

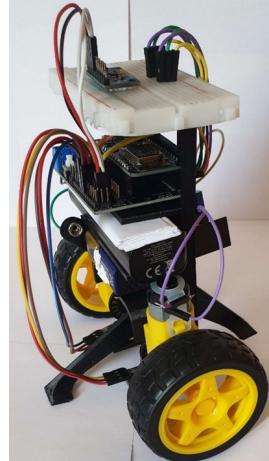
Programmation Arduino

Niveaux débutant et intermédiaire 

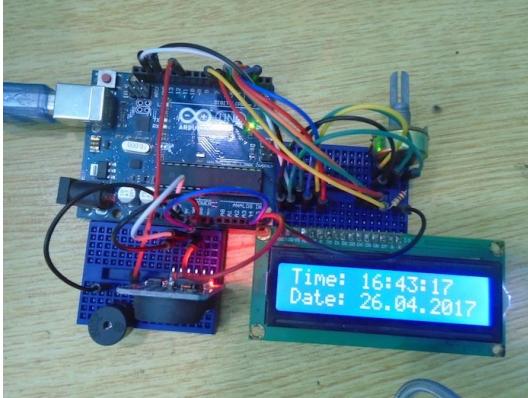
Exemples d'applications



Cardiofréquencemètre



Gyropode



Radio réveil



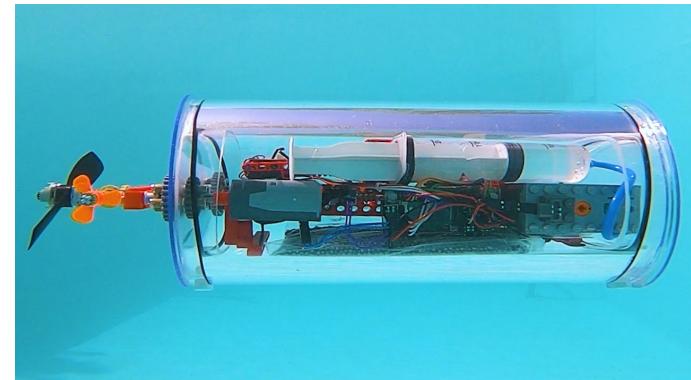
Drone



Piano

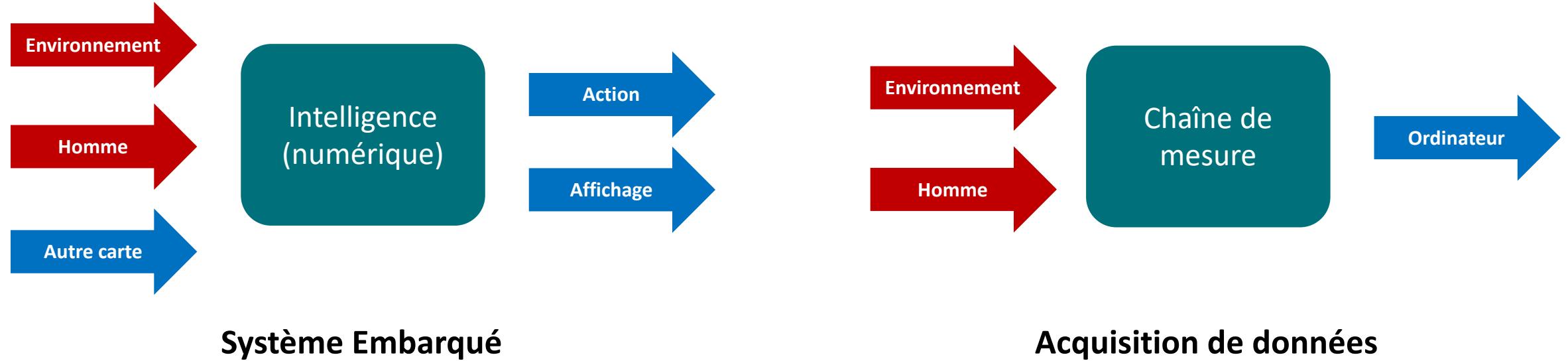


Machine à cocktail



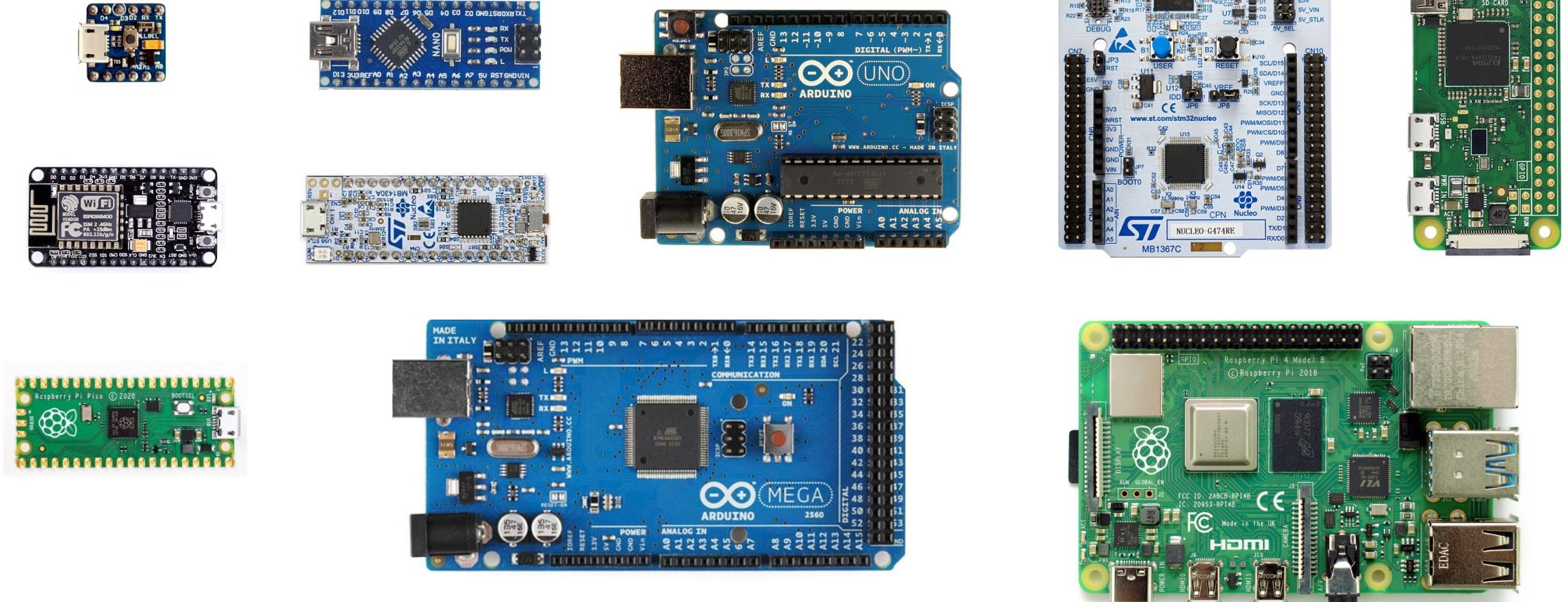
Sous marin

Cas d'usage



→ Signal numérique (0100011)
→ Signal analogique (Volts)

Les plateformes de prototypage

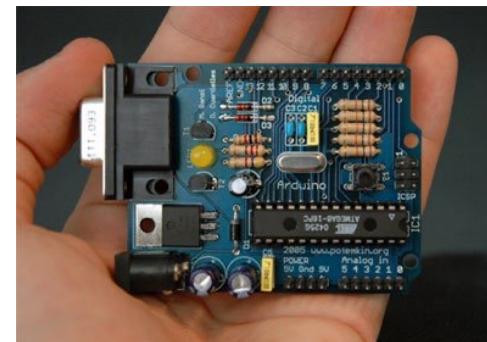
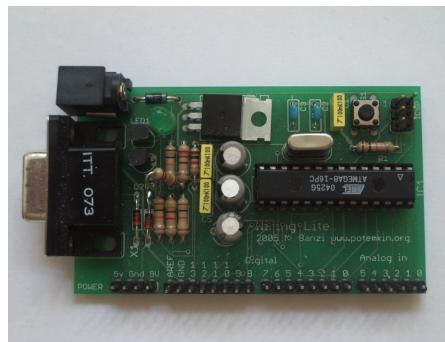
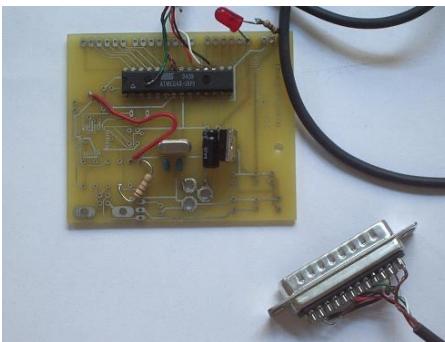


Tous les acteurs du marché s'y lancent...

Famille	Carte	Processeur	Nombre de bits	Fréquence horloge	Tension	Flash (Octets)	SRAM (Octets)	EEPROM (Octets)	GPIO	Résolution ADC (bits)	Pins PWM	Doc.	Bluetooth / Wifi	OS	Prix (TTC)
Arduino	Nano 33 BLE	Cortex M-4	32	64 MHz	3,3 V	1 M	256 k	NON (Flash)	14	12	14	****	OUI	NON	30 €
	RP2040	Cortex M0+	32	133 MHz	3,3 V	16 M	264 k	NON (Flash)	14	12	14		OUI	NON	30 €
	Nano	ATMega328	8	16 MHz	5 V	32 k	2 k	1k	22	10	6		NON	NON	20 €
	Due	SAM 3X8E	32	84 MHz	3,3 V	512 k	96 k	NON (Flash)	54	12	12		NON	NON	50 €
NodeMCU	ESP8266	ESP8266	32	80 MHz	3,3 V	4 M	64 k	NON (Flash)	11	10	9	***	OUI	NON	10 €
	ESP32	ESP32	32	240 MHz	3,3 V	4 M	520 k	NON (Flash)	34	12	18		OUI	NON	20 €
Nucleo	32	STM32	32	24 MHz	5 V	16 – 256 k	16 k	NON (Flash)	20	12	17	**	NON	NON	15 €
	64	STM32	32	24 MHz	5 V	64 k – 1 M	256 k	NON (Flash)	64	12	14		NON	NON	20 €
Raspberry Pi	Pico	Cortex M0+	32	133 MHz	3,3V	2 M	264 k	NON (Flash)	26	12	16	***	OUI	NON	10 €
	4 v. 4Go	Cortex A-72	32	1,5 GHz	5 V	Micro SD	4 G	NON (micro SD)	40	12	4		OUI	OUI	90 €

Arduino

- Arduino est la marque d'une **plateforme de prototypage** constituée d'une carte électronique basée autour d'un **microcontrôleur** (plusieurs cartes existent) et d'un environnement de développement (Arduino IDE).
- Il s'agit d'un écosystème **open source** (copie, modification et fabrication libre de droits), **multiplateforme**, très documenté (et doté d'une grande communauté d'utilisateurs) et peu cher permettant de répondre à un grand nombre de cas d'usages.
- Le projet Arduino a vu le jour en 2005 à l'Interaction Design Institute Ivrea et porte le nom du bar « di Re Arduin » (bar du roi Arduin), lieu de réunion des concepteurs de la carte dans le nord de l'Italie, à Ivrée.



Arduino

- La **programmation se fait en C et C++**, permettant ainsi à tout un chacun de se débrouiller, mais aussi d'avoir une grande maîtrise de la mémoire qui peut alors être vue du point de vue de l'électronicien (registres).
- On peut étendre les possibilités de la carte à l'aide de façon :
 - Matérielle : à l'aide de composants (capteurs, afficheurs, etc.), d'un shield (carte électronique fille) ;
 - Logicielle : à l'aide de bibliothèques que l'on importe dans le code du microcontrôleur.

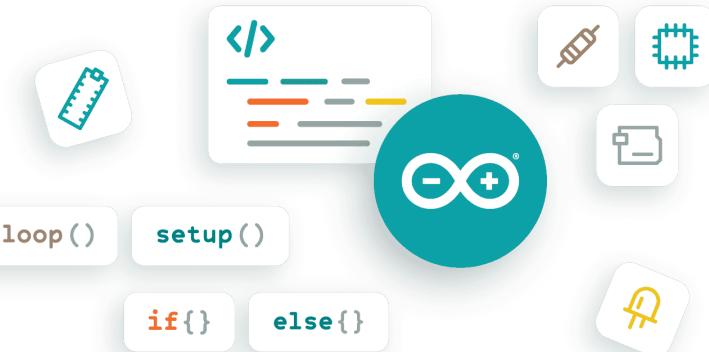
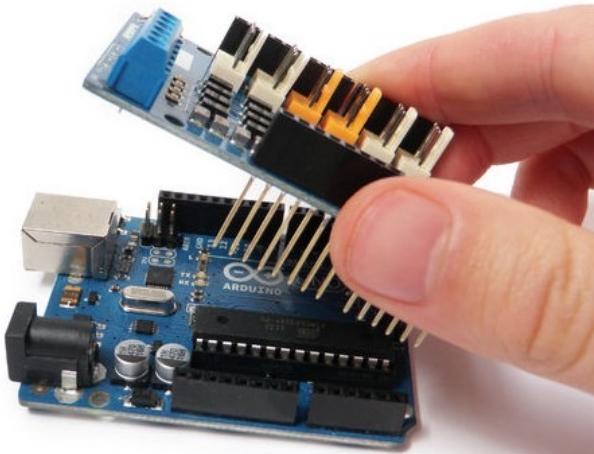


Table des matières

1. La carte Arduino Nano

- Approche de développement _____ p. 10
- La carte Arduino Nano _____ p. 11
- Programmation Arduino _____ p. 14
- Pins de l'Arduino et du microcontrôleur _____ p. 15
- Le microcontrôleur ATMega328P _____ p. 16
- Fonctionnement du microcontrôleur _____ p. 18
- Temps d'exécution du programme _____ p. 19
- 1^{er} code Arduino _____ p. 20
- Les entrées-sorties numériques _____ p. 21
- Gestion du temps _____ p. 25
- Précautions à prendre... _____ p. 26

2. Programmation en C

- Où est le main(void) ? _____ p. 28
- Les directives du processeur _____ p. 29
- Types de variables _____ p. 30
- Les opérateurs arithmétiques et relationnels _____ p. 34
- Les opérateurs binaires _____ p. 35
- Structure algorithmique _____ p. 36

3. Conversion analogique - numérique

- Le CAN de l'Arduino _____ p. 38
- Utiliser un potentiomètre _____ p. 39

4. Timers et PWM

- Les Timers de l'Arduino _____ p. 42
- Principe du Timer _____ p. 43
- Modes de fonctionnement _____ p. 44
- Génération de signaux PWM _____ p. 45
- La fonction Tone _____ p. 48

5. Communication série

- Le module de communication série de l'Arduino _____ p. 50
- Serial plotter _____ p. 51

6. Bibliothèques et fonctions mathématiques

- Utilisation d'un bibliothèque _____ p. 54
- Fonctions mathématiques _____ p. 55

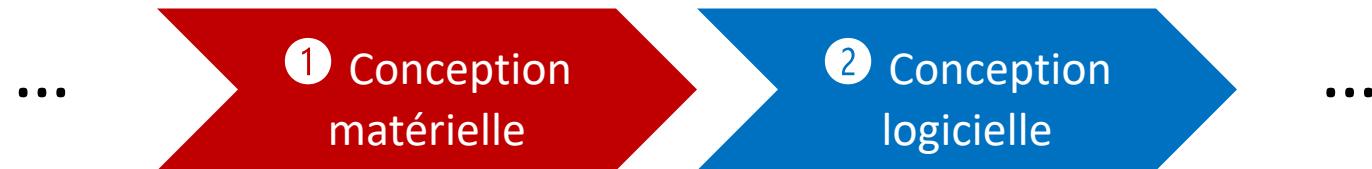
7. Les fonctions d'interruption

- Programmer des fonctions d'interruption _____ p. 57
- Interruptions externes _____ p. 58

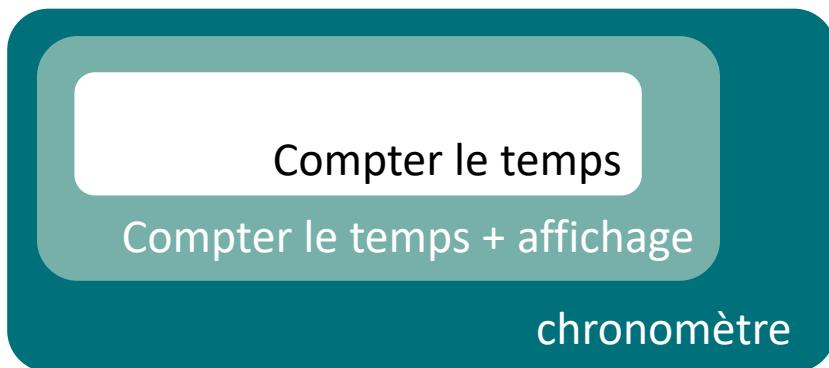
1 La carte Arduino Nano

Approche de développement

- **Cycles itératifs de développements et de tests** afin d'ajouter des fonctionnalités et obtenir un système complexe en suivant le principe suivant (effet boule de neige) :

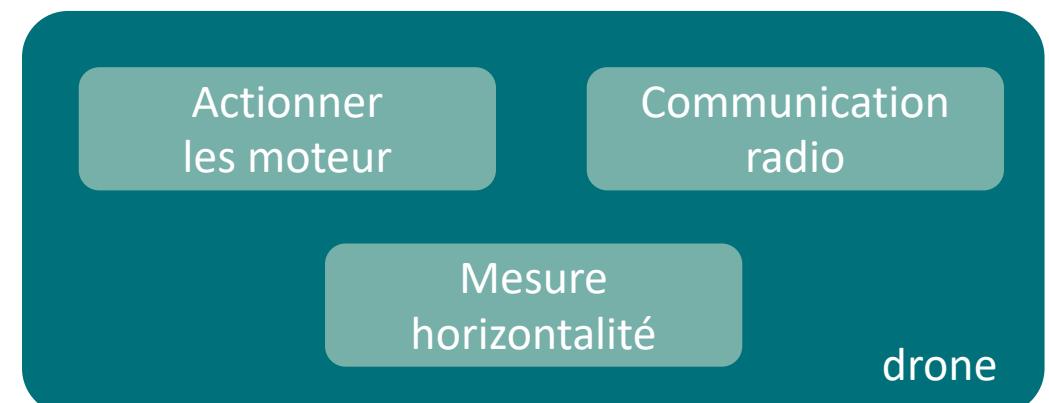


- On commence par parfaitement valider des fonctionnalités simples avant d'en ajouter d'autres.

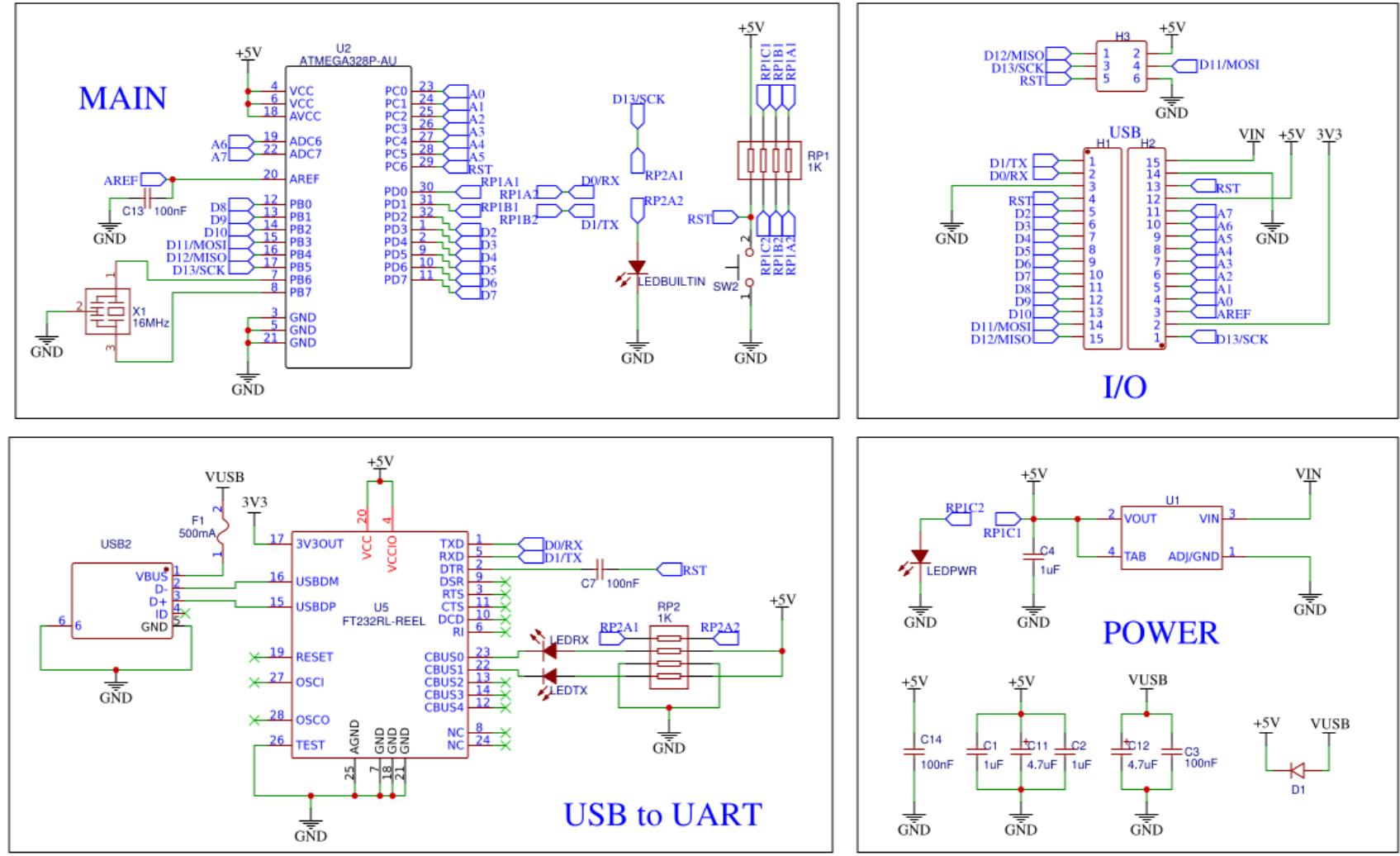


- On peut également en développer plusieurs en parallèle puis les intégrer à la fin.

Mais prévoir un temps conséquent pour l'intégration (logicielle + matérielle !)



La carte Arduino Nano



La carte Arduino Nano

30 connecteurs mâles pour 22 GPIO
(*General Purpose Input Output*)

Bouton RESET

LEDs :

- RX (*Receive* – com. série UART)
- TX (*Transmit* – com. série UART)
- POWER
- LED (PIN 13)



Connecteur mini USB

Microcontrôleur 8-bit
ATMEGA328P

Quartz 16 MHz

Connecteur mâle ICSP
(*In Circuit Serial Programmer*)

RESET	SCK	MISO
GND	MOSI	VCC

La carte Arduino Nano

Alimentation 3,3 V

Broches analogiques

Résolution 10 bits
donc codage sur $2^{10} = 1024$ valeurs

A4 = SDA et A5 = SCL (communication série I2C)

Alimentation 5 V

Reset

Masse

Alimentation de la carte (6 à 20 V)

D13
3V3
REF
A0
A1
A2
A3
A4 / SDA
A5 / SCL
A6
A7
5V
RST
GND
VIN



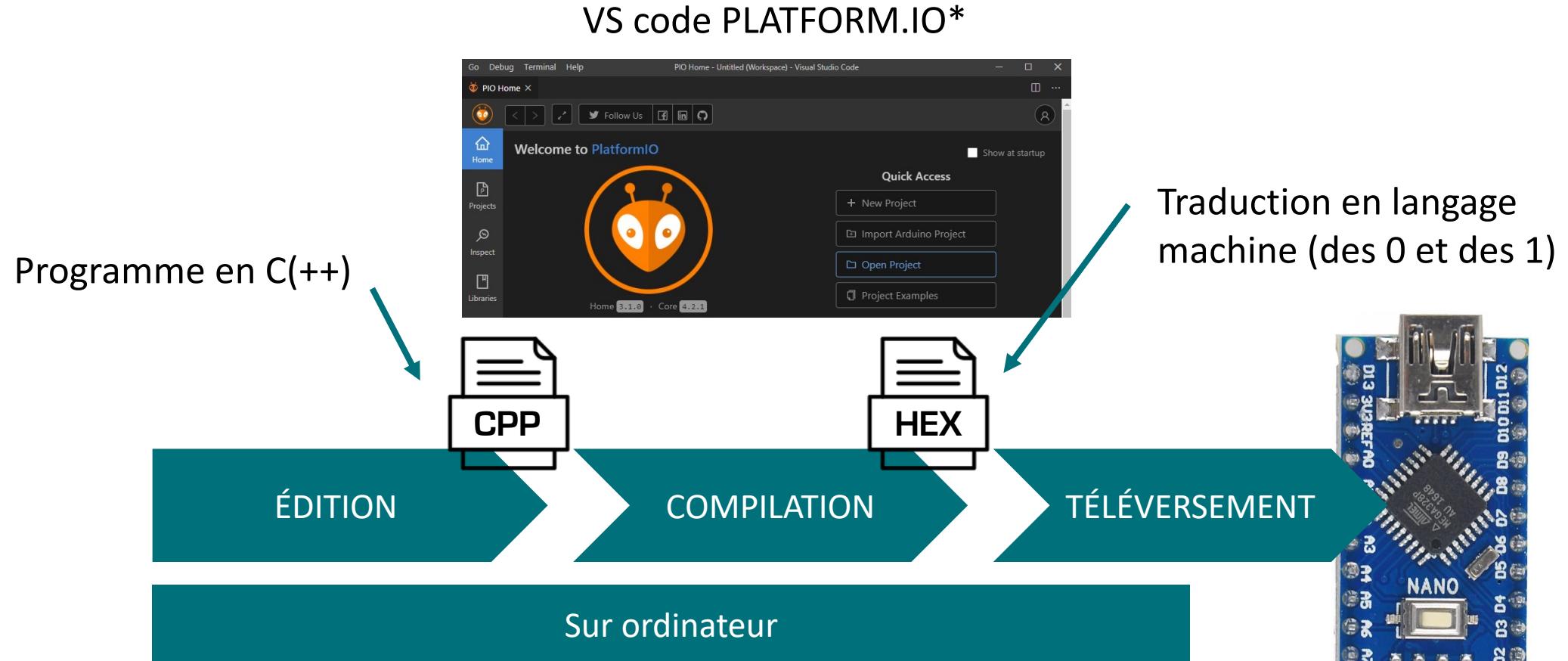
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
GND
RST
D1 / RX
D0 / TX

Broches numériques

Masse
Reset

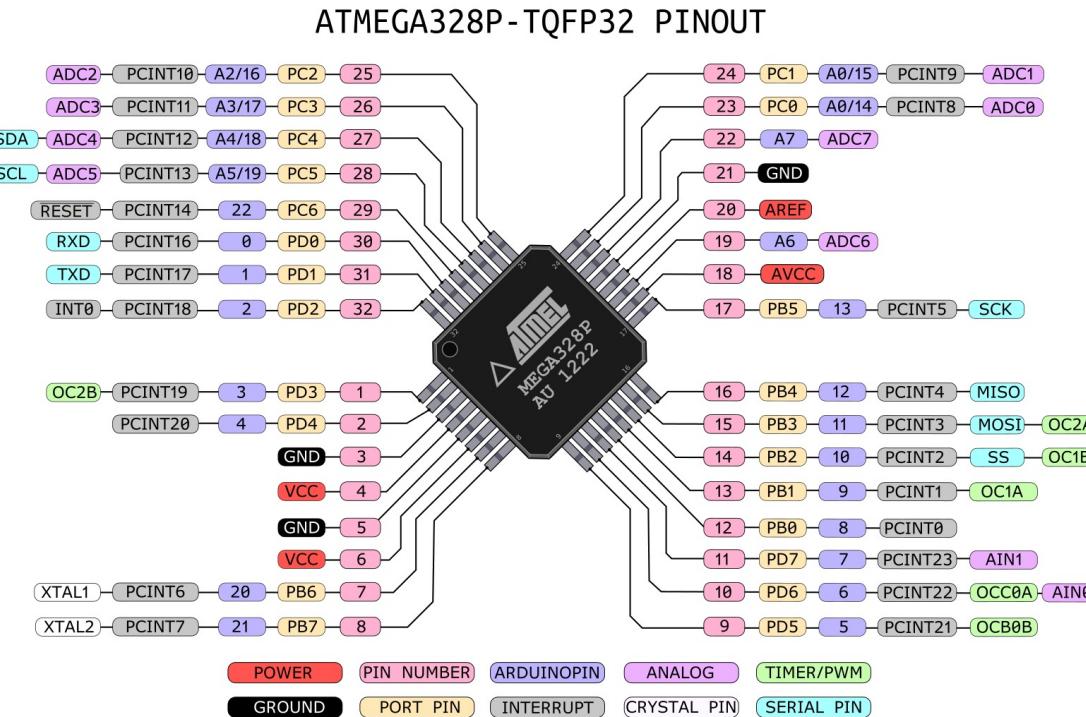
D0 = RX et D1 = TX (communication série UART)

Programmation Arduino



Pins de l'Arduino et du microcontrôleur

- Il faut bien garder à l'esprit que le microcontrôleur ATmega328P a sa propre numérotation et que son branchement aux GPIO de la carte est imposée par le design de la carte Arduino.



EXEMPLE

La Pin numérique 2 (D2)
correspond à la pin 4 du microcontrôleur.

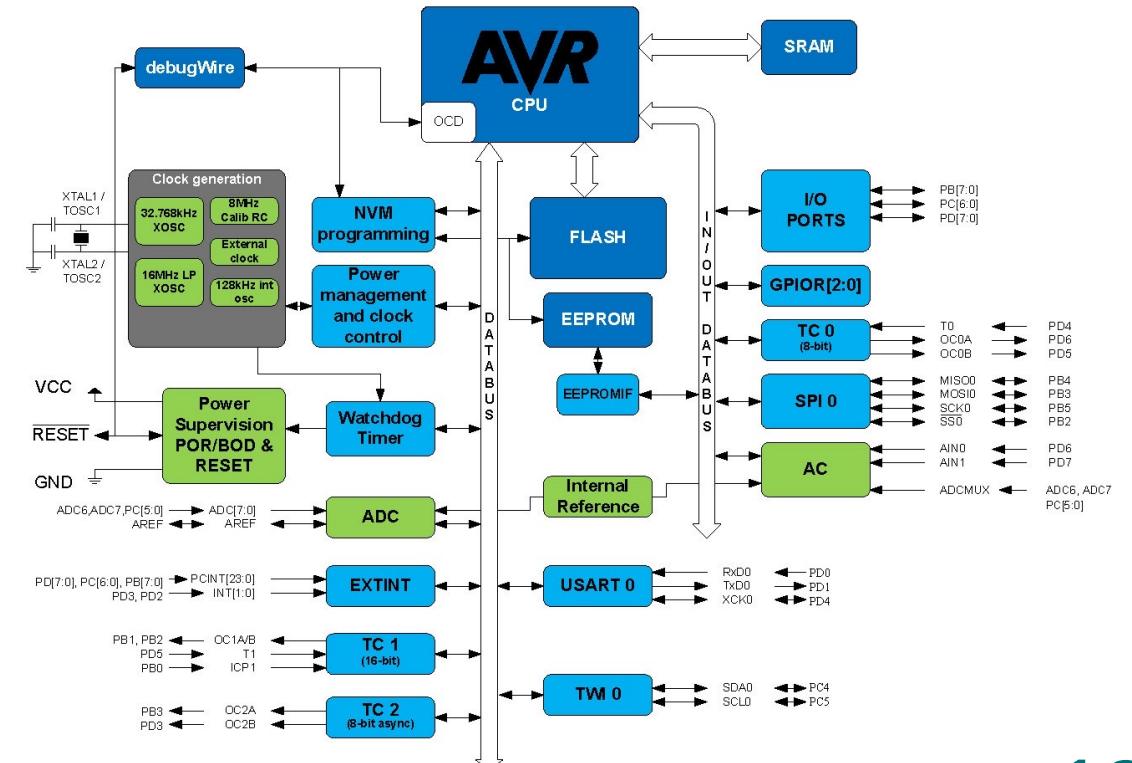
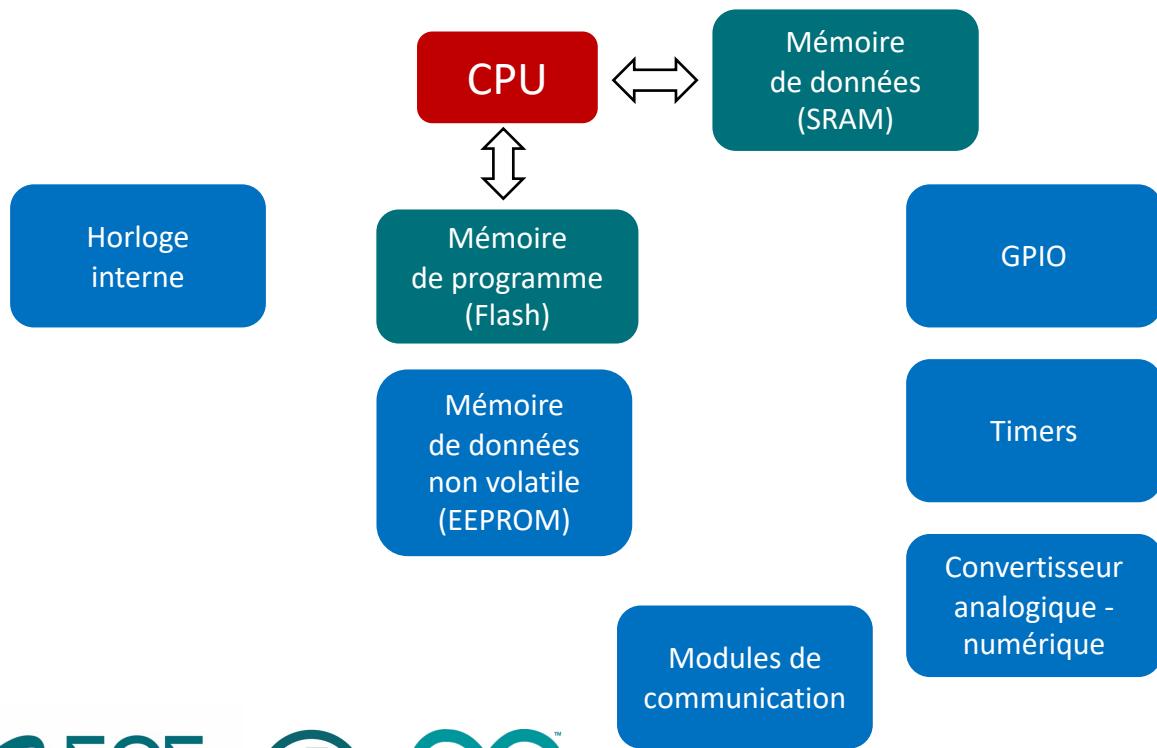
Lors de la programmation de la carte Arduino,
on ne s'en préoccupe guère.

→ Comment le code est-il exécuté ?

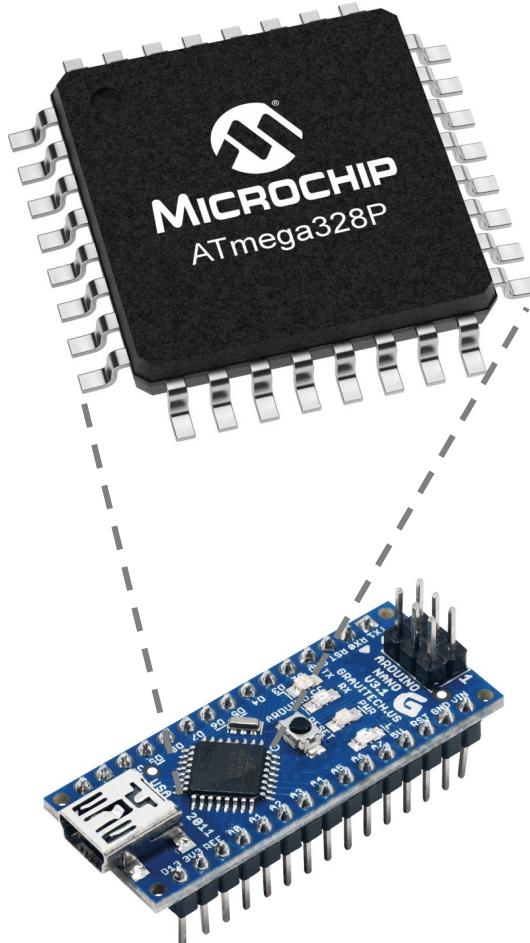
Le microcontrôleur ATMega328P

■ Un microcontrôleur comprend :

- Une unité de calcul (*Central Processing Unit*)
- De la mémoire ;
- Des périphériques de contrôle (GPIO, Timers, modules de communication, ADC, etc.)

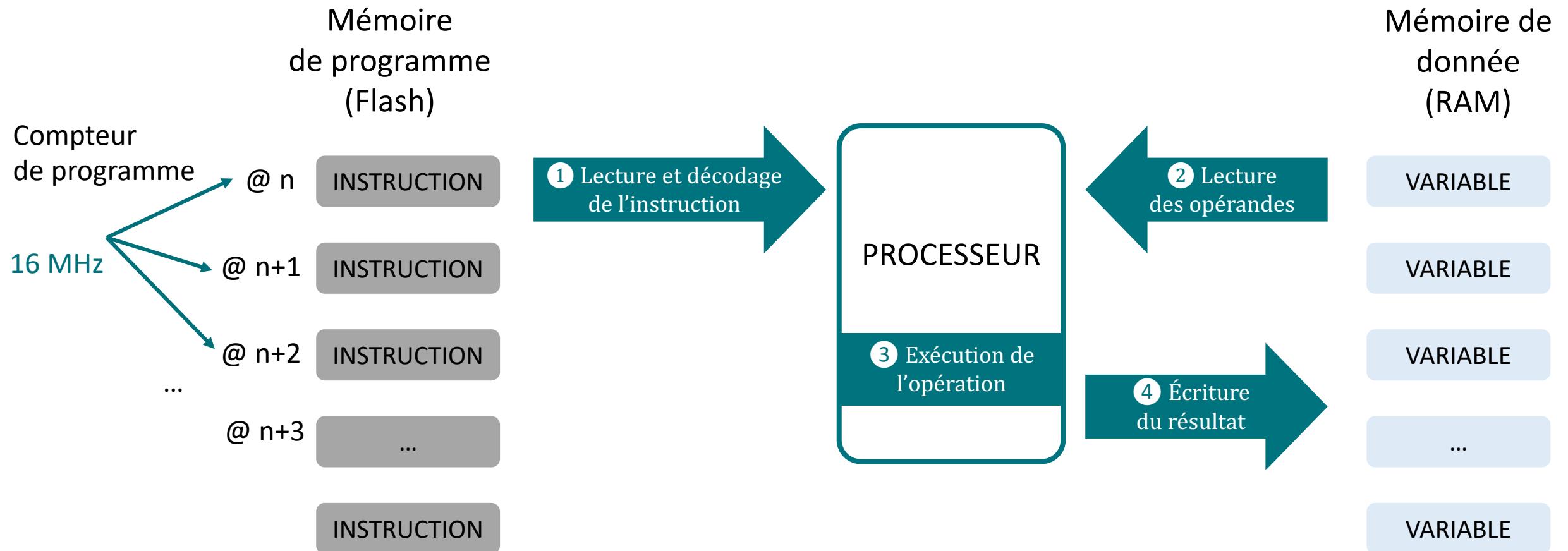


Le microcontrôleur ATMega328P



- Microcontrôleur 8-bit
- Mémoire
 - ▶ Programme : NAND Flash 32 ko
 - ▶ Données volatiles : SRAM 2 ko
 - ▶ Données non-volatiles : EEPROM 1 ko
- Alimentation 1,8 – 5,5 V
- Horloge
 - ▶ Oscillateur interne 32,8 kHz ou 7,37 MHz
 - ▶ Oscillateur externe jusqu'à 40 MHz, Arduino Nano : **16 MHz**
 - ▶ Fréquence d'un cycle : $F_{CY} = \frac{F_{osc}}{2}$
- 23 entrées-sorties dont
 - ▶ **8 entrées analogiques 10 bits (par défaut 9,6 kHz)**
 - ▶ 6 pins PWM
 - ▶ 1 liaison série asynchrone UART
 - ▶ 2 liaisons série synchrones SPI
 - ▶ 1 bus I2C
- 3 timers 8-bit dont 1 timer 16-bit (TIMER1)

Fonctionnement du microcontrôleur



Temps d'exécution du programme

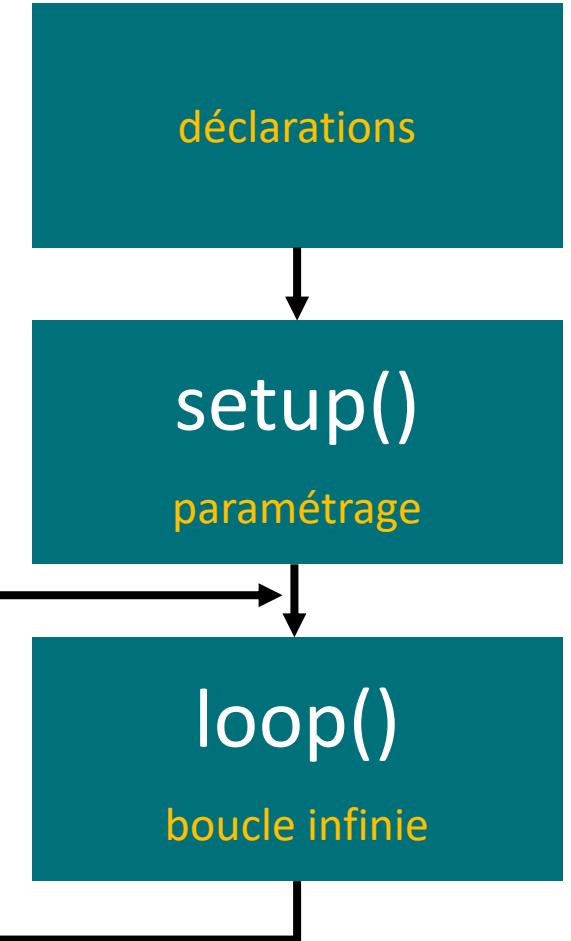
- Le temps d'exécution d'un programme sur un microcontrôleur donné (ATMega328P) peut varier en fonction de la compilation des instructions Arduino, des types de variable, de l'algorithme, des arguments passés en paramètre dans les fonctions, etc.
 - Il peut être opportun d'optimiser le code en choisissant des implémentations plus optimales, par exemple `>>1` au lieu de `/2`.
- Exemple : $(^/_2)_{10} = (0110 \gg 1)_2 = (0011)_2 = (3)_{10}$

Commande	Temps d'exécution
<code>maVariable = 255 ;</code>	$1,7 \mu s$
<code>maVariable++ ;</code>	$2 \mu s$
<code>maVariable = maVariable / 2 ;</code>	$1,75 \mu s$
<code>REGISTRE = valeurOctet ;</code>	$0,38 \mu s$
<code>digitalWrite(pin,etat);</code>	$5,5 \mu s$
<code>digitalRead(pin);</code>	$5,3 \mu s$
<code>digitalWrite(pin1,digitalRead(pin2));</code>	$10 \mu s$
<code>valInt = analogRead(pinA);</code>	$110 \mu s$

Commande	Temps d'exécution
<code>Serial.println(valeurINT);</code>	$380 \mu s$
<code>If(variable == valeur){}</code>	$1,5 \mu s$
<code>delayMicroseconds(1);</code>	$1,96 \mu s$
<code>delayMicroseconds(2);</code>	$2,7 \mu s$
<code>delayMicroseconds(10);</code>	$10,8 \mu s$
<code>delayMicroseconds(50);</code>	$50 \mu s$

1^{er} code Arduino

```
1 ///////////////
2 // DECLARATIONS //
3 ///////////////
4 const int Led=3;
5
6 ///////////////
7 // FONCTION SETUP //
8 ///////////////
9 void setup()
10{
11    pinMode(Led, OUTPUT); // la broche 3 est une sortie
12}
13
14 ///////////////
15 // FONCTION LOOP //
16 ///////////////
17 void loop()
18{
19    digitalWrite(Led,HIGH); // met la broche au niveau haut (5V): on allume la LED
20    delay(500); // pause de 500 millisecondes
21    digitalWrite(Led,LOW); // met la broche au niveau bas (0V): on éteind la LED
22    delay(500); // pause de 500ms
23}
```



Les entrées – sorties numériques

- Toutes les 14 GPIO sont d'entrée numériques (*digital* en anglais) – le microcontrôleur fonctionne naturellement en numérique.

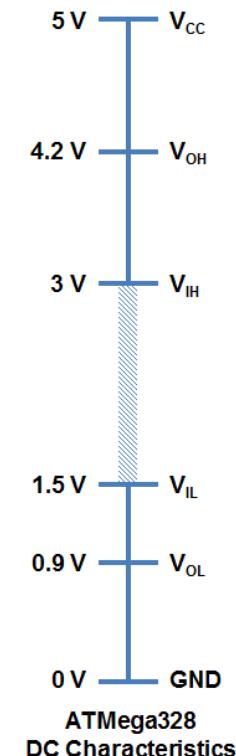
digitalRead

E
N
E
N
T
R
É
E

État logique haut
= '1' logique
= HIGH

État indéterminé

État logique bas
= '0' logique
= LOW



État logique haut
= '1' logique
= HIGH

digitalWrite

E
N
S
O
R
T
I
E

État logique bas
= '0' logique
= LOW

Permet
d'allumer
une LED

Les entrées – sorties numériques

■ Pour les utiliser :

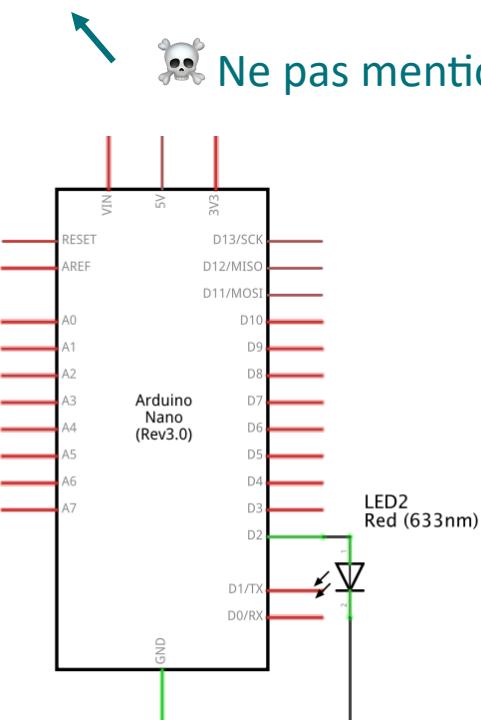
- ① Bien identifier son emplacement matériel, vérifier le branchement.
- ② Configurer la pin à l'aide de la fonction `pinMode(numeroDePin, INPUT, INPUT_PULLUP ou OUTPUT)` ; dans la procédure `setup()`

③ L'utiliser :

- Pour une lecture : `variable = digitalRead(numeroDePin);`
- Pour une écriture : `digitalWrite(numeroDePin, LOW ou HIGH);`

EXEMPLE 1

LED branchée sur la pin 2
qui clignote toutes les secondes.



```
void setup()
{
    pinMode(2, OUTPUT);
}

void loop()
{
    digitalWrite(2, HIGH);
    delay(1000);
    digitalWrite(2, LOW);
    delay(1000);
}
```

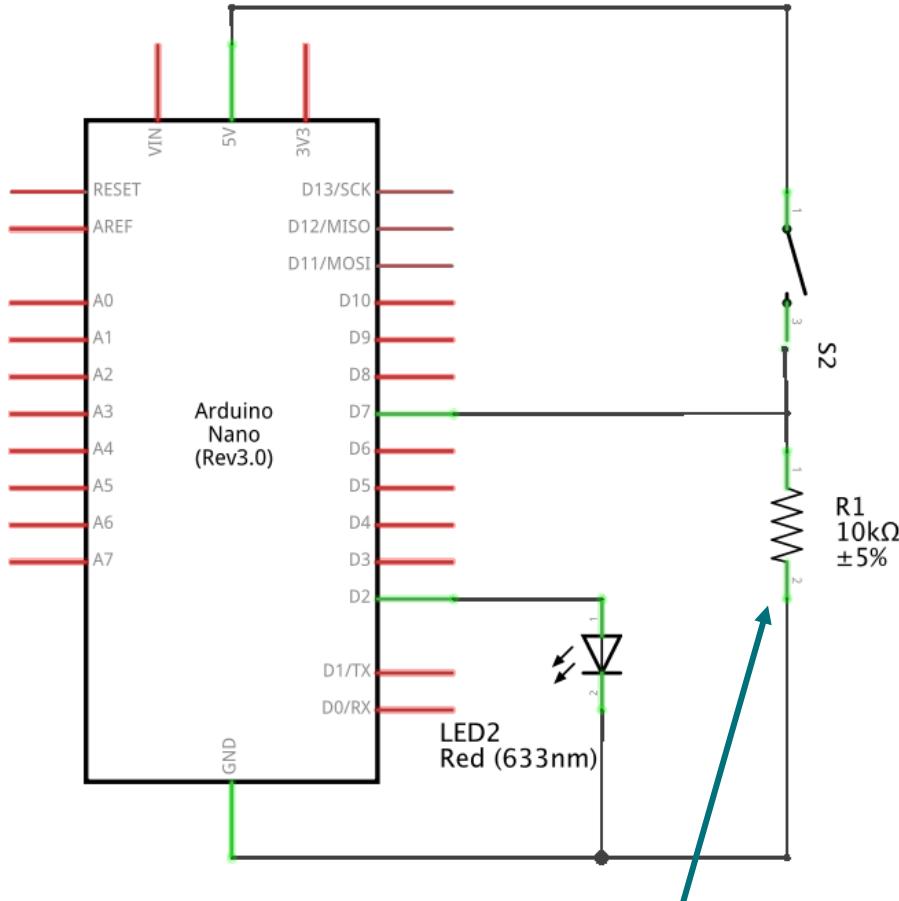
Les entrées – sorties numériques

EXEMPLE 2

LED branchée sur la pin 2

Bouton en pull-down sur la pin 7

LED qui s'allume pendant 5 sec
si l'on appuie sur le bouton



évite de tirer un trop gros courant
si l'interrupteur est fermé.

```
void setup()
{
    pinMode(2, OUTPUT);
    pinMode(7, INPUT);
}

void loop()
{
    if(digitalRead(7))
    {
        digitalWrite(2, HIGH);
        delay(5000);
        digitalWrite(2, LOW);
    }
    else
        digitalWrite(2, LOW);
}
```

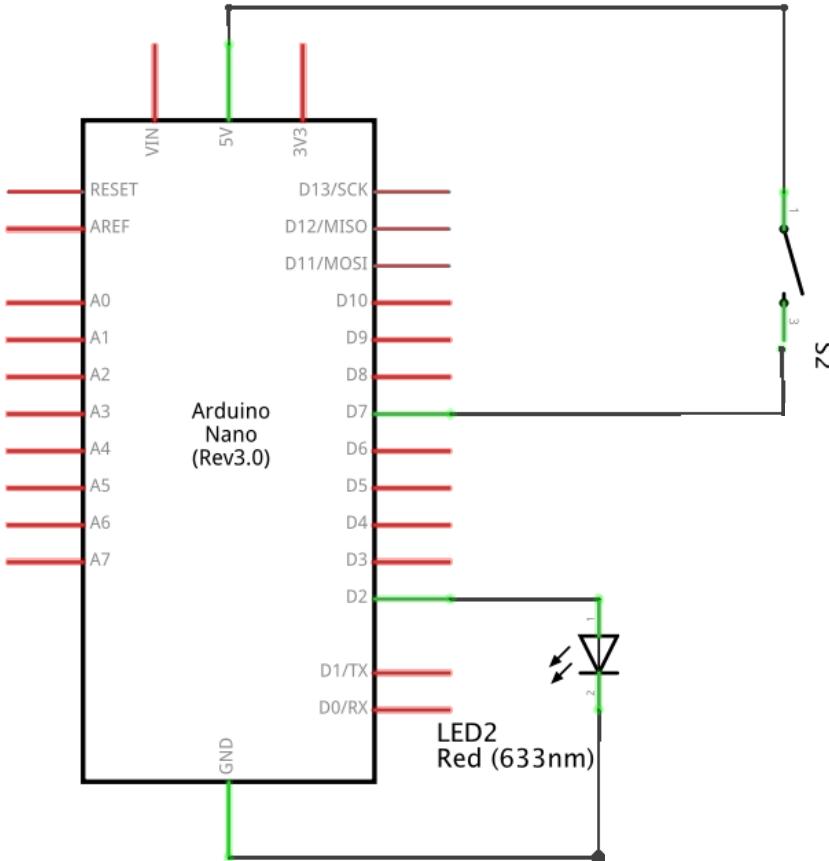
Les entrées – sorties numériques

EXEMPLE 3

LED branchée sur la pin 2

Bouton sans résistance sur la pin 7

LED qui s'allume pendant 5 sec
si l'on appuie sur le bouton



Utilise la résistance de pull-up
interne de 20 kΩ

```
void setup()
{
    pinMode(2, OUTPUT);
    pinMode(7, INPUT_PULLUP);
}

void loop()
{
    if(digitalRead(7))
    {
        digitalWrite(2, HIGH);
        delay(5000);
        digitalWrite(2, LOW);
    }
    else
        digitalWrite(2, LOW);
}
```

Gestion du temps : pause du processeur et mesure du temps

- Il est parfois nécessaire de ralentir l'exécution du programme. Cela peut se faire à l'aide des fonctions **delay(ms)** et **delayMicroseconds(us)**
- Ces fonctions « bloquent » le microcontrôleur : il n'est pas possible d'exécuter d'autres actions en parallèle.
- Il est possible de mesurer le temps entre deux instants en utilisant les fonctions **unsigned long millis()** et **unsigned long micros()** qui retournent le temps écoulé depuis le début de l'exécution du code respectivement en millisecondes et en microsecondes.
 - **Remarque** : un unsigned long est codé sur 32 bits donc il compte de 0 à $2^{32} - 1 = 4\,294\,967\,296$
 - Soit un maximum de **49,7 jours** environ avec millis()
 - Ou environ **71,6 minutes** avec micros()

EXEMPLE

Mesurer le temps écoulé entre la fin du setup et le début du loop

```
void setup()
{
    unsigned long CurrentTime, elapsedTime = 0;
    unsigned long StartTime = millis();
}

void loop()
{
    CurrentTime = millis();
    ElapsedTime = CurrentTime - StartTime;
}
```

Précautions à prendre...

- Il est facile d'endommager irréversiblement une carte Arduino en tirant trop de courant ($> 40 \text{ mA / pin}$)

28.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except <u>RESET</u> with respect to Ground	-0.5V to $V_{CC}+0.5V$
Voltage on <u>RESET</u> with respect to Ground.....	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins.....	200.0 mA



💀 Connecter directement la pin 5V ou 3.3V à la masse

$$I = \frac{5 \text{ V}}{25 \Omega} = 200 \text{ mA} \quad I = \frac{3 \text{ V}}{25 \Omega} = 120 \text{ mA}$$

💀 Connecter directement une pin à l'état haut à la masse

💀 Connecter directement une pin à l'état haut à une pin à l'état bas

💀 Appliquer une trop forte tension ($> 5,5 \text{ V}$) à une pin
Ex : pile 9V

...

2 Programmation en C

Où est le main(void) ?

```
#include <Arduino.h>
int main(void)
{
    init();
    initVariant();

#if defined(USBCON)
    USBDevice.attach();
#endif

    setup();

    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }
    return 0;
}
```

- Chaque **instruction** est terminée par le caractère ;
- Une **fonction** a un nom, des paramètres typés et un type de retour
- La fonction main() est obligatoire
- Le comportement d'une fonction est défini dans un **bloc** identifié par des accolades
- Les instructions peuvent être regroupées en bloc si elles doivent être exécutées ensemble
- Les **variables** sont **déclarées** et **initialisées** avant d'être utilisées

Les directives du processeur

- Les **directives du processeur** sont précédées par le caractère **#** et sont placées au début du code, dans la section de déclaration des variables.
- Elles sont traitées avant la compilation selon une logique de « rechercher – remplacer ».
- Deux cas d'usage :
 - Définir des alias pour du texte

EXEMPLES

```
#define LED 1  
  
#define button 2  
  
#define sensor A0
```

- Créer des macros avec des arguments

EXEMPLES

```
#define button_ON (digitalRead(button)==HIGH)  
  
#define max(a,b) ((a)>(b)?(a):(b))
```

Types de variables : le bool et le boolean

1 octet

■ Le **bool**

- Un bool est codé sur 1 octet, donc 8 bits.
- Il ne peut contenir que deux valeurs : **true** = 1 ou **false** = 0.

■ Le **boolean**

- Type non standard équivalent au type bool

■ Remarque : il n'existe pas de type de donnée codée sur 1 bit.

Types de variables : les caractères

1 octet

■ Le **char** = **int8_t**

- Un char est un entier signé qui est codé sur 1 octet, donc **8 bits**.
- Il prend ses valeurs dans l'intervalle $[-2^7 = -128 ; 2^7 - 1 = 127]$
- Utilisé pour coder les caractères (standard ASCII notamment)
- Intervalle non symétrique car il prend en compte le 0
- Numérotation circulaire
 - Exemple 1 : $127 + 1$ donne -128
 - Exemple 2 : $-128 - 1$ donne 127

■ L'**unsigned char** = **uint8_t**

- Un char est un entier signé qui est codé sur 1 octet, donc **8 bits**.
- Il prend ses valeurs dans l'intervalle $[0 ; 2^8 - 1 = 255]$
- Numérotation circulaire
 - Exemple 1 : $255 + 1$ donne 0
 - Exemple 2 : $0 - 1$ donne 255

■ Le **byte**

- Similaire au type **unsigned char**

Types de variables : les entiers

2 octets

■ Le `int = int16_t`

- Un entier signé int est codé sur 2 octets, donc **16 bits**.
- Il prend ses valeurs dans l'intervalle $[-2^{15} = -32\ 768 ; 2^{15} - 1 = 32\ 768]$
- Intervalle non symétrique car il prend en compte le 0
- Numérotation circulaire
 - Exemple 1 : $32\ 767 + 1$ donne $-32\ 768$
 - Exemple 2 : $-32\ 768 - 1$ donne $32\ 767$

■ Le `unsigned int = uint16_t`

- Un entier non signé unsigned int est codé sur 2 octets, donc **16 bits**.
- Il prend ses valeurs dans l'intervalle $[0 ; 2^{16} - 1 = 65\ 535]$
- Numérotation circulaire
 - Exemple 1 : $65\ 535 + 1$ donne 0
 - Exemple 2 : $0 - 1 = 65\ 535$

■ Le `short` est également codé sur 2 octets, donc **16 bits**.

Types de variables : les longs et les floats

4 octets

■ Le **long** = **int32_t**

- Un long est codé sur 4 octets, donc **32 bits**.
- Il prend ses valeurs dans l'intervalle $[-2^{31} = -2\ 147\ 483\ 648 ; 2^{15} - 1 = 2\ 147\ 483\ 647]$
- Intervalle non symétrique car il prend en compte le 0
- Numérotation circulaire
 - Exemple 1 : $2\ 147\ 483\ 647 + 1$ donne $-2\ 147\ 483\ 648$
 - Exemple 2 : $-32\ 768 - 1$ donne $2\ 147\ 483\ 647$

■ Le **unsigned long** = **uint32_t**

- Un unsigned long est codé sur 4 octets, donc **32 bits**.
- Il prend ses valeurs dans l'intervalle $[0 ; 2^{32} - 1 = 4\ 294\ 967\ 296]$
- Numérotation circulaire
 - Exemple 1 : $4\ 294\ 967\ 296 + 1$ donne 0
 - Exemple 2 : $0 - 1 = 4\ 294\ 967\ 296$

■ Le **float**

- Un float est un nombre réel (à virugle et signé) codé sur 4 octets, donc **32 bits**.
- Sa précision est au maximum de 7 décimales



Les opérateurs arithmétiques et relationnels

Opérateurs arithmétiques

Type	Symbole	Exemple
Addition	+	<code>a = a + 2;</code>
Plus unaire	++	<code>a++;</code>
Soustraction	-	<code>b = b - 1;</code>
Moins unaire	--	<code>a--;</code>
Multiplication	*	<code>c = 2 * a;</code>
Division	/	<code>a = c / a;</code>
Reste de la division entière	%	<code>e = r % 10;</code>

Opérateurs relationnels

Type	Symbol	Exemple
Égalité	==	<code>a == 5</code>
Different	!=	<code>b != 2</code>
Supérieur	>	<code>a > b</code>
Supérieur ou égal	>=	<code>a >= 5</code>
Inférieur	<	<code>a < b</code>
Inférieur ou égal	<=	<code>b <= 3</code>

Les opérateurs binaires et binaires bit à bit

Opérateurs binaires

Type	Symbole
ET logique	<code>&&</code>
OU logique	<code> </code>
NON logique	<code>!</code>

Opérateurs binaires bit à bit

Type	Symbole	Exemple
ET binaire	<code>&</code>	<code>c = a & b;</code>
OU binaire	<code> </code>	<code>d = a b;</code>
OU exclusif	<code>^</code>	<code>e = a ^ b;</code>
Complément à 1	<code>~</code>	<code>a = ~a;</code>
Décalage à droite	<code>>></code>	<code>b = b >> 3;</code>
Décalage à gauche	<code><<</code>	<code>c = c << 1;</code>

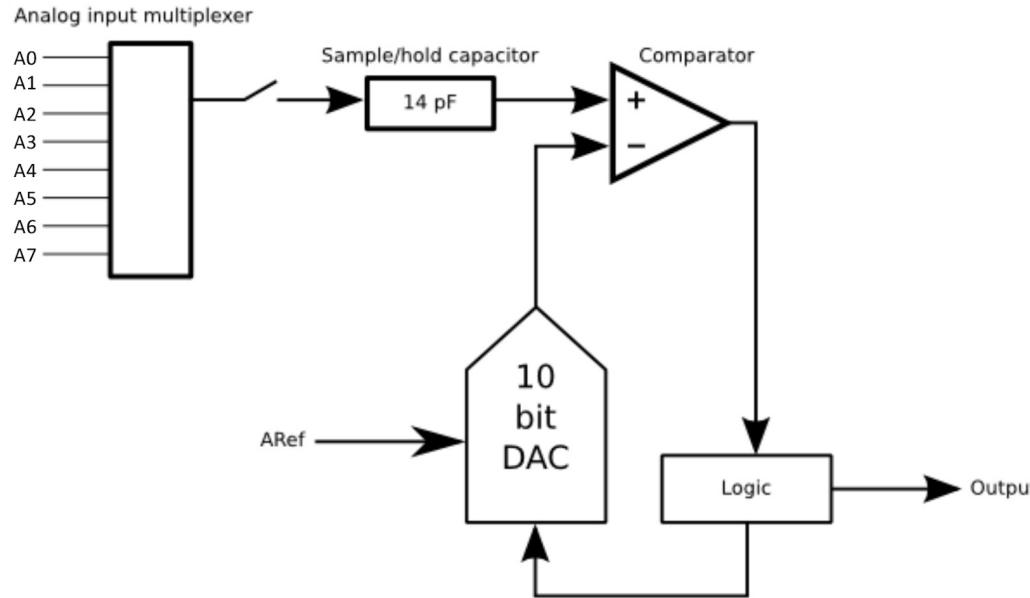
Structures algorithmiques

Structure while ...	Structure do ... while ...	Structure for	Structure if ... else	Structure switch ... case
Tant que <condition> faire ...	Faire ... tant que <condition>	Pour <variable> allant de <val initiale> à <val finale>, faire...	Si <condition> faire ... sinon faire ...	En fonction de la valeur d'une certaine variable, faire ...
<pre>while(condition) { //Action }</pre>	<pre>do { //Action } while(condition); ;</pre>	<pre>for(i=0 ; i<=5 ; i++) { //Action }</pre>	<pre>if(condition) { //Action1 } else { //Action2 }</pre>	<pre>Switch(variable) { case val1 : //actions break; case val2 : //actions break; }</pre>

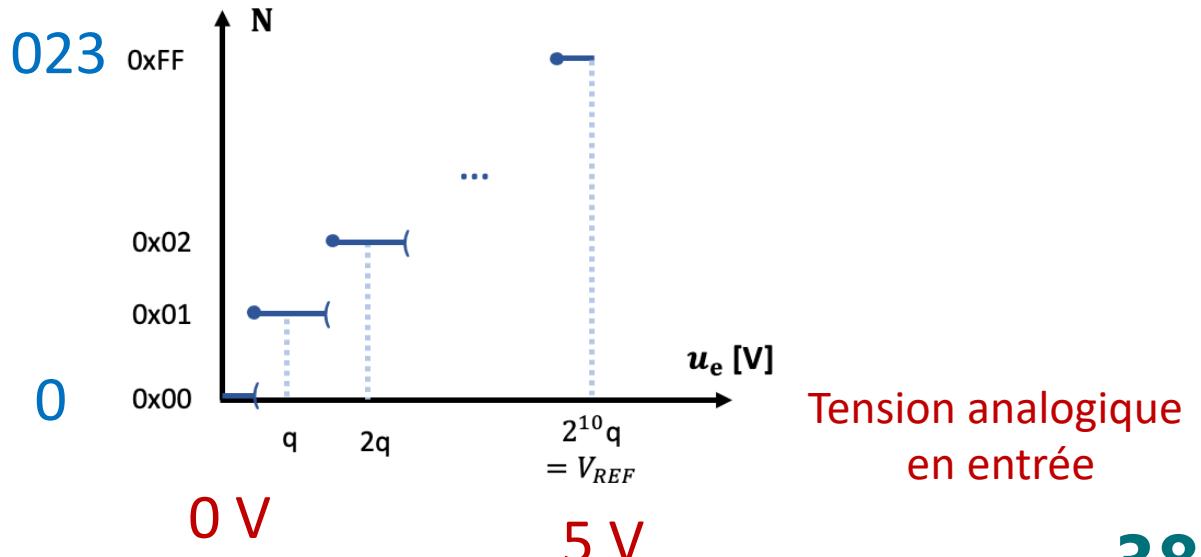
3 Conversion analogique numérique

Le CAN de l'Arduino

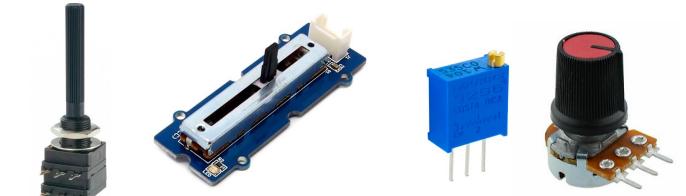
- L'Arduino Nano possède 8 entrées analogiques numérotées A0 à A7 qui sont connectées à un convertisseur analogique – numérique interne **de résolution 10 bits**.
- Le résultat de la conversion est donc codé de 0 à $2^{10} - 1 = 1023$, avec une précision (« **quantum** » q) de $\frac{5\text{ V}}{1\,023} = 4,9\text{ mV}$ par intervalle.



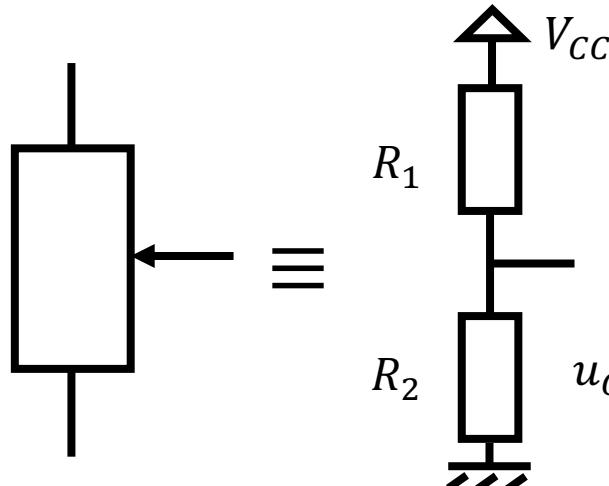
Résultat de la conversion
De $(00\ 0000\ 0000)_2 = (0)_{10}$
à $(11\ 1111\ 1111)_2 = (1\ 023)_{10}$



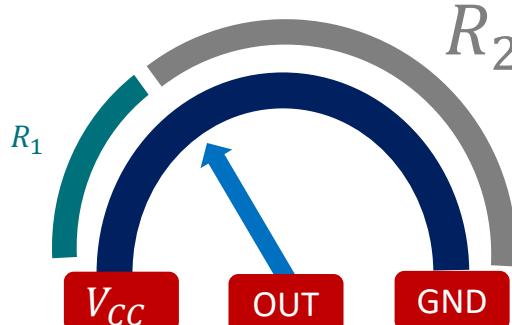
Utiliser un potentiomètre



EXEMPLES



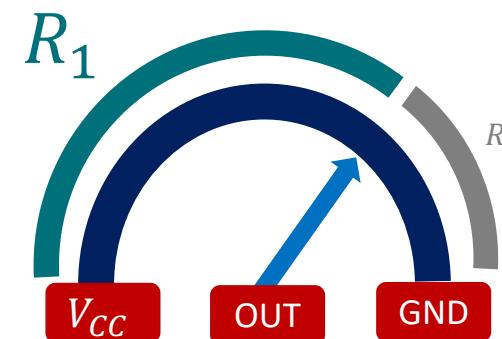
$$u_{OUT} = \frac{R_2}{R_1 + R_2} V_{CC}$$



$$\frac{R_2}{R_1 + R_2} \approx \frac{3}{4}$$

Donc tension entre $u_{OUT} \approx \frac{3}{4} V_{CC}$
= 3,75 V si $V_{CC} = 5V$.

analogRead retournerait $\frac{3}{4} \cdot 1023 = 767$



$$\frac{R_2}{R_1 + R_2} \approx \frac{1}{4}$$

Donc tension entre $u_{OUT} \approx \frac{1}{4} V_{CC}$
= 1,25 V si $V_{CC} = 5V$.

analogRead retournerait $\frac{1}{4} \cdot 1023 = 256$

Utiliser un potentiomètre

- Afin de convertir un signal analogique, il convient :

- 1 De déclarer un entier pour récupérer le résultat de la conversion.

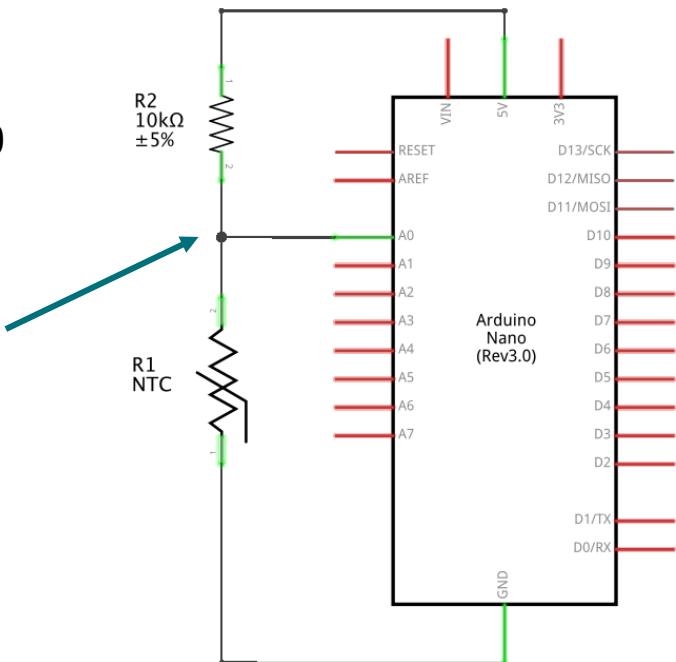
- 2 Configurer la pin en entrée à l'aide de la fonction

- 3 Utiliser la fonction `analogRead()`; pour récupérer le résultat de la conversion (entier entre 0 et 1023) et le stocker dans la variable dédiée.

EXEMPLE

Lire la tension entre la pin A0 et la masse.

$$V_{A0} = 5 \frac{10k}{R1 + 10k}$$



Pas nécessaire de déclarer la pin en entrée car contrairement aux GPIO, une entrée analogique ne peut pas fonctionner comme une sortie donc pas de configuration à choisir.

```
void setup()
{
    int sensorValue = 0;
    float voltage = 0;
    pinMode(A0, INPUT);
}

void loop()
{
    sensorValue = analogRead(A0);
    voltage = sensorValue * 5.0 / 1023.0;
}
```

Pour que le résultat du calcul reste en float (sinon résultats possibles : 0, 1, 2, 3, 4 ou 5)

4 Timers et PWM

Les Timers de l'Arduino

- Les **timers** sont des registres qui s'incrémentent à chaque coup de l'horloge que l'on lui a branché en entrée.
- Sur 8 bits : de 0 à $2^8 - 1 = 256$
- Sur 16 bits : de 0 à $2^{16} - 1 = 65\,536$
- La valeur de chaque timer est stockée dans un registre qui lui est propre:
 - ▶ **La valeur du timer0** est stockée dans le registre **TCNT0 (Timer/Counter register 0)**
 - ▶ **La valeur du timer1** est stockée dans le registre **TCNT1** (et plus précisément, en arrière plan, dans TCNT1H et TCNT1L ; car comme la valeur TCNT1 fait 16 bits, il faut la stocker dans 2 registres de 8 bits)
 - ▶ **La valeur du timer2** est stockée dans le registre **TCNT2**
- L'intérêt est qu'ils sont indépendants du CPU et donc de l'exécution du code : ils s'incrémentent à chaque coup d'horloge à la fréquence de l'horloge qu'on lui donne.

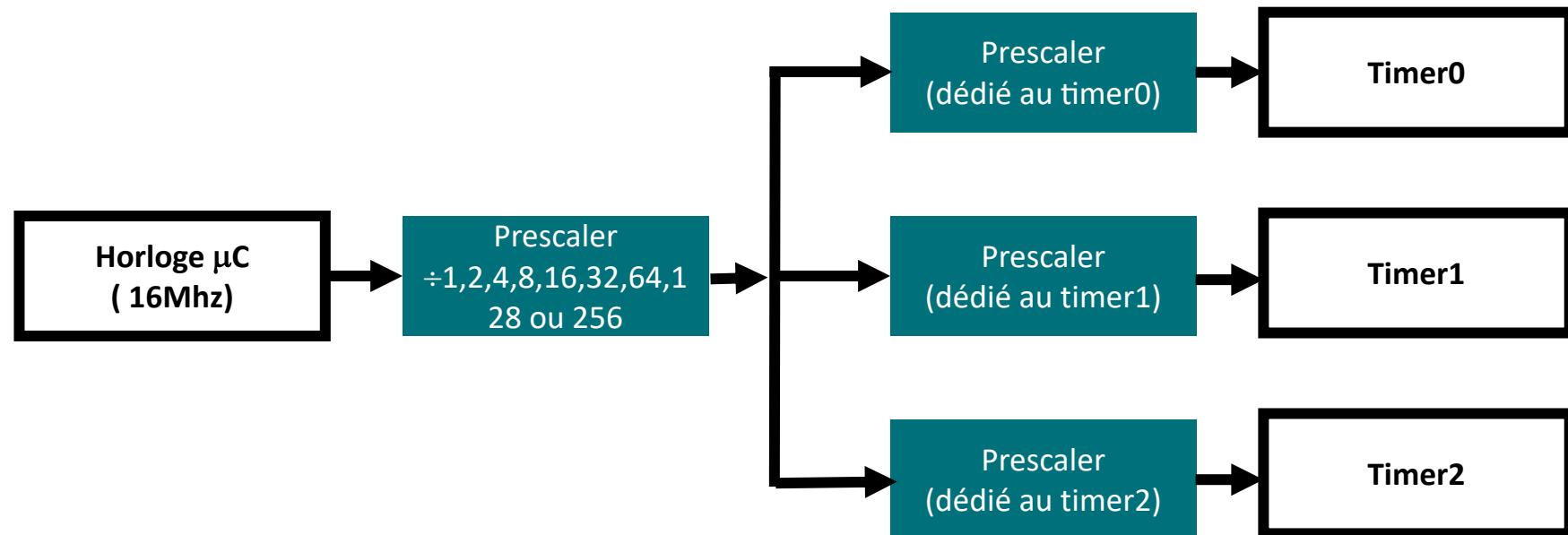
- De nombreuses applications externes :

	taille	utilité
TIMER0	8 bits (compte de 0 à $2^8 - 1 = 255$)	<ul style="list-style-type: none">- <code>delay()</code> : met en pause le processeur pendant X ms- <code>millis()</code> : retourne le nombre de ms depuis le début de l'exécution du code. Retour à 0 au bout d'environ 50 jours.- <code>micros()</code> : retourne le nombre de μs depuis le début de l'exécution du code. Retour à 0 au bout d'environ 70 min.- <code>analogWrite()</code> sur les pins 5 et 6 : signal rectangulaire (PWM)
TIMER1	16 bits (compte de 0 à $2^{16} - 1 = 65\,535$)	<ul style="list-style-type: none">- Bibliothèque Servo- <code>analogWrite()</code> sur les pins 9 et 10 : signal rectangulaire (PWM)
TIMER2	8 bits (compte de 0 à $2^8 - 1 = 255$)	<ul style="list-style-type: none">- Bibliothèque Tone- <code>analogWrite()</code> sur les pins 3 et 11 : signal rectangulaire (PWM)

Principe du Timer

■ Les **timers** reçoivent en entrée un signal d'horloge (16 MHz sur l'Arduino Nano) qui est divisé par un ou deux prescalers (compteurs intermédiaires) :

- ▶ Un **prescaler (diviseur de fréquence « général »)** : permettant de diviser la fréquence du microcontrôleur par 1, 2, 4, 8, 16, 32, 64, 128, ou 256 ;
- ▶ Un **prédiviseur « secondaire » (un pour chaque timer)**, permettant de rediviser la fréquence issue du prescaler.

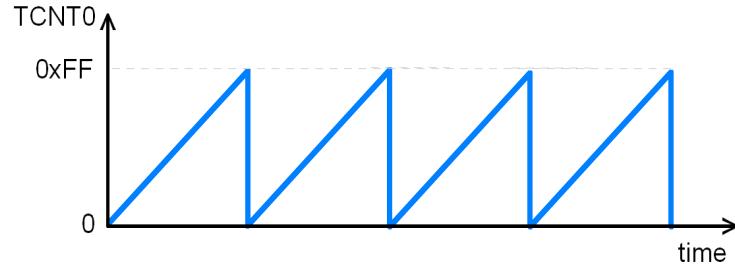


■ Lorsque les Timers atteignent leur valeur maximale (« *overflow* »), ils repassent à 0 et continuent de compter sans interruption.

Modes de fonctionnement

- Deux modes d'opération pour les Timers :

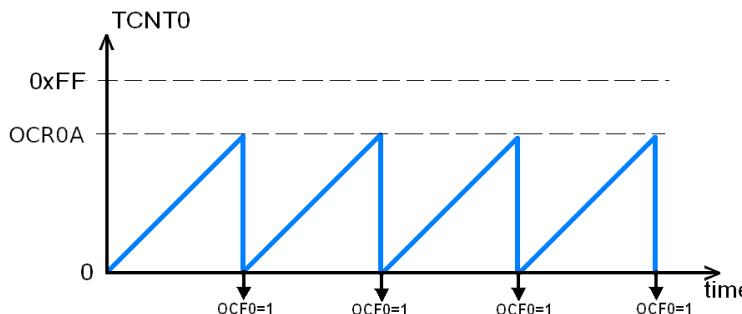
- ▶ **Mode normal** : comptage classique de 0 à sa valeur maximale et rebouclage à 0 une fois la valeur max atteinte



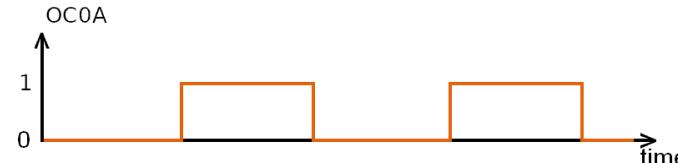
$$T_{cycle} = \frac{\text{prescaler}}{f_{osc}} * \text{valeur maximale}$$

- ▶ **Mode CTC (Clear Timer on Compare Match)** : le domaine de comptage est raccourci, le Timer évolue de 0 à la valeur d'un registre OCR_n (Output Compare Register) et à égalité :

- ▶ le timer revient à 0
 - ▶ le bit OC_nA change d'état



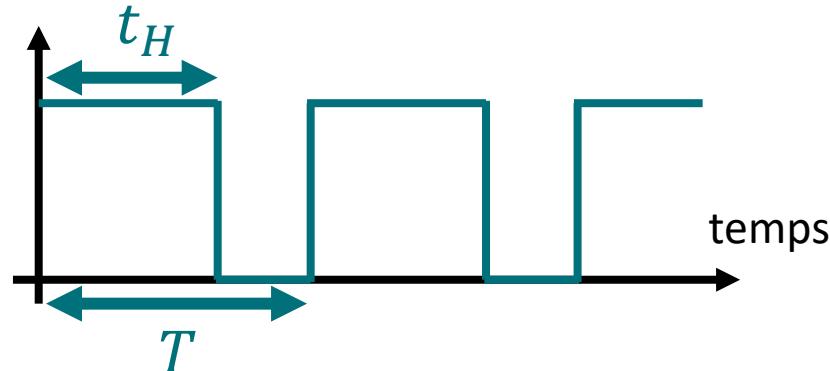
- Fréquence double de celle du comptage



- Signal PWM 😍

Génération de signaux PWM

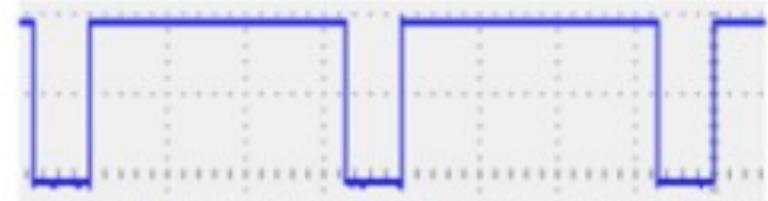
- Pour générer un signal **PWM** (rectangulaire), on utilise la fonction **analogWrite**.
- La fréquence est imposée, et dépend de la pin choisie.
- Le rapport cyclique α est ajustable.



numéro de pin	fréquence [Hz]
3	490
5	980
6	980
9	490
10	490
11	490

$$\alpha = \frac{t_H}{T}$$

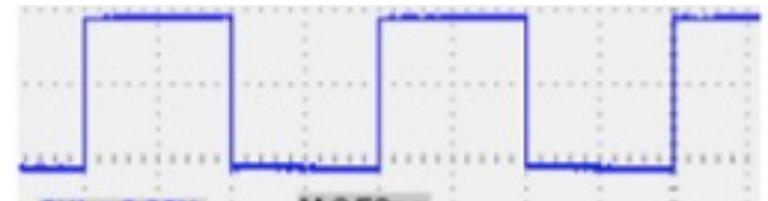
$$\alpha = 0.8$$



`analogWrite(pin,204);`

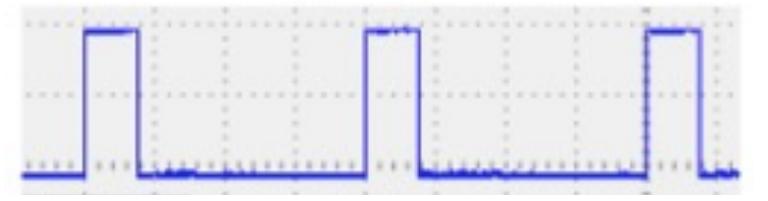
$$\alpha \cdot 255$$

$$\alpha = 0.5$$



`analogWrite(pin,128);`

$$\alpha = 0.2$$



`analogWrite(pin,51);`

Génération de signaux PWM

EXEMPLE 1

Générer une PWM de fréquence 490 Hz et de rapport cyclique 0,25 sur toutes les pins qui le permettent.

numéro de pin	fréquence [Hz]
3	490
5	980
6	980
9	490
10	490
11	490

$$0,25 \cdot 255 = 64$$

```
void setup()
{
    pinMode(3,OUTPUT);
    pinMode(9,OUTPUT);
    pinMode(10,OUTPUT);
    pinMode(11,OUTPUT);
}
void loop()
{
    analogWrite(3,64);
    analogWrite(9,64);
    analogWrite(10,64);
    analogWrite(11,64);
}
```

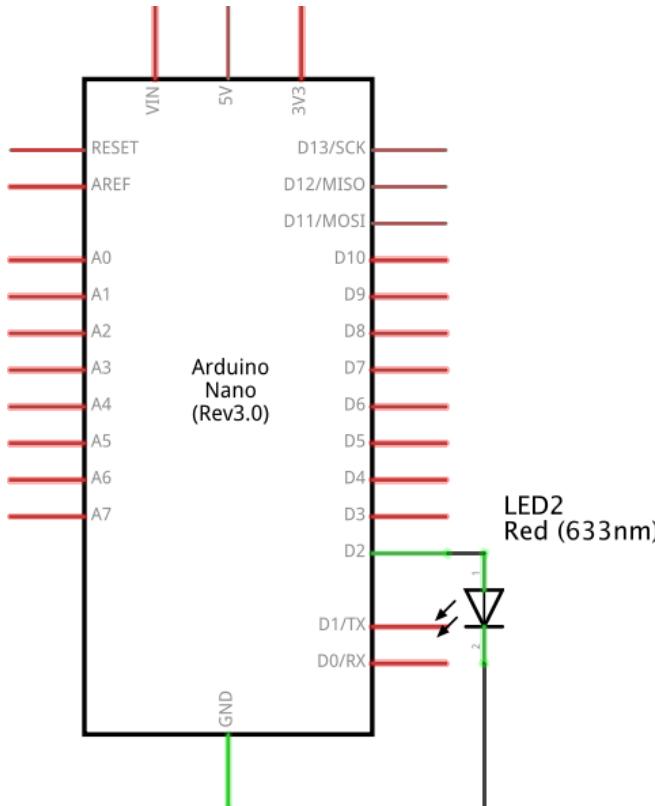
Remarque : il s'agit d'un module dédié donc une fois la ligne de code exécutée, les pins concernées émettent perpetuellement le signal PWM demandé

Génération de signaux PWM

EXEMPLE 2

LED branchée sur la pin 2

Dont la luminosité suit une dent de scie
symétrique, en 51 paliers.



```
const int ledPin = 2;

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

void loop()
{
    for(i=0;i>=255 ; i+=5)
    {
        analogWrite(ledPin,i);
        delay(30);
    }

    for(i=255;i>=0 ; i -=5)
    {
        analogWrite(ledPin,i);
        delay(30);
    }
}
```

La fonction Tone

- La fonction **Tone** permet de générer des signaux rectangulaires de fréquences autres que 490 et 980 Hz, mais à rapport cyclique imposé, de 50%.
- Syntaxes :

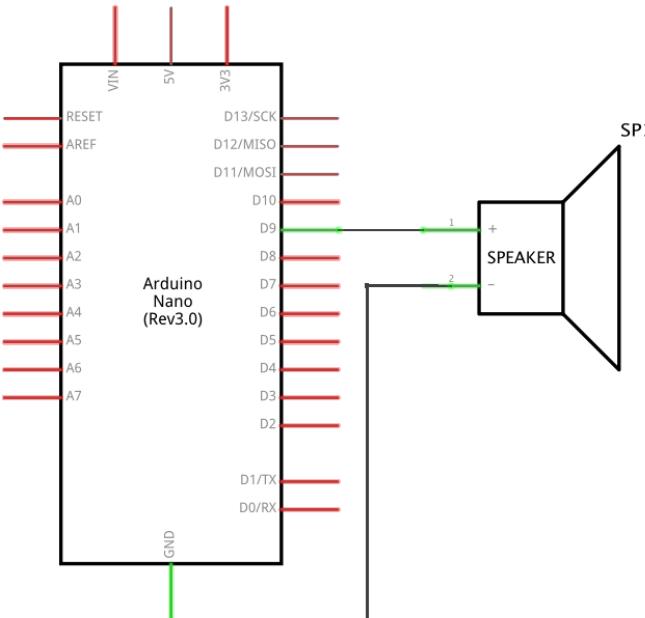
```
tone(pin, frequence);
```

ou

```
tone(pin, frequence, dureeMilliSec);
```

EXEMPLE

Jouer un LA 3 sur la pin 9



```
const int pinBuzzer = 9;  
  
void setup()  
{  
    pinMode(pinBuzzer, OUTPUT);  
}  
  
void loop()  
{  
    tone(pinBuzzer, 440);  
}
```

5 Communication série

Le module de communication série

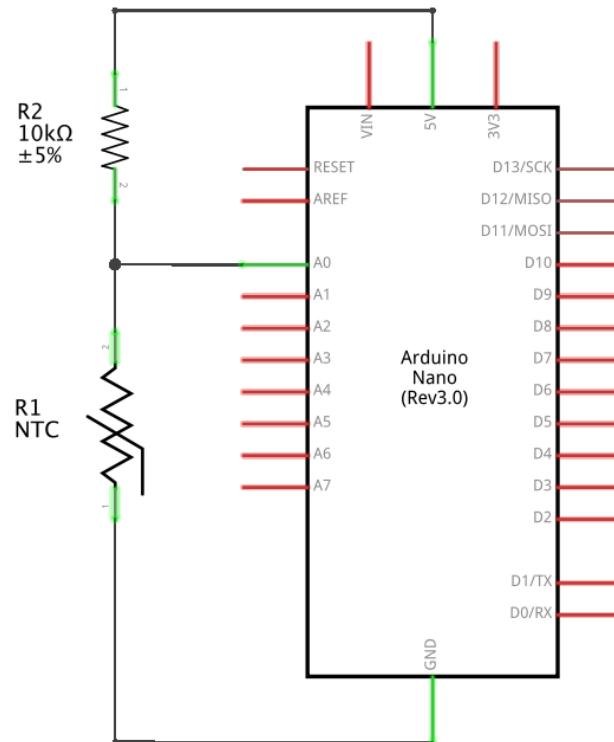
- Le module de communication série (UART) permet d'envoyer et de recevoir des données en série.
- En particulier, on peut envoyer des données dans le moniteur série (PC) pour vérifier ce qui se fait sur le microcontrôleur.

💀 Attention à bien brancher les pins RX (Receive) et TX (Transmit) sur le récepteur et l'émetteur...

EXEMPLE

Lire la tension entre la pin A0
et la masse

L'envoyer dans le port série



```
void setup()
{
    int sensorValue = 0;
    float voltage = 0;
    pinMode(A0, INPUT);
    Serial.begin(9600);
}

void loop()
{
    sensorValue = analogRead(A0);
    voltage = sensorValue * 5.0 / 1023.0;
    Serial.println(voltage);
}
```

Serial plotter

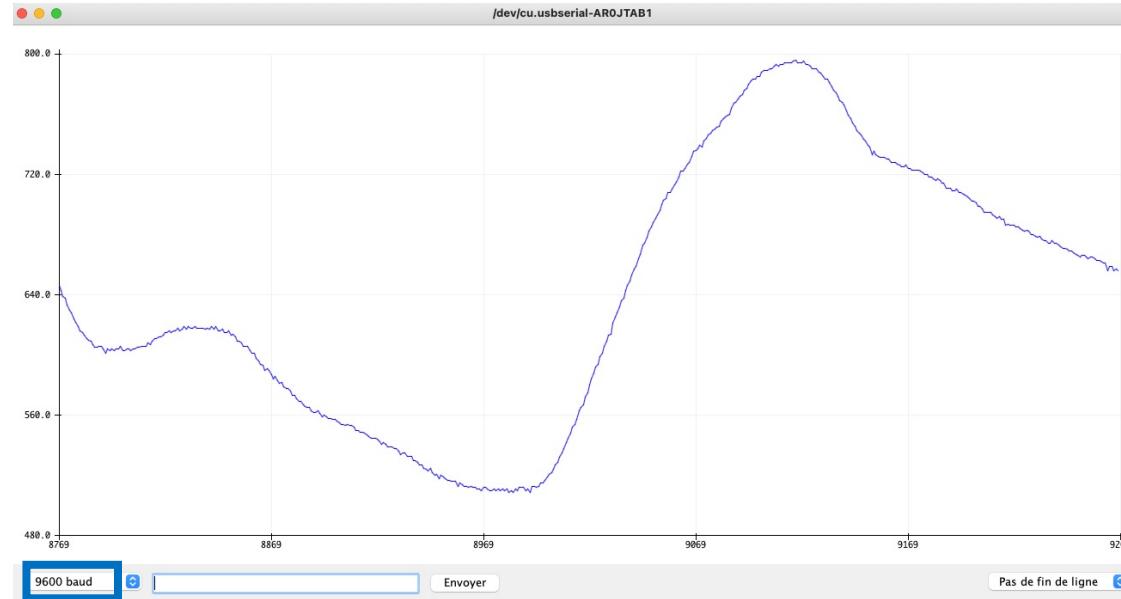
Sur Arduino IDE – tracé d'une courbe



Utiliser `Serial.println()` et pas
`Serial.print()`

```
Arduino Fichier Édition Croquis Outils Aide
projetArduino
1 #include <Arduino.h>
2
3 int maVariable=0;
4
5 void setup() {
6   pinMode(A0, INPUT);
7   Serial.begin(9600);
8 }
9
10 void loop() {
11   maVariable=analogRead(A0);
12   Serial.println(maVariable);
13   delay(1000);
14 }
15
```

The screenshot shows the Arduino IDE interface. The menu bar is visible with options like Arduino, Fichier, Édition, Croquis, Outils, and Aide. A context menu is open over the code editor, with "Traceur série" (Serial Plotter) highlighted in yellow. The code itself is a simple sketch that reads an analog signal from pin A0, prints its value to the serial port at 9600 baud, and then waits for 1000ms before repeating.



! L'axe des x correspond au n° d'échantillon et pas des secondes !

Remarque 1 : adapter le baud rate (vitesse de transmission en bit par seconde) en l'augmentant si le phénomène est rapide, ou en le ralentissant si le phénomène est lent

Remarque 2 : Pas besoin de téléverser le code depuis Arduino IDE pour s'en servir

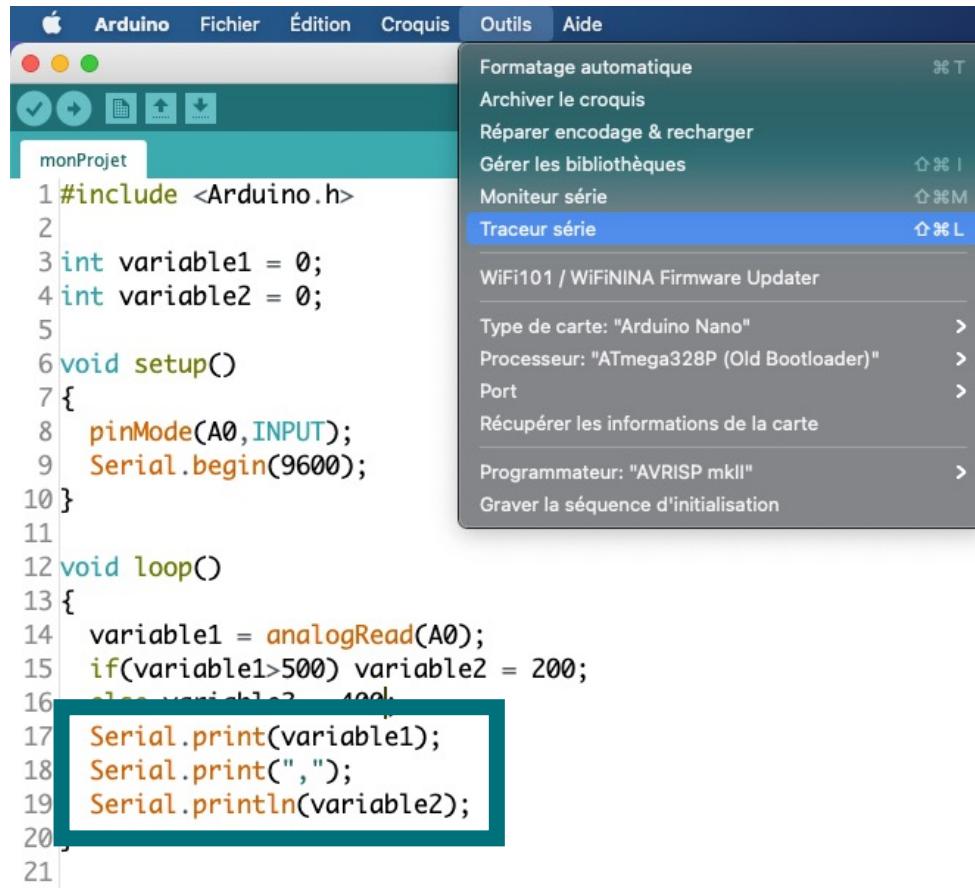
Remarque 3 : faire un produit en croix si nécessaire pour bien voir le phénomène en pleine échelle (utiliser un float et éviter map() qui renvoie un int si les décimales sont décisives).

$\times \frac{100}{1024}$ pour obtenir la grandeur sur une échelle de 0 à 100

$\times \frac{5}{1024}$ pour obtenir le résultat sous forme de tension si $V_{REF} = 5 V$)

Serial plotter

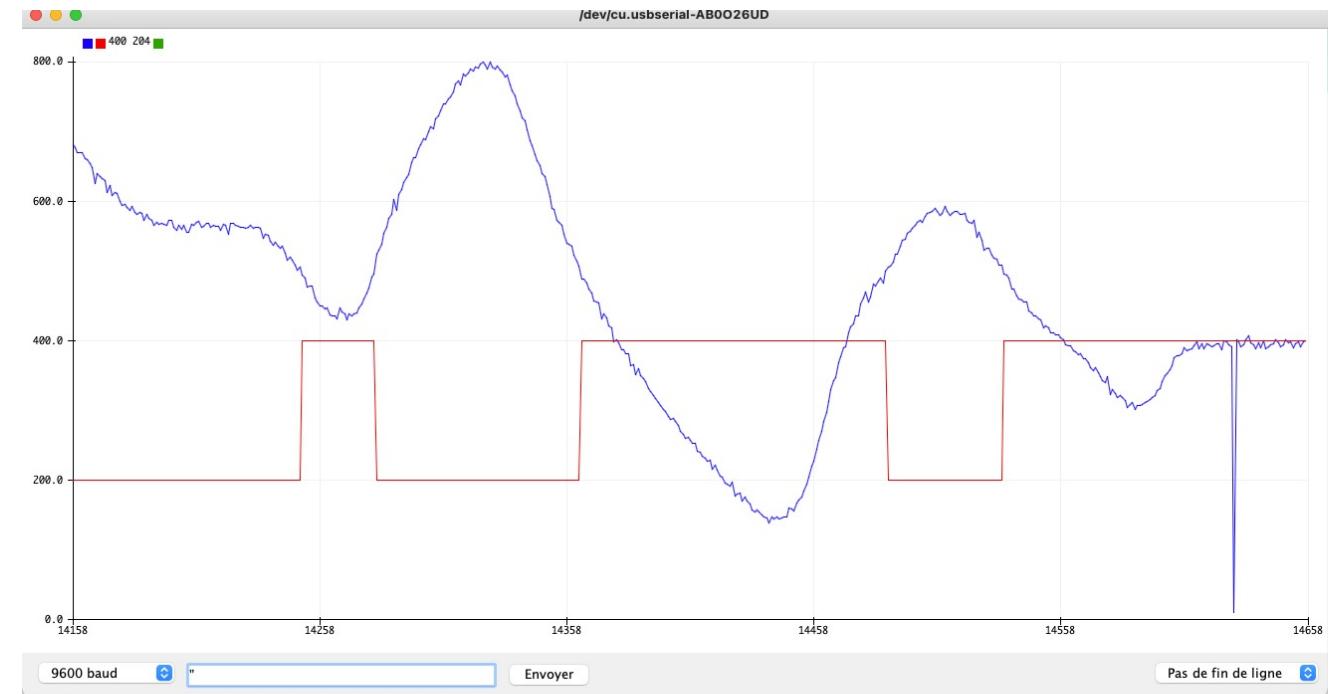
Sur Arduino IDE – tracé de plusieurs courbes



```
#include <Arduino.h>
int variable1 = 0;
int variable2 = 0;

void setup()
{
    pinMode(A0, INPUT);
    Serial.begin(9600);
}

void loop()
{
    variable1 = analogRead(A0);
    if(variable1>500) variable2 = 200;
    else variable2 = 400;
    Serial.print(variable1);
    Serial.print(",");
    Serial.println(variable2);
}
```



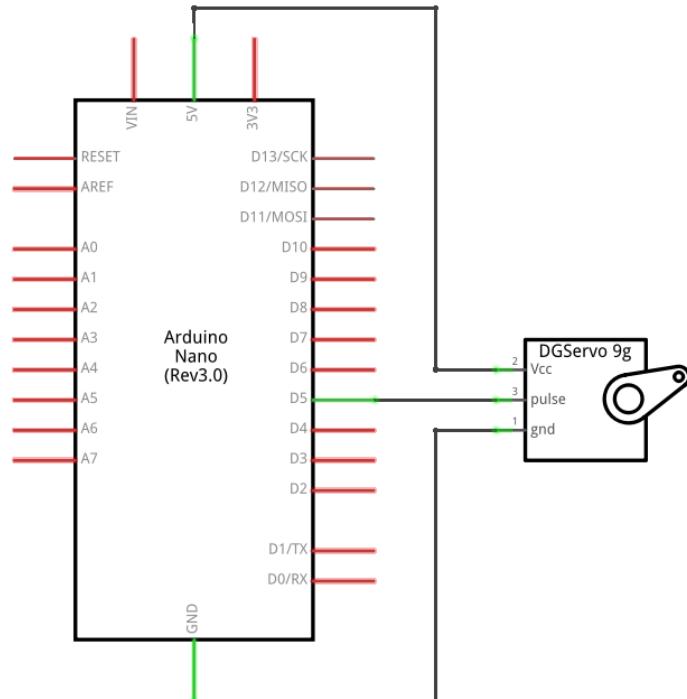
6 Les bibliothèques et les fonctions mathématiques

Utilisation d'une bibliothèque

- L'emploi des **bibliothèques** de fonction se fait à l'aide de l'instruction `#include<nomBibliotheque.h>`

EXEMPLE

**Servomoteur branché sur la pin 5
qui fait un mouvement de balayage.**



```
#include <Servo.h>
servo monServo;

void setup()
{
    monServo.attach(5);
}

void loop()
{
    for(int pos = 0 ; pos <= 180 ; pos++)
    {
        monServo.write(pos);
        delay(15);
    }

    for(int pos = 0 ; pos <= 180 ; pos++)
    {
        monServo.write(pos);
        delay(15);
    }
}
```

Fonctions mathématiques

■ Quelques fonctions mathématiques bien utiles :

- `min(x,y)`
- `max(x,y)`
- `abs(x)`
- `pow(base,exposant)`
- `sq(x)`
- `sqrt(x)`
- `constrain(x,min,max)`
 - Contraint un nombre à rester dans un intervalle :
 - Renvoie x si x est compris entre min et max
 - Renvoie min si x < min
 - Renvoie max si x > max
- `map(valuer,fromLow, fromHigh, toLow, toHigh)`
 - Ré-étalonne un nombre d'une fourchette de valeur vers une autre fourchette.
 - **Remarque : la fonction map retourne un entier. La partie décimale est tronquée.**

EXAMPLE

Lire ce que retourne l'ADC sur la pin A0 et ré-étalonner de sorte à avoir un pourcentage.

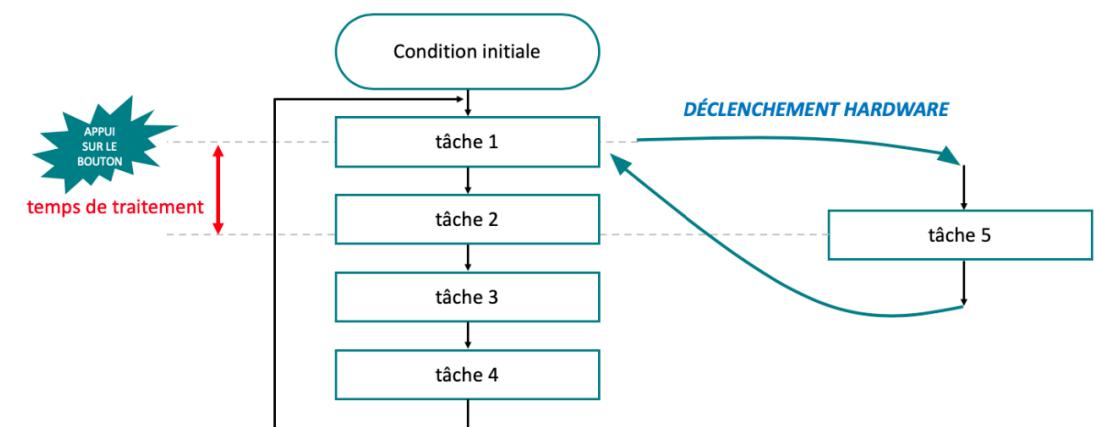
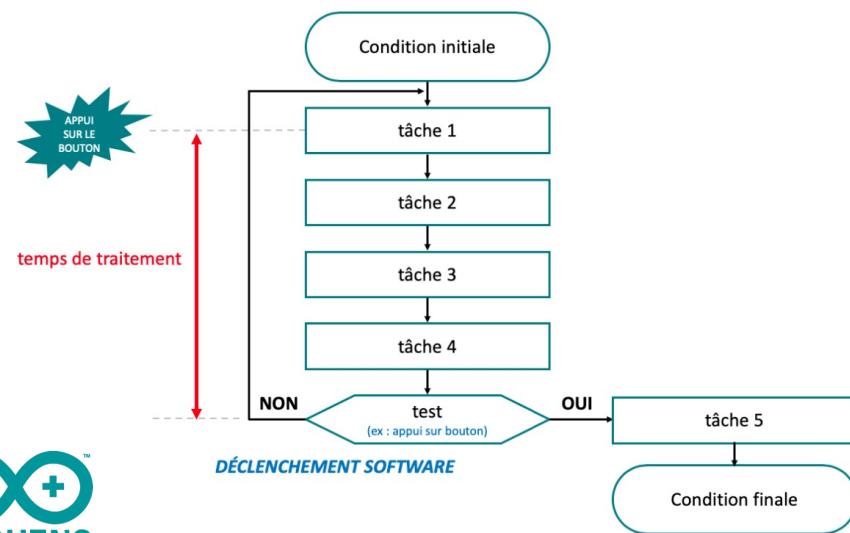
```
val = analogRead(A0);  
pourcent = map(val, 0, 1023, 0, 100);
```

7

Les fonctions d'interruption

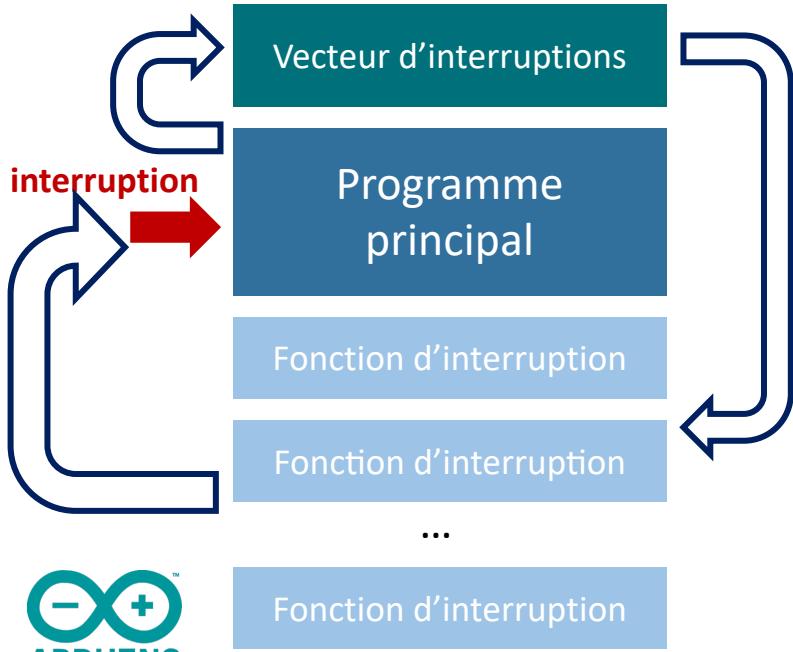
Programmer des fonctions d'interruptions

- Les **fonctions d'interruption** permettent d'interrompre le processeur afin de traiter d'autres tâches.
- Leur déclenchement est **Hardware** et non **Software** (code), ce qui veut dire que pour un microcontrôleur donné, il y a autant de sous-circuits internes qui viennent électriquement regarder si l'événement de déclenchement de la fonction d'interruption a lieu.
- Contrairement au *polling* (écoute active), le temps de traitement est plus court, et le déclenchement de la fonction est plus fiable (si cherche à détecter un appui sur bouton alors que le if() en question est plusieurs lignes de code plus loin, on peut rater l'événement).
- À la manière d'un appel de fonction, l'appel des fonctions d'interruption implique un enregistrement de la ligne de code d'appel (copie du contenu du program counter dans la pile) et un dépilement du contenu de la pile pour pouvoir revenir à l'endroit où a été appelée la fonction.



Programmer des fonctions d'interruptions

- L'ATMega328P possède **26 sources d'interruption**.
- Lorsqu'une interruption est détectée, le compteur de programme pointe vers le **vecteur d'interruption**, qui renvoie à son tour vers la fonction d'interruption dédiée.
- Une fois la fonction d'interruption exécutée, on revient dans le programme principal comme lors d'un appel de fonction (dépilement de l'adresse d'appel).
- Exemples d'utilisation : interruption sur timer, interruption sur ADC, etc.



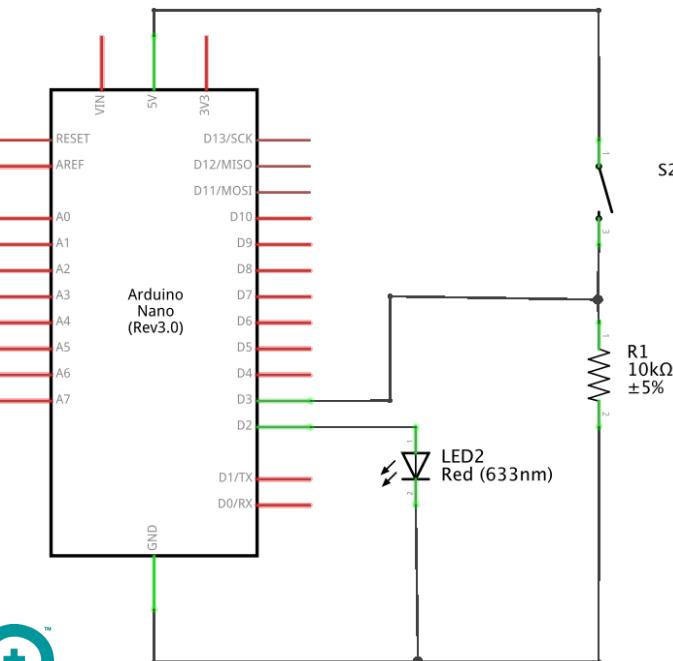
No	Source	Description de la source d'interruption	Nom du vecteur
1	RESET	Reset	
2	INT0	External Interrupt Request 0 (pin D2)	(INT0_vect)
3	INT1	External Interrupt Request 1 (pin D3)	(INT1_vect)
4	PCINT0	Pin Change Interrupt Request 0 (D8 to D13)	(PCINT0_vect)
5	PCINT1	Pin Change Interrupt Request 1 (A0 to A5)	(PCINT1_vect)
6	PCINT2	Pin Change Interrupt Request 2 (D0 to D7)	(PCINT2_vect)
7	WDT	Watchdog Time-out Interrupt	(WDT_vect)
8	TIMER2 COMPA	Timer/Counter2 Compare Match A	(TIMER2_COMPA_vect)
9	TIMER2 COMPB	Timer/Counter2 Compare Match B	(TIMER2_COMPB_vect)
10	TIMER2 OVF	Timer/Counter2 Overflow	(TIMER2_OVF_vect)
11	TIMER1 CAPT	Timer/Counter1 Capture Event	(TIMER1_CAPT_vect)
12	TIMER1 COMPA	Timer/Counter1 Compare Match A	(TIMER1_COMPA_vect)
13	TIMER1 COMPB	Timer/Counter1 Compare Match B	(TIMER1_COMPB_vect)
14	TIMER1 OVF	Timer/Counter1 Overflow	(TIMER1_OVF_vect)
15	TIMO0 COMPA	Timer/Counter0 Compare Match A	(TIMERO_COMPA_vect)
16	TIMO0 COMPB	Timer/Counter0 Compare Match B	(TIMERO_COMPB_vect)
17	TIMO0 OVF	Timer/Counter0 Overflow	(TIMERO_OVF_vect)
18	SPI, STC	SPI Serial Transfer Complete	(SPI_STC_vect)
19	USART, RX	USART, Rx Complete	(USART_RX_vect)
20	USART, UDRE	USART, Data Register Empty	(USART_UDRE_vect)
21	USART, TX	USART, Tx Complete	(USART_TX_vect)
22	ADC	ADC Conversion Complete	(ADC_vect)
23	EE READY	EEPROM Ready	(EE_READY_vect)
24	ANALOG COMP	Analog Comparator	(ANALOG_COMP_vect)
25	TWI	2-wire Serial Interface	(I2C)(TWI_vect)
26	SPM READY	Store Program Memory Ready	(SPM_READY_vect)

Interruptions externes

EXEMPLE

LED branchée sur la pin 2, bouton en pull-down sur la pin 3

LED qui s'allume pendant 5 sec si l'on appuie sur le bouton, en utilisant une fonction d'interruption



```
const byte ledPin = 2;
const byte interruptPin = 3;
volatile byte state = LOW;
void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(interruptPin, INPUT_PULLUP);
    attachInterrupt(1, blink, CHANGE);
    // 0 pour INT0 (sur pin2 de l'arduino)
    // 1 pour INT1 (sur pin3 de l'arduino)
    // ou digitalPinToInterrupt(3) pour la pin 3
}
void loop(){}
void blink()
{
    state = !state;
    digitalWrite(ledPin, state);
}
```

Résumé des principales fonctions

Fonction	Description
<code>void setup(){};</code>	Procédure de paramétrage du microcontrôleur
<code>void loop(){};</code>	Procédure exécutée à l'infini
<code>pinMode(pin,etat) ;</code>	Permet de paramétrer les pins en INPUT, INPUT_PULLUP ou OUTPUT
<code>variable = digitalRead(pin) ;</code>	Permet de lire un état logique LOW ou HIGH
<code>digitalWrite(pin, valeur) ;</code>	Permet d'écrire un état logique LOW ou HIGH
<code>variable = analogRead(pin) ;</code>	Permet de convertir une entrée analogique, retourne un int entre 0 et 1023
<code>analogWrite(pin,rapportCyclique);</code>	Permet de générer un signal PWM de 490 ou 980 Hz de rapport cyclique défini
<code>Tone(pin,frequence,duree);</code>	Permet de générer un signal rectangulaire de fréquence définie à rap. cyclique 0,5
<code>noTone(pin);</code>	Permet l'arrêt de la génération du signal PWM émis par Tone
<code>delay(milliSec);</code>	Met en pause (bloque) le processeur pendant un temps en millisecondes
<code>delayMicroSeconds(microSec);</code>	Met en pause (bloque) le processeur pendant un temps en microsecondes
<code>tic = millis();</code>	Retourne le temps écoulé depuis le début de l'exécution du code, en millisecondes
<code>tac = micros();</code>	Retourne le temps écoulé depuis le début de l'exécution du code, en microsecondes
<code>Serial.begin(9600);</code>	Initie la communication série à une vitesse de transmission de 9 600 bit / seconde
<code>Serial.println(valeur);</code>	Écrit dans le port série et fait un retour chariot
<code>val2 = map(val1, fromMin, fromMax, toMin, toMax);</code>	Ré-étalonne une valeur d'un intervalle vers un autre
<code>Val2 = constrain(val1,min,max);</code>	Contraint un nombre à rester dans un intervalle