

Informatique Industrielle

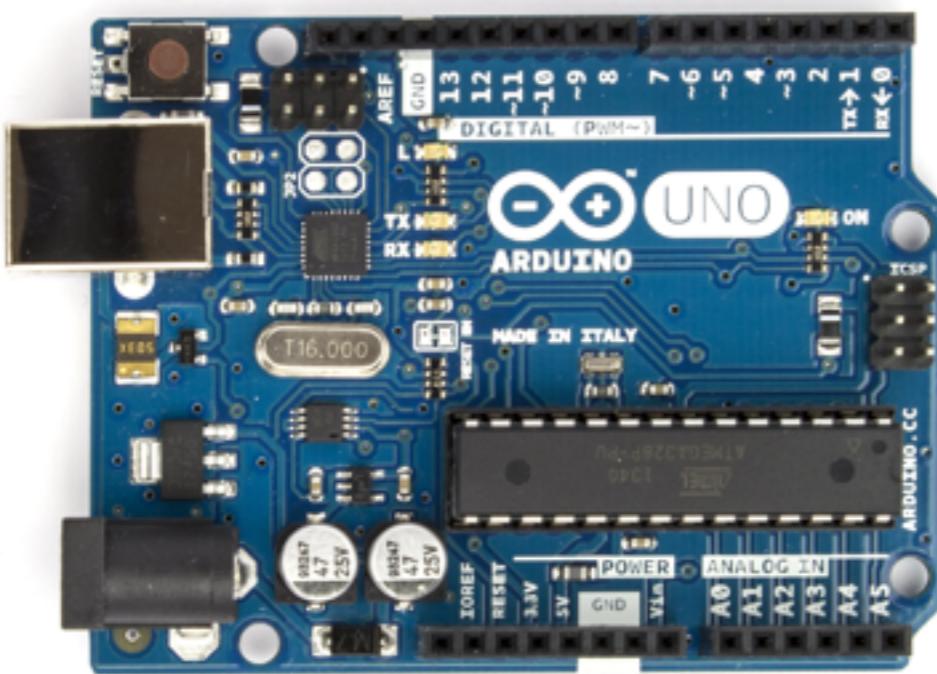
Introduction Arduino

Michaël Lauer
Université de Toulouse
LAAS-CNRS

Le projet Arduino

Plateforme de prototypage cyber-phérique en source libre

- Cartes électroniques développées autour d'un microcontrôleur
- Environnement de développement



The screenshot shows the Arduino IDE interface with the title bar "ask_transmitter | Arduino 1.0.5". The main window displays the following C++ code:

```
// ask_transmitter.pde
// -*- mode: C++ -*-
// Simple example of how to use RadioHead to transmit messages
// with a simple ASK transmitter in a very simple way.
// Implements a simplex (one-way) transmitter with an TX-C1 module

#include <RH_ASK.h>
#include <SPI.h> // Not actually used but needed to compile

RH_ASK driver;

void setup()
{
    Serial.begin(9600); // Debugging only
    if (!driver.init())
        Serial.println("init failed");
}

void loop()
{
    const char *msg = "hello";

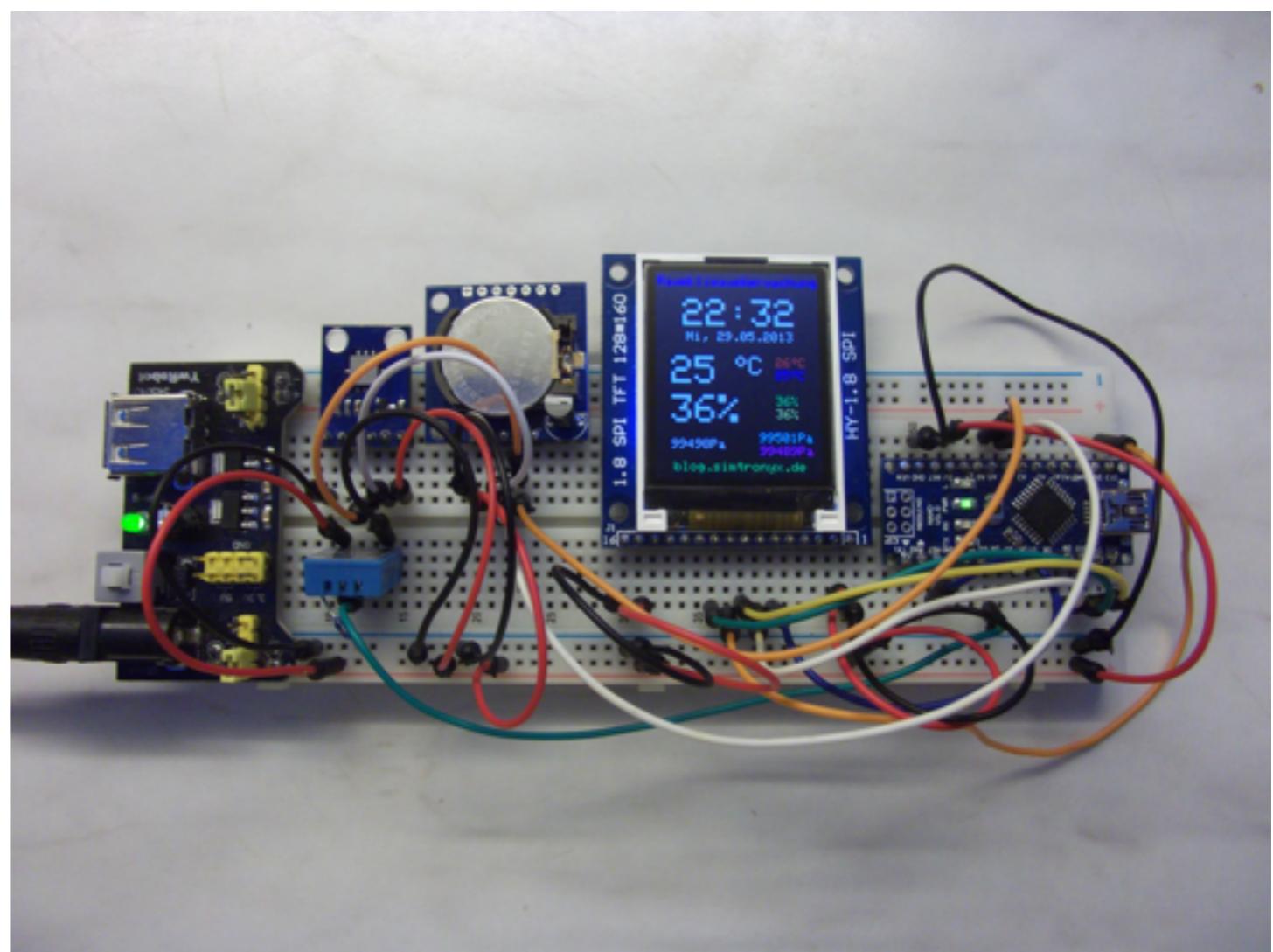
    driver.send((uint8_t *)msg, strlen(msg));
    driver.waitPacketSent();
    delay(200);
}
```

At the bottom of the screen, a status bar indicates "Arduino Uno on /dev/tty.usbmodem1411".

Le projet Arduino

Plateforme de **prototypage** cyber-physique en source libre

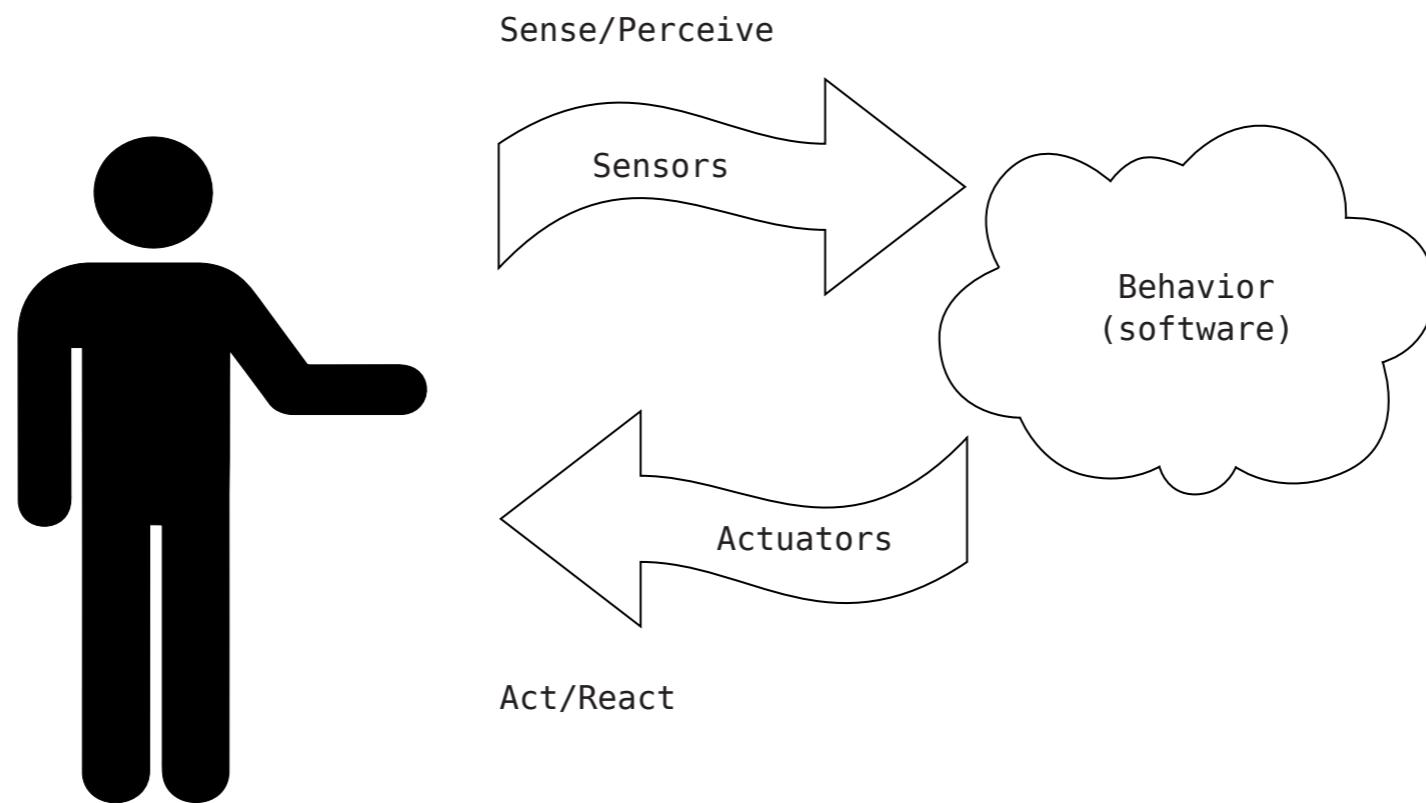
- Utilisé pour la mise au point de produits mais pas pour la production
- Développement rapide
- Programmation simplifiée



Le projet Arduino

Plateforme de prototypage **cyber-physique** en source libre

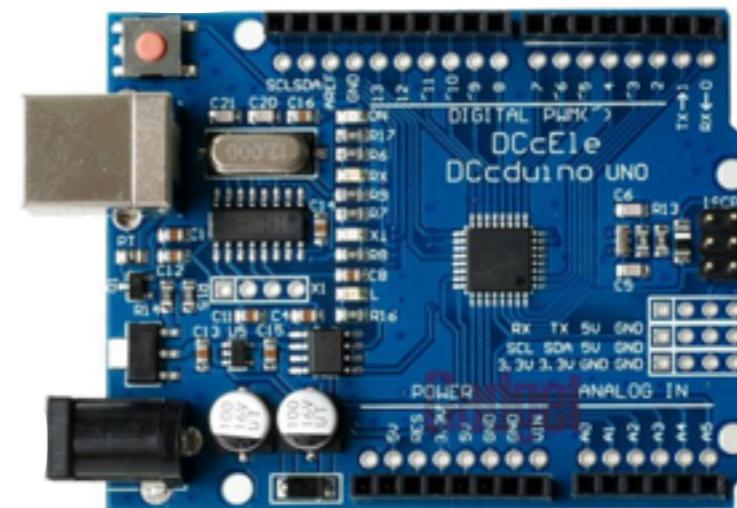
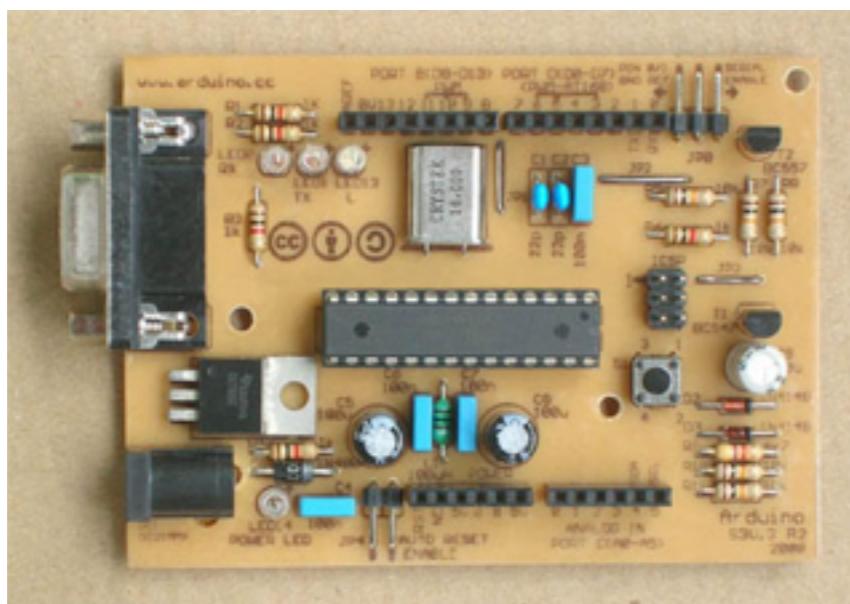
- Interaction avec son environnement physique
- Nombreuses Entrées/Sorties pour collaboration avec Capteurs/Actionneurs



Le projet Arduino

Plateforme de prototypage cyber-physique en **source libre**

- Tout le code est disponible et réutilisable librement (environnement de développement, bootloader, exemples,...)
 - Tous les schéma électroniques des cartes sont disponibles et réutilisables librement => possibilité de construire sa propre carte, apparition de clone compatible à bas coût (à partir de 2,5 euros)
 - Communauté très active et ouverte, nombreux exemples de projets sur internet



Exemple de réalisation

Surveillance de plante



Exemple de réalisation

Robot grimpeur d'arbres



Exemple de réalisation

Contrôleur de vol pour drone



Exemple de réalisation

Contrôle d'un moteur de fusée expérimental



<https://github.com/gNSortino/Arduino-Liquid-Engine-Software>

Différents formats de carte

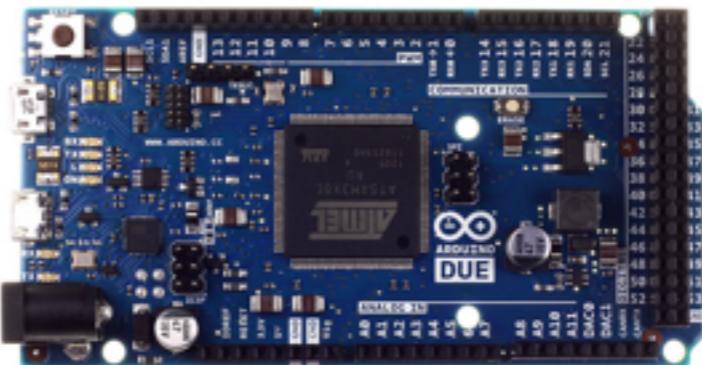
- Différence entre les microcontrôleurs utilisés, la connectique, les I/O,...
- Toujours la même interface de programmation



Arduino Uno

MCU ATmega328

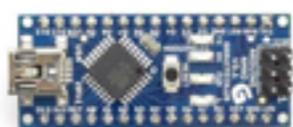
16 bits, 16 MHz, 32 Ko de mémoire
14 I/O numériques, 6 entrées analogiques 10 bits



Arduino Due

MCU Arm Cortex M3

32 bits, 84 MHz, 512 Ko de mémoire
54 I/O numériques, 12 entrées analogiques 12 bits



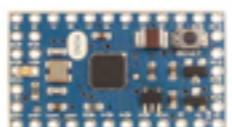
Arduino Nano

MCU ATMega328

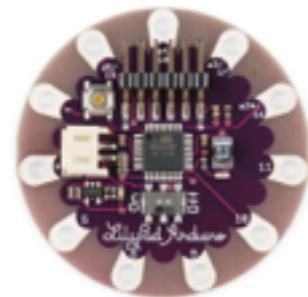
et beaucoup d'autres...



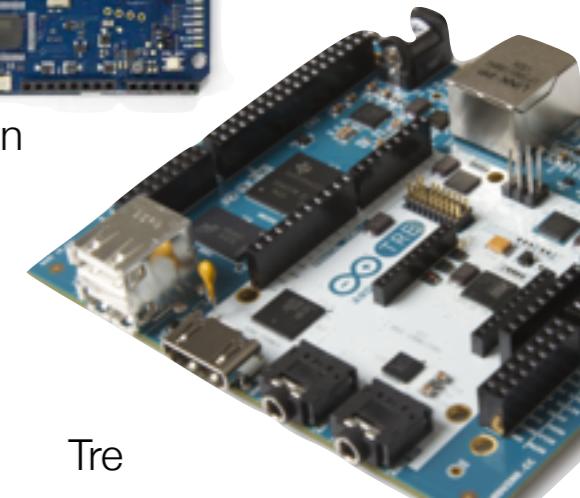
Yún



Mini



LilyPad



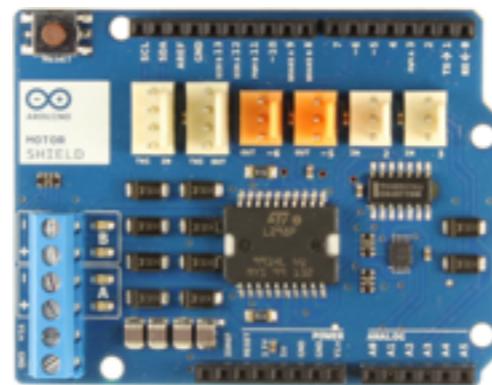
Tre

Cartes d'extension (shields)

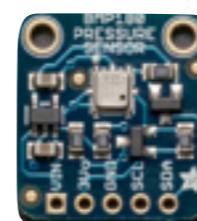
- De nombreuses cartes d'extension compatible Arduino sont disponibles
- *Compatible* : position des broches + librairie logicielle
- Concepteurs : Snootlab, Sparkfun, ChipKit, Adafruit, ...
- Ajout de fonction GSM, Ethernet, Contrôle de moteur, capteurs, actionneurs,...



Platine GSM



Contrôle Moteur

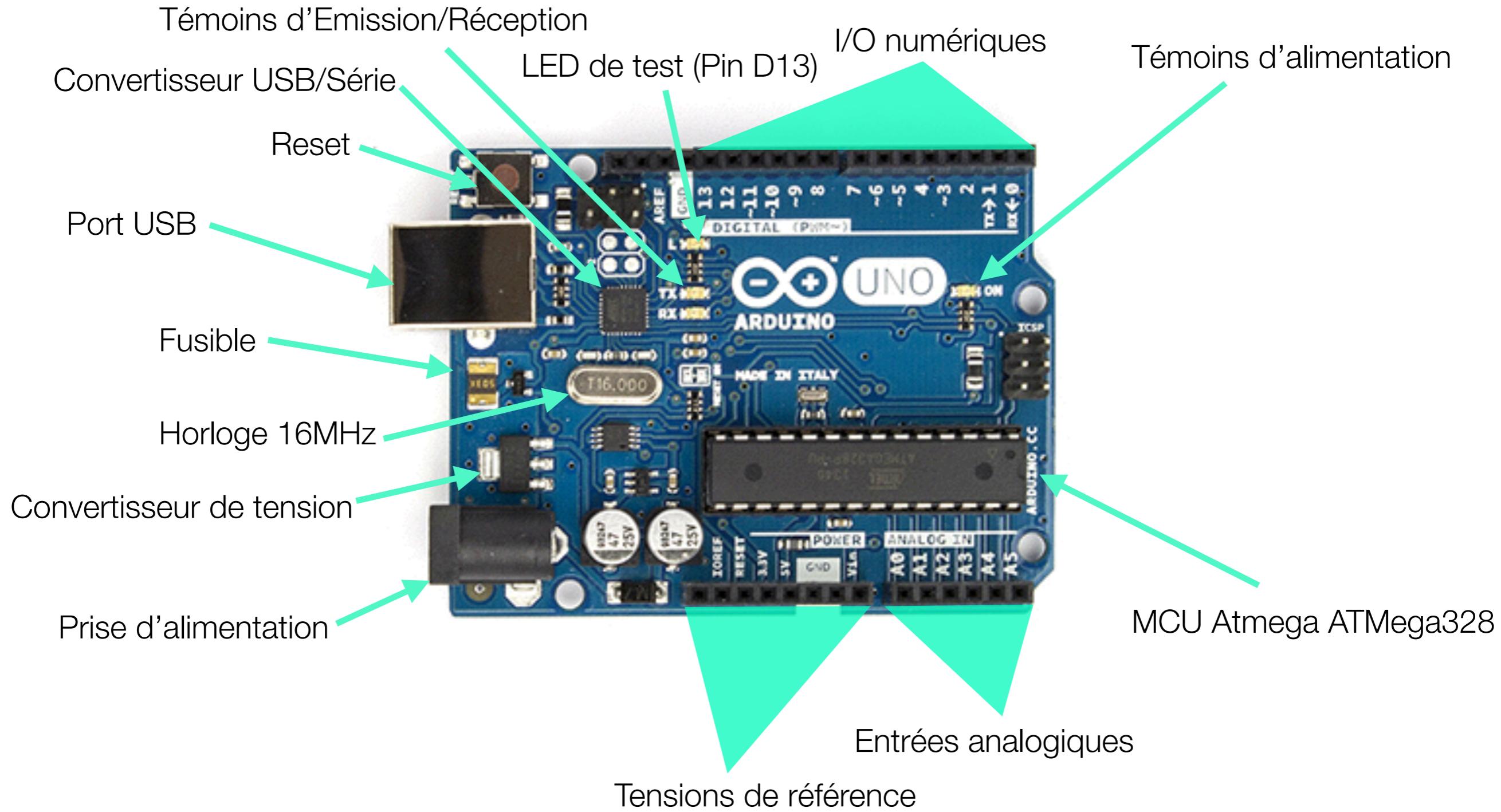


Capteur de pression



Potentiomètre

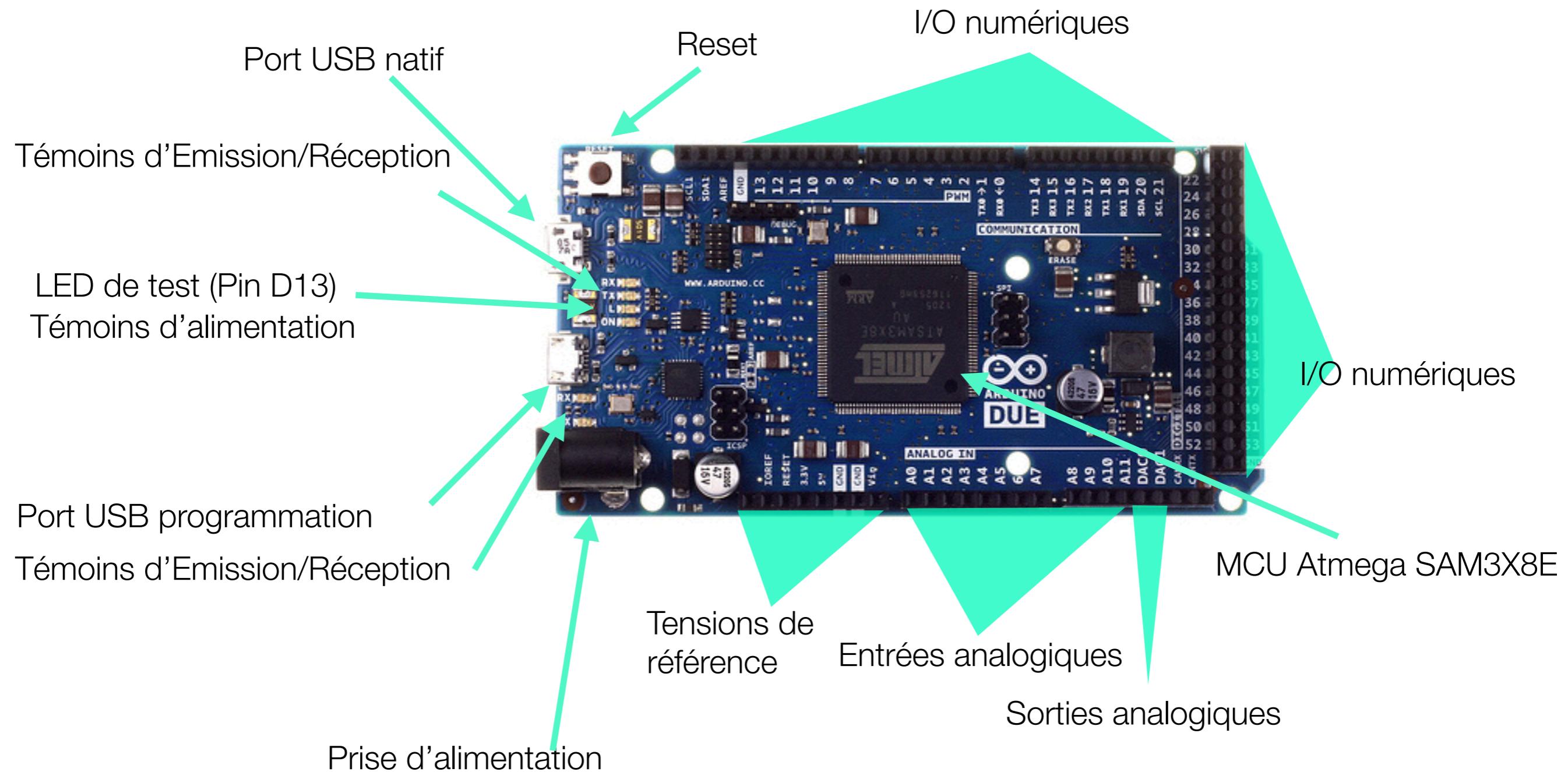
Arduino Uno



Arduino Uno

- Alimentation par le port USB ou la prise d'alimentation (de 7V à 12V)
- Tension de fonctionnement (niveau logique) : 5V
- Courant disponible sur les broches d'E/S : 40 mA
- Port USB a deux fonctions :
 - interface pour charger le code dans la mémoire du microcontrôleur
 - liaison série lors de l'exécution

Arduino Due



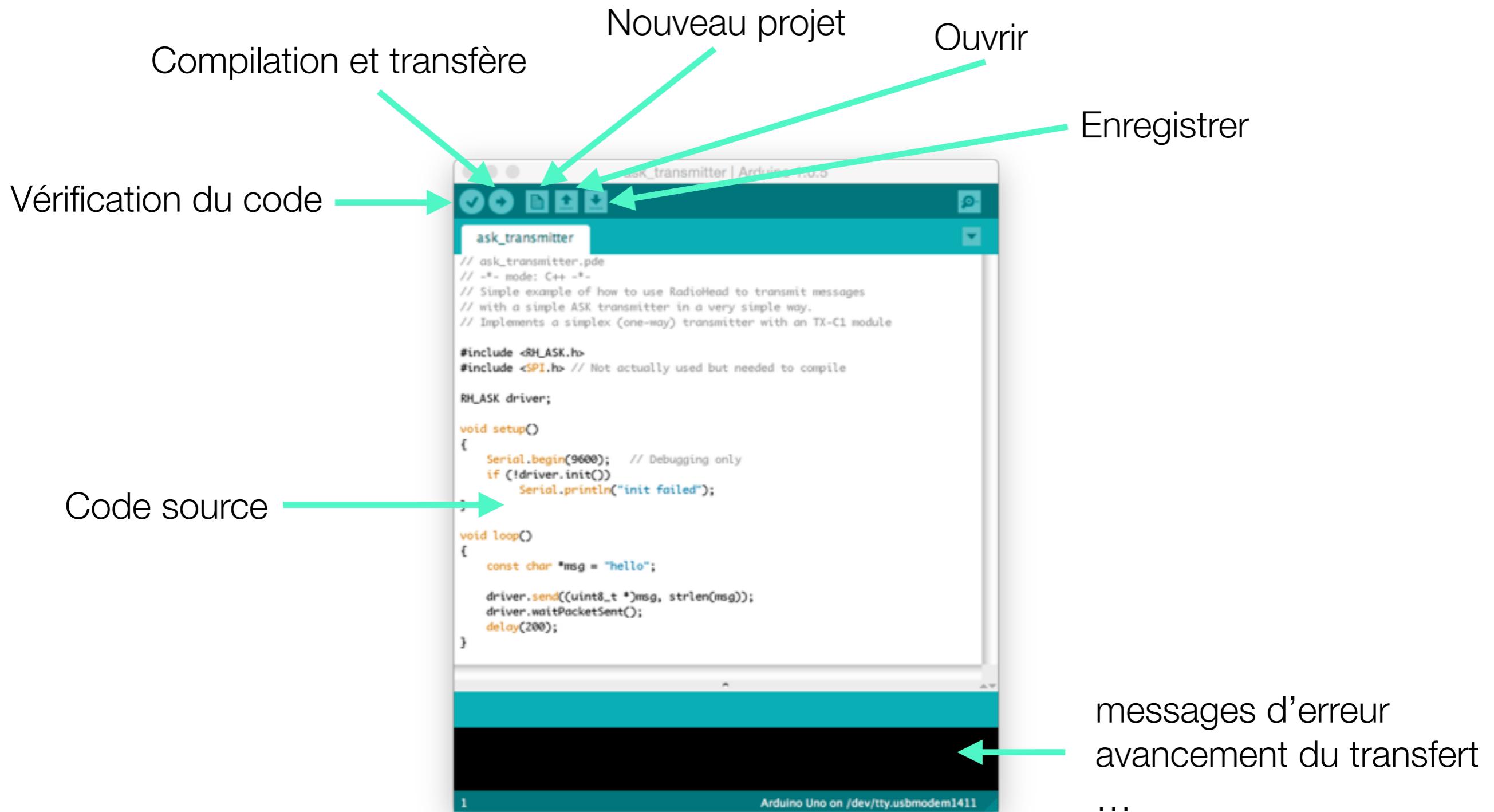
Arduino Due

- Alimentation par le port USB ou la prise d'alimentation (de 7V à 12V)
- Tension de fonctionnement (niveau logique) : **3,3V**
- Courant disponible sur pour toutes les broches d'E/S : 130 mA
- Port USB « programmation » (idem arduino Uno):
 - interface pour charger le code dans la mémoire du microcontrôleur
 - liaison série lors de l'exécution
- Port USB natif :
 - directement relié au microcontrôleur
 - permet communication Série sur USB ou connection de périphérique USB (clavier, sourie,...)

Environnement de développement Arduino

- Tout ce fait au travers de l'IDE (Integrated Development Environment) Arduino
 - Ecriture du code
 - Compilation
 - Chargement sur la carte
 - Monitoring (pas de fonction debug avancée)
- Multi-platorme (OS X, Linux, Windows)
- Open-source (réutilisation d'éléments du projet *Processing*)
- Language C++ avec des fonctions de haut niveau pour simplification et abstraction du matériel (pas de fonction *main*, pas de manipulation explicite du matériel, ...)
- La programmation directe du microcontrôleur est possible (accès au registre)
- Les programmes arduino sont appelés des *sketchs*

Environnement de développement Arduino



Exemple de sketch Arduino

Un sketch est composé au minimum de deux fonctions :

- `setup()` : fonction d'initialisation exécutée une unique fois au démarrage de la carte
- `loop()` : fonction exécutée après l'initialisation puis en boucle tant que la carte est alimentée

```
// the setup function runs once when you press reset or power the board
void setup() {
    pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

Que fait ce sketch?

Exemple de sketch Arduino

Avant la compilation l'IDE ajoute automatiquement au code source :

- un header contenant la déclaration de toutes les fonctions haut niveau Arduino (*digitalWrite()*, *pinMode()*, ...):

```
#include "WProgram.h"
```

- la fonction *main()* appelant *setup()* et *loop()*

```
int main(void)
{
    init(); ←
    setup();

    for (;;)
        loop();

    return 0;
}
```

fonction prédéfinie pour
l'initialisation du hardware

Entrée/Sortie Numérique

3 fonctions pour programmer les entrées/sorties numériques :

- `pinMode(pin, mode)` déclare que la broche *pin* fonctionne en *mode INPUT* ou *OUTPUT*
- `digitalRead(pin)` retourne l'état de la broche *pin* (LOW ou HIGH)
- `digitalWrite(pin, value)` met à l'état LOW ou HIGH la broche *pin*

Les broches d'entrées analogiques peuvent également servir d'E/S numériques. Elles sont alors numérotées A0,A1,... pour les distinguer des broches numériques

Mettre une broche en entrée à l'état haut active une résistance de tirage (pull-up) de $20\text{k}\Omega$ à $150\text{k}\Omega$ en fonction des microcontrôleurs

Entrée/Sortie Numérique : Exemple

```
int ledPin = 13; // LED connected to digital pin 13
int inPin = 7;   // pushbutton connected to digital pin 7
int val = 0;     // variable to store the read value

void setup()
{
    pinMode(ledPin, OUTPUT);          // sets the digital pin 13 as
output
    pinMode(inPin, INPUT);           // sets the digital pin 7 as input
}

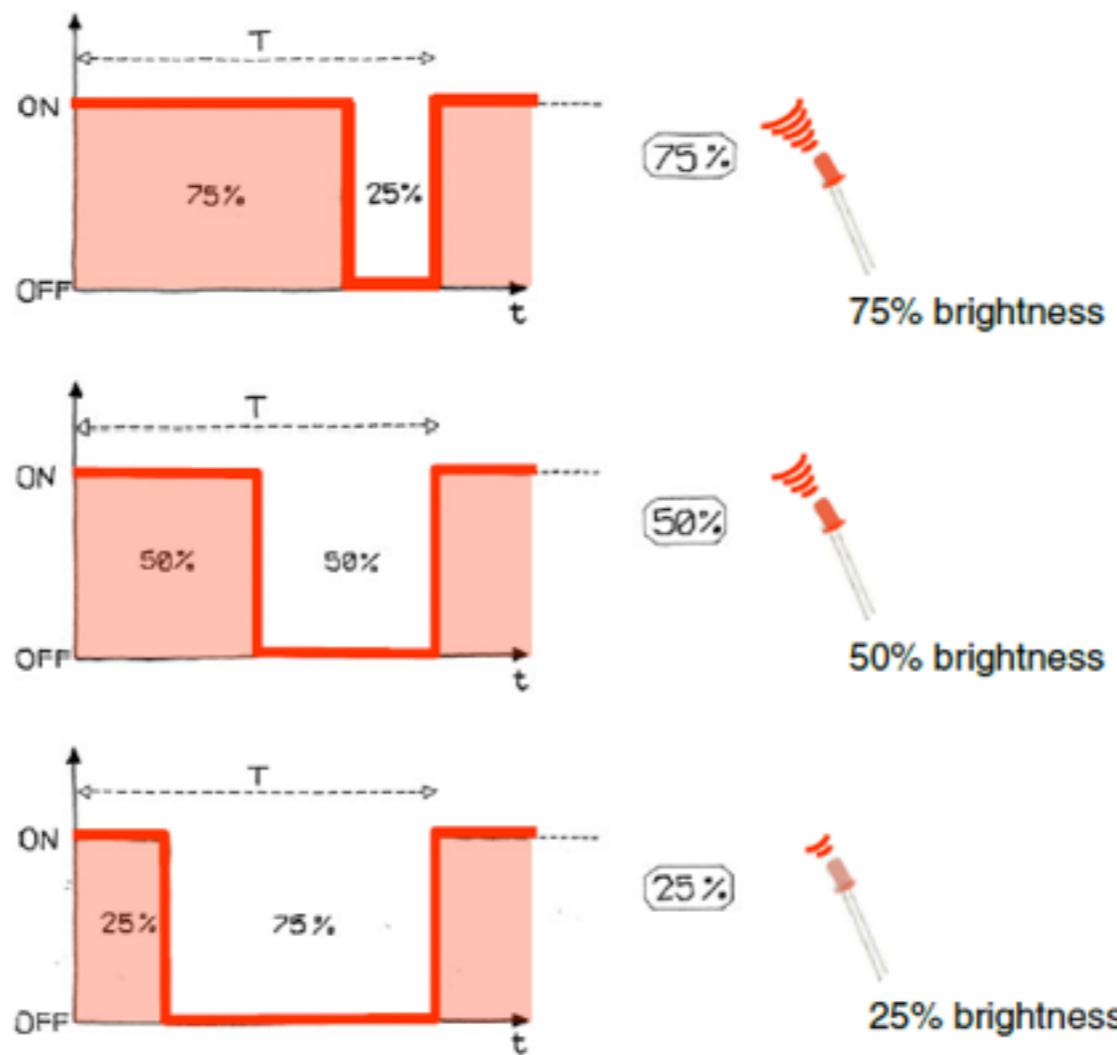
void loop()
{
    val = digitalRead(inPin);        // read the input pin
    digitalWrite(ledPin, val);       // sets the LED to the button's
value
}
```

Entrée Analogique : analogRead()

- Les broches A0, A1, ... sont équipées uniquement de convertisseurs analogique/numérique. Pas besoin de configurer le sens de fonctionnement.
- `analogRead(pin)` retourne un entier de 0 à 2^n , avec n la résolution du convertisseur
- **Par défaut la résolution est de 10 bits.**
- Avec l'Arduino Due, on peut monter la résolution jusqu'à 12 bits avec la fonction `analogReadResolution(bits)` (change la résolution de toutes les entrées analogiques de la carte)

Sortie Analogique : analogWrite()

- Sauf exception (Arduino Due) les cartes Arduino n'ont pas de convertisseur numérique/analogique
- Emulation d'une sortie analogique par un signal PWM transmis sur une sortie numérique



Sortie Analogique : analogWrite()

- La fonction `analogWrite()` n'a donc pas de lien avec les broches analogiques, ni la fonction `analogRead()`
- **`analogWrite(pin, value)`**
 - pin : la broche numérique de sortie (doit être configurée en sortie avant l'écriture)
 - value : un entier de 0 à 255 décrivant le rapport cyclique du signal (0 => 0%, 255 => 100%)
- Broches avec capacité PWM :
 - Arduino UNO : 3, 5, 6, 9, 10, 11
 - Arduino DUE : 2 à 13
- Arduino Due :
 - Deux convertisseurs numérique/analogique (DAC0 et DAC1). Limitation matériel sur la plage de valeur atteignable : 0,55V à 2,75V
 - réglage de la résolution des écritures analogiques (PWM et DAC) avec **`analogWriteResolution(bits)`**

Communication série : Serial

- Il est souvent nécessaire de communiquer avec la carte durant son fonctionnement (debug, monitoring, traitement déporté des mesures, envoie de commandes,...)
- Le moyen le plus simple est d'établir une communication série au travers du port usb de la carte
- L'IDE dispose d'un moniteur série pouvant transmettre et recevoir des messages
- L'émetteur et le récepteur doivent avoir des débits identiques

Communication série : connection

- **Serial.begin(speed)** configure la liaison série à la vitesse speed en bits par seconde.
- Connection asynchrone
- Configuration par défaut : données de 8 bits, sans parité, 1 bit de stop
- **Serial.begin(speed, config)** permet de régler la configuration
 - Valeur de config choisie parmi des constantes notées : SERIAL_xyz
 - x : nombre de bits de données
 - y : E pour activé parité, N sinon
 - z : nombre de bits de stop
 - exemple : SERIAL_5E2 => données de 5 bits, avec parité et 2 bits de stop
- **Serial.end()** pour fermer la liaison

Communication série : émission

- **Serial.print(data)** formate les données envoyées au format ASCII => nombre transmis comme des caractères
 - Serial.print(78) sera affiché « 78 »
 - Serial.print(1.234) sera affiché « 1.23 » (par défaut 2 chiffres après la virgule sont retenus pour les flottants)
 - Serial.print(«Hello World ») est transmis tel quel.
- **Serial.println(data)** ajoute automatiquement un caractère de retour à la ligne
- Serial.print(data,format) permet de choisir la mise en forme, par exemple :
 - Serial.print(78, BIN) sera affiché « 1001110 »
 - Serial.print(1.234, 1) sera affiché « 1.2 »
- Pour envoyer des données sans formatage on peut utiliser **Serial.write()**

Communication série : réception

- **Serial.available()** retourne le nombre de caractères (octets) disponible dans le port de réception
- **Serial.read()** retourne le premier octet du port de réception

```
char 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);
    }
}
```

Gestion du temps

- **delay(ms)** impose une attente de ms millisecondes
- **millis()** retourne un entier long non signé contenant le temps en ms passé depuis le dernier démarrage de la carte
- **delayMicroseconds(us)** impose une attente de us microsecondes
- **micros()** retourne un entier long non signé contenant le temps en us passé depuis le dernier démarrage de la carte (dépassement du compteur en 70 minutes environ) .

Interruption externe

- Permet de réagir à un événement sans faire du questionnement périodiquement
- Meilleure réactivité, économie d'énergie
- Arduino Uno : 2 lignes d'interruption 0 et 1 câblées sur les broches numériques 2 et 3 respectivement.
- Arduino Due : **toutes** les broches peuvent servir de ligne d'interruption
- **attachInterrupt(interrupt, ISR, mode)**
 - interrupt : numéro de la ligne d'interruption
 - ISR : fonction traitant l'interruption
 - mode :
 - LOW -> déclenchement si la broche est à l'état bas
 - CHANGE -> déclenchement sur changement d'état
 - RISING -> déclenchement sur front montant
 - FALLING -> déclenchement sur front descendant
 - (Due seulement) HIGH -> déclenchement si la broche est à l'état haut

Interruption externe

- la fonction de traitement doit être aussi succincte que possible pour ne pas interférer avec le reste du code
- traitement d'une seule interruption à la fois => perte des interruptions concurrentes
 - les fonctions de gestion de temps delay(), millis() ne fonctionnent pas pendant le traitement de l'interruption
 - possibilité de perte des messages series reçus durant le traitement
- Pour garantir qu'une variable est correctement mise à jour dans une fonction d'interruption, il faut la déclarer avec le prédictat « **volatile** » :

```
volatile int state = LOW;
```

Interruption externe : exercice

- On se propose de synchroniser l'allumage de la LED L de la carte (pin 13) avec l'état d'un interrupteur. Proposer un schéma de montage ainsi qu'un sketch arduino pour réaliser cette fonction.

Interruption externe : exercice

```
int pin = 13;
volatile int state = LOW;

void setup()
{
    pinMode(pin, OUTPUT);
    attachInterrupt(0, blink, CHANGE);
}

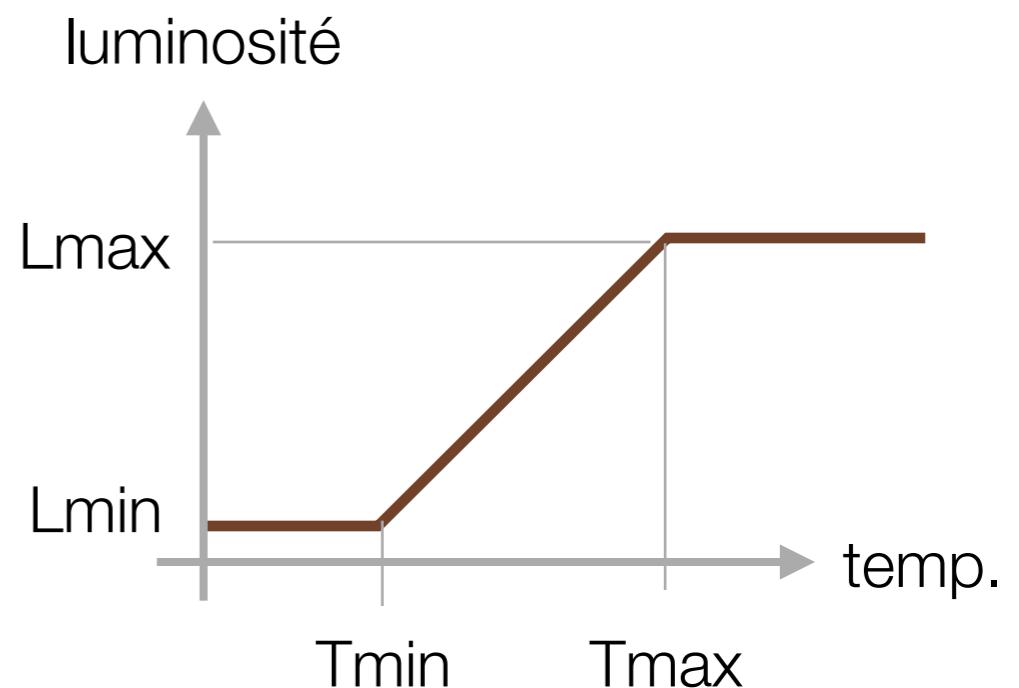
void loop()
{
    digitalWrite(pin, state);
}

void blink()
{
    state = !state;
}
```

Exercice de synthèse

On se propose d'utiliser un Arduino pour asservir la luminosité d'une led à un capteur de température (voir figure). L'Arduino devra transmettre au moniteur série la température mesurée. La vitesse de transmission sera 9600 baups. Le capteur est étalonné de sorte que 0°C correspond à 0V en sortie. La tension croît linéairement de 10mV par $^{\circ}\text{C}$

- Donner un schéma bloc fonctionnel du système en précisant les unités des grandeurs transmises entre chaque bloc.
- Définir les fonctions de conversion qui devront être implémentées dans l'arduino
- Ecrire le sketch arduino correspondant.
On prendra pour valeur $L_{\min} = 0\%$, $L_{\max} = 100\%$, $T_{\min} = 18^{\circ}\text{C}$ et $T_{\max} = 28^{\circ}\text{C}$
- On souhaite désormais pouvoir modifier la valeur des seuils depuis le moniteur série. Modifier le code en fonction.



Arduino et performance

La simplification du modèle de programmation limite les performances du microcontrôleur. Ecriture numérique peut être jusqu'à 10x plus lente.

```
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;                                Protection du matériel

    // If the pin that support PWM output, we need to turn it off
    // before doing a digital write.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);                Protection du matériel

    out = portOutputRegister(port);                               Récupération de l'état actuel du port

    uint8_t oldSREG = SREG;
    cli();                                                       Blocage des interruptions

    if (val == LOW) {
        *out &= ~bit;
    } else {
        *out |= bit;
    }

    SREG = oldSREG;                                              Rétablissement des interruptions
}
```

The annotations highlight several inefficiencies in the `digitalWrite` function:

- Annotations point to the initial variable declarations (`uint8_t`) and the `if (port == NOT_A_PIN)` check, labeled "Récupération des adresses pour programmation de la broche" (Address recovery for programming the pin).
- Annotations point to the `if (timer != NOT_ON_TIMER)` check, labeled "Masque pour écriture" (Mask for writing).
- An annotation points to the `if (port == NOT_A_PIN)` check, labeled "Protection du matériel" (Hardware protection).
- An annotation points to the `turnOffPWM(timer);` call, labeled "Protection du matériel" (Hardware protection).
- An annotation points to the `out = portOutputRegister(port);` assignment, labeled "Récupération de l'état actuel du port" (Recovery of the current state of the port).
- An annotation points to the `cli();` call, labeled "Blocage des interruptions" (Interrupt blocking).
- An annotation points to the conditional assignments (`*out &= ~bit;` and `*out |= bit;`), labeled "Changement de la valeur" (Value change).
- An annotation points to the `SREG = oldSREG;` assignment, labeled "Rétablissement des interruptions" (Restoration of interruptions).