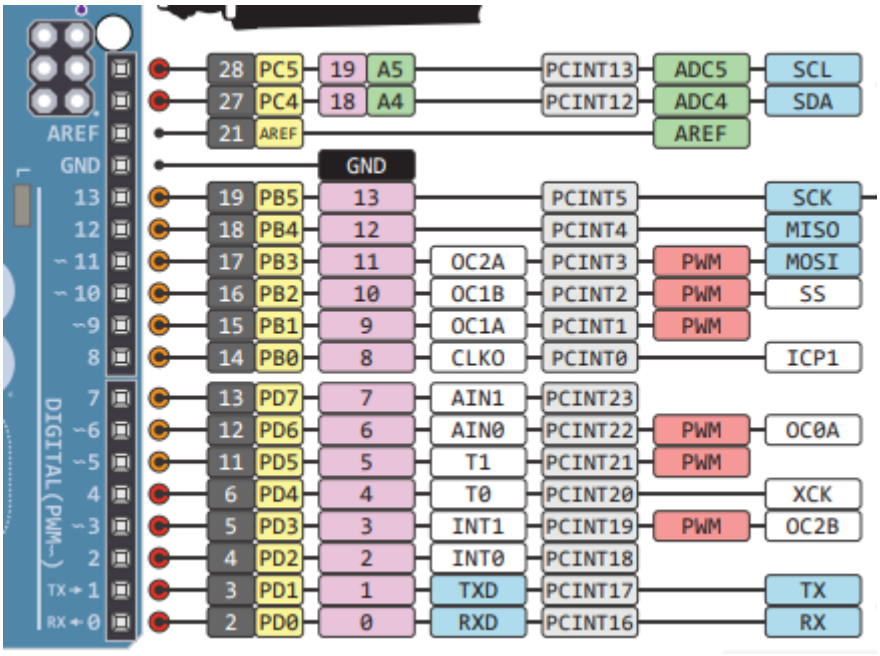
Les entrées, sorties numériques

[Mise à jour le 26/4/2021]

2. Les entrées/sorties numériques de la carte Arduino Uno

Les entrées, sorties numériques sont numérotées entre **0** et **13** sur la carte. Certaines peuvent assurer plusieurs fonctions. Le symbole ~ identifie les broches capables de délivrer un signal PWM.



> Le **schéma** de la carte est téléchargeable ici.

3. Programmation

3.1 Configurer une broche en entrée ou en sortie

Source : pinMode() sur le site arduino.cc

> **Configurer** une broche signifie qu'on précise qu'elle doit se comporter soit comme une entrée d'information, soit comme une sortie d'information.

Exemple

*.cpp

```
// Sauf exception la configuration d'une broche se fait dans la
fonction setup()
void setup() {
  pinMode(13,OUTPUT);     // la broche 13 se comporte comme une sortie
  pinMode(12,INPUT);      // la broche 12 se comporte comme une entrée,
                          // optionnel si la broche n'a pas été
préalablement
                          // configurée en sortie, car c'est le mode
par défaut
}
```

3.2 Lire l'état logique présent sur une broche

Source : digitalRead() sur le site arduino.cc

> La valeur lue sur une broche configurée en entrée doit être sauvegardée dans une **variable**.

Exemple

*.cpp

```
// L'opération de lecture se fait dans la fonction loop().
void loop() {
  int valeur = digitalRead(12);   // l'état logique présent sur la
broche 12 est sauvegardé
                                  // dans la variable valeur
}
```

3.3 Ecrire un état logique sur une broche

Source : digitalWrite() sur le site arduino.cc

> Un état logique **haut(HIGH)** ou **bas(LOW)** peut être écrit sur une broche préalablement configurée en sortie.

Exemple

*.cpp

```
void setup() {
  pinMode(13, OUTPUT);    // la broche 13 se comportera comme une sortie
}

// L'opération d'écriture se fait dans la fonction loop().
void loop() {
   digitalWrite(13, HIGH); // La broche 13 présente un état logique haut (par exemple 5V)
}
```

3.5 Temporisations

- **Sources** : delay, delayMicroseconds, micros et millis sur le site arduino.

3.5.1 Temporisation bloquante

- de **n millisecondes** avec la fonction : *void **delay**(int millisecondes)*

*.cpp

```
// Le code met le programme en pause pendant une seconde avant de faire basculer l'état logique de la broche.

int ledPin = 13;                // LED connected to digital pin 13

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

void loop() {
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000);                // waits for a second
  digitalWrite(ledPin, LOW);  // sets the LED off
  delay(1000);                // waits for a second
}
```

- de **n microsecondes** avec la fonction : *void **delayMicroseconds**(int microsecondes)* (Erreur = +/-3µs)

*.cpp

```
// Le code configure la broche numéro 8 en sortie.
// Il envoie un train d'impulsions d'une période d'environ 100
microsecondes.
// L'approximation est due à l'exécution des autres instructions.

int outPin = 8;                 // digital pin 8

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

void loop() {
  digitalWrite(outPin, HIGH); // sets the pin on
  delayMicroseconds(50);        // pauses for 50 microseconds
  digitalWrite(outPin, LOW);  // sets the pin off
  delayMicroseconds(50);        // pauses for 50 microseconds
}
```

3.5.2 Temporisation non bloquante

- **Temps écoulé**, en **millisecondes**, depuis le démarrage du programme avec la fonction : *unsigned long **millis**()*

> **millis()** renvoie le nombre de millisecondes écoulées depuis que la carte Arduino a commencé à exécuter le programme. Ce nombre va déborder (revenir à zéro), après environ 50 jours.

- *Structure d'une temporisation non bloquante*

*.cpp

```
// Répétition d'un bloc de code tous les intervalles de temps sans
bloquer le programme
unsigned long previousMillis = 0;  // Mémorise le temps écoulé depuis
la dernière mesure
const long interval = 1000;        // Durée non bloquante souhaitée

void setup() {
  // Le contenu de setup dépend de l'application
}

void loop() {
```

```
  // Mesure du temps écoulé depuis que le programme s'exécute
  unsigned long currentMillis = millis();

  // Si l'attente souhaitée est atteinte on exécute le bloc de code
  if (currentMillis - previousMillis >= interval) {
    // la mesure de temps actuelle devient la dernière mesure
    previousMillis = currentMillis;
    //
    // Bloc de code à exécuter tous les intervalles de temps
    //
  }
  //
  // sinon d'autres actions sont possibles
  //
}
```

- **Temps écoulé**, en **microsecondes**, depuis le démarrage du programme avec la fonction : *unsigned long **micros**()*

> **micros()** renvoie le nombre de microsecondes depuis que la carte Arduino a commencé à exécuter le programme. Ce nombre va déborder (revenir à zéro), après environ 70 minutes. Sur les cartes Arduino 16 MHz (par exemple Duemilanove et Nano), cette fonction a une résolution de quatre microsecondes (c'est-à-dire que la valeur renvoyée est toujours un multiple de quatre).

*.cpp

```
unsigned long time;

void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.print("Time: ");
  time = micros();

  Serial.println(time);
  delay(1000);
}
```

3.6 Interruption

3.6.1 Attacher une interruption

- **Source** : attachinterrupt sur le site Arduino.
- **Syntaxe**

*.cpp

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)
```

- **ISR** : gestionnaire d'interruption à appeler lorsque l'interruption se produit
- **pin** : broche de la carte (concernée par l'interruption)
- **mode** : définit le moment de déclenchement de l'interruption
  - **Paramètres**
  - **LOW** déclenche l'interruption chaque fois que la broche est à l'état bas,
  - **CHANGE** déclenche l'interruption chaque fois que la broche change de valeur,
  - **RISING** déclenche l'interruption sur un front montant,
  - **FALLING** déclenche l'interruption sur un front descendant.
- *Exemple*

*.cpp

```
// Le front montant du signal présent sur la broche 3 de la carte Arduino Uno
// déclenche l'exécution du sous-programme blink

const byte ledPin = 11; // 00 du shield Tinkerkit
const byte interruptPin = 3; // 05 du shield Tinkerkit
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

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

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

> Le projet **PlatformIO** pour l'IDE **VSCode** de l'exemple ci-dessus est téléchargeable ici

3.6.2 Détacher une interruption

La fonction *detachInterrupt()* désactive l'interruption donnée.

- **Source** : detachInterrupt sur le site Arduino.
- **Syntaxe**

*.cpp

```
detachInterrupt(digitalPinToInterrupt(pin))
```