UNIVERSITÉ DE LYON

FACULTÉ des Sciences et Technologies

Robotique

Responsable : Sophie Cavassila
Sophie.cavassila@univ-lyon1.fr
28/06/2019

Table des matières

Chapitre 1 Introduction : les microcontrôleurs

Les systèmes embarqués correspondent à des équipements souvent autonomes avec une «intelligence» qui leur permet d'être en interaction directe avec l'environnement dans lequel ils sont placés.

Les systèmes embarqués sont présents dans des applications de plus en plus nombreuses:

- Systèmes mobiles communicants: robotique, téléphones mobiles, récepteurs GPS,
- Électronique grand public, électroménager, domotique
- automobile, transport aérien/maritime/fluvial.
- Santé

Ces applications embarquées nécessitent des systèmes de contrôle/commande intelligents qui intègrent le maximum de fonctions dans un même circuit (diminuer le nombre de composants des systèmes et leur coût global).

Les cartes Arduino à base de microcontrôleurs sont **largement répandues en robotique** et présentent les fonctionnalités suivantes :

Un microcontrôleur intègre les éléments suivants:

- Une unité centrale (CPU ″Central Processing Unit″) qui est le circuit effectuant les calculs et prenant les décisions logiques. L'unité centrale est cadencée par une horloge système.
- Des mémoires qui stockent les codes opératoires des programmes et éventuellement des données qui sont mises à jour pendant l'exécution du programme
- Un ensemble de périphériques tels que des ports de communications numériques assurant la communication du microcontrôleur avec le monde extérieur, convertisseur analogique numérique, des circuits de temporisation, des unités de modulation de largeurs d'impulsion (PWM), des communication série (SPI, I2C, UART,…).

Les cartes Arduino

La carte Arduino Uno est basée sur un ATMega328 cadencé à 16 MHz.
Elle dispose :

- de 14 broches numériques d'entrées/sorties (dont 6 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée)),
- de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
- d'une horloge interne 16Mhz,
- et d'un bouton de réinitialisation (reset).

La carte Arduino Mega 2560 est basée sur un ATMega2560 cadencé à 16 MHz.
Elle dispose :

- de 54 broches numériques d'entrées/sorties (dont 14 peuvent être utilisées en sorties PWM (largeur d'impulsion modulée)),
- de 16 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
- de 4 UART (port série matériel),
- d'une horloge interne 16Mhz,
- d'une connexion USB,

Bibliographie

https://store.arduino.cc/arduino-uno-rev3
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.MaterielUno
https://store.arduino.cc/mega-2560-r3
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.MaterielMega2560)

Chapitre 2 L'environnement de développement Arduino et éléments de programmation

L'IDE Arduino (Arduino Integrated Development Environment (IDE)) est l'environnement de développement dédié aux cartes Arduino.
Le fichier d'installation de l'environnement de développement peut être téléchargé en ligne:
https://www.arduino.cc/en/Main/Software#download

L'environnement de développement Arduino IDE est disponible pour les systèmes d'exploitation: Windows, Mac OS X et Linux.

Avec un support technique ici :
https://www.arduino.cc/en/Guide/HomePage
https://wiki.mdl29.net/lib/exe/fetch.php?media=robotsarduino:installation_arduino_ardublock_windows.pdf

a) Eléments de programmation

Le langage de programmation Arduino est proche du langage C. L'ensemble des éléments de programmation sont disponibles sur le site https://www.arduino.cc/reference/en/#structure

La structure minimale d'un programme est constituée de deux fonctions setup() et loop()

```
void setup() // fonction d'initialisation des périphériques utilisés
{
}

void loop()  // fonction principale décrivant l'algorithme du programme
{
}
```

La fonction **setup()** permet l'initialisation des périphériques utilisés. Elle est exécutée une seule fois lors de l'initialisation de la carte. Par exemple, les directions des ports de communication numériques seront configurés ici.

La fonction **loop**() décrit l'algorithme du programme. Elle se répète à l'infini.

Ces fonctions sont précédées d'un entête où sont déclarées les constantes et les fonctions des bibliothèques :

#define permet de définir des constantes

Exemple : #define ledPin 3 associe la valeur 3 à la constante ledPin.

#include permet de déclarer les bibliothèques de fonctions utilisées dans le programme

Exemple : #include <Servo.h> permet d'utiliser les fonctions pour la commande de servomoteurs.
De nombreuses bibliothèques sont présentées ici :https://www.arduino.cc/en/Reference/Libraries

Les variables et constantes

Une variable est un espace réservé en mémoire pour stocker une donnée. Une variable peut être locale (existe uniquement dans la fonction dans laquelle elle a été déclarée) ou globale (existe pour les fonctions du programme).

Type	Grandeur	Plage des données	Taille en octets
(signed) int	entière	-32768 à +32767	2
unsigned int	entière	0 à 65535	2
(signed) long	entière	-2 147 483 648 à +2 147 483 647	4
unsigned long	entière	0 à 4 294 967 295	4
(signed) char	entière	-128 à +127	1
unsigned char ou byte	entière	0 à 255	1
float	décimale	-3.4028235E+38 à +3.4028235E+38	4
double pour Arduino Uno et Mega	décimale	-3.4028235E+38 à +3.4028235E+38	4
bool	binaire	false ou true	1

Les variables peuvent être exprimées sans conséquence dans les bases binaires, décimale ou hexadécimale.

Exemple :
char valeur ;
valeur=13 ; ou valeur =0x0D ; ou valeur=0b00001101

Les constantes:
false est équivalent à '0'
true est souvent défini comme '1' mais représente en fait tout ce qui n'est pas '0'.

Les opérateurs et les fonctions

Les opérateurs arithmétiques

Opérateur	**Exemples**
* (multiplication)	2*3 égal 6
+ (addition)	2+3 égal 5
- (soustraction)	2-3 égal -1
/ (division entière ou réelle)	5/2 égal 2 (division entière) 5.0/2 égal 2,5 (division réelle)
% (reste de la division)	5%2 égal 1 (reste de la division entière)
= (affectation)	permet l'initialisation d'une variable

Les opérateurs relationnels

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

Les opérateurs booléens

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

Les pointeurs

& (reference operator) * (dereference operator)

Les opérateurs de bits

& (bitwise and) << (bitshift left) >> (bitshift right)

^ (bitwise xor) | (bitwise or) ~ (bitwise not)

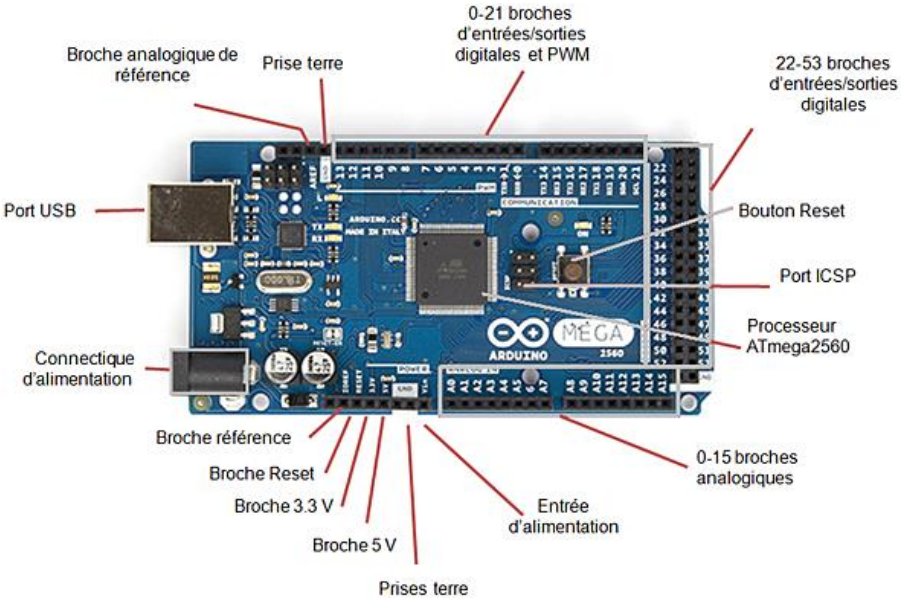
Les structures de contrôle

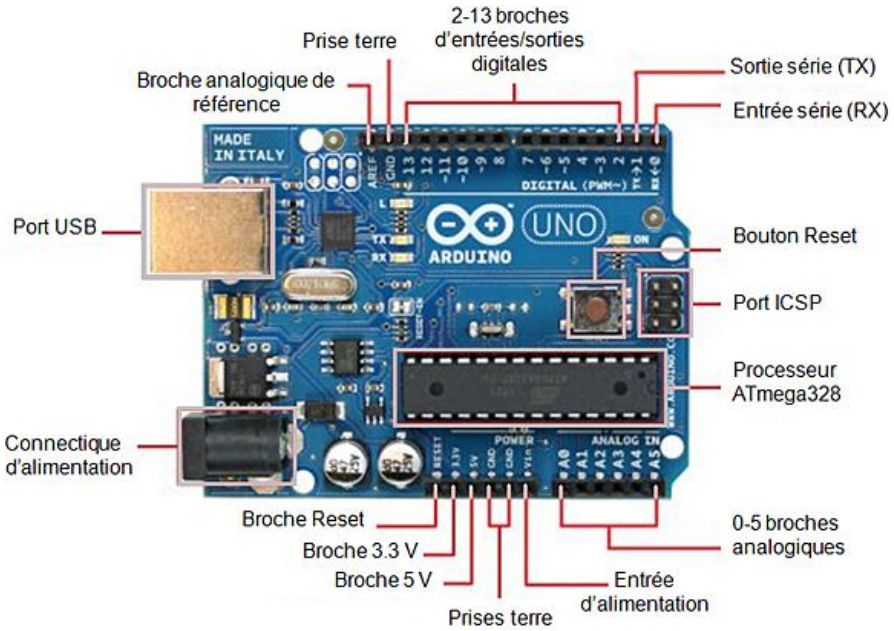
Les boucles de répétition avec test : do...while while
Les boucles de répétition avec compteur: for
Les choix et alternatives: if else switch...case

Chapitre 3 Les entrées et sorties numériques : Les capteurs / Les actionneurs

La carte de développement Arduino Mega dispose de: 54 broches de communications d'entrées/ sorties numériques structurées en ports (14 broches d'entrées/sortes numériques pour Uno). Ces broches peuvent être adressées en entrée ou en sortie (broches bidirectionnelles), individuellement ou par groupe de 8 (port 8 bits).

Le niveau électrique des broches peut être LOW (0 Volt) ou HIGH (5Volts). Chaque broche fournit ou reçoit un maximum de 40 mA. Des résistances de protection (pull-up) doivent être évaluées pour chaque broche que le courant consommé ne dépasse pas cette valeur.





a) Configuration des broches d'entrée/sortie numériques

Configuration en sortie: l'information est envoyée par le microcontrôleur vers le monde extérieur (actionneurs, moteurs).

Configuration en entrée: les broches sont initialisées par le monde extérieur (capteurs) et sont lues par le microcontrôleur.

Broches configurées en sortie

Broches configurées en entrée

La configuration en entrée (INPUT) ou sortie (OUTPUT) est obtenue par la fonction: pinMode()

Syntax : pinMode(pin, mode)

Parameters

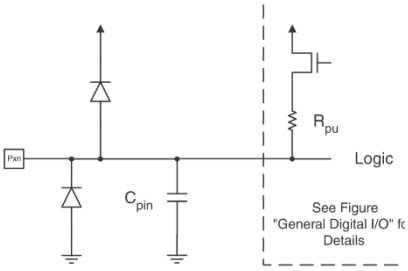
pin: the Arduino pin number to set the mode of.

mode: INPUT, OUTPUT, or INPUT_PULLUP. See the Digital Pins page for a more complete description of the functionality.

Returns

Nothing

Chaque broche dispose d'une résistance de rappel (pull-up) de 20-50kOhms qui est déconnectée par défaut.
Chaque broche est associée à des diodes de protection.



Exemple :

pinMode(13, OUTPUT) permet de configurer la broche n°13 en sortie.

b) Les données échangées

Le niveau électrique d'une **broche configurée en entrée** est lue par le microcontroleur par la fonction digitalRead()

Syntax digitalRead(pin)
Parameters : pin: the Arduino pin number you want to read
Returns : HIGH or LOW

Un niveau haut (HIGH) est mesuré si le niveau électrique est supérieur à 3.0V (5V boards) ou 2.0V volts (3.3V boards).
Un niveau bas (LOW) est mesuré si le niveau électrique est inférieur à 1.5V (5V boards) ou 1.0V volts (3.3V boards).

Le niveau électrique d'une **broche configurée en sortie** est fixée par le microcontroleur: digitalWrite()

Syntax digitalWrite(pin, value)
Parameters
pin: the Arduino pin number.
value: HIGH or LOW.
Returns Nothing

Exemple :
Ce programme permet de faire clignoter la led connectée à la broche 13 (visible sur la carte de développement) avec une fréquence de 0,5Hz.

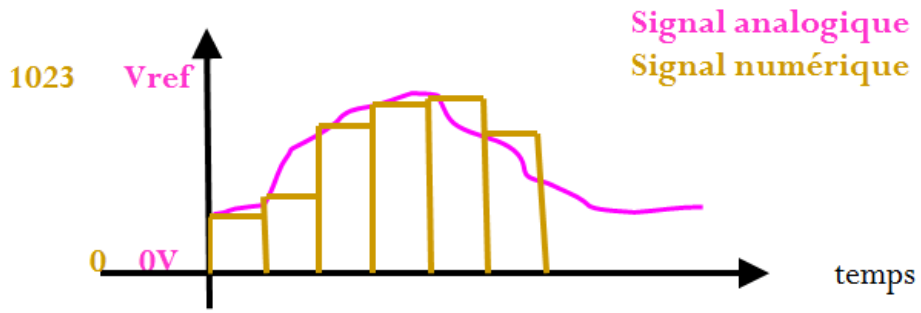

```
int led 13;

void setup() {
  pinMode(led, OUTPUT);    // Configure la broche 13 en sortie
}
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);    // Allume la LED
  delay(1000);                // Attend une seconde
  digitalWrite(led, LOW);     // Eteint la LED
  delay(1000);                // Attend une seconde
}
```

Chapitre 4 Les entrées analogiques : Les capteurs analogiques

Les microcontrôleurs intègrent un convertisseur Analogique Numérique (CAN ou ADC Analog to Digital Converter).

Un tel dispositif est chargé de convertir une tension analogique en une valeur numérique (séquence binaire codée sur 4, 8, 10,12 bits ...).

Ces valeurs numériques sont proportionnelles aux valeurs analogiques.



La plage des tensions analogiques est définie par la tension de référence. La valeur de la tension de référence doit être configurée et est choisie égale à 5V ou 1,1V pour une référence interne.

Les valeurs numériques délivrées par le convertisseur analogique numérique sont dans l'intervalle 0-15 pour un CAN 4 bits, 0-255 pour un CAN 8 bits, 0-1023 pour un CAN 10 bits.

Le quantum, q, représente la résolution du convertisseur. La relation tension analogique/valeur numérique pour un convertisseur 10 bits est la suivante:

$$ADC = \frac{Vin}{q} = \frac{Vin}{Vref} 2^{10} = \frac{Vin}{Vref} 1024$$

Vref est la tension de référence.

Vin est la tension d'entrée (gamme 0-(Vref-q)).

ADC est un nombre entier sur 10 bits.

a) Configuration de la tension de référence du convertisseur analogique-numérique broches d'entrée/sortie numériques

La tension de référence est configurée par la fonction: analogReference()

Syntax

`analogReference(type)`

Parameters

type: which type of reference to use (see list of options in the description).

Returns Nothing

Arduino AVR Boards (Uno, Mega, Leonardo, etc.)

DEFAULT: the default analog reference of 5 volts (on 5V Arduino boards) or 3.3 volts (on 3.3V Arduino boards)

INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 or ATmega328P and 2.56 volts on the ATmega32U4 and ATmega8 (not available on the Arduino Mega)

INTERNAL1V1: a built-in 1.1V reference (Arduino Mega only)

INTERNAL2V56: a built-in 2.56V reference (Arduino Mega only)

EXTERNAL: the voltage applied to the AREF pin (0 to 5V only) is used as the reference.

b) Lecture de la valeur numérique délivrée par le convertisseur analogique numérique

La lecture de la valeur numérique résultat de la conversion analogique- numérique de la tension analogique présente sur le canal spécifié est réalisée par la fonction: analogRead()

Syntax

`analogRead(pin)`

Parameters : `pin`: the name of the analog input pin to read from (A0 to A5 on most boards, A0 to A6 on MKR boards, A0 to A7 on the Mini and Nano, A0 to A15 on the Mega).

Returns

The analog reading on the pin. Although it is limited to the resolution of the analog to digital converter (0-1023 for 10 bits or 0-4095 for 12 bits). Data type: `int`.

c) Example Code

Le programme ci-dessous permet la conversion de la tension présentée sur la broche A3 et affiche la valeur numérique correspondante sur l'écran LCD.

```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

int analogPin = A3; // potentiometer wiper (middle terminal) connected to analog pin 3
                    // outside leads to ground and +5V
int val = 0;  // variable to store the value read

void setup() {
   // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
lcd.setCursor(0, 0);
lcd.print("val= “);
lcd.print(val);
}
```

Chapitre 5 Communication série

Le microcontroleur peut échanger des informations avec d'autres microcontrôleurs ou périphériques (écrans, capteurs numériques, …) via des communications série.

Lorsqu'une communication "série" a été initialisée, les données sont transmises (émises et reçues) bit à bit. Ces communications ont l'avantage de monopoliser peu de broches pour la communication. Ces communications série sont très largement utilisées.

Les protocoles série:

- **USART: Universal Synchronous and Asynchronous serial Receiver and Transmitter**
- **SPI: Serial Peripheral Interface**
- **TWI (I2C): Two-Wire serial Interface(jusqu'à 400kHz)**

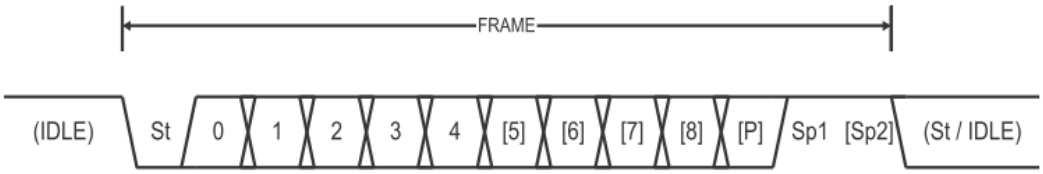
Une transmission série de données est caractérisée par :

- le sens des échanges:
  - simplex : **La liaison simplex** caractérise une liaison dans laquelle les données circulent dans **un seul sens**, c'est-à-dire de l'émetteur vers le récepteur
  - half-duplex : **La liaison half-duplex** caractérise une liaison dans laquelle les données **peuvent circuler dans un sens ou l'autre, mais pas dans les deux sens simultanément**. Avec ce type de liaison chaque extrémité de la liaison émet à son tour.
  - full-duplex : **La liaison full-duplex** caractérise une liaison dans laquelle les données circulent **de façon bidirectionnelle et simultanément**. Chaque extrémité de la ligne peut émettre et recevoir en même temps.

- le type de synchronisation entre l'émetteur et le récepteur:
  - **liaison asynchrone:** les données transmises sont mise en forme avec l'ajout de bits de début et fin de transmission. L'émetteur et le récepteur doivent être configurés de façon à émettre et lire les données à la même vitesse.

    Le récepteur reçoit l'information au rythme où l'émetteur les envoie. La liaison série est constituée de 2 supports nommés TX et RX :
    - La ligne de transmission des données TX
    - La ligne de réception des données RX.

    Chaque émission d'une information est constituée d'une séquence binaire que l'on appelle *format.*
    - une information indiquant le début de la transmission (*bit START*)
    - Les 5,6,7, 8, 9 bits d'information
    - une information permettant de contrôler d'éventuelles erreurs de transmission (bit de parité logique)
    - Suivi d'une information de fin de transmission (1 ou 2 *bit STOP*)



  - **liaison synchrone:** L'émetteur et le récepteur sont synchronisés par un signal d'horloge émis par l'émetteur. Le signal de synchronisation est actif pendant toute la durée de l'émission informant ainsi le récepteur qu'une transmission d'information est en cours. La liaison série est constituée de 3 supports:
    - La ligne de transmission des données TX
    - La ligne de réception des données RX.
    - L'horloge de synchronisation

a) Configuration de la liaison série UART

La liaison série est configurée par la fonction : Serial.begin() qui permet de fixer la vitesse de transmission (en bits par seconde bps) des informations.

Les vitesses typiquement utilisées : 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.

Un second argument optionel permet de configurer la taille des données, la parité et le nombre de stops bits. **Par défaut, le format de la communication est : 8 data bits, aucun bit de parité, 1 stop bit.**

Deux diodes présentes sur la carte Arduino sont associées à la liaison série :

- TX : s'allume lors d'une transmission
- RX : s'allume lors d'une réception

Syntax

Serial.begin(speed)
Serial.begin(speed, config)
Arduino Mega only:
Serial1.begin(speed)
Serial2.begin(speed)
Serial3.begin(speed)
Serial1.begin(speed, config)
Serial2.begin(speed, config)
Serial3.begin(speed, config)

Parameters

speed: in bits per second (baud) - *long*
config: sets data, parity, and stop bits. Valid values are :

- Aucune parité , 1 stop bit : SERIAL_5N1 ; SERIAL_6N1 ; SERIAL_7N1; SERIAL_8N1 (the default)
- Aucune parité , 2 stop bit : SERIAL_5N2 ; SERIAL_6N2 ; SERIAL_7N2 ; SERIAL_8N2
- Parité paire, 1 stop bit: SERIAL_5E1 ; SERIAL_6E1 ; SERIAL_7E1 ; SERIAL_8E1
- Parité paire, 2 stop bit: SERIAL_5E2 ; SERIAL_6E2 ; SERIAL_7E2 ; SERIAL_8E2
- Parité impaire, 1 stop bit: SERIAL_5O1 ; SERIAL_6O1 ; SERIAL_7O1 ; SERIAL_8O1
- Parité impaire, 2 stop bit: SERIAL_5O2 ; SERIAL_6O2 ; SERIAL_7O2; SERIAL_8O2

Returns *nothing*

b) Emission d'une information sur la liaison série :

Les informations sont émises par l'émetteur sur la liaison série par la fonction: Serial.write()

Syntax
Serial.write(val)
Serial.write(str)
Serial.write(buf, len)
Arduino Mega also supports: Serial1, Serial2, Serial3 (in place of Serial)
Parameters
val: a value to send as a single byte
str: a string to send as a series of bytes
buf: an array to send as a series of bytes
len: the length of the buffer
Returns : byte
write() will return the number of bytes written, though reading that number is optional

Example

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

void loop(){
  Serial.write(45); // send a byte with the value 45

   int bytesSent = Serial.write("hello"); //send the string "hello" and return the length of the string.
}
```

Ces informations sont visibles dans la fenêtre « Moniteur Série » de l'environnement de développement préalablement configurée avec une vitesse de transmission de 9600bps.

c) Emission d'une information préalablement convertie en une chaine de caractères sur la liaison série :

L'information spécifiée est convertie en une chaine de caractère puis émise sur la liaison série par la fonction: Serial.print().

Ces informations sont visibles dans la fenêtre « Moniteur Série » de l'environnement de développement préalablement configurée avec une vitesse de transmission identique à la configuration du microcontroleur.

Syntax

Serial.print(val)
Serial.print(val, format)
Le second paramètre *format* spécifie :

- la base numérique à utiliser lors de la conversion d'une grandeur entière en chaine de caractères : BIN(binary, or base 2), OCT(octal, or base 8), DEC(decimal, or base 10), HEX(hexadecimal, or base 16).
- le nombre de décimales à conserver pour les grandeurs décimales.

Serial.print(78, BIN) gives "1001110"
Serial.print(78, OCT) gives "116"
Serial.print(78, DEC) gives "78"
Serial.print(78, HEX) gives "4E"
Serial.print(1.23456, 0) gives "1"

Serial.print(1.23456, 2) gives "1.23"
Parameters : val: the value to print. Allowed data types: any data type.
Returns : print() returns the number of bytes written, though reading that number is optional. Data type: size_t.

d) Example Code

```
/*
  Uses a for loop to print numbers in various formats.
*/
void setup() {
  Serial.begin(9600); // open the serial port at 9600 bps:
}

void loop() {
for (int x = 0; x < 64; x++) { // only part of the ASCII chart, change to suit
    // print it out in many formats:
    Serial.print(x);       // print as an ASCII-encoded decimal - same as "DEC"
    Serial.print("\t\t");  // prints two tabs to accomodate the label lenght

    Serial.print(x, DEC);  // print as an ASCII-encoded decimal
    Serial.print("\t");    // prints a tab

    Serial.print(x, HEX);  // print as an ASCII-encoded hexadecimal
    Serial.print("\t");    // prints a tab

    Serial.println(x, BIN);  // print as an ASCII-encoded binary
    // then adds the carriage return with "println"
    delay(200);            // delay 200 milliseconds
  }
  Serial.println();        // prints another carriage return
}
```

e) Réception d'une information sur la liaison série

Le données reçues sur la liaison série sont lues par la fonction : Serial.read();

Syntax
Serial.read()
Returns
The first byte of incoming serial data available (or -1 if no data is available). Data type: int.

f) Example Code

```
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);
  }
}
```

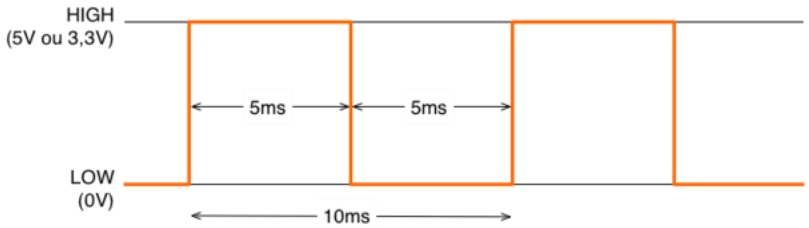
Chapitre 6 Modulation de largeurs d'impulsions

La commande de nombreux actionneurs nécessite une intensité de tension variable dans la gamme 0-5V, par exemple pour faire varier la vitesse de rotation d'un moteur ou encore pour faire varier l'intensité lumineuse d'une led.

Le microcontrôleur a la possibilité de générer des signaux modulés en largeur d'impulsion (Pulse Width Modulation), ou MLI (Modulation de largeur d'impulsion). Le *rapport cyclique* désigne le ratio entre la durée de l'état haut et la période du signal. La modulation en largeur d'impulsion agit donc sur ce rapport cyclique. Un rapport cyclique à 0% correspond à un signal constamment à l'état LOW, et à 100% correspond à un signal constamment à l'état HIGH.

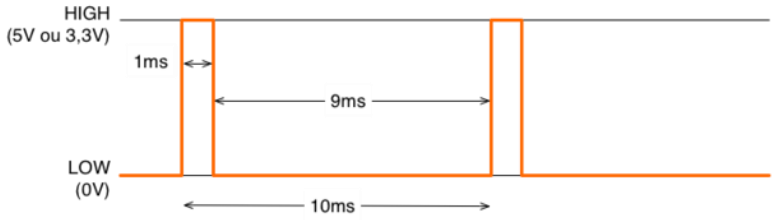
Plusieurs exemples sont donnés sur le site https://www.locoduino.org/spip.php?article47 .

Prenons par exemple un signal de période de 10ms, soit de fréquence de 100Hz. Si la led est allumée pendant 5ms et éteinte pendant 5ms, comme sur la figure ci-dessous, l'impression sera une luminosité de 50% de la luminosité maximale.



PMW à 50% : La fréquence est de 100Hz, le rapport cyclique de 50%

Si la led est allumée pendant 1ms et éteinte pendant 9ms, l'impression sera une luminosité de 10% :.



PWM à 10% : La fréquence est de 100Hz et le rapport cyclique de 10%.

a) Configuration PWM

o Fréquence du signal

La fréquence du signal modulé en largeur d'impulsion est imposée pour les développements Arduino. Pour les cartes les plus utilisées (uno, mega), la fréquence du signal de 490Hz ou 980Hz.

BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev.2	3, 5, 6, 9, 10	976 Hz
MKR boards *	0 - 8, 10, A3 (18), A4 (19)	732 Hz
MKR1000 WiFi *	0 - 8, 10, 11, A3 (18), A4 (19)	732 Hz
Zero *	3 - 13, A0 (14), A1 (15)	732 Hz
Due **	2-13	1000 Hz
101	3, 5, 6, 9	pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz

a

- Rapport cyclique

Le fonction analogWrite() permet de fixer le rapport cyclique.

Syntax

analogWrite(pin, value)

Parameters

pin: the Arduino pin to write to. Allowed data types: int.

value: the duty cycle: between 0 (always off) and 255 (always on). Allowed data types: int.

Returns Nothing

b) Example Code

Un potentiomètre permet de fixer la valeur du rapport cyclique du signal délivrée sur une broche connectée à une led.

```
int ledPin = 9;      // LED connected to digital pin 9
int analogPin = 3;   // potentiometer connected to analog pin 3
int val = 0;         // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the pin as output
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
  analogWrite(ledPin, val / 4); // analogRead values go from 0 to 1023, analogWrite values from 0
to 255
}
```

Chapitre 7 Des projets à réaliser

a) Commande des déplacements d'un robot et de son système lumineux

- Il s'agit de proposer un projet permettant la réalisation d'un robot piloté par une application android communicant en Bluetooth ou une télécommande infra-rouge. L'application permettra le contrôle des mouvements du robot ainsi que de ses signaux lumineux.
- Le robot peut être équipé d'un détecteur d'obstacle qui peut renvoyer des informations au système de pilotage.
- De nombreux éléments de présentation de capteurs et actionneurs sont à disposition (http://www.elegoo.com/download/) et libres pour le téléchargement : **Elegoo Mega 2560 The Most Complete Starter Kit, Elegoo Smart Robot Car Kit V3.0**

b) Domotique : Contrôle de l'intensité de l'éclairage

- En mode automatique : l'intensité lumineuse est fixée en fonction de la luminosité ambiante
- En mode manuel : l'intensité lumineuse est fixée via un variateur de type potentiomètre.
- Le mode automatique ou manuel est choisi par exemple par un interrupteur. Le système d'éclairage est une matrice de leds.
- De nombreux éléments de présentation de capteurs et actionneurs sont à disposition (http://www.elegoo.com/download/) et libres pour le téléchargement : **Elegoo Mega 2560 The Most Complete Starter Kit, Elegoo Smart Robot Car Kit V3.0**

c) Domotique : Contrôle de la vitesse de rotation d'un ventilateur

- En mode automatique : la vitesse de rotation du moteur du ventilateur est fixée en fonction de la température ambiante
- En mode manuel : la vitesse de rotation du moteur du ventilateur est fixée par un potentiomètre
- Le mode automatique ou manuel est choisi par exemple par un interrupteur.
- De nombreux éléments de présentation de capteurs et actionneurs sont à disposition (http://www.elegoo.com/download/) et libres pour le téléchargement : **Elegoo Mega 2560 The Most Complete Starter Kit, Elegoo Smart Robot Car Kit V3.0**

d) Alarme : Système de contrôle des accès

- Les accès d'une pièce sont sécurisés. Le système de sécurité est activé par un code saisi sur un clavier matriciel pu par une carte rfid Les intrusions peuvent détectées par plusieurs capteurs de type capteur de mouvement, capteur de distance, capteur de son. Les alertes sont données par des dispositifs lumineux et sonores.
- De nombreux éléments de présentation de capteurs et actionneurs sont à disposition (http://www.elegoo.com/download/) et libres pour le téléchargement : **Elegoo Mega 2560 The Most Complete Starter Kit, Elegoo Smart Robot Car Kit V3.0**

Chapitre 8 Annexe : Liste des fonctions de référence

FUNCTIONS

Liste des fonctions de reference Arduino: https://www.arduino.cc/reference/en/#page-title

Digital I/O

digitalRead()
digitalWrite()
pinMode()

Analog I/O

analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family

analogReadResolution()
analogWriteResolution()

Advanced I/O

noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time

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

Math

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

Trigonometry

cos()
sin()
tan()

Characters

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

Random Numbers

random()
randomSeed()

Bits and Bytes

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

External Interrupts

attachInterrupt()
detachInterrupt()

Interrupts

interrupts()
noInterrupts()

Communication

Serial
Stream

USB

Keyboard
Mouse