins have 12-bit resolution by default.** These pins read voltage between 0 and 3.3V and then return a value **between 0 and 4095**. The resolution can be changed on the code. For example, you may want to have **just 10-bit resolution to get a value between 0 and 1023**.

The following table shows some differences between analog reading on the ESP8266 and the ESP32:

	ESP8266	ESP32
<b>Analog pins</b>	A0 (ADC 0)	<b>GPIOs: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36, and 39.</b>
<b>Resolution</b>	<b>10-bit (0-1023)</b>	<b>12-bit (0-4095)</b>
<b>Change resolution</b>	No	Yes

# AnalogRead with MicroPython

## How the code works

To read analog inputs, import the ADC class in addition to the Pin class from the machine module. We also import the sleep method.

```
from machine import Pin, ADC  
from time import sleep
```

Then, create an ADC object called pot on **GPIO 34**

```
pot = ADC(Pin(34))
```

The following line defines that we want to be able to read voltage in full range.

```
pot.atten(ADC.ATTN_11DB)
```

This means we want to read voltage from **0 to 3.3V**.

This corresponds to setting the attenuation ratio of 11db.

For that, we use the **atten()** method and pass as argument: **ADC.ATTN\_11DB**.

The **atten()** method can take the following arguments:

- **ADC.ATTN\_0DB** – the full range voltage: 1.2V
- **ADC.ATTN\_2\_5DB** – the full range voltage: 1.5V
- **ADC.ATTN\_6DB** – the full range voltage: 2.0V
- **ADC.ATTN\_11DB** – the full range voltage: 3.3V

# AnalogRead with MicroPython

In the **while** loop, read the pot value and save it in the **pot\_value** variable.  
To read the value from the pot, simply use the **read()** method on the **pot** object.  
Then, print the pot value  
At the end, **add a delay of 100 ms.**

```
pot_value = pot.read()  
print(pot_value)  
sleep(0.1)
```

When you rotate the potentiometer, you get values from 0 to 4095 – that's because the ADC pins have a 12-bit resolution by default. You may want to get values in other ranges. **You can change the resolution using the width() method as follows:**

```
ADC.width(bit)
```

The bit argument can be one of the following parameters:

- **ADC.WIDTH\_9BIT : range 0 to 511**
- **ADC.WIDTH\_10BIT: range 0 to 1023**
- **ADC.WIDTH\_11BIT: range 0 to 2047**
- **ADC.WIDTH\_12BIT: range 0 to 4095**

```
ADC.width(ADC.WIDTH_12BIT)
```

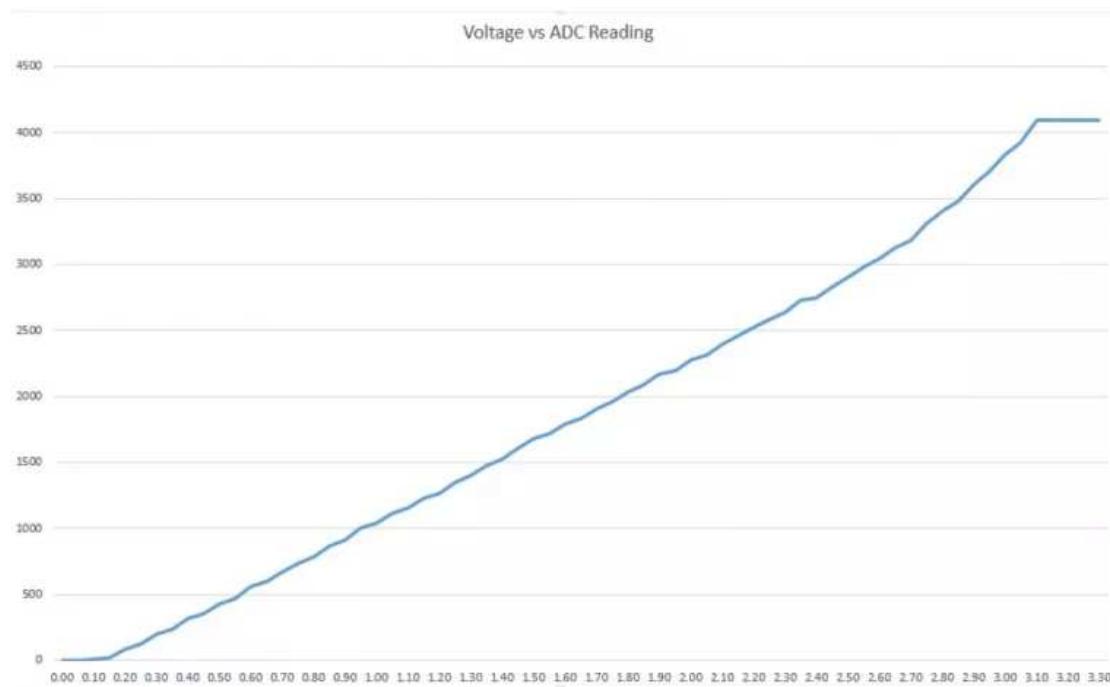
## In summary:

- To read an analog value you need to import the ADC class;
- To create an ADC object simply use **ADC(Pin(GPIO))**, in which GPIO is the number of the GPIO you want to read the analog values;
- To read the analog value, simply use the **read()** method on the ADC object.

## TIPS

### ADC is Non-linear

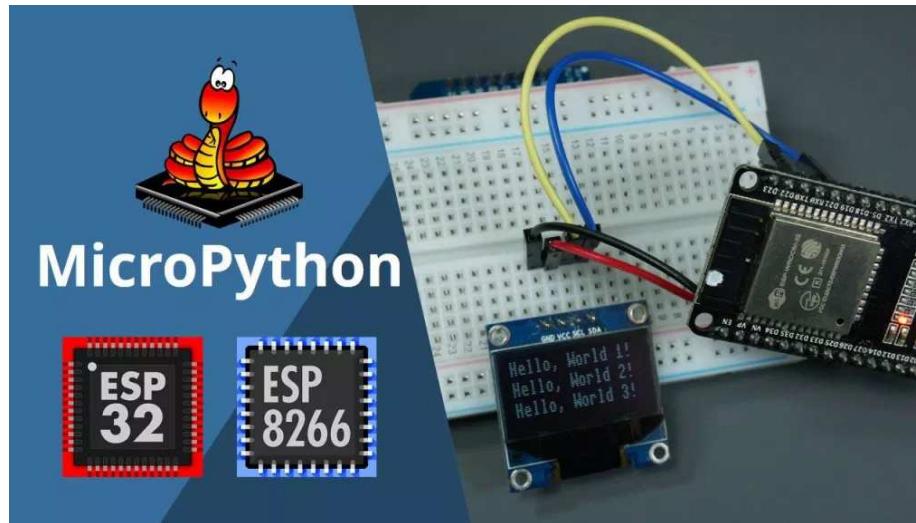
**Ideally, you would expect a linear behavior when using the ESP32 ADC pins.** However, that doesn't happen. What you'll get is a behavior as shown in the following chart:



Note: ADC2 pins cannot be used when Wi-Fi is used. So, if you're using Wi-Fi and you're having trouble getting the value from an ADC2 GPIO, you may consider using an ADC1 GPIO instead, that should solve your problem.

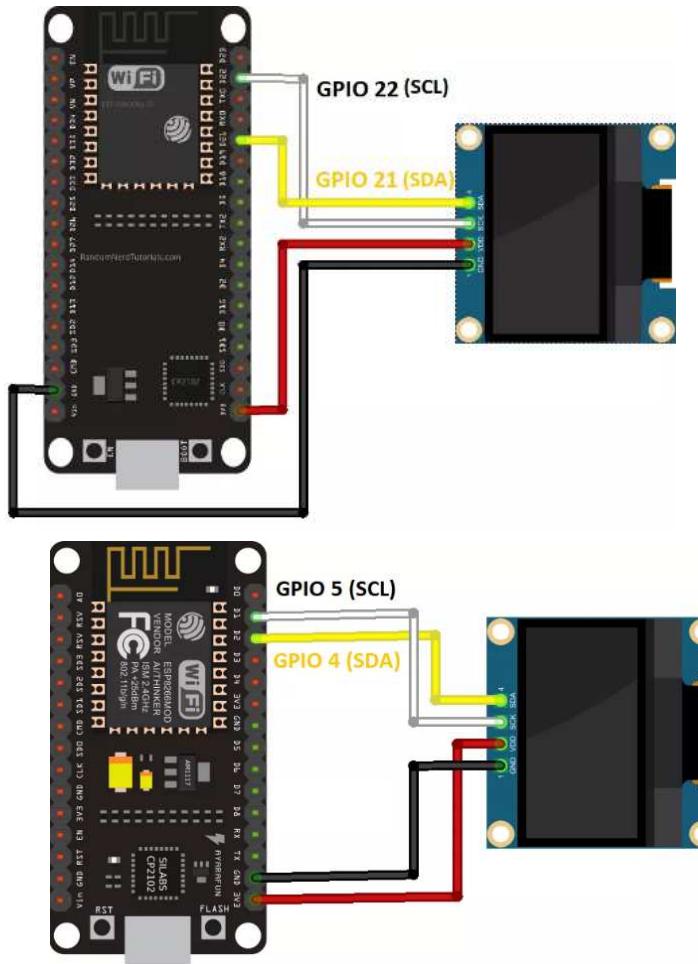
<https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/>

## TP8 - 2 – OLED ON ESP32



# MicroPython: OLED Display with ESP32 and ESP8266

In this guide we'll use the [0.96 inch SSD1306 OLED display](#) that is 128x64 pixels and uses I2C communication protocol.



## I2C communication

For the I2C OLED display, these are the connections you need to make:

OLED	ESP32	ESP8266
Vin	3.3V	3.3V
GND	GND	GND
SCL	GPIO 22	GPIO 5 (D1)
SDA	GPIO 21	GPIO 4 (D2)

<https://randomnerdtutorials.com/micropython-oled-display-esp32-esp8266/>

<https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/>

# SSD1306 OLED Library – Part 1

```
#MicroPython SSD1306 OLED driver, I2C and SPI interfaces created by Adafruit
import time
import framebuf
```

```
# register definitions
SET_CONTRAST      = const(0x81)
SET_ENTIRE_ON     = const(0xa4)
SET_NORM_INV       = const(0xa6)
SET_DISP           = const(0xae)
SET_MEM_ADDR      = const(0x20)
SET_COL_ADDR      = const(0x21)
SET_PAGE_ADDR     = const(0x22)
SET_DISP_START_LINE= const(0x40)
SET_SEG_REMAP      = const(0xa0)
SET_MUX_RATIO     = const(0xa8)
SET_COM_OUT_DIR   = const(0xc0)
SET_DISP_OFFSET    = const(0xd3)
SET_COM_PIN_CFG   = const(0xda)
SET_DISP_CLK_DIV  = const(0xd5)
SET_PRECHARGE     = const(0xd9)
SET_VCOM_DESEL    = const(0xdb)
SET_CHARGE_PUMP   = const(0x8d)
```

```
class SSD1306:
    def __init__(self, width, height, external_vcc):
        self.width = width
        self.height = height
        self.external_vcc = external_vcc
        self.pages = self.height // 8
        # Note the subclass must initialize self.framebuf to a framebuffer.
        # This is necessary because the underlying data buffer is different
        # between I2C and SPI implementations (I2C needs an extra byte).
        self.poweron()
        self.init_display()

    def show(self):
```

```
        x0 = 0
        x1 = self.width - 1
        if self.width == 64:
            # displays with width of 64 pixels are shifted by 32
            x0 += 32
            x1 += 32
        self.write_cmd(SET_COL_ADDR)
        self.write_cmd(x0)
        self.write_cmd(x1)
        self.write_cmd(SET_PAGE_ADDR)
        self.write_cmd(0)
        self.write_cmd(self.pages - 1)
        self.write_framebuf()
```

```
def init_display(self):
    for cmd in (
        SET_DISP | 0x00, # off
        # address setting
        SET_MEM_ADDR, 0x00, # horizontal
        # resolution and layout
        SET_DISP_START_LINE | 0x00,
        SET_SEG_REMAP | 0x01, # column addr 127 mapped to SEG0
        SET_MUX_RATIO, self.height - 1,
        SET_COM_OUT_DIR | 0x08, # scan from COM[N] to COM0
        SET_DISP_OFFSET, 0x00,
        SET_COM_PIN_CFG, 0x02 if self.height == 32 else 0x12,
        # timing and driving scheme
        SET_DISP_CLK_DIV, 0x80,
        SET_PRECHARGE, 0x22 if self.external_vcc else 0xf1,
        SET_VCOM_DESEL, 0x30, # 0.83*Vcc
        # display
        SET_CONTRAST, 0xff, # maximum
        SET_ENTIRE_ON, # output follows RAM contents
        SET_NORM_INV, # not inverted
        # charge pump
        SET_CHARGE_PUMP, 0x10 if self.external_vcc else 0x14,
        SET_DISP | 0x01): # on
            self.write_cmd(cmd)
    self.fill(0)
    self.show()
```

```
def poweroff(self):
    self.write_cmd(SET_DISP | 0x00)
```

```
def contrast(self, contrast):
    self.write_cmd(SET_CONTRAST)
    self.write_cmd(contrast)
```

```
def invert(self, invert):
    self.write_cmd(SET_NORM_INV | (invert & 1))
```

## SSD1306 OLED Library – Part 2

```
def fill(self, col):
    self.framebuf.fill(col)

def pixel(self, x, y, col):
    self.framebuf.pixel(x, y, col)

def scroll(self, dx, dy):
    self.framebuf.scroll(dx, dy)

def text(self, string, x, y, col=1):
    self.framebuf.text(string, x, y, col)

class SSD1306_I2C(SSD1306):
    def __init__(self, width, height, i2c, addr=0x3c, external_vcc=False):
        self.i2c = i2c
        self.addr = addr
        self.temp = bytearray(2)
        # Add an extra byte to the data buffer to hold an I2C data/command byte
        # to use hardware-compatible I2C transactions. A memoryview of the
        # buffer is used to mask this byte from the framebuffer operations
        # (without a major memory hit as memoryview doesn't copy to a separate
        # buffer).
        self.buffer = bytearray((height // 8) * width + 1)
        self.buffer[0] = 0x40 # Set first byte of data buffer to Co=0, D/C=1
        self.framebuf = framebuffer.FrameBuffer1(memoryview(self.buffer)[1:], width, height)
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.temp[0] = 0x80 # Co=1, D/C#=0
        self.temp[1] = cmd
        self.i2c.writeto(self.addr, self.temp)

    def write_framebuf(self):
        # Blast out the frame buffer using a single I2C transaction to support
        # hardware I2C interfaces.
        self.i2c.writeto(self.addr, self.buffer)

    def poweron(self):
        pass

class SSD1306_SPI(SSD1306):
    def __init__(self, width, height, spi, dc, res, cs, external_vcc=False):
        self.rate = 10 * 1024 * 1024
        dc.init(dc.OUT, value=0)
        res.init(res.OUT, value=0)
        cs.init(cs.OUT, value=1)
        self.spi = spi
        self.dc = dc
        self.res = res
        self.cs = cs
        self.buffer = bytearray((height // 8) * width)
        self.framebuf = framebuffer.FrameBuffer1(self.buffer, width, height)
        super().__init__(width, height, external_vcc)

    def write_cmd(self, cmd):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs.high()
        self.dc.low()
        self.cs.low()
        self.spi.write(bytearray([cmd]))
        self.cs.high()

    def write_framebuf(self):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs.high()
        self.dc.high()
        self.cs.low()
        self.spi.write(self.buffer)
        self.cs.high()

    def poweron(self):
        self.res.high()
        time.sleep_ms(1)
        self.res.low()
        time.sleep_ms(10)
        self.res.high()
```

## A. Upload OLED library with uPyCraft IDE

How to upload a library using uPyCraft IDE ?

1. Create a new file by pressing the **New File** button.

2. Copy the OLED library code into that file.

The [OLED library code can be found here](#).

**Note:** the SSD1306 OLED display library was built by Adafruit and will no longer be updated. At the moment, it works fine.

3. After copying the code, save the file by pressing the **Save** button.

4. Call this new file “**ssd1306.py**” and press **ok**.

5. Click the **Download and Run** button.



uPyCraft V1.1

File Edit Tools Help

device

boot

sd

uPy\_L1b

workspace

untitled.x

```
1 # MicroPython SSD1306 OLED driver, I2C and SPI interfaces
2
3 import time
4 import framebuf
5
6
7 # register definitions
8 SET_CONTRAST = const(0x81)
9 SET_ENTIRE_ON = const(0xa4)
10 SET_NORM_INV = const(0xa0)
11 SET_DISP = const(0xae)
12 SET_MEM_ADDR = const(0x20)
13 SET_COL_ADDR = const(0x21)
14 SET_PAGE_ADDR = const(0x22)
15 SET_DISP_START_LINE = const(0x40)
16 SET_SEG_REMAP = const(0xa0)
17 SET_MUX_RATIO = const(0x80)
18 SET_COM_OUT_DIR = const(0xc0)
19 SET_DISP_OFFSET = const(0xd3)
20 SET_COM_PIN_CFG = const(0xda)
21 SET_DISP_CLK_DIV = const(0xd5)
22 SET_PRECHARGE = const(0xd9)
23 SET_VCOM_DESEL = const(0xdb)
```

save at your workspace

Input file name: ssd1306.py

ok cancel

uPyCraft V1.1

File Edit Tools Help

device

boot.py

sd

uPy\_L1b

workspace

ssd1306.py

```
1 # MicroPython SSD1306 OLED driver, I2C and SPI interfaces
2
3 import time
4 import framebuf
5
6
7 # register definitions
8 SET_CONTRAST = const(0x81)
9 SET_ENTIRE_ON = const(0xa4)
10 SET_NORM_INV = const(0xa0)
11 SET_DISP = const(0xae)
12 SET_MEM_ADDR = const(0x20)
13 SET_COL_ADDR = const(0x21)
14 SET_PAGE_ADDR = const(0x22)
15 SET_DISP_START_LINE = const(0x40)
16 SET_SEG_REMAP = const(0xa0)
17 SET_MUX_RATIO = const(0x80)
18 SET_COM_OUT_DIR = const(0xc0)
19 SET_DISP_OFFSET = const(0xd3)
20 SET_COM_PIN_CFG = const(0xda)
21 SET_DISP_CLK_DIV = const(0xd5)
22 SET_PRECHARGE = const(0xd9)
23 SET_VCOM_DESEL = const(0xdb)

>>>
>>>

Ready to download this file, please wait!
```

download ok

exec(open('ssd1306.py').read(),globals())

>>>

The file should be saved on the device folder with the name “**ssd1306.py**” as highlighted in the following figure.

Now, you can use the library functionalities in your code by importing the library.

# How the OLED code works

Start by importing the necessary modules to interact with the GPIOs and send data to the OLED via I2C communication.

You need to import the Pin and I2C classes from the machine module AND also the OLED library that you previously uploaded to the board as `ssd1306.py` file.

The ESP32 default I2C pins are **GPIO 22 (SCL)** and **GPIO 21 (SDA)**.

The ESP8266 default I2C pins are **GPIO 5 (SCL)** and **GPIO 4 (SDA)**.

Define the OLED width and height on the following variables:

After that, create an `SSD1306_I2C` object called `oled`.

This object accepts the OLED width, height, and the I2C pins you've defined earlier.

After initializing the OLED display, you just need to use the `text()` function on the `oled` object to write text. After the `text()` function, you need to call the `show()` method to update the OLED.

The `text()` method accepts the following arguments:

- **Message:** must be of type String
- **X position:** where the text starts
- **Y position:** where the text is displayed vertically
- **Text color:** it can be either black or white. The default color is white and this parameter is optional:  
    0 = black / 1 = white

For example, the following line writes the 'Hello, World 1!' message in white color. The text starts on x = 0 and y = 0.

The next line of code writes the text on the next line (y =10).

Finally, for the changes to take effect, use the `show()` method on the `oled` object.

```
from machine import Pin, I2C
import ssd1306
from time import sleep

# ESP32 Pin assignment
i2c = I2C(-1, scl=Pin(22), sda=Pin(21))

# ESP8266 Pin assignment
#i2c = I2C(-1, scl=Pin(5), sda=Pin(4))

oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

oled.text('Hello, World 1!', 0, 0)
oled.text('Hello, World 2!', 0, 10)
oled.text('Hello, World 3!', 0, 20)

oled.show()
```

# Other OLED functions

## Fill the screen

To fill the entire screen with white, use the `fill()` function as follows:

```
oled.fill(1)  
oled.show()
```

```
oled.fill(0)  
oled.show()
```

To clear the screen use the `fill()` method as pass 0 as argument. (Sets all pixels to black):

## Draw a pixel

To draw a pixel, use the `pixel()` method, followed by the `show()` method. The `pixel()` method accepts the following arguments:

- **X coordinate:** pixel location horizontally
- **Y coordinate:** pixel location vertically
- **Pixel color:** can be black or white
  - 0 = black
  - 1 = white

For example, to draw a white pixel on the top left corner:

```
oled.pixel(0,0,1)  
oled.show()
```



## Invert colors

You can also invert the OLED colors: white with black and vice versa, using the `invert()` method:

To get back to the original colors, use:

```
oled.invert(False)  
oled.show()
```

# DISPLAYING DATA FROM SENSORS

The `text()` function only accepts variables of type `String` as a message. Sensor readings are usually stored in `int` or `float` variables.

If you want to display sensor readings and they are stored in `int` or `float` variables, they should be converted to a `String`. To convert the data to a string you can use the `str()` function:

```
temperature = 12.34
temperature_string = str(temperature)
oled.text(temperature_string)
oled.show()
```

Then, you can display the `temperature_string` variable on the OLED using the `text()` and `show()` methods:

```
from machine import Pin, I2C, ADC
import ssd1306
from time import sleep

# ESP32 Pin assignment
i2c = I2C(-1, scl=Pin(22), sda=Pin(21))

# ESP8266 Pin assignment
#i2c = I2C(-1, scl=Pin(5), sda=Pin(4))

pot= ADC(Pin(34))
pot.atten(ADC.ATTN_11DB)  #Full range 3.3V

oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

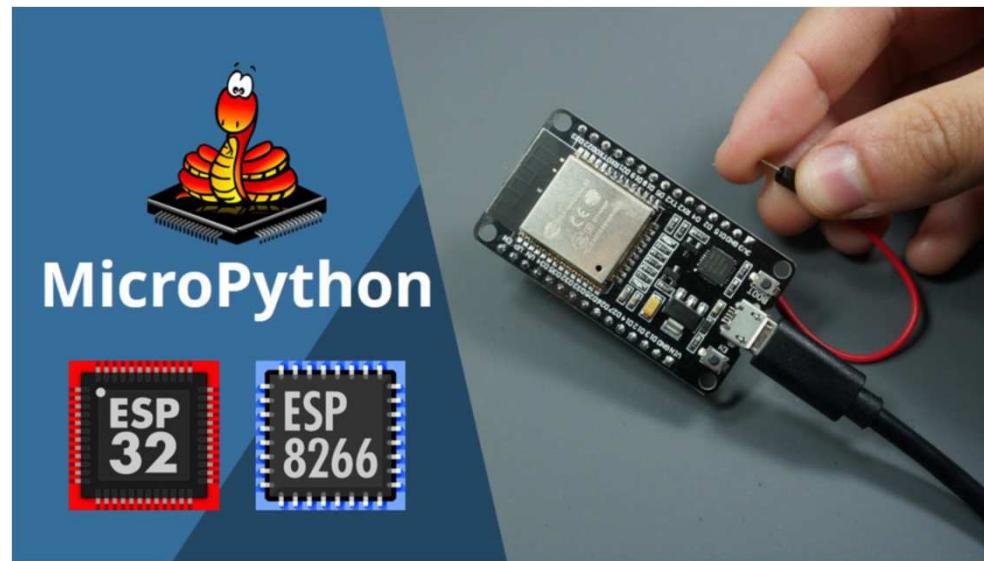
oled.text('Hello, World 1!', 0, 0)
oled.text('Hello, World 2!', 0, 10)
oled.text('Hello, World 3!', 0, 20)

while True:
    #led.value(not led.value())
    pot_value = pot.read()
    print(pot_value)

    #Efface l'écran
    oled.fill(0)

    sleep(0.1)
    #Passe la valeur du potard à l'écran
    #oled.text('Hello, World 4!', 0, 30)
    oled.text(str(pot_value), 50, 30)
    oled.show()
```

## TP8 - 4 – CAPACITIVE TOUCH

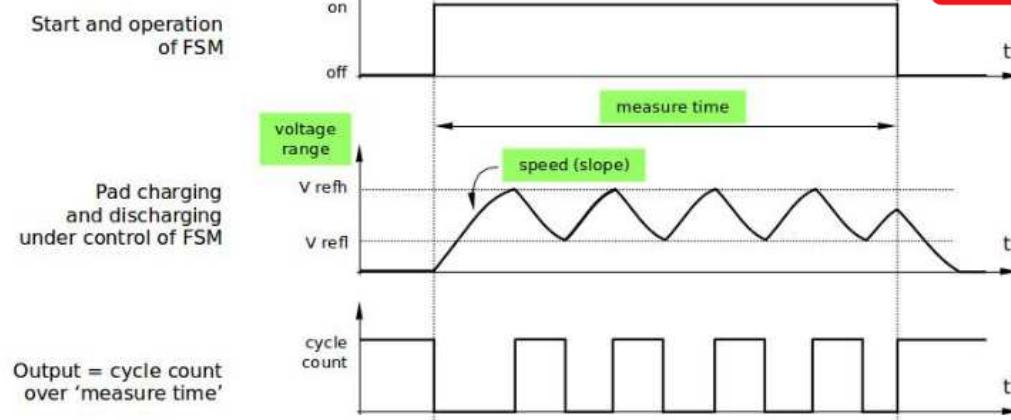


# Introducing the ESP32 Touch Sensor

The **ESP32 has 10 capacitive sensing GPIOs.**

These GPIOs can sense variations in anything that holds an electrical charge, like the human skin. So, they can detect variations induced when touching the GPIOs with a finger. Here are the capacitive touch GPIOs:

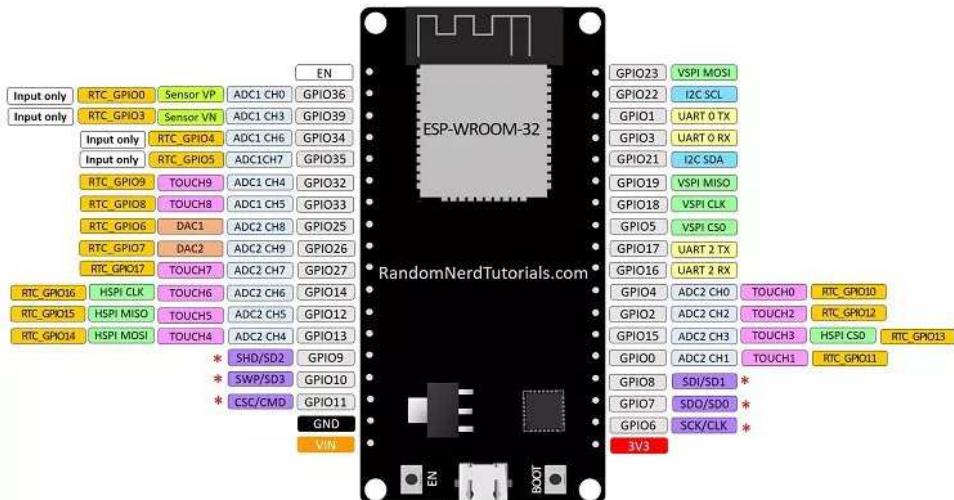
- **T0 (GPIO 4)**
- **T1 (GPIO 0)**
- **T2 (GPIO 2)**
- **T3 (GPIO 15)**
- **T4 (GPIO 13)**
- **T5 (GPIO 12)**
- **T6 (GPIO 14)**
- **T7 (GPIO 27)**
- **T8 (GPIO 33)**
- **T9 (GPIO 32)**



[https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/touch\\_pad.html](https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/peripherals/touch_pad.html)

## ESP32 DEVKIT V1 - DOIT

version with 36 GPIOs



\* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

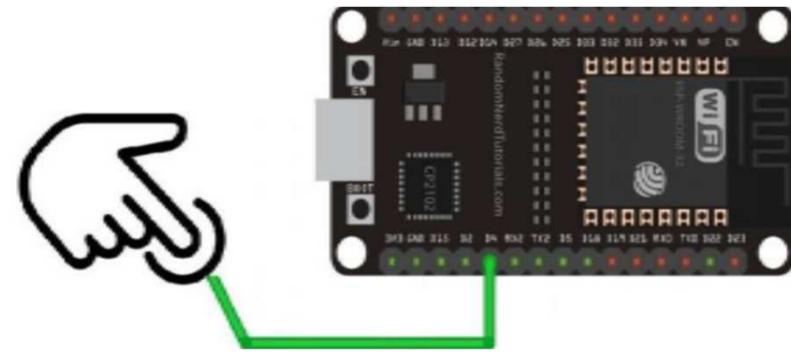


**Pink PIN are touch sensitive**

Note: Touch sensor 1 is GPIO 0. However, it's not available as a pin in the ESP32 model with 30 GPIOs. GPIO 0 is available on the board with 36 pins.

## MICRO-PYTHON SCRIPT

**Pin 0 that corresponds to GPIO 4.  
Wire a cable to GPIO 4, in a way  
that you can touch the metal part:**



To read from the ESP32 touch pins, you need:

- 1 - to import the TouchPad class from the machine module.
- 2 - the Pin class to setup the GPIOs, and
- 3 - the sleep() method to add delays.

Then, create a TouchPad object called touch\_pin on GPIO 4:

Finally, to read the value from the touch pin, use the read() method on the TouchPad object as follows:

Then, the touch pin reading is printed on the Shell:

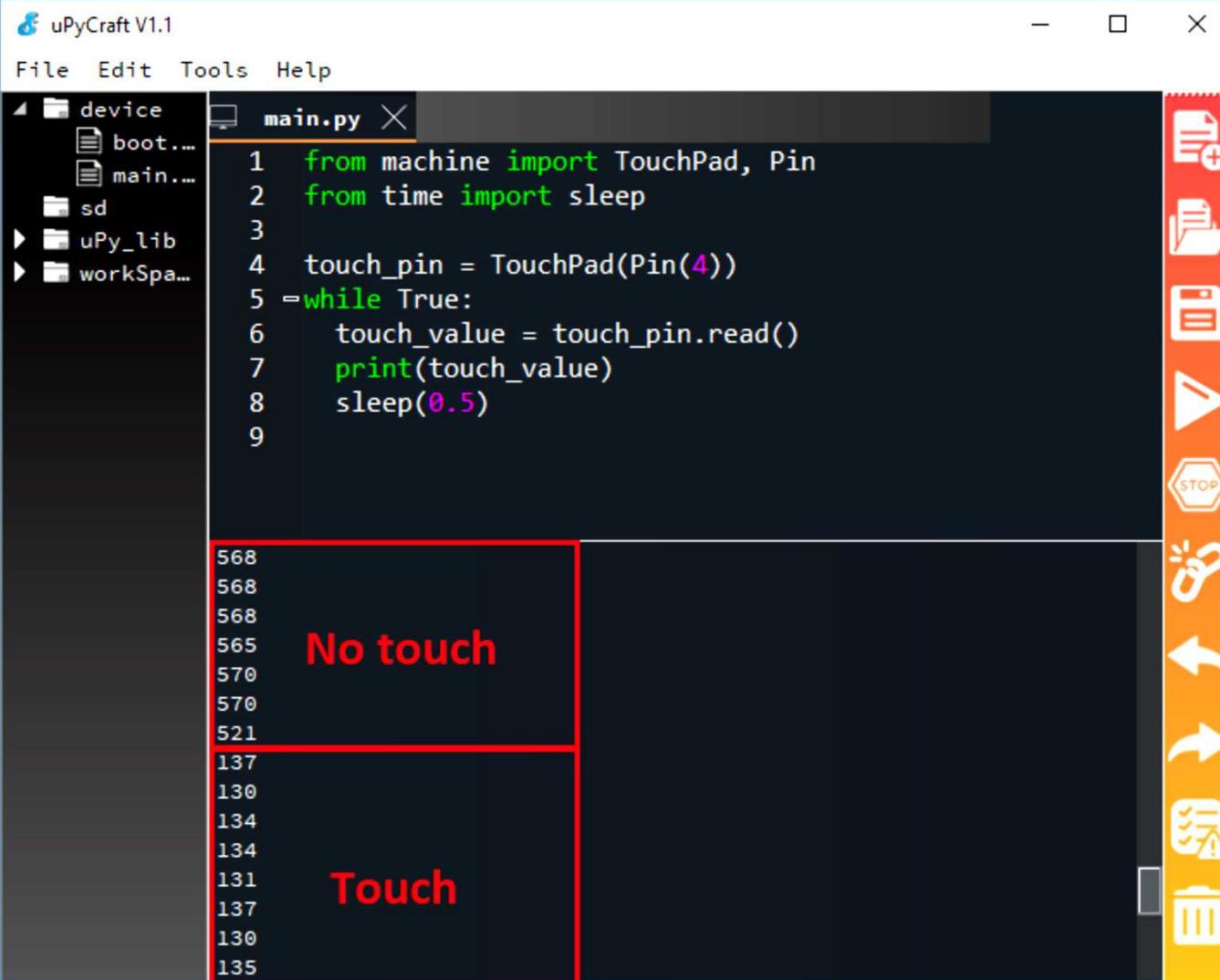
```
from machine import TouchPad, Pin
from time import sleep

touch_pin = TouchPad(Pin(4))

while True:

    touch_value = touch_pin.read()
    print(touch_value)
    sleep(0.5)
```

# DEMONSTRATION



The screenshot shows the uPyCraft V1.1 IDE interface. The left sidebar displays a file tree with 'device', 'boot...', 'main...', 'sd', 'uPy\_lib', and 'workSpa...'. The main window shows a code editor for 'main.py' with the following content:

```
from machine import TouchPad, Pin
from time import sleep

touch_pin = TouchPad(Pin(4))
while True:
    touch_value = touch_pin.read()
    print(touch_value)
    sleep(0.5)
```

The output window below the code editor shows the results of the script's execution. The output is split into two sections by a red box:

- No touch**:  
568  
568  
568  
565  
570  
570  
521
- Touch**:  
137  
130  
134  
134  
131  
137  
130  
135

The right side of the interface features a vertical toolbar with various icons for file operations, project management, and debugging.

**Note:** the values you get depend on the board, on your surroundings, on the wires you're using and other external factors.

## OTHER CAPACITIVE TOUCH EXAMPLE

Use the TouchPad class in the machine module:

```
from machine import TouchPad, Pin  
t = TouchPad(Pin(14))  
t.read() # Returns a smaller number when touched
```

TouchPad.read returns a value relative to the capacitive variation. Small numbers (typically in the tens) are common when a pin is touched, larger numbers (above one thousand) when no touch is present. However the values are *relative* and can vary depending on the board and surrounding composition so some calibration may be required.

There are ten capacitive touch-enabled pins that can be used on the ESP32: 0, 2, 4, 12, 13 14, 15, 27, 32, 33. Trying to assign to any other pins will result in a ValueError.

Note that TouchPads can be used to wake an ESP32 from sleep:

```
import machine  
from machine import TouchPad, Pin  
import esp32  
  
t = TouchPad(Pin(14))  
t.config(500) # configure the threshold at which the pin is considered touched  
esp32.wake_on_touch(True)  
machine.lightsleep() # put the MCU to sleep until a touchpad is touched
```

For more details on touchpads refer to [Espressif Touch Sensor](#)

## touchRead()

Reading the touch sensor is straightforward. In the Arduino IDE, you use the **touchRead()** function, that accepts as argument, the GPIO you want to read.

This example reads the touch pin 0 and displays the results in the serial monitor.

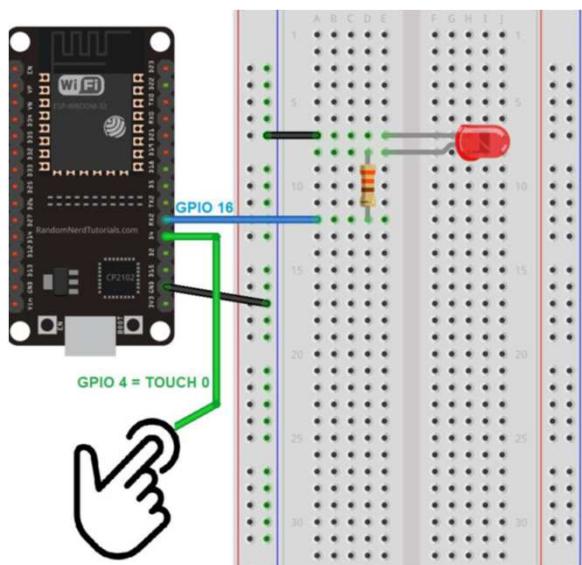
The T0 pin (touch pin 0), corresponds to GPIO 4, as we've seen previously in the pinout.

```
// ESP32 Touch Test
// Just test touch pin - Touch0 is T0
which is on GPIO 4.

void setup()
{
    Serial.begin(115200);
    delay(1000); // give me time to bring up
    serial monitor
    Serial.println("ESP32 Touch Test");
}

void loop()
{
    Serial.println(touchRead(4)); // get
    value of Touch 0 pin = GPIO 4
    delay(1000);
}
```

# TOUCH SENSITIVE LED



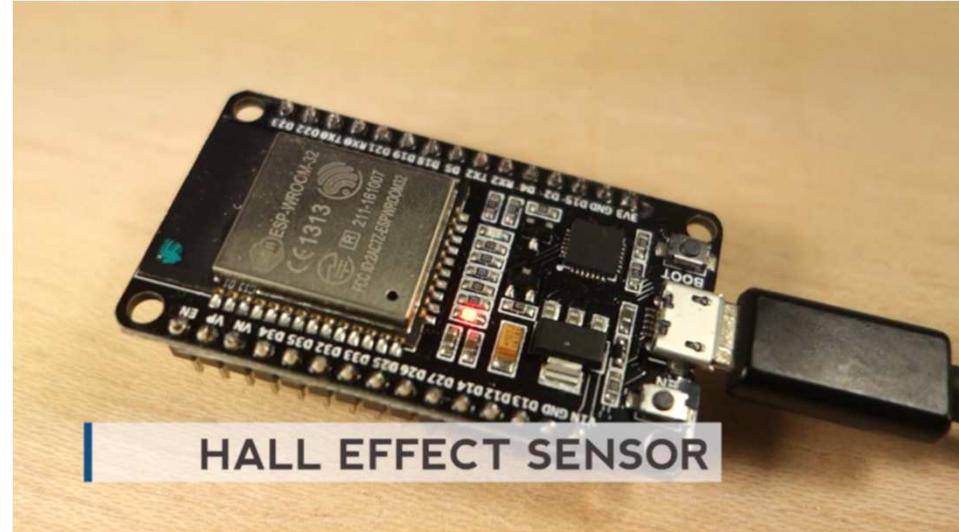
In our case, when we aren't touching the pin, the normal value is above **70** and when we touch the aluminum foil it drops to some value below **10**.

So, we can set a threshold value, and when the reading goes below that value, an LED lights up.

**A good threshold value in this case is 20, for example.**

```
// set pin numbers
const int touchPin = 4;
const int ledPin = 16;
// change with your threshold value
const int threshold = 20;
// variable for storing the touch pin value
int touchValue;
void setup(){
Serial.begin(115200);
delay(1000);
pinMode (ledPin, OUTPUT);
}
void loop(){
// read the state of the pushbutton value:
touchValue = touchRead(touchPin);
Serial.print(touchValue);
// check if the touchValue is below the
threshold
// if it is, set ledPin to HIGH
if(touchValue < threshold){
// turn LED on
digitalWrite(ledPin, HIGH);
Serial.println(" - LED on");
}
else{
// turn LED off
digitalWrite(ledPin, LOW);
Serial.println(" - LED off");
}
delay(500);
}
```

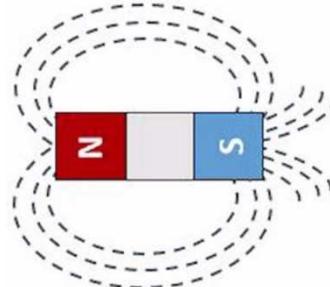
## TP8 - 5 – HALL EFFECT SENSOR



# READ HALL EFFECT SENSOR

The ESP32 features a built-in hall effect sensor which is located as shown in the figure below (behind that metal cap):

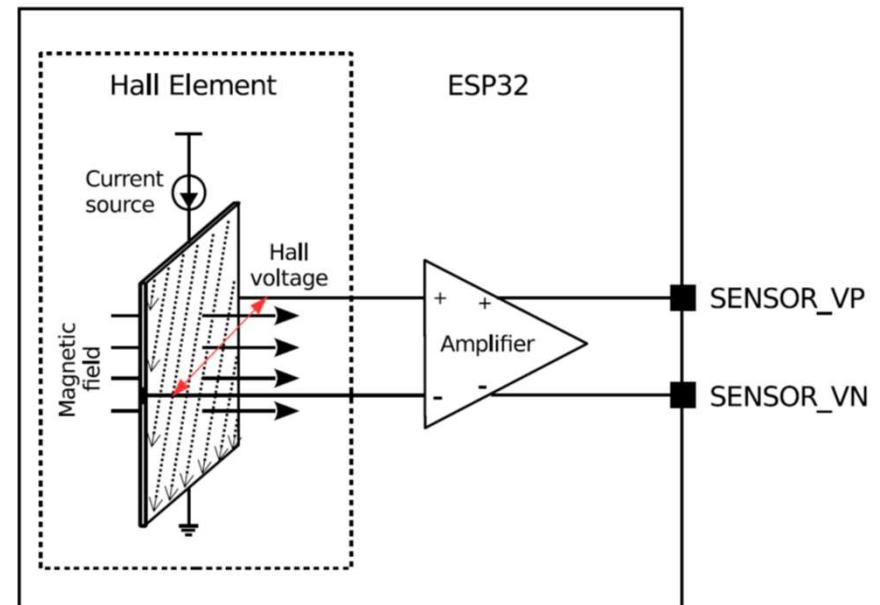
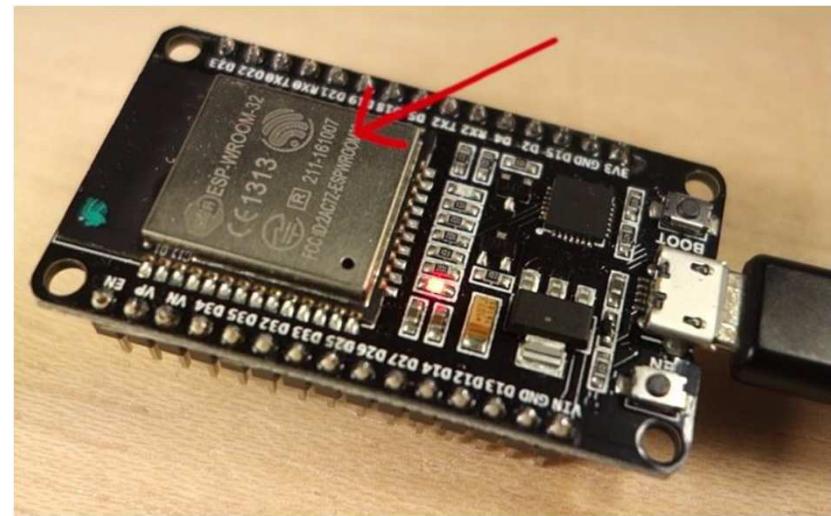
**A hall effect sensor can detect variations in the magnetic field in its surroundings.  
The greater the magnetic field, the greater the output voltage.**



Hall effect sensor

**A hall effect sensor can be combined with a threshold detection to act as a switch. Hall effect sensors are mainly used to:**

- Detect proximity;
- Calculate positioning;
- Count the number of revolutions of a wheel;
- Detect a door closing;
- And much more.



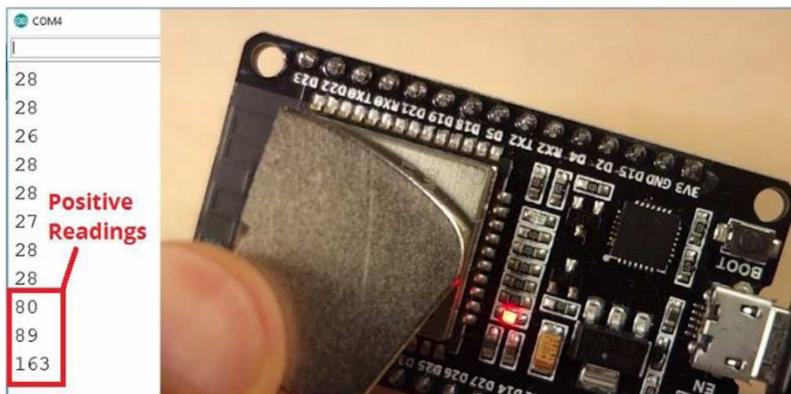
# READ HALL EFFECT SENSOR

## Reading the hall effect sensor

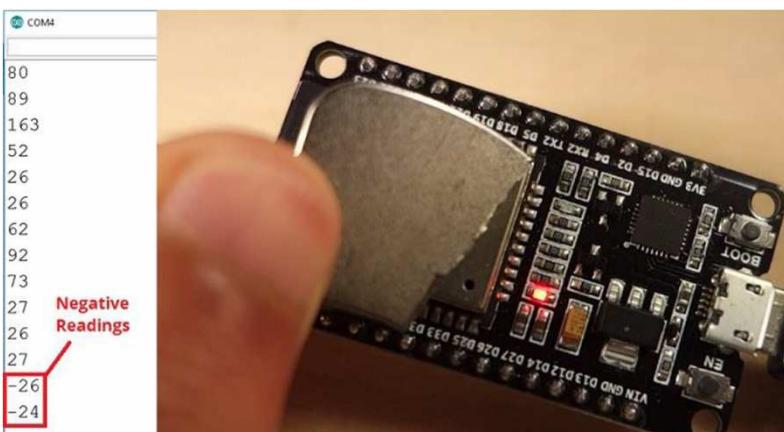
measurements with the ESP32 using the Arduino IDE is as simple as using the **hallRead()** function.

```
// Access the internal hall effect detector  
on the esp32 values can be quite low.  
int val = 0;  
void setup() {  
    Serial.begin(9600);  
}  
void loop() {  
    // read hall effect sensor value  
    val = hallRead();  
    Serial.println(val);  
    delay(1000);  
}
```

Approaching a magnet to the ESP32 hall sensor and see the values increasing...



Or decreasing depending on the magnet pole that is facing the sensor:



**NOTE: The closer the magnet is to the sensor, the greater the absolute values are.**

# MICROPYTHON SCRIPT: main.py

```
from machine import TouchPad, Pin
import esp32
from time import sleep

touch_pin = TouchPad(Pin(4))

while True:
    touch_value = touch_pin.read()
    #print(touch_value)
    hall_value = esp32.hall_sensor()
    print("C=" + str(touch_value) + "|"
          "B=" + str(hall_value))
    sleep(0.5)
```

## esp32 – functionality specific to the ESP32

The `esp32` module contains functions and classes specifically aimed at controlling ESP32 modules.

### Functions

#### `esp32.wake_on_touch(wake)`

Configure whether or not a touch will wake the device from sleep. `wake` should be a boolean value.

#### `esp32.wake_on_ext0(pin, level)`

Configure how EXT0 wakes the device from sleep. `pin` can be `None` or a valid Pin object. `level` should be `esp32.WAKEUP_ALL_LOW` or `esp32.WAKEUP_ANY_HIGH`.

#### `esp32.wake_on_ext1(pins, level)`

Configure how EXT1 wakes the device from sleep. `pins` can be `None` or a tuple/list of valid Pin objects. `level` should be `esp32.WAKEUP_ALL_LOW` or `esp32.WAKEUP_ANY_HIGH`.

#### `esp32.raw_temperature()`

Read the raw value of the internal temperature `sensor`, returning an integer.

#### `esp32.hall_sensor()`

Read the raw value of the internal `Hall sensor`, returning an integer.

[https://docs.micropython.org/en/latest/library/esp32.html?highlight=hall%20sensor#esp32.hall\\_sensor](https://docs.micropython.org/en/latest/library/esp32.html?highlight=hall%20sensor#esp32.hall_sensor)

# HOW MQTT WORKS

# What is MQTT and How It Works

MQTT stands for Message Queuing Telemetry Transport.

It is a lightweight publish and subscribe system where you can publish and receive messages as a client.



## MQTT – How It Works

Send a command to **control an output**



**Read and publish data**



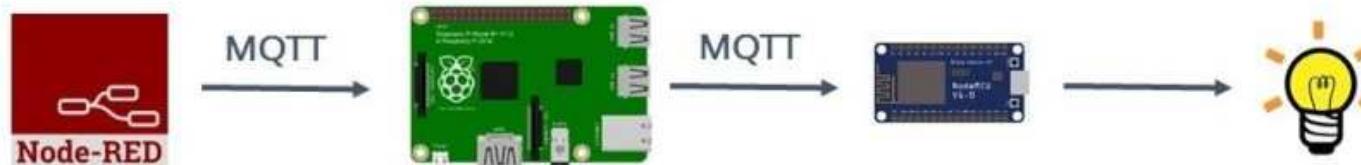
MQTT is a simple messaging protocol, designed for constrained devices with low-bandwidth. So, it's the perfect solution for Internet of Things applications. MQTT allows you to send commands to control outputs, read and publish data from sensor nodes and much more.

Therefore, it makes it really easy to establish a communication between multiple devices.

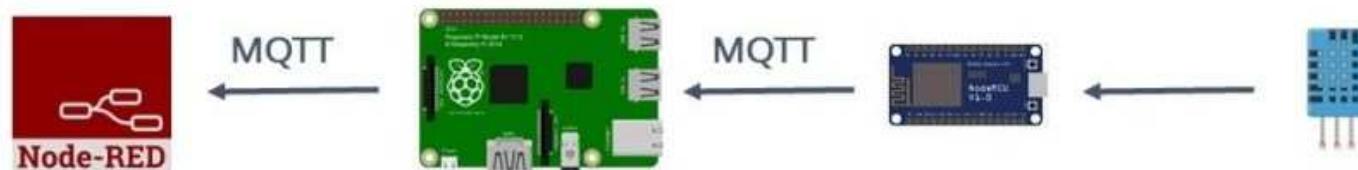
If you like home automation and you want to build a complete home automation system, I recommend downloading my [home automation course](#).

# HIGH LEVEL OVERVIEW

You can **send a command** with a client ([like Node-RED](#)) to control an **output**:



Or you can **read data** from a **sensor** and **publish** it to a client ([like Node-RED](#)):



## MQTT Basic Concepts

In MQTT there are a few basic concepts that you need to understand:

- **Publish/Subscribe**
- **Messages**
- **Topics**
- **Broker**

# HIGH LEVEL OVERVIEW

## MQTT - Publish/Subscribe:

The first concept is the *publish and subscribe* system. In a publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a particular topic to receive messages



- For example **Device 1** publishes on a topic.
- **Device 2** is subscribed to the same topic as **device 1** is publishing in.
- So, **device 2** receives the message.

## MQTT - Messages:

Messages are the information that you want to exchange between your devices. **Whether it's a command or data.**

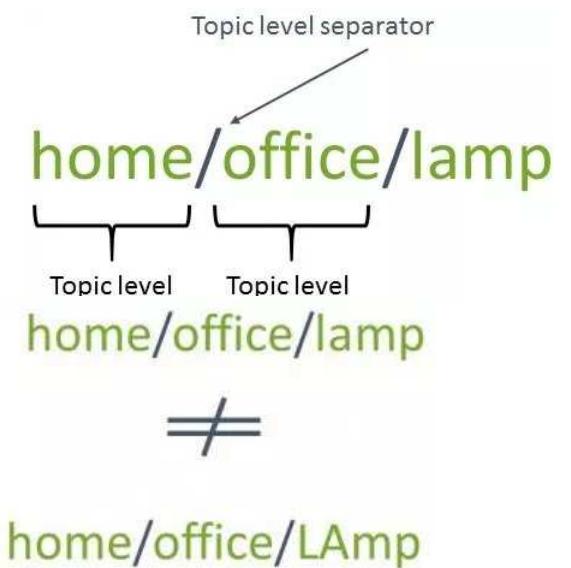
## MQTT - Topics:

Another important concept are the *topics*.

Topics are the way you register interest for incoming messages or how you specify where you want to publish the message.

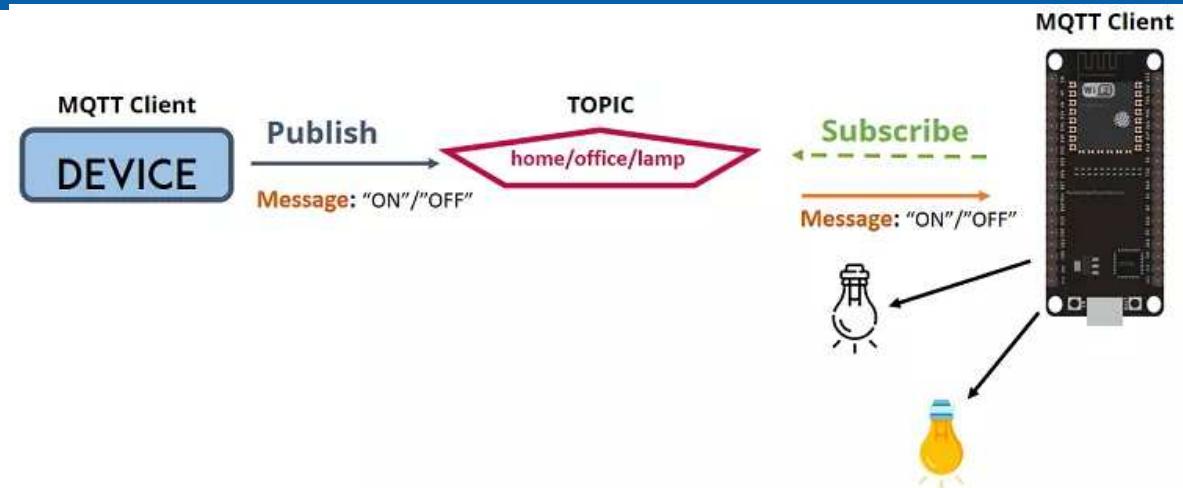
Topics are **represented with strings separated by a forward slash**. Each forward slash indicates a topic level. Here's an example on how you would **create a topic for a lamp in your home office**:

**Note:** topics are case-sensitive, **which makes these two topics different:**



# HIGH LEVEL OVERVIEW

If you would like to turn on a lamp in your home office using MQTT you can imagine the following scenario:



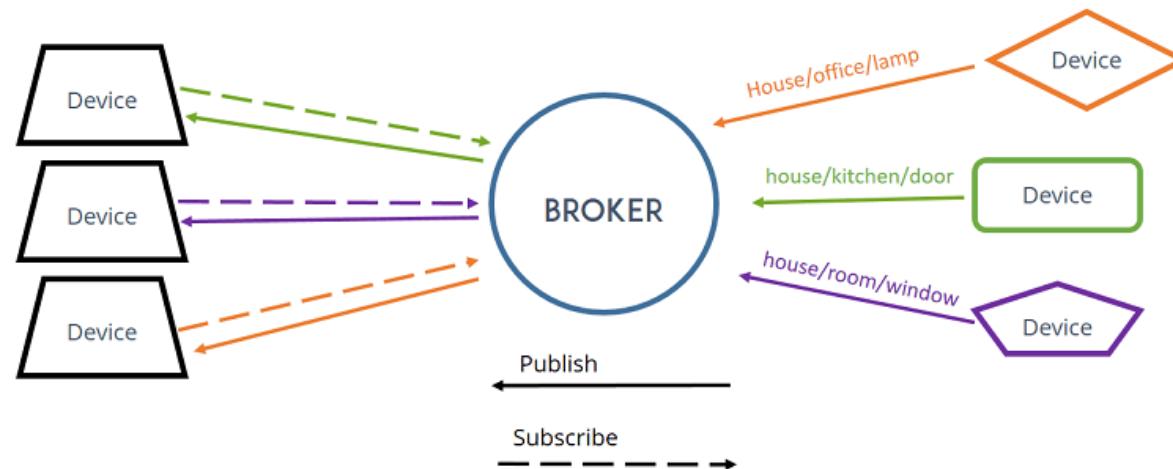
1. You have a device that publishes “on” and “off” messages on the **home/office/lamp** topic.
2. You have a device that controls a lamp (it can be an ESP32, ESP8266, or any other board). The ESP32 that controls your lamp, is subscribed to that topic: **home/office/lamp**.
- 3. So, when a new message is published on that topic,** the ESP32 receives the “on” or “off” message and turns the lamp on or off.

This first device, can be an ESP32, an ESP8266, or an Home Automation controller platform like Node-RED, Home Assistant, Domoticz, or OpenHAB, for example.



## MQTT – Broker

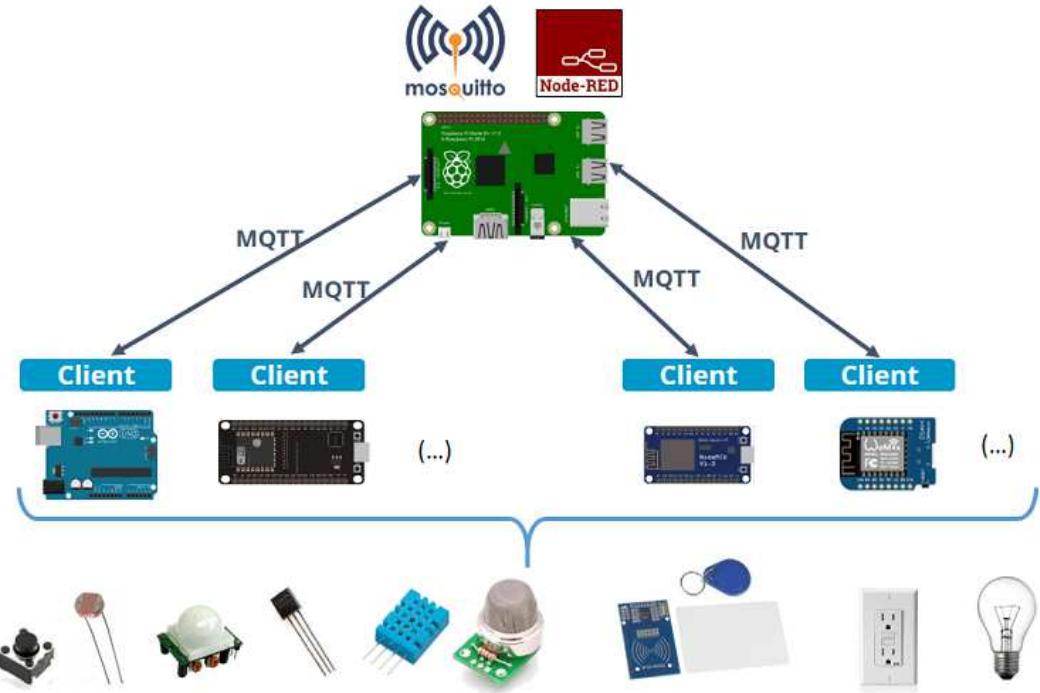
At last, you also need to be aware of the term *broker*.  
The broker is primarily responsible for **receiving** all messages, **filtering** the messages, **decide** who is interested in them and then **publishing** the message to all subscribed clients.



There are several brokers you can use. In our home automation projects we use the [Mosquitto broker](#) which can be installed in the Raspberry Pi.  
**Alternatively, you can use a cloud MQTT broker.**

# How to Use MQTT in Home Automation and IoT Projects

As we've seen previously, MQTT is great for home automation and internet of things projects. If you want to start making your own projects using MQTT here's an example of what you can do.



Here's the steps you should follow:

- 1) Set up your Raspberry Pi. Follow our [Getting Started Guide with Raspberry Pi](#).
- 2) [Enable and Connect your Raspberry Pi with SSH](#).
- 3) You need [Node-RED installed on your Pi](#) and [Node-RED Dashboard](#).
- 4) Install the [Mosquitto broker on the Raspberry Pi](#).
- 5) Add the ESP8266 or the ESP32 to this system. You can follow the next MQTT tutorials:
  - [ESP32 and Node-RED with MQTT – Publish and Subscribe](#)
  - [ESP8266 and Node-RED with MQTT – Publish and Subscribe](#)

# Installing Mosquitto Broker on Raspbian OS

## Installing Mosquitto Broker on Raspbian OS

Open a new Raspberry Pi terminal window:



To install the Mosquitto Broker enter these next commands:

```
pi@raspberry:~ $ sudo apt update
```

```
pi@raspberry:~ $ sudo apt install -y mosquitto mosquitto-clients
```

You'll have to type **Y** and press **Enter** to confirm the installation.

To make Mosquitto auto start on boot up enter:

```
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```

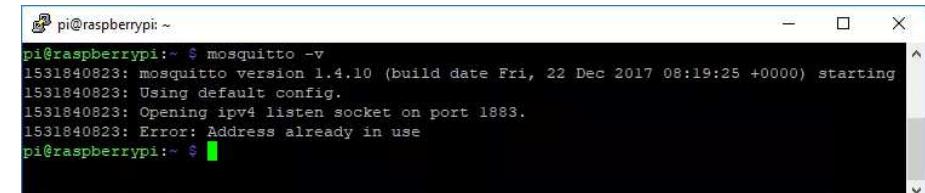
## Testing Installation

Send the command:

```
pi@raspberry:~ $ mosquitto -v
```

This returns the Mosquitto version that is currently running in your Raspberry Pi.

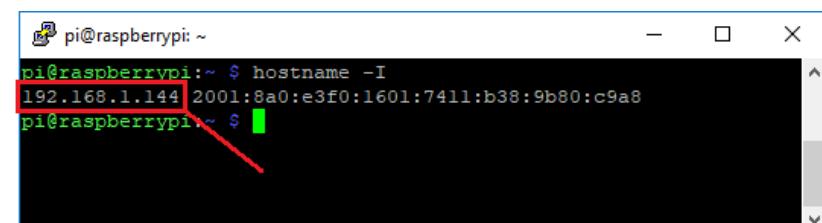
It should be 1.4.X or above.



## Raspberry Pi IP Address

To use Mosquitto broker later on your projects, you'll need your Raspberry Pi IP address. To retrieve your Raspberry Pi IP address, type the next command in your Terminal window:

```
pi@raspberry:~ $ hostname -I
```



# Testing Mosquitto Broker on Raspbian OS

After [installing MQTT Broker](#), I recommend installing an MQTT Client to test the Broker installation and publish sample messages.

The next command shows how to install MQTT Mosquitto Client:

```
pi@raspberry:~ $ sudo apt-get install mosquitto-clients
```

You'll have to type **Y** and press **Enter** to confirm the installation.

Run Mosquitto on background as a daemon: `pi@raspberry:~ $ mosquitto -d`

## Subscribing to `testTopic` Topic

To subscribe to an MQTT topic with Mosquitto Client open a terminal Window #1 and enter the command:

```
pi@raspberry:~ $ mosquitto_sub -d -t testTopic
```

You're now subscribed to a topic called **testTopic**.

## Publishing “Hello World!” Message to `testTopic` Topic

To publish a sample message to `testTopic`, open a terminal Window #2 and run this command:

```
pi@raspberry:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
```

The message “Hello World!” is received in Window #1 as illustrated in the figure above.

```
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
Client mosqsub/865-raspberrypi sending CONNECT
Client mosqsub/865-raspberrypi received CONNACK
Client mosqsub/865-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)
Client mosqsub/865-raspberrypi received SUBACK
Subscribed (mid: 1): 0
```

```
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
Client mosqsub/867-raspberrypi sending CONNECT
Client mosqsub/867-raspberrypi received CONNACK
Client mosqsub/867-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)
Client mosqsub/867-raspberrypi received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/867-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))
Hello world!
```

```
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
Client mosqpub/868-raspberrypi sending CONNECT
Client mosqpub/868-raspberrypi received CONNACK
Client mosqpub/868-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))
Client mosqpub/868-raspberrypi sending DISCONNECT
pi@raspberrypi:~ $
```

# PUBLISHING A MESSAGE TO MULTIPLE CLIENTS

Having Window #1 still subscribed to topic **testTopic**, open a new terminal Window #3 and run this command to subscribe to **testTopic** topic:

```
pi@raspberry:~ $ mosquitto_sub -d -t testTopic
```

On Window #2 publish the “**Hello World!**” message:

```
pi@raspberry:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
```

Since two clients are subscribed to **testTopic** topic, they will both receive “**Hello world!**” message.

The image shows three terminal windows (Window #1, Window #2, and Window #3) running on a Raspberry Pi. Window #1 (top) shows a subscriber client connected to the broker, publishing the message "Hello world!" to the "testTopic". Window #2 (middle) shows a publisher client publishing the same message. Window #3 (bottom) shows another subscriber client receiving the published message. The terminal output includes MQTT protocol messages like CONNECT, CONNACK, SUBSCRIBE, SUBACK, PUBLISH, and DISCONNECT.

```
Window #1
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
Client mosqsub/919-raspberrypi sending CONNECT
Client mosqsub/919-raspberrypi received CONNACK
Client mosqsub/919-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)
Client mosqsub/919-raspberrypi received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/919-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))
Hello world!
Client mosqsub/919-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))
Hello world!
pi@raspberrypi: ~

Window #2
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
Client mosqpub/920-raspberrypi sending CONNECT
Client mosqpub/920-raspberrypi received CONNACK
Client mosqpub/920-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))
Client mosqpub/920-raspberrypi sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
Client mosqpub/922-raspberrypi sending CONNECT
Client mosqpub/922-raspberrypi received CONNACK
Client mosqpub/922-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))
Client mosqpub/922-raspberrypi sending DISCONNECT
pi@raspberrypi: ~

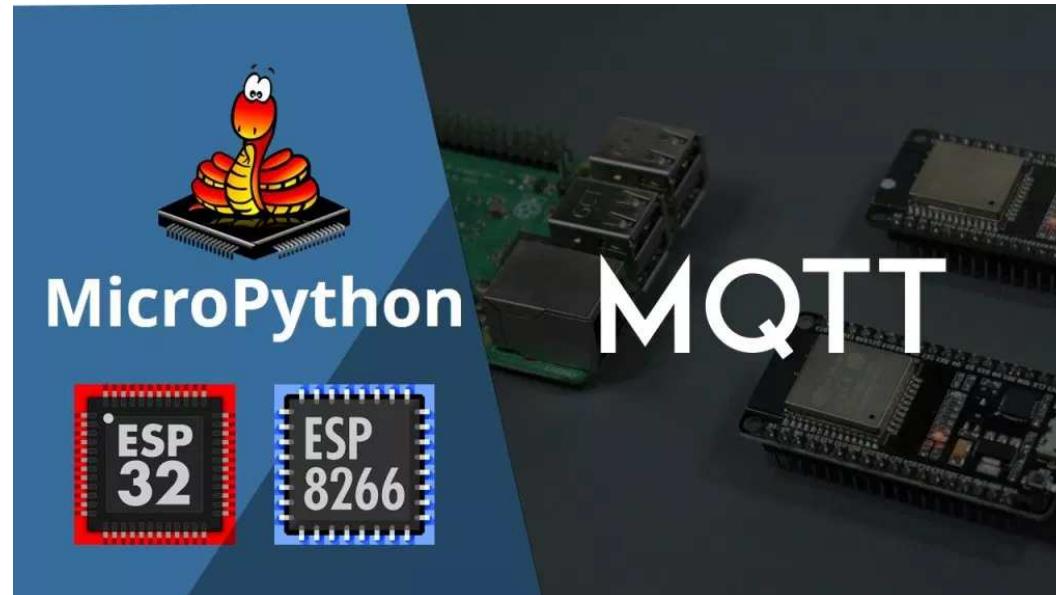
Window #3
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
Client mosqsub/921-raspberrypi sending CONNECT
Client mosqsub/921-raspberrypi received CONNACK
Client mosqsub/921-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)
Client mosqsub/921-raspberrypi received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/921-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))
Hello world!
pi@raspberrypi: ~
```

This simple example shows how MQTT works and how your devices (for example: ESP8266) could be subscribed to the same topic to receive messages or a device could publish messages to multiple devices.

<https://randomnerdtutorials.com/testing-mosquitto-broker-and-client-on-raspberry-pi/>

# TP9 – 1

## MQTT ON ESP32/ESP8266



# MicroPython – Getting Started with MQTT on ESP32/ESP8266

As an example, we'll exchange simple text messages between two ESP boards. The idea is to use the concepts learned here to exchange sensor readings, or commands.

## MQTT Broker

To use MQTT, you need a broker. We'll be using [Mosquitto broker](#) installed on a Raspberry Pi.

Read [How to Install Mosquitto Broker on Raspberry Pi](#).

If you're not familiar with MQTT make sure you read our introductory tutorial: [What is MQTT and How It Works](#)

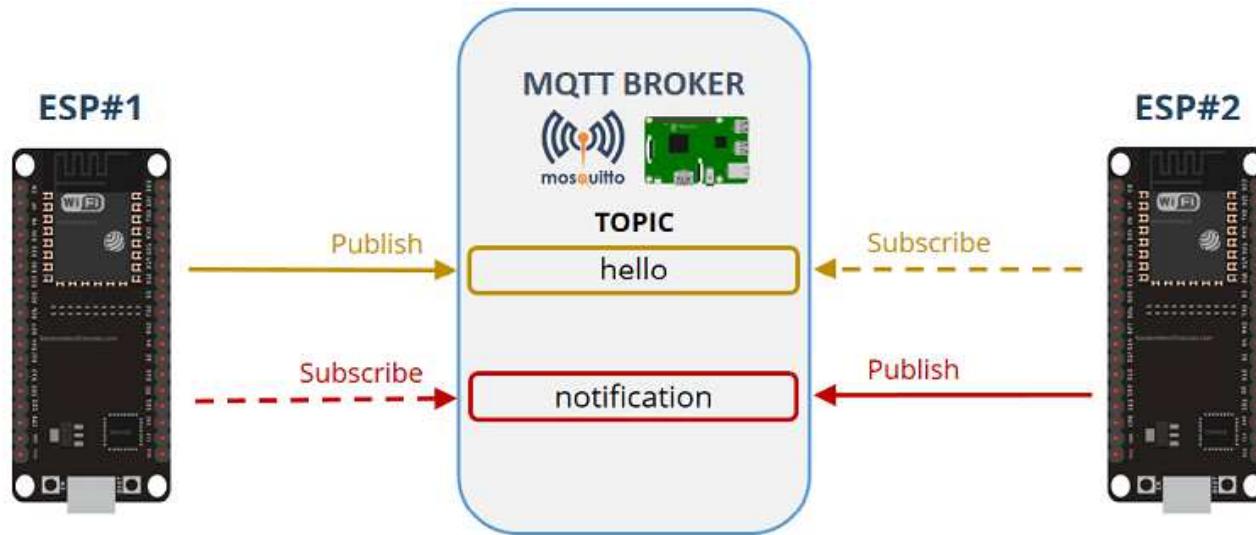


<https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/>

<https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>

# PROJECT OVERVIEW

Here's a high-level overview of the project we'll build:



- ESP#1 publishes messages on the ***hello*** topic. It publishes a “Hello” message followed by a counter (Hello 1, Hello 2, Hello 3, ...). It publishes a new message every 5 seconds.
- ESP#1 is subscribed to the ***notification*** topic to receive notifications from the ESP#2 board.
- ESP#2 is subscribed to the ***hello*** topic. ESP #1 is publishing in this topic. Therefore, ESP#2 receives ESP#1 messages.
- When ESP#2 receives the messages, it sends a message saying ‘received’. This message is published on the ***notification*** topic. ESP#1 is subscribed to that topic, so it receives the message.

# PREPARING ESP#1

Let's start by preparing ESP#1:

- It is subscribed to the **notification** topic
  - It publishes on the **hello** topic

## Importing umqttsimple library:

To use MQTT with the ESP32/ESP8266 and MicroPython, you need to install the `umqtt.simple` library.

- 1.** Create a new file by pressing the **New File** button.



## New file

- 2.** Copy the *umqttsimple* library code into it. You can access the *umqttsimple* library code in the following link:

- <https://raw.githubusercontent.com/RuiSantosdotme/ESP-MicroPython/master/code/MQTT/umqttsimple.py>



Save file

3. Save the file by pressing the **Save** button.

4. Call this new file “**umqttsimple.py**” and press **ok**.

- 5. Click the Download and Run button.**



Download and run

6. The file should be saved on the device folder with the name "**umqttsimple.py**" as highlighted in the figure below.

Now, you can use the library functionalities in your code by importing the library.

# PREPARING ESP#1: How the Code Works - *boot.py*

## How the Code Works

You need to import all the following libraries:

Set the debug to None and activate the garbage collector.

In the following variables, you need to enter your network credentials and your broker IP address.

To create an MQTT client, we need to get the ESP unique ID. That's what we do in the following line (it is saved on the `client_id` variable).

Next, write the topic the ESP#1 is subscribed to, and the topic it will be publishing messages:

Then, create the following variables:

The `last_message` variable will hold the last time a message was sent.

The `message_interval` is the time between each message sent. Here, we're setting it to 5 seconds (this means a new message will be sent every 5 seconds).

The `counter` variable is simply a counter to be added to the message.

After that, we make the procedures to connect to the network.

```
import time
from umqttsimple import MQTTClient
import ubinascii
import machine
import micropython
import network
import esp
esp.osdebug(None)
import gc
gc.collect()

ssid = 'REPLACE_WITH_YOUR_SSID'
password = 'REPLACE_WITH_YOUR_PASSWORD'
mqtt_server = 'REPLACE_WITH_YOUR_MQTT_BROKER_IP'
#EXAMPLE IP ADDRESS
#mqtt_server = '192.168.1.144'
client_id = ubinascii.hexlify(machine.unique_id())
topic_sub = b'notification'
topic_pub = b'hello'

last_message = 0
message_interval = 5
counter = 0

station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)

while station.isconnected() == False:
    pass

print('Connection successful')
print(station.ifconfig())
```

# PREPARING ESP#1: How the Code Works - *main.py*

In the *main.py* file is where we'll write the code to publish and receive the messages:

1 The first thing you should do is creating a **callback function (cb)** that accept as parameters the topic and the message and WILL run whenever a message is published on a topic the ESP is subscribed to.

- Start by printing the topic and the message.
- Then, check if the message was published on the **notification** topic, and if the content of the message is 'received'.
- If this if statement is True, it means that ESP#2 received the 'hello' message sent by ESP#1. Basically, this callback function handles what happens when a certain message is received on a certain topic.

2 **Connect and subscribe : connect\_and\_subscribe()** function responsible for connecting to the broker as well as to subscribe to a topic.

Start by declaring the client\_id, mqtt\_server and topic\_sub variables as global variables.

**This way, we can access these variables throughout the code.**

Then, create a MQTTClient object called client and need to pass as parameters the client\_id, and the IP address of the MQTT broker (mqtt\_server).

**These variables were set on the boot.py file.**

After that, set the callback function to the client (sub\_cb).

Next, connect the client to the broker using the connect() method on the MQTTClient object.

After connecting, we subscribe to the topic\_sub topic. Set the topic\_sub on the boot.py file (notification).

**Finally, print a message and return the client:**

```
def sub_cb(topic, msg):
    print((topic, msg))
    if topic == b'notification' and msg == b'received':
        print('ESP received hello message')

def connect_and_subscribe():
    global client_id, mqtt_server, topic_sub
    client = MQTTClient(client_id, mqtt_server)
    client.set_callback(sub_cb)
    client.connect()
    client.subscribe(topic_sub)
    print('Connected to %s MQTT broker, subscribed to %s topic' % (mqtt_server, topic_sub))
    return client

def restart_and_reconnect():
    print('Failed to connect to MQTT broker. Reconnecting...')
    time.sleep(10)
    machine.reset()

try:
    client = connect_and_subscribe()
except OSError as e:
    restart_and_reconnect()

while True:
    try:
        client.check_msg()
        if (time.time() - last_message) > message_interval:
            msg = b'Hello #%d' % counter
            client.publish(topic_pub, msg)
            last_message = time.time()
            counter += 1
    except OSError as e:
        restart_and_reconnect()
```

# PREPARING ESP#1: How the Code Works - *main.py*

**③ Restart and reconnect:** a function called `restart_and_reconnect()` will be called in case the ESP32 or ESP8266 fails to connect to the broker. It prints a message to inform that the connection was not successful. We wait 10 seconds. Then, we reset the ESP using the `reset()` method.

## Receive and publish messages

Until now, we've created functions to handle tasks related with the MQTT communication. From now on, the code will call those functions to make things happen.

The first thing we need to do is to connect to the MQTT broker and subscribe to a topic. So, we **create a client by calling the `connect_and_subscribe()` function.**

**In case we're not able to connect to the MQTT broker, we'll restart the ESP by calling the `restart_and_reconnect()` function**

In the while loop is where we'll be receiving and publishing the messages. We use try and except statements to prevent the ESP from crashing in case something goes wrong.

Inside the try block, we start by applying the `check_msg()` method on the client.

The `check_msg()` method checks whether a pending message from the server is available. It waits for a single incoming MQTT message and process it. The subscribed messages are delivered to the callback function we've defined earlier (the `sub_cb()` function). If there isn't a pending message, it returns with None.

Then, we add an if statement to checker whether 5 seconds (`message_interval`) have passed since the last message was sent.

If it is time to send a new message, we create a `msg` variable with the "Hello" text followed by a counter.

To publish a message on a certain topic, you just need to apply the `publish()` method on the client and pass as arguments, the topic and the message. The `topic_pub` variable was set to `hello` in the `boot.py` file.

After sending the message, we update the last time a message was received by setting the `last_message` variable to the current time.

Finally, we increase the counter variable in every loop.

**If something unexpected happens, we call the `restart_and_reconnect()` function.**

```
def sub_cb(topic, msg):
    print((topic, msg))
    if topic == b'notification' and msg == b'received':
        print('ESP received hello message')

def connect_and_subscribe():
    global client_id, mqtt_server, topic_sub
    client = MQTTClient(client_id, mqtt_server)
    client.set_callback(sub_cb)
    client.connect()
    client.subscribe(topic_sub)
    print('Connected to %s MQTT broker, subscribed to %s topic' % (mqtt_server, topic_sub))
    return client

def restart_and_reconnect():
    print('Failed to connect to MQTT broker. Reconnecting...')
    time.sleep(10)
    machine.reset()

try:
    client = connect_and_subscribe()
except OSError as e:
    restart_and_reconnect()

while True:
    try:
        client.check_msg()
        if (time.time() - last_message) > message_interval:
            msg = b'Hello #%d' % counter
            client.publish(topic_pub, msg)
            last_message = time.time()
            counter += 1
    except OSError as e:
        restart_and_reconnect()
```

## UPLOADING SCRIPTS

That's it for ESP#1. Remember that you need to upload all the next files to make the project work (you should upload the files in order):



- 1.*umqttsimple.py*;
- 2.*boot.py*;
- 3.*main.py*.

After uploading all files, you should get success messages on: establishing a network connection; connecting to the broker; and subscribing to the topic.

AND see all the files under device folder

## PREPARING ESP#2: umqttsimple.py and boot.py

Let's now prepare ESP#2:

- It is subscribed to the **hello** topic
- It publishes on the **notification** topic

Like the ESP#1, you also need to upload the *umqttsimple.py*, *boot.py*, and *main.py* files.

### Importing umqttsimple

To use MQTT with the ESP32/ESP8266 and MicroPython, you need to install the *umqttsimple* library. Follow the steps described earlier to install the *umqttsimple* library in ESP#2.

You can access the *umqttsimple* library code in the following link:

<https://raw.githubusercontent.com/RuiSantosdotme/ESP-MicroPython/master/code/MQTT/umqttsimple.py>

This code is very similar with the previous **boot.py file**. You need to replace the following variables with your network credentials and the broker IP address.

```
ssid = 'REPLACE_WITH_YOUR_SSID'  
password = 'REPLACE_WITH_YOUR_PASSWORD'  
mqtt_server = 'REPLACE_WITH_YOUR_MQTT_BROKER_IP'
```

The only difference here is that we subscribe to the **hello** topic and publish on the **notification** topic.

```
topic_sub = b'hello'  
topic_pub = b'notification'
```



## PREPARING ESP#2: main.py

This code is very similar with the `main.py` from ESP#1.

We create the `sub_cb()`, the `connect_and_subscribe()` and the `restart_and_reconnect()` functions.

This time, the `sub_cb()` function just prints information about the topic and received message.

In the while loop, we check if we got a new message and save it in the `new_message` variable.

If we receive a new message, we publish a message saying 'received' on the `topic_sub` topic  
**(in this case we set it to notification in the `boot.py` file).**

```
def sub_cb(topic, msg):
    print((topic, msg))
```

```
new_message = client.check_msg()
```

```
if new_message != 'None':
    client.publish(topic_pub, b'received')
```

That's it for ESP#2.



Remember that you need to upload all the next files to make the project work (you should upload the files in order):

- 1.`umqttsimple.py`;
- 2.`boot.py`;
- 3.`main.py`.

The ESP32/ESP8266 should establish a network connection and connect to the broker successfully.

# DEMONSTRATION

After uploading all the necessary scripts to both ESP boards and having both boards and the Raspberry Pi with the Mosquitto broker running, you are ready to test the setup.

The ESP#2 should be receiving the “Hello” messages from ESP#1, as shown in the figure below

On the other side, ESP#1 board should receive the “received” message. The “received” message is published by ESP#2 on the **notification** topic. ESP#1 is subscribed to that topic, so it receives the message.

The image shows two instances of the uPyCraft V1.1 IDE. Both instances have three tabs open: boot.py, main.py, and umqttsimple.py. The main.py tab contains MQTT client code. The boot.py tab contains MQTT broker code. The right-hand window shows the serial output for the first ESP board (ESP#1). It starts with a connection message, followed by a series of "Hello" messages from the second board (ESP#2) on the 'notification' topic. The left-hand window shows the serial output for the second ESP board (ESP#2), which shows it publishing "Hello" messages to the 'notification' topic.

uPyCraft V1.1

File Edit Tools Help

device

boot.py main.py umqttsimple.py

```
4 import socket
5 import ustruct as struct
6 from ubinascii import hexlify
7
8 class MQTTException(Exception):
9     pass
10
11 class MQTTClient:
12
13     def __init__(self, client_id, server, port=0, user=None, passw
Connection successful
('192.168.1.147', 255, 255, 255, 0, '192.168.1.254', '192.168.1.254')
Connected to 192.168.1.144 MQTT broker, subscribed to b'hello' topic
(b'Hello', b'Hello #0')
(b'Hello', b'Hello #1')
(b'Hello', b'Hello #2')
(b'Hello', b'Hello #3')
(b'Hello', b'Hello #4')
(b'Hello', b'Hello #5')
(b'Hello', b'Hello #6')
(b'Hello', b'Hello #7')
(b'Hello', b'Hello #8')
(b'Hello', b'Hello #9')
(b'Hello', b'Hello #10')
(b'Hello', b'Hello #11')
(b'Hello', b'Hello #12')
(b'Hello', b'Hello #13')
(b'Hello', b'Hello #14')
(b'Hello', b'Hello #15')
```

uPyCraft V1.1

File Edit Tools Help

device

boot.py main.py umqttsimple.py

```
4 import socket
5 import ustruct as struct
6 from ubinascii import hexlify
7
8 class MQTTException(Exception):
9     pass
10
11 class MQTTClient:
12
13     def __init__(self, client_id, server, port=0, user=None, passw
Connection successful
('192.168.1.148', 255, 255, 255, 0, '192.168.1.254', '192.168.1.254')
Connected to 192.168.1.144 MQTT broker, subscribed to b'notification' topic
(b'notification', b'received')
ESP received hello message
(b'notification', b'received')
```

## TP9 – 2

# MQTT - Connect ESP32/ESP8266 to Node-RED

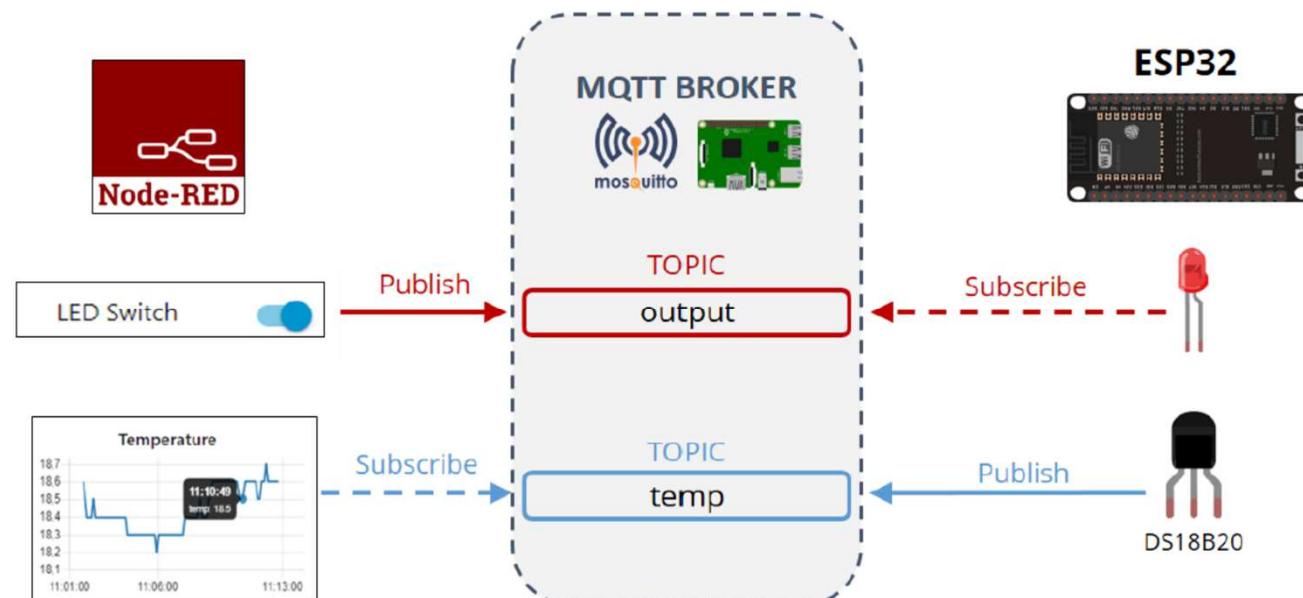
# PROJECT OVERVIEW

- In the Node-RED Dashboard, a slider switch allows you to control an output of the ESP32/ESP8266. To simplify the project, we'll control the on-board LED (GPIO 2).
- When you turn the slider switch on, Node-RED publishes a "on" message on the **output** topic. When the slider switch changes its state to off, it publishes a "off" message on the **output** topic.

The ESP32/ESP8266 is subscribed to the **output** topic.

When it receives a message, it turns the LED on or off accordingly - instead of an LED you can control any other output on a different GPIO;

- The ESP32 takes temperature readings with the DS18B20 temperature sensor.  
=>The readings are published on the **temp** topic every 5 seconds;
- Node-RED is subscribed to the **temp** topic.  
=> So, it receives the DS18B20 temperature readings and publishes the readings in a chart.

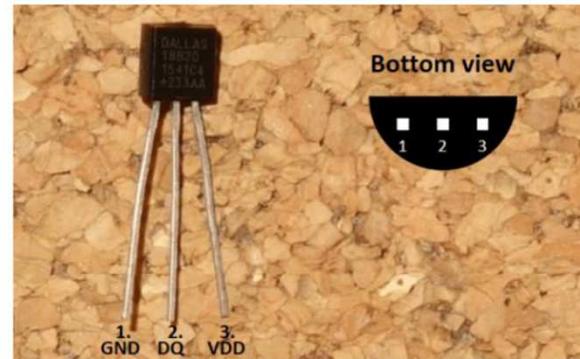


# Introducing the DS18B20 Temperature Sensor

The DS18B20 temperature sensor is a one-wire digital temperature sensor.

This means that you can read the temperature with a very simple circuit setup.

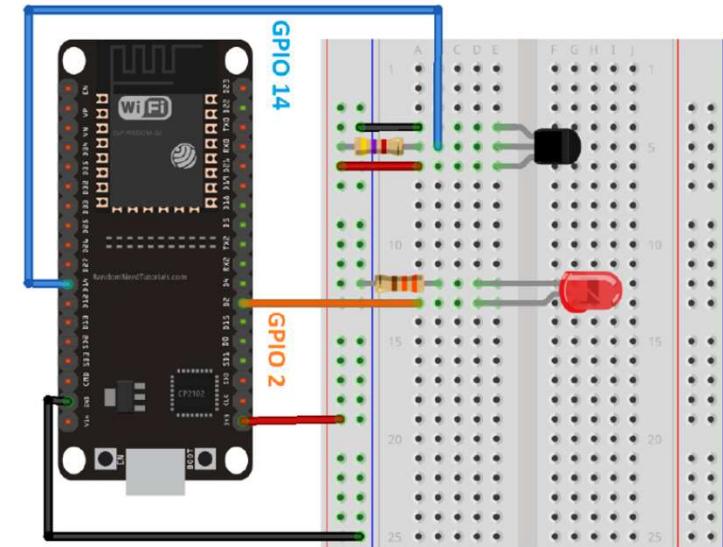
It communicates on common bus, which means that you can connect several devices and read their values using just one digital pin.



## Features

Here's some main features of the DS18B20 temperature sensor:

- Communicates over 1-Wire bus communication
- Operating range temperature: -55°C to 125°C
- Accuracy +/-0.5 °C (between the range -10°C to 85°C)



*This schematic uses the ESP32 DEVKIT DOIT board version with 36 GPIOs. Before assembling the circuit double-check the pinout for the board you're using.*

**Resistance to DS18B20 = 4,7kOhm**

Copy :

<https://github.com/RuiSantosdotme/ESPMicroPython/blob/master/code/MQTT/umqttsimple.py>

Save to device (download and run)

Copy :

[https://github.com/RuiSantosdotme/ESP-MicroPython/blob/master/code/MQTT/Node\\_RED\\_Client/boot.py](https://github.com/RuiSantosdotme/ESP-MicroPython/blob/master/code/MQTT/Node_RED_Client/boot.py)

Save to device (download and run)

# MICRO-PYTHON SCRIPT: BOOT.PY

```
import time
import onewire
import ds18x20
from umqttsimple import MQTTClient
import ubinascii
import machine
import micropython
import network
import esp
esp.osdebug(None)
import gc
gc.collect()
ssid = 'REPLACE_WITH_YOUR_SSID'
```

```
password = 'REPLACE_WITH_YOUR_PASSWORD'
mqtt_server =
'REPLACE_WITH_YOUR_MQTT_BROKER_IP'
#EXAMPLE IP ADDRESS
#mqtt_server = '192.168.1.144'
client_id =
ubinascii.hexlify(machine.unique_id())
topic_sub = b'output'
topic_pub = b'temp'
last_sensor_reading = 0
readings_interval = 5
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)
while station.isconnected() == False:
    pass
print('Connection successful')
print(station.ifconfig())
ds_pin = machine.Pin(14)
ds_sensor =
ds18x20.DS18X20(onewire.OneWire(ds_pin))
led = machine.Pin(2, machine.Pin.OUT,
value=0)
```

## MICRO-PYTHON SCRIPT: BOOT.PY

Among other libraries, you need to import the `onewire` and the `ds18x20` library to read from the DS18B20 temperature sensor.

```
import onewire  
import ds18x20
```

You also need to import the `MQTTClient` class from the `umqttsimple` library to set the ESP32/ESP8266 as an MQTT client.

```
from umqttsimple import MQTTClient
```

You need to include your network credentials as well as the broker IP address on the following variables:

```
ssid = 'REPLACE_WITH_YOUR_SSID'  
password = 'REPLACE_WITH_YOUR_PASSWORD'  
mqtt_server = 'REPLACE_WITH_YOUR_MQTT_BROKER_IP'  
ssid = 'Livebox-BBB6'  
password = 'xxxxxxxxxxxxxx'  
mqtt_server = '192.168.1.31'
```

The ESP32/ESP8266 is subscribed to the **output** topic and publishes on the **temp** topic.

```
topic_sub = b'output'  
topic_pub = b'temp'
```

Create auxiliary variables to save the last time a sensor reading was taken (`last_sensor_reading`) and to save the interval between readings (`readings_interval`), here, we're setting the interval to 5 seconds.

```
last_sensor_reading = 0  
readings_interval = 5
```

Create a variable called `ds_pin` that refers to GPIO 14, data wire pin of DS18B20 sensor is connected to.

```
ds_pin = machine.Pin(14)
```

Then, create a `ds18x20` object called `ds_sensor` on the `ds_pin` defined earlier.

If you want to read the sensor using a different pin, you need to modify the previous line.

```
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
```

Finally, create a `Pin` object on GPIO 2 called `led` that is off when the ESP32 first boots.

```
led = machine.Pin(2, machine.Pin.OUT, value=0)
```

That's it for the `boot.py` file. Proceed to the `main.py` file.

# MICRO-PYTHON SCRIPT: MAIN.PY

```
def read_ds_sensor():
    roms = ds_sensor.scan()
    print('Found DS devices: ', roms)
    print('Temperatures: ')
    ds_sensor.convert_temp()
    time.sleep(1)
    for rom in roms:
        temp = ds_sensor.read_temp(rom)
        if isinstance(temp, float):
            # uncomment for Fahrenheit
            #temp = temp * (9/5) + 32.0
            msg = (b'{0:3.1f}'.format(temp))
            print(temp, end=' ')
            print('Valid temperature')
        return msg
    return b'0.0'
def sub_cb(topic, msg):
    print((topic, msg))
    if msg == b'on':
        led.value(1)
    elif msg == b'off':
        led.value(0)
def connect_and_subscribe():
    global client_id, mqtt_server, topic_sub
    client = MQTTClient(client_id, mqtt_server)
    client.set_callback(sub_cb)
    client.connect()
    client.subscribe(topic_sub)
    print('Connected to %s MQTT broker, subscribed to %s topic'
    %
    (mqtt_server, topic_sub))
    return client
def restart_and_reconnect():
    print('Failed to connect to MQTT broker. Reconnecting...')
    time.sleep(10)
    machine.reset()
    try:
        client = connect_and_subscribe()
```

```
except OSError as e:
    restart_and_reconnect()
while True:
    try:
        client.check_msg()
        if (time.time() - last_sensor_reading) >
            readings_interval:
            msg = read_ds_sensor()
            client.publish(topic_pub, msg)
            last_sensor_reading = time.time()
    except onewire.OneWireError:
        print('Failed to read/publish sensor
            readings.')
        time.sleep(1)
    except OSError as e:
        restart_and_reconnect()
```

# MICRO-PYTHON SCRIPT: MAIN.PY

This script reads the temperature from the DS18B20 temperature sensor and publishes the sensor readings on the **temp** topic. It also subscribes the ESP32/ESP8266 to the **output** topic and turns the on-board LED on or off accordingly.

## DS18B20 temperature readings

We start by creating a function called `read_ds_sensor()`. You can use this function in other projects that use the DS18B20 sensor.

```
def read_ds_sensor():
```

The DS18B20 communicates via 1-Wire communication protocol.

This means you can read several temperature sensors wired on the same GPIO.

The function starts by scanning for DS18B20 sensors and saves the found addresses on the `roms` variable.

```
roms = ds_sensor.scan()
```

Then, you need to execute the `convert_temp()` function to initiate a temperature reading and wait a second.

```
ds_sensor.convert_temp()
```

After that, we can read the temperature on the addresses found earlier by using the `read_temp()` method and passing the address as argument.

```
temp = ds_sensor.read_temp(rom)
```

Then, check if it has have valid temperature reading and create a `msg` variable with the sensor readings.

```
if isinstance(temp, float):
    # uncomment for Fahrenheit
    # temp = temp * (9/5) + 32.0
    msg = (b'{0:3.1f}'.format(temp))
```

By default, we're getting the temperature readings in Celsius degrees. If you want to get the temperature readings in Fahrenheit degrees you need to uncomment the following line of code.

```
# temp = temp * (9/5) + 32.0
```

# MICRO-PYTHON SCRIPT: MAIN.PY

## MQTT functions

Then, we create functions to handle MQTT tasks: `sub_cb`, `connect_and_subscribe`, and `restart_and_reconnect`. The `connect_and_subscribe`, and `restart_and_reconnect` functions were already explained in detail in previous Units.

The `sub_cb` function handles what happens when a new message is received on the topic we are subscribed to. In this case, the ESP32/ESP8266 is subscribed to the `output` topic.

When it receives a new message, we check if the content is either "on" or "off" and set the LED state accordingly.

```
print((topic, msg))
if msg == b'on':
    led.value(1)
elif msg == b'off':
    led.value(0)
```

In the following `try` and `except` statement, we create a new client by calling the `connect_and_subscribe()` function.

In case we're not able to create a new client, call the `restart_and_reconnect()` function.

```
try:
    client = connect_and_subscribe()
except OSError as e:
    restart_and_reconnect()
```

# Getting and publishing messages

## Getting and publishing messages

In the `while` loop, we check if we have a new message. New messages are then handled by the `sub_cb` function.

```
client.check_msg()
```

Then, check if 5 seconds (`readings_interval`) have passed since the last sensor reading.

```
if (time.time() - last_sensor_reading) > readings_interval:
```

If yes, call the `read_ds_sensor()` function to save a new temperature reading in the `msg` variable.

```
msg = read_ds_sensor()
```

After getting the temperature, publish it on the **temp** topic (`topic_pub`).

```
client.publish(topic_pub, msg)
```

Finally, update the last time the ESP32/ESP8266 published the temperature.

```
last_sensor_reading = time.time()
```

In case there's an error getting the temperature from the sensor, we print an error message, and wait for one second before trying again.

```
except onewire.OneWireError:  
    print('Failed to read/publish sensor readings.')  
    time.sleep(1)
```

# Importing the Node-RED flow

We won't show you how to create the Node-RED flow from scratch.

We provide the code to import the complete flow, and then we'll take a closer look at each node and see what they do.

To import the Node-RED flow, go to the GitHub repository and copy the code provided:

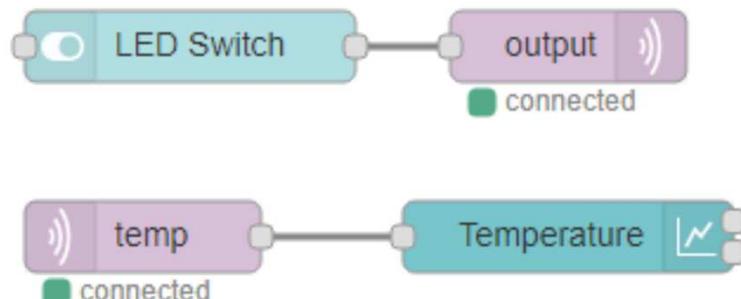
## Node-RED flow:

[https://raw.githubusercontent.com/RuiSantosdotme/ESPMicroPython/master/code/MQTT/Node\\_RED\\_Client/Node\\_RED\\_Flow.txt](https://raw.githubusercontent.com/RuiSantosdotme/ESPMicroPython/master/code/MQTT/Node_RED_Client/Node_RED_Flow.txt)

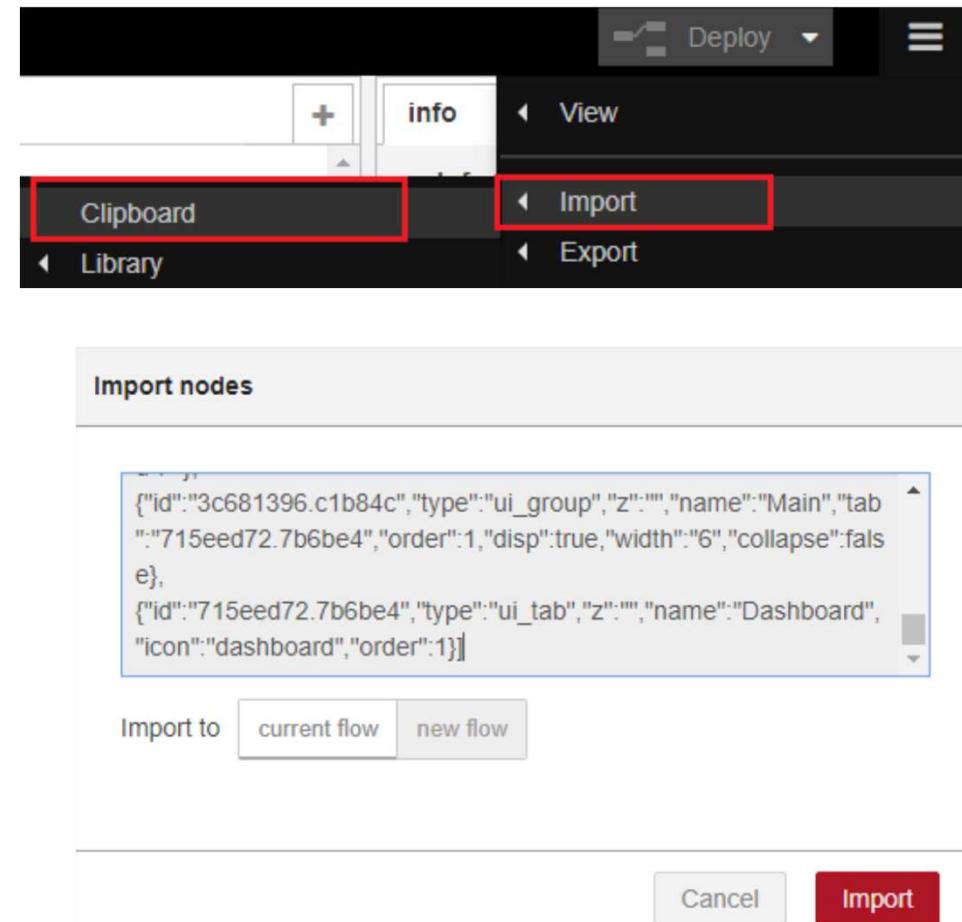
Next, in the Node-RED window, at the top right corner, select the menu, and go to **Import > Clipboard**.

Then, paste the code provided and click **Import**.

After a successful import, you should see the following nodes on your flow.



**DEPLOY**



# Understanding the flow - 1

Let's look at each node and understand what they do.

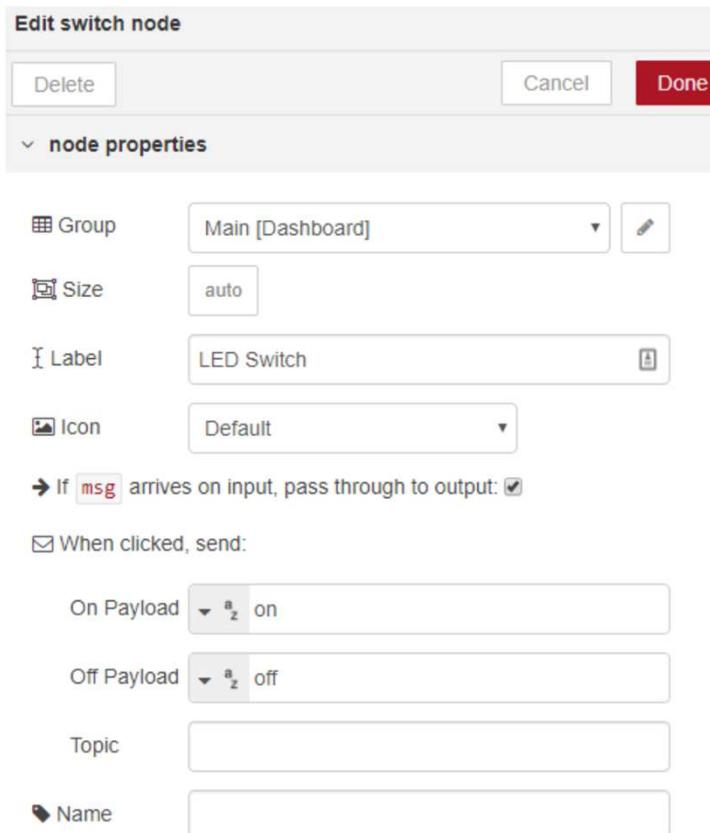
## LED Switch Node

The LED Switch node creates a switch in the UI (User Interface).

The switch is connected to an MQTT node called "output".

The switch sends different messages to the output accordingly to its state.

If you double-click on the LED Switch node, you should get something as follows:



The "Group" field indicates in which part of the UI the switch appears.

For this example, we'll leave the default settings.

When the switch is turned on ("On Payload"), we send a String message with "on" to the output (the MQTT output node).

When the switch is off, we send a "off" message to the output.

## Understanding the flow - 2

### MQTT output node

When the switch changes its state, it sends an "on" or "off" message to an output node (in this case an MQTT output node).

This node receives the messages and publishes them on a specific topic.

**Double-click the MQTT output node.**

The important fields are the "Server" and the "Topic".

The "Server" field refers to the MQTT broker.

**Because we're using the Raspberry Pi to run both Node-RED and Mosquitto broker, we can set the "Server" to "localhost".**

The "topic" field refers to the topic where the messages will be published. As we've seen previously in this Unit, the switch publishes messages on the **output** topic.

Edit mqtt out node

Delete      Cancel      Done

node properties

Server	localhost:1883	
Topic	output	
QoS		Retain
Name	Name	

Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

## Understanding the flow - 3

### MQTT temp node

The MQTT temp node listens for messages on a specific topic.

In this case, Node-RED wants to receive messages on the **temp** topic.

This is the topic the ESP32/ESP8266 is publishing the temperature readings.

**Double-click the MQTT temp node.**

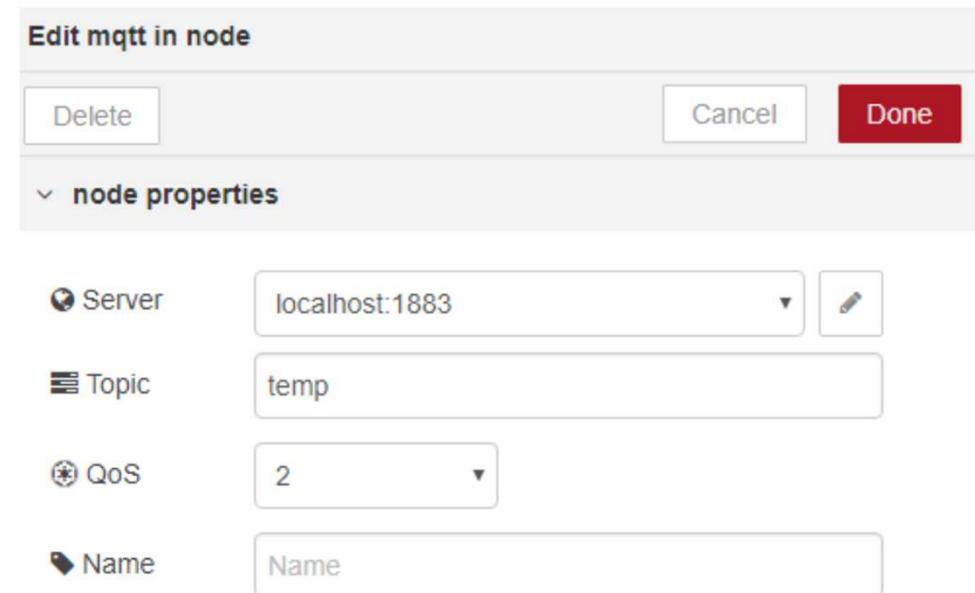
As with the previous node, we set the "Server" field to the "localhost".

Set the topic to **temp** and the QoS to 2.

QoS means quality of service. A QoS of 2 ensures that the message is always delivered exactly once.

You can learn more about MQTT QoS [here](#).

In summary, the MQTT temp node receives the messages published on the **temp** topic and sends them to a chart.



# Understanding the flow - 4

## Temperature chart node

The temperature chart node receives the temperature readings from the MQTT temp node and publishes them on a chart.

Double-click the temperature chart node.

You can select the type of chart, the X-axis and the Y-axis label, color and others.

In this example, the chart is set to display the sensor readings from the last hour.

Edit chart node

Delete      Cancel      Done

node properties

Group: Main [Dashboard]     

Size: auto

Label: Temperature

Type: Line chart       enlarge points

X-axis: last 1 hours OR 1000 points

X-axis Label: HH:mm:ss

Y-axis: min      max

Legend: None      Interpolate linear

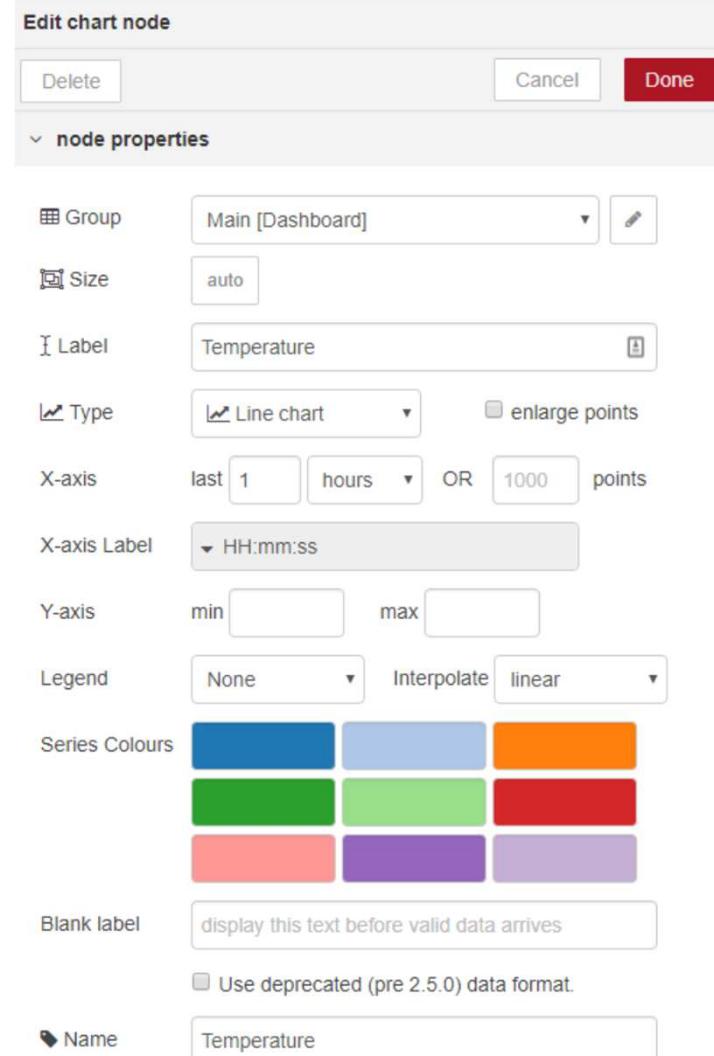
Series Colours:

Blue	Light Blue	Orange
Green	Light Green	Red
Pink	Purple	Lavender

Blank label: display this text before valid data arrives

Use deprecated (pre 2.5.0) data format.

Name: Temperature



# Node-RED

Click the **Deploy** button to save all the changes.

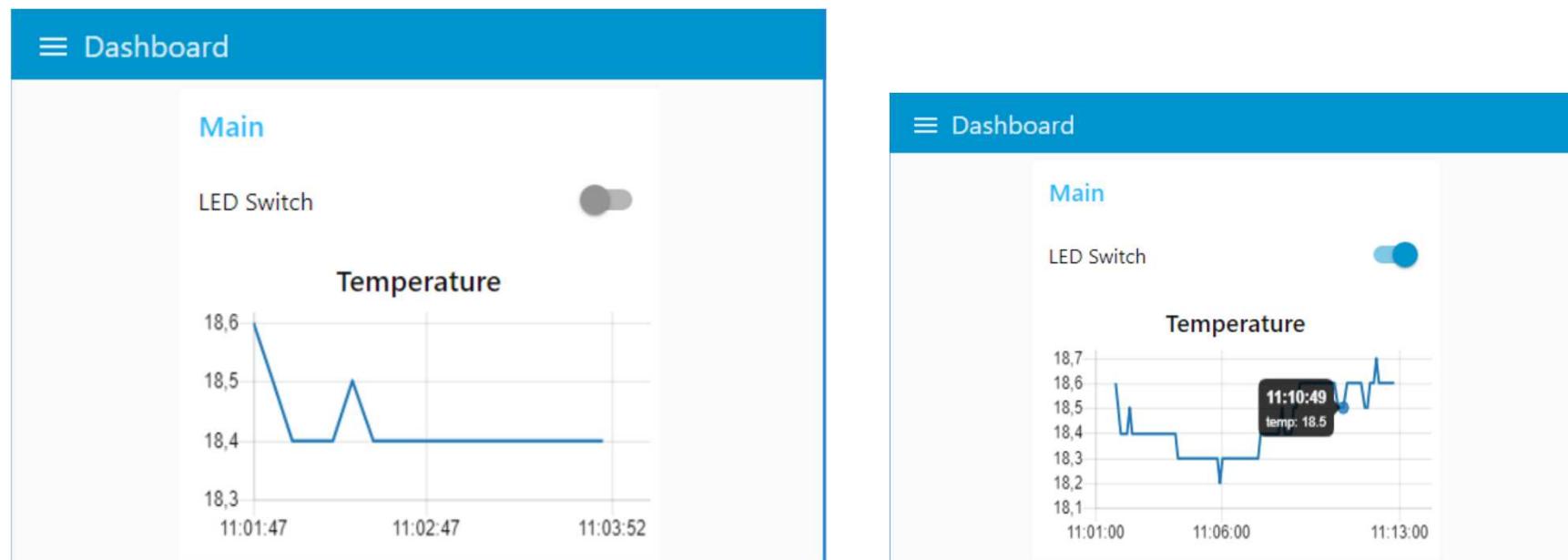
Now, your Node-RED application is ready. To access Node-RED Dashboard and see how your application looks, access any browser in your local network and type:

**[http://Your\\_RPi\\_IP\\_address:1880/ui](http://Your_RPi_IP_address:1880/ui)**

Your application should look as in the following figure.

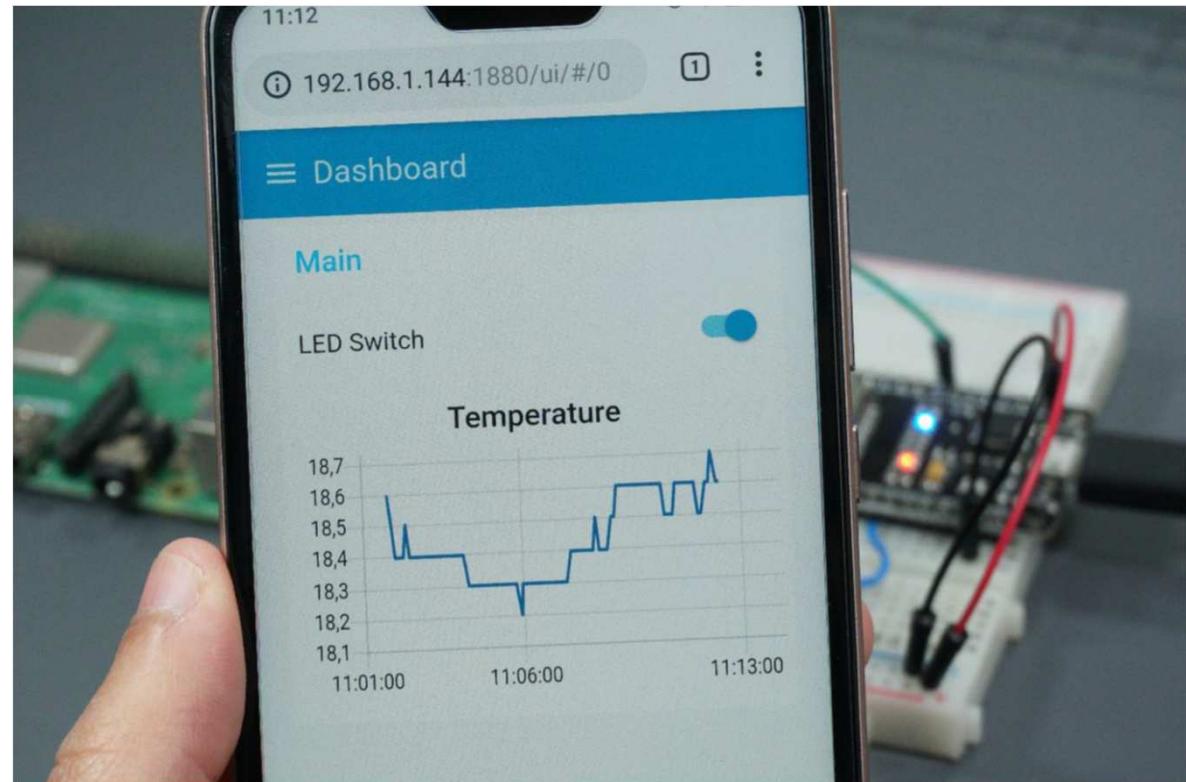
**You should have an LED switch, and a Temperature chart.**

You can check the data at a specific data point by sliding the cursor over a point in the chart.



# Wrapping Up

Now, you should be able to control the ESP32/ESP8266 on-board LED from the Node-RED Dashboard and check the latest sensor readings on the chart.



## Wrapping Up

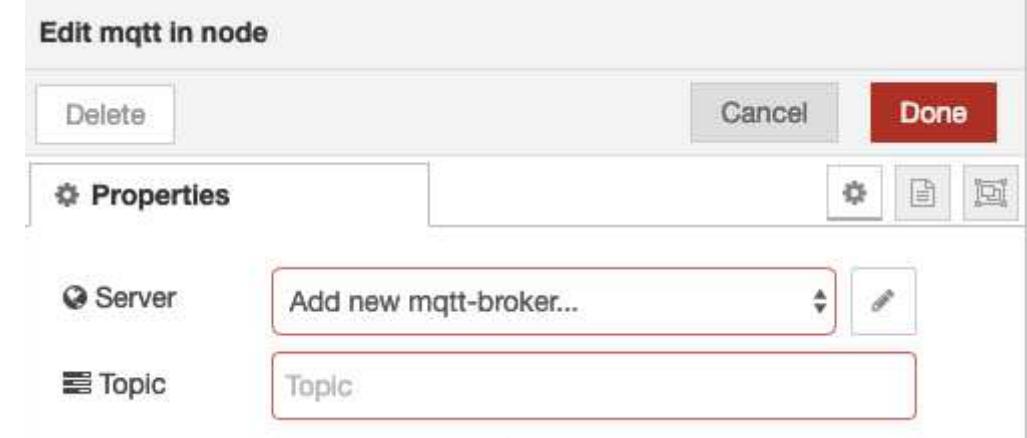
In this project, you've learned the basic concepts to publish and subscribe messages using Node-RED and how to establish a communication between Node-RED and your ESP32/ESP8266 board. You can modify the project presented here to control any other outputs and display readings from other sensors or other pieces of information you may find useful in your home automation system.

# ROUTEUR MQTT: NODE-RED

La première chose à faire et d'ajouter un Node MQTT au flow courant



Faites un double clic pour éditer les paramètres du Node. Cliquez sur le **crayon** à coté du sélecteur **Add new mqtt-broker**.



## Configurer la connexion à Mosquitto

- **Server** : indiquez l'adresse IP ou localhost si Mosquitto est installé sur le même Raspberry Pi que NodeRED
- **Port** : **1883 par défaut**
- **User et password** à l'onglet Security (conseillé)



<https://projetsdiy.fr/routeur-mqtt-node-red-sonoff-tasmota-cloudmqtt/>

# ROUTEUR MQTT: NODE-RED

Il est possible de publier des messages automatiquement pour connaître l'état du routeur en allant sur l'onglet **Messages** pour

- Le démarrage du routeur (birth message)
- L'arrêt du routeur (close message)
- Une erreur (will message) droit.

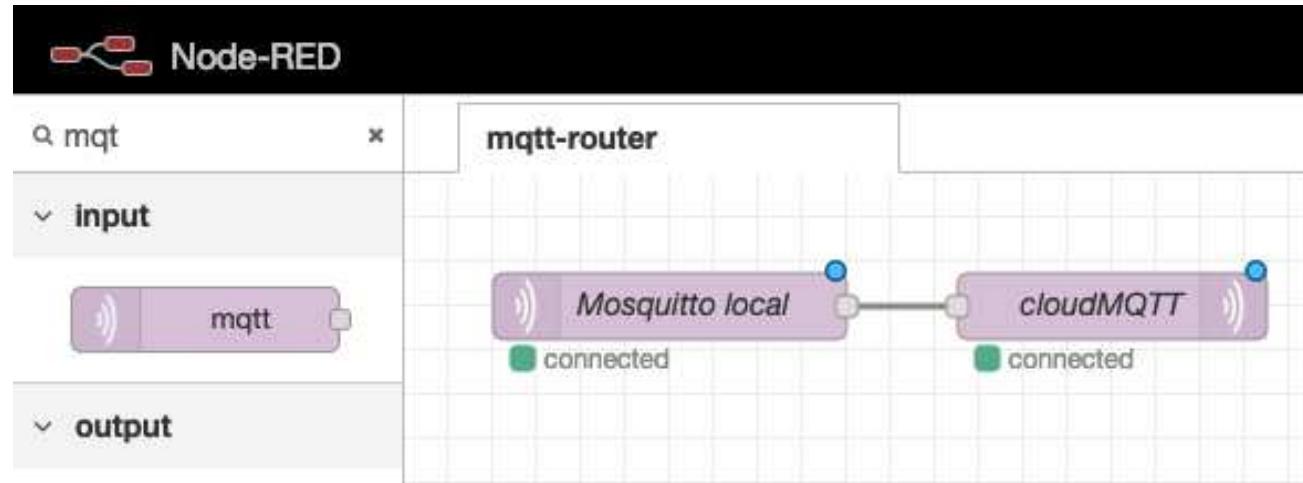
Je vous conseille de fixer le QoS à 2.  
Terminer la configuration en cliquant sur **Done** dans le coin supérieur

The screenshot shows the MQTT configuration interface in Node-RED. It includes tabs for Connection, Security, and Messages, with the Messages tab selected. Three sections are displayed:

- Message sent on connection (birth message):** Topic: node-red-router, Retain: false, QoS: 2. Payload: started.
- Message sent before disconnecting (close message):** Topic: node-red-router, Retain: false, QoS: 2. Payload: stopped.
- Message sent on an unexpected disconnection (will message):** Topic: node-red-router, Retain: false, QoS: 2. Payload: stopped.

# Déployer le routeur MQTT sur Node-RED

Il ne reste plus qu'à relier les deux Nodes et déployer le flow en cliquant sur **Deploy** dans le coin supérieur droit.



Ouvrez sur le **WebSocket UI** sur **cloudMQTT** pour vérifier que le routeur est bien démarré.

Ici, j'ai également allumé une prise connectée [Sonoff S26 hackée](#) avec le firmware Tasmota. Le message avec l'état stopped a été envoyé au moment du déploiement du flow, c'est parfaitement normal.

Received messages	
Topic	Message
node-red-router/status	stopped
node-red-router/status	online
stat/sonoff_s26/RESULT	{"POWER":"ON"}
stat/sonoff_s26/POWER	ON

# Déployer le routeur MQTT sur Node-RED

## Routeur bi-directionnel pour les accessoires Tasmota

Cette configuration **ne fonctionne malheureusement que dans un sens**. Avec le flow précédent, On va pouvoir récupérer tous les états et les valeurs (sur Homy par exemple), mais on ne pourra pas renvoyer des commandes pour allumer / éteindre un module Sonoff. Pour faire cela, il va falloir renvoyer les messages entrant sur cloudMQTT et les renvoyer vers le broker local.

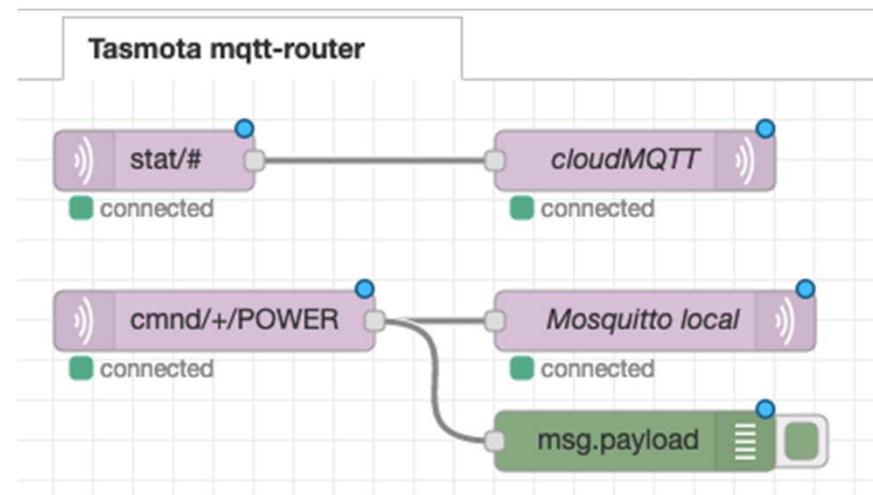
Pour que ça fonctionne, on va filtrer les messages suivants le type de topic. Tasmota publie des messages sur 3 topics différents :

- **cmnd** : commande à exécuter sur le module (on, off, toggle)
- **stat** : retour d'état d'une commande sur le module
- **tele** : information sur l'état du module.

Le premier bloc transfert tous les messages publiés sur le topic **stat/#** vers cloudMQTT. Le # permet de renvoyer de tous les accessoires Tasmota

Le second flow transfert les commandes envoyées sur cloudMQTT sur le broker local (sur lequel sont connectés les [accessoires Sonoff](#) - Tasmota sont connectés). Cette fois on filtre uniquement les messages publiés sur le topic cmnd et qui se terminent par POWER. Pour cela on utilise le + (wildcard un seul niveau\*)

N'oubliez pas d'adapter le topic si vous avez configuré un préfix sur Tasmota  
(\*) le wildcard multi-niveau # ne fonctionne pas sur Node-RED.



# Déployer le routeur MQTT sur Node-RED

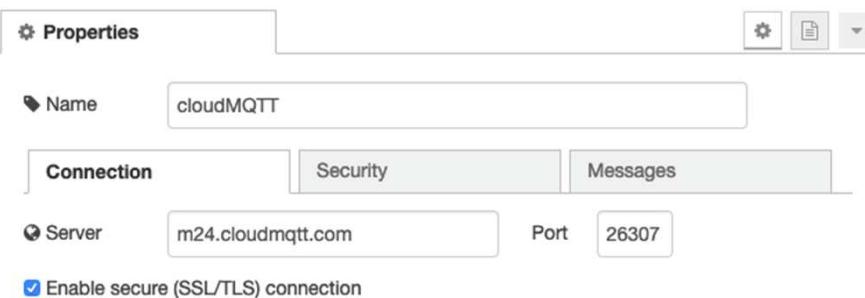
## Connecter le routeur à cloudMQTT

Ajouter un nouveau Node **MQTT** de type **Output** sur le flow. Faites un double clic pour le configurer.

Configurer la connexion avec les paramètres récupérés sur l'instance cloudMQTT :

- **Server** : l'instance cloudMQTT
- **Port** : indiquez le port SSL pour que la connexion soit sécurisée
- Cocher **Enable secure SSL/TLS connection**
- Saisissez le **user** et **password** de l'instance dans l'onglet **Security**

**Vous pouvez également configurer des messages qui permettront de connaître l'état du routeur. Voici un exemple de configuration.**



Topic	Payload	Retain	QoS
cloudmqtt-router/status	online	false	2
cloudmqtt-router/status	stopped	false	2
cloudmqtt-router/status	stopped	false	2

<https://projetsdiy.fr/routeur-mqtt-node-red-sonoff-tasmota-cloudmqtt/>

# Déployer le routeur MQTT sur Node-RED

## Code source du routeur MQTT Node-RED

Coller ce code de ce flow sur votre Node-RED puis modifiez les paramètres de connexion avant de le déployer.

```
[{"id": "4de21b42.3cb0d4", "type": "mqtt
in", "z": "1c02b62a.c1ff4a", "name": "", "topic": "stat/#", "qos": "0", "datatype": "auto", "broker": "80ccacd4.35a24
", "x": 90, "y": 100, "wires": [{"id": "ddff4f0.0c79fb"}]}, {"id": "ddff4f0.0c79fb", "type": "mqtt
out", "z": "1c02b62a.c1ff4a", "name": "cloudMQTT", "topic": "", "qos": "", "retain": "", "broker": "6c628311.0486ac",
"x": 330, "y": 100, "wires": []}, {"id": "e6695679.c6a678", "type": "mqtt
in", "z": "1c02b62a.c1ff4a", "name": "", "topic": "cmnd/+POWER", "qos": "0", "datatype": "auto", "broker": "6c628311
.0486ac", "x": 120, "y": 180, "wires": [{"id": "d756894a.745608", "z": "80b760e.ba703a"}]}, {"id": "d756894a.745608", "type": "
mqtt out", "z": "1c02b62a.c1ff4a", "name": "Mosquitto
local", "topic": "", "qos": "", "retain": "", "broker": "80ccacd4.35a24", "x": 340, "y": 180, "wires": []}, {"id": "80b76
0e.ba703a", "type": "debug", "z": "1c02b62a.c1ff4a", "name": "", "active": true, "tosidebar": true, "console": false,
"tostatus": false, "complete": false, "x": 330, "y": 240, "wires": []}, {"id": "80ccacd4.35a24", "type": "mqtt-
broker", "z": "", "name": "mosquitto", "broker": "192.168.1.90", "port": "1883", "clientid": "", "usetls": false, "com
patmode": true, "keepalive": "60", "cleansession": true, "birthTopic": "node-red-
router", "birthQos": "2", "birthRetain": "false", "birthPayload": "started", "closeTopic": "node-red-
router", "closeQos": "2", "closePayload": "stopped", "willTopic": "node-red-
router", "willQos": "2", "willPayload": "stopped"}, {"id": "6c628311.0486ac", "type": "mqtt-
broker", "z": "", "name": "cloudMQTT", "broker": "m24.cloudmqtt.com", "port": "26307", "tls": "", "clientid": "", "use
tls": true, "compatmode": true, "keepalive": "60", "cleansession": true, "birthTopic": "node-red-
router/status", "birthQos": "2", "birthRetain": "false", "birthPayload": "online", "closeTopic": "node-red-
router/status", "closeQos": "2", "closePayload": "stopped", "willTopic": "node-red-
router/status", "willQos": "2", "willPayload": "stopped"}]
```

**Voilà, vous pouvez maintenant connecter autant d'accessoires domotiques et objets connectés MQTT et les piloter hors de chez vous depuis Homy ou une appli.**

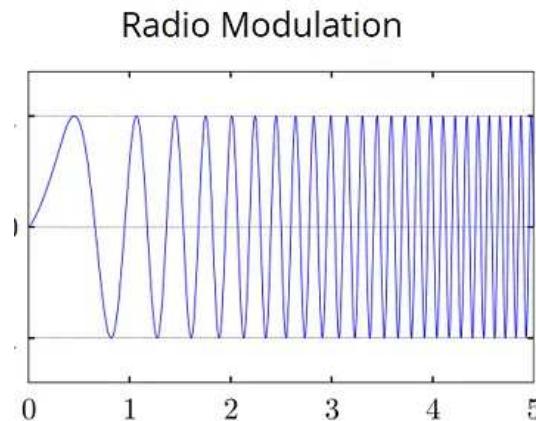
**Rien ne vous empêche d'adapter le principe pour l'adapter à d'autres firmware (Espurna, ESPEasy, code Arduino ou Python...)**

# TP10 – LoRa ON ESP32

# LoRa

What is LoRa?

LoRa is a wireless data communication technology that uses a radio modulation technique that can be generated by Semtech LoRa transceiver chips.



LoRa transceiver chips

This modulation technique allows long range communication of small amounts of data (which means a low bandwidth), high immunity to interference, while minimizing power consumption. So, it allows long distance communication with low power requirements.



Long distance communication



Small amounts of data (low bandwidth)



High immunity to interference



Low power consumption

# LoRa

## LoRa Frequencies :

LoRa uses unlicensed frequencies that are available worldwide. These are the most widely used frequencies:

- 868 MHz for Europe
- 915 MHz for North America
- 433 MHz band for Asia

Because these bands are unlicensed, anyone can freely use them without paying or having to get a license. Check the [frequencies used in your country](#).

## LoRa Applications :

LoRa long range and low power features, makes it perfect for battery-operated sensors and low-power applications in:

- Internet of Things (IoT)
- Smart home
- Machine-to-machine communication
- And much more...

So, LoRa is a good choice for sensor nodes running on a coin cell or solar powered, that transmit small amounts of data.

Keep in mind that LoRa is not suitable for projects that:

- Require high data-rate transmission;
- Need very frequent transmissions;
- Or are in highly populated networks.



# LoRa Topologies

You can use LoRa in:

- **Point to point communication**
- **Or build a LoRa network (using LoRaWAN for example)**

## Point to Point Communication

In point to point communication, two LoRa enabled devices talk with each other using RF signals.

For example, this is useful to exchange data between two ESP32 boards equipped with LoRa transceiver chips that are relatively far from each other or in environments without Wi-Fi coverage.

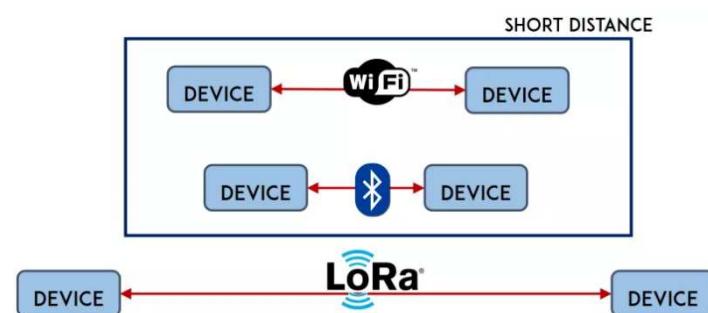
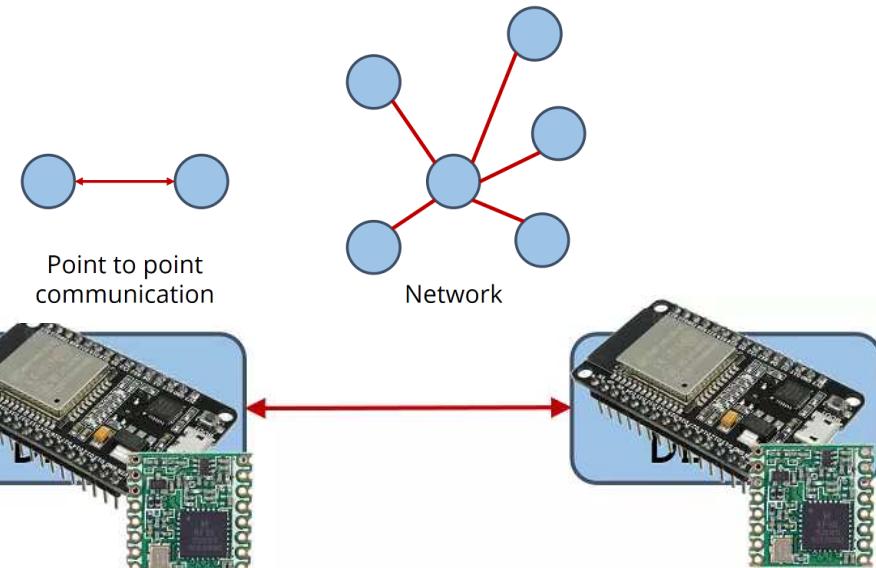
Unlike Wi-Fi or Bluetooth that only support short distance communication, two LoRa devices with a proper antenna can exchange data over a long distance.

**You can easily configure your ESP32 with a LoRa chip to transmit and receive data reliably at more than 200 meters distance (you can get better results depending on your environment and LoRa settings).** There are also other LoRa solutions that easily have a range of more than 30Km.

## LoRaWAN

You can also build a LoRa network using LoRaWAN.

The LoRaWAN protocol is a Low Power Wide Area Network (LPWAN) specification derived from LoRa technology standardized by the LoRa Alliance. We won't explore LoRaWAN in this tutorial, but for more information you can check the [LoRa Alliance](#) and [The Things Network](#) websites.

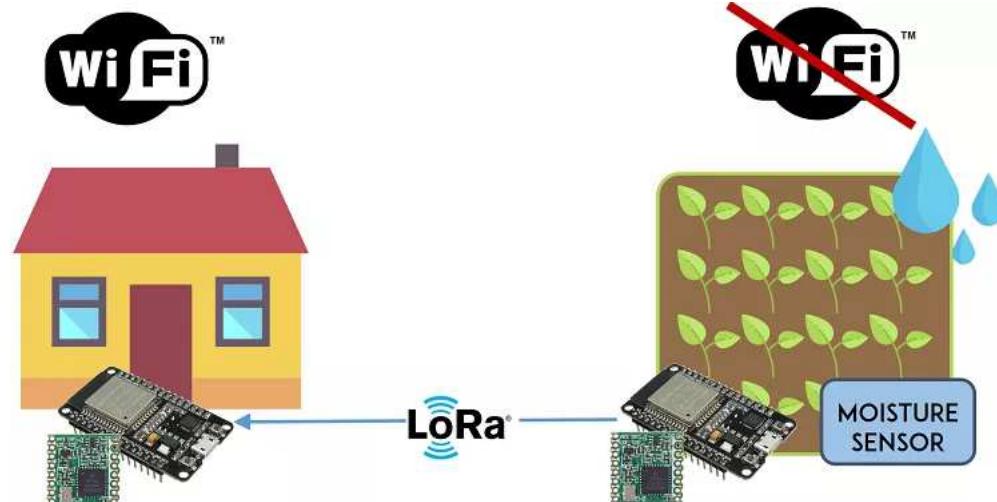


# LoRa

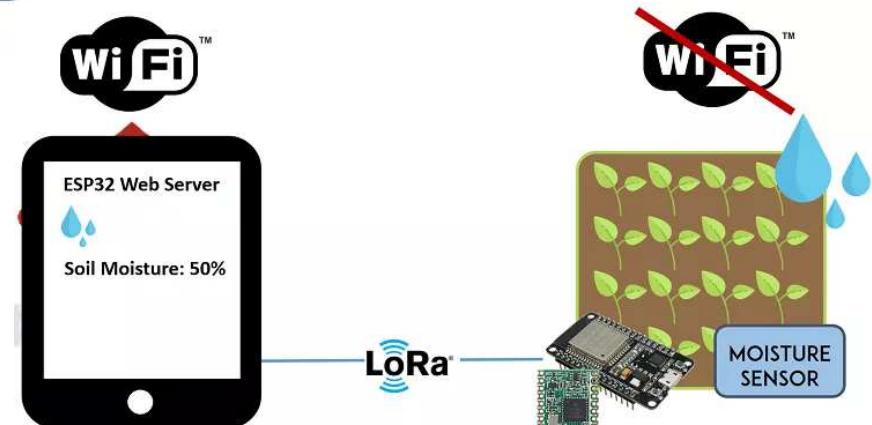
## How can LoRa be useful in your home automation projects?

Let's take a look at a practical application.

Imagine that you want to measure the moisture in your field. Although, it is not far from your house, it probably doesn't have Wi-Fi coverage. So, you can build a sensor node with an ESP32 and a moisture sensor, that sends the moisture readings once or twice a day to another ESP32 using LoRa.



The later ESP32 has access to Wi-Fi, and it can run a web server that displays the moisture readings.



# LoRa module – RFM95/96/97/98

## RFM95/96/97/98 - Low Power Long Range Transceiver Module V1.0

### GENERAL DESCRIPTION

The RFM95/96/97/98 transceivers feature the LoRa™ long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimising current consumption.

Using Hope RF's patented LoRa™ modulation technique RFM95/96/97/98 can achieve a sensitivity of over -148dBm using a low cost crystal and bill of materials. The high sensitivity combined with the integrated +20 dBm power amplifier yields industry leading link budget making it optimal for any application requiring range or robustness. LoRa™ also provides significant advantages in both blocking and selectivity over conventional modulation techniques, solving the traditional design compromise between range, interference immunity and energy consumption.

These devices also support high performance (G)FSK modes for systems including WMBus, IEEE802.15.4g. The RFM95/96/97/98 deliver exceptional phase noise, selectivity, receiver linearity and IIP3 for significantly lower current consumption than competing devices.

### KEY PRODUCT FEATURES

- ◆ LoRa™ Modem.
- ◆ 168 dB maximum link budget.
- ◆ +20 dBm - 100 mW constant RF output vs. V supply.
- ◆ +14 dBm high efficiency PA.
- ◆ Programmable bit rate up to 300 kbps.
- ◆ High sensitivity: down to -148 dBm.
- ◆ Bullet-proof front end: IIP3 = -12.5 dBm.
- ◆ Excellent blocking immunity.
- ◆ Low RX current of 10.3 mA, 200 nA register retention.
- ◆ Fully integrated synthesizer with a resolution of 61 Hz.
- ◆ FSK, GFSK, MSK, GMSK, LoRa™ and OOK modulation.
- ◆ Built-in bit synchronizer for clock recovery.
- ◆ Preamble detection.
- ◆ 127 dB Dynamic Range RSSI.
- ◆ Automatic RF Sense and CAD with ultra-fast AFC.
- ◆ Packet engine up to 256 bytes with CRC.
- ◆ Built-in temperature sensor and low battery indicator.
- ◆ Module Size: 16\*16mm

[https://www.aliexpress.com/snapshot/0.html?spm=a2g0s.buyer\\_leave\\_review.0.0.3a846c1bhgQ&Uif&orderId=3001775197871894&productId=32831799362](https://www.aliexpress.com/snapshot/0.html?spm=a2g0s.buyer_leave_review.0.0.3a846c1bhgQ&Uif&orderId=3001775197871894&productId=32831799362)



### 1.2. Product Versions

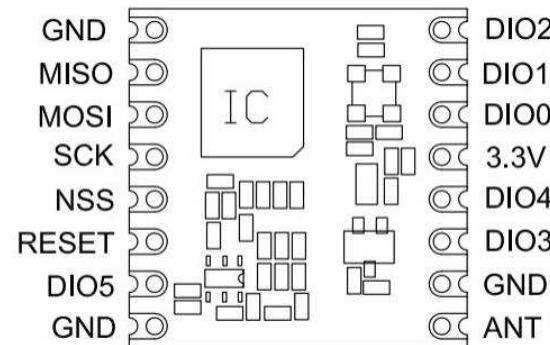
The features of the three product variants are detailed in the following table.

Table 48 RFM95/96/97/98 Device Variants and Key Parameters

Part Number	Frequency Range	Spreading Factor	Bandwidth	Effective Bitrate	Est. Sensitivity
RFM95	868/915 MHz	6 - 12	7.8 - 500 kHz	.018 - 37.5 kbps	-111 to -148 dBm
RFM97	868/915 MHz	6 - 9	7.8 - 500 kHz	0.11 - 37.5 kbps	-111 to -139 dBm
RFM96/RFM98	433/470MHz	6 - 12	7.8 - 500 kHz	.018 - 37.5 kbps	-111 to -148 dBm

### 1.3. Pin Diagram

The following diagram shows the pin arrangement, top view.



### 1.4. Pin Description

Number	Name	Type	Description	
			Description Stand Alone Mode	
1	GND	-		Ground
2	MISO	I		SPI Data output
3	MOSI	O		SPI Data input
4	SCK	I		SPI Clock input
5	NSS	I		SPI Chip select input
6	RESET	I/O		Reset trigger input
7	DIO5	I/O		Digital I/O, software configured
8	GND	-		Ground
9	ANT	-		RF signal output/input
10	GND	-		Ground
11	DIO3	I/O		Digital I/O, software configured
12	DIO4	I/O		Digital I/O, software configured
13	3.3V	-		Supply voltage
14	DIO0	I/O		Digital I/O, software configured
15	DIO1	I/O		Digital I/O, software configured
16	DIO2	I/O		Digital I/O, software configured

# LoRa module – RFM95/96/97/98

## Features

This section gives a high-level overview of the functionality of the RFM95/96/97/98 low-power, highly integrated transceiver. The following figure shows a simplified block diagram of the RFM95/96/97/98.

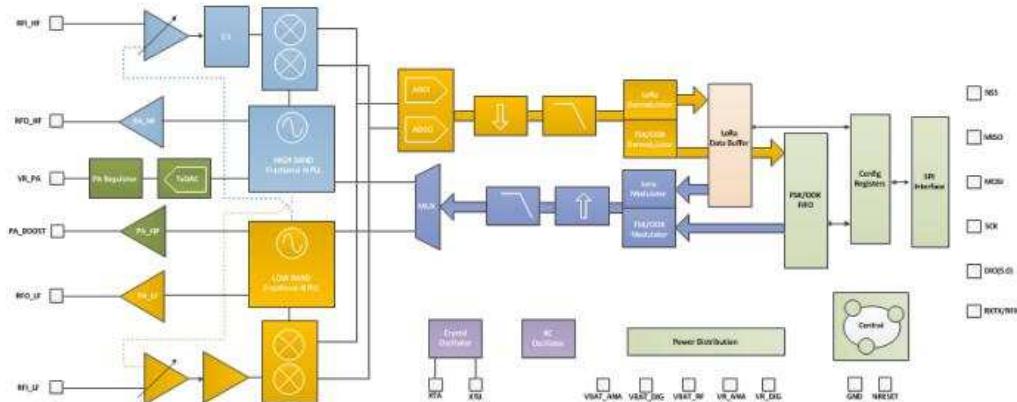


Figure 3. RFM95/96/97/98 Block Schematic Diagram

RFM95/96/97/98 is a half-duplex, low-IF transceiver. Here the received RF signal is first amplified by the LNA. The LNA inputs are single ended to minimise the external BoM and for ease of design. Following the LNA inputs, the conversion to differential is made to improve the second order linearity and harmonic rejection. The signal is then down-converted to in-phase and quadrature (I&Q) components at the intermediate frequency (IF) by the mixer stage. A pair of sigma delta ADCs then perform data conversion, with all subsequent signal processing and demodulation performed in the digital domain. The digital state machine also controls the automatic frequency correction (AFC), received signal strength indicator (RSSI) and automatic gain control (AGC). It also features the higher-level packet and protocol level functionality of the top level sequencer (TLS).

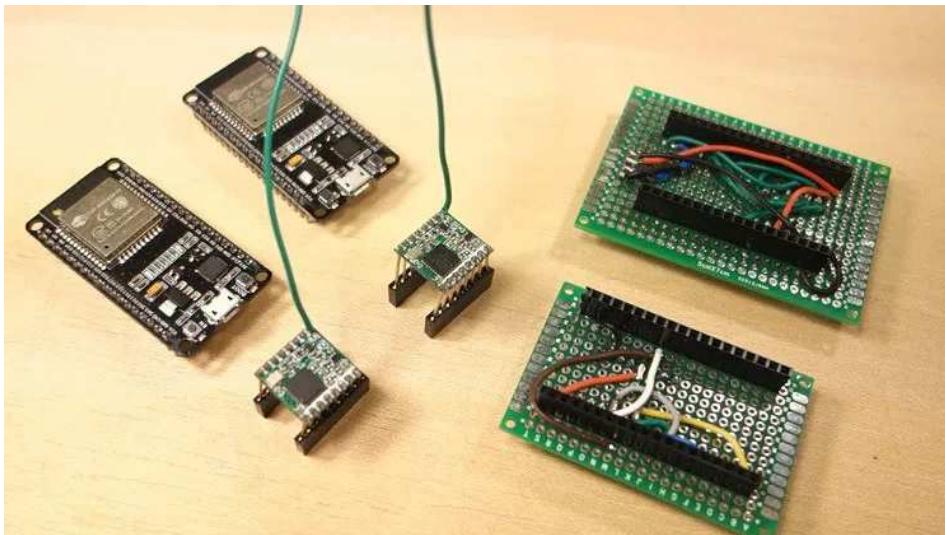
The frequency synthesisers generate the local oscillator (LO) frequency for both receiver and transmitter, one covering the lower UHF bands (up to 525 MHz), and the other one covering the upper UHF bands (from 860 MHz). The PLLs are optimized for user-transparent low lock time and fast auto-calibrating operation. In transmission, frequency modulation is performed digitally within the PLL bandwidth. The PLL also features optional pre-filtering of the bit stream to improve spectral purity.

RFM95/96/97/98 feature three distinct RF power amplifiers. Two of those, connected to RFO\_LF and RFO\_HF, can deliver up to +14 dBm, are unregulated for high power efficiency and can be connected directly to their respective RF receiver inputs via a pair of passive components to form a single antenna port high efficiency transceiver. The third PA, connected to the PA\_BOOST pin and can deliver up to +20 dBm via a dedicated matching network. Unlike the high efficiency PAs, this high-stability PA covers all frequency bands that the frequency synthesizer addresses.

RFM95/96/97/98 also include two timing references, an RC oscillator and a 32 MHz crystal oscillator.

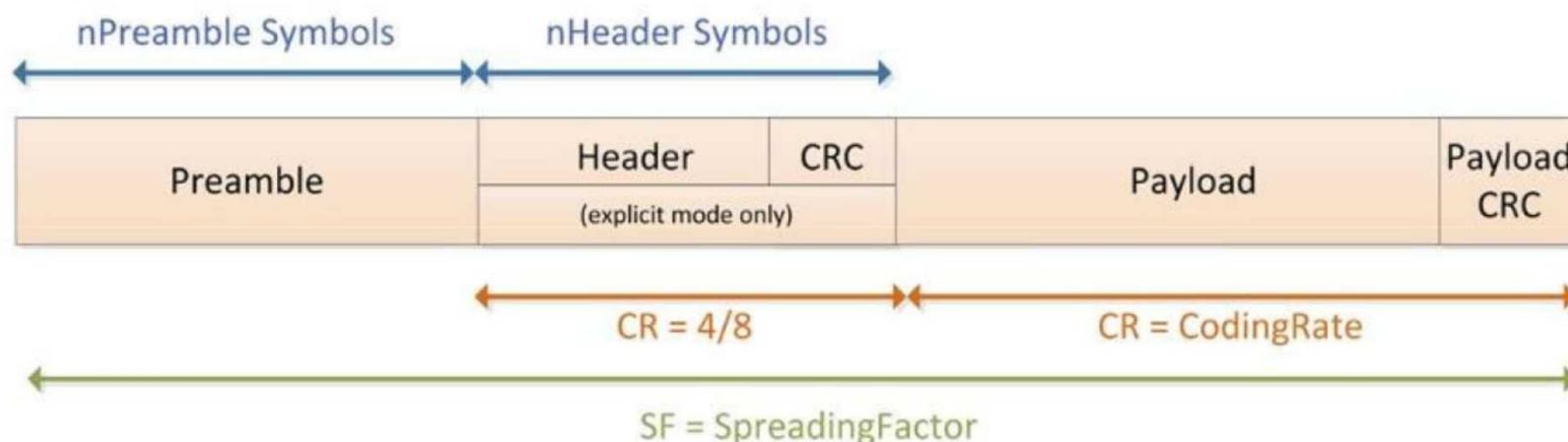
# LoRa - Schematics

In this section we'll show you how to get started with LoRa with your ESP32 using Arduino IDE. As an example, we'll build a simple LoRa Sender and a LoRa Receiver. The LoRa Sender will be sending a "hello" message followed by a counter for testing purposes. This message can be easily replaced with useful data like sensor readings or notifications



To follow this part you need the following components:

- [2x ESP32 DOIT DEVKIT V1 Board](#)
- [2x LoRa Transceiver modules \(RFM95\)](#)
- RFM95 LoRa breakout board (optional)
- [Jumper wires](#)
- [Breadboard](#) or [stripboard](#)



## LoRa - Alternative

- 2x TTGO LoRa32 SX1276 OLED

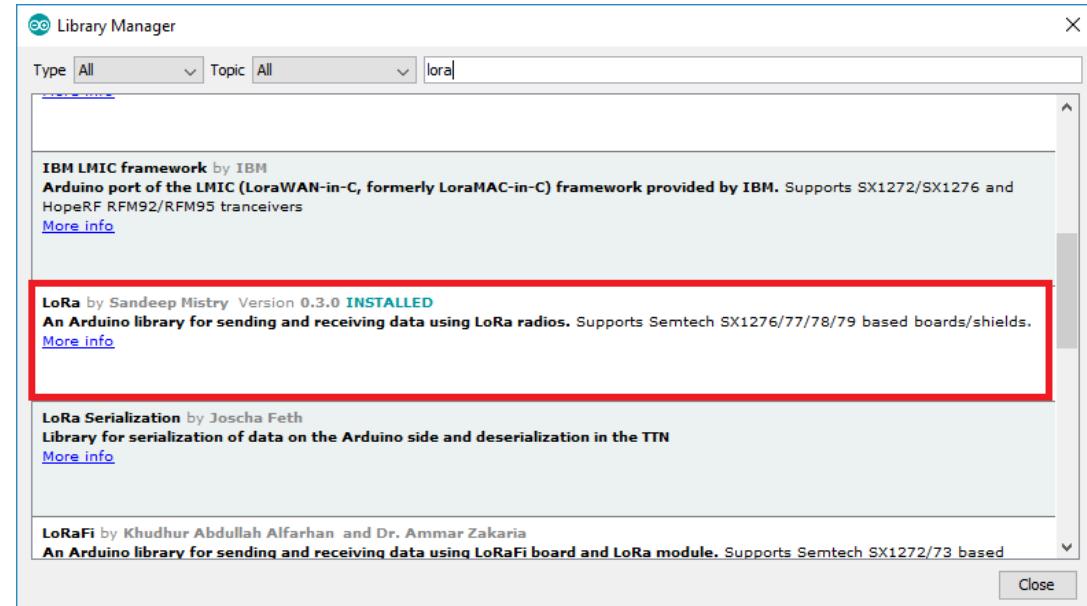
Instead of using an ESP32 and a separated LoRa transceiver module, there are ESP32 development boards with a LoRa chip and an OLED built-in, which makes wiring much simpler. If you have one of those boards, you can follow: [TTGO LoRa32 SX1276 OLED Board: Getting Started with Arduino IDE](#).



# Preparing the project: Arduino IDE and RFM modules

## Installing the LoRa Library

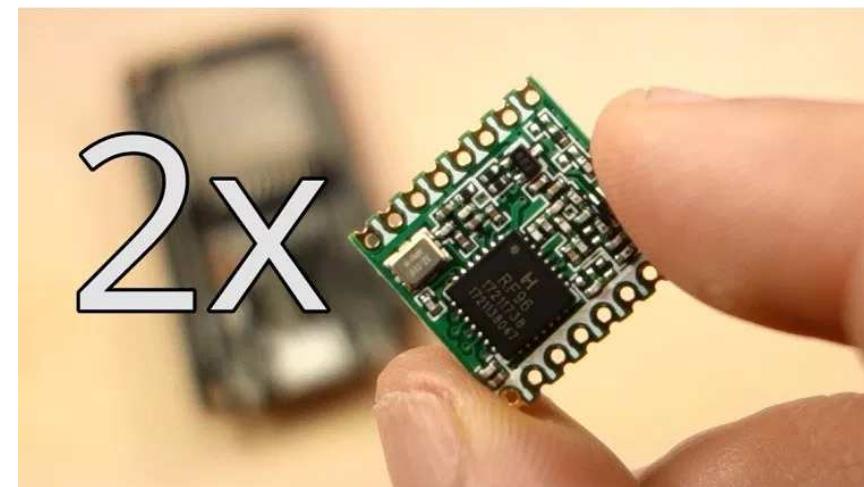
There are several libraries available to easily send and receive LoRa packets with the ESP32. In this example we'll be using the [arduino-LoRa library by sandeep mistry](#).



## Getting LoRa Tranceiver Modules

To send and receive LoRa messages with the ESP32 we'll be using the [RFM95 transceiver module](#). All LoRa modules are transceivers, which means they can send and receive information. You'll need 2 of them.

You can also use other compatible modules like Semtech SX1276/77/78/79 based boards including: RFM96W, RFM98W, etc...

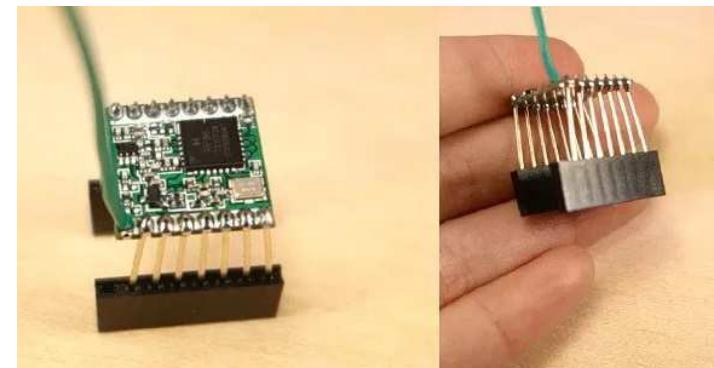


# WIRING

There are a few options that you can use to access the transceiver pins.

- You may solder some wires directly to the transceiver;
- Break header pins and solder each one separately;
- Or you can buy a breakout board that makes the pins breadboard friendly.

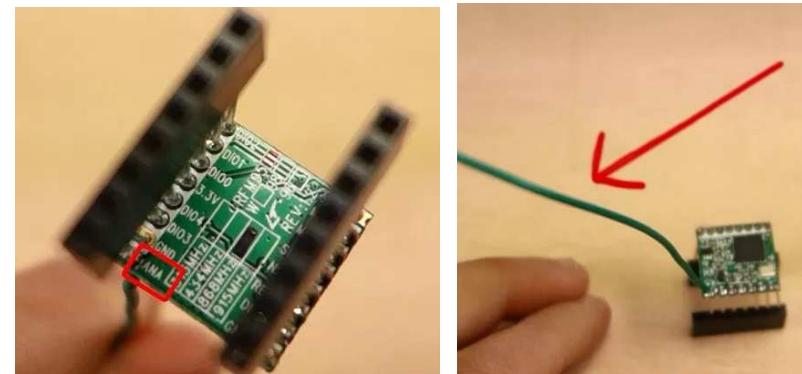
We've soldered a header to the module as shown in the figure below.



## Antenna

The RFM95 transceiver chip requires an external antenna connected to the ANA pin.

You can connect a “real” antenna, or you can make one yourself by using a conductive wire as shown in the figure below. Some breakout boards come with a special connector to add a proper antenna.



### The wire length depends on the frequency:

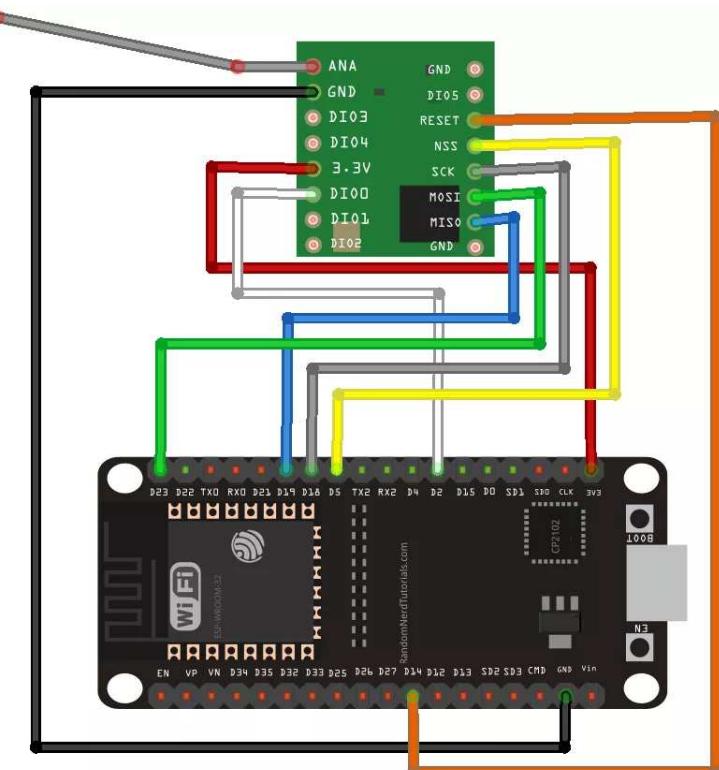
- 868 MHz: 86,3 mm (3.4 inch) (*Remainder:  $l = \lambda/4 = c/4f \Rightarrow l = 3 \times 10^8 / (4 \times 868 \times 10^6) = 86,4 \text{mm}$* )
- 915 MHz: 81,9 mm (3.22 inch)
- 433 MHz: 173,1 mm (6.8 inch)

For our module we need to use a 86,3 mm wire soldered directly to the transceiver's ANA pin. Note that using a proper antenna will extend the communication range.

**Important:** you MUST attach an antenna to the module.

# Wiring the RFM95 LoRa Transceiver Module

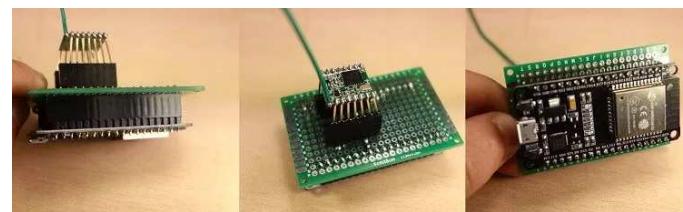
The RFM95 LoRa transceiver module communicates with the ESP32 using SPI communication protocol. So, we'll use the ESP32 default SPI pins. Wire both ESP32 boards to the corresponding transceiver modules as shown in the next schematic diagram:



Here's the connections between the RFM95 LoRa transceiver module and the ESP32:

- ANA: Antenna
- GND: **GND**
- DI03: don't connect
- DI04: don't connect
- 3.3V: **3.3V**
- DI00: **GPIO 2**
- DI01: don't connect
- DI02: don't connect
- GND: don't connect
- DI05: don't connect
- RESET: **GPIO 14**
- NSS: **GPIO 5**
- SCK: **GPIO 18**
- MOSI: **GPIO 23**
- MISO: **GPIO 19**
- GND: don't connect

**Note:** the RFM95 transceiver module has 3 GND pins. It doesn't matter which one you use, but you need to connect at least one.



# The LoRa Sender Sketch

This sketch transmits messages every 10 seconds using LoRa. It sends a "hello" followed by a number that is incremented in every message

**It starts by including the needed libraries.**

**Then, define the pins used by your LoRa module.** If you've followed the previous schematic, you can use the pin definition used in the code.

If you're using an ESP32 board with LoRa built-in, check the pins used by the LoRa module in your board and make the right pin assignment.

**In the setup(), you initialize a serial communication.**

**Set the pins for the LoRa module.**

**And initialize the transceiver module with a specified frequency.**

LoRa transceiver modules listen to packets within its range. It doesn't matter where the packets come from. To ensure you only receive packets from your sender, you can set a sync word (ranges from 0 to 0xFF).

Both the receiver and the sender need to use the same sync word. This way, the receiver ignores any LoRa packets that don't contain that sync word.

Next, in the loop() you send the LoRa packets. You initialize a packet with the beginPacket() method.

You write data into the packet using the print() method. As you can see in the following two lines, we're sending a hello message followed by the counter

Then, close the packet with the endPacket() method.

After this, the counter message is incremented by one in every loop, which happens every 10 seconds.

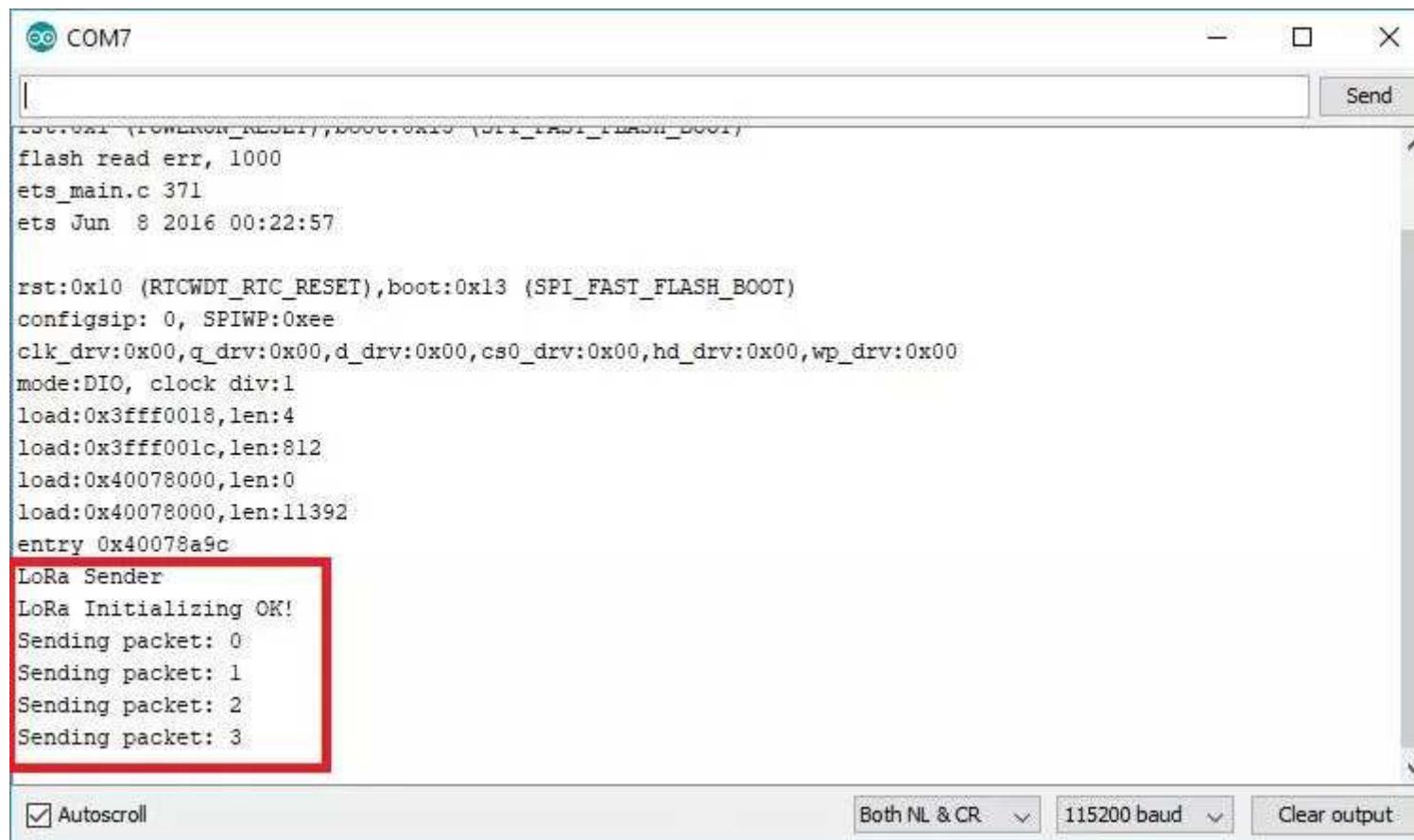
```
1 //*****
2 // Modified from the examples of the Arduino LoRa library
3 // More resources: https://randomnerdtutorials.com
4 *****/
5 #include <SPI.h>
6 #include <LoRa.h>
7
8 //define the pins used by the transceiver module
9 #define ss 5
10 #define rst 14
11 #define dio0 2
12
13 int counter = 0;
14
15 void setup() {
16     //initialize Serial Monitor
17     Serial.begin(115200);
18     while (!Serial);
19     Serial.println("LoRa Sender");
20
21     //setup LoRa transceiver module
22     LoRa.setPins(ss, rst, dio0);
23
24     //replace the LoRa.begin(--E-) argument with your location's frequency
25     //433E6 for Asia //866E6 for Europe //915E6 for North America
26     while (!LoRa.begin(866E6)) {
27         Serial.println(".");
28         delay(500);
29     }
30     // Change sync word (0xF3) to match the receiver
31     // The sync word assures you don't get LoRa messages from other LoRa transceivers
32     // ranges from 0-0xFF
33     LoRa.setSyncWord(0xF3);
34     Serial.println("LoRa Initializing OK!");
35 }
36
37 void loop() {
38     Serial.print("Sending packet: ");
39     Serial.println(counter);
40
41     //Send LoRa packet to receiver
42     LoRa.beginPacket();
43     LoRa.print("hello ");
44     LoRa.print(counter);
45     LoRa.endPacket();
46
47     counter++;
48
49     delay(10000);
50 }
51
```

## Testing the Sender Sketch

Upload the code to your ESP32 board.

Make sure you have the right board and COM port selected.

After that, open the Serial Monitor, and press the ESP32 enable button. You should see a success message as shown in the figure below. The counter should be incremented every 10 seconds.



```
flash read err, 1000
ets_main.c 371
ets Jun 8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:812
load:0x40078000,len:0
load:0x40078000,len:11392
entry 0x40078a9c

LoRa Sender
LoRa Initializing OK!
Sending packet: 0
Sending packet: 1
Sending packet: 2
Sending packet: 3
```

The screenshot shows a Windows-style serial monitor window titled "COM7". The window displays the boot logs of an ESP32 chip, followed by the output from a "LoRa Sender" sketch. The "LoRa Sender" output is highlighted with a red rectangular box. It shows the initialization of the LoRa module and the transmission of four packets at intervals of approximately 10 seconds. The bottom of the window includes standard serial monitor controls: "Autoscroll" (checked), "Both NL & CR" (selected), "115200 baud" (selected), and "Clear output".

# The LoRa Receiver Sketch

Now, grab another ESP32 and upload the LoRa receiver sketch.

This sketch listens for LoRa packets with the sync word you've defined and prints the content of the packets on the Serial Monitor, as well as the RSSI.

The RSSI measures the relative received signal strength.

This sketch is very similar to the previous one. Only the loop() is different!:

You might need to change the frequency and the syncword to match the one used in the sender sketch.

In the loop() the code checks if a new packet has been received using the parsePacket() method.

If there's a new packet,

we'll read its content while it is available.

To read the incoming data you use the readString() method.

The incoming data is saved on the LoRaData variable and printed in the Serial Monitor.

Finally, the code print the RSSI of the received packet in dB.

```
32
33 void loop() {
34     // try to parse packet
35     int packetSize = LoRa.parsePacket();
36     if (packetSize) {
37         // received a packet
38         Serial.print("Received packet ''");
39
40         // read packet
41         while (LoRa.available()) {
42             String LoRaData = LoRa.readString();
43             Serial.print(LoRaData);
44         }
45
46         // print RSSI of packet
47         Serial.print("' with RSSI ");
48         Serial.println(LoRa.packetRssi());
49     }
50 }
```

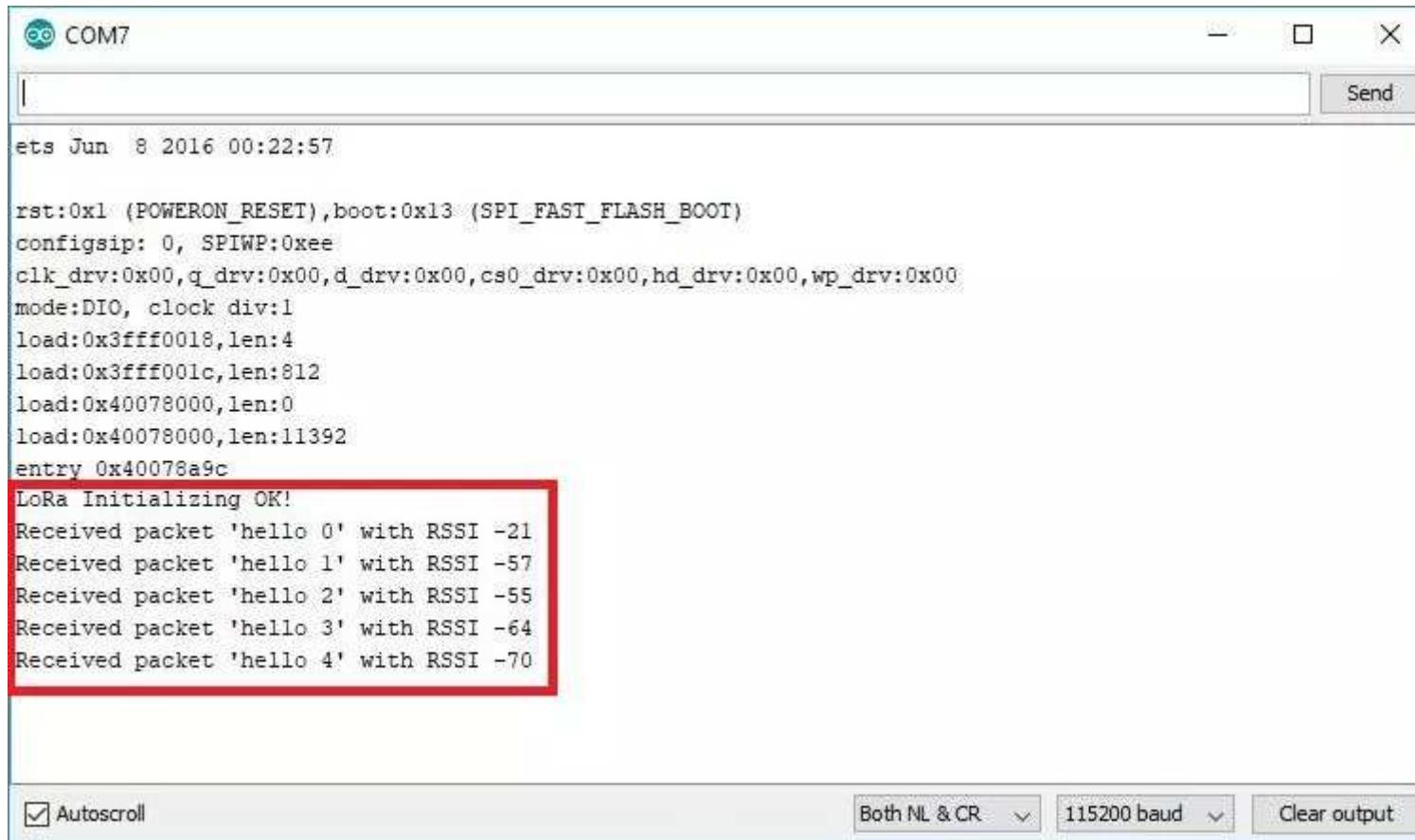
## Testing the LoRa Receiver Sketch

Upload this code to your ESP32.

At this point you should have two ESP32 boards with different sketches:

- the sender and
- the receiver.

Open the Serial Monitor for the LoRa Receiver, **and press the LoRa Sender enable button.** You should start getting the LoRa packets on the receiver.



The screenshot shows the Arduino Serial Monitor window titled "COM7". The text area displays the following log:

```
ets Jun 8 2016 00:22:57

rst:0xl (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:812
load:0x40078000,len:0
load:0x40078000,len:11392
entry 0x40078a9c

LoRa Initializing OK!
Received packet 'hello 0' with RSSI -21
Received packet 'hello 1' with RSSI -57
Received packet 'hello 2' with RSSI -55
Received packet 'hello 3' with RSSI -64
Received packet 'hello 4' with RSSI -70
```

At the bottom of the window, there are three buttons: "Autoscroll" (checked), "Both NL & CR" (selected), and "115200 baud". There is also a "Clear output" button.

# Wrapping Up

In summary, in this tutorial we've shown you the basics of LoRa technology:

- LoRa is a radio modulation technique;
- LoRa allows long-distance communication of small amounts of data and requires low power;
- You can use LoRa in point to point communication or in a network;
- **LoRa can be especially useful if you want to monitor sensors that are not covered by your Wi-Fi network and that are several meters apart.**

## Taking It Further

Now, you should test the communication range between the Sender and the Receiver on your area.

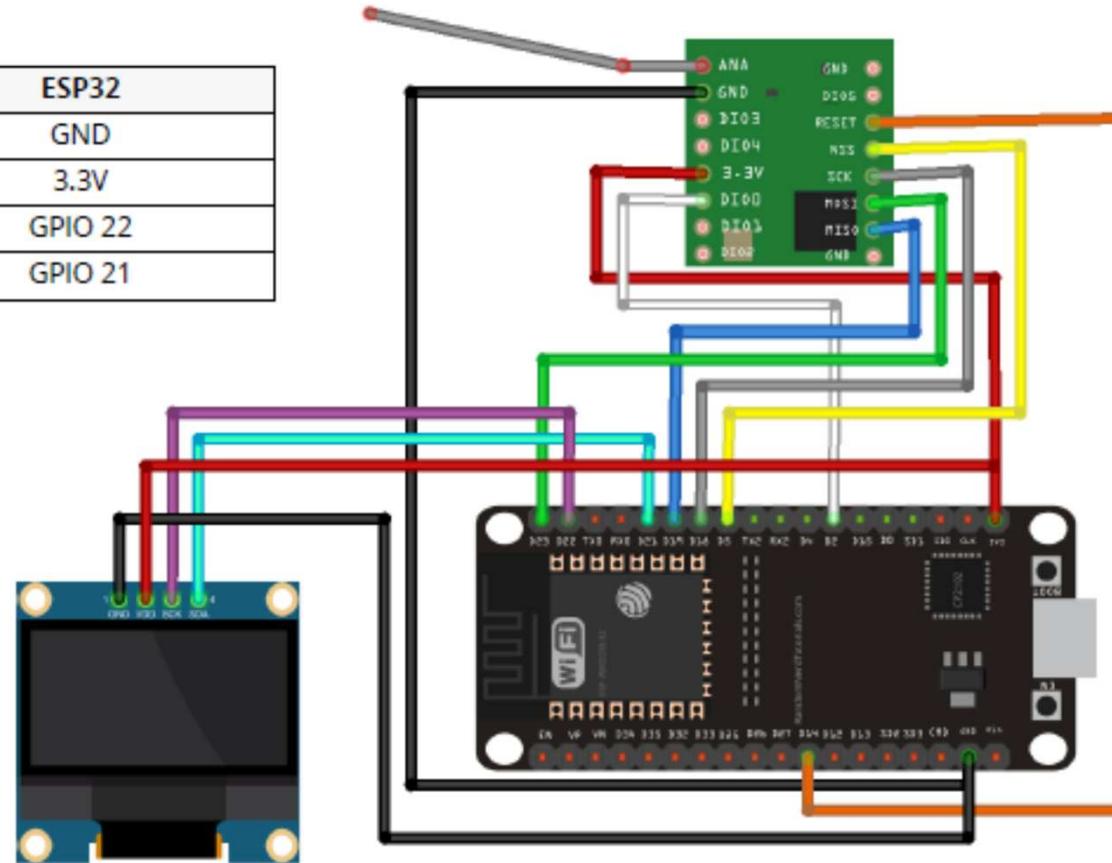
The communication range greatly varies depending on your environment (if you live in a rural or urban area with a lot of tall buildings).

**To test the communication range you can add an OLED display to the LoRa receiver and go for a walk to see how far you can get a communication.**



# Taking It Further – OLED version

OLED Display	ESP32
GND	GND
VCC	3.3V
SCK	GPIO 22
SDA	GPIO 21



(This schematic uses the *ESP32 DEVKIT V1* module version with 36 GPIOs – if you’re using another model, please check the pinout for the board you’re using.)

Define the OLED width and height:

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels  
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

You also need to create an object for the OLED display.

```
#define OLED_RESET 4  
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);  
setup()
```

In the **setup()** initialize the display

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)){  
    Serial.println(F("SSD1306 allocation failed"));  
    for(;); // Don't proceed, loop forever  
}
```

Print the message "LoRa Receiver";

```
display.clearDisplay();  
display.setTextSize(1);  
display.setTextColor(WHITE);  
display.setCursor(0,0);  
display.println("LoRa Receiver");  
display.display();
```

If the LoRa module is properly initialized, print a success message.

```
Serial.println("LoRa Initializing OK!");  
display.setCursor(0,10);  
display.println("LoRa Initializing OK!");  
display.display();
```

The incoming data is saved on the LoRaData variable and printed on the OLED display and in the serial monitor.

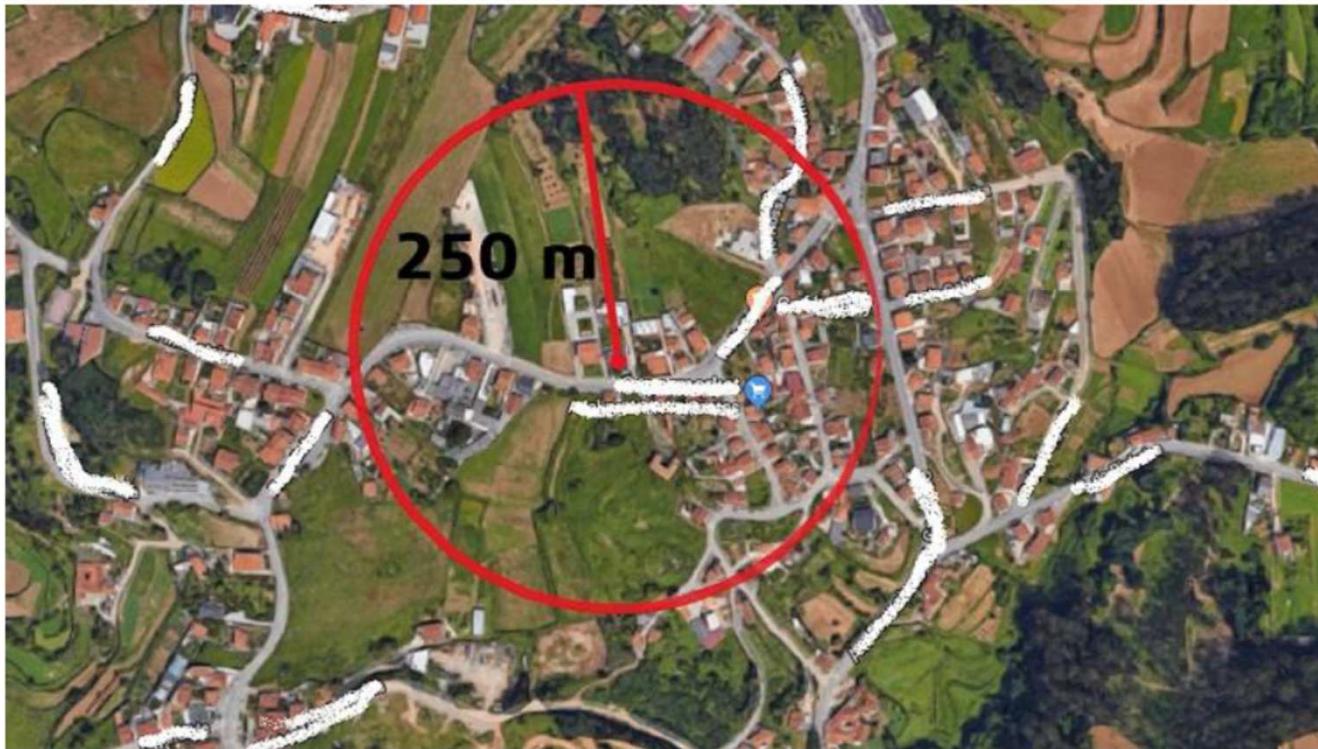
```
Serial.print(LoRaData);  
display.setCursor(0,10);  
display.print(LoRaData);  
display.display();
```

Finally, the next few lines of code print the RSSI of the received packet in dB.

```
int rssi = LoRa.packetRssi();  
Serial.print(" with RSSI ");  
Serial.println(rssi);  
display.setCursor(0,20);  
display.print("RSSI: ");  
display.setCursor(30,20);  
display.print(rssi);  
display.display();
```

## TESTING THE DISTANCE

With this setup we get a stable communication between the two ESPs up to 250 meters. This will greatly vary depending if you are in an urban or rural area, if there are a lot of buildings in between, etc.



Additionally, note that we're using the library's default definitions for LoRa, you may get a wider communication range by changing some settings. We'll not explore this topic in this Unit, but there's a [discussion on github on how to configure the LoRa library for better range](#).  
**Feel free to take a look.**

# TESTING THE DISTANCE AND ENERGY



saedelman commented on 13 Feb 2018

...

@MauriPastorini - the answers provided are not quite complete. Please download and play with the Semtech "Lora Calculator Tool" that you can download from their website (Windows only). It will clearly show you how the spreading factor (SF), Bandwidth (BW) and coding rate (CR) affect time-on-air (affects power consumption) and receiver sensitivity (affects distance). If you don't care about time on air/power consumption, set the bandwidth to a minimum and SF to maximum. You'll get an incredible link budgets with line of sight communication of 100km or more. Check out Andreas Spiess' Youtube videos on Lora for a test he did over very large distances. The antenna had very little impact.



3



saedelman commented on 13 Feb 2018

...

@MauriPastorini - One additional detail that is often overlooked - the lower the bandwidth, the more stable of a clock you need driving the Semtech chipset. For example, a number of Hope RF modules use a crystal oscillator with 10ppm drift, which means you cannot select bandwidths of less than 41.7 kHz (31.25 kHz may work) or you will not be able to demodulate any packets. Note that operation in high G environments will affect clock drift as well. In general, if you are designing strictly based on link budget (receiver sensitivity), choose the higher bandwidths (more immune to clock drift). For example, SF 10, BW 500 kHz and CR 4/5 yields a receiver sensitivity of -126 dBm with an on-air time of approx. 100ms for a 32-byte packet (6 byte preamble) but SF 9, BW 250 kHz and CR 4/5 also yields a receiver sensitivity of -126dBm with a similar on-air time (around 109ms). The latter allows for a maximum crystal drift of 72ppm whereas the 500kHz bandwidth will allow 144ppm drift. Therefore, for the same link budget, I would choose the wider bandwidth (500kHz) and and corresponding larger spreading factor (10) (less prone to interference than SF 9). Hope this helps.



2

The following table is based on midband frequency of 915MHz (North America - you'll get better results when using 868MHz) and sender and receiver isotropic antenna gains of 1 (e.g. a whip antenna without any gain)

## Distance - SF/BW

**10km - 7/500**

**20km - 8/500**

**30km - 9/500**

**40km - 10/500**

**60km - 11/500**

**80km - 12/500**

**100km - 11/250**

**125km - 12/250**

<https://www.loratools.nl/#/airtime>

<https://sx1272-lora-calculator.software.informer.com/download/>

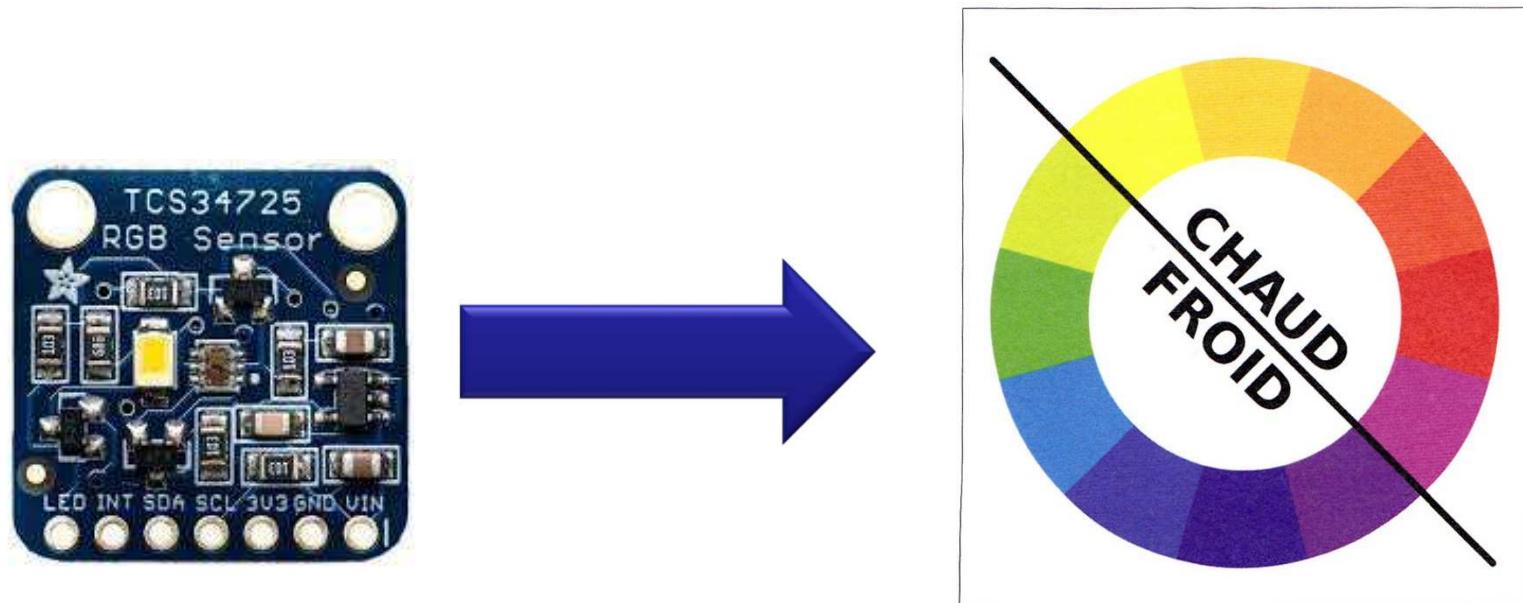
<https://dramco.be/tools/lora-calculator/>

# ARDUINO ET INTELLIGENCE ARTIFICIELLE

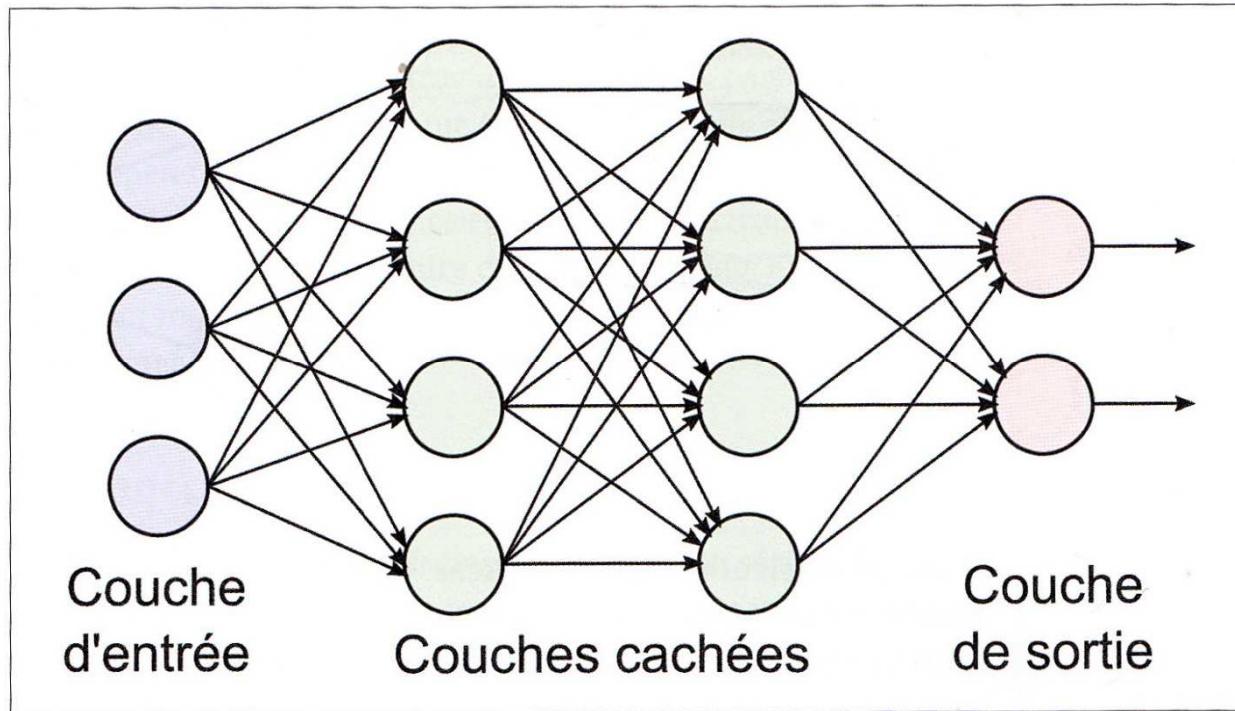


## APPLICATION INTELLIGENCE ARTIFICIELLE

Entrainer un réseau de neurone sur la base d'une correspondance entre jeu de valeurs R, G, B et le caractère chaud ou froid de la couleur issue du mélange



# APPLICATION INTELLIGENCE ARTIFICIELLE

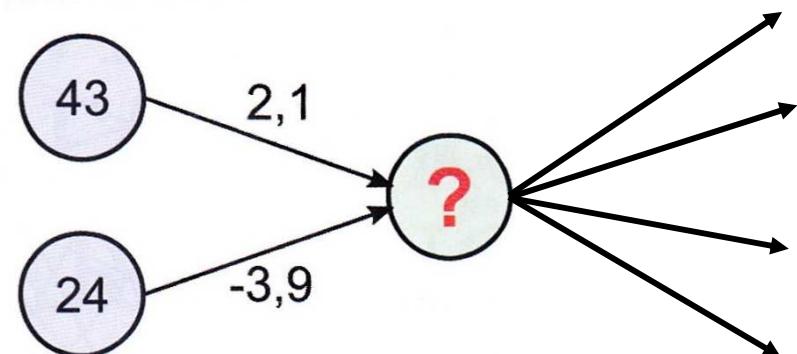


Couche  
d'entrée

Couches cachées

Couche  
de sortie

Réseau de neurones: perceptron multicouche  
organisés en couches interconnectées

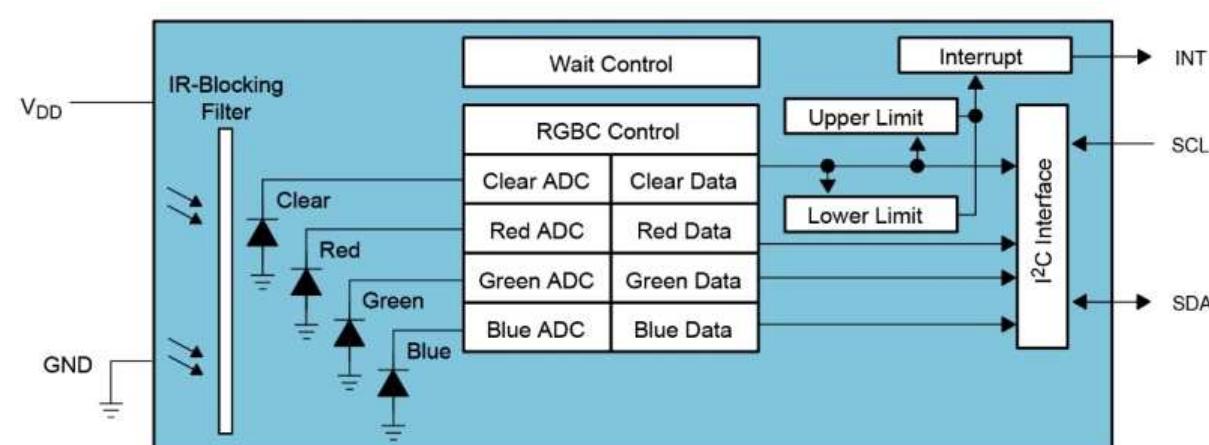
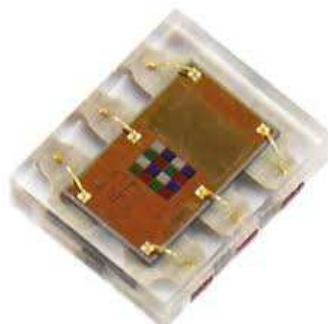


L'information circule au travers du réseau !

# LE CAPTEUR R, G ,B : TCS34725

## Description

The TCS34725 RGB Color Sensor Module incorporates high precision RGB and ambient light sensors, white LED light source and I2C interface.



Matrice de photodiodes

# TP11

# TP11 - LE SKETCH D'APPRENTISSAGE

```
1 #include <NeuralNetwork.h>
2
3 #define ITER 4000
4
5 unsigned int couches[] = {3, 12, 12, 1};
6 float *sorties;
7
8 const float entrees2[6][3] = {
9     {0.64, 0.18, 0.18}, // ROUGE
10    {0.26, 0.50, 0.24}, // VERT
11    {0.13, 0.35, 0.51}, // BLEU
12    {0.43, 0.41, 0.17}, // JAUNE
13    {0.30, 0.29, 0.41}, // VIOLET
14    {0.58, 0.26, 0.15}, // ORANGE
15 };
16
17 const float entrees[6][3] = {
18    {1.00, 0.00, 0.00}, // ROUGE
19    {0.00, 1.00, 0.00}, // VERT
20    {0.00, 0.00, 1.00}, // BLEU
21    {1.00, 1.00, 0.00}, // JAUNE
22    {0.00, 1.00, 1.00}, // CYAN
23    {1.00, 0.00, 1.00}, // MAGENTA (couleur limite)
24 };
25
26 const float sortiesAttendues[6][1] = {
27    {1}, // ROUGE
28    {0}, // VERT
29    {0}, // BLEU
30    {1}, // JAUNE
31    {0}, // CYAN
32    {1}, // MAGENTA
33 };
34
35 void setup() {
36     Serial.begin(115200);
37
38     NeuralNetwork NN(couches, 4);
39
40     for (int i = 0; i < ITER; i++) {
41         if(i%(ITER/100)==0) {
42             Serial.print("Entrainement ");
43             Serial.print(i/(ITER/100));
44             Serial.println("%");
45         }
46         for (int j = 0; j < 6; j++) {
47             NN.FeedForward(entrees[j]);
48             NN.BackProp(sortiesAttendues[j]);
49         }
50     }
51
52     for (int i = 0; i < 6; i++) {
53         sorties = NN.FeedForward(entrees[i]);
54         Serial.print(i);
55         Serial.print(": ");
56         Serial.println(sorties[0], 7);
57     }
58
59     Serial.println();
60     for (int i = 0; i < 6; i++) {
61         sorties = NN.FeedForward(entrees2[i]);
62         Serial.print(i);
63         Serial.print(": ");
64         Serial.println(sorties[0], 7);
65     }
66
67     NN.print();
68 }
69
70 void loop() {
71 }
```

# LE RESULTAT DU TEST D'APPRENTISSAGE

```
• 0: 0.9972061
• 1: 0.0058303
• 2: 0.0066308
• 3: 0.9937487
• 4: 0.0009738
• 5: 0.9930828

• 0: 0.9766245
• 1: 0.1751021
• 2: 0.0299375
• 3: 0.7535102
• 4: 0.2796261
• 5: 0.9583353

-----
• 3 12 | bias:1.00
1 W:-3.2345151 W:0.1309597 W:0.5248704
2 W:1.7743554 W:-1.1128615 W:-1.0229023
3 W:1.0439237 W:-0.9548942 W:-0.7500634
4 W:-0.2013842 W:-0.2736966 W:0.6675353
5 W:1.3244508 W:-0.5756721 W:0.0717406
6 W:1.2812261 W:-1.0434827 W:0.1039599
7 W:0.0799494 W:-0.1168651 W:0.4985597
8 W:-0.0033359 W:0.4396488 W:0.0837857
9 W:-1.7958540 W:0.0385646 W:-0.3189595
10 W:-0.7197753 W:-0.6555705 W:0.1462377
11 W:1.8314944 W:-0.9454978 W:-1.2856529
12 W:-2.2472562 W:0.3385596 W:0.6582292

-----
• 12 12 | bias:1.00
1 W:0.2504118 W:-0.0748784 W:0.1988482 W:0.3846305 W:-0.1931987 W:-0.2184220 W:0.4247591 W:0.8196946 W:0.7680702 W:-0.4695768 W:0.5541190 W:0.8695870
2 W:0.5016391 W:-0.3038916 W:0.0342278 W:0.0589705 W:0.1627688 W:0.1904305 W:-0.6407276 W:-0.1182652 W:-0.0600728 W:-0.1725170 W:-0.8090185 W:0.2743827
3 W:2.0405766 W:-1.4764641 W:-0.1413182 W:-0.3740782 W:-1.1956558 W:-0.6935209 W:-0.4548573 W:-0.0658056 W:0.2264159 W:0.6869005 W:-0.4933598 W:1.5914235
4 W:1.1155270 W:-0.8572950 W:-0.8008734 W:-0.6697721 W:-0.5013657 W:-0.1460065 W:-0.4328084 W:0.7530125 W:-0.0634269 W:0.3862528 W:-0.7250428 W:0.6111167
5 W:0.2705692 W:0.1760945 W:-0.1719677 W:-0.1129410 W:0.6726030 W:-0.9636604 W:-0.8179512 W:-0.0434211 W:0.4941928 W:0.0682246 W:-0.6848159 W:0.8007918
6 W:0.9544466 W:-0.6757089 W:-0.10301469 W:0.8338283 W:-0.7996262 W:-0.4537288 W:0.2324169 W:0.0687901 W:1.4843548 W:-0.6338199 W:-1.5104135 W:0.4744215
7 W:-1.7229041 W:-0.1798249 W:0.8212229 W:-0.3724416 W:0.0407116 W:0.9176830 W:0.1829736 W:0.5380781 W:-1.4646476 W:-0.7382750 W:1.0299470 W:-1.6410176
8 W:-1.0853009 W:1.3814204 W:0.6354837 W:0.0697226 W:0.4225459 W:0.1417617 W:-0.1362197 W:-0.6431421 W:-1.2836674 W:-0.4104369 W:-0.0603579 W:-1.0465948
9 W:1.0762183 W:-0.1442365 W:0.6064275 W:0.5933339 W:-0.9387536 W:-0.9469555 W:-0.0846194 W:-0.6753968 W:0.0977126 W:0.0045570 W:-1.2911614 W:0.6011035
10 W:-0.9536161 W:0.6358527 W:0.8789533 W:-0.1719329 W:0.8641534 W:-0.5096537 W:-0.8671501 W:-0.4959279 W:-0.1789059 W:-0.7331338 W:0.6689547 W:-0.4755179
11 W:0.6226391 W:0.1799965 W:-0.3603446 W:-0.9016899 W:0.4566050 W:-0.9944236 W:0.2678848 W:0.5542941 W:0.1134568 W:-0.2819874 W:-0.5971943 W:-0.1648310
12 W:1.5341858 W:-0.9963873 W:-0.8347433 W:-0.4640187 W:0.0578763 W:-0.9646052 W:0.5995245 W:0.2797670 W:-0.3659082 W:0.7878521 W:-0.6523962 W:-0.4423539

-----
• 12 1 | bias:1.07
1 W:0.8991649 W:-0.6896954 W:-3.0657675 W:-1.5418012 W:-0.5206811 W:-2.5220344 W:3.0342688 W:2.4633121 W:-1.7153653 W:1.5582633 W:-0.4350041 W:-1.4809799
```

Connexion des 3 premiers neurones  
à la première couche de 12

Connexion des 12 neurones de la couche 2  
Aux 12 neurones de la couche 3

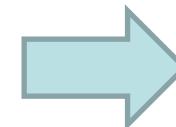
Connexion des 12 neurones de la couche 3  
Au seul neurone de sortie

# LE RESULTAT DU TEST D'APPRENTISSAGE

- 0: 0.9972061
- 1: 0.0058303
- 2: 0.0066308
- 3: 0.9937487
- 4: 0.0009738
- 5: 0.9930828
  
- 
- 3 12 | bias:1.00
- 1 W:-3.2345151 W:0.1309597 W:0.5248704
- 2 W:1.7743554 W:-1.1128615 W:-1.0229023
- 3 W:1.0439237 W:-0.9548942 W:-0.7500634
- 4 W:-0.2013842 W:-0.2736966 W:0.6675353
- 5 W:1.3244508 W:-0.5756721 W:0.0717406
- 6 W:1.2812261 W:-1.0434827 W:0.1039599
- 7 W:0.0799494 W:-0.1168651 W:0.4985597
- 8 W:-0.0033359 W:0.4396488 W:0.0837857
- 9 W:-1.7958540 W:0.0385646 W:-0.3189595
- 10 W:-0.7197753 W:-0.6555705 W:0.1462377
- 11 W:1.8314944 W:-0.9454978 W:-1.2856529
- 12 W:-2.2472562 W:0.3385596 W:0.6582292
  
- 
- 12 12 | bias:1.00
- 1 W:0.2504118 W:-0.0748784 W:0.1988482 W:0.3846305 W:-0.1931987 W:-0.2184220 W:0.4247591 W:0.8196946 W:-0.7680702 W:-0.4695768 W:0.5541190 W:0.8695870
- 2 W:0.5016391 W:-0.3038916 W:0.0342278 W:0.0589705 W:0.1627688 W:0.1904305 W:-0.6407276 W:-0.1182652 W:-0.0600728 W:-0.1725170 W:-0.8090185 W:0.2743827
- 3 W:2.0405766 W:-1.4764641 W:-0.1413182 W:-0.3740782 W:-1.1956558 W:-0.6935209 W:-0.4548573 W:-0.0658056 W:0.2264159 W:0.6869005 W:-0.4933598 W:1.5914235
- 4 W:1.1155270 W:-0.8572950 W:-0.8008734 W:-0.6697721 W:-0.5013657 W:0.1460065 W:-0.4328084 W:0.7530125 W:-0.0634269 W:0.3862528 W:-0.7250428 W:0.6111167
- 5 W:0.2705692 W:0.1760945 W:-0.1719677 W:-0.1129410 W:0.6726030 W:-0.9636604 W:-0.8179512 W:-0.0434211 W:0.4941928 W:0.0682246 W:-0.6848159 W:0.8007918
- 6 W:0.9544466 W:-0.6757089 W:-1.0301469 W:0.8338283 W:-0.7996262 W:-0.4537288 W:0.2324169 W:0.0687901 W:1.4843548 W:-0.6383199 W:-1.5104135 W:0.4744215
- 7 W:-1.7229041 W:-0.1798249 W:0.8212229 W:-0.3724416 W:0.0407116 W:0.9176830 W:0.1829736 W:0.5380781 W:-1.4646476 W:-0.7382750 W:1.0299470 W:-1.6410176
- 8 W:-1.0853009 W:1.3814204 W:0.6354837 W:0.0697226 W:0.4225459 W:0.1417617 W:-0.1362197 W:-0.6431421 W:-1.2836674 W:-0.4104369 W:-0.0603579 W:-1.0465948
- 9 W:1.0762183 W:-0.1442365 W:0.6064275 W:0.5933339 W:-0.9387536 W:-0.9469555 W:-0.0846194 W:-0.6753968 W:0.0977126 W:0.0045570 W:-1.2911614 W:0.6011035
- 10 W:-0.9536161 W:0.6358527 W:0.8789533 W:-0.1719329 W:0.8641534 W:-0.5096537 W:-0.8671501 W:-0.4959279 W:-0.1789059 W:-0.7331338 W:0.6689547 W:-0.4755179
- 11 W:0.6226391 W:0.1799965 W:-0.3603446 W:-0.9016899 W:0.4566050 W:-0.9944236 W:0.2678848 W:0.5542941 W:0.1134568 W:-0.2819874 W:-0.5971943 W:-0.1648310
- 12 W:1.5341858 W:-0.9963873 W:-0.8347433 W:-0.4640187 W:0.0578763 W:-0.9646052 W:0.5995245 W:0.2797670 W:-0.3659082 W:0.7878527 W:-0.6523962 W:-0.4423539
  
- 
- 12 1 | bias:1.07
- 1 W:0.8991649 W:-0.6896954 W:-3.0657675 W:-1.5418012 W:-0.5206811 W:-2.5220344 W:3.0342688 W:2.4633121 W:-1.7153653 W:1.5582633 W:-0.4350041 W:-1.4809799
- 

**SORTIE INTRODUITE**

0: 1 // ROUGE  
1: 0 // VERT  
2: 0 // BLEU  
3: 1 // JAUNE  
4: 0 // CYAN  
5: 1 // MAGENTA



**RESULTAT**

0: 0.9972061  
1: 0.0058303  
2: 0.0066308  
3: 0.9937487  
4: 0.0009738  
5: 0.9930828

**Connexion des 3 premiers neurones à la première couche de 12**

**Connexion des 12 neurones de la couche 2 Aux 12 neurones de la couche 3**

**Connexion des 12 neurones de la couche 3 Au seul neurone de sortie**

# LE SKETCH DE DETECTION

```

14 #include <NeuralNetwork.h>
15 #include <Wire.h>
16 #include <Adafruit_TCS34725.h>
17 #define LEDB 7
18 Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
19 const unsigned int couches[] = {9, 12, 12, 1};
20 float *sorties;
21 float entrees[3];
22
23 const float biais[] = {1.00, 1.00, 1.07};
24 //Ici on copie le resultat de l'entrainement du reseau de neurones
25 // 3*12 + 12*12 + 12*1
26 const float poids[] = {
27     -3.2345151, 0.1309587, 0.5248704,
28     1.7743554, -1.1128615, -1.0229023,
29     1.0439237, -0.9548942, -0.7500634,
30     -0.2013842, -0.2736966, 0.6675353,
31     1.3245508, -0.6756721, 0.0717406,
32     1.2812261, -1.0434827, 0.1038588,
33     0.0799494, -0.1168651, 0.4988597,
34     -0.0033389, 0.4396488, 0.0837857,
35     -1.7958540, 0.0395646, -0.3109595,
36     -0.7197753, -0.6555705, 0.1462377,
37     1.8319844, -0.9454978, -1.2856829,
38     -2.2472562, 0.3385859, 0.6582293,
39
40     0.2504118, -0.0748784, 0.1988482, 0.3846305, -0.1931987, -0.2184220, 0.4247581, 0.8196946, 0.7680702, -0.4695768, 0.5541190, 0.8685870,
41     0.5016391, -0.3038916, 0.0342378, 0.0589705, 0.1627688, 0.1904305, -0.6407376, -0.1182652, -0.0600728, -0.1725170, -0.8090185, 0.2743827,
42     0.2040766, -1.4764641, -0.1413182, -0.3740782, -1.1956558, -0.6935209, -0.4548573, -0.0658056, 0.2264159, 0.6869005, -0.4933598, 1.5914235,
43     1.1155270, -0.6572950, -0.8008734, -0.6697721, -0.5013657, -0.1460065, -0.4320084, 0.7530125, -0.0634269, 0.3862528, -0.7250428, 0.6111167,
44     0.2705692, 0.1760545, -0.1719677, -0.1129410, 0.6726030, -0.5636604, -0.8179512, -0.0434211, 0.5941928, 0.0682246, -0.6848159, 0.8007918,
45     0.9544666, -0.6757089, -1.0801469, 0.8338203, -0.7996262, -0.4537288, 0.2324169, 0.0687901, 1.4843548, -0.6338199, -1.5104135, 0.4744215,
46     -1.7229041, -0.1798249, 0.8212229, -0.3724416, 0.0407116, 0.9176830, 0.1529736, 0.5380781, -1.4646476, -0.7352750, 1.0299470, -1.6410176,
47     -1.0053009, 1.3914204, 0.6354837, 0.0697226, 0.4225459, 0.1417617, -0.1362197, -0.6431421, -1.2336674, -0.4104939, -0.0603379, -1.0465848,
48     1.0762183, -0.1442365, 0.6064275, 0.5933338, -0.9387536, -0.0846194, -0.6753960, 0.0977126, 0.0045570, -1.2911614, 0.6011035,
49     -0.9536161, 0.6358527, 0.8789533, -0.1719329, 0.8641534, -0.5086537, -0.8671501, -0.4959279, -0.1789059, -0.7331338, 0.6689547, -0.6755179,
50     0.6226391, 0.1799965, -0.3603446, -0.9016899, 0.4566050, -0.9944236, 0.2678848, 0.5542841, 0.1134568, -0.2819874, -0.5971943, -0.1648310,
51     1.5341858, -0.9963873, -0.0347433, -0.4640187, 0.0578763, -0.9646052, 0.5995245, 0.2797670, -0.3659082, 0.7878527, -0.6523962, -0.4423593,
52
53     0.8991649, -0.6896954, -3.0657675, -1.5418013, -0.5206811, -2.5220344, 3.0342688, 2.4633121, -1.7153653, 1.5582633, -0.4350041, -1.4809799
54 };

```

```

58 void setup() {
59     Serial.begin(115200);
60
61     pinMode(LEDB, OUTPUT);
62
63     if (!tcs.begin()) {
64         Serial.println("Erreur TCS34725 !");
65         while (1);
66     }
67 /*
68     for (int i = 0; i < 6; i++) {
69         sorties = NN.FeedForward(entrees[i]);
70         Serial.print(sorties[0]>0.5?"Chaud ":"Froid ");
71         Serial.print(sorties[0], 7);
72         Serial.println("");
73     }
74 */
75
76 void loop() {
77     uint16_t rouge, vert, bleu, tout;
78
79     digitalWrite(LEDB, HIGH);
80     delay(300);
81     tcs.getRawData(&rouge, &vert, &bleu, &tout);
82     digitalWrite(LEDB, LOW);
83
84     entrees[0]=(rouge*1.0)/(rouge+vert+bleu);
85     entrees[1]=(vert*1.0)/(rouge+vert+bleu);
86     entrees[2]=(bleu*1.0)/(rouge+vert+bleu);
87
88     sorties = NN.FeedForward(entrees);
89
90     Serial.print(rouge); Serial.print("/");
91     Serial.print(vert); Serial.print("/");
92     Serial.print(bleu); Serial.print(" ");
93     Serial.print(sorties[0]>0.5?"Chaud ":"Froid ");
94     Serial.print(sorties[0], 7);
95     Serial.println("");
96     delay(500);
97 }

```

- Simple MLP - NeuralNetwork Library For Microcontrollers
- Nothing "Import ant", just a simple library for implementing Neural-Networks(NNs) easily and effectively on any Arduino board and other microcontrollers.

<https://github.com/GiorgosXou/NeuralNetworks>

# LES FONCTIONS DE L'IA

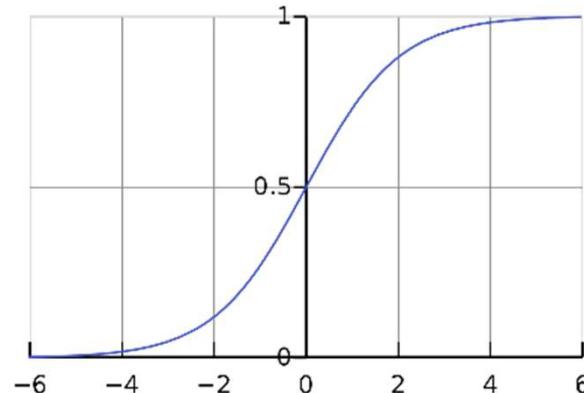
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) \underset{x=0}{\text{is undefined}} \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) <sup>[2]</sup>		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) <sup>[3]</sup>		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

# LES FONCTIONS DE L'IA

En [mathématiques](#), la fonction sigmoïde (dite aussi *courbe en S*) est définie par :

$$f(x) = \frac{1}{1 + e^{-x}}$$

pour tout  $x$  réel



The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to **predict the probability** as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

⇒ Fait activer le neurone rapidement dès qu'elle est supérieure à 0.5 ou inversement le désactiver rapidement si elle est inférieure à 0.5

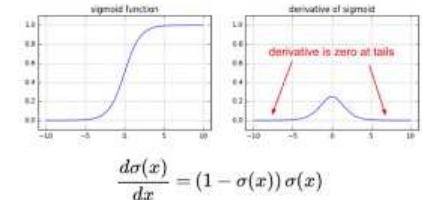
Elle représente la [fonction de répartition](#) de la [loi logistique](#).

Elle est souvent utilisée dans les [réseaux de neurones](#) parce qu'elle est [dérivable](#), ce qui est une contrainte pour l'algorithme de [rétropropagation](#) de Werbos.

La forme de la [dérivée](#) de sa [fonction inverse](#) est extrêmement simple et facile à calculer, ce qui améliore les performances des algorithmes.

La courbe sigmoïde générée par [transformation affine](#) une partie des [courbes logistiques](#) et en est donc un représentant privilégié.

Optimization is Hard: Vanishing Gradients



⇒ Ne trouvez-vous pas une similitude avec la [fonction de Fermi-Dirac](#) utilisée en physique du solide ;-) ?

$$f_\lambda(x) = f(\lambda x) = \frac{1}{1 + e^{-\lambda x}}$$

[https://fr.wikipedia.org/wiki/Sigmo%C3%AFde\\_\(math%C3%A9matiques\)](https://fr.wikipedia.org/wiki/Sigmo%C3%AFde_(math%C3%A9matiques))

# COULEURS DE TEST

ROUGE



JAUNE



VERT



CYAN

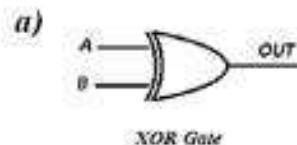


BLEU



MAGENTA

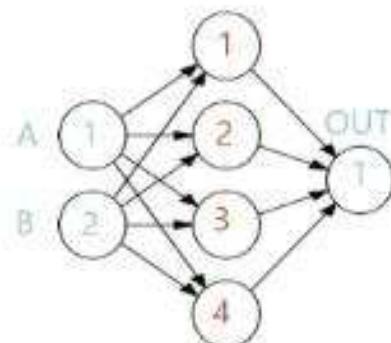




b)

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

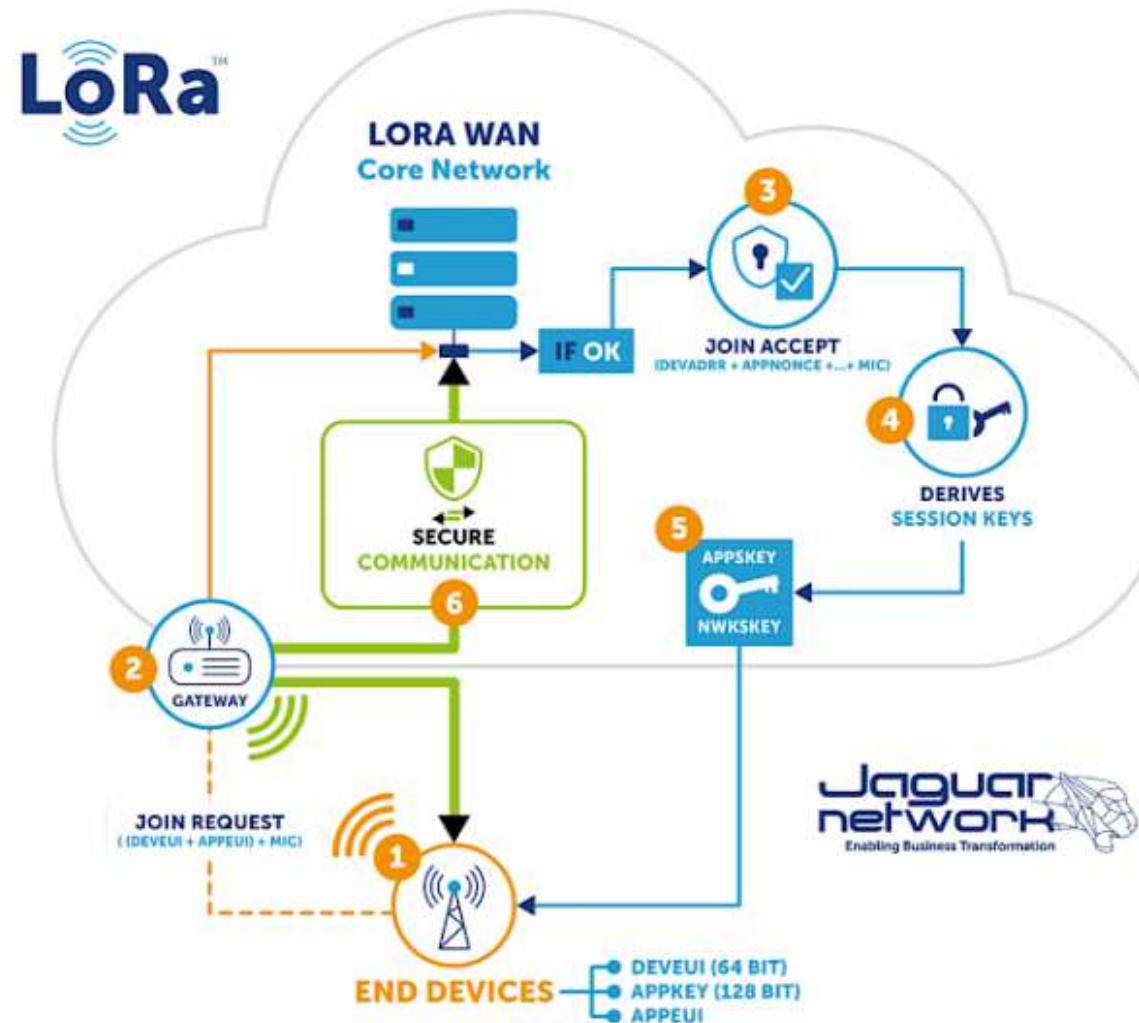
XOR accuracy table



```
1 #define NumberOfArgs ((unsigned int) (argc - sizeof (argc) / sizeof (arg [0]))) //calculates the amount of layers (in this case 3)
2
3 #include <NeuralNetwork.h> 241
4
5 const unsigned int layers[] = {2,4,1}; //3 layers (1st)layer with 2 input neurons (2nd)layer with 4 hidden neurons and (3rd)layer with 1 output neuron
6 float *outputs; // 3rd layer's outputs (in this case output)
7
8 //Default Inputs
9 const float inputs[4][2] =
10 { {0, 0}, //0
11 {0, 1}, //1
12 {1, 0}, //1
13 {1, 1} //0
14 }; //A B OUT
15
16 const float expectedOutput[4][1] = {{0},{1},{1},{0}}; // values that we were expecting to get from the last/output layer of Neural-network
17
18 void setup()
19 {
20
21   Serial.begin(9600);
22
23   NeuralNetwork NN(layers,NumberOfLayers); // Creating a NeuralNetwork with default learning-rates
24
25   //Trains the NeuralNetwork for 3000 epochs = Training loops
26   for(int i=0; i < 3000; i++) // epochs = Training loops
27   {
28     for (int j = 0; j < NumberOfInputs; j++)
29     {
30       NN.FeedForward(inputs[j]); // Feeds-Forward the inputs to the first layer of the NN and Gets the output,
31       NN.BackProp(expectedOutput[j]); // Tells to the NN if the output was right/the-expectedOutput and then, teaches it.
32     }
33   }
34
35   //Goes through all inputs
36   for (int i = 0; i < NumberOfInputs; i++)
37   {
38     outputs = NN.FeedForward(inputs[i]); // Feeds-Forward the inputs[i] to the first layer of the NN and Gets the output
39     serial.print(outputs[0], 7); // prints the first 7 digits after the comma.
40   }
41
42   NN.print(); // prints the weights and biases of each layer
43
44 }
```

# TP12 – The Things Network (TTN) + Node-Red

# OTAA vs ABP ?



ACTIVATION D'UN EQUIPEMENT PAR OTA

# « LoRaWAN In a nutshell » : principes de jonction au réseau

Deux modes de jonction existent qui se distinguent par leur sécurité et leur facilité de mise en œuvre :

1. ABP : Activation By Personalization
2. OTAA : Over The Air Activation

## L'ABP:

•**Avantage** : Raccordement au réseau simplifié ; l'objet est rapidement opérationnel.

•**Inconvénient** : Les clés de chiffrement permettant la communication avec le réseau, sont préconfigurées dans l'objet : sécurité affaiblie.

**ATTENTION** : Cette stratégie de simplicité dans la procédure de jonction se fait au détriment de la sécurité. S'il y a intrusion physique dans l'objet, le vol des clés est possible et peut conduire à l'usurpation de l'identité de l'objet, entraînant une corruption des données collectées

**Exemple** : Un compteur d'eau pourrait envoyer une indication de consommation d'eau erronée pouvant entraîner des nuisances (inondation, surfacturation...).

Afin d'établir la jonction au réseau et d'identifier l'objet, il est nécessaire de connaître plusieurs informations :

•**AppEUI** : C'est un identifiant unique d'application qui permet de regrouper les objets. Cette adresse, sur 64 bits, permet de classer les périphériques par application. **Ce paramètre est modifiable**.

•**DevEUI** : c'est un identifiant qui rend unique chaque objet, **programmé en usine**. **Ce paramètre n'est théoriquement pas modifiable**.

•**AppKey** : Il s'agit d'un secret partagé entre le périphérique et le réseau, utilisé pour dériver les clefs de session. **Ce paramètre peut être modifié**.

**Les clefs de chiffrements sont préprogrammées dans le périphérique :**

•**DevAddr** : il s'agit d'une adresse logique pour identifier l'objet dans le réseau.

•**NetSKey (Network Session Key)** : Clé de chiffrement entre l'objet et l'opérateur utilisée pour les transmissions et valider l'intégrité des messages.

•**AppSKey (Application Session Key)** : Clé de chiffrement entre l'objet et l'utilisateur (via l'application) utilisée pour les transmissions et valider l'intégrité des messages.

## L'OTAA:

•**Avantage** : Le réseau génère et envoie les clés de chiffrement ; la sécurité est renforcée.

**C'est la méthode la plus utilisée dans le monde de l'IoT/LoRaWAN, car la plus sécurisée.**

•**Inconvénient** : L'objet doit implémenter ce mécanisme de jonction ce qui introduit une complexité supplémentaire.

Afin d'établir la jonction au réseau et d'identifier l'objet, il est nécessaire de connaître plusieurs informations.

•**AppEUI** : C'est un identifiant unique d'application qui permet de regrouper les objets. Cette adresse, sur 64 bits, permet de classer les périphériques par application. **Ce paramètre est modifiable**.

•**DevEUI** : c'est un identifiant qui rend unique chaque objet, programmé en usine. **Ce paramètre n'est théoriquement pas modifiable**.

•**AppKey** : Il s'agit d'un secret partagé entre le périphérique et le réseau, utilisé pour dériver les clefs de session. **Ce paramètre peut être modifié**.

### Zoom sur l'OTAA

Le composant logiciel en charge d'établir le raccordement avec les objets et d'animer le cœur de réseau, est le **"Network Server"**. Lors d'une jonction de type OTAA, si l'objet est autorisé à rejoindre le réseau, celui-ci va échanger des clés de chiffrement propres à cette session avec le cœur de réseau.

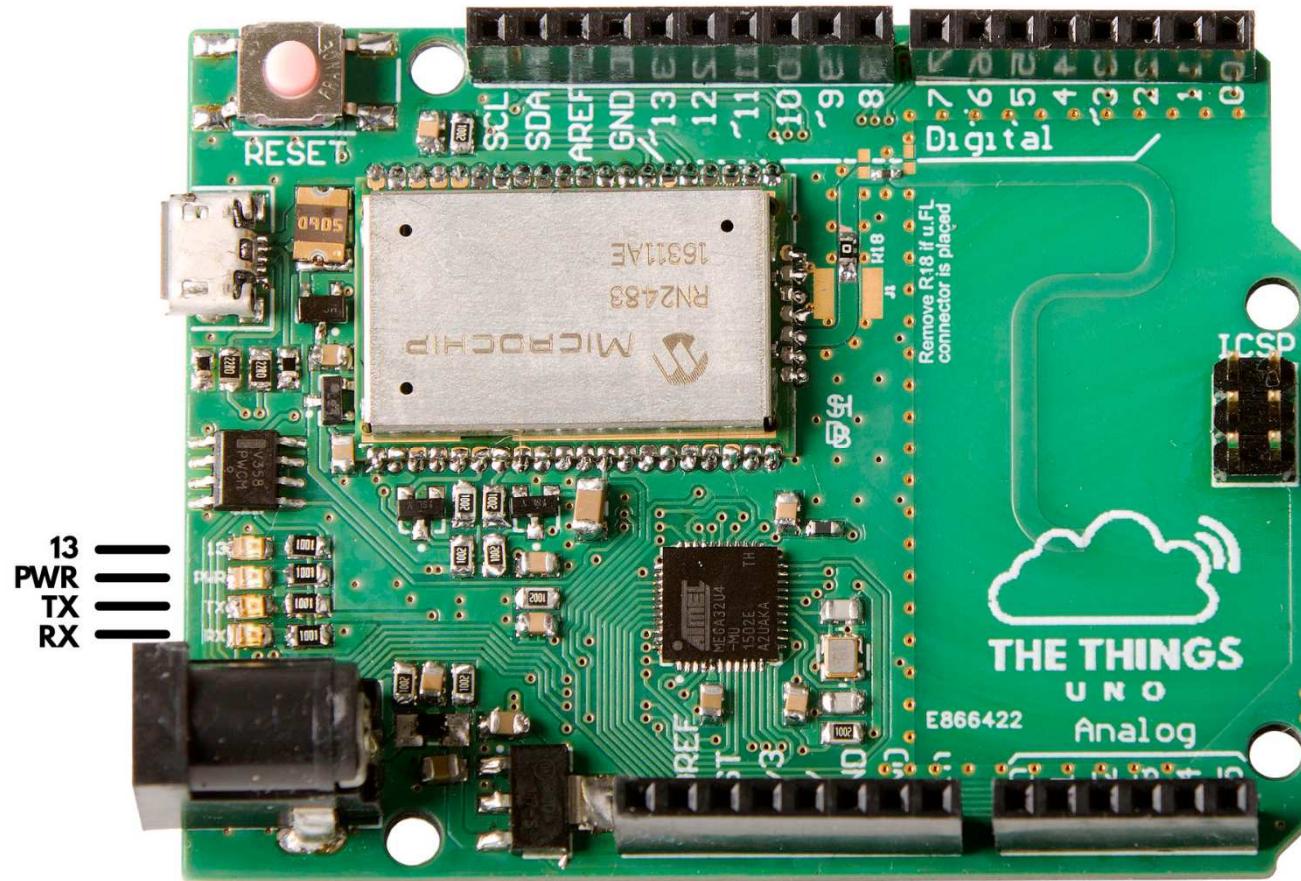
**Le Network Server va alors attribuer des informations propres à la session et les communiquer au périphérique :**

•**DevAddr** : adresse logique (équivalent d'une adresse IP) qui sera utilisé pour toutes les communications ultérieures.

•**NetSKey (Network Session Key)** : Clé de chiffrement entre l'objet et l'opérateur utilisée pour les transmissions et valider l'intégrité des messages.

•**AppSKey (Application Session Key)** : Clé de chiffrement entre l'objet et l'utilisateur (via l'application) utilisée pour les transmissions et valider l'intégrité des messages.

# The THINGS UNO



<https://www.thethingsnetwork.org/docs/devices/uno/>

<https://www.microchip.com/design-centers/wireless-connectivity/low-power-wide-area-networks/lora-technology/sam-r34-r35>

Troubles: <https://sighmon.com/says/connecting-to-the-things-network/>

# TUTORIEL THE THINGS UNO

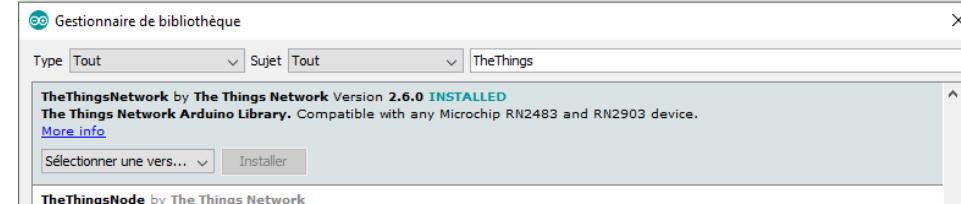
1 – Dans l'IDE Arduino: installer la librairie TTN

2 – Charger le sketch DeviceInfo pour récupérer les identifiants EUI du RN2483 (Arduino Leonardo + PORT COM):  
(remplacer freqPlan REPLACE\_ME par la freqPlan de notre lieu (eg EU ou US))

3 – Ouvrir le port Série (9600 baud): => Device Information

EUI: 0004A30B001BE16C  
Battery: 3283  
AppEUI: 0000000000000000  
**DevEUI: 0004A30B001BE16C**  
Data Rate: 5  
RX Delay 1: 1000  
RX Delay 2: 2000

Use the EUI to register the device for OTAA



4 – Recopier les DevEUI: (eg **DevEUI: 0004A30B001BE16C**)

5 – Sur The Things Networks:

Device registration <https://www.thethingsnetwork.org/docs/devices/registration.html>

Console <https://console.thethingsnetwork.org/>

Créer une application: eg thingsuno\_test

=> Génère l'**APPEUI: 70B3D57ED0036E7D**

Dans l'application, cliquer sur DEVICE: [https://console.thethingsnetwork.org/applications/thingsuno\\_test/devices](https://console.thethingsnetwork.org/applications/thingsuno_test/devices)

Activer le device sur TheThingsNetwork: soit par la méthode OTA soit par la méthode ABP

*Attention, pour OTA, il faut au moins le firmware V1.4 sur le RN2483 pour ne pas avoir l'erreur "No Channel" (lié au duty cycle)*

6 – Choisir le sketch en fonction d'ABP ou OTAP

6.1 rendre l'exemple SendABP.ino:

Depuis la console TTN, récupérer les constantes du device en ABP :

```
const char *devAddr = "26013946";
const char *nwkSKey = "E400FBFA5C76C7836C2380F27ADB77FF";
const char *appSKey = "7B0218AE8B0ACBCBE1414FE19F5ABD61";
```

Et le coller dans l'exemple de l'IDE Arduino:

Vérifier que l'on reçoit les données: 1 – sur TTN ou 2 – sur le port série

6.2 Prendre l'exemple SendOTAA.ino:

Depuis la console TTN, recopier

```
const char *appEui = "70B3D57ED0036E7D";
const char *appKey = "35ED785B4848E6F1A8F55C697F029706";
```

Et le coller dans l'exemple de l'IDE Arduino et uploader le sketch:

Vérifier que l'on reçoit les données: 1 – sur TTN ou 2 – sur le port série

# TUTORIEL SENDING DATA

## 6. Test de data sensor: SendABp\_DataSensor.ino, ajouter:

```
uint32_t humidity = 7625;//à diviser par 100
uint32_t temperature = 3215;//à diviser par 100
byte payload[4];
payload[0]=highByte(humidity);
payload[1]=lowByte(humidity);
payload[2]=highByte(temperature);
payload[3]=lowByte(temperature);
// Send it off
ttn.sendBytes(payload, sizeof(payload));
```

## 7. Sur TTN: Payload functions

Dans Application/Payload Formats

Decoder function => turn bytes into json objects !

*Initial decoder:*

```
function Decoder(bytes, port) {
  // Here can decode the payload into json.
  // bytes is of type Buffer.
  var length = bytes.length;
  var my_payload = "";
  for (i = 0; i < length; i++) {
    //my_payload += String.fromCharCode(bytes[i]);
    my_payload += parseInt(bytes[i]);}
```

*Custom decoder:*

```
function Decoder(bytes, port) {
  // Decoder
  // bytes is of type Buffer.
  var humidity = (bytes[0] << 8) | bytes[1];
  var temperature = (bytes[2] << 8) | bytes[3];
  return{
    humidity: humidity,
    temperature: temperature
  }
```

A tester, dans le champ Payload:

Eg 1D C9 0C 8F + SAVE Payload Format

<https://www.thethingsnetwork.org/docs/devices/bytes.html>

The screenshot shows the TTN Payload Formats configuration page. At the top, there are tabs for Overview, Devices, Payload Formats (which is selected), Integrations, Data, and Settings. Below the tabs, the title "PAYLOAD FORMATS" is displayed. A sub-section titled "Payload Format" contains the text "The payload format sent by your devices" and a dropdown menu set to "Custom". Under the "decoder" tab, a code editor displays the following JavaScript function:

```
1 function Decoder(bytes, port) {
2   // Decoder
3   // bytes is of type Buffer. dans un format *100 => il faudra diviser /100 dans NodeRed
4   var humidity = (bytes[0] << 8) | bytes[1];
5   var temperature = (bytes[2] << 8) | bytes[3];
6
7   return{
8     humidity: humidity,
9     temperature: temperature
10   }
11
12 }
13
```

Below the code editor, a message says "decoder has no changes". To the right of the code editor is a "Test" button. Further down, under the "Payload" section, there is a text input field containing a JSON object:

```
{
  "humidity": 7625,
  "temperature": 3215
}
```

The screenshot shows the TTN Application Data interface. At the top, there are tabs for Overview, Devices, Payload Formats, Integrations, Data (selected), and Settings. Below the tabs, the title "APPLICATION DATA" is displayed. A table lists received data frames with columns for time, counter, port, devid, payload, humidity, and temperature. Two rows are highlighted with a green border:

time	counter	port	devid	payload	humidity	temperature
▲ 10:00:36	34	1	myunodemo	1DC90C8F	7625	3215
▲ 09:59:29	31	1	myunodemo	1DC90C8F	7625	3215
▲ 09:57:59	27	1	myunodemo	1DC90C8F	7625	3215

# TEST WITH RN2483

# TEST AVEC ARDUINO UNO + RN2483

1 – Dans l’IDE Arduino:

- Installer la librairie TTN
- Installer la librairie RN2483
- Charger Demo\_Arduino\_Unc\_RN2483
- Configurer les pins du RN2483

```
#define TX 10
#define RX 11
#define RST 12
SoftwareSerial mySerial(TX, RX);
```

Dans le port Série: récupérer le DEV EUI: **0004A30B00201396**

2 – Recopier les DevEUI: (eg DevEUI: **0004A30B001BE16C**)

3 – Sur The Things Networks:

Device registration <https://www.thethingsnetwork.org/docs/devices/registration>

Console <https://console.thethingsnetwork.org/>

Créer ou choisir une application: eg thingsuno\_test

Dans l’application, cliquer sur DEVICE: [https://console.thethingsnetwork.org/applications/thingsuno\\_test/devices](https://console.thethingsnetwork.org/applications/thingsuno_test/devices)

Activer le device sur TheThingsNetwork: soit par la méthode OTA soit par la méthode ABP

*Attention, pour OTAA, il faut avoir au moins le firmware V1.4 sur le RN2483*

*pour ne pas avoir l’erreur “No Channel” (lié au duty cycle)*



3.1 Depuis la console TTN, récupérer les constantes du device en ABP :

```
const char *devAddr = "26013946";
const char *nwkSKey = "E400FBFA5C76C7836C2380F27ADB77FF";
const char *appSKey = "7B0218AE8B0ACBCBE1414FE19F5ABD61";
```

Et le coller dans l’exemple de l’IDE Arduino + Vérifier que l’on reçoit les données: 1 – sur TTN ou 2 – sur le port série

4. Insérer les lignes suivantes dans le sketch:

```
61 float humidity_float = 76.25;
62 float temperature_float = 32.15;
63 uint32_t humidity = humidity_float * 100; //à diviser par 100
64 uint32_t temperature = temperature_float * 100; //à diviser par 100
```

```
74 //FORMATAGE POUR TTN
75 byte payload[4];
76 payload[0]=highByte(humidity);
77 payload[1]=lowByte(humidity);
78 payload[2]=highByte(temperature);
79 payload[3]=lowByte(temperature);
85 //myLora.tx(payload);
86 myLora.txBytes(payload, sizeof(payload));
87
```

# COMMANDE RN2483

## RN2483 LoRa™ TECHNOLOGY MODULE COMMAND REFERENCE USER'S GUIDE

### 2.4.2 mac tx <type> <portno> <data>

<type>: string representing the uplink payload type, either cnf or uncnf (cnf – confirmed, uncnf – unconfirmed)

<portno>: decimal number representing the port number, from 1 to 223

<data>: hexadecimal value. The length of <data> bytes capable of being transmitted are dependent upon the set data rate (please refer to the LoRaWAN™ Specification for further details).

Response: this command may reply with two responses. The first response will be received immediately after entering the command. In case the command is valid (ok reply received), a second reply will be received after the end of the uplink transmission. Please refer to the LoRaWAN™ Specification for further details.

Response after entering the command:

- ok – if parameters and configurations are valid and the packet was forwarded to the radio transceiver for transmission
- invalid\_param – if parameters (<type> <portno> <data>) are not valid
- not\_joined – if the network is not joined
- no\_free\_ch – if all channels are busy
- silent – if the module is in a Silent Immediately state
- frame\_counter\_err\_rejoin\_needed – if the frame counter rolled over
- busy – if MAC state is not in an Idle state
- mac\_paused – if MAC was paused and not resumed back
- invalid\_data\_len if application payload length is greater than the maximum application payload length corresponding to the current data rate

Response after the uplink transmission:

- mac\_tx\_ok if uplink transmission was successful and no downlink data was received back from the server;
- mac\_rx <portno> <data> if transmission was successful, <portno>: port number, from 1 to 223; <data>: hexadecimal value that was received from the server;
- mac\_err if transmission was unsuccessful, ACK not received back from the server
- invalid\_data\_len if application payload length is greater than the maximum application payload length corresponding to the current data rate

A confirmed message will expect an acknowledgment from the server; otherwise, the message will be retransmitted by the number indicated by the command mac set retrx <value>, whereas an unconfirmed message will not expect any acknowledgment back from the server. Please refer to the LoRaWAN™ Specification for further details.

```
TX_RETURN_TYPE rn2xx3::tx(String data)
{
    return txUncnf(data); //we are unsure which mode we're in. Better not to wait for acks.
}

TX_RETURN_TYPE rn2xx3::txBytes(const byte* data, uint8_t size)
{
    char msgBuffer[size*2 + 1];
    char buffer[3];
    for (unsigned i=0; i<size; i++)
    {
        sprintf(buffer, "%02X", data[i]);
        memcpy(&msgBuffer[i*2], &buffer, sizeof(buffer));
    }
    String dataToTx(msgBuffer);
    return txCommand("mac tx uncnf 1 ", dataToTx, false);
}

TX_RETURN_TYPE rn2xx3::txCnf(String data)
{
    return txCommand("mac tx cnf 1 ", data, true);
}

TX_RETURN_TYPE rn2xx3::txUncnf(String data)
{
    return txCommand("mac tx uncnf 1 ", data, true);
}

TX_RETURN_TYPE rn2xx3::txCommand(String command, String data, bool shouldEncode)
{
    bool send_success = false;
    uint8_t busy_count = 0;
    uint8_t retry_count = 0;

    //clear serial buffer
    while(_serial.available())
        _serial.read();

    while(!send_success)
    {
        //retransmit a maximum of 10 times
        retry_count++;
        if(retry_count>10)
        {
            return TX_FAIL;
        }

        _serial.print(command);
        if(shouldEncode)
        {
            sendEncoded(data);
        }
        else
        {
            _serial.write(data);
        }
    }
}
```

# TEST AVEC ARDUINO UNO + RN2483



Demo\_Arduino\_Undo\_RN2483 | Arduino 1.8.5  
Fichier Édition Croquis Outils Aide

```
#include <rn2xx3.h>
#include <SoftwareSerial.h>

#define DEBUG
#define LEDPIN 13
//Sur le shield => // TX = 10 // RX = 11 // RST = 12
#define TX 10
#define RX 11
#define RST 12
SoftwareSerial mySerial(TX, RX); // RX, TX

// Copy the following lines from TTN Console -> Devices -> Overview tab -> "EXAMPLE CODE"
//CONFIGURATION EN VERSION ABP
const char *devAddr = "260138C9";
const char *nwkSKey = "C4A63E95D4C336221CDD41BACF411F0C";
const char *appSKey = "6EFB6054989BFE8F6042A2A392DFC5F3";

rn2xx3 myLora(mySerial);

void led_on() {digitalWrite(LEDPIN, HIGH);}
void led_off() {digitalWrite(LEDPIN, LOW);}

void setup() {
  pinMode(LEDPIN, OUTPUT);
  led_on();

  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  mySerial.begin(9600);
  Serial.println("Startup");

  // Reset rn2483
  pinMode(RST, OUTPUT);
  digitalWrite(RST, HIGH);
  digitalWrite(RST, LOW);
  delay(500);
  digitalWrite(RST, HIGH);

  // Initialise the rn2483 module
  myLora.autobaud();

  Serial.println("When using OTAA, register this DevEUI:");
  Serial.println(myLora.hweui());
  Serial.print("RN2483 version number: ");
}

Enregistrement terminé.
```

```
43   Serial.println(myLora.hweui());
44   Serial.print("RN2483 version number: ");
45   Serial.println(myLora.sysver());
46
47   myLora.initABP(devAddr, appSKey, nwkSKey);
48
49   led_off();
50   delay(2000);
51 }
52
53 void loop() {
54   led_on();
55   #ifdef DEBUG
56     Serial.println("TXing");
57   #endif
58
59   //myLora.txUncnf("X");
60   //myLora.tx("hello jere!"); //one byte, blocking function
61   float humidity_float = 76.25;
62   float temperature_float = 32.15;
63   uint32_t humidity = humidity_float *100 ;//à diviser par 100
64   uint32_t temperature = temperature_float * 100;//à diviser par 100
65   #ifdef DEBUG
66     Serial.println("Humidity Float: " + String (humidity_float));
67     Serial.println("Temperature Float: " + String (temperature_float));
68     Serial.println("Humidity: " + String (humidity));
69     Serial.println("Temperature: " + String (temperature));
70   #endif
71
72
73   //FORMATAGE POUR TTN
74   byte payload[4];
75   payload[0]=highByte(humidity);
76   payload[1]=lowByte(humidity);
77   payload[2]=highByte(temperature);
78   payload[3]=lowByte(temperature);
79
80
81   #ifdef DEBUG
82     for (int i=0; i < sizeof(payload); i++) {Serial.println(payload[i]);}
83   #endif
84
85   //myLora.tx(payload);
86   myLora.txBytes(payload, sizeof(payload));
87
88   led_off();
89   delay(20000);
90 }
91 }
```

# TUTORIEL DOWNLINK

Test de Downlink + data sensor: SendABp\_DataSensor\_Downlink, ajouter:

```
#define LED_PIN 13

void led_on() { digitalWrite(LED_PIN, HIGH); }
void led_off() { digitalWrite(LED_PIN, LOW); }

void message(const uint8_t *payload, size_t size, port_t port)
{
    debugSerial.println("-- MESSAGE");
    debugSerial.print("Received " + String(size) + " bytes on port " + String(port) + ":");
    for (int i = 0; i < size; i++)
    {
        debugSerial.print(" " + String(payload[i]));
    }
    debugSerial.println();
    if (payload[0] == 1){led_on();} else {led_off();}
}
```

Dans le void Setup():

```
ttn.onMessage(message); //For downlink
```

## TEST WITH RFM95

# TEST AVEC ARDUINO UNO + RFM95

1 – Dans l’IDE Arduino:

- Installer la librairie TTN
- Installer la librairie LMIC
- Installer la librairie Low Power Master
- Configurer les pins du RFM95 par exemple comme ci-dessous

```
78 // Pin mapping
79 const lmic_pinmap lmic_pins = {
80     .nss = 10,
81     .rxtx = LMIC_UNUSED_PIN,
82     .rst = 9,
83     .dio = {2, 6, 7},
84 };
```

2 – Recopier les DevEUI: (eg DevEUI: 0004A30B001BE16C)

3 – Sur The Things Networks:

Dans l’application, créer un DEVICE: [https://console.thethingsnetwork.org/applications/thingsuno\\_test/devices](https://console.thethingsnetwork.org/applications/thingsuno_test/devices)

Activer le device sur TheThingsNetwork: méthode ABP

3.1 Depuis la console TTN, récupérer les constantes du device en ABP :

```
const char *devAddr = "26013946";
const char *nwkSKey = "E400FBFA5C76C7836C2380F27ADB77FF";
const char *appSKey = "7B0218AE8B0ACBCBE1414FE19F5ABD61";
```

Et le coller dans l’exemple de l’IDE Arduino + Vérifier que l’on reçoit les données: 1 – sur TTN ou 2 – sur le port série

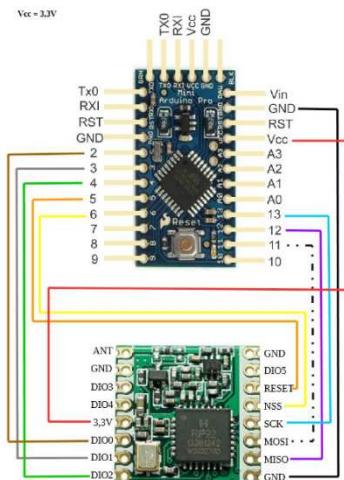
4. Charger le sketch Arduino\_LMIC.ino

Ou faire un autre exemple

```
66
67 // Pin mapping
68 const lmic_pinmap lmic_pins = {
69     .nss = 6,
70     .rxtx = LMIC_UNUSED_PIN,
71     .rst = 5,
72     .dio = {2, 3, 4},
73 };
74
```

<https://github.com/matthijskooijman/arduino-lmic>

<https://www.thethingsnetwork.org/labs/story/build-the-cheapest-possible-node-yourself>

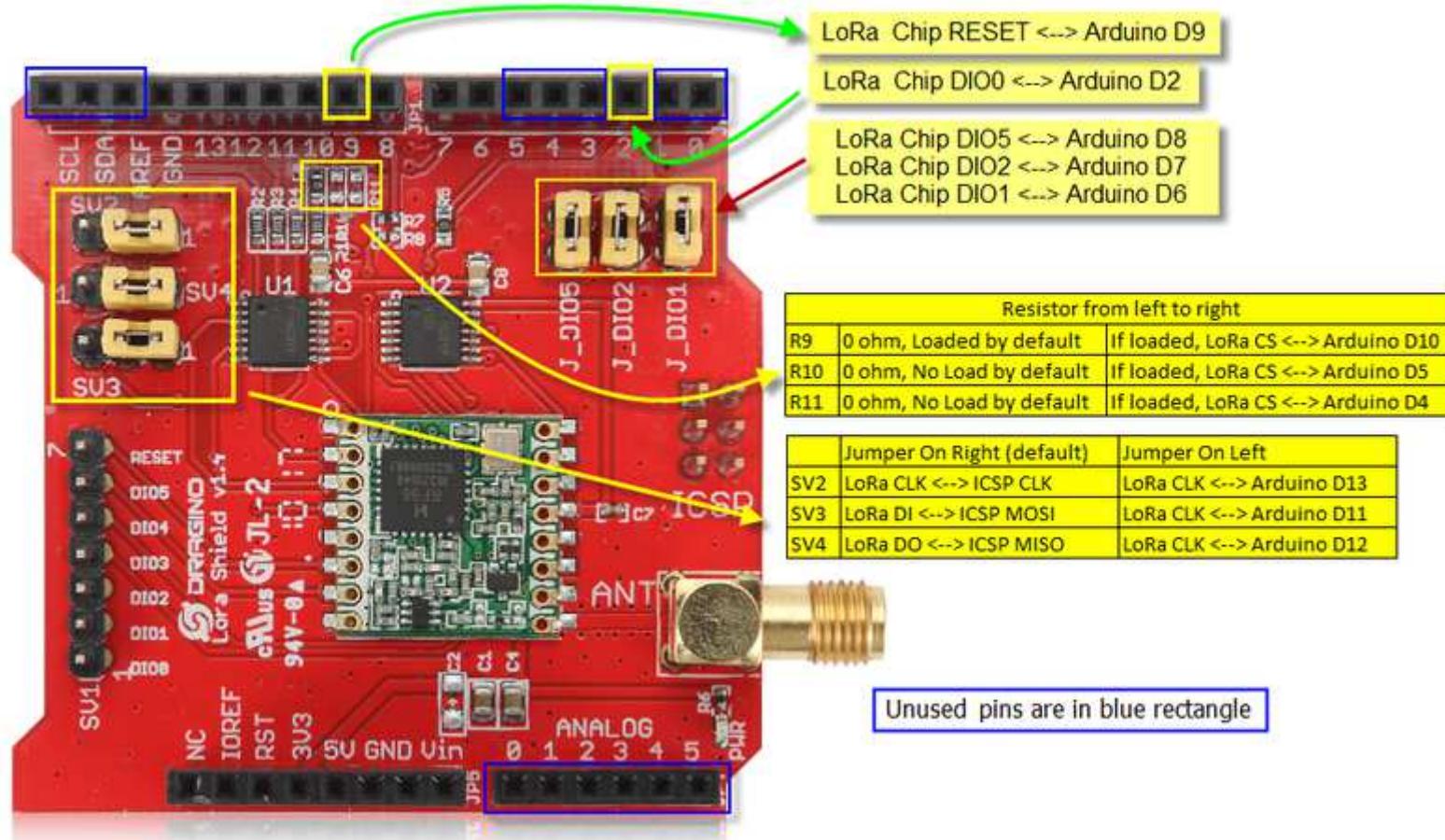


<https://github.com/dragino/Arduino-Profile-Examples>

[https://wiki.dragino.com/index.php?title=Connect\\_to\\_TTN](https://wiki.dragino.com/index.php?title=Connect_to_TTN)

# DRAGINO LoRa + GPS SHIELD

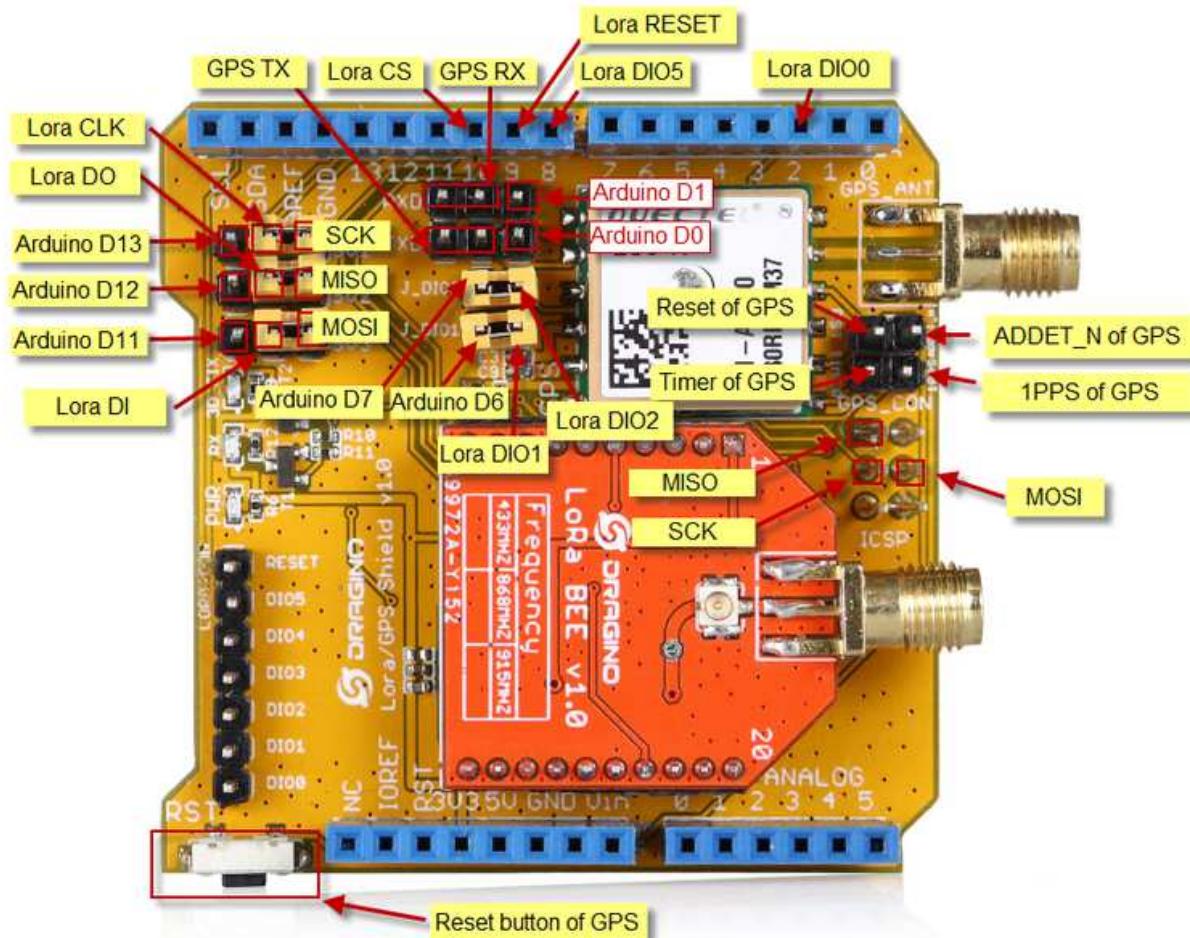
## Pin Mapping For LoRa



[https://wiki.dragino.com/index.php?title=Lora\\_Shield](https://wiki.dragino.com/index.php?title=Lora_Shield)

# TEST GPS + RFM95

# DRAGINO LoRa + GPS SHIELD



1. Tester avec le sketch TinyGPS.ino pour vérifier que le GPS fonctionne
2. Tester avec le sketch LoRa\_GPS\_Shield\_TTN.ino

<https://github.com/dragino/Arduino-Profile-Examples/tree/master/libraries/Dragino/examples/GPS>

[https://wiki.dragino.com/index.php?title=Lora/GPS\\_Shield](https://wiki.dragino.com/index.php?title=Lora/GPS_Shield)

# LoRaWAN End Device Example

1 – Dans l'IDE Arduino:

- Installer la librairie TTN
- Installer la librairie LMIC
- Installer la librairie Low Power Master
- Installer la librairie TinyGPS
- Configurer les pins du RFM95 Shield comme ci-contre

2 – Sur The Things Networks:

Dans l'application, créer un DEVICE:

[https://console.thethingsnetwork.org/applications/thingsuno\\_test/devices](https://console.thethingsnetwork.org/applications/thingsuno_test/devices)

Mettre un DevEUI:

Activer le device sur TheThingsNetwork: méthode ABP

3.1 Depuis la console TTN, récupérer les constantes du device en ABP :

```
const char *devAddr = "26013946";
const char *nwkSKey = "E400FBFA5C76C7836C2380F27ADB77FF";
const char *appSKey = "7B0218AE8B0ACBCBE1414FE19F5ABD61";
```

Et le coller dans l'exemple de l'IDE Arduino + Vérifier que l'on reçoit les données: 1 – sur TTN ou 2 – sur le port série

3 - Charger le sketch **LoRa\_GPS\_Shield\_TTN.ino**

4 – Dans le port série:

Receive data:

3914227: 10

EV\_TXCOMPLETE (includes waiting for RX windows)

Packet queued

LMIC.freq:868100000

Receive data:

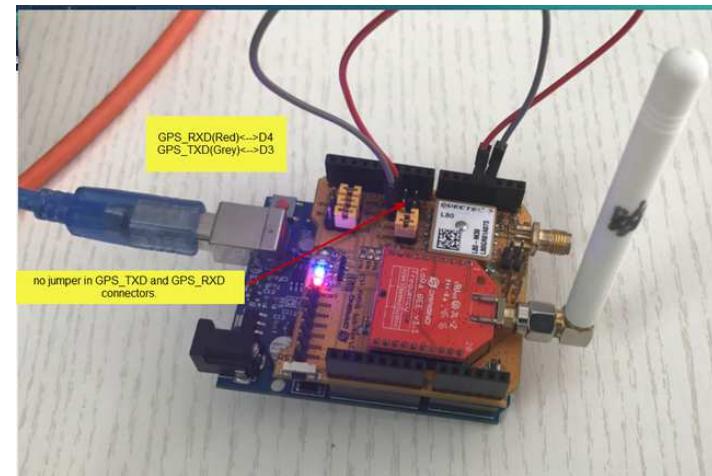
5614245: 10

EV\_TXCOMPLETE (includes waiting for RX windows)

##### NO.1 #####

The longitude and latitude and altitude are:

[1.320810,43.678528,1000000/00]



```
COM3
LoRa GPS Example ----
Connect to TTN
Packet queued
LMIC.freq:868100000

Receive data:
***** NO.1 *****
The longitude and latitude and altitude are:
[114.208820,22.722872,91.80]

***** NO.2 *****
The longitude and latitude and altitude are:
[114.208820,22.722872,91.80]

***** NO.3 *****
The longitude and latitude and altitude are:
[114.208820,22.722872,91.80]

***** NO.4 *****
The longitude and latitude and altitude are:
[114.208820,22.722872,91.80]

***** NO.5 *****
The longitude and latitude and altitude are:
[114.208820,22.722872,91.80]

***** NO.6 *****
The longitude and latitude and altitude are:
[114.208820,22.722872,91.80]

***** NO.7 *****
```

[http://wiki.dragino.com/index.php?title=Connect\\_to\\_TTN#Use\\_LoRa\\_GPS\\_Shield\\_and\\_Arduino\\_as\\_LoRa\\_End\\_Device](http://wiki.dragino.com/index.php?title=Connect_to_TTN#Use_LoRa_GPS_Shield_and_Arduino_as_LoRa_End_Device)

# PAYOUT FUNCTIONS

The screenshot shows the The Things Network Console interface. At the top, there's a navigation bar with 'THE THINGS NETWORK' logo, 'CONSOLE COMMUNITY EDITION', 'Applications' (selected), 'Gateways', 'Support', and a user profile 'dragino\_cheney'. Below the navigation is a breadcrumb path: 'Applications > 1720153058307193 > Payload Formats'. A secondary navigation bar includes 'Overview', 'Devices', 'PayloadFormats' (selected), 'Integrations', 'Data', and 'Settings'.

The main area is titled 'PAYLOAD FORMATS' and contains a 'Payload Format' section. It says 'The payload format sent by your devices' and has a 'Custom' tab selected. Below it are tabs for 'decoder', 'converter', 'validator', and 'encoder'. A red button 'remove.decoder' is visible. A code editor shows the following JavaScript function:

```
1: function Decoder(b, port){  
2:  
3:  
4: var lat = (b[0] | b[1]<<8 | b[2]<<16 | (b[2] & 0x80 ? 0xFF<<24 : 0)) / 10000;  
5:  
6: var lng = (b[3] | b[4]<<8 | b[5]<<16 | (b[5] & 0x80 ? 0xFF<<24 : 0)) / 10000;  
7:  
8: return {  
9: location:{  
10}:  
11};
```

A message 'decoder has no changes' is displayed. Below this is a 'Payload' section with the text 'The payload: function Decoder(b, port) {'. The rest of the payload code is identical to the decoder function above.

On the right side of the interface, there's another 'CONSOLE COMMUNITY EDITION' section showing 'Devices > gps\_test > Data'. It lists several data entries with columns for 'time', 'counter', 'port', 'payload', 'location.latitude', and 'location.longitude'. Red arrows point from the highlighted 'location.longitude' values in the table back to the 'longitude' variable in the payload function code.

```
var lat = (b[0] | b[1]<<8 | b[2]<<16 | (b[2] & 0x80 ? 0xFF<<24 : 0)) / 10000;  
var lng = (b[3] | b[4]<<8 | b[5]<<16 | (b[5] & 0x80 ? 0xFF<<24 : 0)) / 10000;  
return {  
location:{  
latitude: lat,  
longitude: lng  
},  
love: "TTN payload functions"  
};
```

# GATEWAY LoRa

# GATEWAY LoRa: e.g. DRAGINO

Dragino:

[https://wiki.dragino.com/index.php?title=Connect\\_to\\_TTN](https://wiki.dragino.com/index.php?title=Connect_to_TTN)

<https://medium.com/@marcozecchini.2594/connect-dragino-lg01-p-gateway-and-a-dragino-lora-shield-arduino-uno-to-the-things-network-301d27184e05>

Dans le navigateur:

<http://192.168.1.10/cgi-bin/luci/admin>

User will see the login interface of LG01.

The account for Web Login is:

**User Name: root**

**Password: dragino**

dragino-146d78 Status Sensor System Network Logout

**LoRa Gateway Settings**  
Configuration to communicate with LoRa devices and LoRaWAN server.

**LoRaWAN Server Settings**

Server Address	router.eu.thethings.network
Server Port	1700
Gateway ID	a84041168f24ffff

Mail Address: edwin@dragino.com  
Latitude: Location Info  
Longitude: Location Info

Retour Mes équipements connectés

Carte Liste

Paramétrer l'équipement

Wi-Fi	Type d'équipement	Ordinateur
Chromecast	nom	dragino-16f2ec
Chromecast-1	Adresse IP	192.168.1.10
MSI	Adresse MAC	A8:40:41:16:F2:EE
RE450	Connexion Internet	connecté
Galaxy-Tab-A-2016		
Galaxy-A40-de-lilan-1		
PC-53		

Paramétrer son accès à Internet

Autoriser en permanence  
 Bloquer en permanence  
 Planifier

Annuler Enregistrer

Ethernet

DESKTOP-H62D01L

dragino-16f2ec

PC-15

[http://wiki.dragino.com/index.php?title=Connect\\_to\\_TTN#Use\\_LoRa\\_GPS\\_Shield\\_and\\_Arduino\\_as\\_LoRa\\_End\\_Device\\_2](http://wiki.dragino.com/index.php?title=Connect_to_TTN#Use_LoRa_GPS_Shield_and_Arduino_as_LoRa_End_Device_2)

<https://medium.com/@marcozecchini.2594/connect-dragino-lg01-p-gateway-and-a-dragino-lora-shield-arduino-uno-to-the-things-network-301d27184e05>

# TTN to Node-Red

# TTN to NODE-RED

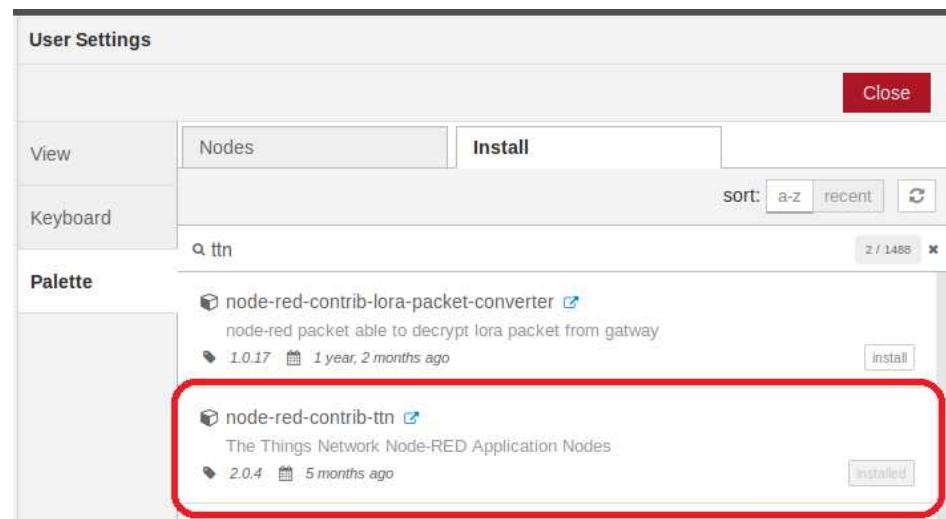
On peut installer ces nodes :

1 - directement à partir du gestionnaire de palette de Node Red

2 - à partir d'un terminal :

```
$ cd $HOME/.node-red $  
npm install node-red-contrib-ttn
```

L'installation terminée, on dispose de 3 nouveaux nodes dans la palette :



<http://silanus.fr/sin/?p=1163>

<https://www.thethingsnetwork.org/docs/applications/nodered/>

<https://www.thethingsnetwork.org/docs/devices/uno/quick-start.html>

<https://www.thethingsnetwork.org/docs/applications/nodered/quick-start.html>

# RECEPTION DES DONNEES DE TTN DANS NODE-RED



Double cliquer sur le node **ttn uplink** pour le configurer :

The screenshot shows the Node-RED interface with two open configuration dialogs. The left dialog is for the 'ttn uplink' node, where fields like Name, App, Device ID, and Field are set. The right dialog is for the 'Edit ttn app node' under 'dragino\_test\_application3', showing fields for App ID, Access Key (with a redacted value), and Discovery address. To the right of these dialogs is the application configuration page for 'dragino\_test\_application3', specifically the 'APPLICATION EUIS' tab. It displays sections for DEVICES (6 registered devices), COLLABORATORS (jgrisolia), and ACCESS KEYS (default key). An arrow points from the 'Access Key' field in the middle dialog to the 'ttngateways' section of the application configuration page.

Déployer le flow et observer les trace Debug :

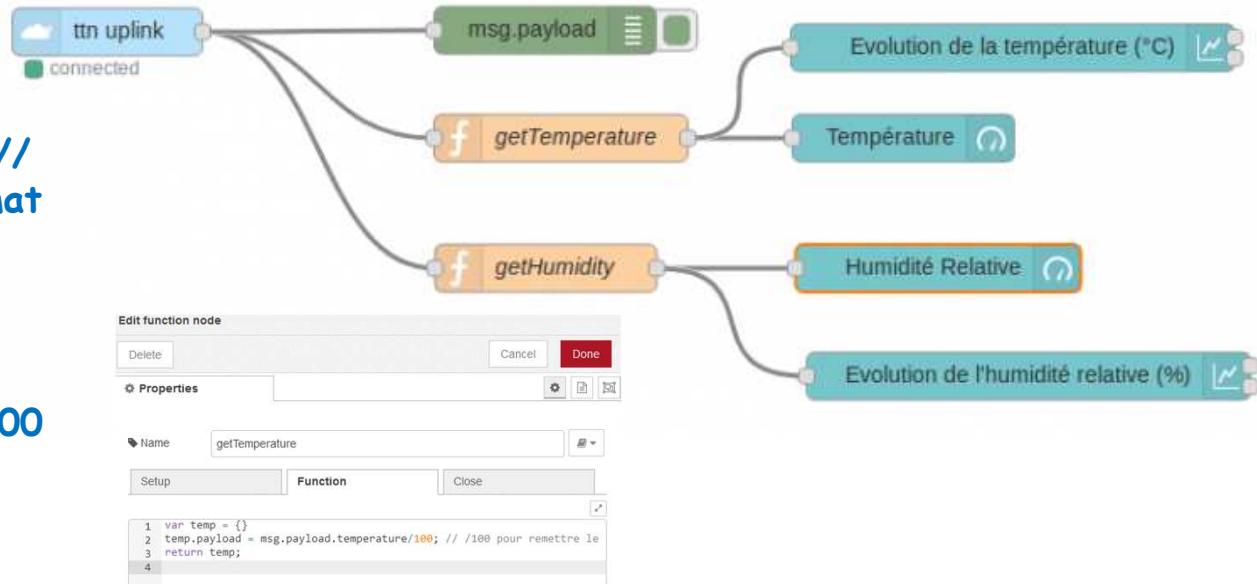
```
26/10/2020 à 10:22:31  node: 43673ca3.91f42c
msg.payload : Object
  ▼object
    humidity: 7625
    temperature: 3215
```

# CONSTRUCTION DU FRONT END NODE-RED

Les fonctions getTemperature() et getHumidity() ont pour rôle de fournir un payload adapté au nodes du dashboard :

```
var temp = {}  
temp.payload =  
msg.payload.temperature/100; //  
/100 pour remettre le bon format  
return temp;
```

```
var humidity = {}  
humidity.payload =  
msg.payload.humidity/100; // 100  
pour remettre le bon format;  
return humidity;
```



```
[{"id": "2c0d3796.eef9e8", "type": "tab", "label": "Flow"}, {"id": "ead5f2fa.e891c8", "type": "debug", "z": "2c0d3796.eef9e8", "name": "", "active": true, "tosidebar": true, "console": false, "complete": false, "statusVal": "", "statusType": "auto", "x": 610, "y": 140, "wires": []}, {"id": "19503597.e16fa2", "type": "ttn uplink", "z": "2c0d3796.eef9e8", "name": "", "app_id": "d89401f7.eba358", "dev_id": "myunodemo", "field": "", "x": 280, "y": 140, "wires": [{"id": "ead5f2fa.e891c8", "z": "99bf72ce.614e28", "x": 6539180a.36143}], {"id": "99bf72ce.614e28", "type": "function", "z": "2c0d3796.eef9e8", "name": "getTemperature", "func": "var temp = {}\\ntemp.payload = msg.payload.temperature/100; // /100 pour remettre le bon format\\nreturn temp;\\n", "x": 610, "y": 280, "wires": [{"id": "3375a2db.276366", "z": "b9214035.335d58"}]}, {"id": "6539180a.36143", "type": "function", "z": "2c0d3796.eef9e8", "name": "getHumidity", "func": "var humidity = {}\\nhumidity.payload = msg.payload.humidity/100; // /100 pour remettre le bon format\\nreturn humidity;\\n", "x": 610, "y": 420, "wires": [{"id": "17080723.4a0889", "z": "3778c092.f7a348"}]}, {"id": "3375a2db.276366", "type": "ui_chart", "z": "2c0d3796.eef9e8", "name": "temperature", "group": "cea21180.eca388", "order": 0, "width": 12, "height": 8, "label": "Evolution de la température (°C)", "chartType": "line", "legend": false, "xformat": "HH:mm:ss", "interpolate": "linear", "nodata": "", "dot": false, "ymin": 0, "ymax": 10000, "removeOlder": 1, "removeOlderPoints": "", "removeOlderUnit": "3600", "cutout": 0, "useOneColor": false, "useUTC": false, "colors": ["#1f77b4", "#aec7e8", "#ff7f0e", "#2ca02c", "#98df8a", "#d62728", "#ff9896", "#9467bd", "#c5b0d5"], "useOldStyle": false, "outputs": 1, "x": 870, "y": 180, "wires": []}, {"id": "b9214035.335d58", "type": "ui_gauge", "z": "2c0d3796.eef9e8", "name": "temperature", "group": "cea21180.eca388", "order": 4, "width": 12, "height": 6, "gtype": "gauge", "title": "Température", "label": "°C", "format": "{{value}}", "min": 0, "max": 1000, "colors": ["#00b500", "#e6e600", "#ca3838"], "seg1": "", "seg2": "", "x": 860, "y": 280, "wires": []}, {"id": "17080723.4a0889", "type": "ui_chart", "z": "2c0d3796.eef9e8", "name": "humidite", "group": "cea21180.eca388", "order": 0, "width": 12, "height": 8, "label": "Evolution de l'humidité (%)", "chartType": "line", "legend": true, "xformat": "HH:mm:ss", "interpolate": "linear", "nodata": "", "dot": false, "ymin": 0, "ymax": 100, "removeOlder": 1, "removeOlderPoints": "", "removeOlderUnit": "3600", "cutout": 0, "useOneColor": false, "useUTC": false, "colors": ["#1f77b4", "#aec7e8", "#ff7f0e", "#2ca02c", "#98df8a", "#d62728", "#ff9896", "#9467bd", "#c5b0d5"], "useOldStyle": false, "outputs": 1, "x": 860, "y": 320, "wires": []}, {"id": "3778c092.f7a348", "type": "ui_gauge", "z": "2c0d3796.eef9e8", "name": "humidite", "group": "cea21180.eca388", "order": 4, "width": 12, "height": 6, "gtype": "gauge", "title": "Humidité Relative", "label": "%HR", "format": "{{value}}", "min": 0, "max": 100, "colors": ["#00b500", "#e6e600", "#ca3838"], "seg1": "", "seg2": "", "x": 860, "y": 420, "wires": []}, {"id": "d89401f7.eba358", "type": "ttn app", "app_id": "dragino_test_application3", "accessKey": "ttn-account-v2.NZWVxiBngSwuBVmiIPcmT3Vh3jTehfyjLGK-MrPNw4", "discovery": "discovery.thethingsnetwork.org:1900"}, {"id": "cea21180.eca388", "type": "ui_group", "name": "TTN", "tab": "5a0a53f3.c08c3c", "order": 1, "disp": true, "width": 24, "collaps": false}, {"id": "5a0a53f3.c08c3c", "type": "ui_tab", "name": "TTN", "icon": "dashboard"}]
```

# PROPRIETES DES NODES ET CHARTS NODE-RED

The image displays three separate configuration panels from the Node-RED interface, each showing different properties for a node.

**Edit chart node > Edit dashboard group node**

- Name:** TTN
- Tab:** TTN
- Width:** 24
- Display group name:**
- Allow group to be collapsed:**

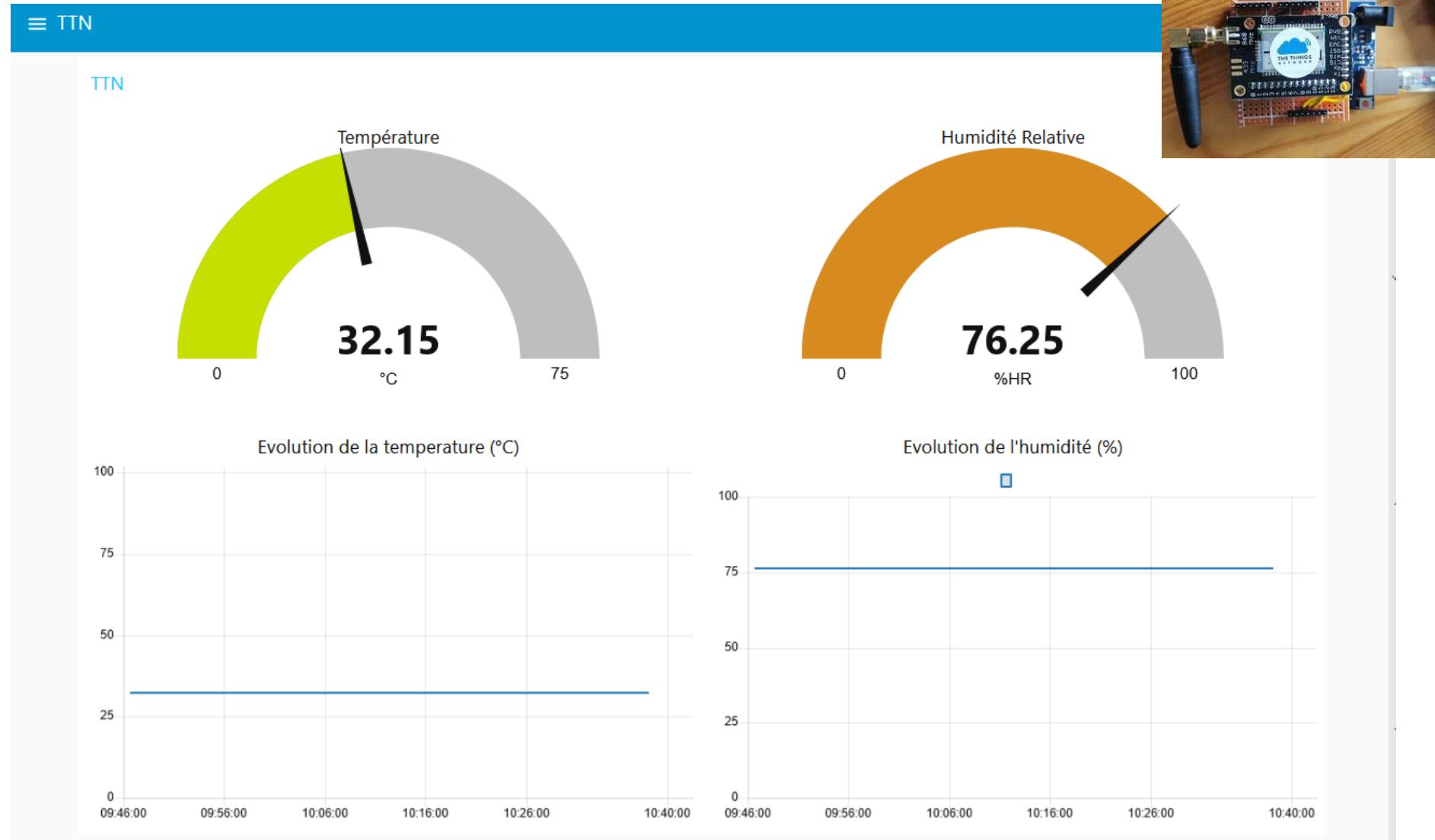
**Edit chart node**

- Group:** [TTN] TTN
- Size:** 12 x 8
- Label:** Evolution de la température (°C)
- Type:** Line chart  enlarge points
- X-axis:** last 1 hours OR 1000 points
- X-axis Label:** HH:mm:ss  as UTC
- Y-axis:** min [ ] max [ ]
- Legend:** None  Interpolate linear
- Series Colours:** A grid of six color swatches: blue, light blue, orange, green, light green, red, pink, purple, and light purple.
- Blank label:** display this text before valid data arrives
- Name:** temperature

**Edit gauge node**

- Group:** [TTN] TTN
- Size:** 12 x 6
- Type:** Gauge
- Label:** Température
- Value format:** {{value}}
- Units:** °C
- Range:** min 0 max 1000
- Colour gradient:** A horizontal bar with three segments: green, yellow, and red.
- Sectors:** 0 ... optional ... optional ... 1000
- Name:** temperature

# VISUALISATION DES DONNEES SUR LE FRONT-END NODE-RED



# MQTT from TTN

# MQTT FROM TTN to NODE-RED

Saisir le flow suivant :

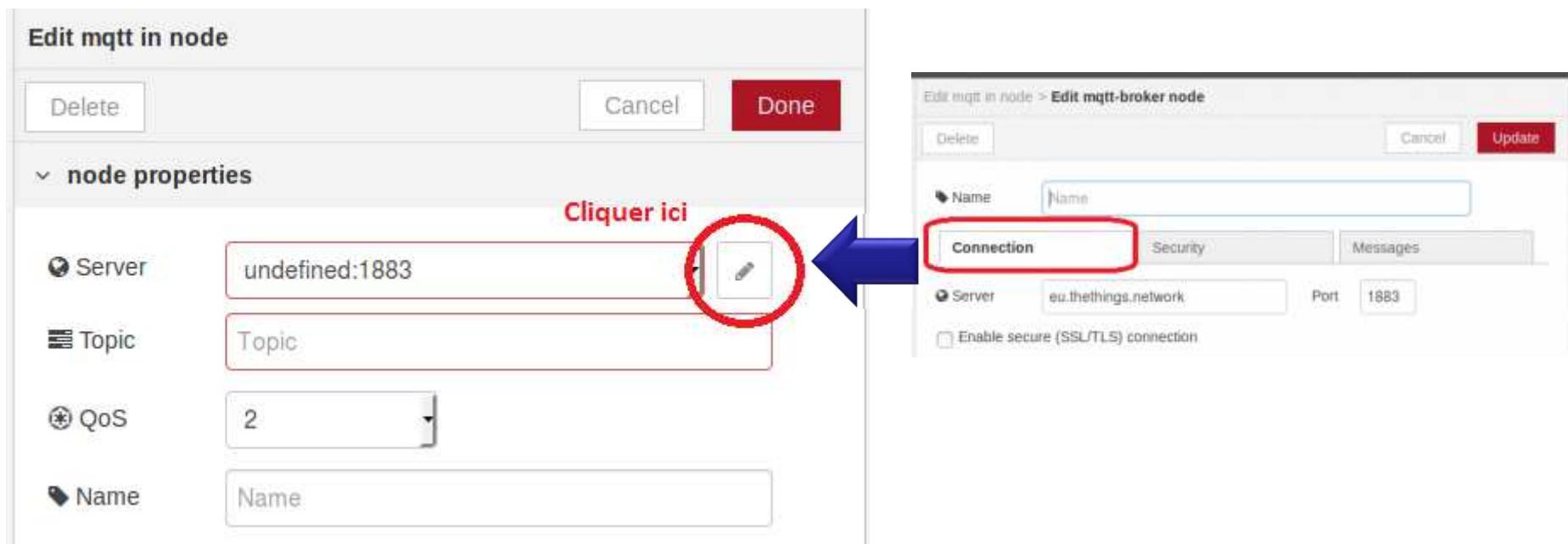


- Configurer le node mqtt in :
- La consultation de la documentation de l'[API](#) de **The Things Network** permet de trouver les éléments suivants nécessaires à la configuration du node :

- **Host:** <Region>.thethings.network, where <Region> is last part of the handler you registered your application to, e.g. eu.
- **Port:** 1883
- **Username:** Application ID
- **Password:** Application Access Key

- **Topic:** <AppID>/devices/<DevID>/up

Double-cliquer sur le node mqtt in pour faire apparaître la boîte de dialogue de configuration.



# MQTT FROM TTN to NODE-RED

The screenshot displays three main sections of the The Things Network Console:

- Left Panel (Edit mqtt in node > Edit mqtt-broker node):** Shows the "Connection" tab selected. It includes fields for Server (eu.thethings.network), Port (1883), and Client ID (Leave blank). A red box highlights the "Connection" tab. Below this is the "APPLICATION OVERVIEW" section, which shows Application ID (gestion-clim-lab-c12) highlighted with a red box. Other details include Description (Gestion des paramètres de la climatisation salle C12 du Lycée Alphonse Benoit), Created (yesterday), and Handler (ttn-handler-eu).
- Middle Panel (Edit mqtt in node > Edit mqtt-broker node):** Shows the "Security" tab selected. It includes fields for Username (gestion-clim-lab-c12) and Password (\*\*\*\*\*). A red box highlights the "Security" tab. An arrow points from the "Security" tab in the left panel to this panel.
- Right Panel (Edit mqtt in node):** Shows the "node properties" section. It includes fields for Server (eu.thethings.network:1883), Topic (gestion-clim-lab-c12/devices/temp-hum-sensor-1/up), QoS (2), and Name (mqtt from TTN). A red box highlights the "Topic" field. Below this is the "DEVICE OVERVIEW" section, which shows Application ID (gestion-clim-lab-c12) and Device ID (temp-hum-sensor-1) both highlighted with red boxes. Other details include Activation Method (OTAA), Device EUI (00 04 A3 0B 00 1E 7D 95), and Application EUI (70 B3 D5 7E D0 01 0A 30).

## II – 3 ADRESSABLE DEVICES ?

II – 3 ADRESSABLE  
DEVICES ?

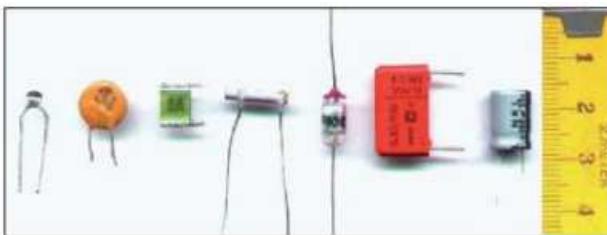
# WHAT COMPONENTS TO USE WITH THE ARDUINO?



La résistance

La résistance s'oppose au passage du courant, proportionnellement à sa "résistance" exprimée en Ohm. Un code de couleurs, ci dessous permet de reconnaître cette valeur.

Symbol européen



Les condensateurs

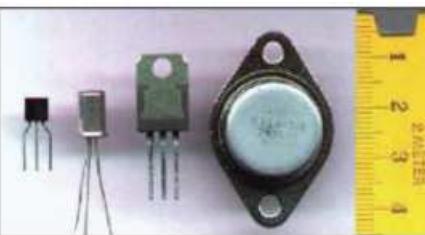
Les condensateurs peuvent stocker un peu de courant si on les charge, mais comme un tonneau percé, ils renvoient ce courant instantanément si ils sont branchés à un organe consommateur de courant.

Ils peuvent être polarisés ou non, dans des boîtiers très divers. Leur valeur s'exprime en Farad (F)



Symbol

Reconnaissance de leur valeur:  
A cause de la diversité des modèles, se reporter aux ressources sur le web



Le transistor

Le transistor est généralement utilisé comme une sorte de multiplicateur de puissance: lorsqu'on lui fait passer un courant faible, mais variable dans un de ses 3 pattes, il autorise proportionnellement le passage d'un gros courant dans une autre des 3 pattes.



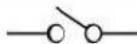
Symboles



transistor PNP

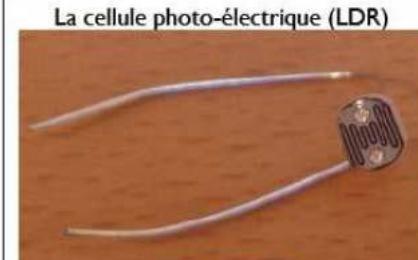


L'interrupteur



L'interrupteur ouvre ou ferme un circuit. Il y a toutes sortes d'interrupteurs.

Sur l'Arduino, utiliser un interrupteur pour déclencher un événement nécessite d'utiliser un composant supplémentaire: une résistance de 10K ohms. Voir "Montages d'électronique interactive".



La cellule photo-électrique (LDR)



photocell

La cellule photo-électrique (LDR)  
C'est une résistance variable, en fonction de la luminosité qu'elle reçoit. Sa résistance diminue quand elle reçoit de la lumière. On s'en sert donc de capteur de luminosité. Non polarisée. Pour lire sa valeur avec une Arduino, il faut également l'associer avec une résistance équivalente à sa résistance maxi (dans le noir). Voir "Montages d'électronique interactive".



Le piezo

Le transducteur piezo-électrique est un composant réversible: il peut aussi bien être utilisé en capteur de chocs ou de vibrations qu'en actionneur pouvant émettre des sons stridents parfois modulables.

# QUELS COMPOSANTS UTILISER AVEC L'ARDUINO ?

**Le potentiomètre**

**Le potentiomètre**

Le potentiomètre, rotatif comme ici, ou à glissière, est une résistance variable. Entre les extrémités, il y a la résistance maximale. La patte centrale est le curseur. C'est la résistance entre cette patte centrale et une extrémité que l'on peut faire varier en tournant le bouton. Le potentiomètre est donc un capteur. Il se branche sur les entrées analogiques de l'Arduino. De très nombreux capteurs sont basés sur le principe de résistance variable et se câblent presque de la même façon: la cellule photo-électrique, le capteur de pression, le fil résistif, etc.

**Le relais**

**Le relais**

Le relais est un composant à 4 broches minimum. C'est un interrupteur que l'on peut commander en envoyant un petit courant. Au repos, il est normalement fermé, ou normalement ouvert, selon le modèle. On peut s'en servir avec l'Arduino pour commander des machines en haute tension ( 230V par exemple), ou pour déclencher toute machine ou lumière.

**La diode**

**La diode**

La diode ne laisse passer le courant que dans un seul sens. C'est un composant polarisé: on reconnaît toujours son anneau coloré d'un côté du composant, correspondant à la cathode.

**La LED**

**La LED**

La diode électroluminescente (LED) émet de la lumière. Elle est polarisée: la patte "+" est la plus longue, l'autre patte est la patte "-". Les broches numériques de l'Arduino, lorsqu'elles sont configurées en sorties et qu'elles sont à l'état 1 ou haut (HIGH), fournissent une tension de 5 volts, supérieure à ce que peut accepter une LED courante, sauf certaines LEDs. Les LED doivent donc être couplées en série avec une résistance.

## Gearhead Motor

### ★ DC Motor with gearbox

- Not fast but provide more torque

### ★ Servo Motor

- Gearhead motor with position feedback
- Feedback is often from potentiometer
- Pulsing the motor moves it to particular position within 180 degree range
- Can't move 360 degrees but can be positioned precisely within the 180 degree range



## Stepper Motor

### ★ Precise positioning & 360 degrees range

- Move in discrete steps around a circle
- A 200 step motor would move 1.8 degrees per step around the full 360 degrees
- Continuous rotation in either direction
- Good torque
- Complex to connect



## Solenoids and Actuators

### ★ Linear Motion

- Pull or Push

### ★ Types

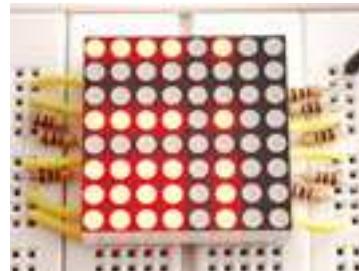
- Solenoid
- Actuator
- Microactuator



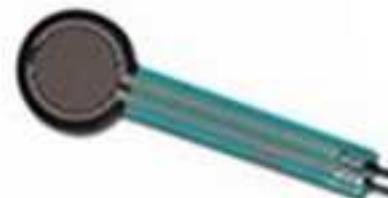
# QUELS COMPOSANTS UTILISER AVEC L'ARDUINO ?



GPS



Matrice de LED



Capteur Force



Capteur Effet Hall



Flex Sensor



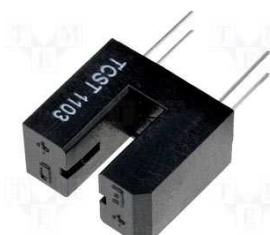
Modules Sonar



Capteur de Gaz



Capteur de mouvement PIR



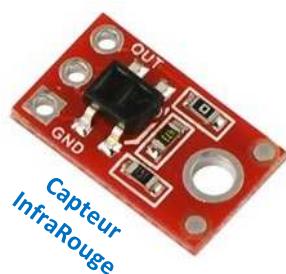
OptoCoupleur



Capteur Inclinaison



Capteur Température/Humidité



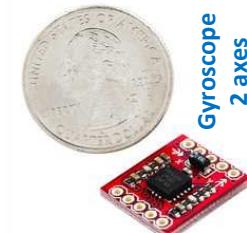
Capteur InfraRouge



Capteur Son



Capteur Flamme



Gyroscope 2 axes



Ecran LCD 16\*2



Digital Strip

# QUELS COMPOSANTS UTILISER AVEC L'ARDUINO ?



Capteur Rotation



Mini-capteur Manche



Capteur de chaleur



Capteur Couleur



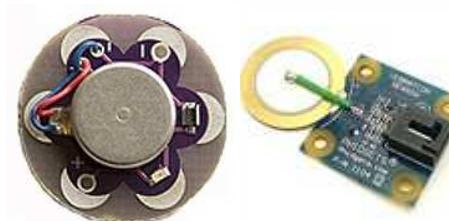
Camera JPEG



Capteur Lumière



Capteur Lumière



Capteur Vibrations



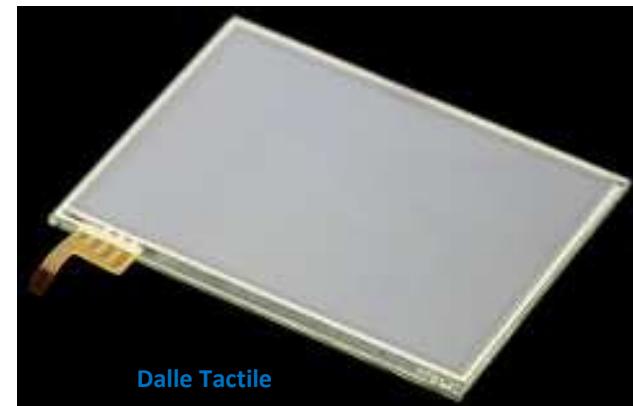
Capteur Pression-  
Barométrique



Kinect



Nunchuk



Dalle Tactile

et bien d'autres que nous ne pourrions citer par manque de place...

# SENSORS Vs ACTUATORS

## Sensors :

Contact	Pression	Numérique	<ul style="list-style-type: none"> <li>• Interrupteur</li> <li>• DFRobot Digital Push Button</li> </ul>
		Analogique	<ul style="list-style-type: none"> <li>• Capteur de force</li> </ul>
	Rotation	Analogue	<ul style="list-style-type: none"> <li>• Potentiomètre</li> </ul>
		TTL	<ul style="list-style-type: none"> <li>• Encodeur de rotation</li> </ul>
Espace	Mouvement	Numérique	<ul style="list-style-type: none"> <li>• Capteur PIR de mouvement</li> </ul>
	Présence	Numérique	<ul style="list-style-type: none"> <li>• Tapis sensitif</li> <li>• TimeTech Laser Sensor</li> </ul>
		Numérique	<ul style="list-style-type: none"> <li>• DFRobot Capacitive Touch Sensor</li> </ul>
	Proximité	Analogue	<ul style="list-style-type: none"> <li>• IR compound eye</li> </ul>
		Intervalle	<ul style="list-style-type: none"> <li>• Capacitance</li> </ul>
		I2C	<ul style="list-style-type: none"> <li>• Capteur de chaleur 8 pixels</li> </ul>
	Distance	Analogue	<ul style="list-style-type: none"> <li>• Capteur de distance Sharp GP2D12</li> <li>• MaxSonar EZ1</li> </ul>
		Intervalle ou série	<ul style="list-style-type: none"> <li>• Capteur de Distance Ultrasonique SeeedStudio</li> </ul>
	Lignes	Numérique	<ul style="list-style-type: none"> <li>• DFRobot Line Tracking Sensor</li> </ul>
		Analogique	<ul style="list-style-type: none"> <li>• DFRobot Grayscale Sensor</li> </ul>
	Couleur	TTL	<ul style="list-style-type: none"> <li>• TCS230-DB Color Sensor</li> </ul>
Lumière	Visible	Analogue	<ul style="list-style-type: none"> <li>• Photo-résistance</li> <li>• DFRobot Ambient Light Sensor</li> </ul>
	Invisible	Numérique	<ul style="list-style-type: none"> <li>• Capteur d'infrarouge</li> <li>• Télécommande</li> </ul>
Son	Amplitude	Analogue	<ul style="list-style-type: none"> <li>• Inex Sound Detector</li> </ul>
	Vibration	Analogue	<ul style="list-style-type: none"> <li>• Piezo</li> </ul>
Déplacement	Localisation	Série	<ul style="list-style-type: none"> <li>• Boussole numérique HM55B</li> <li>• Parallax GPS Module</li> </ul>
	Accélération	Série	<ul style="list-style-type: none"> <li>• NunChucky</li> <li>• DE-ACCM5G</li> </ul>
Identification	RFID	Série	<ul style="list-style-type: none"> <li>• Seeedstudio RFID</li> </ul>
Biométrie	Empreintes		<ul style="list-style-type: none"> <li>• ARA-ME-01</li> </ul>
	Alcool		<ul style="list-style-type: none"> <li>• Alcohol Gas Sensor MQ-3</li> </ul>
	Rythme cardiaque		<ul style="list-style-type: none"> <li>• Polar Heart Rate Module - RMCM01</li> </ul>
Environnement	Température	Analogique	<ul style="list-style-type: none"> <li>• Capteur de température TMP36</li> <li>• Capteur de Température Ambiante Robotics Connection</li> </ul>
	Humidité		<ul style="list-style-type: none"> <li>• Sensirion Temp/Humidity</li> </ul>
	Vent		<ul style="list-style-type: none"> <li>• VORTEX Wind Speed Sensor</li> </ul>

## Actuators:

Lumière	Simple	<ul style="list-style-type: none"> <li>• DEL</li> <li>• DFRobot White LED</li> <li>• BlinkM MaxM</li> <li>• Relais</li> <li>• DFRobot Single Relay</li> </ul>
	Matrice	<ul style="list-style-type: none"> <li>• Démultiplexeur</li> <li>• MAX7219</li> <li>• MondoMatrix LEDMatrix ↗</li> <li>• Rainbowduino</li> </ul>
	PWM multiples	<ul style="list-style-type: none"> <li>• TLC5940</li> </ul>
Moteurs	Courant continu (CC)	<ul style="list-style-type: none"> <li>• Circuit de base pour un moteur CC</li> <li>• Adafruit Motor Shield</li> <li>• Relais</li> <li>• DFRobot Single Relay</li> <li>• Seeedstudio Relay Shield ↗</li> </ul>
	Servomoteur	<ul style="list-style-type: none"> <li>• Circuit de base pour un servomoteur</li> <li>• DFRobot IO Expansion Shield</li> </ul>
	Moteur à pas	<ul style="list-style-type: none"> <li>• Circuit de base pour un moteur à pas</li> <li>• Adafruit Motor Shield</li> </ul>
	120V AC	<ul style="list-style-type: none"> <li>• Relais</li> <li>• DFRobot Single Relay</li> <li>• Seeedstudio Relay Shield ↗</li> <li>• DMX</li> </ul>
	Solénoïde	<ul style="list-style-type: none"> <li>• Relais</li> </ul>
	Audio	<ul style="list-style-type: none"> <li>• Resistor ladder audio DAC ↗</li> <li>• Arduino Realtime Audio Effects ↗</li> </ul>

<http://wiki.t-o-f.info/index.php?n=Arduino.Capteur>

=> PAY US ON THE COMPONENTS OF THE ARDUINO KIT!

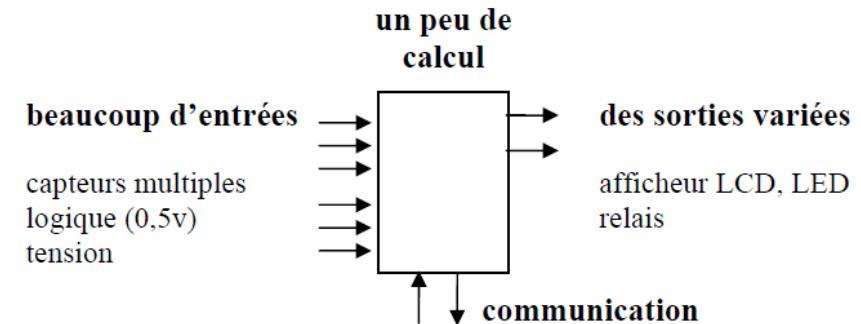
# I - MICROCONTROLLERS

## I - MICROCONTROLLERS

## BUT BY THE WAY: A MICROCONTROLLER IS NEEDED FOR WHAT?

There is so much to say... here's an explanation simple, reductive, but gives you an overview of its features:

- 1 - A microcontroller takes something as input
- 2 - he takes a decision in relation to a program you entered in his memory, he calculates, he communicates...
- 3 - It produces an output result



Current platforms based on microcontrollers (such as the Arduino, Teensy, ESP32...) are real small computers taking lots of entries, dialogue between them by WIFI, showing items on OLED screens, pressing the engines etc.

Programs can be quite complex. But this isn't the goal in reality! Even if microcontrollers are powerful and can misrepresent them in computer,

**GOAL:** be normally independent interfaces, aggregating the sensors and actuators and which delegate to more complex computer calculations tasks.

### Applications:

- microcontrollers are often used in embedded systems development, requiring specialized treatment (car stereos, mobile phones, mp3 player, GPS, etc.)
- but also to achieve rapid prototyping applications.

### Remember that:

a software implementation is always slower than a hard-wired logic implementation: the microprocessor executes a statement at the time (see also language VHDL-AMS=> FPGA programming)

# MICROCONTROLLERS: PRO & CONS

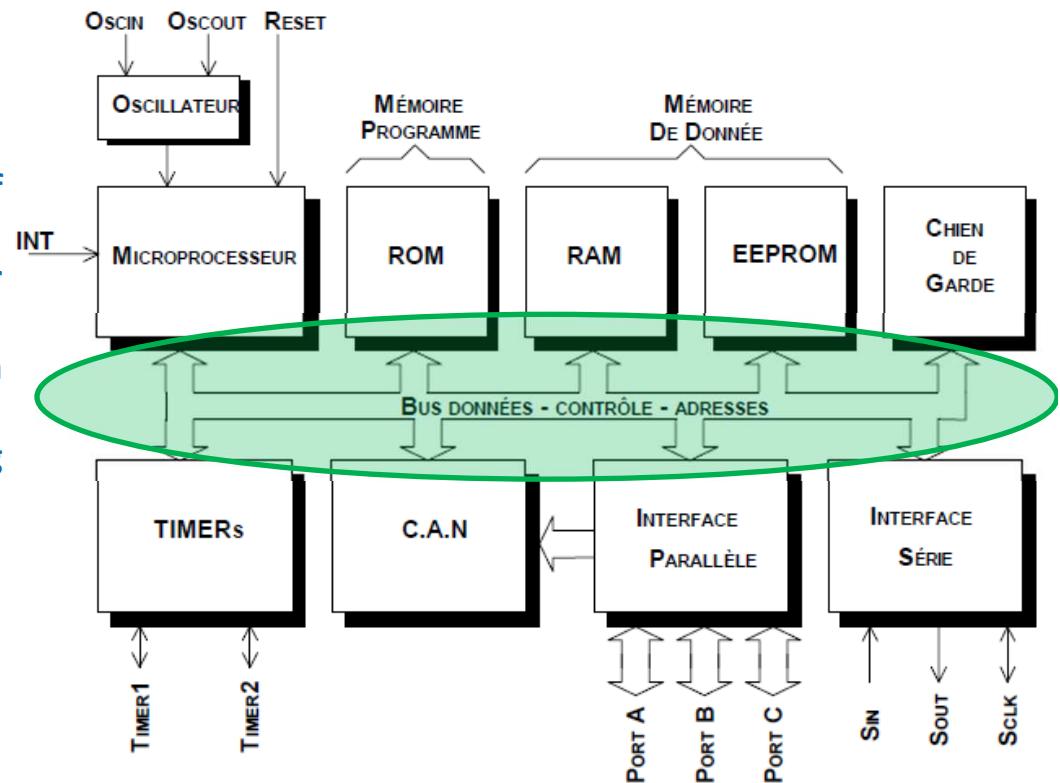
The  $\mu$ controleurs = benefit of the  $\mu$ processeurs but limited to applications that do not require too much computing power, number of components very reduced, but often oversizing before the needs of the application

- Benefits of microcontrollers:
  - Simplification of the layout of the circuit board (no need to draw bus!)
  - Reduced costs of Labor (design and Assembly)
  - reduction of congestion of the material and the circuit board
  - increase in the reliability of the system the
    - number of components ↓
    - connections components/materials and component PCB ↓.
  - Integration in technology MOS, CMOS, or HCMOS (High Speed CMOS) decrease in consumption - programming and advanced simulation environments
- ⇒ The defects of microcontrollers:
- often oversized before the application needs
  - investment in development tools (except for the Arduino...)
  - write programs, test and test their implementation on the material that surrounds the  $\mu$ controleur possible
  - incompatibility of the development tools for microcontrollers same brand

# DEFINITION

A microcontroller is an integrated circuit gathering in one case all the electronic functions of a system:

- a microprocessor (C.P.U.),
- memory (RAM and EEPROM) data,
- program memory (ROM,...),
- parallel interfaces for the connection of inputs/outputs,
- interfaces (synchronous or asynchronous) series for dialogue with other units.
- timers for generating or measuring signals with a high temporal precision,
- analog-to-digital converters for processing analog signals.
- a structure receiving the oscillator to pace the µproc



-And the communications bus: address, data and controls that connect these various devices !

# INSIDE OF THE MICROCONTROLLER

**1- C.P.U. (MICROPROCESSOR):** usually consists of the following elements:

- a counter program pointing the address of the next instruction to execute
- an arithmetic and logical unit (ALU) to perform operations between the accumulator and an operand: loaded with calculations +, -, \*, /, AND, OR, NOT
- different registers containing the operands, the results of the operations, the addressing mode...

**Typical operation:**

- 1 - it sequentially executes the instructions stored in the program memory.
- 2 - it is capable of operating on binary words whose size, in bits, is that of the data bus

**the microprocessor needs some elements to work:**

- so-called read only memory ROM: mainly to store the program
- Free memory RAM : mainly to store the variables,
- I/O devices (mainly to interact with the outside world)
- a clock to pace it (mainly in quartz).

**It may be noted that there are 2 categories of microprocessors: the CISC and the RISC.**

- **CISC (Complex Instruction Set Computer):** the microprocessor has a significant number of instructions. Each of them runs in several periods of clocks.
- **RISC (Reduced Instruction Set Computer):** the microprocessor has a small number of instructions. Each of them runs in a clock period.

## RISC VS CISC

In 1975, David Patterson (University of California, Berkeley):

80% of the instructions of a program use only 20% of the instructions of the CPU

idea: remove instructions that are not and thanks to space-saving add useful elements (such as the memory, records...)

=> strategic choice to shorten the time CPU:

$$CPU\_time = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} * \frac{\text{avg\_cycles}}{\text{instructions}} * \frac{\text{seconds}}{\text{avg\_cycles}}$$

1 - The ICSC shorten the execution time by reducing the number of instructions per program a expensive processor (CISC) as complex and much heat

2 - the RISC shorten the execution time by reducing the number of clock cycles per statement (i.e., simple instructions take less time to interpret)

=> An economic processor (RISC) as simple and which consumes few... but that the program requires more standard and so more memory.

-In addition, with fewer transistors RISC heater less => can more easily increase in frequency.

Saved transistors are used to create memory or registers (the PowerPC RISC has 2 or 3 x more than a PC (CISC)), which avoids too often access to memory (slowing the processor).

In the end: the ICSC decrease processing speed, but instructions are more complex, more powerful, and more numerous

on the contrary, the RISC is to move the major complexities of the hardware to the software,

=> a CISC to make a small complex,

=> a RISC program to make a big complicated program.

# RISC VS CISC

ASM x 86 program fragment (multiplication of two integers):

**CISC**

```
mov ax, 10
mov bx, 5
mul bx, ax
```

$$10 * 5 = 50$$

**RISC**

```
Begin      mov ax, 0
           mov bx, 10
           mov cx, 5
           add ax, bx
loop Begin
```

$$10 + 10 + 10 + 10 + 10 = 50$$



**RISC 1 / CISC 0**

The number of total cycle for the CISC:  $(2 \text{ movs} + 1 \text{ cycle}) + (1 \text{ mul} \times 30 \text{ cycles}) = 32 \text{ cycles}$

the number of total cycle for the RISC:  $(3 \text{ movs} \times 1 \text{ cycle}) + (5 \times \text{adds} 1 \text{ cycle}) + (5 \times \text{loops} 1 \text{ cycle}) = 13 \text{ cycles}$

What happens to a multiplication of  $10 * 500000$ ?

**RISC 1 / CISC 1**

$\Rightarrow$  not so simple !

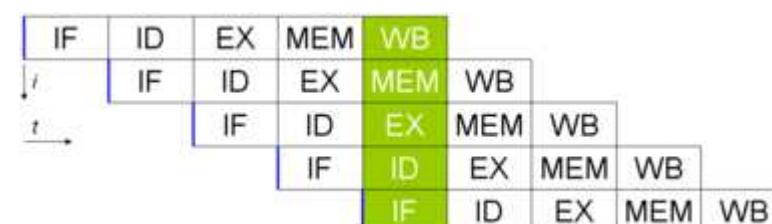
Unfortunately, the RISC technology also has flaws:

indeed, fewer instructions is that the resulting software, to functions to accomplish equal, occupies more memory than a machine ICSC, as well statically than dynamically.

All RISC machines use the technique of "pipelining" to increase their benefits in terms of statements executed in the unit of time.



Sequencing of instructions in a processor without pipeline. It takes 15 cycles to run 3 instructions.



Sequencing of instructions in a processor equipped with a pipeline to 5 floors. It takes 9 cycles to run 5 instructions. At  $t = 5$ , all the floors of the pipeline are solicited, and 5 operations take place at the same time.

# RISC VS CISC

In summary:

There is no winner!

It depends on the context:

CISC	RISC
Used in laptops and desktop computers, made by Intel or AMD	Used in smartphones and tablets, based around ARM processor
Has more complex hardware	Has simpler hardware
Multiple machine cycles per instruction	Single machine cycle per instruction
Physically larger in size and require more silicon to make thus more expensive	Smaller in size as less complex circuitry required, less silicon needed to make thus cheaper
Greater energy consumption	Lower energy requirements, and can go into "sleep mode" when not actively processing
More intensive tasks will do better with CISC	Run at lower clock speed, but can perform simpler tasks more quickly than CISC
Can't support pipelining	Can support pipelining

-In the CISC architecture, used especially in the range of the x 86 AMD, and INTEL, Cyrix..., designers rely on the reduction of the number of instructions required to run the program, designing very powerful instructions, which has the inconvenience of increase in average number of cycles machine needed to complete an instruction.

In this case, the frequency of the system is reduced because we have to introduce a phase of interpretation of the machine through the firmware code.

- In the RISC architecture, we put a lot on the minimization of the number of cycles machine and it makes most executable instructions in a single clock cycle, allowing to increase the frequency of the system.
- This is possible by eliminating the phase of interpretation through the simplicity of instructions that can be decoded and executed directly by a simple cable control unit. The simplification of the type RISC machines control units is particularly advantageous for the realization of the CPU on a single VLSI chip.
- Obtained space allows, equal area of silica, to substantially increase the number of internal registers and embed directly on the chip's cache to exploit to the maximum speed of the microprocessor.

## RISC VS CISC



### 2019 INTEL & AMD VS ARM

INTEL

**i7 8565U** 1.8GHz - 4.6Ghz, 4/8 Cores, 15W

AMD

**R7 3700U** 2.2Ghz - 3.8Ghz, 4/8 Cores, 15W

QUALCOMM

**SD 8CX** 2.75Ghz - ??, 8/8 Cores, 7W

**Do INTEL and AMD have any concerns  
to worry about?**

## THE MICROCONTROLLER INSIDE: MEMORY...

### 2.2 memory programs:

it contains instructions of the program to be run by the CPU.

This type of memory, called ROM, is only read.

Its programming requires a special procedure and equipment.

There are different types according to their programming mode: ROM content is programmed during its manufacture:

- The PROM programmable electrically only once by the developer (also called OTPROM)
- the EPROM programmable electrically and erasable by UV light (also called UV PROM).
- the programmable and erasable EEPROM electrically,
- The Flash, much faster than the others.

### 2.3 data memory:

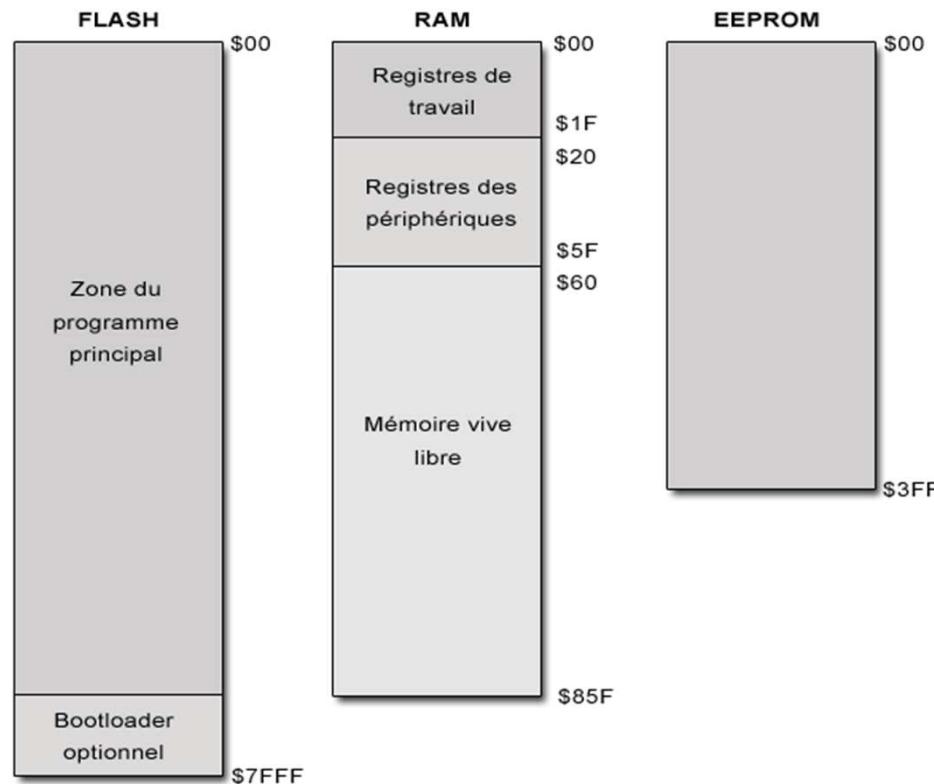
allows you to temporarily store the data generated by the CPU during the different phases of the digital processing (results of operations, States of sensors...).

These briefs are available in writing and reading.

There are 2 types:

- of the memory (RAM) volatile (lost data in case of power failure) with a time of reading and writing short (a few ns),
- Read only memory (EEPROM) (permanently data even if power supply failure) with a writing time high enough (a few ms) compared to the time of reading which is quite low (a few ns).

# THE MICROCONTROLLER INSIDE: MEMORY...



For example: the ATMEGA32 is equipped with 3 types of main memory:

1 - the **FLASH**: memory that keeps its content when it is no longer under tension. Capacity is 32 KB (\$00 to \$7FFF).

At the end of this memory-online space preset to the **BOOTLOADER**: small piece of program that runs before your own program in order to perform various operations.

2 - **RAM**: memory that loses its content when it is no longer supplied. Capacity of 2 k (\$00 to \$85F).

Allows to:

- store temporary information when running a program,
- contains the return address during a call to a sub program or an interrupt routine.

3 - In the **EEPROM**: memory that retains data when power failure. Capacity of 1 KB (\$00 to \$3FF).

It allows to memorize information that must be recharged to power.

Eeprom (Electrically Erasable Programmable Read Only Memory)

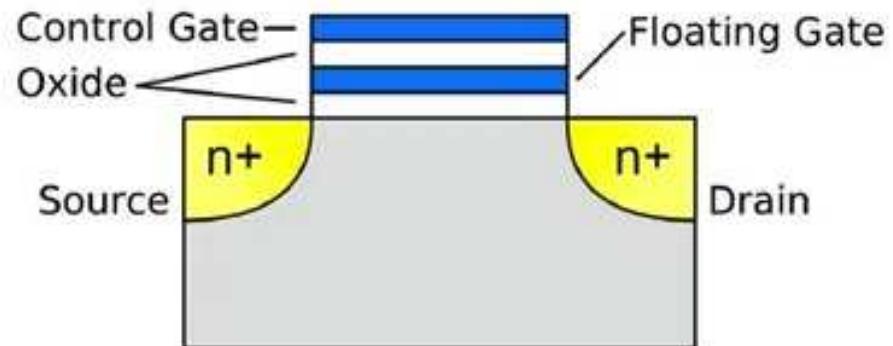
Taille 64 – 2k bytes

Durée de rétention > 10 ans

Cycle d'écriture lent (3-10ms)

nombre de cycles limités (100'000 à 10<sup>6</sup>)

Cycle de lecture normal



## THE MICROCONTROLLER INSIDE: PARALLEL INTERFACE

### 2.4 the parallel INTERFACE :

this type of interface, distributed on several ports (maximum 8-bit), allows to take into account:

- logical States applied input (State of sensors) or
- generate binary output (control of actuators) signals.

-The pins of these ports can be configured as input or output, with different options (open reminder resistance, output collector, interruption....).

-The configuration as well as the logical state of these pins is achieved by reading or writing operations in different registers associated with each port.

There are usually:

- A registry of direction for a configuration input or output,
- a data register copying logical States of each pin of ports,
- a registry of option allowing several configurations as input or output.

# INSIDE OF THE MICROCONTROLLER

## 2.5 the INTERFACE series:

it allows the microcontroller to communicate with other systems microprocessor based.

The data sent or received arise in the form of a temporal (on a single bit) succession of values binary images of a Word.

There are 2 types of serial link: asynchronous and SYNCHRONOUS

### Ø asynchronous serial communication :

this device does not provide synchronization clock signal.

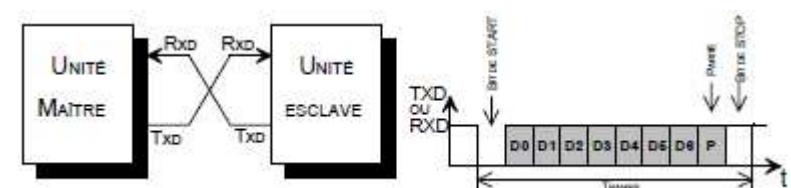
Liaison units each have an internal clock clocked at the same frequency.

When one wants to make a binary Word, it generates a front descending on its issuing line.

At the end of the show of this word, the line returns to the high level.

*The data to be transmitted may contain an additional bit called "parity" and used for error correction.*

Example: your favorite serial RS232 managed by a UART or USART (Universal Asynchronous Receiver Transmitter) which is a universal asynchronous transceiver, (which has a buffer of 128 characters in the Arduino 7bits + 1 parity bit), and who does too, through a converter USB-serial, the connection to the PC.



### Ø SYNCHRONOUS serial :

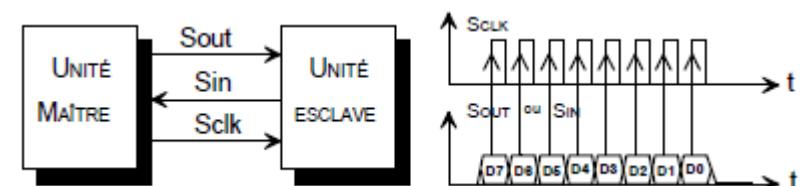
in this device the transmission is synchronized by a clock signal from the master unit, jointly to the signal carrying data.

Examples:

SPI (Serial Peripheral Interface Bus) by Motorola => 3 wires

I2C (Inter Integrated Circuit) by Philips ~ 1980 => 2 wires

One Wire (Protocol of MAXIM DALLAS) => 1 wire



## INSIDE OF THE MICROCONTROLLER

### 2.6 the CAN: analog-to-digital converter:

the built-in microcontrollers CAN is usually the type of "Successive Approximations".

It has several accessible multiplexed inputs via the pins of the parallel interface ports.

The CAN has normally 2 registers:

- a register of data containing the result of the conversion,
- a registry of control to launch and monitor conversion.

### 2.7. the TIMER: the Timers allow you to perform the following functions:

- Generation of a periodic signal modulated (PWM) pulse width or not,
- Generation of a calibrated pulse
- timing,
- counting of events.

Several registers associated with Timers allow to configure the modes described here.

### 2.8. the WATCHDOG (Watch Dog): is a system anti-plantage of the microcontroller, which ensures that there is no prolonged executions of a same sequence of instruction.

Example: If the program is waiting for the result of an external system (e.g. CAN) or if the program runs in an endless loop => no response-online block.

In practice: it resets periodically (at a definable interval) internal record thanks to the instruction `clrwdt` (clear watchdog).

If the program is blocked, more happening in the branch of reset and counting goes to the end (e.g. 16-bit counter that causes a Reset when it reaches its maximum (0xFFFF)), triggers the watchdog that restarts the program.

The watchdog of the AVR microcontrollers is timed by an internal clock at 1 MHz, so it can work also in the absence of the system clock because it is independent of it.

# Inside of the microcontroller: The ATMEGA328

The registers of devices allow to access

- the configuration of embedded devices (USART, CAN, SPI,...) and timers, EEPROM...
- and get the States on their functioning.

## Timers 8 and 16 bits

For example: the table below lists the name of the registers of the ATMEGA32

### Memory

### Parallel I/O interfaces

### Serie port

### Analog/Device converter

<http://www.atmicroprog.com/cours/atmega/regsystem.php>

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
\$3F (\$5F)	SREG	I	T	H	S	V	N	Z	C
\$3E (\$5E)	SPH	—	—	—	—	SP11	SP10	SP9	SP8
\$3D (\$5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
\$3C (\$5C)	OCR0	Timer/Counter0 Output Compare Register							
\$3B (\$5B)	GICR	INT1	INT0	INT2	—	—	—	IVSEL	IVCE
\$3A (\$5A)	GPIO	INTF1	INTF0	INTF2	—	—	—	—	—
\$39 (\$59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
\$38 (\$58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
\$37 (\$57)	SPMCR	SPMIE	RWWSRE	—	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
\$36 (\$56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	—	TWIE
\$35 (\$55)	MUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
\$34 (\$54)	MCUCSR	JTD	ISC2	—	JTRF	WDRF	BORF	EXTRF	PORF
\$33 (\$53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
\$32 (\$52)	TCNT0	Timer/Counter0 (8 Bits)							
\$31 <sup>(1)</sup> (\$51) <sup>(1)</sup>	OSCCAL	Analog Oscillator Register							
\$30 (\$50)	OCDR	ADTS2	ADTS1	ADTS0	—	ACME	PUD	PSR2	PSR10
\$2F (\$4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
\$2E (\$4E)	TCCR1B	ICNC1	ICES1	—	WGM13	WGM12	CS12	CS11	CS10
\$2D (\$4D)	TCNT1H	Timer/Counter1 - Counter Register High Byte							
\$2C (\$4C)	TCNT1L	Timer/Counter1 - Counter Register Low Byte							
\$2B (\$4B)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte							
\$2A (\$4A)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte							
\$29 (\$49)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte							
\$28 (\$48)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte							
\$27 (\$47)	ICR1H	Timer/Counter1 - Input Capture Register High Byte							
\$26 (\$46)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte							
\$25 (\$45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
\$24 (\$44)	TCNT2	Timer/Counter2 (8 Bits)							
\$23 (\$43)	OCR2	Timer/Counter2 Output Compare Register							
\$22 (\$42)	ASSR	—	—	—	—	AS2	TCN2UB	OCR2UB	TCR2UB
\$21 (\$41)	WDTCR	—	—	—	WDTOE	WDE	WDP2	WDP1	WDP0
\$20 <sup>(2)</sup> (\$40) <sup>(2)</sup>	UBRRH	URSEL	—	—	—	UBRR[11:8]			
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
\$1F (\$3F)	EEARH	EEPROM Address Register Low Byte							
\$1E (\$3E)	EEARL	EEPROM Address Register High Byte							
\$1D (\$3D)	EEDR	EEPROM Data Register							
\$1C (\$3C)	ECCR								
\$1B (\$3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
\$1A (\$3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
\$19 (\$39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
\$18 (\$38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
\$17 (\$37)	DDRB	DBB7	DBB6	DBB5	DBB4	DBB3	DBB2	DBB1	DBB0
\$16 (\$36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
\$15 (\$35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
\$14 (\$34)	DDRC	DCD7	DCD6	DCD5	DCD4	DCD3	DCD2	DCD1	DCD0
\$13 (\$33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
\$12 (\$32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
\$11 (\$31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
\$10 (\$30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
\$0F (\$2F)	SPDR	SPI Data Register							
\$0E (\$2E)	SPSR	SSRE	WGOE	—	—	—	—	—	SPI2X
\$0D (\$2D)	SPCR	SPFE	SPBE	—	MSTR	CPOL	CPHA	SPR1	SPR0
\$0C (\$2C)	UDR	USART I/O Data Register							
\$0B (\$2B)	UCSRA	RA0	RA1	UDRE	FE	DOR	PE	U2X	MPCM
\$0A (\$2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
\$09 (\$29)	UBRRH	USART Baud Rate Register Low Byte							
\$08 (\$28)	ACSR	ACD	ACBG	AC0	AC1	ACIE	ACIC	ACIS1	ACIS0
\$07 (\$27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
\$06 (\$26)	ADCsRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
\$05 (\$25)	ADCH	ADC Data Register High Byte							
\$04 (\$24)	ADCL	ADC Data Register Low Byte							
\$03 (\$23)	TWDR	Two-wire Serial Interface Data Register							
\$02 (\$22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
\$01 (\$21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	—	TWPS1	TWPS0
\$00 (\$20)	TWBR	Two-wire Serial Interface Bit Rate Register							



# INSIDE OF THE MICROCONTROLLER: THE BUS...

The various blocks are connected by 3 bus:

- the address bus: allows the µprocessor to select the memory box or device to which it wants access to read or write information (instruction or given);
- the data bus: allows the transfer of information between different blocks. This information will be:
  - either instructions,
  - the data source or destination memory or devices;
- the control bus: indicates if the current operation is a read or a write, if a device requires an interruption etc.

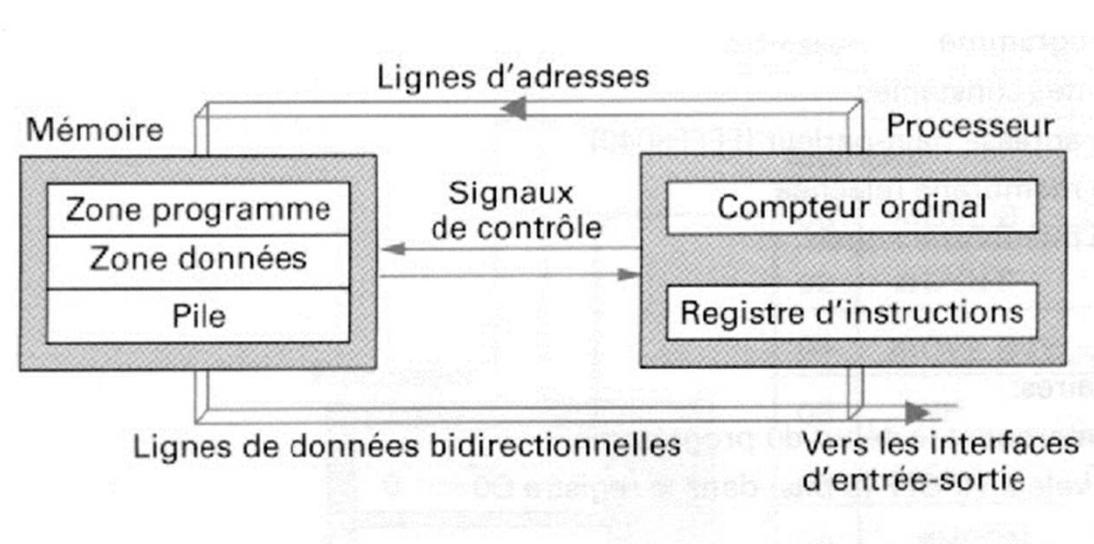
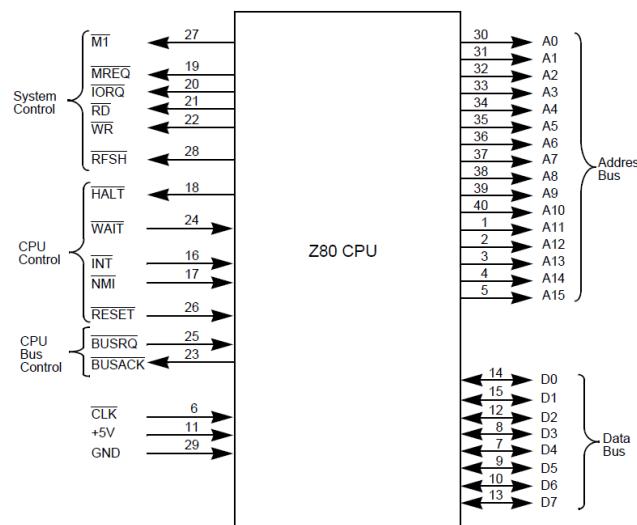


Figure 3. Z80 I/O Pin Configuration

# FUSES

Fuses have always existed only for protection against the downloading of program in order to prevent duplication or disassembly:

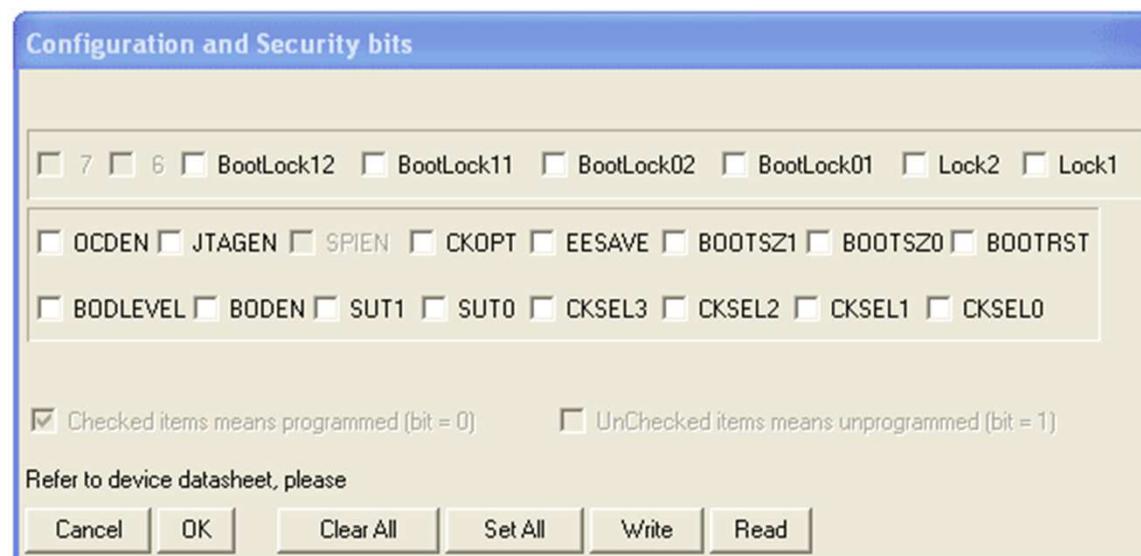
*In short of what lock all industrial curiosity program!*

For example: the ATMEGA class has 21 fuses allowing multiple settings: source and clock speed, detector of undervoltage (brown-out)...

Jargon used by Atmel:

A programmed fuse as for value '0'  
An unprogrammed fuse as for value "1"

so we can retain the intellectual logic is reversed.



Screenshot of the fuses on an ATMEGA32 with Pony Prog software

# FUSES

## THE PROTECTION fuses:

**Bootlock2, Bootlock1:** pair of fuses for control rights of access to FLASH and EEPROM memories.

Bootlock2	Bootlock1	Protection type
1	1	No restrictions of access to the memory.
1	0	Programming the FLASH or EEPROM by programming series and parallel. The programming of the fuses is disabled
0	0	No programming and checking the FLASH or EEPROM by programming series and parallel. The programming of the fuses is disabled

## CALIBRATED internal RC OSCILLATOR MODE:

Uses the internal oscillator of RC type at speeds pre-set 1, 2, 4, 8 MHz. These frequencies are given for a supply voltage of 5V and 25 ° C.

CKOPT	CKSEL3	CKSEL2	CKSEL1	CKSELO	Frequency (MHz)
1	0	0	0	1	1
1	0	0	1	0	2
1	0	0	1	1	4
1	0	1	0	0	8

## EXTERNAL quartz OSCILLATOR MODE:

The most used Mode.

CKOPT allows you to select 2 different modes:

1 - if programmed (0): rail to rail, either the maximum capacity.

2. If not programmed (1): the oscillation circuit will consume less energy but will have a limited operating frequency.

ATMEL calls to program this fuse for f=16 MHz

CKOPT	CKSEL3	CKSEL2	CKSEL1	Frequency range (MHz)	oscillation capacitor Recommended
1	1	0	1	0.4 -0.9 *	-
1	1	1	0	0.9 - 3.0	12 - 22 pF
1	1	1	1	3.0 - 8.0	12 - 22 pF
0	101,110,111			1.0 <=	12 - 22 pF

# INSIDE OF THE MICROCONTROLLER

**INSTRUCTION set:** we class the instructions that a µcontroleur is able to perform in groups:

**1 - transfer Instructions:** the µcontroller spends much of his time to move bytes from one place to another in the system:

- a device to a registry internal or vice versa,
- a registry internal to the RAM memory or vice versa.

*Attention: a direct transfer from one memory to another or to a device, or an entry in ROM memory, generally cannot be done, the structure of the microcontroller makes mandatory the passage of information by one of its internal registers. Note that, with some exceptions, it is rather a copy that a transfer since the memory of origin box keeps its information (as long as there is no written something else instead).*

**2 - Arithmetic instructions:** A µcontroller, especially 8-bit, isn't a great mathematician. It is only able to make additions, subtractions, multiplications and divisions. All complex mathematical operations (treatment of large numbers, fractional, square roots, trigonometric functions,...) must be reduced to a succession of simple operations only on bytes.

**3 - Logical instructions:** the µcontroleurs perform logical operations: and, or, XOR, not (inverter), rotations, shifts... Operations are operated simultaneously on the bits corresponding to the two registers.

Example: compare two bytes A and B according to a logical operation, is to realize a subtraction which overlooked the result;

We just want to know if it is null ( $\Rightarrow A = B$ ), positive ( $A > B$ ) or negative ( $A < B$ ). These indications are registered in States indicators (small stores 1 bit located in the processor)

# A L'INTERIEUR DU MICRO-CONTROLEUR

## 4 instructions of E/S, used to:

- read the status of a port of entry (allowing interfacing to switches, switches, optocouplers, analog/digital converters, keyboards, etc.);
- write information to the registry to an output port, which keeps the information at the disposal of the external circuits (leds, motors, relays, digital/analog converters, etc.)
- write or read information in the registers of a serial port.

## 5 - Connection instructions: instructions that alter the normal conduct of the program.

Jumps and subroutines are distinguished:

- the jumps cause a branch of the program to a memory address that is not contiguous to the place where you are.
- The subroutine or subprogram is a program part that is needed in several places in the execution of the main program.

The big difference compared to the jump, this is when connecting to memorize the address where we come from, in order to come back once the subprogram completed. This is done by memorizing the start address in a special register (stack) of the microcontroller.

Both jumps that subroutines can be:

- unconditional
- conditional, which is achieved if a certain condition is met

## 6 - Instructions:

- instructions of the battery management (area of memory RAM to store data during the execution of the program);
- the processor control instructions: e.g. switching mode low consumption, control of devices shipped (i.e. on the same chip as the processor);
- instructions to position the processor internal indicators.

*These instructions vary highly according to families of microcontrollers.*

# ALL ABOUT THE MICROCONTROLLER

In view of the integration of all these elements in a single integrated circuit case, it takes very few external electronic components around the µcontroller to make it work:

## 1 - The power:

all current µcontrollers work under a voltage of 1.8 to 6V with a predilection for the 5V (as a result the abundance of the so-called family TTL logic circuits that use this tension).

Soon however => 3.3V.

Very important: the voltage must be stabilized so that it works correctly.

2 - The clock: A µcontroller is a sequential logic circuit, i.e. that works at the pace of a rectangular signal, called clock that cadence all his internal circuitry.

Range: a few kHz to several MHz (compared to a PC we're over the GHz.)

Note: most  $f \uparrow$ , it is quick but more it consumes, and more it heats up!

Attention: it is recommended to have a stable clock-online controlled by a quartz or a ceramic resonator, only electronic components capable of generating stable and precise frequency signals.

3 - Reset circuit: just like a PC, it runs a program permanently, and if the program crashes, there must be out of this situation

=> reset restart the program without disconnecting the power supply.

## II WHAT FAMILY OF MICROCONTROLLERS?

# BRIEF HISTORY AND DEVELOPMENT OF MICROCONTROLLERS

3 major steps:

1 - first microcontrollers date back to the 1970s: (Texas TMS1000 and Intel 4004)  
mask-ROM



Mask - ROM: hidden memory, written directly in Silicon memory.)

Needed:

- 1 - write a program,
- 2 - pass it on to the manufacturer and a few months later:
- 3 - receive a number of copies of our integrated circuit
- 4 - hope that our program is fair and it works!

2 - EPROM: Erasable Programmable Read Only Memory (with erasure by ultraviolet rays): Intel 8748, Motorola 68705, etc...



3 - EEPROM: since PIC 16c84 (1993)



## A FEW FAMILIES OF MICROCONTROLLERS



### 8 bits:

- PIC Microchip
- AVR Atmel
- Derivatives 80C51, 80C52



### 16 bits:

- dsPIC
- MSP430 Texas Instruments



### 32 bits:

- AVR32, PIC 32, MIPS, Power PC, STM32
- ARM produced by various companies: NxP, STMicro, Texas, Samsung, Infineon, Toshiba, Analog Device, Qualcomm, Freescale...

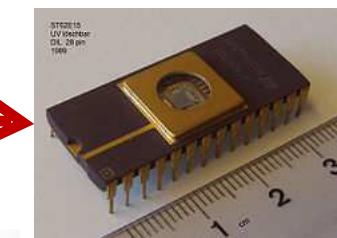
# FAMILLES DE MICRO-CONTROLEUR

- Atmel AT91 family;
- **Atmel AVR** family; 
- C167 Siemens/Infineon ;
- Hitachi H8 family;
- Intel 8051 family (some recent processors using a 8051 core, which is supplemented by various devices (IO ports, counters/timers, A/D and D/A, watch dog, voltage supervisor, etc.);)
- Intel 8085, originally designed to be a microprocessor, often practice as a microcontroller.
- **Motorola/Freescale 68HC08 , 68HC12 et 68HC11** family : barcode readers, hotel card programmers, robots of amateurs, and other systems onboard.
- **PIC Microchip family**;
- **ST6 STMicroelectronics family**;
- ADuC d'Analog Devices ;
- **PICBASIC Comfile Technology**;
- MSP430 Texas Instruments.
- **8080, z80, Rabbit**: the 8080 is one of the great ancestors, but z80 and Rabbit are still very much alive
- **PSoC Cypress**
- LPC21xx ARM7-TDMI Philips
- NEC: V800, K0...

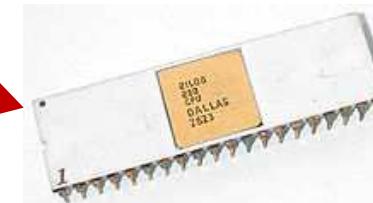


Le MC68HC11A8 est disponible en version DIP à 48 broches, ainsi que PLCC à 52 broches, comme ici.

Quatre microcontrôleurs PIC de familles différentes : 18F, 16F, 12F et 10F.



Microcontrôleur ST6 équipé d'une mémoire EPROM effaçable aux UV.



This processor was released for the first time in July 1976. In the early 1980s he was very popular in the design of 8 - bit computers like the Radio Shack TRS-80, the Sinclair ZX80, ZX81, ZX Spectrum, MSX, the Amstrad CPC standard and later in embedded systems.

# AUTOPSY OF AN ARDUINO UNO PLATFORM

## THE ATMEL AVR microcontrollers

The Arduino platform is based on the architecture of the Atmel ATMEGA microcontrollers that use a heart AVR (Advanced Virtual RISC):



AVR is the term used by Atmel for the heart of the processor and the family of microcontrollers RISC implementing them (developed in 1996 by ATMEL corporation).

The architecture was designed by two students of the Institute of technology of Norway (NTH : Norges Tekniske Høgskole): Alf-Egil Bogen and Vegard Wollan

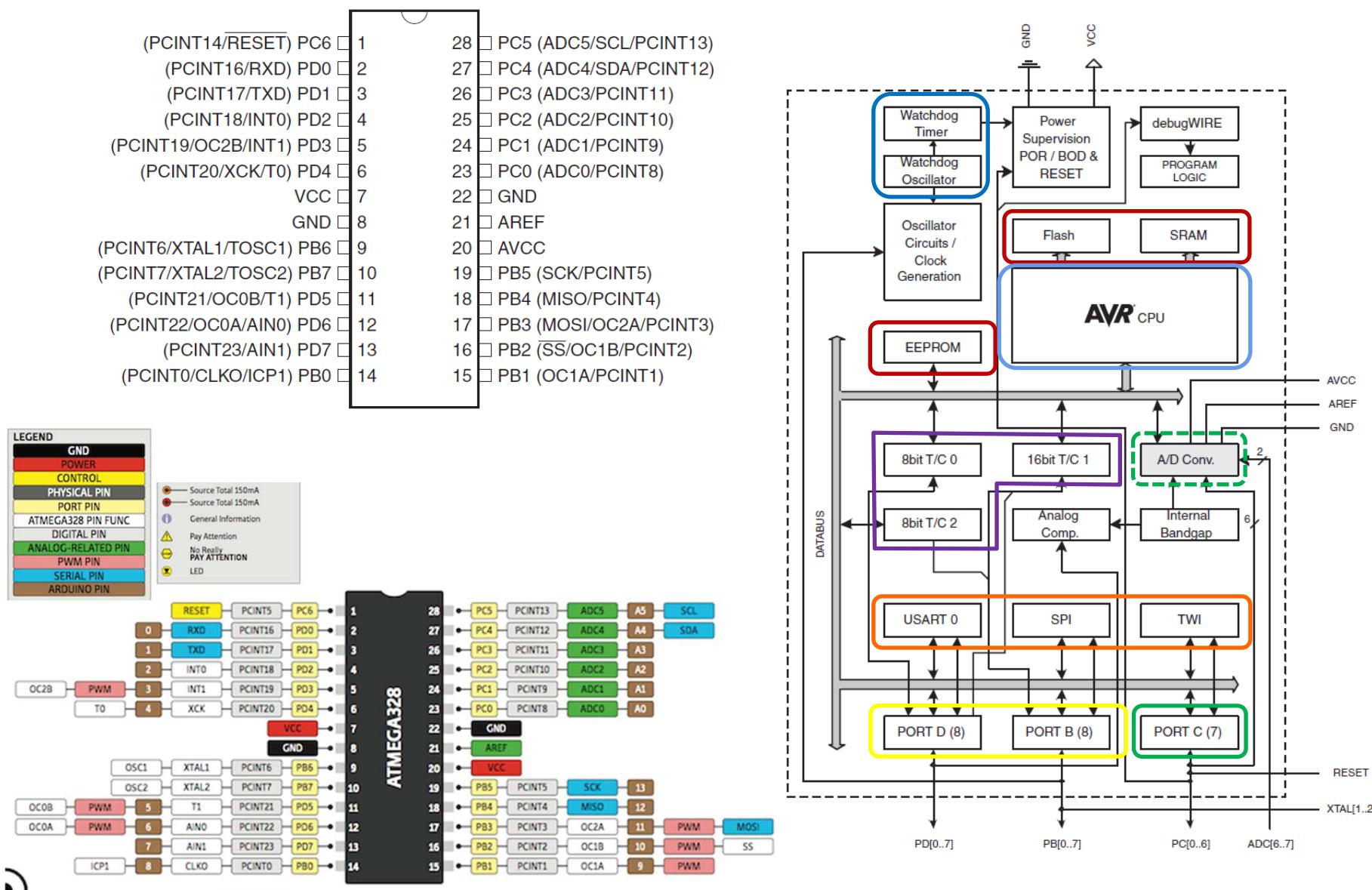
=> Alf-Egil Bogen and Vegard Wollan RISC microcontroller that gives also Advanced Virtual RISC.

Historically: the AT90S8515 was the first microcontroller based on the AVR architecture, but it's the AT90S1200 who was the 1st microcontroller to meet commercial success in 1997.



***Attention: the language machine of AVR is incompatible with the Microchip PIC family.***

# ATMEL AVR ARCHITECTURE



SOURCE: [http://www.atmel.com/dyn/resources/prod\\_documents/8271S.pdf](http://www.atmel.com/dyn/resources/prod_documents/8271S.pdf)

# LES MICROCONTROLEURS AVR d'ATMEL – ATMEGA328

The heart of the kernel is based on 131 instructions RISC architecture. This number of instructions is quite high for such an architecture => Advanced RISC Architecture

Up to 20 million instructions per second (MIPS) for a 20 MHz Quartz (the majority of the instructions being conducted in 1 or 2 cycles)

⇒ microcontrollers AVR are so very fast!

⇒ The internal architecture of the ATMEGA328 allows us to point out the following devices:

2 Timer/Counter 8 bit with independent pre-division factor

1 Timer/Counter 16-bit with independent pre-division

1 clock time real factor with external quartz

6 channels PWM Converter

1 Analog/digital 8 channels (10 bit resolution)

1 Communication interface asynchronous USART

1 interface series synchronous I2C (Philips I2C Compatible)

1 communication Interface synchronous SPI (also used to programming In-situ)

2 interrupts on the changes of States of pins

1 timer Watchdog programmable...

The above list is the list of devices that have a major role in communication with the outside world to the microcontroller. It takes also on other typical internal functions, such as different types of Flash memory...

## Features

- High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C<sup>(1)</sup>
  - Optional Boot Code Section with Independent Lock Bits
  - In-System Programming by On-chip Boot Program
  - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
  - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4MHz @ 1.8 - 5.5V, 0 - 10MHz @ 2.7 - 5.5V, 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

# AVR instructions

## Table of Contents

Instruction Set Nomenclature.....	1
1. I/O Registers.....	13
1.1. RAMPX, RAMPY, and RAMPZ.....	13
1.2. RAMPD.....	13
1.3. EIND.....	13
1.4. Stack.....	13
1.5. Flags.....	13
2. The Program and Data Addressing Modes.....	14
2.1. Register Direct, Single Register Rd.....	14
2.2. Register Direct - Two Registers, Rd and Rr.....	15
2.3. I/O Direct.....	15
2.4. Data Direct.....	16
2.5. Data Indirect with Displacement.....	16
2.6. Data Indirect.....	17
2.7. Data Indirect with Pre-decrement.....	17
2.8. Data Indirect with Post-increment.....	18
2.9. Program Memory Constant Addressing using the LPM, ELPM, and SPM Instructions.....	18
2.10. Program Memory with Post-increment using the LPM Z+ and ELPM Z+ Instruction.....	19
2.11. Direct Program Addressing, JMP and CALL.....	19
2.12. Indirect Program Addressing, IJMP and ICALL.....	20
2.13. Relative Program Addressing, RJMP and RCALL.....	20
3. Conditional Branch Summary.....	21
4. Instruction Set Summary.....	22
5. ADC – Add with Carry.....	30
5.1. Description.....	30
5.2. Status Register (SREG) and Boolean Formula.....	30
6. ADD – Add without Carry.....	32
6.1. Description.....	32
6.2. Status Register (SREG) and Boolean Formula.....	32
7. ADIW – Add Immediate to Word.....	33
7.1. Description.....	33
7.2. Status Register (SREG) and Boolean Formula.....	33
8. AND – Logical AND.....	35
8.1. Description.....	35
8.2. Status Register (SREG) and Boolean Formula.....	35
9. ANDI – Logical AND with Immediate.....	36
9.1. Description.....	36

71.1. Description.....	109
71.2. Status Register (SREG) and Boolean Formula.....	110
72. LD (LDD) – Load Indirect From Data Space to Register using Index Z.....	112
72.1. Description.....	112
72.2. Status Register (SREG) and Boolean Formula.....	113
73. LDI – Load Immediate.....	115
73.1. Description.....	115
73.2. Status Register (SREG) and Boolean Formula.....	115
74. LDS – Load Direct from Data Space.....	116
74.1. Description.....	116
74.2. Status Register (SREG) and Boolean Formula.....	116
75. LDS (16-bit) – Load Direct from Data Space.....	117
75.1. Description.....	117
75.2. Status Register (SREG) and Boolean Formula.....	117
76. LPM – Load Program Memory.....	118
76.1. Description.....	118
76.2. Status Register (SREG) and Boolean Formula.....	118
77. LSL – Logical Shift Left.....	120
77.1. Description.....	120
77.2. Status Register (SREG) and Boolean Formula.....	120
78. LSR – Logical Shift Right.....	122
78.1. Description.....	122
78.2. Status Register (SREG) and Boolean Formula.....	122
79. MOV – Copy Register.....	123
79.1. Description.....	123
79.2. Status Register (SREG) and Boolean Formula.....	123
80. MOVW – Copy Register Word.....	124
80.1. Description.....	124
80.2. Status Register (SREG) and Boolean Formula.....	124
81. MUL – Multiply Unsigned.....	125
81.1. Description.....	125
81.2. Status Register (SREG) and Boolean Formula.....	125
82. MULS – Multiply Signed.....	126
82.1. Description.....	126
82.2. Status Register (SREG) and Boolean Formula.....	126
83. MULSU – Multiply Signed with Unsigned.....	127
83.1. Description.....	127

Atmel

Atmel AVR Instruction Set Manual [OTHER]  
Atmel-0856 AVR-Instruction-Set-Manual\_Other-11/2016

[https://www.microchip.com/webdoc/avr assembler/avr assembler.wb\\_instructions.html](https://www.microchip.com/webdoc/avr assembler/avr assembler.wb_instructions.html)



# AVR Assembler : I/O Registers

## 1. I/O Registers

### 1.1. RAMPX, RAMPY, and RAMPZ

Registers concatenated with the X-, Y-, and Z-registers enabling indirect addressing of the whole data space on MCUs with more than 64KB data space, and constant data fetch on MCUs with more than 64KB program space.

### 1.2. RAMPD

Register concatenated with the Z-register enabling direct addressing of the whole data space on MCUs with more than 64KB data space.

### 1.3. EIND

Register concatenated with the Z-register enabling indirect jump and call to the whole program space on MCUs with more than 64K words (128KB) program space.

### 1.4. Stack

STACK      Stack for return address and pushed registers  
SP          Stack Pointer to STACK

### 1.5. Flags

- $\Rightarrow$  Flag affected by instruction
- 0 Flag cleared by instruction
- 1 Flag set by instruction
- Flag not affected by instruction

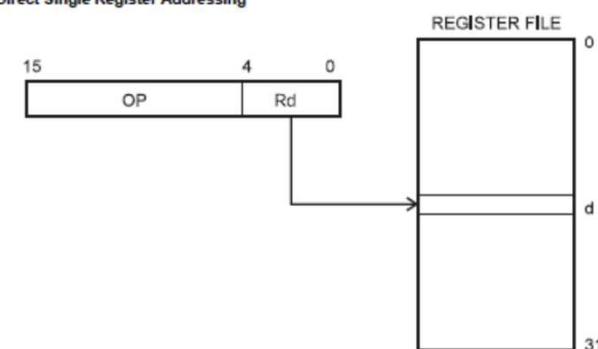
## 2. The Program and Data Addressing Modes

The AVR® Enhanced RISC microcontroller supports powerful and efficient addressing modes for access to the Program memory (Flash) and Data memory (SRAM, Register file, I/O Memory, and Extended I/O Memory). This chapter describes the various addressing modes supported by the AVR architecture. In the following figures, OP means the operation code part of the instruction word. To simplify, not all figures show the exact location of the addressing bits. To generalize, the abstract terms RAMEND and FLASHEND have been used to represent the highest location in data and program space, respectively.

**Note:** Not all addressing modes are present in all devices. Refer to the device specific instruction summary.

### 2.1. Register Direct, Single Register Rd

Figure 2-1. Direct Single Register Addressing



The operand is contained in register d (Rd).

# AVR Assembler : Instructions & Directives

The screenshot shows the AVR Assembler Instruction mnemonics page. The left sidebar contains a navigation menu with links to Documentation Home, AVR Assembler, AVR Assembler Known Issues, AVR Assembler Command Line Options, AVR Assembler source, AVR Assembler Syntax, AVR Assembler directives, AVR Assembler Preprocessor, Expressions, Instruction mnemonics (Arithmetic and logic instructions, Branch Instructions, Data Transfer Instructions, Bit and Bit-test Instructions, I/O Registers, Instruction Set Nomenclature, Instructions), and a search bar. The main content area is titled "Data Transfer Instructions" and lists various AVR assembly instructions with their mnemonics, operands, descriptions, operations, flags, and cycles. The "SIDEBAR" button is visible at the top right of the content area.

Mnemonic	Operands	Description	Operation	Flags	Cycles
MOV	Rd,Rr	Copy register	Rd = Rr	None	1
MOVW	Rd,Rr	Copy register pair	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Load Immediate	Rd = K	None	1
LDS	Rd,k	Load Direct	Rd = (k)	None	2*
LD	Rd,X,Y,Z	Load Indirect	Rd = (X)	None	2*
LD	Rd,X,Y,Z	Load Indirect and Post-Increment	Rd = (X), X=X+1	None	2*
LD	Rd,X,Y,Z	Load Indirect and Pre-Decrement	X=X-1, Rd = (X)	None	2*
LD	Rd,X,Y,Z	Load Indirect	Rd = (Y)	None	2*
LD	Rd,X,Y,Z	Load Indirect and Post-Increment	Rd = (Y), Y=Y+1	None	2*
LD	Rd,X,Y,Z	Load Indirect and Pre-Decrement	Y=Y-1, Rd = (Y)	None	2*
LD	Rd,X,Y,Z+q	Load Indirect with displacement	Rd = (Y+q)	None	2*
LD	Rd,X,Y,Z	Load Indirect	Rd = (Z)	None	2*
LD	Rd,X,Y,Z	Load Indirect and Post-Increment	Rd = (Z), Z=Z+1	None	2*
LD	Rd,X,Y,Z	Load Indirect and Pre-Decrement	Z=Z-1, Rd = (Z)	None	2*
LAC	Rd,X,Y,Z	Load and Clear	Z = Rd •(\$FF-Z)	None	2
LAT	Rd,X,Y,Z	Load and Toggle	Z = Rd ⊕ (Z)	None	2
LAS	Rd,X,Y,Z	Load and Set	Z = Rd v (Z)	None	2
XCH	Rd,X,Y,Z	Exchange	Z = Rd, Rd = Z	None	2

## AVR Assembler Assembler directives

### Assembler directives

Directive	Description
BYTE	<a href="#">Reserve byte(s) to a variable.</a>
CSEG	<a href="#">Code Segment</a>
CSEGSIZE	<a href="#">Program memory size</a>
DB	<a href="#">Define constant byte(s)</a>
DEF	<a href="#">Define a symbolic name on a register</a>
DSEG	<a href="#">Data Segment</a>
DW	<a href="#">Define Constant word(s)</a>
ENDM, ENDMACRO	<a href="#">EndMacro</a>
EQU	<a href="#">Set a symbol equal to an expression</a>
ESEG	<a href="#">EEPROM Segment</a>
EXIT	<a href="#">Exit from file</a>
INCLUDE	<a href="#">Read source from another file</a>
LIST	<a href="#">Turn listfile generation on</a>
LISTMAC	<a href="#">Turn Macro expansion in list file on</a>
MACRO	<a href="#">Begin Macro</a>
NOLIST	<a href="#">Turn listfile generation off</a>
ORG	<a href="#">Set program origin</a>
SET	<a href="#">Set a symbol to an expression</a>
ELSE,ELIF	<a href="#">Conditional assembly</a>

# AVR instructions

Table 4-2. Arithmetic and Logic Instructions

Mnemonic	Operands	Description		Op		Flags	#Clocks AVR	#Clocks AVRxm	#Clocks AVRxt	#Clocks AVRrc
ADD	Rd, Rr	Add without Carry	Rd	←	Rd + Rr	Z,C,N,V,S,H	1	1	1	1
ADC	Rd, Rr	Add with Carry	Rd	←	Rd + Rr + C	Z,C,N,V,S,H	1	1	1	1
ADIW	Rd, K	Add Immediate to Word	Rd	←	Rd + 1:Rd + K	Z,C,N,V,S	2	2	2	N/A
SUB	Rd, Rr	Subtract without Carry	Rd	←	Rd - Rr	Z,C,N,V,S,H	1	1	1	1
SUBI	Rd, K	Subtract Immediate	Rd	←	Rd - K	Z,C,N,V,S,H	1	1	1	1
SBC	Rd, Rr	Subtract with Carry	Rd	←	Rd - Rr - C	Z,C,N,V,S,H	1	1	1	1
SBCI	Rd, K	Subtract Immediate with Carry	Rd	←	Rd - K - C	Z,C,N,V,S,H	1	1	1	1
SBIW	Rd, K	Subtract Immediate from Word	Rd + 1:Rd	←	Rd + 1:Rd - K	Z,C,N,V,S	2	2	2	N/A
AND	Rd, Rr	Logical AND	Rd	←	Rd • Rr	Z,N,V,S	1	1	1	1

## Instruction Set Nomenclature

### Status Register (SREG)

**SREG** Status Register  
**C** Carry Flag  
**Z** Zero Flag  
**N** Negative Flag  
**V** Two's complement overflow indicator  
**S** N or V, for signed tests  
**H** Half Carry Flag  
**T** Transfer bit used by BLD and BST instructions  
**I** Global Interrupt Enable/Disable Flag

### Registers and Operands

**Rd:** Destination (and source) register in the Register File  
**Rr:** Source register in the Register File  
**R:** Result after instruction is executed  
**K:** Constant data  
**k:** Constant address  
**b:** Bit in the Register File or I/O Register (3-bit)  
**s:** Bit in the Status Register (3-bit)  
**X,Y,Z:** Indirect Address Register (X=R27:R26, Y=R29:R28, and Z=R31:R30)  
**A:** I/O location address  
**q:** Displacement for direct addressing (6-bit)

## 5. ADC – Add with Carry

### 5.1. Description

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

Operation:

(i)  $Rd \leftarrow Rd + Rr + C$

Syntax:

(i) ADC Rd,Rr      Operands:  $0 \leq d \leq 31, 0 \leq r \leq 31$

Program Counter:

$PC \leftarrow PC + 1$

16-bit Opcode:

0001	11rd	dddd	rrr
------	------	------	-----

### 5.2. Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
-	-	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$	$\Rightarrow$

**H**  $Rd3 \cdot Rr3 + Rr3 \cdot R3 + R3 \cdot Rd3$

Set if there was a carry from bit 3; cleared otherwise.

**S**  $N \oplus V$ , for signed tests.

**V**  $Rd7 \cdot Rr7 + R7 \cdot Rd7 + R7 \cdot R7$

Set if two's complement overflow resulted from the operation; cleared otherwise.

**N**  $R7$

Set if MSB of the result is set; cleared otherwise.

**Z**  $R7 \cdot R6 \cdot R5 \cdot R4 \cdot R3 \cdot R2 \cdot R1 \cdot R0$

Set if the result is \$00; cleared otherwise.

**C**  $Rd7 \cdot Rr7 + Rr7 \cdot R7 + R7 \cdot Rd7$

Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add r1,r0      ; Add R1:R0 to R2:R2
add r2,r0      ; Add low byte
add r3,r1      ; Add with carry high byte
```

Words

1 (2 bytes)

## 73. LDI – Load Immediate

### 73.1. Description

Loads an 8-bit constant directly to register 16 to 31.

Operation:

(i)  $Rd \leftarrow K$

Syntax:

(i) LDI Rd,K      Operands:  $16 \leq d \leq 31, 0 \leq K \leq 255$

Program Counter:

16-bit Opcode:

1110	KKKK	dddd	KKKK
------	------	------	------

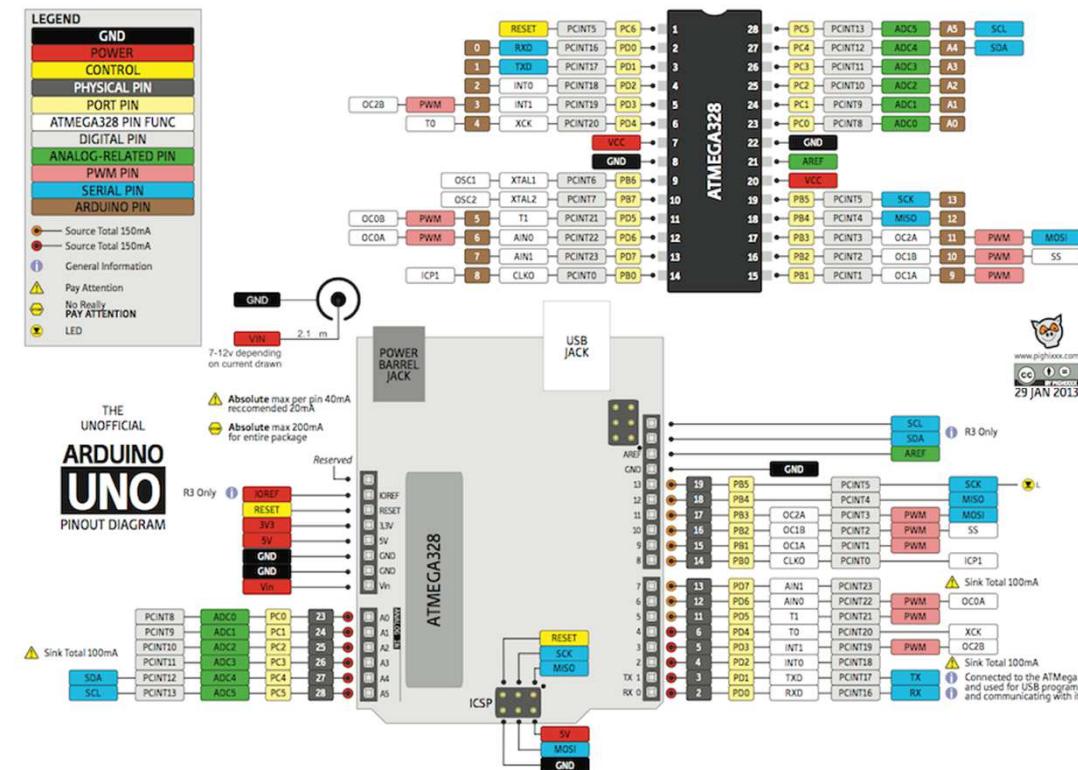


# LA CARTE ARDUINO « UNO »

The Arduino "Uno" is a microcontroller based on the ATmega328 which features:

- of 14 digital pins of I/o (of which 6 output PWM (pulse modulated width)),
- of 6 analog inputs (which can also be used in pins entrees/sorties digital),
- of a 16 MHz quartz
- a USB connection,
- of a power jack connector,
- a connector CPSI (programming "in-circuit"), cf. <http://arduino.cc/en/Hacking/Programmer>
- and a reset button (reset).

It contains all that is necessary for the operation of the microcontroller; To use it, simply connect it to a PC by USB (or with a power adapter or a battery).



# ARDUINO « UNO » BOARD

Microcontroller	ATmega328
Working voltage	5V
Power supply voltage (recommended)	7-12V
Power supply voltage (limits)	6-20V
I/O digital pins	14 (among 6 with PWM output)
Broches d'entrées analogiques	6 (used as I/O digital)
Maximum current available per I/O pin (5V)	40 mA ( <b>ATTENTION : 200mA cumulé pour l'ensemble des broches E/S</b> )
Maximum current available per I/O pin (3,3V)	50 mA
Maximum current available for 5V output	Depending on power supply - 500 mA max if USB port is used alone
Flash programmable memory	32 KB (ATmega328) with 0.5 KB used for bootloader
SRAM memory (volatile)	2 KB (ATmega328)
EEPROM memory (non volatile)	1 KB (ATmega328)
Clock frequency	16 MHz

## Analog Inputs/Outputs :

Arduino cards have 6 analog inputs (A0 to A5), each of which can provide a measure of resolution 10-bit (i.e. on 1024 levels) using the function `analogRead()` of the Arduino language.

By default, these pins measure between the (value 0) 0V and 5V (value 1023), but it is possible to change the upper measuring range reference using the AREF pin and the `analogReference()` Arduino language instruction.

Specific analog pins: A4 (SDA) and A5 (SCL). Support protocol I2C communications (or interface TWI (Two Wire Interface - Interface "2-wire"), available by using the Wire/I2C library.)

Note: the analog pins can also be used as digital pins: in this case, they are numbered as digital pin 14 to 19.

# ARDUINO « UNO » BOARD

## Digital INPUTS and Outputs :

The 14 digital pins of the UNO (D0 to D13) card are used in 5V either as a digital input, digital output, using pinMode(), digitalWrite(), and digitalRead() of the Arduino language instructions.

Each pin can provide or receive up to 40mA intensity and has a "reminder at most" internal resistance (pull up) (disconnected by default) of 20-50 KOhms. This internal resistance is activated on a PIN entry using the statement digitalWrite (PIN, HIGH).

In addition, some pins have specialized functions:

- Communication series: pins D0 (RX) and D1 (TX). Used for receive (RX) and transmit (TX) data series of TTL level.
- External interrupts: Pin D2 and D3. These pins can be configured to trigger an interrupt to a low value, a front rising or falling, or a change in value, see attachInterrupt()...
- Pulse PWM (pulse modulated width): pins 3, 5, 6, 9, 10, and 11. Provide a 8 - bit PWM pulse using the analogWrite() statement.
- SPI (serial device Interface): Pin D10 (SS), D11 (MOSI), D12 D13 (SCK) (MISO). Support communication SPI (Interface Series device) available with the library for communication SPI. The SPI pins are also connected on the ICSP connector.
- LED: Pin D13. A LED is included in the map is connected to pin 13. When PIN is at the top level, the LED is lit, when the PIN is at the bottom level, the LED is off.

Other pins:

- AREF: reference to (if different from the 5V) analog input voltage. Used with analogReference() instruction.
- RESET: Put this PIN to the bottom level causes the reset of the microcontroller. Typically, this PIN is used to add a reset button on the circuit which blocks one present on the map.

## IV-4 ET DANS LE FUTUR ?

### IV-4 ARDUINO FAMILY

# ARDUINO FAMILY ...

## Entry Level

Get started with Arduino using Entry Level products: easy to use and ready to power your first creative projects. These boards and modules are the best to start learning and tinkering with electronics and coding. The StarterKit includes a book with 15 tutorials that will walk you through the basics up to complex projects.



## Internet of Things

Make connected devices easily with one of these IoT products and open your creativity with the opportunities of the world wide web.



## LORA

The image shows a product page for the MKR WAN 1300 board. At the top, the product name "MKR WAN 1300" and a price of "€35.00" (VAT excluded) are displayed. Below this is a "Quantity" selector set to "1" and a "BUY NOW" button. Further down is a "PRE-ORDER" button and a link "Want to learn more?". On the right, there's a "1300" identifier. The main image shows the MKR WAN 1300 board. Below the board are several small colored squares representing different product categories: LoRa (red), 32-bit (light blue), ARM (teal), 8-bit (green), Microcontroller (yellow), WiFi (purple), Standard (grey), and Regular (light grey). At the bottom, a note says "NOW AVAILABLE FOR PRE-ORDER. ESTIMATED SHIPPING: NOVEMBER 15th". A detailed description follows: "MKR WAN 1300 is a powerful board that combines the functionality of the MKR Zero and LoRa connectivity. It is the ideal solution for makers wanting to design IoT projects with minimal previous experience in networking."

# ARDUINO FAMILY !

ENTRY LEVEL	UNO   LEONARDO   101   ESPLORA   MICRO   NANO   MINI   MKR2UNO ADAPTER
	STARTER KIT   LCD SCREEN
ENHANCED FEATURES	MEGA   ZERO   DUE   MEGA ADK   M0   M0 PRO   MKR ZERO   MOTOR SHIELD  USB HOST SHIELD   PROTO SHIELD   MKR PROTO SHIELD   4 RELAYS SHIELD  MEGA PROTO SHIELD   MKR RELAY PROTO SHIELD   ISP   USB2SERIAL MICRO  USB2SERIAL CONVERTER
INTERNET OF THINGS	YÚN   ETHERNET   TIAN   INDUSTRIAL 101   LEONARDO ETH   MKR FOX 1200   MKR1000  YUN MINI   YÚN SHIELD   WIRELESS SD SHIELD   WIRELESS PROTO SHIELD   ETHERNET SHIELD V2  GSM SHIELD V2   MKR IoT BUNDLE
EDUCATION	CTC 101
WEARABLE	GEMMA   LILYPAD ARDUINO USB   LILYPAD ARDUINO MAIN BOARD   LILYPAD ARDUINO SIMPLE  LILYPAD ARDUINO SIMPLE SNAP
3D PRINTING	MATERIA 101

BOARDS

MODULES

SHIELDS

KITS

ACCESSORIES

COMING NEXT

# ARDUINO FAMILY !

Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
101	Intel® Curie	3.3 V / 7-12V	32MHz	6/0	14/4	-	24	196	Regular	-
Gemma	ATtiny85	3.3 V / 4-16 V	8 MHz	1/0	3/2	0.5	0.5	8	Micro	0
LilyPad	ATmega168V ATmega328P	2.7-5.5 V / 2.7-5.5 V	8MHz	6/0	14/6	0.512	1	16	-	-
LilyPad SimpleSnap	ATmega328P	2.7-5.5 V / 2.7-5.5 V	8 MHz	4/0	9/4	1	2	32	-	-
LilyPad USB	ATmega32U4	3.3 V / 3.8-5 V	8 MHz	4/0	9/4	1	2.5	32	Micro	-
Mega 2560	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Micro	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
MKR1000	SAMD21 Cortex-M0+	3.3 V / 5V	48MHz	7/1	8/4	-	32	256	Micro	1
Pro	ATmega168 ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	0.512	1 2	16 32	-	1
Pro Mini	ATmega328P	3.3 V / 3.35-12 V 5 V / 5-12 V	8 MHz 16 MHz	6/0	14/6	1	2	32	-	1
Uno	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/6	1	2	32	Regular	1
Zero	ATSAMD21G18	3.3 V / 7-12 V	48 MHz	6/1	14/10	-	32	256	2 Micro	2
Due	ATSAM3X8E	3.3 V / 7-12 V	84 MHz	12/2	54/12	-	96	512	2 Micro	4
Explora	ATmega32U4	5 V / 7-12 V	16 MHz	-	-	1	2.5	32	Micro	-
Ethernet	ATmega328P	5 V / 7-12 V	16 MHz	6/0	14/4	1	2	32	Regular	-

Name	Processor	Operating/Input Voltage	CPU Speed	Analog In/Out	Digital IO/PWM	EEPROM [kB]	SRAM [kB]	Flash [kB]	USB	UART
Leonardo	ATmega32U4	5 V / 7-12 V	16 MHz	12/0	20/7	1	2.5	32	Micro	1
Mega ADK	ATmega2560	5 V / 7-12 V	16 MHz	16/0	54/15	4	8	256	Regular	4
Mini	ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	1	2	32	-	-
Nano	ATmega168 ATmega328P	5 V / 7-9 V	16 MHz	8/0	14/6	0.512 1	1 2	16 32	Mini	1
Yún	ATmega32U4 AR9331 Linux	5 V	16 MHz 400MHz	12/0	20/7	1	2.5 16MB	32 64MB	Micro	1
Arduino Robot	ATmega32u4	5 V	16 MHz	6/0	20/6	1 KB (ATmega32u4)/ 512 Kbit (I2C)	2.5 KB (ATmega32u4)	1	1	32 KB (ATmega32u4) of which 4 KB used by bootloader
MKRZero	SAMD21 Cortex-M0+ 32bit low power ARM MCU	3.3 V	48 MHz	7 (ADC 8/10/12 bit)/1 (DAC 10 bit)	22/12	No	32 KB	256 KB	1	1



## IV-4 ET DANS LE FUTUR ?

IV-4 OTHER INTERESTING  
TRACKS ?

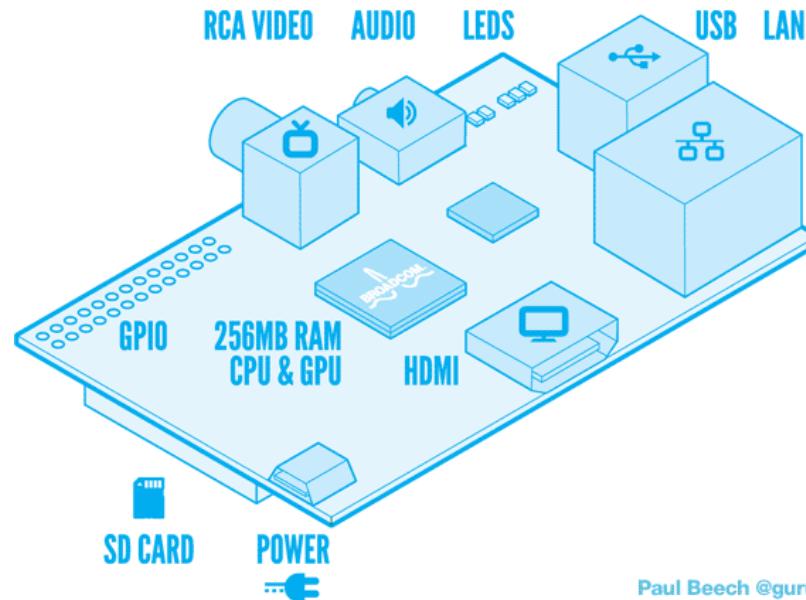
## IV-2 OTHER INTERESTING TRACKS!: LES MICROCHIP

Quand Microchip se réveille? Il fait des 32bits compatibles avec l'Arduino (8bits) et au même prix !!!!

 <p><b>chipKIT Uno32™ Arduino™-Compatible Prototyping Platform</b></p> <p>\$26.95 <a href="#">Add to Cart</a> <small>Shipping immediately</small></p> <p><b>IC:</b> Microchip® PIC32MX320F128</p> <p><b>Programming:</b> Application development using an environment based on the original Arduino™ IDE modified to support PIC32 that also still supports the original Arduino™ line. Leverages <a href="#">existing code examples, tutorials and resources</a>.</p> <p>To download the IDE, please visit: <a href="https://github.com/chipKIT32/chipKIT32-MAX/downloads">https://github.com/chipKIT32/chipKIT32-MAX/downloads</a>.</p> <ul style="list-style-type: none"> <li>Microchip® PIC32MX320F128 processor           <ul style="list-style-type: none"> <li>80 Mhz 32-bit MIPS</li> <li>128K Flash, 16K SRAM</li> </ul> </li> <li>Compatible with existing Arduino™ code examples, reference materials and other resources</li> <li>Can also be programmed using Microchip's MPLAB® IDE (along with a PICkit 3 and our PICkit3 Programming Cable Kit, seen below)</li> <li>Arduino™ "Uno" form factor</li> <li>Compatible with Arduino™ shields</li> <li>42 available I/O</li> <li>User LED</li> <li>Connects to a PC using a USB A-&gt; mini B cable (not included)</li> </ul> <p>The chipKIT™ Uno32™ combines compatibility with the popular Arduino™ open source hardware prototyping platform with the performance of the Microchip PIC32 microcontroller. The Uno32 is the same form factor as the Arduino™ Uno board and is compatible with Arduino™ shields. It features a USB serial port interface for connection to the IDE and can be powered via USB or an external power supply.</p> <p>The Uno32 board takes advantage of the powerful PIC32MX320F128 microcontroller. This microcontroller features a 32-bit MIPS processor core running at 80Mhz, 128K of flash program memory and 16K of SRAM data memory.</p> <p>The Uno32 can be programmed using an environment based on the original Arduino™ IDE modified to support PIC32. In addition, the Uno32 is fully compatible with the advanced Microchip MPLAB® IDE and the PICkit3 in-system programmer/debugger.</p> <p>For additional platform-specific support for your chipKIT, please visit: <a href="http://www.chipkit.org/forum/">http://www.chipkit.org/forum/</a>.</p>	 <p><b>chipKIT Max32™ Arduino™-Compatible Prototyping Platform</b></p> <p>\$49.50 <a href="#">Add to Cart</a> <small>Shipping immediately</small></p> <p><b>IC:</b> Microchip® PIC32MX795F512</p> <p><b>Programming:</b> Application development using an environment based on the original Arduino™ IDE modified to support PIC32 that also still supports the original Arduino™ line. Leverages <a href="#">existing code examples, tutorials and resources</a>.</p> <p>To download the IDE, please visit: <a href="https://github.com/chipKIT32/chipKIT32-MAX/downloads">https://github.com/chipKIT32/chipKIT32-MAX/downloads</a>.</p> <ul style="list-style-type: none"> <li>Microchip® PIC32MX795F512 processor           <ul style="list-style-type: none"> <li>80 Mhz 32-bit MIPS</li> <li>512K Flash, 128K RAM</li> <li>USB 2.0 OTG controller</li> <li>10/100 Ethernet MAC</li> <li>Dual CAN controllers</li> </ul> </li> <li>Provides additional memory and advanced communications peripherals</li> <li>Compatible with existing Arduino code examples, reference materials and other resources</li> <li>Can also be programmed using Microchip's MPLAB® IDE (along with a PICkit 3 and our PICkit3 Programming Cable Kit, seen below)</li> <li>Arduino™ "Mega" form factor</li> <li>Compatible with Arduino™ shields</li> <li>83 available I/O</li> <li>User LED</li> <li>Connects to a PC using a USB A-&gt; mini B cable (not included)</li> </ul> <p>The chipKIT™ Max32™ combines compatibility with the popular Arduino™ open source hardware prototyping platform with the performance of the Microchip PIC32 microcontroller. The Max32 is the same form factor as the Arduino Mega board and is compatible with standard Arduino™ shields as well as larger shields for use with the Mega boards. It features a USB serial port interface for connection to the IDE and can be powered via USB or an external power supply.</p> <p>The Max32 board takes advantage of the powerful PIC32MX795F512 microcontroller. This microcontroller features a 32-bit MIPS processor core running at 80Mhz, 512K of flash program memory and 128K of SRAM data memory. In addition, the processor provides a USB 2 OTG controller, 10/100 Ethernet MAC and dual CAN controllers that can be accessed via add-on I/O shields.</p> <p>The Max32 can be programmed using an environment based on the original Arduino™ IDE modified to support PIC32. In addition, the Max32 is fully compatible with the advanced Microchip MPLAB® IDE and the PICkit3 in-system programmer/debugger.</p> <p>For additional platform-specific support for your chipKIT, please visit: <a href="http://www.chipkit.org/forum/">http://www.chipkit.org/forum/</a>.</p>
--	--

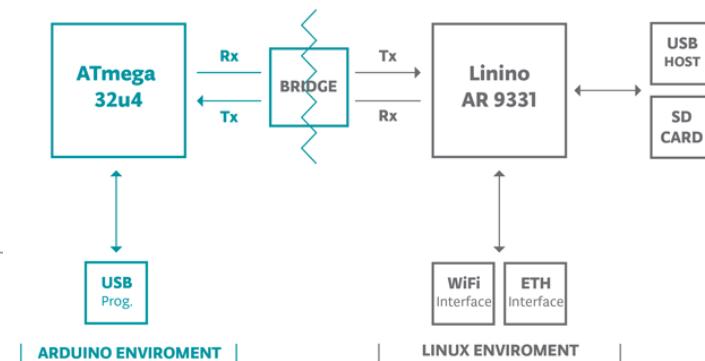
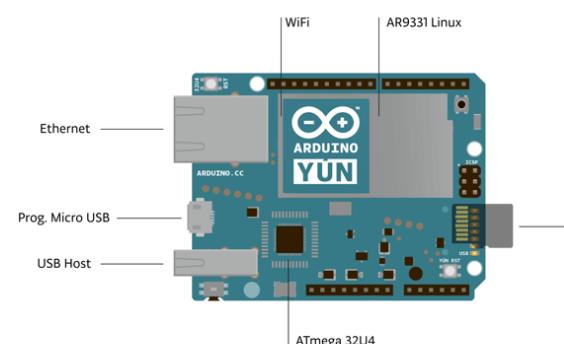


## IV-2 OTHER INTERESTING TRACKS! : LE RASPBERRY PI

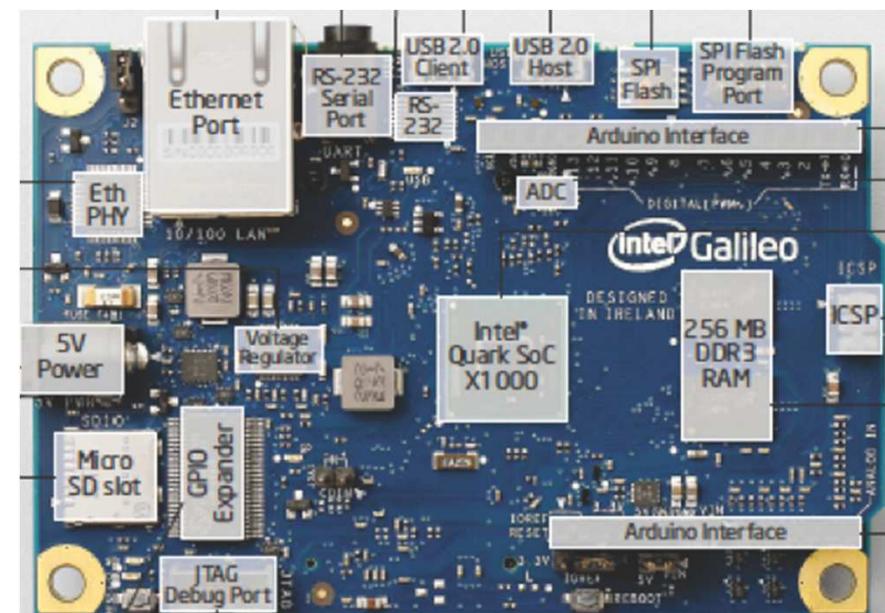


An ARM GNU/Linux box for \$25. Take a byte!

Mais attention à l'arrivée de l'ARDUINO YUN:



# INTEL GALILEO



# GENUINO 101



## Technical specs

Microcontroller	Intel Curie
Operating Voltage	3.3V (5V tolerant I/O)
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 4 provide PWM output)
PWM Digital I/O Pins	4
Analog Input Pins	6
DC Current per I/O Pin	4 mA
Flash Memory	196 kB
SRAM	24 kB
Clock Speed	32MHz
Features	Bluetooth LE, 6-axis accelerometer/gyro
Length	68.6 mm
Width	53.4 mm

## Overview

3.3V 32-bit 32 MHz ARC Core

A learning and development board that delivers the performance and low-power consumption of the Intel® Curie™ Module with the simplicity of Arduino at an entry-level price.

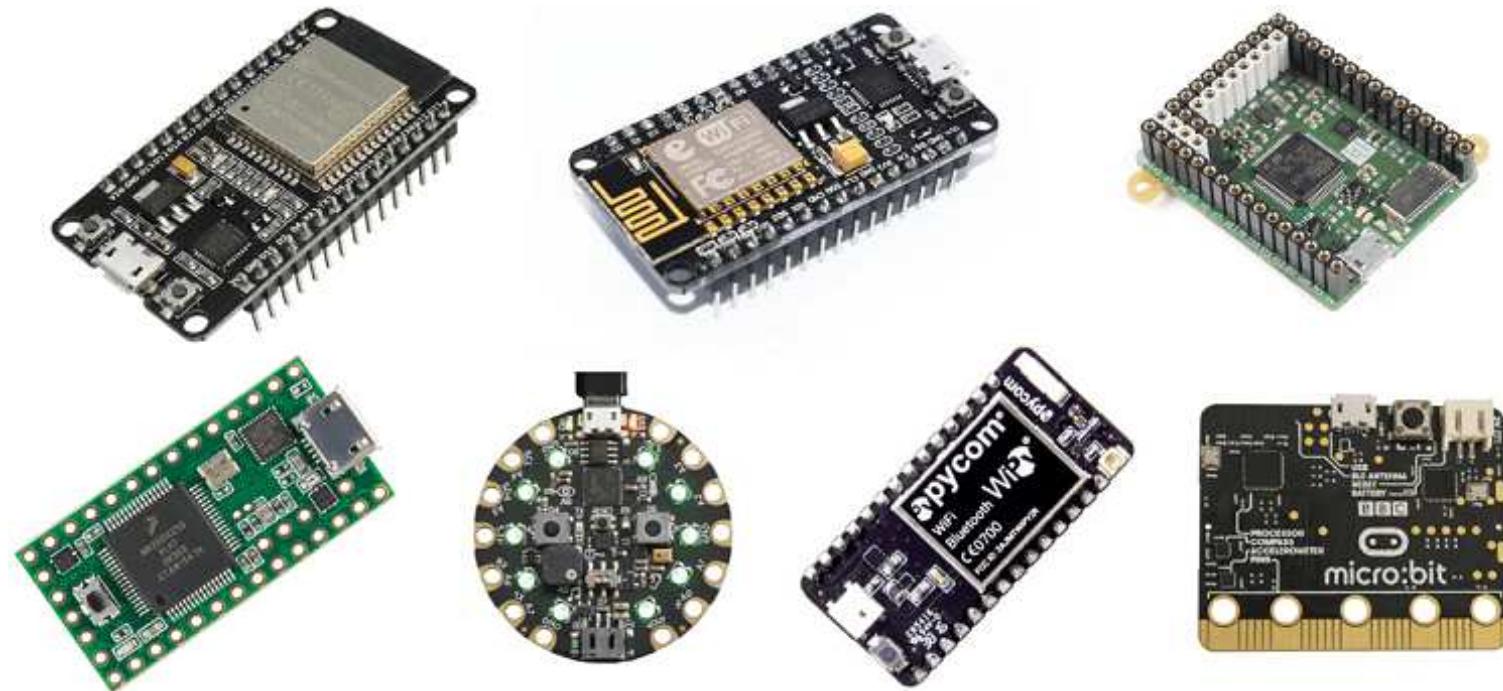
It keeps the same robust form factor and peripheral list of the UNO with the addition of onboard Bluetooth LE capabilities and a 6-axis accelerometer/gyro to help you easily expand your creativity into the connected world.

The module contains two tiny cores, an x86 (Quark) and an ARC, both clocked at 32Mhz. The Quark core runs ViperOS RTOS and helps the Arduino core to accomplish the most demanding tasks. It comes with 14 digital input/output pins (of which 4 can be used as PWM outputs), 6 analog inputs, a USB connector for serial communication and sketch upload, a power jack, an ICSP header with SPI signals and I2C dedicated pins.

The board operating voltage and I/O is 3.3V but all pins are protected against 5V overvoltage.

The Arduino 101 (USA only) and the Genuino 101 (outside USA) has been designed in collaboration with Intel®.

## ESP32 – ESP8266



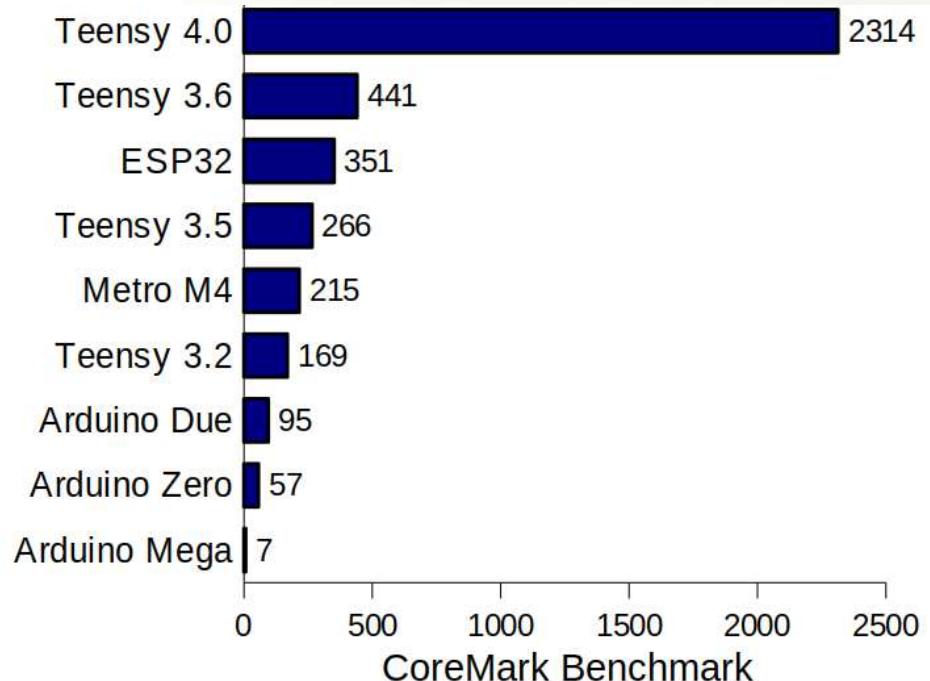
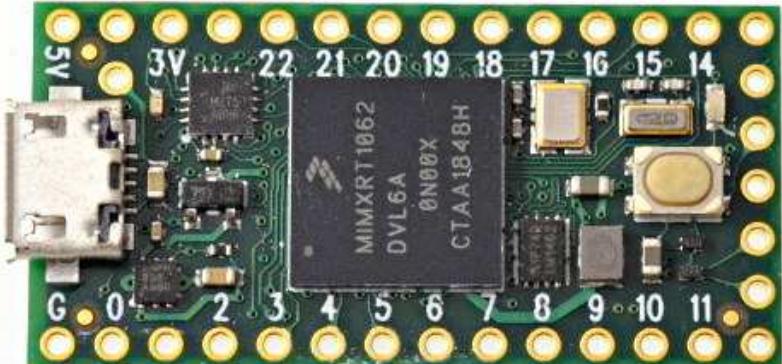
- [\*\*ESP32\*\*](#)
- [\*\*ESP8266\*\*](#)
- PyBoard
- Micro:Bit
- [\*\*Teensy 3.X\*\*](#)
- WiPy – Pycom
- Adafruit Circuit Playground Express
- Other ESP32/ESP8266 based boards

# TEENSY FAMILY

## Teensy 4.0 Development Board

### Technical Specifications

- ARM Cortex-M7 at 600 MHz
- 1024K RAM (512K is tightly coupled)
- 2048K Flash (64K reserved for recovery & EEPROM emulation)
- 2 USB ports, both 480 MBit/sec
- 3 CAN Bus (1 with CAN FD)
- 2 I2S Digital Audio
- 1 S/PDIF Digital Audio
- 1 SDIO (4 bit) native SD
- 3 SPI, all with 16 word FIFO
- 3 I2C, all with 4 byte FIFO
- 7 Serial, all with 4 byte FIFO
- 32 general purpose DMA channels
- 31 PWM pins
- 40 digital pins, all interrupt capable
- 14 analog pins, 2 ADCs on chip
- Cryptographic Acceleration
- Random Number Generator
- RTC for date/time
- Programmable FlexIO
- Pixel Processing Pipeline
- Peripheral cross triggering
- Power On/Off management



<https://www.pjrc.com/store/teensy40.html>

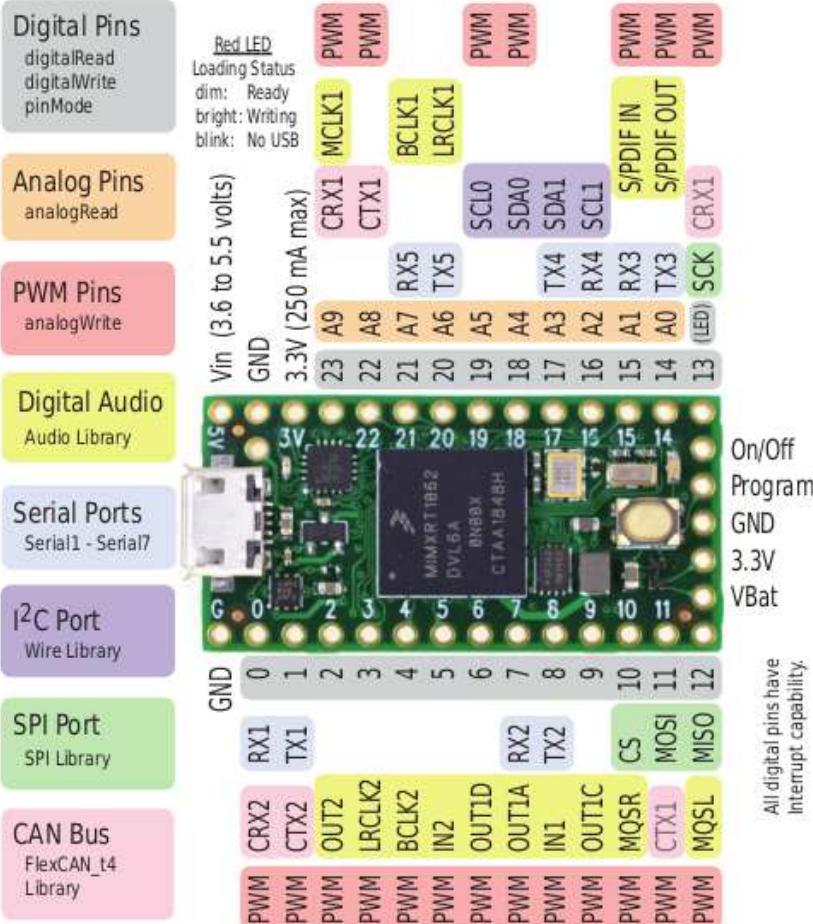
# TEENSY FAMILY

## Welcome to Teensy® 4.0

32 Bit Arduino-Compatible Microcontroller

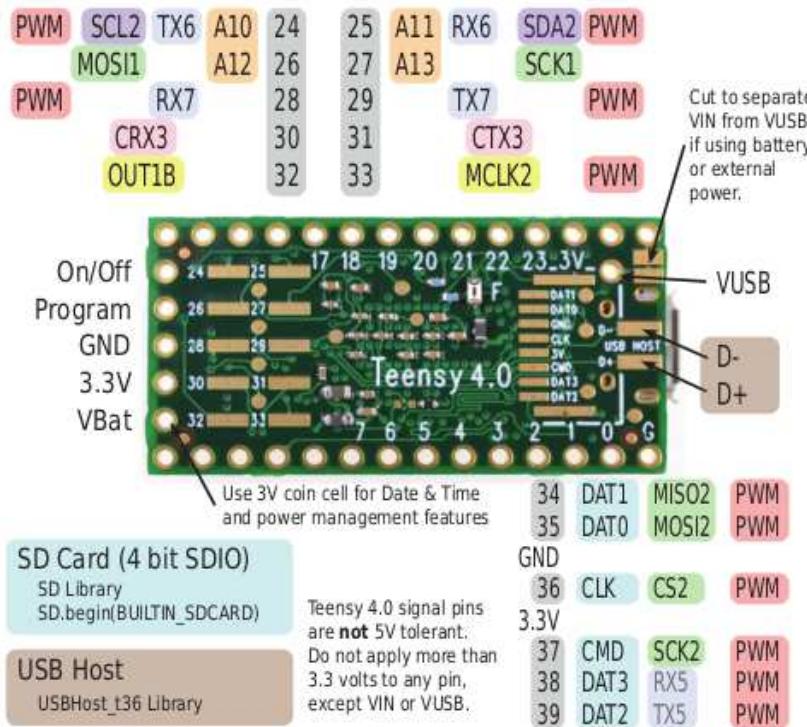
To begin using Teensy, please visit the website & click [Getting Started](#).

[www.pjrc.com/teensy](http://www.pjrc.com/teensy)



## Teensy® 4.0 Back Side

Additional pins and features available on the back side



For solutions to the most common issues and technical support, please visit:

[www.pjrc.com/help](http://www.pjrc.com/help)

Teensy 4.0 System Requirements:  
PC computer with Windows 7, 8, 10 or later  
or Ubuntu Linux 14.04 or later  
or Macintosh OS-X 10.8 or later  
USB Micro-B Cable



# TEENSY FAMILY

Feature	Teensy 2.0	Teensy++ 2.0	Teensy LC	Teensy 3.2	Teensy 3.5	Teensy 3.6	Units
Price	\$16.00	\$24.00	\$11.65	\$19.80	\$24.25	\$29.25	US Dollars
Processor Core	ATMEGA32U4 AVR	AT90USB1286 AVR	MKL26Z64VFT4 Cortex-M0+	MK20DX256VLH7 Cortex-M4	MK64FX512VMD12 Cortex-M4F	MK66FX1M0VMD18 Cortex-M4F	
Rated Speed Overclockable	16	16	48	72	120	180	MHz
	-	-	-	96	-	240	MHz
Flash Memory	31.5	127	62	256	512	1024	kbytes
Bandwidth	32	32	96	192	192	411	Mbytes/sec
Cache	-	-	64	256	256	8192	Bytes
RAM	2.5	8	8	64	256	256	kbytes
EEPROM	1024	4096	128 (emu)	2048	4096	4096	bytes
Direct Memory Access	-	-	4	16	16	32	Channels
Digital I/O	25	46	27	34	58	58	Pins
Breadboard I/O	22	36	24	24	40+2	40+2	Pins
Voltage Output	5V	5V	3.3V / 5V	3.3V	3.3V	3.3V	Volts
Current Output	20mA	20mA	5mA / 20mA	10mA	10mA	10mA	milliAmps
Voltage Input	5V	5V	3.3V Only	5V Tolerant	5V Tolerant	3.3V Only	Volts
Interrupts	4	8	18	34	58	58	Pins
Analog Input Converters	12	8	13	21	27	25	Pins
Usable Resolution	1	1	1	2	2	2	Bits
Prog Gain Amp	10	10	12	13	13	13	
Touch Sensing	1	1	-	2	-	-	Pins
Comparators	-	-	11	12	-	11	
	1	1	1	3	3	4	
Analog Output DAC Resolution	-	-	1	1	2	2	Pins
	-	-	12	12	12	12	Bits
Timers	4 Total	4 Total	7 Total	12 Total	17 Total	19 Total	
PWM, 16 bit	2	2	3	3	4	6	
PWM, 8-10 bit	2	2	-	-	-	-	
Total PWM Outputs	7	9	10	12	20	22	Pins
PDB Type	-	-	-	1	1	1	
CMT Type	-	-	-	1	1	1	
LPTMR Type	-	-	1	1	1	1	
PIT/Interval	-	-	2	4	4	4	
IEEE1588	-	-	-	-	4	4	
Systick	-	-	1	1	1	1	
RTC	-	-	0 **	1 **	1	1	
Communication							
USB	1	1	1	1	1	2	
Serial	1	1	3	3	6	6	
With FIFOs	-	-	-	2	2	2	
High Res Baud SPI	-	-	-	3	6	5	
With FIFOs	1	1	2	1	3	3	
I2C	1	1	2	2	3	4	
CAN Bus	-	-	-	1	1	2	
Digital Audio	-	-	1	2	2	2	
SD Card	-	-	-	-	1	1	
Ethernet	-	-	-	-	1	1	

# GITHUB

# GITHUB

- Vous écrivez un peu de code et souhaitez collaborer avec d'autres développeurs ?
- Vous voulez toujours savoir pourquoi une modification a été faite et ne plus risquer de perdre des jours de travail ?
- Vous avez clairement besoin de versionner votre code !

Un logiciel de gestion de versions est un outil incontournable pour tout développeur. Il en existe de nombreux, et dans ce cours vous découvrirez Git, le logiciel créé par Linus Torvald, auteur du Kernel Linux.

Vous découvrirez ce qu'est la gestion de version et les avantages que cela apporte.

Puis, vous plongerez dans l'univers de Git : commit, branches, merge... tous ces termes n'auront plus de secrets pour vous !

Enfin vous apprendrez à utiliser GitHub pour héberger votre code et collaborer facilement sur des projets open-source.

A screenshot of a GitHub repository page. At the top, there's a navigation bar with links for "Why GitHub?", "Enterprise", "Explore", "Marketplace", and "Pricing". On the right side of the bar are "Search", "Sign in", and "Sign up" buttons. Below the bar, the repository name "etalab / calculette-impots-exemples" is displayed, along with statistics: 8 Watchers, 0 Stars, and 1 Fork. A prominent "Join GitHub today" call-to-action button is visible, along with a message about GitHub's mission to host and review code. Below this, there's a section titled "Exemples d'utilisations de la calculatrice de l'impôt sur le revenu" showing a list of commits. The commits are listed in a table with columns for author, file, description, and date. The commits are as follows:

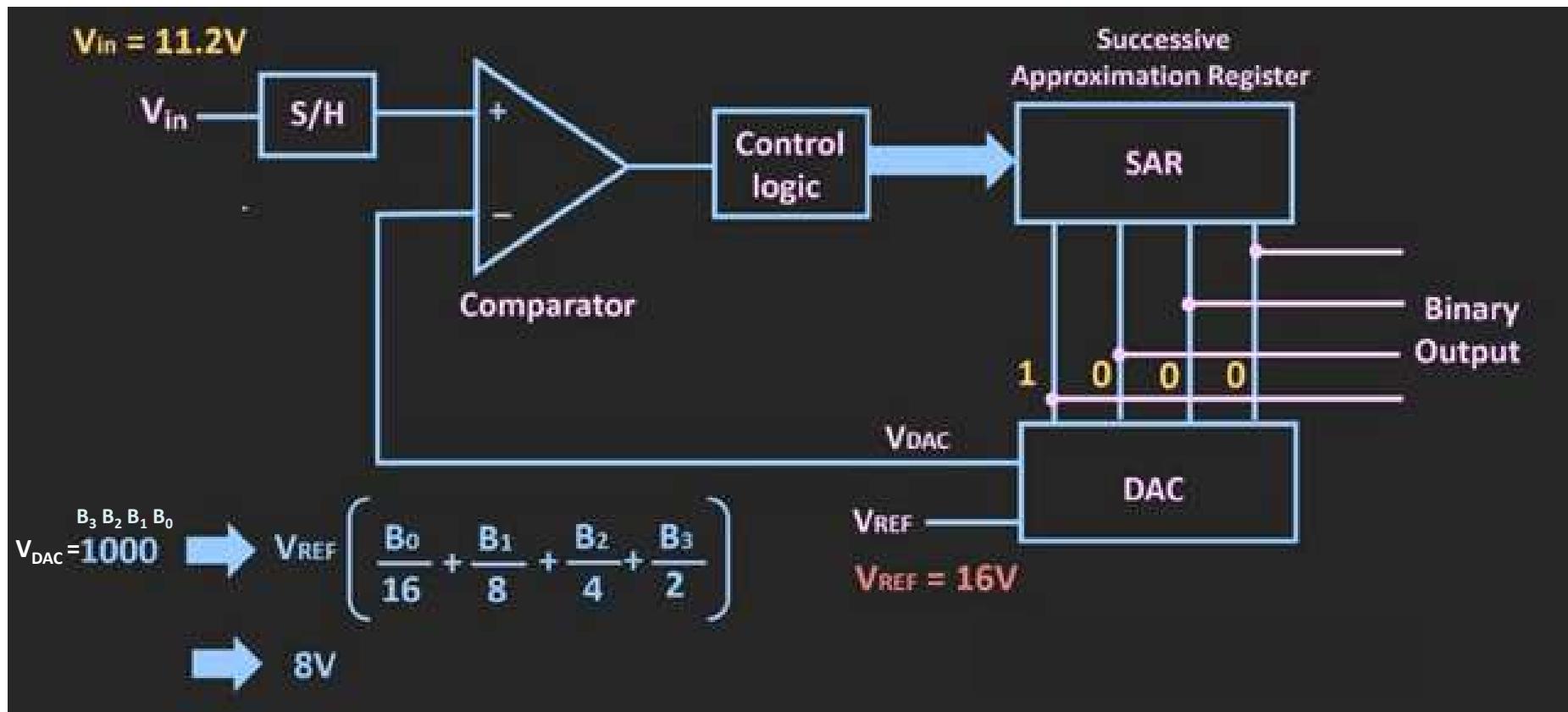
Author	File	Description	Date
Michel Blancard	Documentation	Documentation et licences	Latest commit 2c1c66f on 14 Sep 2017
	calculette_impots_exemples	Merge branch 'hackathon_code_impot'	2 years ago
	graph	precompute graph on uncommon input variables	4 years ago
	json	Met à jour les notebooks	2 years ago
	notebooks	Complète .gitignore	2 years ago
	.gitignore	Documentation et licences	2 years ago
	LICENCE_CeCILL_v2-1	Documentation et licences	2 years ago
	LICENSE_MIT	Documentation et licences	2 years ago
	README.md	Documentation	2 years ago
	setup.py	Refactore le code	2 years ago

# RAPPEL

## SUCCESSIVE APPROXIMATION ADC

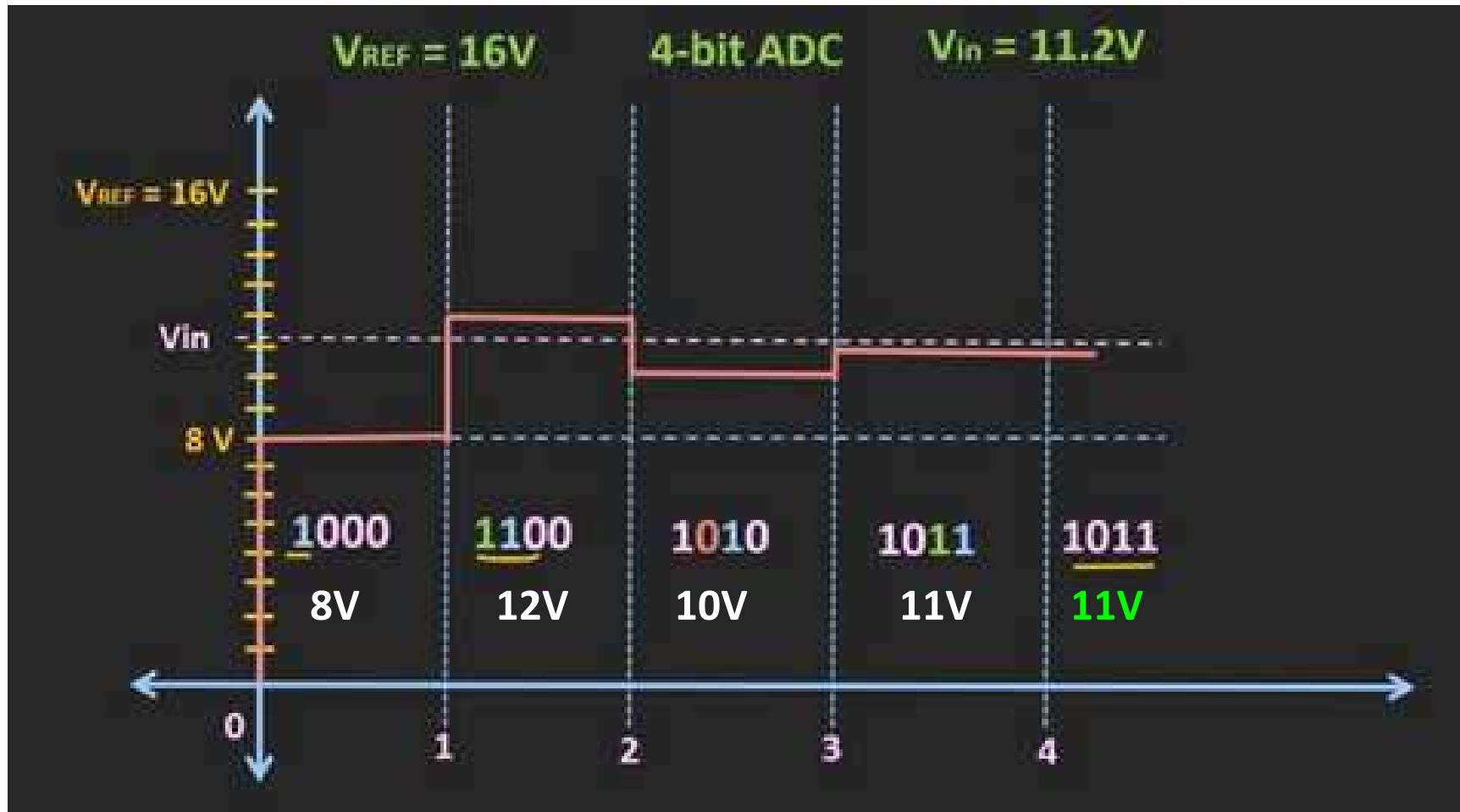
# SUCCESSIVE APPROXIMATION ADC - CIRCUIT

Example with a 4bit ADC:



We put the MSB to 0 ( $B_3=1$ ) (in the middle of  $V_{REF}-V_{GND}$ )  
and the performs successive approximation by using a comparator.

## SUCCESSIVE APPROXIMATION ADC – USE CASE



1.  $V_{in}=11.2 > V_{DAC}=8V$   
 $\Rightarrow 1000 \Rightarrow 1100 = 12V$

2.  $V_{in}=11.2 < V_{DAC}= 12V$   
 $\Rightarrow 1100 \Rightarrow 1010 = 10V$

3.  $V_{in}=11.2 > V_{DAC}= 10V$   
 $\Rightarrow 1010 \Rightarrow 1011 = 11V$

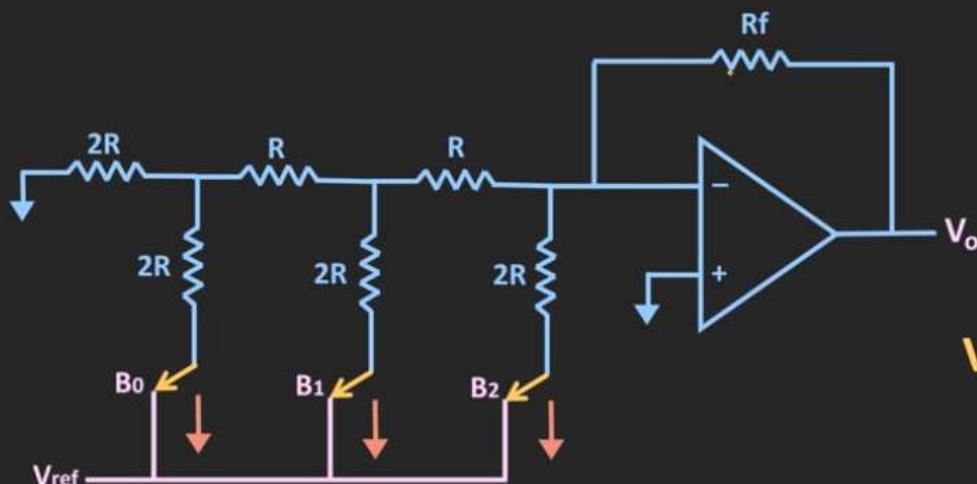
4.  $V_{in}=11.2 > V_{DAC}= 11V$   
 $\Rightarrow 1011$  holds at 1011

*Cause resolution =  $16V/2^4 = 1V$   
And  $V_{in}$  is in between 12V and 11V*

<https://www.youtube.com/watch?v=h0CGtr4SC9s>

## R-2R LADDER DAC

### R-2R ladder DAC



$$V_o = \left( -\frac{V_{ref} \times R_f}{R} \right) \sum_{i=0}^{i=N-1} \frac{B_i}{2^{(N-i)}}$$

$$V_{ref} \left( \frac{-R_f}{R} \right) \left( \frac{B_0}{2^n} + \frac{B_1}{2^{n-1}} + \frac{B_2}{2^{n-2}} + \dots + \frac{B_{n-2}}{2^2} + \frac{B_{n-1}}{2^1} \right)$$

# **COMMANDES LINUX**

## **LES PLUS UTILES**

# COMMANDES LES PLUS UTILES SUR LINUX - RASPBIAN

## Les 42 commandes les plus utiles sur Raspberry PI

Si vous souhaitez connaître les commandes, il vous suffira de rentre la commande suivie de **-help**.

### Commandes générales

- **apt-get update** : Met à jour votre version de Raspbian.
- **apt-get upgrade** : Met à niveau tous les logiciels que vous avez installés.
- **clear** : Efface l'écran du terminal des commandes et du texte précédemment exécutés.
- **date** : Affiche la date actuelle.
- **find / -name exemple.txt** : Recherche dans le système entier le fichier exemple.txt et génère une liste de tous les répertoires qui contiennent le fichier.
- **nano example.txt** : Ouvre le fichier example.txt dans "Nano", l'éditeur de texte de [linux](#).
- **poweroff** : Arrêt immédiat du [raspberry](#) PI.
- **raspi-config** : Ouvre le menu des paramètres de configuration.
- **reboot** : Pour rebooter immédiatement le Raspberry PI.
- **shutdown -h now** : Pour éteindre immédiatement le Raspberry PI.
- **shutdown -h 01:22** : Pour éteindre le Raspberry PI à 1:22 AM.
- **startx** : Ouvrir l'interface graphique GUI (Graphical User Interface).

### Commandes sur les fichiers et répertoires

- **cat exemple.txt** : Affiche le contenu du fichier exemple.txt.
- **cd /abc/xyz** : Change de répertoire courant pour le répertoire /abc/xyz.
- **cp XXX** : Copie le fichier ou le répertoire XXX et le colle à un emplacement spécifique. Par exemple: **cp exemplefile.txt /home/pi/office/** copie exemplefile.txt du répertoire courant et le colle dans le répertoire /home/pi/ directory. Si le fichier n'est pas présent dans le répertoire courant, vous pouvez ajouter son chemin en préfixe (par exemple : **cp /home/pi/documents/examplefile.txt /home/pi/office/** copie le fichier du répertoire documents dans le répertoire office).
- **ls -l** : Liste tous les fichiers du répertoire en cours, ainsi que la taille du fichier, la date de modification et les autorisations.
- **mkdir exemple\_directory** : Créer dans le répertoire courant un nouveau répertoire exemple\_directory.
- **mv XXX** : Déplace un fichier ou un répertoire nomé XXX à un emplacement spécifique. Par exemple, **mv exemplefile.txt /home/pi/office/** déplace **exemplefile.txt** dans le répertoire /home/pi/office. Si le fichier n'est pas présent dans le répertoire courant, vous pouvez ajouter son chemin en préfixe (par exemple : **cp /home/pi/documents/examplefile.txt /home/pi/office/** déplace le fichier du répertoire documents dans le répertoire office). Cette commande peut aussi être utilisée pour renommer des fichiers (mais seulement dans le même répertoire). par exemple, **mv examplefile.txt newfile.txt** renomme examplefile.txt en newfile.txt, et conserve le fichier dans le même répertoire.
- **rm example.txt** : Effacer le fichier example.txt.
- **rmdir example\_directory** : Effacer le répertoire example\_directory (seulement si il est vide).
- **scp user@10.0.0.32 :/some/path/file.txt** : Copier un fichier à travers SSH. Peut être utilisé pour télécharger un fichier à partir d'un ordinateur de bureau / ordinateur portable sur le Raspberry PI. user@10.0.0.32 est le nom d'utilisateur et l'adresse IP locale du bureau / ordinateur portable et /some/path/file.txt est le chemin d'accès et le nom du fichier sur le bureau / ordinateur portable.
- **touch** :Crée un nouveau fichier vide dans le répertoire courant.

# COMMANDES LES PLUS UTILES SUR LINUX - RASPBIAN

## Commandes Réseau et Internet:

- **ifconfig** : Pour vérifier l'état de la connexion réseau que vous utilisez (pour voir si wlan0 dispose d'une adresse IP par exemple).
- **iwconfig** : Pour vérifier quel réseau l'adaptateur sans fil utilise par exemple.
- **iwlist wlan0 scan** : Affiche une liste des réseaux sans fil actuellement disponibles sur wlan0.
- **iwlist wlan0 scan | grep ESSID** : Utilisez grep avec le nom d'un champ pour répertorier uniquement les champs dont vous avez besoin (par exemple, pour lister les ESSID uniquement).
- **nmap** : Analyse votre réseau et répertorie les périphériques connectés, le numéro de port, le protocole, le système d'exploitation, l'état (ouvert ou fermé), les adresses MAC et d'autres informations.
- **ping** : Teste la connectivité entre deux périphériques connectés sur un réseau. Par exemple, ping 10.0.0.32 envoie un paquet à l'appareil à IP 10.0.0.32 et attend une réponse. Il fonctionne également avec les adresses de sites Web.
- **wget http://www.website.com/example.txt** : Télécharge le fichier example.txt depuis le Web et l'enregistre dans le répertoire courant.

## Commandes d'informations systèmes:

- **cat /proc/meminfo** : Affiche des détails sur votre mémoire.
- **cat /proc/partitions** : Affiche la taille et le nombre de partitions sur votre carte SD ou votre disque dur.
- **cat /proc/version** : Affiche la version de la Raspberry Pi que vous utilisez.
- **df -h** : Affiche des informations sur l'espace disque disponible.
- **df /** : Indique la quantité d'espace disque disponible.
- **dpkg --get-selections | grep XXX** : Affiche tous les packages installés qui sont liés à XXX.
- **dpkg --get-selections** : Affiche tous les paquets installés.
- **free** : Indique la quantité de mémoire libre disponible.
- **hostname -I** : Affiche l'adresse IP de votre Raspberry Pi.
- **lsusb** : Liste tous les périphériques USB connectés à votre Raspberry Pi.
- **Touche HAUT** : En appuyant sur la touche HAUT, vous entrez la dernière commande entrée dans l'invite de commande. C'est un moyen rapide de corriger les commandes qui ont été faites par erreur.
- **vcgencmd measure\_temp** : Affiche la température de la CPU.
- **vcgencmd get\_mem arm && vcgencmd get\_mem gpu** : Affiche la mémoire divisée entre le processeur et le GPU.

# RASPBERRY PI 4

```
pi@raspberrypi:~ $ pinout
[...]
Revision      : c03111
SoC           : BCM2711
RAM           : 4096Mb
Storage        : MicroSD
USB ports     : 4 (excluding power)
Ethernet ports : 1
Wi-fi          : True
Bluetooth      : True
Camera ports (CSI) : 1
Display ports (DSI) : 1

J8:
  3V3  (1)  (2)  5V
  GPIO2 (3)  (4)  5V
  GPIO3 (5)  (6)  GND
  GPIO4 (7)  (8)  GPIO14
  GND   (9)  (10) GPIO15
  GPIO17 (11) (12) GPIO18
  GPIO27 (13) (14) GND
  GPIO22 (15) (16) GPIO23
  3V3  (17) (18) GPIO24
  GPIO10 (19) (20) GND
  GPIO9  (21) (22) GPIO25
  GPIO11 (23) (24) GPIO8
  GND   (25) (26) GPIO7
  GPIO0  (27) (28) GPIO1
  GPIO5  (29) (30) GND
  GPIO6  (31) (32) GPIO12
  GPIO13 (33) (34) GND
  GPIO19 (35) (36) GPIO16
  GPIO26 (37) (38) GPIO20
  GND   (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
pi@raspberrypi:~ $
```

```
from gpiozero import LED
from time import sleep
```

```
led = LED(17)
```

```
while True:
    led.on()
    sleep(1)
    led.off()
    sleep(1)
```

<https://gpiozero.readthedocs.io/en/stable/installing.html>

```
import RPi.GPIO as GPIO
import time
```

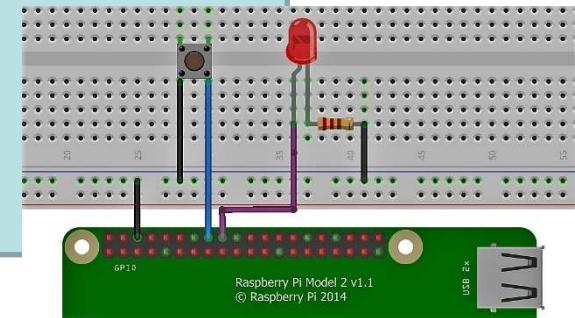
```
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#Button to GPIO23
GPIO.setup(24, GPIO.OUT) #LED to GPIO24
```

try:

```
    while True:
        button_state = GPIO.input(23)
        if button_state == False:
            GPIO.output(24, True)
            print('Button Pressed...')
            time.sleep(0.2)
        else:
            GPIO.output(24, False)
except:
    GPIO.cleanup()
```

GPIO.PUD\_UP , the  
pull-up resistor is  
enabled;



# COMPARATIFS

# QUEL MCU, MINI-PC CHOISIR EN FONCTION DE L'APPLICATION ?

	Atmel AVR (Arduino et compatibles)	ATTiny	ESP8266	ESP32	STM32	Mini-PC ARM*	Mini-PC x86**
Initiation à la programmation	*****	****	*****	***	***	*****	*****
Initiation à la mesure avec des capteurs	*****	**	*****	***	***	***	***
Initiation au pilotage d'actionneur : moteur, servo-moteur, LED, pompe...	*****	**	*****	***	***	***	***
Communiquer avec un serveur domotique en WiFi	**	*	*****	***	**	*****	*****
Publier des mesures sur un serveur distant	**	*	*****	***	**	*****	*****
Projets électronique vestimentaire (cosplay, wearable...)	**	****	***	***	***	—	—
Projets							
RC, drone	***	*	***	*****	***	***	*
Robotique	***	*	*****	*****	*****	*****	*****
Environnemental	***	**	*****	*****	*****	*****	*****
CNC, impression 3D	*****	*	**	***	*****	*****	*****
Accessoires domotiques	***	*	*****	*****	***	**	*
Mobile, embarqué	*	**	***	*****	***	*****	**
Vision	—	—	—	*	**	*****	***
Traitement du signal	**	*	**	***	***	***	***
IA, réseau de neurones	—	—	—	—	—	***	***

(\*) Ordinateur carte à processeur ARM

(\*\*) Mini-PC à architecture x86 Intel ou AMD

<https://projetsdiy.fr/guide-projet-objet-connecte-diy-choix-micro-controleur-bus donnees-format/>

# COMPARAISON DES MCU

Spécification	<b>ATMega328P</b>	<b>ESP8266EX</b>	<b>ESP32 (2019)</b>	<b>SAMD21</b>	<b>STM32F103C8T6</b>	<b>STM32F407VET6</b>
<b>Core</b>		Tensilica L106 32-bit RISC	Xtensa 32-bit LX6 (1 ou 2 Cores)	ARM M0	ARM M3	ARM M4
<b>Puissance DMIPS</b>	20	?	600	?	90	225
<b>Fréquence</b>	20 MHz	160 MHz	80 ou 240 MHz	48 MHz	72 MHz	180 MHz
<b>Mémoire Flash</b>	4/8/16/32 KB	2 MB	4 MB	32/64/128/256 KB	64 ou 128 Kbytes	1 Mbytes
<b>SRAM</b>	512/1K/1K/2KB	Partagée < 36KB	520 KB	4/8/16/32 KB	20 KB	192+4 Kbytes
<b>Timers</b>	x2 (8 bits) x1 (16 bits)			x5 (16-bits) x3 (24-bits)	x3 (16-bits) x2 watchdog x1 SysTick	x12 (16-bits) x2 (32-bits)
<b>Cryptage (accélération matériel)</b>			AES (FIPS PUB 197), SHA (FIPS PUB 180-4), RSA, et ECC			AES 128, 192, 256, DES, MD5, SHA-1
<b>Debug</b>	USB (série)	USB (série)	USB (série) + JTAG	USB (série)	Série (SWD) + JTAG	Série (SWD) + JTAG
<b>Economie énergie</b>	Idle, ADC Noise Reduction, Power-save, Power-down, Standby, Extended Standby	Active, modem-sleep, light-sleep, deep-sleep	Active, modem-sleep, light-sleep, deep-sleep	Idle, Standby	Sleep, Stop, Standby	Sleep, Stop, Standby
<b>Interface et connectivité</b>						
<b>UART</b>	x1	x2	x3	x6**	x3	x6 (11.25 Mbit/s max.)
<b>SPI</b>	x1	x2	x3	x6**	x2 (18 Mbit/s)	x3 (45 Mbit/s max.)
<b>I2C</b>	x2	x1	x2	x1	x2	x3
<b>I2S</b>		x1	x2	x6**		x2
<b>PWM</b>	x6	x4	x16		x1 (16-bits)	
<b>ADC</b>	x8 (10-bits)	x1 (10-bits)	x2 (12-bits)	x1 (12-bits)	x2 (12-bits)	x3 (12-bits)
<b>DAC</b>			x8 (8-bits)	x1 (10-bits)		x2 (12-bits)
<b>CAN 2.0</b>			x1		x1	x2
<b>E/S (GPIO)</b>	x23	x17	x34	x52	26/37/51/80	x140 (dont 138 tolérantes 5V)
<b>SDIO</b>		x2 (slave)	x1			x1
<b>Interface tactile</b>			x10 (capacitif)	x256 (capacitif, proximité)		
<b>Infrarouge</b>		x1	x8			
<b>Caméra</b>						x1
<b>Ethernet</b>			x1			x1 IEEE 1588v2
<b>USB 2.0 micro-OTG</b>			x1	x1	x1	x2
<b>WiFi</b>		802.11 b/g/n	802.11 b/g/n			
<b>Bluetooth LE 4.x</b>			Oui			
<b>Documentation technique</b>	<a href="#">Lire</a>	<a href="#">Lire</a>	<a href="#">Lire</a>	<a href="#">Lire</a>	<a href="#">Lire</a>	<a href="#">Lire</a>

<https://projetsdiy.fr/guide-projet-objet-connecte-diy-choix-micro-controleur-bus donnees-format/>

# QUELLE SOLUTION WIRELESS CHOISIR ?

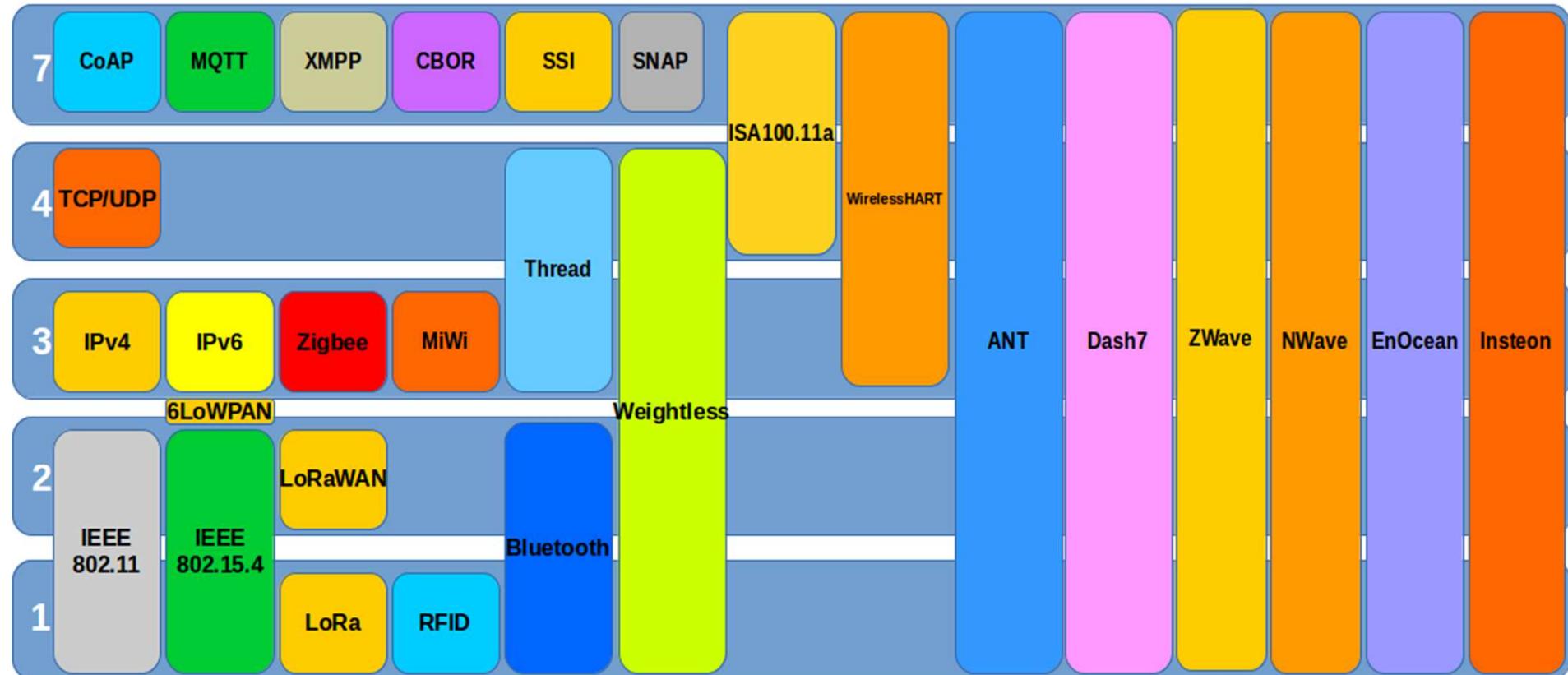
## Summary: What wireless technology to choose for a DIY project?

Rather than giving answers, the best to answer this question and help you choose wireless technology best suited to your project. Here are several criteria that you can consider:

- Transmission range: Short (remote control), medium (a hundred meters) or long (several hundred meters)
- Configuration: in a house, outdoor, mixed
  - Wall: thickness, constitution (reinforced concrete does not really help), number
  - flat or embossed
  - Vegetation, fence
  - electromagnetic disturbance: transformer, pumps ...
- Running on battery or AC power
- Budget
- Space available
- Protocol:
  - You manage communication
  - The object will interact with an online service: IFTTT, ThingSpeak ...
  - Automation Server: MQTT, MySensors, RFLink, OpenZWave ...
- Your level. If you begin, choose a technology used by a large community of users. You will find many examples and projects.

	2nd criteria	WiFi	Bluetooth	XBee	Other (433 MHz...)	LoRaWan or Sigfox
Transmission range	Short	*****	*****	*****	*****	*****
	Medium	***	-	*****	*****	*****
	Long	*	-	***	*****	*****
Configuration of the land	Esay	*****	*****		*****	*****
	Medium	****	-		***	***
	Hard	**	-		***	***
Battery powered		**	***	*****	*****	*****
Compact		*****	***	***	***	***
Budget		*****	***	**	****	**
Protocol	Point to point	***	****	*****	*****	*****
	Cloud	*****	*	**	*	****
	Smart Home	*****	**	***	*****	**
Community		*****	*****	***	***	**

To conclude this article, conventional radio ESP8266 WiFi modules and antennas (433, 868 or 2.4GHz) are the easiest and cheapest solutions to implement for self even develop online projects. If you develop a remote-controlled project, you can turn to the **Bluetooth** (or **XBee**). If this is a project for professional use (industry, logistics, medical, transport ...), go directly to the Zigbee in association with LoRaWAN or **SigFox**.



<http://www.linuxembedded.fr/2016/03/protocoles-de-communication-frameworks-et-systemes-dexploitation-pour-les-objets-connectes/#lora-lorawan>

## IV-5 AND SECURITY ?

# AND issues of security in the Internet of things!

The screenshot shows the Hackaday homepage with a navigation bar at the top featuring links for HOME, BLOG, HACKADAY.IO, STORE, HACKADAY PRIZE, SUBMIT, and ABOUT. A skull and wrenches logo is on the left. The main article title is "ARDUWORM: A MALWARE FOR YOUR ARDUINO YUN" by Elliot Williams, published on November 11, 2016, with 23 comments. Below the title is a photograph of an Arduino Yun board. A sidebar on the right includes search, social media (Facebook, Google+), newsletter sign-up, and a "IF YOU" section.

We've been waiting for this one. A worm was written for the Internet-connected Arduino Yun that gets in through a memory corruption exploit in the ATmega32u4 that's used as the serial bridge. [The paper](#) (as PDF) is a bit technical, but if you're interested, it's a great read. (Edit: The link went dead. [Here is our local copy.](#))

We've been waiting for this one. A worm was written for the Internet-connected Arduino Yun that gets in through a memory corruption exploit in the ATmega32u4 that's used as the serial bridge.

[The paper](#) (as PDF) is a bit technical, but if you're interested, it's a great read. (Edit: The link went dead. [Here is our local copy.](#))

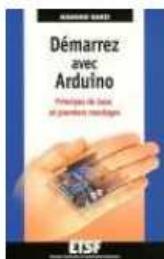
The crux of the hack is getting the AVR to run out of RAM, which more than a few of us have done accidentally from time to time. Here, the hackers write more and more data into memory until they end up writing into the heap, where data that's used to control the program lives. Writing a worm for the AVR isn't as easy as it was in the 1990's on PCs, because a lot of the code that you'd like to run is in flash, and thus immutable. However, if you know where enough functions are located in flash, you can just use what's there. These kind of [return-oriented programming](#) (ROP) tricks were enough for the researchers to write a worm.

In the end, the worm is persistent, can spread from Yun to Yun, and can do most everything that you'd love/hate a worm to do. In security, we all know that a chain is only as strong as its weakest link, and here the attack *isn't* against the OpenWRT Linux system running on the big chip, but rather against the small AVR chip playing a support role. Because the AVR is completely trusted by the Linux system, once you've got that, you've won.

Will this amount to anything in practice? Probably not. There are tons of systems out there with much more easily accessed vulnerabilities: hard-coded passwords and poor encryption protocols. Attacking all the Yuns in the world wouldn't be worth one's time. It's a very cool proof of concept, and in our opinion, that's even better.

# REFERENCES

# REFERENCE BOOKS



**Démarez avec Arduino: Principes de base et premiers montages, Massimo Banzi [français, 128 pages]**

Editions Dunod, 2011, ISBN-13: 978-2100562916

[chez l'éditeur](#)

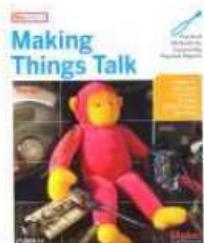


**Arduino - Maîtrisez sa programmation et ses cartes d'interface (shields), Christian Tavernier [français, 232 pages]**

Editions Dunod, 2011, ISBN-13: 978-2100559275

[Chez l'éditeur](#)

Site de l'auteur : <http://www.tavernier-c.com/>



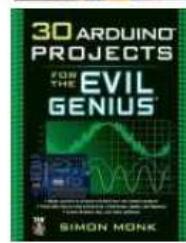
**Making Things Talk, Tom Igoe [anglais, 426 pages]**

Editions O'Reilly, 2007, ISBN-13 : 978-0-596-51051-0

Site de l'auteur : <http://www.tigoe.net/pcomp/>

Chez l'éditeur : <http://oreilly.com/catalog/9780596510510/>

Seconde édition prévue pour août 2011 : <http://oreilly.com/catalog/0636920010920/>

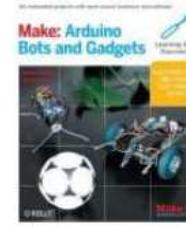


**30 Arduino Projects for the Evil Genius, Simon Monk [anglais, 208 pages]**

Editions McGraw Hill, 2010, ISBN : 978-0071741330

Chez l'éditeur : <http://www.mcgraw-hill.co.uk/html/007174133X.html>

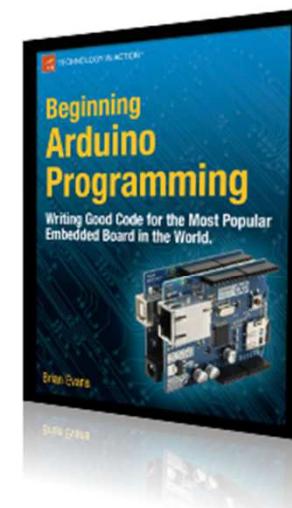
Site : <http://www.arduinoevilgenius.com/>



**Make: Arduino Bots and Gadgets, Learning by Discovery, Tero Karvinen & Kimmo Karvinen [anglais, 296 pages]**

Editions O'Reilly, 2011, ISBN-13: 978-1449389710

Chez l'éditeur : <http://oreilly.com/catalog/0636920010371/>



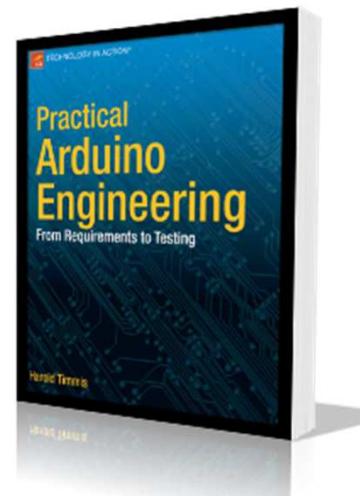
**Beginning Arduino Programming**

Writing Good Code for the Most Popular Embedded Board in the World.



Brian Evans

EUR 24,99

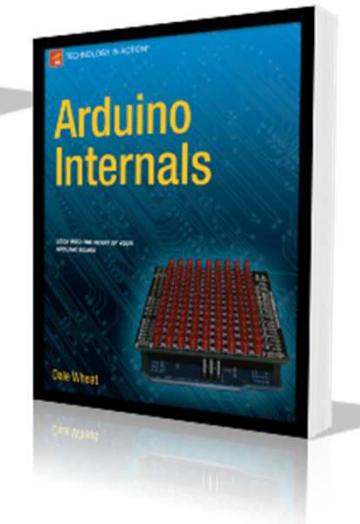


**Practical Arduino Engineering**

From Requirements to Testing

Helge Tørris

EUR 24,99



**Arduino Internals**

2014 RELEASE

With CD

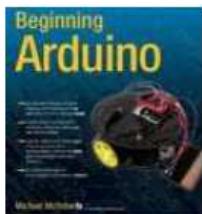
EUR 24,99



Dale Wheat

EUR 24,99

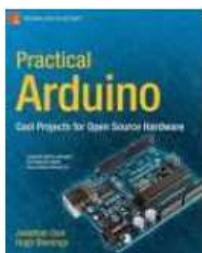
# REFERENCE BOOKS



**Beginning Arduino, Michael McRoberts [anglais, 472 pages]**

Editions Apress, 2011, ISBN13: 978-1-4302-3240-7

Chez l'éditeur : <http://www.apress.com/9781430232407>  
Code source : [http://www.apress.com/downloadable/down...le\\_id/358/](http://www.apress.com/downloadable/down...le_id/358/)



**Practical Arduino, Jonathan Oxer & Hugh Blemings [anglais, 456 pages]**

Editions Apress, 2010, ISBN13: 978-1-4302-2477-8

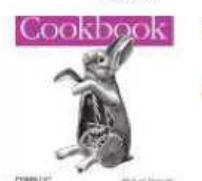
Chez l'éditeur : <http://www.apress.com/9781430224778>  
Site : <http://www.practicalarduino.com/>  
Twitter : <http://twitter.com/parduino>



**Arduino, a Quick Start Guide, Maik Schmidt [anglais, 296 pages]**

Editions The Pragmatic Programmers, 2011, ISBN: 978-1-93435-666-1

Sur le site de l'éditeur : <http://pragprog.com/titles/msard/arduino>  
code source : [http://pragprog.com/titles/msard/source\\_code](http://pragprog.com/titles/msard/source_code)



Editions O'Reilly, 2011, ISBN-13 : 978-0-596-80247-9

Chez l'éditeur : <http://oreilly.com/catalog/9780596802486>

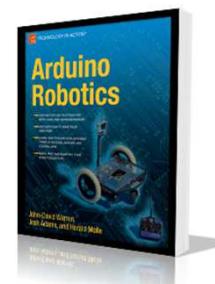


**Programming Interactivity, Joshua Noble [anglais, 712 pages]**

Editions O'Reilly, 2009, ISBN-13 : 978-0-596-15414-1

« Programming Interactivity » couvre processing, arduino et openFrameworks.  
<http://oreilly.com/catalog/9780596154158> ([google preview](#))

blog : <http://programminginteractivity.com>



- ISBN13: 978-1-4302-3183-7
- 628 Pages
- User Level: Intermediate to Advanced
- Publication Date: July 18, 2011



**Open Software [anglais, 104 pages]**

A paraître  
A paraître, BlushingBoy Publishing, 2011, ISBN : 978-91-97-95540-9

« Open Software is a book about fashion and technology. More precisely it is a book about Arduino boards, conductive fabric, resistive thread, soft buttons, LEDs, and some other things.»  
site : <http://software.cc/>  
La version beta de 2009 est téléchargeable sur le site

# MicroPython on ESP32 - REFERENCES

## Installing Python 3.7.X - Windows PC

<https://randomnerdtutorials.com/install-upycraft-ide-windows-pc-instructions/>

## Installing uPyCraft IDE - Windows PC

<https://randomnerdtutorials.com/uPyCraftWindows.>

[adobe-fonts/source-code-pro](#) <https://github.com/adobe-fonts/source-code-pro>

## Flash/Upload MicroPython Firmware to ESP32 and ESP8266

<https://randomnerdtutorials.com/flash-upload-micropython-firmware-esp32-esp8266/>

<https://docs.micropython.org/en/latest/esp32/tutorial/intro.html>

<https://randomnerdtutorials.com/esp32-esp8266-analog-readings-micropython/>

<https://randomnerdtutorials.com/micropython-programming-basics-esp32-esp8266/>

## ESP32 + LoRa :

<https://randomnerdtutorials.com/esp32-lora-rfm95-transceiver-arduino-ide/>

## Pinout Reference

<https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>

## My Scenari:

[https://lecluseo.scenari-community.org/CircuitPython/co/AA\\_MQTT\\_1.html](https://lecluseo.scenari-community.org/CircuitPython/co/AA_MQTT_1.html)

## How to use ESP32 Dual Core with Arduino IDE

<https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/>

# MAIN RESSOURCES

## MAIN RESSOURCES (ARDUINO/PROCESSING/FRITZING) & OPEN SOURCE HARDWARE:

Site principal : <http://arduino.cc/>

Téléchargement : <http://arduino.cc/en/Main/Software>

Guide de référence du langage : <http://arduino.cc/en/Reference/HomePage>

Forum de discussion : <http://arduino.cc/forum/>

Le forum français : <http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?board=francais>

Blog : <http://arduino.cc/blog/>

Wiki / playground (incontournable) : <http://arduino.cc/playground/>

wikipedia (fr) <http://fr.wikipedia.org/wiki/Arduino>

Liste des Shields: <http://shieldlist.org/>

Education: <http://scuola.arduino.cc/>

Processing: <http://processing.org/>

Fritzing : <http://fritzing.org/> (Prototypage Dessin des cartes , circuit graphique, schématique ou électronique)

Open Source Hardware: <http://dangerousprototypes.com/2011/06/30/tutorial-arduino-and-the-spi-bus/>

Et quand cela ne marche PAS !!! : <http://arduino.cc/en/Guide/Troubleshooting>

## SUR LE WEB:

Référence arduino en français : [http://www.mon-club-elec.fr/pmwiki\\_refe...n.HomePage](http://www.mon-club-elec.fr/pmwiki_refe...n.HomePage)

Référence en français: <http://www.mon-club-elec.fr/> : site perso amateur sur lequel je mets en ligne mes développements en électronique numérique programmée, dans une optique ludique et expérimentale, "just for fun" !

Initiation à Arduino (2006) : <http://www.craslab.org/arduino/livretht...oCRAS.html> (ou en [version pdf](#))

The World Famous Index of Arduino & Freeduino Knowledge : <http://www.freeduino.org/>

Tutoriels de Tronixstuff <http://tronixstuff.wordpress.com/tutorials/>

Tutoriels d'Adafruit: <http://www.adafruit.com/tutorials>

Ressources / tutoriaux pour arduino sur le site du labomedia : [http://wiki.labomedia.org/index.php/Lie...ux\\_Arduino](http://wiki.labomedia.org/index.php/Lie...ux_Arduino)

Scratch pour Arduino : <http://seaside.citilab.eu/scratch/arduino>

Liste de discussion des développeurs d'arduino : <http://arduino.cc/mailman/listinfo/deve...arduino.cc>

Documentaire sur Arduino: <http://vimeo.com/18539129>

## WEB LINKS

### GENERALISTES:

<http://www.bricoelec.com/>[fr] Portail francophone de l'arduino : Robotique, Bidouillage et hacking matériel (DIY)!

[http://arts-numeriques.codedrops.net/-Microcontrolleur-arduino-\[fr\]](http://arts-numeriques.codedrops.net/-Microcontrolleur-arduino-[fr])

<http://www.ladyada.net/learn/arduino/index.html>

<http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard>

<http://todbot.com/blog/bionicarduino/>[en] It is an introduction to microcontroller programming and interfacing with the real world using the Arduino physical computing platform. It focuses on building new physical senses and making motion with the building blocks of robotics, using Arduino as a platform.

•Très bon tutoriels progressifs : <http://todbot.com/blog/spookyarduino/>

•le site de Tom Igoe <http://tigoe.net/pcomp/>

•les sites Make, Hackaday, Sensorwiki, We make money not art, et bien d'autres encore pour avoir des conseils, des schémas, des idées....

•les sites de service de liens, de vidéos et d'images ( google, delicious, youtube, flickr,)

<http://luckyLarry.co.uk/arduino-projects/arduino-weblinks-projects/>

<http://hacknmod.com/hack/top-40-arduino-projects-of-the-web/>

<http://santy.wikidot.com/arduino>

<http://zedomax.com/blog/2010/02/16/top-10-diy-arduino-projects-and-how-to-tutorials/>

Liens divers: <http://www.uni-weimar.de/medien/wiki/Arduino/Links>

<http://barzilouik.free.fr/wiki/doku.php?id=arduino:tutorial>

### REVENDEURS:

<http://shop.snootlab.com/> (Toulouse !!!)

<http://www.lextronic.fr/>

<http://www.robotshop.com/eu/arduino-en.html>

<http://boutique.semageek.com/fr/2-arduino>

<http://adafruit.com/>

<http://www.sparkfun.com/>

## TO GO FURTHER : BUS I2C

**COMMUNICATION EN GENERAL:** <http://arduino.cc/playground/Main/InterfacingWithHardware#Communication>

I<sup>2</sup>C: <http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html>

<http://www.atmicroprog.com/cours/I2C/i2c.php>

<http://www.esacademy.com/en/library/technical-articles-and-documents/miscellaneous/i2c-bus.html>

[http://www.robot-electronics.co.uk/acatalog/I2C\\_Tutorial.html](http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html)

<http://www.neufeld.newton.ks.us/electronics/?p=241>

Librairie Wii-Nunchuk: <https://github.com/lgeek/Arduino-Wii-Nunchuk>

TOP 30 utilisation du Nunchuk: <http://hacknmod.com/hack/top-30-wiimote-hacks-of-the-web/>

<http://johnnylee.net/projects/wii/>

<http://www.planete-sciences.org/robot/ressources/electronique/protocoleI2C/index.html>

<http://www.pobot.org/Controle-avec-un-Wii-Nunchuck.html>

Wii Motion Plus + Nunchuck: <http://voidbot.net/nunchchuk-and-wii-motion-plus-6-DOF.html>

<http://randomhacksofboredom.blogspot.com/2009/06/wii-motion-plus-arduino-love.html>

<http://www.freelug.org/spip.php?article917>

<http://www.aurel32.net/elec/i2c.php>

<http://todbot.com/blog/bionicarduino/>

<http://todbot.com/blog/2010/09/25/softi2cmaster-add-i2c-to-any-arduino-pins/>

<http://tronixstuff.wordpress.com/2010/10/20/tutorial-arduino-and-the-i2c-bus/>

<http://tronixstuff.wordpress.com/2010/10/29/tutorial-arduino-and-the-i2c-bus-part-two/>

<http://tronixstuff.wordpress.com/2010/05/20/getting-started-with-arduino-%E2%80%93-chapter-seven/>

<http://www.nearfuturelaboratory.com/2009/07/17/lis302dl-a-3-axis-accelerometer/>

[http://wiire.org/Main\\_Page](http://wiire.org/Main_Page)

[http://wiire.org/Wii/protocols/wiimote\\_bus#Wiimote\\_Bus\\_Pins\\_.286-pin\\_proprietary\\_connector\\_on\\_Wiimote.29](http://wiire.org/Wii/protocols/wiimote_bus#Wiimote_Bus_Pins_.286-pin_proprietary_connector_on_Wiimote.29)

Note:

There are both 7- and 8-bit versions of I<sup>2</sup>C addresses. 7 bits identify the device, and the eighth bit determines if it's being written to or read from. The Wire library uses 7 bit addresses throughout. If you have a datasheet or sample code that uses 8 bit address, you'll want to drop the low bit (i.e. shift the value one bit to the right), yielding an address between 0 and 127.

## WEB LINKS

CAPTEURS: <http://itp.nyu.edu/physcomp/sensors/Reports/Reports>

vers l'interface MIDI avec une arduino: <http://itp.nyu.edu/physcomp/Labs/MIDIOutput>

RFID: <http://www.sparkfun.com/products/8419>

<http://jadiema.blogspot.com/2011/03/arduino-rfid-reader-piezo-sounder.html>

COMPASS: <http://lusorobotica.com/index.php/topic,36.0.html>, <http://wiring.org.co/learning/libraries/hmc6352sparkfun.html>

ROBOT: <http://arduino103.blogspot.com/2011/06/comment-dimensionner-un-moteur.html>

<http://www.pobot.org/> [fr] L'association POBOT a été créée en 2003, par un groupe de passionnés de robotique, dans le but : d'apprendre ensemble, de se perfectionner, d'échanger, de s'entre-aider.

<http://www.robot-maker.com/index.php?/topic/3380-divers-robots-a-a-base-darduino-et-de-wifi/>

<http://robot.aspi.free.fr/>

APPROCHE CLIENT-SERVEUR AVEC LIAISON WIFI : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoExpertSymbioseArduinoPCPrincipeDeploiement](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertSymbioseArduinoPCPrincipeDeploiement)

NUNCHUK MEETS NXT: <http://www.mindstormsforum.de/viewtopic.php?t=3828>

ANDROID ADK MEETS ARDUINO: <http://www.amarino-toolkit.net/index.php/my-apps.html>

<http://www.mobot.es/MobotBTCar.html>

Vélo pour Geek: <http://www.michaelfretz.com/2011/01/03/punkt-fizen-android-and-arduino-powered-bike-navigation/>

<http://www.labradoc.com/i/follower/p/android-arduino-accessory>

<http://www.labradoc.com/i/follower>

<http://www.semageek.com/handbag-une-application-android-pour-piloter-ladk-arduino-sans-programmation/>

CELLULES SOLAIRES : <http://www.doctormonk.com/2011/09/arduino-solar-radio.html>

RASPBERRY-Pi : <http://www.doctormonk.com/2012/04/raspberry-pi-and-arduino.html>

POUR LES ARTISTES:

<http://softwear.cc/book/2011/open-softwear-2nd-edition/page/2/>

<http://web.media.mit.edu/~leah/LilyPad/>

TAPIS SENSITIF, ÉCRANS MULTI-TOUCH...: <http://numuscus.pascaq.org/?p=551>

<http://www.interface-z.com/>

<http://codelab.fr/2552>, partie table Multi-Touch et reactive: <http://codelab.fr/1076>

# LINKS

Introduction au système Arduino :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.DebuterIntroduction](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.DebuterIntroduction)

Vue d'ensemble des cartes :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.Materiel](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.Materiel)

FAQ : => [http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.FAQ](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.FAQ)

Définition de l'open source ! et plein d'autres bricoles => Lien avec les shields...

Le Logiciel :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.DebuterPresentationLogiciel](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.DebuterPresentationLogiciel)

La carte Duemilanove :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.MaterielDuemilanove](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.MaterielDuemilanove)

Le brochage :

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.REFERENCESBrochageDuemilanove](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.REFERENCESBrochageDuemilanove)

Compilation :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.ApprendreProcedureCompilation](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.ApprendreProcedureCompilation)

Premier programme :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.ApprendreProgrammeTypeMinimum](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.ApprendreProgrammeTypeMinimum)

Références :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.Reference](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.Reference)

Références Etendues:

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.ReferenceEtendue](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.ReferenceEtendue)

Les librairies de références :

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.ReferenceLibrairies](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.ReferenceLibrairies)

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.Librairies](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.Librairies)

Serial : [http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.Serial](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.Serial)

Stepper: [http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.LibrairieStepper](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.LibrairieStepper)

Servo: [http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.LibrairieServo](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.LibrairieServo)

LCD: [http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n>Main.LibrairieLCD](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n>Main.LibrairieLCD)

<http://www.aurel32.net/elec/lcd.php>, [http://fabrice.sincere.pagesperso-orange.fr/cm\\_electronique/projet\\_pic/LCDAlpha/LCDAlpha.htm](http://fabrice.sincere.pagesperso-orange.fr/cm_electronique/projet_pic/LCDAlpha/LCDAlpha.htm)

## LINKS

Comment écrire une librairie: <http://www.robotix.fr/tutoriel-1-59-creer-une-bibliotheque-arduino.html>

[http://www.robot-maker.com/index.php?/user/4963-Philippe-RX/page\\_tab\\_tutorials](http://www.robot-maker.com/index.php?/user/4963-Philippe-RX/page_tab_tutorials)

Fiche sur LCD => Communication avec LCD : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD)

Fiche Touch Shield Nintendo DS (Entrée analogique et Sortie Digitale)

Fiche sur Nunchuck (accelerometer) => Bus I2C

Bus SPI <http://www.arduino.cc/playground/Code/Spi>

Pourquoi ne pas écrire dans une carte SD ? => il faut le shield Ethernet officiel:

<http://www.ladyada.net/learn/arduino/ethfiles.html>

Fiche sur l'Ethernet Shield : <http://www.ladyada.net/learn/arduino/ethfiles.html>

Tester les interruptions externes => Optocoupleur à fourche : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoInitiationEntreesOnOffOptoFourcheTestSimple](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoInitiationEntreesOnOffOptoFourcheTestSimple)

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.MaterielCapteurOptofourcheLTH301-07](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.MaterielCapteurOptofourcheLTH301-07)

<http://www.gotronic.fr/catalog/opts/interrupteurs.htm>

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoExpertServoPanSuiviBalle](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertServoPanSuiviBalle)

Focus sur les Shields

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.MATERIEL](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.MATERIEL)

Alimentation de PC:

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.TECHNIQUETrucsUtiliserAlimPC](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TECHNIQUETrucsUtiliserAlimPC)

LIENS PROCESSING :

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.OUTILSProcessing](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.OUTILSProcessing)

<http://www.scoop.it/t/arduino-processing>

**ARDUINO MEETS PROCESSING:**

[http://webzone.k3.mah.se/projects/arduino-workshop/projects/arduino\\_meets\\_processing/instructions/index.html](http://webzone.k3.mah.se/projects/arduino-workshop/projects/arduino_meets_processing/instructions/index.html)

**ARDUINO MEETS FLASH:**

<http://www.fabiobiondi.com/blog/2009/11/arduino-and-flash-modify-your-existent-flash-games-to-work-with-a-joystick/>

<http://www.fabiobiondi.com/blog/2009/10/arduino-and-flex-connect-a-joystick-to-your-application/>

Divers :

Générateur de code: [http://www.mon-club-elec.fr/pmwiki\\_generateur\\_code/pmwiki.php?n>Main.HomePage](http://www.mon-club-elec.fr/pmwiki_generateur_code/pmwiki.php?n>Main.HomePage)

R2D2 Translator: <http://www.r2d2translator.com/>

**ARDUINO MEETS LABVIEW:** <http://sine.ni.com/nips/cds/view/p/lang/en/nid/209835>

<http://vishots.com/getting-started-with-the-labview-interface-for-arduino/>

Et un article pour réfléchir: <http://www.framablog.org/index.php/post/2010/06/02/arduino-materiel-libre>

## LINKS

### OPEN SOURCE HARDWARE ET LICENCE LIBRE

[http://en.wikipedia.org/wiki/Open\\_source\\_hardware](http://en.wikipedia.org/wiki/Open_source_hardware)

<http://fr.wikipedia.org/wiki/Copyleft>

[http://fr.wikipedia.org/wiki/Licence\\_Creative\\_Commons](http://fr.wikipedia.org/wiki/Licence_Creative_Commons)

<http://freedomdefined.org/Definition/Fr>

<http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>

<http://opensource.org/osd>

### PWM

<http://www.siteduzero.com/sciences/tutoriels/arduino-pour-bien-commencer-en-electronique-et-en-programmation/transformation-pwm-signal-analogique>

<http://arduino-info.wikispaces.com/Arduino-PWM-Frequency>

[https://fr.wikiversity.org/wiki/Micro\\_contr%C3%B4leurs\\_AVR](https://fr.wikiversity.org/wiki/Micro_contr%C3%B4leurs_AVR)

<http://blog.cicatrice.eu/electronique/avr>

## LINKS ON TOULOUSE

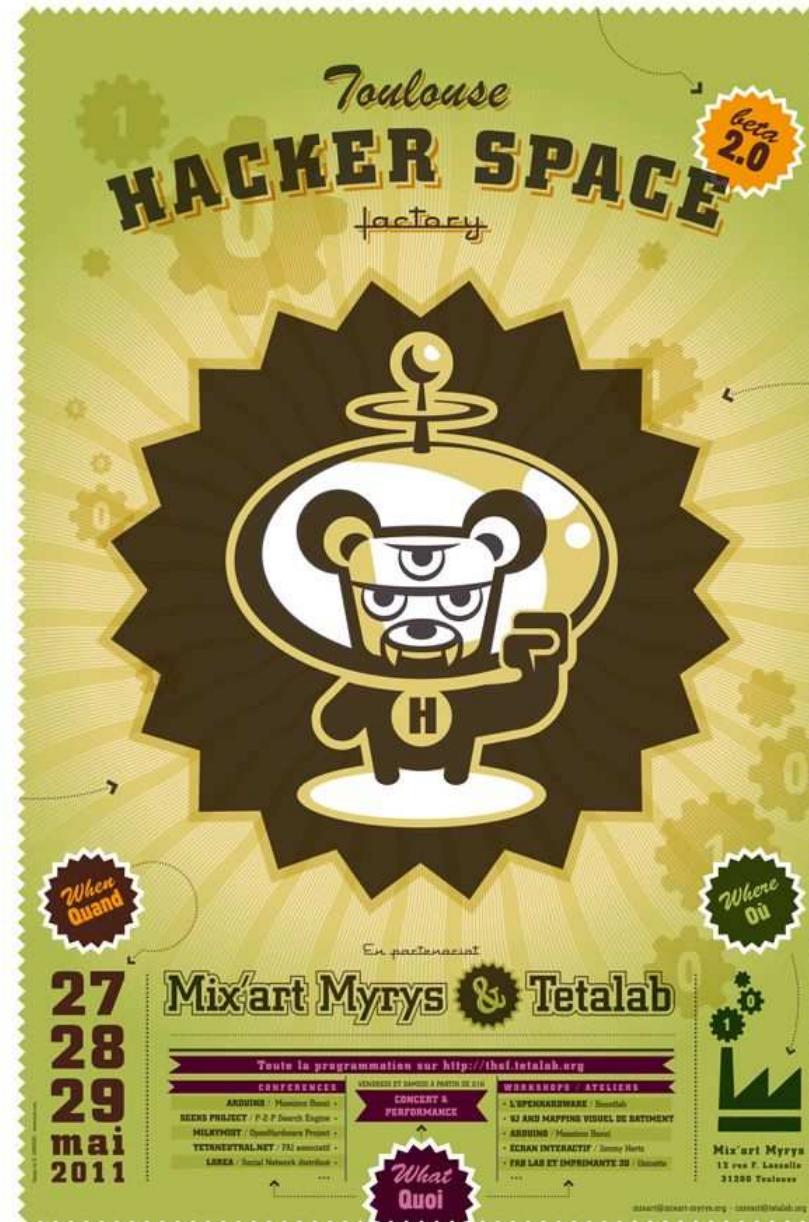
<http://linuxedu.ac-toulouse.fr/doku.php?id=materiel:arduino:ressources:pedagogiques>

[http://www.planete-sciences.org/robot/boiteabots/index.php?option=com\\_content&task=view&id=44](http://www.planete-sciences.org/robot/boiteabots/index.php?option=com_content&task=view&id=44)

<http://planet.madeinfr.org/tag/arduino>

TETALAB: <http://tetalab.org/>

The Tetalab is a hackerspace Toulouse created in June 2009, with a framework for software/hardware projects.



**Structure**`void setup() void loop()`**Control Structures**

```
if (x<5){} else {}  
switch (myvar) {  
    case 1:  
    break;  
    case 2:  
    break;  
    default:  
}  
for (int i=0; i <= 255; i++){}  
while (x<5){}  
do {} while (x<5);  
continue; // Go to next in  
do/for/while loop  
return x; // Or 'return;' for voids.  
goto // considered harmful :-)
```

**Further Syntax**

```
// (single line comment)  
/* (multi-line comment) */  
#define DOZEN 12 //Not baker's!  
#include <avr/pgmspace.h>
```

**General Operators**

```
= (assignment operator)  
+ (addition) - (subtraction)  
* (multiplication) / (division)  
% (modulo)  
== (equal to) != (not equal to)  
< (less than) > (greater than)  
<= (less than or equal to)  
>= (greater than or equal to)  
&& (and) || (or) ! (not)
```

**Pointer Access**

```
& reference operator  
* dereference operator
```

**Bitwise Operators**

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

**Compound Operators**

```
++ (increment) -- (decrement)  
+= (compound addition)  
-= (compound subtraction)  
*= (compound multiplication)  
/= (compound division)  
&= (compound bitwise and)  
!= (compound bitwise or)
```

**Constants**

```
HIGH | LOW  
INPUT | OUTPUT  
true | false  
143 // Decimal number  
0173 // Octal number  
0b11011111 //Binary  
0xB // Hex number  
7U // Force unsigned  
10L // Force long  
15UL // Force long unsigned  
10.0 // Forces floating point  
2.4e5 // 240000
```

**Data Types**

```
void  
boolean (0, 1, false, true)  
char (e.g. 'a'-128 to 127)  
unsigned char (0 to 255)  
byte (0 to 255)  
int (-32,768 to 32,767)  
unsigned int (0 to 65535)  
word (0 to 65505 (0 to 65535)  
long (-2,147,483,648 to  
2,147,483,647)  
unsigned long (0 to 4,294,967,295)  
float (-3.4028235E+38 to  
3.4028235E+38)  
double (currently same as float)  
sizeof(myint) // returns 2 bytes
```

**Strings**

```
char S1[15];  
char S2[8]={'a','r','d','u','l','n','o'};  
char S3[8]={'a','r','d','u','l','n','o','\0'};  
//Included \0 null termination  
char S4[] = "arduino";  
char S5[8] = "arduino";  
char S6[15] = "arduino";
```

**Arrays**

```
int myInts[6];  
int myPins[] = {2, 4, 8, 3, 6};  
int mySensVals[6] = {2, 4, -8, 3, 2};
```

**Conversion**

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

**Qualifiers**

```
static // persists between calls  
volatile // use RAM (nice for ISR)  
const // make read-only  
PROGMEM // use flash
```

**Digital I/O**

```
pinMode(pin, [INPUT,OUTPUT])  
digitalWrite(pin, value)  
int digitalRead(pin)  
//Write High to inputs to use pull-up res
```

**Analog I/O**

```
analogReference([DEFAULT,  
INTERNAL,EXTERNAL])  
int analogRead(pin) //Call twice if  
switching pins from high Z source.  
analogWrite(pin, value) // PWM
```

**Advanced I/O**

```
tone(pin, freqhz)  
tone(pin, freqhz, duration_ms)  
noTone(pin)  
shiftOut(dataPin, clockPin,  
[MSBFIRST,LSBFIRST], value)  
unsigned long pulseIn(pin,[HIGH,LOW])
```

**Time**

```
unsigned long millis() // 50 days overflow.  
unsigned long micros() // 70 min overflow  
delay(ms)  
delayMicroseconds(us)
```

**Math**

```
min(x, y) max(x, y) abs(x)  
constrain(x, minval, maxval)  
map(val, fromL, fromH, toL, toH)  
pow(base, exponent) sqrt(x)  
sin(rad) cos(rad) tan(rad)
```

**Random Numbers**

```
randomSeed(seed) // Long or int  
long random(max)  
long random(min, max)
```

**Bits and Bytes**

```
lowByte()  
highByte()  
bitRead(x,bitn)  
bitWrite(x,bitn)  
bitSet(x,bitn)  
bitClear(x,bitn)  
bit(bitn) //bitn: 0-LSB 7-MSB
```

**External Interrupts**

```
attachInterrupt(interrupt, function,  
[LOW,CHANGE,RISING,FALLING])  
detachInterrupt(interrupt)  
interrupts()  
noInterrupts()
```

**Libraries:****Serial.**

```
begin([300, 1200, 2400, 4800,  
9600,14400, 19200, 28800, 38400,  
57600,115200])  
end()  
int available()  
int read()  
flush()  
print()  
println()  
write()
```

**EEPROM** (#include <EEPROM.h>)  
byte read(intAddr)  
write(intAddr,myByte)

**Servo** (#include <Servo.h>)

```
attach(pin , [min_uS, max_uS])  
write(angle) // 0-180  
writeMicroseconds(uS) //1000-  
2000,1500 is midpoint  
read() // 0-180  
attached() //Returns boolean  
detach()
```

**SoftwareSerial**(RxPin,TxPin)

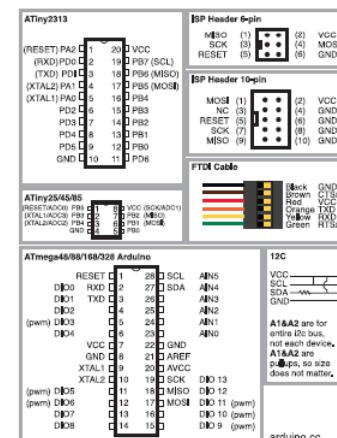
```
// #include<SoftwareSerial.h>  
begin(longSpeed) // up to 9600  
char read() // blocks till data  
print(myData) or println(myData)
```

**Wire** (#include <Wire.h>) // For I2C

```
begin() // Join as master  
begin(addr) // Join as slave @ addr  
requestFrom(address, count)  
beginTransmission(addr) // Step 1  
send(mybyte) // Step 2  
send(char * mystring)  
send(byte * data, size)  
endTransmission() // Step 3  
byte available() // Num of bytes  
byte receive() //Return next byte  
onReceive(handler)  
onRequest(handler)
```

	ATMega168	ATMega328	ATMega1280
Flash (2k for bootloader)	16kB	32kB	128kB
SRAM	1kB	2kB	8kB
EEPROM	512B	1kB	4kB

	Duemilanove/ Nano/Pro/ ProMini	Mega
# of IO	14 + 6 analog (Nano has 14 + 8)	54 + 16 analog
Serial Pins	0 - RX 1 - TX	0 - RX1 1 - TX1 19 - RX2 18 - TX2 17 - RX3 16 - TX3 15 - RX4 14 - TX4
Ext Interrupts	2 - (Int 0) 1 - (Int 1)	2,3,21,20,19,18 (IRQ0 - IRQ5)
PWM Pins	5,6 - Timer 0 9,10 - Timer 1 3,11 - Timer 2	0 - 13
SPI	10 - SS 11 - MOSI 12 - MISO 13 - SCK	53 - SS 51 - MOSI 50 - MISO 52 - SCK
I2C	Analog4 - SDA Analog5 - SCK	20 - SDA 21 - SCL



# GLOSSARY & VOCABULARY OF THE ARDUINO

**ANALOG** : Analogique (0 à 1023 sur l'arduino) .

**AREF** : Abréviaison pour Analog REFérence, référence analogique.

**AVAILABLE** : Disponible.

**BEGIN** : Début.

**BIT** : bit, unité d'information informatique pouvant prendre soit la valeur 0 soit la valeur 1.

**BUFFER** : Tampon, dans le sens de "zone tampon". Mémoire temporaire

**BYTE** : Octet, soit un groupe de 8 bits.

**BPS** : Abréviaison pour Bits Per Second, Bits Par Seconde. Attention, abréviaiton toujours en minuscules.

**BREADBOARD**: plaque d'expérimentation

**CAPACITOR**: condensateur

**CHAR** : Pour CHARacter, caractère (typographique). Type de variable d'une taille d'un octet. C'est un synonyme de "byte" utilisé pour déclarer des variables stockant un caractère ou des chaines de caractères.

**DEFINE** : Définit.

**DIGITAL** : Numérique (0 (LOW) ou 1 (HIGH))

**DO** : Faire.

**FALSE** : Faux.

**FOR** : Pour. Jusqu'à ce que.

**GND** : Abréviaiton pour GrouND, la terre. C'est la masse, 0 Volt.

**HIGH** : Haut.

**ICSP**: Abréviaiton pour In Circuit Serial Programming, programmation série sur circuit.

**IF / THEN / ELSE** : Si / Alors / Sinon.

**IN** : Souvent l'abréviaiton pour INput, Entrée. Est toujours en rapport avec le sens extérieur vers carte Arduino.

**INCLUDE** : Inclut.

**INPUT** : Entrée.

**IS** : Est (souvent dans le sens d'une question : Est ?).

**INT** : Abréviaiton pour INTeger, entier. Groupe de 16 bits, 2 octets groupés, considérés comme représentant un nombre entier négatif ou positif.

**LONG** : Abréviaiton pour "entier long". Groupe de 32 bits, 4 octets groupés, considérés comme représentant un nombre entier négatif ou positif.

**LOOP** : Boucle.

**LOW** : Bas.

**OUT** : Souvent l'abréviaiton pour OUTput, Sortie. Est toujours en rapport avec le sens carte Arduino vers extérieur.

**OUTPUT** : Sortie.

**PIN** : Broche d'E/S connectée à quelque chose (e.g. une LED).

**POWER** : Puissance, alimentation.

**PWM** : Abréviaiton de (Pulse Width Modulation), soit Modulation en Largeur d'Impulsion.

**PWR** : Abréviaiton pour PoWeR, puissance, alimentation.

**READ**: Lire.

**RESISTOR**: résistance.

**RELAY**: relais.

**RX** : Abréviaiton pour Receive, réception.

**SERIAL** : Série.

**SETUP** : Initialisation.

**SENSOR**: capteur

**SKETCH**: programme qui tourne sur le micro-contrôleur

**SWITCH** : basculer, interrupteur

**TRUE** : Vrai.

**TX**: Abréviaiton pour Transmit, transmission.

**WIRING**: câble, fils montrant les interconnections physiques des composants.

**WHILE** : Tant que.

**WORD** : mot, soit dans le sens de langage ; soit dans le sens d'un groupe de 16 bits, 2 octets groupés considérés comme représentant un nombre entier positif ( $\geq 0$ ).

**WRITE**: Ecrire.

## A QUIZ FOR YOUR TRAINING...

After taking this course, here's a quiz to test your knowledge:



Quiz section of the course on MOODLE :

*<http://moodle.insa-toulouse.fr/mod/quiz/view.php?id=18029>*

