Introduction to Arduino

What's this "Arduino" thing everyone's talking about ?

Bordeaux INP ENSEIRB MATMECA

Anthony Gautier http://silica.io

Table of contents

License

This guide was originally written for the members of **EirSpace**, the aeronautics and space club of the ENSEIRB-MATMECA engineering school in Bordeaux, France.

It is now released for everyone under the Creative Commons permissive licence.

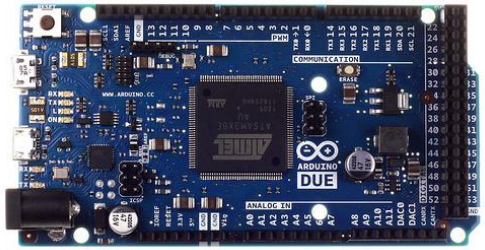
Find more about EirSpace at www.eirspace.fr.

“Arduino”, what is it ?

1) Some electronic boards
2) A software
3) A framework

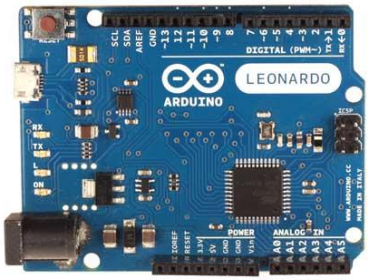
1) Some electronic boards

Uno



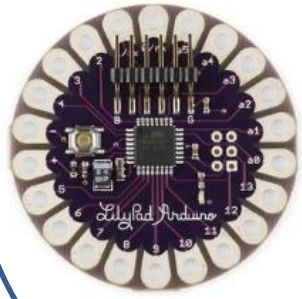
Due

SAM3X8E ARM Cortex-M3



Leonardo

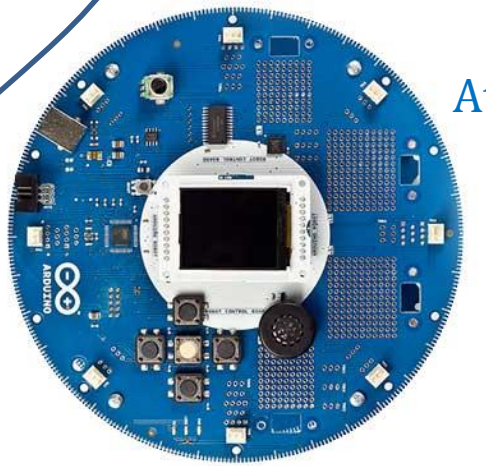
Atmega328



Lilypad

Nano

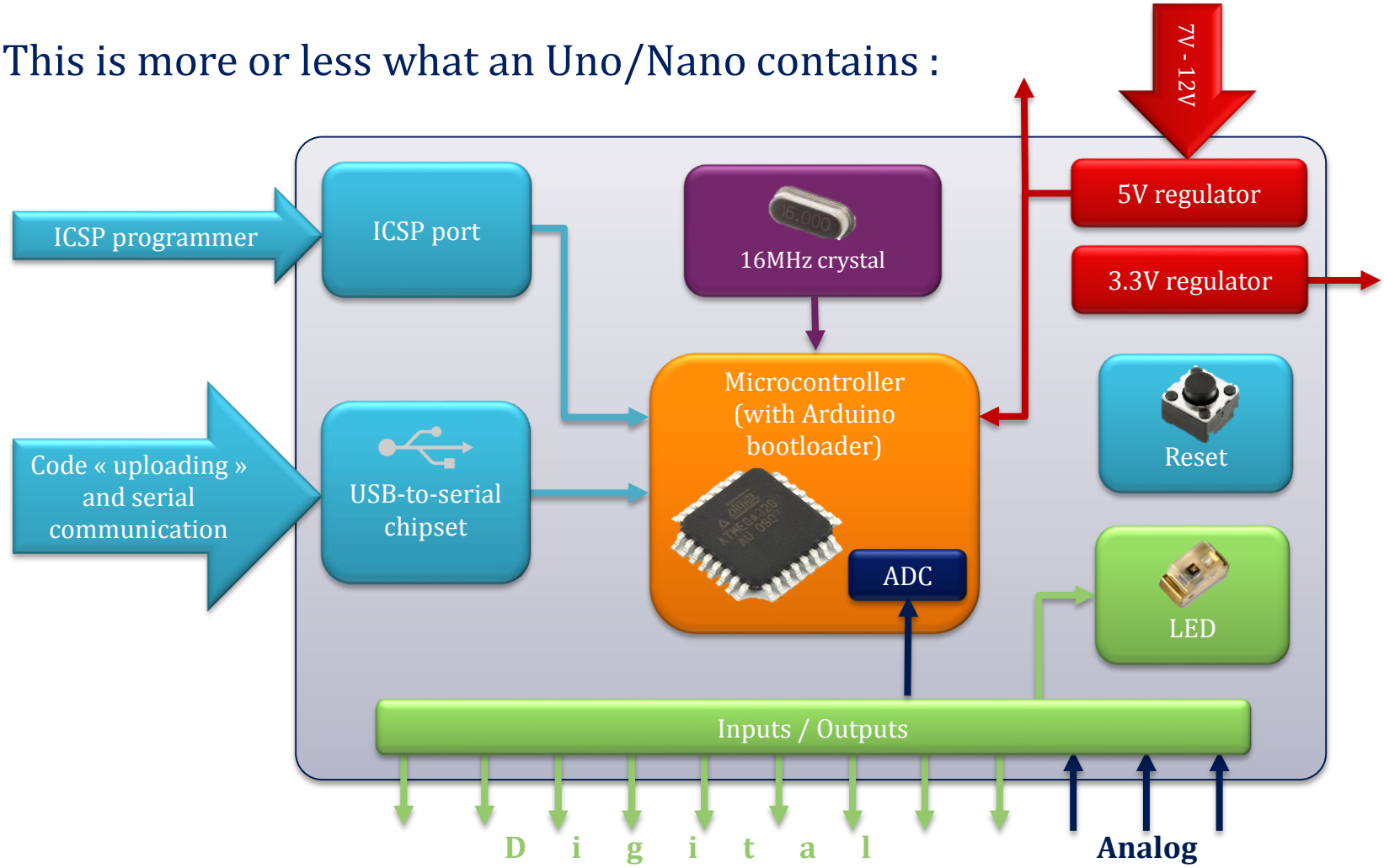
Atmega32u4



Arduino Robot

1) Some electronic boards

This is more or less what an Uno/Nano contains :



1) Some electronic boards

- Microcontroller embedded in most common Arduinos (Uno, Nano, Duemilanove…) :
  - Atmel ATMEGA328P
  - All-purpose, widely used
  - Reliable and reasonably powerful
  - Cheap and easy to find

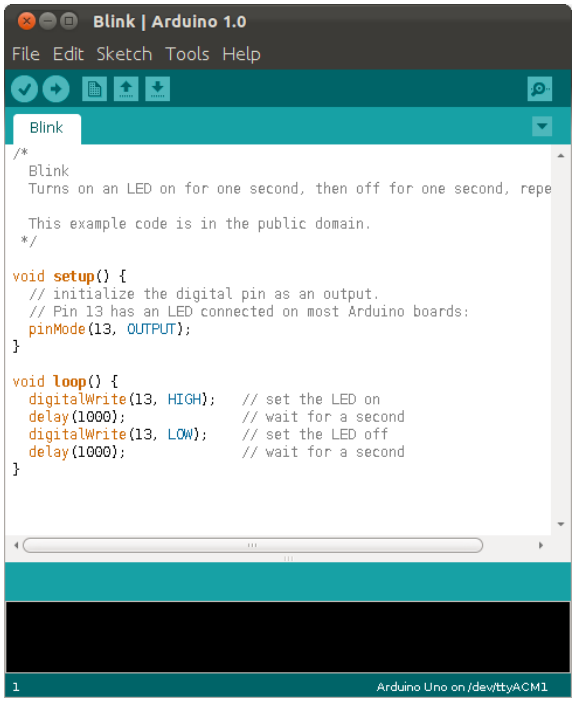
- These ATMEGAs are usually loaded (or « flashed ») using an ICSP programmer. Examples :
  - USBASP : cheap and reliable programmer (<10$ on EBay)
  - "Arduino as ISP" : uses a second Arduino (running a special sketch) as a programmer (see Examples in the Arduino software)

1) Some electronic boards

- Bootloader
  - Specific to each microcontroller model
  - Can be reinstalled on an Arduino if necessary

- "Arduino Bootloader"
  - Basic code installed in the microcontroller in factory (through ICSP)
  - Allows the computer to send some code via a Serial (RS232) port (pins 0=RX and 1=TX), which it will write in Flash memory and execute
  - Can be installed in any brand new, non-Arduino ATMEGA328 on a custom board!

2) A software



Arduino provides a dedicated IDE

(Integrated Development Environment)

- Included features for :
  - Cross-compilation
  - Bootloader connection and code « uploading »
  - Bootloader (re)installation

- `#include <Arduino.h>` automatically prepended during compilation
- Examples and libraries directly accessible

2) A software

STINO v1.2

Using the Arduino is (thankfully) not mandatory. For example, you can use Sublime Text to program on Arduino using the Stino plugin : https://github.com/Robot-Will/Stino

- Install the Package Control plugin https://sublime.wbond.net/installation
- Open the editor's command-line interface (Ctrl+Maj+P), look for "Arduino IDE" and install it
- Restart Sublime Text
- The Arduino menu is now available, offering the same features than the official Arduino IDE

2) A software

You can also use your favorite text editor (such as vim or emacs) then compile and upload your code directly from the command line.

This project provides a nice Makefile to ease this task :

https://github.com/sudar/Arduino-Makefile

3) A framwork

Structure

- setup()
- loop()

Control Structures

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

Further Syntax

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

Arithmetic Operators

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

Variables

Constants

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

Data Types

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

Conversion

- char()
- byte()
- int()

Functions

Digital I/O

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

Analog I/O

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

Due only

- analogReadResolution()
- analogWriteResolution()

Advanced I/O

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

Time

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

Math

- min()
- max()

- Framework : collection of tools and libraries used together to develop a software in a specific context.

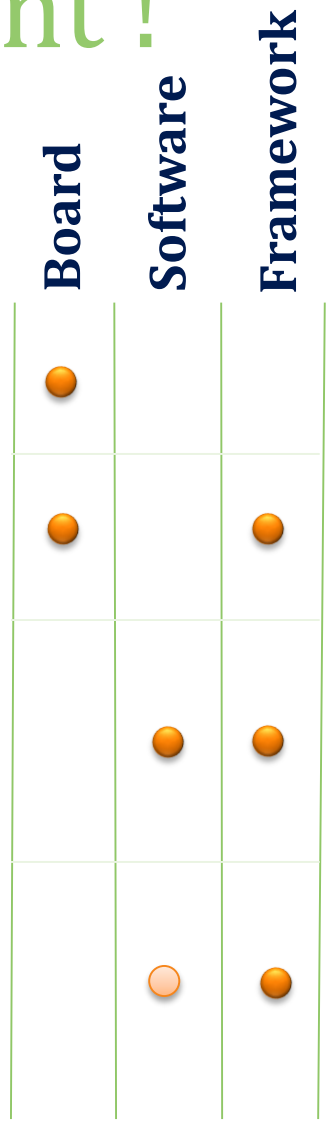
- Examples :
  - Web framework (Symfony, Flask, Wt, Ruby on Rails, …)
  - GUI framework (Qt, GTK, …)
  - Cross-compilation environments (Arduino, Android SDK, …)

- The Arduino Framework :
  - Low-level libraries and drivers for the microcontroller's components : input/output, digital communication, …
  - Tools for cross-compiling and uploading code to the bootloader

The three are independant !

You can...

- Erase the bootloader and program an Arduino board using pure C/C++ or AVR ASM
- Use another IDE/text editor
  - Example : Sublime Text with the Stino plugin
- Use the same microcontroller than on an Arduino but on a custom board, flash the bootloader into it, and use it like an Arduino
- Code with the Arduino IDE but on a different board, with or without the official software
  - Examples : Teensy, RFDuino



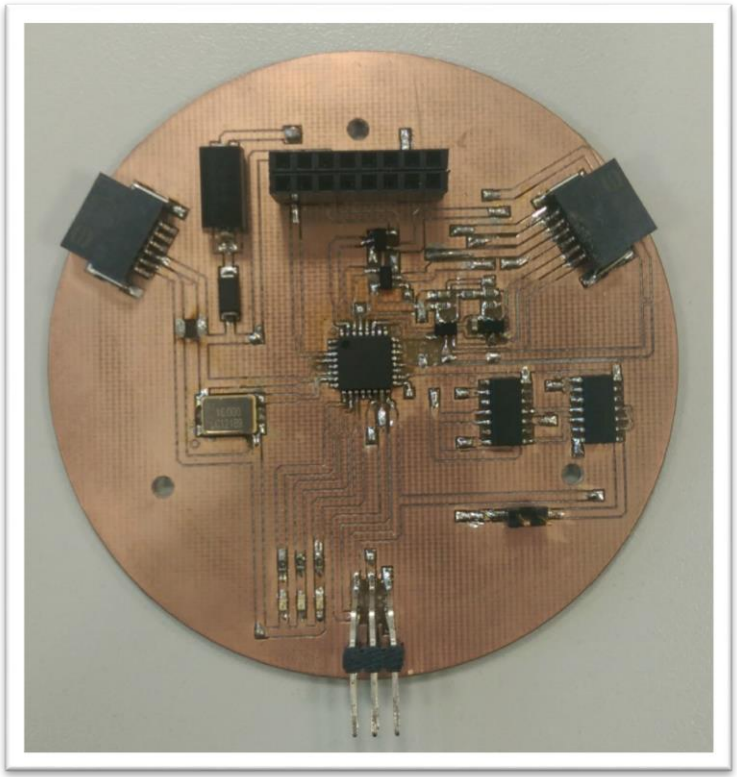
So, Arduino...?

... is an ambiguous name, but in the following, it will designate the central part : the framework.

Example : Artémis

EirSpace's 2nd experimental rocket

- ATMEGA328 microcontroller with 16MHz quartz identical to the ones on the Uno/Nano
- ICSP port to initialize the Atmega with an Uno bootloader
- Rx and Tx pins connected to the main communication bus for external code uploading
- Reset handled with a 3-bit address and NAND gates for multiplexing



Arduino programming

Introduction

Features and code structures

Introduction

- Programmed using C++
- The main() and the framework headers are (unfortunately) hidden by the IDE
- Two mandatory functions :
  - `setup()` : called when the microcontroller initializes
  - `loop()` : called continuously (called back everytime it returns)

Inside, the main() looks like this :

```
void main() {
  setup();
  while (1) {
    loop();
  }
}
```

Features and code structure

Most (if not all) features of C++ are available :

- Functions with default parameters
  - `void myfunction(int arg1, int arg2=42);`
- Dynamic allocation with `new` et `delete` (even though not recommended due to very limited memory resources)
- `bool` type and `true`/`false` constants
- OOP (Object Oriented Programming)
  - Classes
  - Inheritence
- Functions/operators overloading
- Exceptions : `try` / `catch`
- Namespaces
- Templates

Basic structure of an Arduino code

```
void setup() {
      // Code
}

void loop() {
      // Code
}
```

Using GPIOs

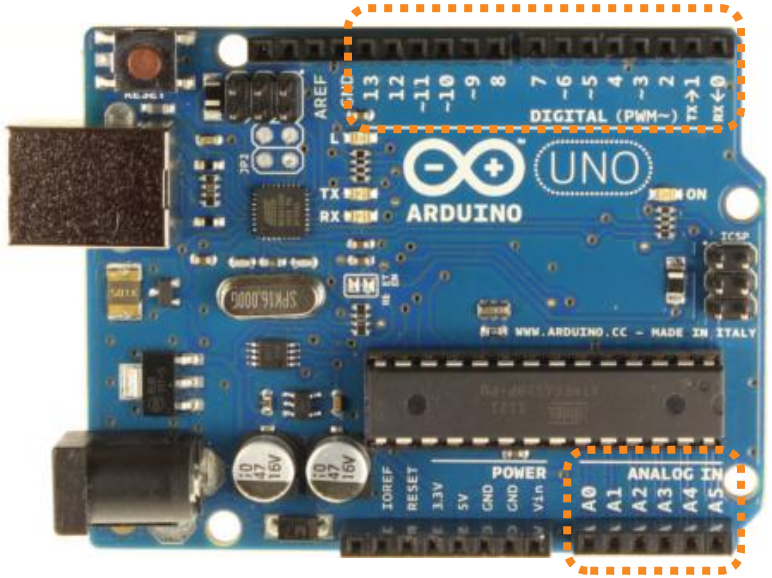
Introduction
Using GPIOs on Arduino
Analog pins and examples

Introduction

- General Purpose Input/Output
- Digital operation
  - Either low (Vout = GND) of high (Vout = Vcc) state
- Either input or output, not both
- Number of pins depends on the microcontroller : 19 on an Atmega328 for example

Using GPIOs on Arduino

- No complicated registers programming, everything is handled by the framework
- Pins numerotation :
  - Digital : 0 through 13
  - Analog : A0 through A5



- A LED is often connected on pin 13

Some basic fonctions

- `pinMode(number, INPUT/OUTPUT/INPUT_PULLUP)`
  - Initializes the given pin either in Input or in Output, usually inside `setup()`
  - Allows the use of an embedded pull-up
  - **Always** initialize the pins you use
  - If `OUTPUT`, **always** specify its initial state using `digitalWrite()` immediately after `pinMode()`

`INPUT` pin	`OUTPUT` pin
`digitalRead(number)`	`digitalWrite(number,LOW/HIGH)`
Returns: ➢LOW if 0 V < U_{in} < 1,2 V ➢HIGH if 3 V < U_{in} < 5 V ➢HIGH or LOW (unreliable) if 1,2V < U_{in} < 3 V	Set the output to either 0V or 5V

Making an LED blink

Hello World!

```
// The LED is connected to the D13 pin
const int PIN_LED = 13;

void setup() {
        // Initialize the LED output to LOW
        pinMode(PIN_LED, OUTPUT);
        digitalWrite(PIN_LED, LOW);
}

void loop() {
        // Turn on
        digitalWrite(PIN_LED, HIGH);

        // Wait 500ms
        delay(500);

        // Turn off
        digitalWrite(PIN_LED, LOW);

        // Wait 500ms
        delay(500);
}
```

Analog operation

- Pins A0 through A5
- Connected to an ADC
  - Analog to Digital Convertor
  - Can be used to read the output of an analog sensor
  - Only work in input
- ADC resolution :
  - 10 bits
  - Values between 0 and 4095
  - Linear image of the voltage ($0 = 0V, 4095 = V_{ref}$)
- These pins can also be used as classic digital GPIOs

Analog operation

- Pins in input :
  - `digitalRead()` is the usual digital operation (boolean)
  - `analogRead()` reads the value on the ADC (int)
- V_{ref} is configurable with `analogReference(DEFAULT/INTERNAL/EXTERNAL)`
  - `DEFAULT` : on-board general 5V
  - `INTERNAL` : internal 1,1V reference, for measuring small voltages
  - `EXTERNAL` : external reference voltage given on the V_{ref} pin

Fonctionnement analogique

- Warning : `analogWrite()` generates a PWM signal
- Not a true analog voltage but a square signal with fixed frequency and configurable duty cycle
- Frequency ~500Hz (depends on the pin)
- Useful when you only care about the mean voltage of the signal
  - example : LED
- Only some compatible pins
  - On Uno and equivalent : 3, 5, 6, 9, 10, et 11

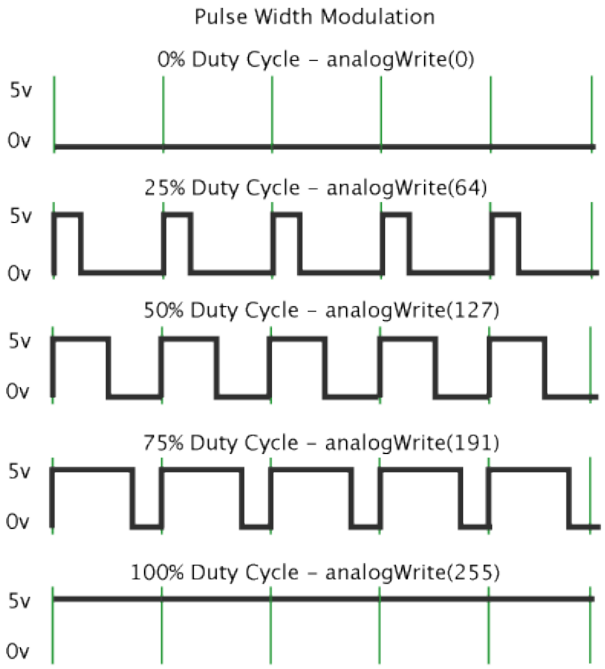


Image source : arduino.cc

Communication buses

Serial
SPI
I2C

3 protocols available

Bus	Serial	SPI	I2C
Type	Direct communication between 2 peripherals	Slave/Master bus	Slave/Master bus
Peripherals number	Always 2	Usually between 2 and 5	Theoretically up to 127
Duplex	Full duplex	Full duplex	Half duplex
Bandwith	Low to medium	High	Medium

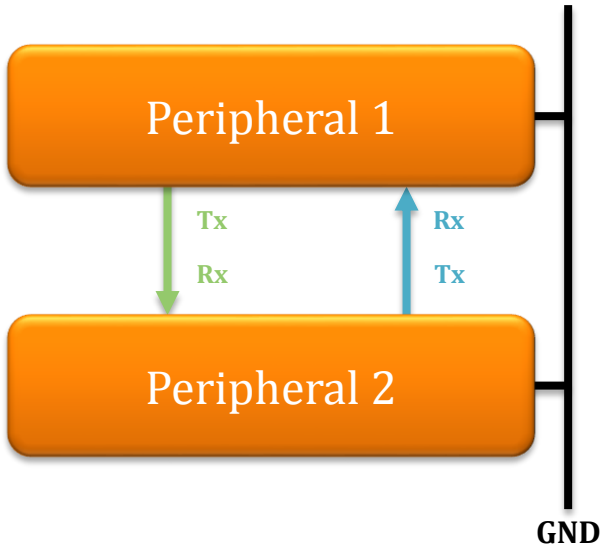
Common API

Common operation for the majority of functions (except SPI) :

- `begin()`: initialize the bus
- `available()`: returns the number of unread bytes on the bus
- `read()`: read a byte
- `write(char value)`: write a byte
- `readBytes(char[] buffer, int n)`: read n bytes and put them into the buffer
- `parseInt()`: read the next bytes and convert into int the number written in ASCII
- `parseFloat()`: same for float
- `flush()`: flush the incoming buffer of unread bytes

Serial

- Peer-to-peer protocol
- Two wires : Tx and Rx (+ GND)
  - Tx : transmitting
  - Rx : receiving
- Tx from one peripheral connected to the Rx on the other peripheral
- Clock speed fixed in advance
  - examples : 9600 bauds*, 38400 bauds, 115200 bauds etc.
- Examples of peripheral using serial :
  - F-Tech FMP04 GPS receiver
  - Xbee
  - Atmega programming with an Arduino bootloader
  - Communication between the microcontroller and the computer

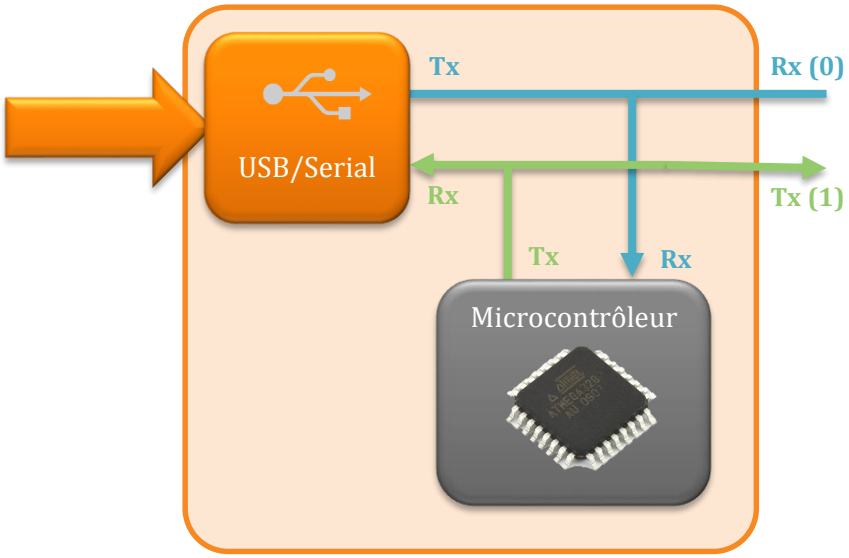


On the usual Arduino boards :
- Rx : pin 0
- Tx : pin 1

* bauds = bits/s

Using Serial with common Arduino boards

- The Atmega328 only has one Serial port, shared between :
  - The embedded USB to serial converter
  - 0 (Rx) and 1 (Tx) pins
- You can't communicate with a Serial peripheral and send data to the computer at the same time
- You can however use an Arduino as a simple USB to Serial converter
  - Put the microcontroller in forced reset (or unplug it from the board)
  - Warning : the Tx written on the board (pin 1) is relative to the microcontroller, therefore it's the Rx for the USB to Serial (see ->)
  - Connect the peripheral's Tx pin to the pin marked "Tx"

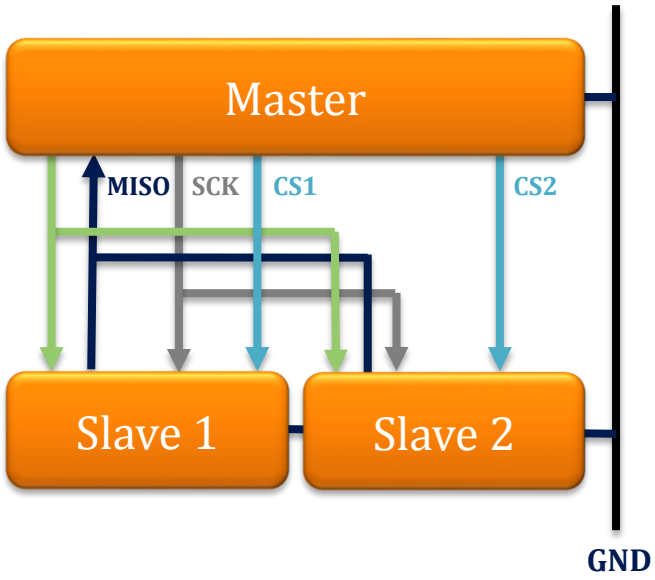


Make an LED blink via Serial

```
const int PIN_LED = 13; // The LED is connected to the pin D13

void setup() {
        // Initialize the LED output
        pinMode(PIN_LED, OUTPUT);
        digitalWrite(PIN_LED, LOW);

        // Initialize the Serial port to 9600 bauds
        Serial.begin(9600);
}
void loop() {
        // If the peripheral has sent data
        if (Serial.available() > 0) {
                // Read a byte
                char commande = Serial.read();

                if (commande == 'H') { // Turn on
                        digitalWrite(PIN_LED, HIGH);

                } else if (commande == 'L') { // Turn off
                        digitalWrite(PIN_LED, LOW);
                }
        }
}
```

SPI

- Master/Slave protocol
- Three fixed wires : MOSI, MISO, SCK + a CS wire per slave (+ GND)
  - MOSI : Master In Slave Out
  - MISO : Master Out Slave In
  - SCK : clock signal (generated by the master)
  - CS : Chip Select (high by default, set to low to select a slave)
- All the MOSI are wired together, all the MISO together, all the SCK together
- The master selects a slave by asserting the corresponding CS pin
- Frequency : usually around 5MHz

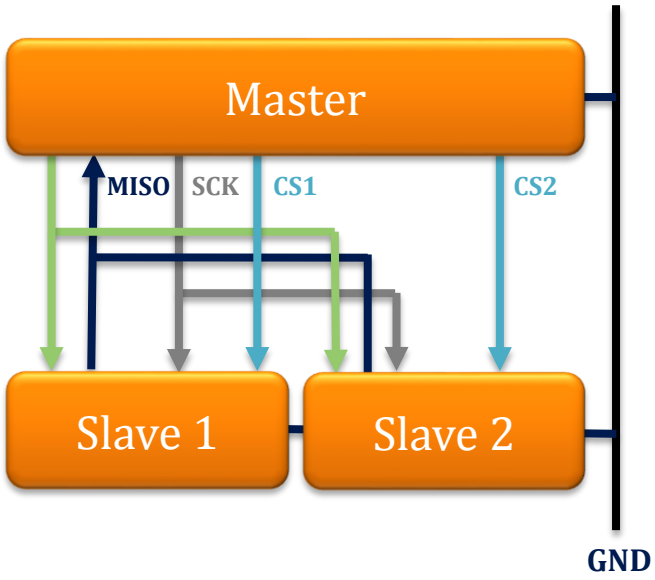


On common Arduino boards :

- MOSI : pin 11
- MISO : pin 12
- SCK : pin 13
- CS : any GPIO configured as OUTPUT

SPI – Examples

- ICSP is based on SPI
- Quick communication between two microcontrollers
- ADC (analog to digital converter) MCP3201
- Lots of digital sensors



On common Arduino boards :

- MOSI : pin 11
- MISO : pin 12
- SCK : pin 13
- CS : any GPIO configured as OUTPUT

Operation of bus communication

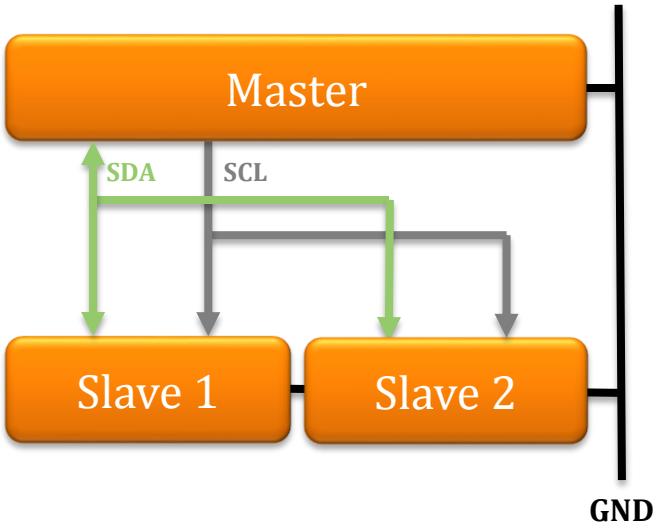
- Example of SPI bus :
  - Send the state of the LED to the peripheral every second
  - Receive in response a command to modify the LED states

```
#include <SPI.h>

// LED : D10, CS of peripheral : D9
const int PIN_LED = 10;
const int PIN_CS = 9;

// LED state
char ledState = 'L';

void setup() {
          // Initialize the outputs
          pinMode(PIN_LED, OUTPUT);
          digitalWrite(PIN_LED, LOW);
          pinMode(PIN_CS, OUTPUT);
          digitalWrite(PIN_CS, HIGH);

          // Initialize SPI
          SPI.begin();
}
```

```
void loop() {
          // Activate the peripheral
          digitalWrite(PIN_CS, LOW);

          // Send the current state and receive
          // the response
          char received = SPI.transfer(ledState);

          if (received == 'H') {
                    // Turn on
                    digitalWrite(PIN_LED, HIGH);
          } else if (received == 'L') {
                    // Turn off
                    digitalWrite(PIN_LED, LOW);
          }
          // Deactivate the peripheral
          digitalWrite(PIN_CS, HIGH);

          delay(1000);
}
```

I2C

- Master/Slave protocol
- Two wires : SDA and SCL (+ GND)
  - SDA : data
  - SCL : clock (generated by the master)
- All the SDA are wired together, all the SCL wired together
- All the slaves have an hard-coded address on 7 bits
- The master send frames on SDA with a header indicating the slave it wants to communicate with
- The selected slave answers on the same line (half-duplex)
- Frequency : around 100 kHz to 400kHz
- Library called Wire on Arduino



Sur les cartes Arduino courantes :
- SDA : pin A4
- SCL : pin A5

I2C – Examples

- Used on lots of digital sensors
- Every second, the master ask the peripheral for data and displays it

```
// Master code
#include <Wire.h>

void setup() {
          // No argument to start
          // in master
          Wire.begin();
          Serial.begin(9600);
}

void loop() {
          Wire.request(42, 8);

          while (Wire.available()) {
                    char c = Wire.read();
                    Serial.print(c);
          }

          delay(1000);
}
```

```
// Peripheral code
#include <Wire.h>

void setup() {
          // Start in slave with address 42
          Wire.begin(42);
          // Register a function to call when
          // a request arrives (callback)
          Wire.onRequest(i2cHandler);
}

void loop() {
          // Do nothing
          delay(1000);
}

void i2cHandler() {
          // Answers
          Wire.write("EirSpace");
}
```

Tips and tricks

Asynchronous wait

Bits vectors

Interrupts

Asynchronous wait

- `delay(int ms)` and `delayMicroseconds(int us)` wait *synchronously* : the execution is blocked
- It is often useful to wait *asynchronously* : wait for a fixed time, but still continue to execute something else
- For this, use the `millis()` function in order to save the starting time and at each turn of the loop, compare to a new call to `millis()`

Asynchrone wait - Example

Turn the LED on after 5 seconds

```
// Synchronous wait

const int PIN_LED = 13;

void setup() {
          // Initialize the output
          pinMode(PIN_LED, OUTPUT);
          digitalWrite(PIN_LED, LOW);

          // Wait 5 seconds
          delay(5000);

          // Turn the LED on
          digitalWrite(PIN_LED, HIGH);

}

void loop() {
          // During the 5 seconds, this
          // is not executed

          // Code…
}
```

```
// Asynchronous wait

const int PIN_LED = 13;
long temps_depart = 0; // Always use a long

void setup() {
          // Initialize the output
          pinMode(PIN_LED, OUTPUT);
          digitalWrite(PIN_LED, LOW);

          // Save the current time
          start_time = millis();
}

void loop() {
          if (start_time >= 0
              && millis() > start_time + 5000) {
                    digitalWrite(PIN_LED, HIGH);
                    start_time = -1;
          }

          // This code is executed during the wait

          // Code…
}
```

Bits vectors

- Bits vectors : int where only the bits taken independently have a meaning
- Exemple : a register where each bit indicates the state of a corresponding Input GPIO pin
- The standard Arduino library provides functions for handling bits vectors, easier than with usual arithmetic operations :
  - `bitRead(int register, int n)` : return the nth bit of the register
  - `bitWrite(int register, int n, int value)` : writes value (0 or 1) in the nth bit of the register
  - `bitSet(int register, int n)` : equivalent to bitWrite(register, n, 1)
  - `bitClear(int register, int n)` : equivalent to bitWrite(register, n, 0)
  - `lowByte(int register)` : return the low byte of the register
  - `highByte(int register)` : return the high byte of the register

Interrupts

- Allows interruptions only on a specific subset of pins (only D2 and D3 on boards powered by an Atmega328)
- In order to register a function to call when the interruption is triggered (callback), use :
  `attachInterrupt(int number, fonction, LOW/CHANGE/RISING/FALLING)`
- Warning : `number` is the interrupt number : 0 for D2 and 1 for D3
- `detachInterrupt(int number)` remove the callback
- `noInterrupts()` temporarily disable all interrupts (useful for critical code sections)
- Use `interrupts()` in order to re-enable them