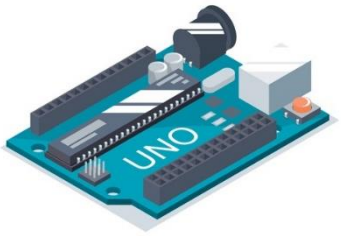
Contrôle de la carte Arduino

I. Introduction

La carte Arduino, tout comme d'autres cartes de programmation, est conçue pour contrôler des périphériques afin d'exécuter des tâches spécifiques. Équipée d'un microcontrôleur programmable, la carte Arduino utilise principalement le langage de programmation C++. Dans cette partie, nous nous concentrerons sur les différentes fonctions utilisées pour contrôler la carte Arduino Uno, offrant ainsi une vue d'ensemble des capacités de programmation offertes par cette plateforme.



II. Contrôle des entrées/sorties numériques

Ces trois fonctions sont essentielles pour contrôler les entrées et sorties numériques sur les cartes Arduino. Pin 0 à pin 13

1. Fonction 1 : *pinMode ()*

Cette fonction est utilisée pour configurer le mode d'une broche (pin) comme entrée ou sortie. Elle prend deux arguments : le numéro de la broche et le mode souhaité (INPUT pour entrée, OUTPUT pour sortie).

Exemple :

```
pinMode(7, OUTPUT) ; // Configure la broche 7 en tant que sortie
pinMode(2, INPUT) ;  // Configure la broche 2 en tant qu'entrée
```

2. Fonction 2 : *digitalRead()*

Cette fonction est utilisée pour lire la valeur numérique (HIGH ou LOW) d'une broche configurée en entrée. Elle prend un argument : le numéro de la broche à lire.

Exemple :

```
int value = digitalRead(2) ; // Lit la valeur de la broche 2 et la stocke dans la variable 'value'
```

3. Fonction 2 : *digitalWrite()*

Cette fonction est utilisée pour écrire une valeur numérique (HIGH ou LOW) sur une broche configurée en sortie. Elle prend deux arguments : le numéro de la broche et la valeur à écrire (HIGH ou LOW).

Exemple :

```
digitalWrite(7, HIGH) ; // Écrit une valeur HIGH sur la broche 7
digitalWrite(13, LOW) ; // Écrit une valeur LOW sur la broche 13
```

III. Contrôle des entrées/sorties analogiques

Ces fonctions sont essentielles pour contrôler les entrées et sorties analogiques sur les cartes Arduino. Il sont largement utilisées pour lire des valeurs de capteurs analogiques, générer des signaux PWM pour contrôler des moteurs et des LED, et ajuster les paramètres de lecture et d'écriture en fonction de vos besoins.

1. Fonction 1 : *analogRead()*

Cette fonction permet de lire la valeur d'une broche analogique sur Arduino. Cette fonction convertit les tensions entre 0 V et la tension d'alimentation en valeurs numériques entre 0 et 1023, avec une résolution de 4.9 mV. La vitesse de conversion (CAN d'Arduino) est environ 100 microsecondes.

Les broches	CAN	Temps de conversion	Tension plein échelle	Résolution q
A0 à A5	10 bits	100 µs	5 V	4.9 mV

Exemple :

```
int value = analogRead(A0); // Lit la valeur analogique de la broche A0 et la stocke dans la variable 'value'
```

2. Fonction 2 : *analogReference()*

Cette fonction est utilisée pour définir la référence de tension utilisée (tension plein échelle PE) pour les lectures analogiques. Elle prend un argument : la référence de tension souhaitée (par exemple, DEFAULT, INTERNAL ou EXTERNAL).

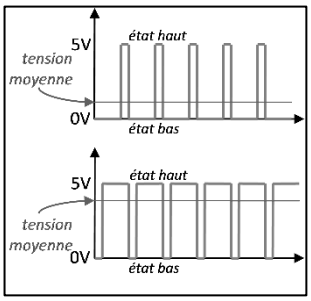
- **DEFAULT** : utilise la tension d'alimentation de la carte Arduino (habituellement 5 volts.
- **INTERNAL** : utilise une tension de référence interne (habituellement 1.1 volts) fournie par le microcontrôleur de la carte Arduino Uno lorsque la précision est demandée.
- **EXTERNAL** : Cette option permet d'utiliser une référence de tension externe fournie par l'utilisateur via la broche AREF (Analog REFerence).

Exemple :

```
analogReference(DEFAULT) ; // Définit la référence de tension par défaut pour les lectures analogiques (5V)
```

3. Fonction 3 : *analogWrite()*

Cette fonction est utilisée pour écrire une valeur analogique (Générer un signal PWM - modulation de largeur d'impulsion MLI) sur une broche. Elle peut être utilisée pour allumer une LED à différentes intensités ou pour entraîner un moteur à différentes vitesses.



Carte Arduino	Broches	Fréquence de signal MLI
UNO	3, 5, 6, 9, 10, 11	○ Broches 3, 9, 10, 11 : **f = 490 Hz** ○ Broche 5, 6 : **f = 980 Hz**

Elle prend deux arguments : le numéro de la broche et la valeur à écrire (de 0 à 255). La valeur 0 correspond à un rapport cyclique de 0%, et 255 correspond à 100%.

Exemple :

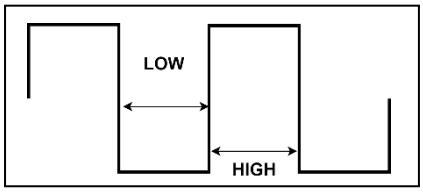
```
analogWrite(9, 127) ; // Génère un signal MLI de rapport cyclique 50% (127) sur la broche 9
```

IV. Entrée/Sortie avancées

Ces fonctions avancées de la carte Arduino peuvent être très utiles pour des applications variées, allant de la mesure de durées d'impulsions à la génération de tonalités sonores ou à la communication avec des périphériques externes via des registres à décalage.

1. Fonction 1 : *pulseIn()*

- **Utilisation :** Mesure la durée d'une impulsion sur un pin spécifié.
- **Syntaxe :** `pulseIn(pin, value)`
- **Remarque :** La fonction attend le signal spécifié (HIGH ou LOW) sur le pin et retourne la durée de l'impulsion en microsecondes.



Exemple :

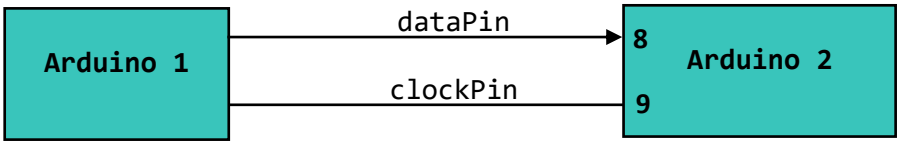
```
int Tb = pulseIn(2, LOW) ; // mesure la dure de l'état bas du signal présenté à la broche 2 et le stocke dans Tb
```

2. **Fonction 2 : *pulseInLong()***
   - Utilisation : Similaire à pulseIn(), mais prend en charge des durées d'impulsion plus longues (jusqu'à 3 minutes).
   - Syntaxe : `pulseInLong(pin, value)`
   - Remarque : Idéale pour les impulsions de longue durée où pulseIn() pourrait dépasser sa limite.

Exemple :

```
int Th = pulseInLong(5, LOW) ; // mesure la dure de l'état bas du signal présenté à la broche 5
```

3. **Fonction 3 : *shiftIn()***
   - Utilisation : Lecture des données séquentielles à partir d'un registre à décalage (shift register).
   - Syntaxe : `shiftIn(dataPin, clockPin, bitOrder)`
   - Remarque : Cette fonction permet de lire les données provenant d'un registre à décalage connecté aux broches spécifiées.



Exemple :

```
int dataPin = 8;          // Pin pour les données (input)
int clockPin = 9;         // Pin pour l'horloge (output)
int bitOrder = MSBFIRST; // Ordre de bits pour la lecture (Most Significant Bit First)
byte incomingData;        // Variable pour stocker les données reçues

   void setup() {
    pinMode(dataPin, INPUT);    // Broche de données en entrée
    pinMode(clockPin, OUTPUT);  // Broche d'horloge en sortie
   }

   void loop() {
    // Lecture des données à partir du registre à décalage
    incomingData = shiftIn(dataPin, clockPin, bitOrder);
   }
```

4. **Fonction 4 : *shiftOut()***
   - Utilisation : Envoie des données séquentielles à un registre à décalage.
   - Syntaxe : `shiftOut(dataPin, clockPin, bitOrder, value)`
   - Remarque : Permet d'envoyer des données binaires à un registre à décalage, contrôlant ainsi des périphériques externes tels que des afficheurs LED ou des afficheurs à 7 segments.

Exemple :

```
int dataPin = 8;
int clockPin = 9;
int bitOrder = MSBFIRST;
byte dataToSend = B10101010;

   void setup() {
     pinMode(dataPin, OUTPUT);
```

```
  pinMode(clockPin, OUTPUT);
}
void loop() {
  shiftOut(dataPin, clockPin, bitOrder, dataToSend);
  delay(1000);
}
```