Institut d'Optique Graduate School
Interfaçage Numérique

Interfaçage Numérique

Travaux Pratiques

Semestre 6

Arduino et STM32

---

1 séance

Ce sujet est disponible au format électronique sur le site du LEnsE - https ://lense.institutoptique.fr/ dans la rubrique Année / Première Année / Interfaçage Numérique S6 / Bloc Systèmes embarqués / Intro Arduino et STM32.

Séance 1 / Arduino et Nucléo-STM32 (sans maquette ! !)

Objectifs de la séance

Cette première séance est consacrée à la découverte de la **programmation de systèmes embarqués** (ici des cartes **Nucléo** de *STMicroelectronics*, basées sur des microcontroleurs *STM32*) et la prise en main de l'interface de développement **Arduino**.

Etape 0 - 30 min Installer les drivers STM32 et tester un premier programme

Etape 1 - 45 min Piloter des sorties numériques - LED

Etape 2 - 45 min Acquérir des données numériques - Bouton-poussoirs

Etape 3 - 45 min Mettre en œuvre des interruptions sur des événements externes

Etape 4 - 45 min Utiliser des sorties modulées en largeur d'impulsion (PWM) - LEDs

Etape 5 - 60 min Acquérir des données analogiques - Potentiomètre

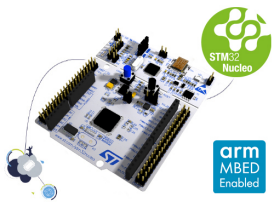
IDE Arduino

Arduino est une plateforme open-source utilisée pour créer des projets électroniques. Elle est composée de deux éléments principaux : **une carte matérielle** (contenant un microcontrôleur) et **un environnement de développement** (IDE Arduino) qui permet de programmer la carte.

Nous nous intéresserons ici qu'à l'environnement de développement qui, après installation d'une extension, permet de programmer d'autres cartes à microcontrôleurs.

Carte Nucleo-STM32

Les cartes Nucleo sont des **plateformes de développement** basées sur les **microcontrôleurs STM32** de *STMicroelectronics*. Elles sont conçues pour faciliter le prototypage et le développement de projets embarqués, similaires aux cartes Arduino, mais elles sont souvent utilisées pour des applications plus complexes et performantes.



Elles sont équipées d'un débogueur ST-LINK intégré, ce qui permet de programmer et de déboguer le microcontrôleur directement sans matériel additionnel.

Le brochage de la carte Nucleo L476RG est fournie en annexe à ce document : Brochage Nucléo L476RG

Etape 0a / Installation des cartes STM32

L'interface de développement **Arduino**, ainsi que les bibliothèques associées, est populaire pour les projets *Do It Yourself*, l'éducation et le prototypage rapide en raison de sa simplicité et de son accessibilité. Cependant, elle est initialement prévue pour programmer des cartes de type **Arduino**.

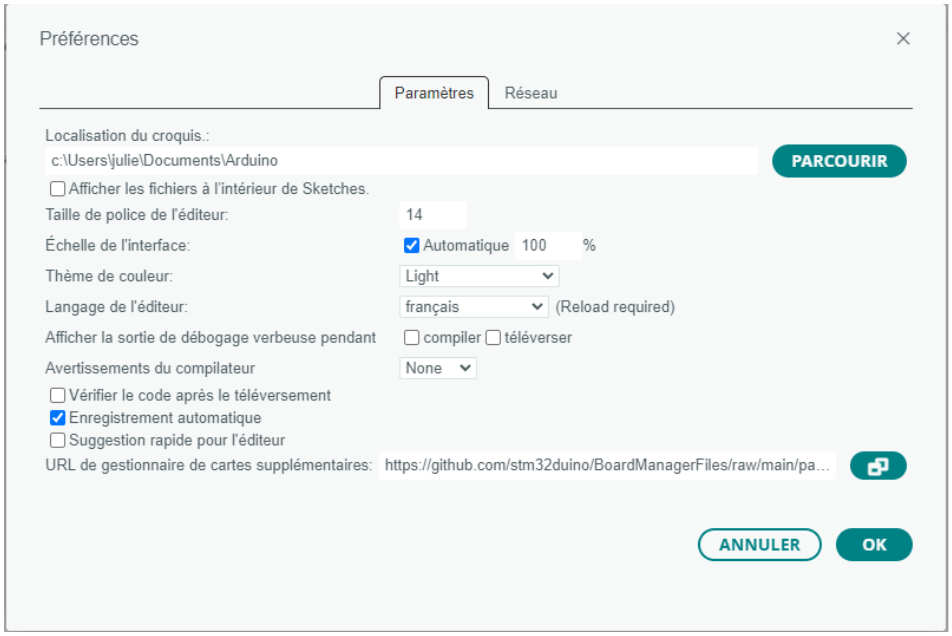
Pour pouvoir bénéficier de l'environnement **Arduino** pour **d'autres cartes de prototypage**, il est indispensable d'installer les extensions associées à ces autres cartes.

Attention ! La version 2 de l'IDE Arduino est fortement conseillée pour bénéficier des dernières évolutions du langage et de l'interface, ainsi que pour garantir une pleine compatibilité avec les cartes Nucléo.

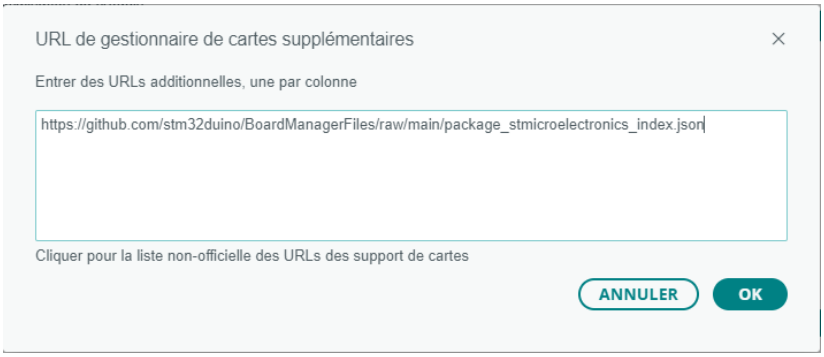
Support des cartes STM32

Avant de pouvoir utiliser l'environnement Arduino pour programmer des cartes intégrant des microcontrôleurs de type STM32, il faut installer le **support pour ces microcontrôleurs**.

Dans le menu **Fichiers / Préférences**, sélectionner le volet **Paramètres**.



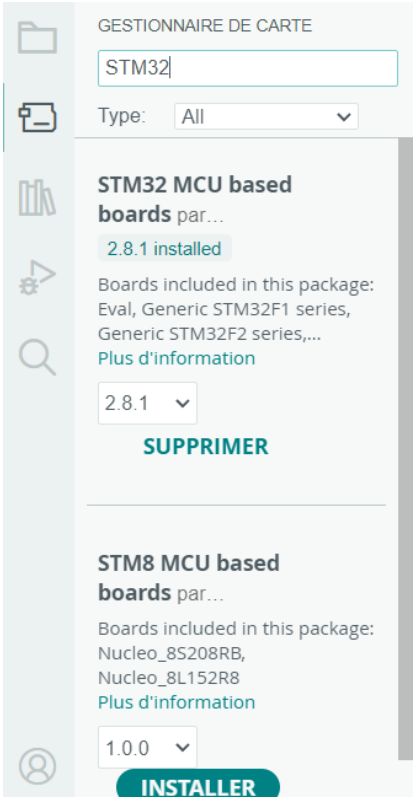
Dans la fenêtre URL de gestionnaire de cartes supplémentaires, ajouter l'adresse suivante :
https://GitHub.com/stm32duino/BoardManagerFiles/raw/main/package_stmicroelectronics_index.json



Extension STM32 MCU based boards

Il faut ensuite télécharger les **bibliothèques** liées aux cartes intégrant des **microcontrôleurs STM32** de *STMicroelectronics*.

Aller dans le menu **Outils** / **Carte** / **Gestionnaire de carte**.



Dans la partie droite de l'interface Arduino, un volet **Gestionnaire de carte** s'ouvre. Dans la zone de recherche, taper STM32.

Dans la liste, installer alors l'extension : **STM32 based boards par STMicroelectronics**.

*Vous trouverez également des ressources concernant les **microcontrôleurs** et les **systèmes embarqués** à l'adresse suivante :*

https://iogs-lense-training.github.io/nucleo-basics/contents/general.html

Etape 0b / Tester un premier programme

Afin de vérifier que toute la chaîne de prototypage est opérationnelle, nous allons nous intéresser à un **programme de base** permettant de faire **clignoter une LED** présente par défaut sur la carte Nucléo (c'est également vrai sur les cartes Arduino).

Sélectionner **FICHIER** / **EXEMPLES** / **01.BASICS** / **BLINK** dans la barre de menu.

Le programme ressemble à celui-ci :

```
1  void setup() {
2    // initialize digital pin LED_BUILTIN as an output.
3    pinMode(LED_BUILTIN, OUTPUT);
4  }
5
6  void loop() {
7    digitalWrite(LED_BUILTIN, HIGH);
8     // turn the LED on (HIGH is the voltage level)
9    delay(1000);
10    // wait for a second
11   digitalWrite(LED_BUILTIN, LOW);
12    // turn the LED off by making the voltage LOW
13   delay(1000);
14    // wait for a second
15 }
```

Sur la carte Nucléo la sortie LED_BUILTIN correspond à la LED nommée LD2.

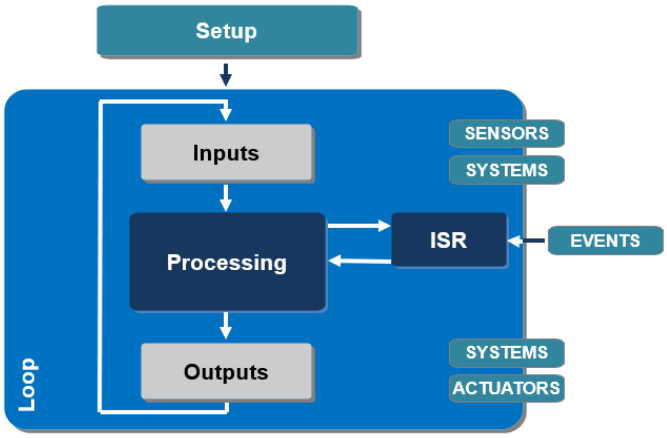
Le langage utilisé par l'environnement Arduino est un langage proche du C++. Le programme ainsi écrit doit nécessairement **être compilé** avant de pouvoir **être téléversé** sur la carte où il sera ensuite **exécuté**.

Structure du code

Un programme Arduino (comme tout autre programme embarqué) est constitué de **deux étapes principales** :

— une **initialisation** (fonction *setup()* pour Arduino) : exécutée une fois à la mise sous tension de la carte ou lors de l'appui sur le bouton Reset.

— une **boucle infinie** (fonction *loop()* pour Arduino) : exécutée de manière infinie. Cette boucle a pour principale mission, sur un système embarqué, de récupérer les valeurs des entrées, de calculer les valeurs des sorties et de mettre à jour les sorties.



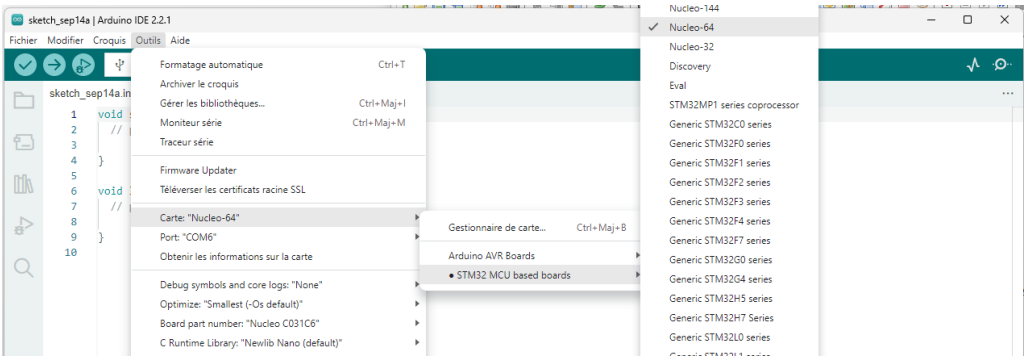
D'autres étapes sont possibles lorsqu'on autorise le fonctionnement par interruption (voir dans la suite de ce document).

Choix d'une carte

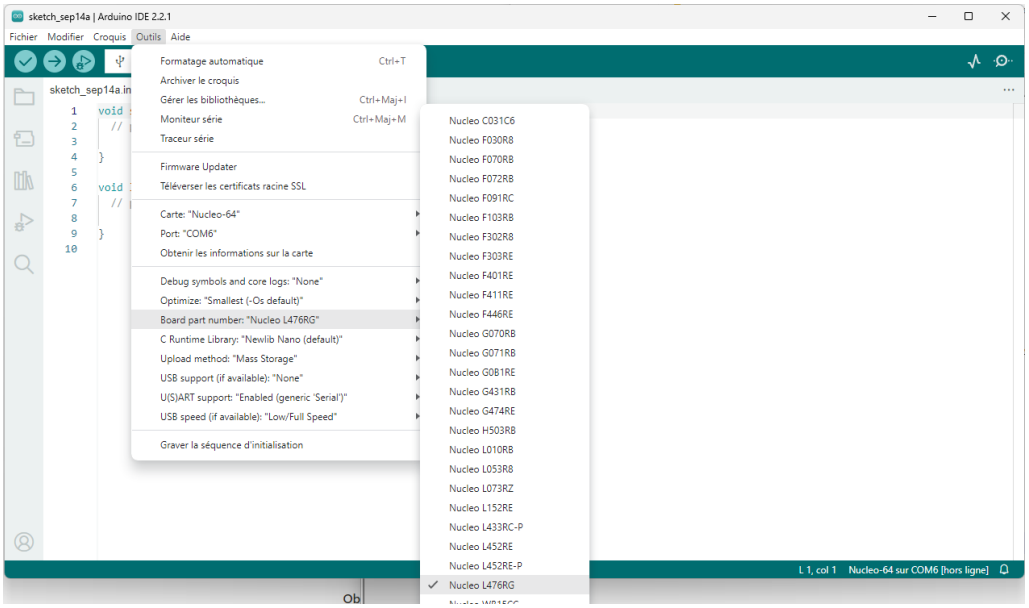
La compilation d'un tel programme se fait pour une cible particulière. Avant de pouvoir compilé, il est donc nécessaire de préciser sur quel microcontrôleur (ou quelle carte de prototypage) ce code sera exécuté.

Pour cela, dans la barre de menu, sélectionner **Outils / Cartes**.

Dans le cas d'une carte Nucléo de type L476RG, sélectionner ensuite **STM32 MCU based boards / Nucleo-64**. Le format pourra changer s'il s'agit d'une autre carte.



Puis, dans le menu **Outils / Board part number**, sélectionner **Nucleo L476RG**.



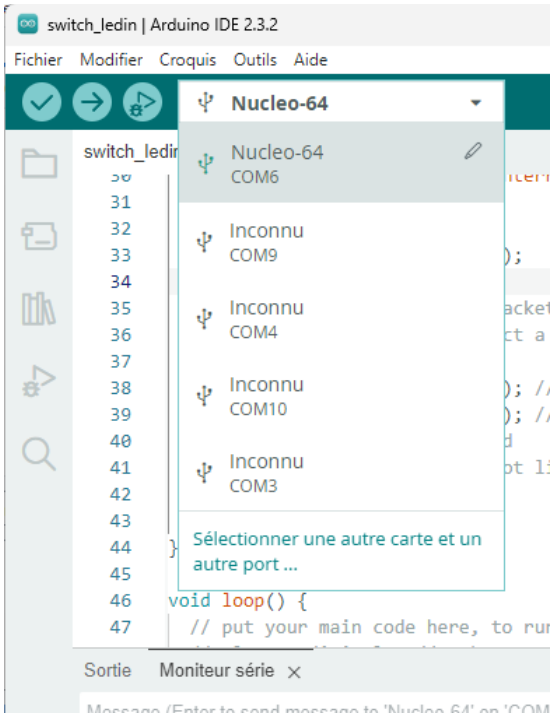
Compilation

Il est maintenant possible de compiler le programme. Pour cela, cliquer sur la première icône de la barre d'action :

Connexion à une carte Nucléo et téléversement

Il faut ensuite connecter la carte en USB.

Dans la barre des actions possibles sous Arduino, sélectionner le port de communication sur lequel est connectée la carte Nucléo.



Pour téléverser ensuite le code dans la carte, cliquer sur la seconde icône de la barre d'actions (en forme de flèche vers la droite).

Etape 1 / Piloter des sorties numériques - LED

Afin de pouvoir interagir avec le monde extérieur, les microcontrôleurs disposent d'un ensemble d'**entrées** et de **sorties**.

Chacune de ces entrées-sorties portent un nom, au format PX_N, où X est le nom du port (A, B...) et N le numéro de la broche.

Exemple de la broche PA_7 (port A, broche 7)

Choix d'une broche

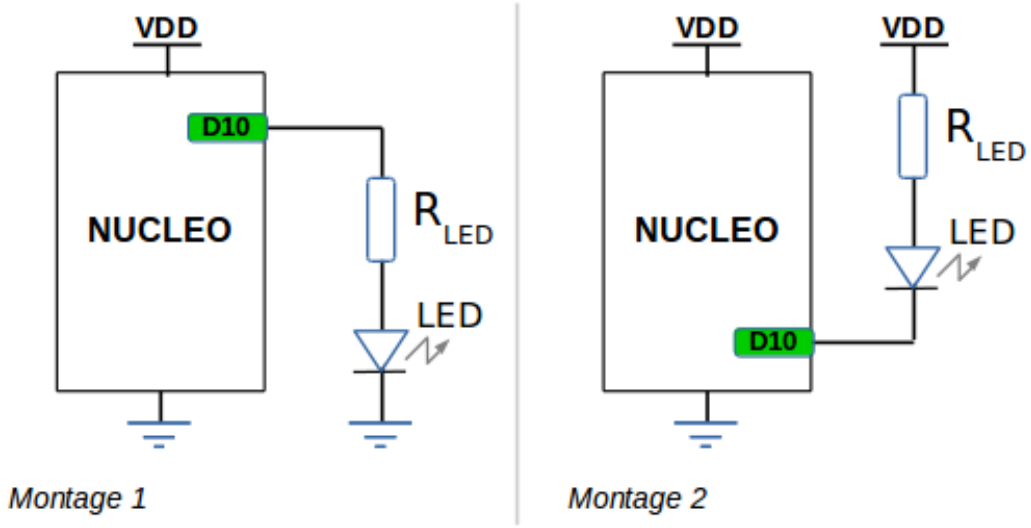
Toutes les broches peuvent être utilisées en **entrée** ou en **sortie numérique**, c'est à dire un type de signal qui ne peut prendre que **deux états** : haut ou bas (aussi appelés 1 ou 0, ou encore *HIGH* et *LOW* en Arduino).

Certaines broches ont également d'autres fonctionnalités : entrées analogiques, sorties modulées PWM, communication série...

Câblage d'une LED

Nous allons voir ici comment connecter une LED à la carte Nucléo sur la broche D10 par exemple de la carte (ou PB6 - port B, broche 6).

Les schémas de câblage possibles sont les suivants :



Exemple de la broche D10 reliée à une LED (ou PB6 - port B, broche 6)

Dans le cas des cartes Nucléo, la tension VDD est égale à $3.3\,\mathrm{V}$.

Il est indispensable d'associer la LED à une **résistance de protection**, permettant de limiter le courant :

$$R_{LED} > \frac{VDD - V_F}{I_{Fmax}}$$

Les valeurs V_F et I_{Fmax} dépendent de la LED utilisée et sont à chercher dans sa documentation technique.

Programme

Pour le programme, il est possible de s'inspirer des exemples fournis dans le logiciel Arduino : FICHIER / EXEMPLES.

Par exemple, pour piloter une sortie numérique, on pourra utiliser le programme 01. BASICS / BLINK.

Paramétrage

Pour **configurer une broche en sortie**, il faut ajouter l'instruction suivante dans la fonction *setup()* (où *LED1* est le nom d'une broche du composant) :

```
1  pinMode(PB6, OUTPUT);
```

Utilisation

Pour **affecter une valeur à une broche en sortie**, il faut utiliser une des deux instructions suivantes (où *LED1* est le nom d'une broche du composant) selon que l'on veut mettre la sortie à l'état bas (*LOW*) ou à l'état haut (*HIGH*) :

```
1  digitalWrite(PB6, LOW);
2  digitalWrite(PB6, HIGH);
```

Utilisation de constantes

Afin de simplifier la lecture du code, il est possible d'**attribuer un nom différent à votre broche** en affectant une **constante entière** à la valeur de la broche utilisée. Cette définition devra se faire en dehors de toute fonction, afin que la constante associée soit **globale**.

```
1  const int led1 = PB6;
```

Il sera alors possible d'utiliser cette constante dans le reste du programme :

```
1  pinMode(led1, OUTPUT);
```

```
1  digitalWrite(led1, LOW);
```

Travail à réaliser

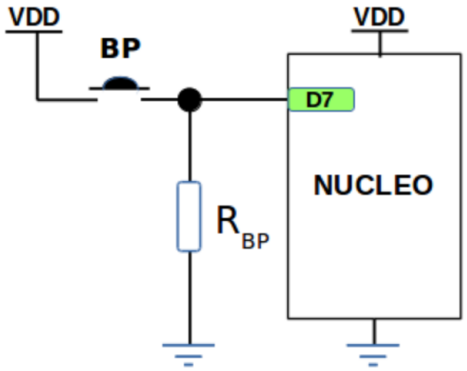
➜ **M** Réaliser un programme qui permet d'allumer deux LEDs rouges cablées sur les sorties D10 et D11. Les LEDs devront s'allumer de manière complémentaire chaque seconde.

Etape 2 / Acquérir des données numériques - Bouton-poussoirs

Câblage d'un bouton-poussoir

Nous allons voir ici comment brancher un bouton-poussoir à la carte Nucléo sur la broche D7 par exemple de la carte (ou PA8 - port A, broche 8).

Le schéma de câblage est le suivant :



Exemple de la broche D7 reliée à un bouton-poussoir (ou PA8 - port A, broche 8)

Avant de câbler la sortie du bouton-poussoir à l'entrée de la carte, vérifier que vous avez bien une tension de $0\,\mathrm{V}$ lorsque le bouton-poussoir n'est pas activé et une tension de $3.3\,\mathrm{V}$.

Programme

Pour récupérer la valeur d'une entrée numérique, on pourra s'inspirer du programme 02. DIGITAL / DIGITALINPUTPULLUP.

Paramétrage

Pour **configurer une broche en entrée**, il faut ajouter l'instruction suivante dans la fonction *setup()* :

```
1  pinMode(PA8, INPUT_PULLUP);
```

Utilisation

Pour **affecter une valeur à une broche en sortie**, il faut utiliser une des deux instructions suivantes (où *LED1* est le nom d'une broche du composant) selon que l'on veut mettre la sortie à l'état bas (*LOW*) ou à l'état haut (*HIGH*) :

```
1  int bpVal = digitalRead(PA8);
```

La variable *bpVal* peut valoir HIGH ou LOW.

Travail à réaliser

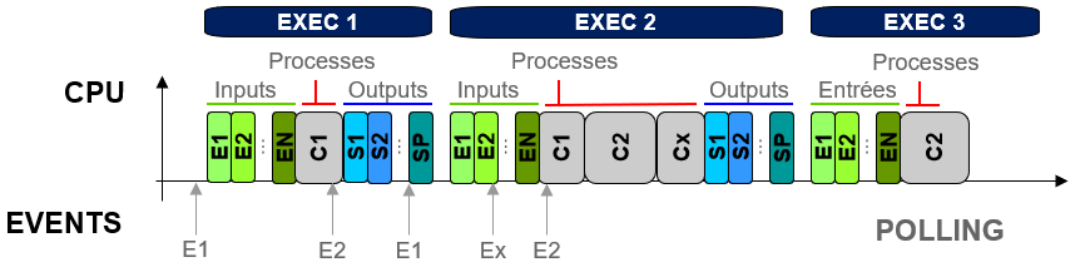
➜ **M** Réaliser un programme qui permet d'allumer chacune des LEDs rouges précédentes à l'aide de deux bouton-poussoirs cablés sur les entrées D6 et D7. Lors d'un premier appui les LEDs s'allumeront, lors d'un second appui, les LEDs s'éteindront.

Etape 3 / Mettre en œuvre des interruptions sur des événements externes

Scrutation

La **scrutation** (ou *polling* en anglais) est une méthode de **vérification régulière de l'état des périphériques ou des capteurs** dans un système embarqué pour détecter si un événement spécifique s'est produit.

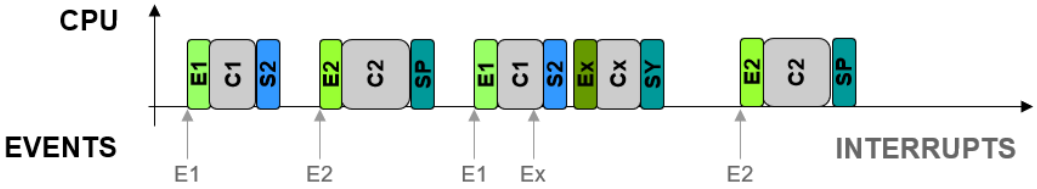
Dans ce contexte, le programme principal exécute une boucle continue (fonction *loop()* sous Arduino) où il interroge périodiquement chaque périphérique pour voir s'il nécessite une action.



La **scrutation** ne permet pas de gérer d'un événement rapidement et monopolise le microcontrôleur pour vérifier constamment l'état des capteurs ou des périphériques.

Interruptions

Une **interruption** est un signal envoyé au microcontrôleur pour lui demander d'**arrêter temporairement l'exécution de son programme principal** et de **s'occuper d'une tâche prioritaire spécifique**. Lorsque l'interruption est terminée, le microcontrôleur reprend son programme là où il s'était arrêté.



Cette méthode permet au système de **réagir rapidement à des événements externes** - changement de signal sur une broche, compteur qui atteint une certaine valeur - sans avoir besoin de vérifier constamment si l'événement a eu lieu, contrairement à la scrutation.

Pour en savoir plus sur les interruptions, vous pouvez consulter le document à l'adresse suivante :
https://iogs-lense-training.github.io/nucleo-basics/contents/polling_interrupts_rtos.html

Les interruptions sont également souvent utilisées dans les applications où un timing précis est nécessaire (par exemple, un compteur de temps).

Paramétrage

Pour **configurer une broche en entrée permettant une interruption**, il faut ajouter l'instruction suivante dans la fonction *setup()* (où *LED1* est le nom d'une broche du composant) :

```
1  pinMode(PA8, INPUT_PULLUP);
2  attachInterrupt(digitalPinToInterrupt(PA8), swInt, FALLING);
```

Il existe 3 modes possibles pour les interruptions sur des signaux externes (entrées numériques) :
— **RISING** - front montant d'un signal
— **FALLING** - front descendant d'un signal
— **CHANGE** - fronts montant et descendant d'un signal

Routine d'interruption

Une **routine d'interruption**, aussi appelée **ISR** (*Interrupt Service Routine*), est la fonction qui s'exécute automatiquement en réponse à une interruption.

C'est une fonction classique, mais qui ne retourne pas de donnée.

```
1 void swInt(void){
2   bool ledState = digitalRead(led1);
3   digitalWrite(led1, !ledState);
4 }
```

Dans cet exemple, à chaque front descendant sur l'entrée *PA8*, la fonction *swInt()* est appelée. Elle lit alors l'état de la sortie *led1* pour ensuite l'inverser.

Travail à réaliser

On se propose de tester le code *interrupt_on_input.ino*. Ce fichier est disponible sur le site du LEnsE dans la rubrique *Année / Première Année / Interfaçage Numérique S6 / Blocs Systèmes embarqués / Exemples Arduino pour STM32*.

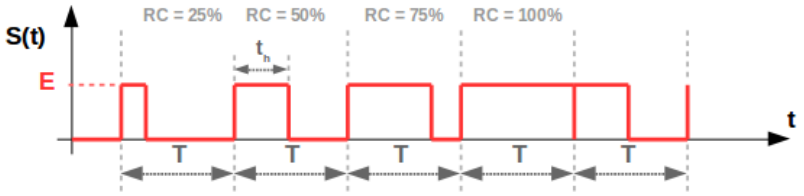
➜ **M** Tester le code fourni.

➜ **Q** A quel moment est appelée la fonction *swInt()* ?

➜ **M** Réaliser un programme qui permet d'allumer à l'aide d'un premier interrupteur, câblé sur l'entrée D7, une LED câblée sur la sortie D10 et de l'éteindre à l'aide d'un second interrupteur câblé sur l'entrée D6. **Votre programme devra utiliser uniquement le principe d'interruption.**

Etape 4 / Utiliser des sorties modulées en largeur d'impulsion (PWM) - LEDs

La **modulation en largeur d'impulsions** (MLI ou *PWM – Pulsed Witdh Modulation* – en anglais) est une méthode permettant de générer un signal rectangulaire de **période** T **fixe** (ou de fréquence $1/T$ fixe) et dont le **rapport cyclique**, i.e. le rapport du temps haut sur la période, noté $RC = \frac{th}{T}$, est variable.



Choix de la broche de sortie

Toutes les broches du microcontroleur ne sont pas utilisables comme sortie modulée en largeur d'impulsion. Cela nécessite l'utilisation de modules internes spécifiques (*timer* et comparateur notamment).

Il faut choisir une broche où le terme *PWM* est mentionné.

Exemple de la broche PA_7 (port A, broche 7), utilisable en PWM

Programme

Pour appliquer un signal modulé en largeur d'impulsion sur une sortie, on pourra s'inspirer du programme 01. BASICS / FADE.

Travail à réaliser

➜ **M** Tester le code *Fade* fourni par Arduino, en adaptant la sortie utilisée à celle déjà câblée sur l'une des LEDs précédentes.

➜ **M** Visualiser à l'aide d'un oscilloscope le signal électrique appliqué sur la LED.

➜ **Q** Expliquer le principe de fonctionnement de ce système. Pourquoi parle-t-on d'une sortie analogique ? (fonction *analogWrite()*

➜ **M** Réaliser un programme qui permet de modifier la luminosité de la LED câblée sur la sortie D7 à l'aide des deux bouton-poussoirs, câblés sur D10 et D11 (un pour augmenter, l'autre pour diminuer la luminosité). **Votre programme devra utiliser uniquement le principe d'interruption.**

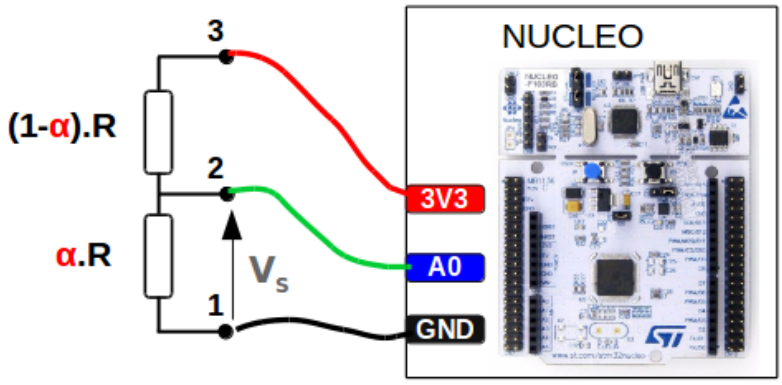
Etape 5 / Acquérir des données analogiques - Potentiomètre

Sur les systèmes numériques, et les microcontrôleurs en particulier, les broches sont naturellement des entrées/sorties numériques.

Or, la plupart des signaux qui nous entourent et que l'on cherche à mesurer (luminosité, température, vitesse...) sont des **grandeurs analogiques**. Pour palier à ce problème, la plupart des fabricants de microcontrôleurs ont intégré des **convertisseurs analogiques-numériques** (*ADC – Analog to Digital Converter*) à leur système, afin d'éviter de passer par des composants externes.

Câblage d'un potentiomètre

Afin de pouvoir générer une tension analogique dont la tension est variable, nous allons utiliser un potentiomètre.



Programme

Pour convertir un signal analogique sur une entrée analogique de la carte Nucléo, on pourra s'inspirer du programme 01. BASICS / ANALOGREADSERIAL.

On pourra utiliser une des entrées notées **Ax** (x étant un entier de 0 à 5).

Travail à réaliser

➜ **M** Câbler un potentiomètre de $10\,\mathrm{k\Omega}$ entre $0\,\mathrm{V}$ et $3.3\,\mathrm{V}$, sur ses broches 1 et 3. Vérifier que la broche 2 varie selon la position du curseur de $0\,\mathrm{V}$ à $3.3\,\mathrm{V}$.

On pourra utiliser la sortie $3.3\,\mathrm{V}$ et la référence GND de la carte Nucléo pour alimenter le potentiomètre.

➜ **M** Tester le code *AnalogReadSerial* fourni par Arduino, en adaptant l'entrée utilisée. L'affichage se fait dans le moniteur série d'Arduino, accessible depuis le menu OUTILS / MONITEUR SÉRIE.

➜ **Q** Expliquer le fonctionnement de ce code.

➜ **M** Réaliser un programme qui permet de modifier la luminosité de la LED câblée sur la sortie D7 à l'aide de la valeur convertie du potentiomètre.