

Systèmes embarqués

Introduction à la programmation microcontrôleurs
et à l'impression 3D.



Pape Abdoulaye BARRO, Ph.D,

Enseignant-chercheur
UFR des Sciences et technologies
Département Informatique

E-LabTP, Laboratoire des TP à Distance, UFR-SET,
Marconi-Lab, Laboratoire de Télécommunications, ACTP, Italie

Email: pape.abdoulaye.barro@gmail.com

- Généralités
- Architecture et familles de microcontrôleur
- Capteurs actionneurs et programmation des microcontrôleurs
 - Etudes de cas (avec Arduino et Raspberry Pi)
- Initiation à l'impression 3D

PLAN

Généralités

- Définitions et illustrations
- Historique
- Opportunités, défis et caractéristiques

Généralités

Définitions et illustrations

Définition 1.1:

Les systèmes embarqués, souvent en temps réel, sont des systèmes électroniques et informatiques autonomes.

- Ils sont constitués d'une partie matérielle et d'une partie logicielle et leurs architectures sont construites autour d'un microcontrôleur.
- Ils disposent de périphériques et de capteurs spécifiques pour leurs applications et se greffent facilement dans les objets que nous utilisons au quotidien [1].

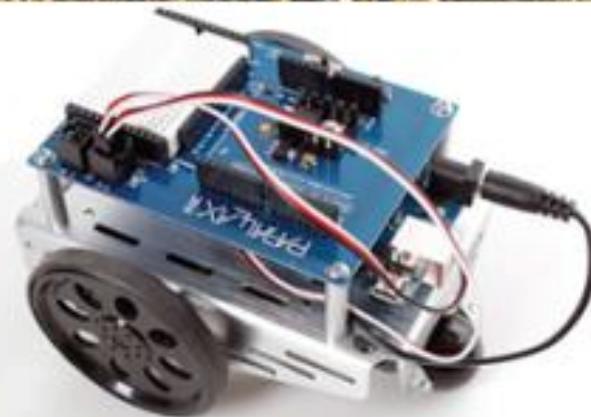
Définition 1.2.:

Un système embarqué est un système informatique, une combinaison d'un processeur, d'une mémoire et de périphériques d'entrée/sortie, qui a une fonction dédiée dans un système mécanique ou électrique plus grand.

- Il fait partie d'un dispositif complet comprenant souvent du matériel électrique ou électronique et des pièces mécaniques [2].

Généralités

Définitions et illustrations



Généralités

Historique

Les tout premiers systèmes embarqués datent du début des années 1960 (avec Sputnik en 1957 et Apollo Guidance Computer en 1967).

- En 1962, l'Air Force a introduit le missile LGM-30 Minuteman;
- En 1969, Marcian Hoff et Federico Faggin ont inventé le premier microprocesseur. Nous entrons donc dans l'air des microprocesseurs personnalisables par logiciel. Ouvrant ainsi la porte aux microcontrôleurs et par conséquent aux solutions embarquées ;
- De nos jours, nous disposons de cartes à base de microcontrôleur/microprocesseur facilement programmables et permettant le développement d'applications interactives complexes.

Généralités

Opportunités, défis et caractéristiques

Les types d'applications possibles dans le contexte des systèmes embarqués sont diverses et variées. Nous les trouvons dans la fabrication des équipements (amélioration des technologies de production) et autres domaines tels que:

- la robotique,
- les applications militaires (armes automatiques, systèmes de radar, ...),
- l'agriculture et l'élevage (les systèmes d'irrigation, les systèmes de traçabilité des animaux, ...),
- la surveillance de la santé des structures,
- la sécurité publique (identification/authentification des personnes avec des capteurs d'empreintes digitales ou des systèmes de reconnaissance faciale, ...),
- la Mobilité et transport (Électronique automobile, avionique, chemins de fer, technologie océanique et systèmes maritimes),
- le secteur de la santé (détection rapide et fiable des maladies, mesures préventives, ...),
- les bâtiments intelligents (réduction de la consommation d'énergie, amélioration de la sécurité et de la sûreté, ...),
- l'expérimentations scientifiques.

Généralités

Opportunités, défis et caractéristiques

Malheureusement, la conception des systèmes embarqués s'accompagne d'un grand nombre de problèmes difficiles. Les problèmes les plus courants sont les suivants :

- La **fiabilité**: un système capable de fournir le service prévu avec une probabilité élevée et ne cause aucun dommage ;
- Les systèmes embarqués doivent être **conscients des ressources** ;
- Les systèmes embarqués sont généralement constitués de composants matériels et logiciels hétérogènes de divers fournisseurs et doivent fonctionner dans un **environnement en évolution** ;
- La **conception de systèmes embarqués** implique des connaissances dans de nombreux domaines ;
- **Impact au-delà des problèmes techniques** : En raison de l'impact majeur sur la société, les impacts juridiques, économiques, sociaux, humains et environnementaux doivent également être pris en compte.

Généralités

Opportunités, défis et caractéristiques

Indépendamment du domaine d'application, il existe des caractéristiques plus communes des systèmes embarqués :

- Les systèmes embarqués sont des systèmes réactifs. Ils utilisent des capteurs et des actionneurs pour interagir avec le monde physique externe ;
- La plupart des systèmes embarqués n'utilisent pas de claviers, de souris et de grands écrans d'ordinateur pour leur interface utilisateur. Au lieu de cela, il existe une interface utilisateur dédiée composée de boutons poussoirs, de volants, de pédales, etc. De ce fait, l'utilisateur reconnaît à peine que le traitement de l'information est impliqué ;
- Ils sont souvent dédiés à une certaine application;
- Les systèmes embarqués sont sous-représentés dans l'enseignement et dans les débats publics. Les vrais systèmes embarqués sont complexes. Par conséquent, un équipement complet est nécessaire pour enseigner de manière réaliste la conception de systèmes embarqués.

Architecture et familles de microcontrôleur

- Microprocesseurs et microcontrôleurs
- Les exigences génériques
- Composants d'un microcontrôleur
- Structure de la mémoire, architecture de microprocesseurs et famille de microcontrôleurs

Architecture et familles de microcontrôleur

Microprocesseurs et microcontrôleurs

Un micro-ordinateur générique se compose de trois blocs fondamentaux :

- Unité Centrale de Traitement (Central Processing Unit (**CPU**)) ;
- Mémoire (**memory**);
- Système d'Entrée/Sortie (E/S) (Input/Output (**I/O**) system).

Ces blocs sont interconnectés par des groupes de lignes électriques appelés bus. Nous distinguons:

- les **bus d'adresses** transportant de la mémoire ou les adresses des E/S;
- Les **bus de données** transportant des données ou des instructions;
- Et les **bus de commande ou de contrôle** transportant des signaux de commande.

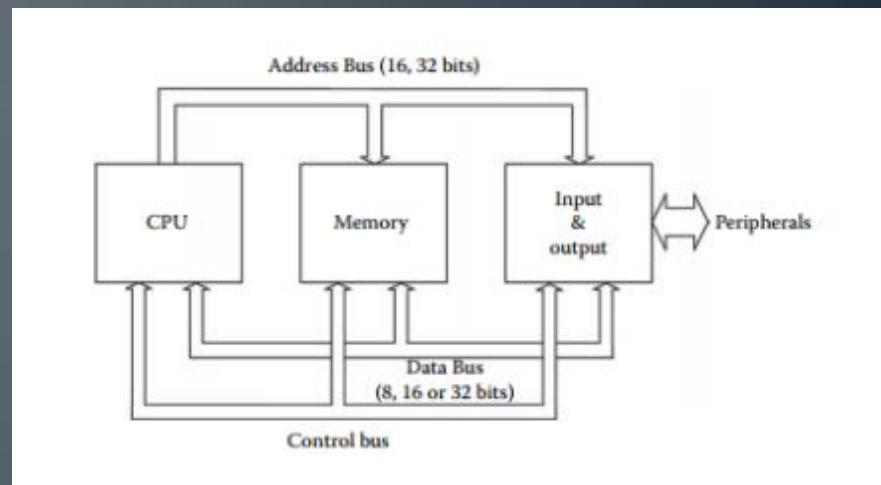


Schéma fonctionnel générique d'un micro-ordinateur.

Architecture et familles de microcontrôleur

Microprocesseurs et microcontrôleurs

Dans un micro-ordinateur, la **CPU** (le circuit intégré) est son microprocesseur. étant sous le contrôle du programme stocké en mémoire, la CPU est le cerveau du micro-ordinateur.

- Sa tache consiste à récupérer les instructions stockées en mémoire, à les interpréter et à les exécuter.

La **CPU** comprend également les circuits nécessaires pour effectuer des opérations arithmétiques et logiques avec des données binaires. Ce circuit spécial est appelé unité arithmétique et logique (Arithmetic and Logic Unit (ALU)).

Architecture et familles de microcontrôleur

Microprocesseurs et microcontrôleurs

Un microcontrôleur peut être considéré comme un micro-ordinateur construit sur un seul circuit intégré ou puce. Historiquement, les microcontrôleurs sont apparus après les microprocesseurs et ont suivi des chemins indépendants :

- ❑ Les microprocesseurs se trouvent principalement dans les ordinateurs personnels et les stations de travail, car ils nécessitent une forte puissance de calcul et la capacité de gérer de grands ensembles de données et d'instructions à grande vitesse.
- ❑ Un paramètre très important pour les microprocesseurs est la taille de leurs registres internes (8, 16, 32 ou 64 bits), car elle détermine le nombre de bits qui peuvent être traités simultanément.

Architecture et familles de microcontrôleur

Micropcesseurs et microcontrôleurs

- ❑ les microcontrôleurs peuvent être trouvé dans l'industrie automobile, les systèmes de communication, l'instrumentation électronique, les équipements hospitaliers, les équipements et applications industriels, les appareils ménagers, les jouets, etc. Ils ont donc été conçus pour être utilisés dans des applications où ils doivent effectuer un petit nombre de tâches au coût économique le plus bas possible.
- ❑ Ces tâches sont rendues possible grâce à un programme stocké de façon permanant dans leur mémoire en utilisant aux besoins les ports d'entrée-sortie pour interagir avec le monde extérieur.
- ❑ Le microcontrôleur devient une partie de l'application: c'est un contrôleur intégré au système.
- ❑ Les applications complexes peuvent utiliser plusieurs microcontrôleurs, chacun d'entre eux se concentrant sur un petit groupe de tâches.

Architecture et familles de microcontrôleur

Les exigences génériques

Les exigences suivantes sont importantes pour les microcontrôleurs et les conceptions utilisant des microcontrôleurs :

- ❑ **Ressources E/S:** Contrairement aux microprocesseurs dans lesquels l'accent est mis sur la puissance de calcul, les microcontrôleurs mettent l'accent sur leurs ressources d'entrée/sortie, telles que la capacité à gérer les interruptions, les signaux analogiques, le nombre de lignes d'entrée et de sortie différentes, etc.
- ❑ **Optimisation de l'espace:** Il est important d'utiliser le plus petit encombrement possible à un coût raisonnable. Etant donné que le nombre de broches dans une puce dépend de son emballage, l'encombrement peut être optimisé en ayant une broche capable de remplir plusieurs fonctions différentes.
- ❑ **Utiliser le microcontrôleur le plus approprié pour une application donnée:** Les fabricants de microcontrôleurs ont développé des familles de périphériques avec le même jeu d'instructions mais des aspects matériels différents, tels que la taille de la mémoire, les périphériques d'entrée/sortie, etc. Cela permet au concepteur de sélectionner l'appareil le plus approprié dans une famille donnée.
- ❑ **Protection contre les pannes:** Il est essentiel pour la sécurité de garantir que le microcontrôleur exécute le programme correct. Si, pour une raison quelconque, le programme s'égare, la situation doit être immédiatement corrigée. Les microcontrôleurs ont une minuterie de surveillance ou watchdog timer (WDT) pour s'assurer que le programme s'est exécuté correctement. Les minuteries Watchdog n'existent pas sur les ordinateurs personnels.
- ❑ **Faible consommation d'énergie:** Comme les batteries alimentent de nombreuses applications utilisant des microcontrôleurs, il est important de veiller à la faible consommation d'énergie des microcontrôleurs. En outre, l'énergie utilisée lorsque le microcontrôleur ne fait rien, doit être réduite au minimum. Pour ce faire, le microcontrôleur est mis en état de veille jusqu'à ce qu'il reprenne l'exécution du programme.
- ❑ **Protection des programmes contre les copies:** Le programme stocké en mémoire doit être protégé contre toute lecture non autorisée. Pour ce faire, les microcontrôleurs intègrent des mécanismes de protection contre la copie.

Architecture et familles de microcontrôleur

Composants d'un microcontrôleur

Les microcontrôleurs combinent les ressources fondamentales disponibles dans un micro-ordinateur telles que le processeur, la mémoire et les ressources d'E/S dans une seule puce.

- **Oscillateur (Oscillator)** : Les microcontrôleurs ont un oscillateur (en cristal de quartz (XTAL)) pour générer le signal nécessaire à la synchronisation de toutes les opérations internes.
- **RAM/ROM** : La mémoire du microcontrôleur stocke à la fois les instructions de programme et les données. Tout microcontrôleur a deux types de mémoire : la mémoire vive ou Random-Access Memory(RAM) (peut être lue et écrite mais volatile) et la mémoire morte ou read-only memory (ROM) (ne peut être que lue et est non volatile). La RAM et la ROM sont des mémoires à accès aléatoire (différent de l'accès séquentiel), ce qui signifie que le temps d'accès à des données spécifiques ne dépend pas de leur emplacement stocké. Il existe différents types pour la ROM :
 - **EPROM**(Erasable Programmable Read-Only Memory ou mémoire morte programmable effaçable) ;
 - **EEPROM**(Electrical Erasable Programmable Read-Only Memory ou mémoire morte programmable effaçable électrique) ;
 - **OTP**(One-Time Programmable ou programmable une fois) ;
 - **FLASH**

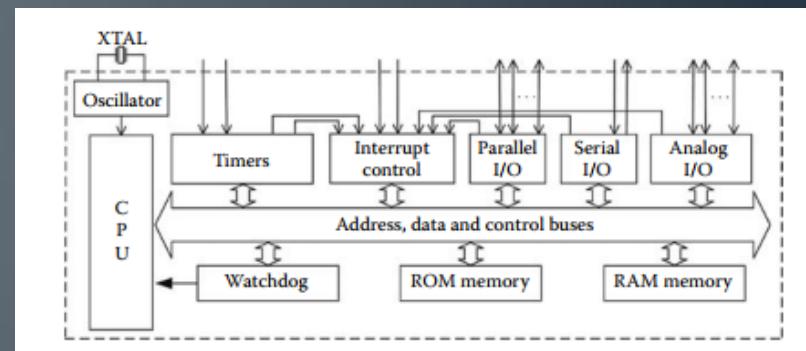


Schéma de principe d'un microcontrôleur

Architecture et familles de microcontrôleur

Composants d'un microcontrôleur

Les microcontrôleurs combinent les ressources fondamentales disponibles dans un micro-ordinateur telles que le processeur, la mémoire et les ressources d'E/S dans une seule puce.

□ **I/O ou E/S** : Les ressources E/S se composent des ports série et parallèle, des temporiseurs et des gestionnaires d'interruption. Certains microcontrôleurs intègrent également des lignes d'entrée et de sortie analogiques associées à des convertisseurs analogique-numérique ou analog-to-digital (A/D) et numérique-analogique ou digital-to-analog (D/A). Les ressources nécessaires pour assurer le fonctionnement régulier des microcontrôleurs tels que le chien de garde ou watchdog sont également considérées comme faisant partie des ressources d'entrée/sortie.

□ Les ports série peuvent être de différentes technologies: **RS-232C** (Recommended Standard 232, Revision C), **I₂C** (inter-integrated circuit), **USB** (universal serial bus) et **Ethernet**.

□ En général, un microcontrôleur disposera du plus grand nombre possible de ressources d'entrée/sortie pour le nombre de broches disponibles dans son boîtier de circuit intégré.

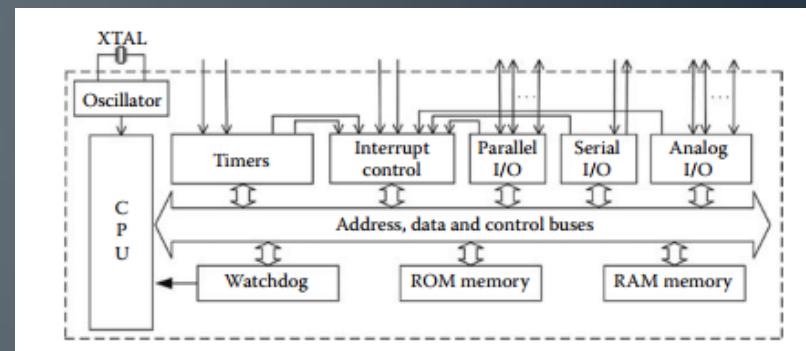


Schéma de principe d'un microcontrôleur

Architecture et familles de microcontrôleur

Composants d'un microcontrôleur

□ **CPU (Central Processing Unit)** : Semblable aux micro-ordinateurs, la CPU est le cerveau du microcontrôleur. Il est constitué essentiellement de trois parties :

- **L'unité de commande** : Qui cherche les instructions en mémoire, les décode et coordonne le reste du processeur pour les exécuter.
- **L'unité Arithmétique et Logique (ALU)** : Qui exécute les instructions arithmétiques et logiques demandées par l'unité de commande.
- **Les registres** : Ils servent à stocker des variables, les résultats intermédiaires d'opérations (arithmétiques ou logiques) ou encore des informations de contrôle du processeur. Ils sont variés, certains sont destinés à un usage général, et d'autres ont un but spécifique. On distingue entre autres :
 - **Le registre d'instruction** ou Instruction Register (IR) qui contient le code de l'instruction qui est traitée par le décodeur/séquenceur ;
 - **L'accumulateur** ou accumulator (ACC) qui est principalement destiné à contenir les données qui doivent être traitées par l'ALU ;
 - **Les registres généraux** qui servent au stockage de résultats intermédiaires ;
 - **Le registre d'adresses de données** ou Data Address Register (DAR) qui stocke les adresses de données de la mémoire ;
 - **Le compteur ordinal** ou Program Counter (PC) qui contient l'adresse de la prochaine instruction à exécuter ;
 - **Le registre d'état** ou status register (STATUS) qui contient les bits qui présentent différentes caractéristiques liées aux opérations effectuées par l'ALU.

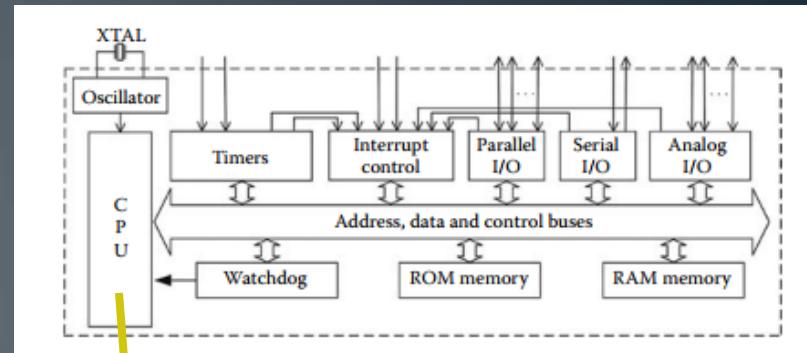


Schéma de principe d'un microcontrôleur

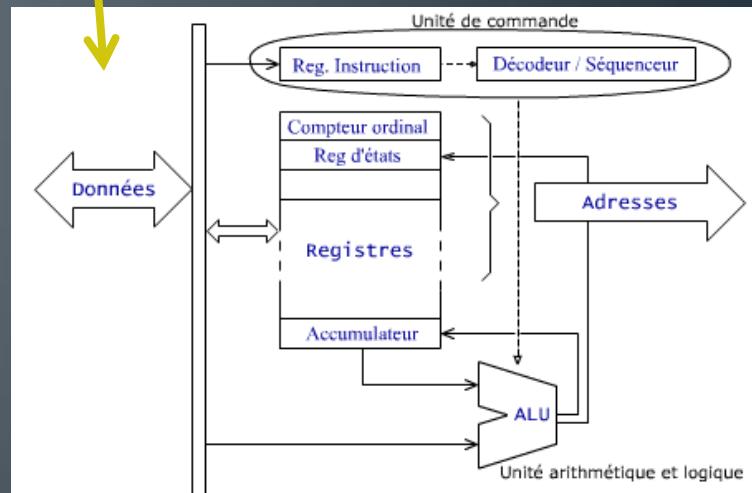
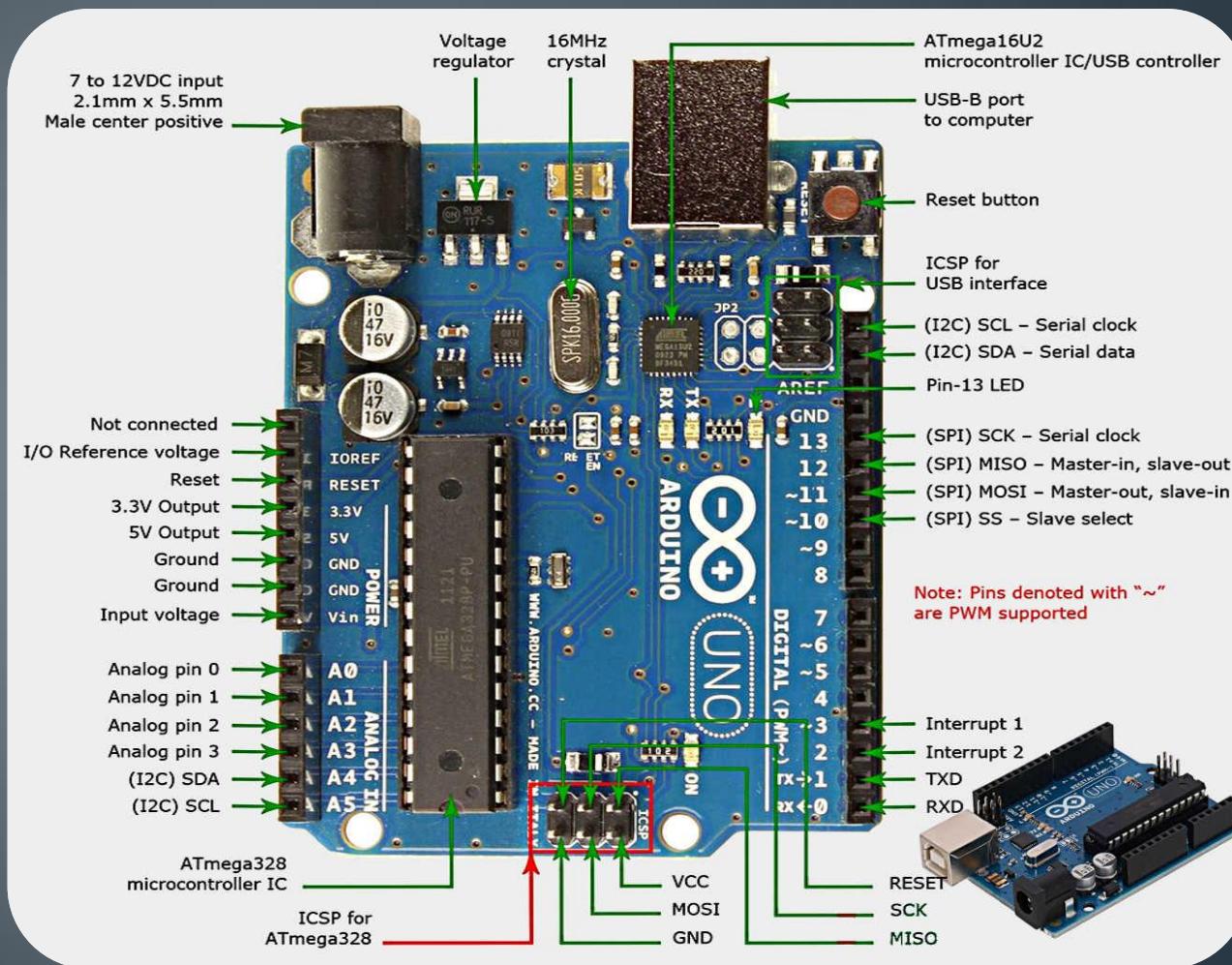


Schéma de la CPU

Architecture et familles de microcontrôleur

Composants d'un microcontrôleur



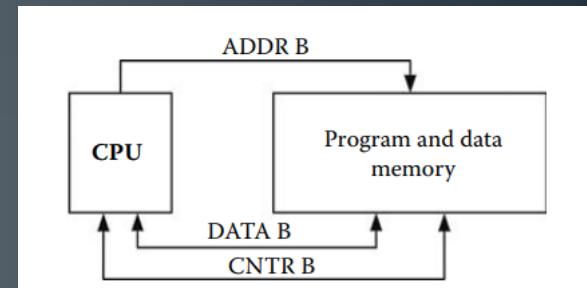
Les composants de la carte Arduino UNO

Architecture et familles de microcontrôleur

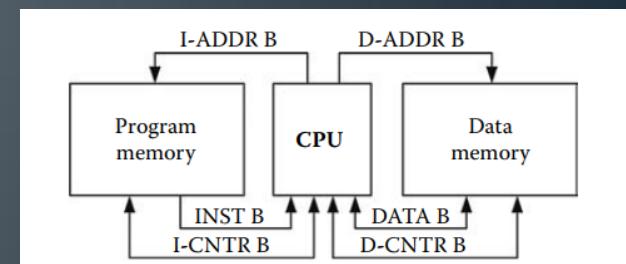
Structure de la mémoire, architecture de microprocesseurs et famille de microcontrôleurs

La façon dont la mémoire est organisée et dont elle communique avec le CPU détermine les performances de l'appareil. Les deux modèles génériques de matériel pour la structure de la mémoire sont appelés architectures Von Neumann et Harvard.

- L'architecture **Von Neumann** a été proposée par le mathématicien John von Neumann lorsqu'il a conçu l'Intégrateur et calculateur numérique électronique ou Electronic Numerical Integrator and Calculator (ENIAC) à l'Université de Pennsylvanie pendant la seconde guerre mondiale. Il a eu l'idée originale de développer un ordinateur à programmes stockés. L'architecture von Neumann utilise moins de lignes que l'architecture Harvard, ce qui rend la connexion entre le processeur et la mémoire beaucoup plus simple. Toutefois, cette structure ne permet pas le traitement simultané de données et d'instructions car il n'y a qu'un seul bus.
- L'architecture **Harvard** a été proposée par Howard Aiken lorsqu'il a développé les ordinateurs connus sous le nom de Mark I, II, III et IV à l'université de Harvard. Ce furent les premiers ordinateurs à utiliser des mémoires différentes pour stocker séparément les données et les instructions, ce qui constitue une approche très différente de l'ordinateur à programme stocké. L'architecture de Harvard utilise différentes mémoires pour stocker les instructions et les données. La mémoire de programme possède son propre bus d'adresse (bus d'adresse d'instruction), son propre bus de données (plus correctement appelé bus d'instruction) et son propre bus de contrôle. La mémoire de données possède son propre bus d'adresse, son propre bus de données et son propre bus de commande, indépendamment des bus d'instruction.



Architecture Von Neumann



Architecture Harvard

Architecture et familles de microcontrôleur

Structure de la mémoire, architecture de microprocesseurs et famille de microcontrôleurs

L'ordinateur à jeu d'instructions complexes ou Complex Instruction Set Computer (**CISC**) et l'ordinateur à jeu d'instructions réduit ou Reduced Instruction Set Computer (**RISC**) sont des architectures de microprocesseurs développées pour permettre une compilation efficace des programmes.

- **RISC** fonctionne avec un minimum de jeu d'instructions hautement optimisé, au lieu du jeu d'instructions plus spécialisé que l'on trouve dans d'autres types d'architectures. Cela signifie que le microprocesseur aura moins de cycles par instruction. Il contient des instructions relativement simples et basiques à partir desquelles des instructions plus complexes peuvent être produites.
- **Quelques avantages des processeurs RISC :**
 - Il permet aux compilateurs de langage de haut niveau de produire un code plus efficace ;
 - RISC maximise l'efficacité en minimisant le temps d'exécution ;
 - Comme il est relativement simple, très peu d'instructions et quelques modes d'adressage sont nécessaires ;
- **Inconvénients des processeurs RISC :**
 - La performance des processeurs RISC dépend du compilateur ou du programmeur car les instructions suivantes peuvent dépendre de l'instruction précédente pour compléter leur exécution ;
 - Les processeurs RISC nécessitent des systèmes de mémoire très rapides pour alimenter les différentes instructions.

Architecture et familles de microcontrôleur

Structure de la mémoire, architecture de microprocesseurs et famille de microcontrôleurs

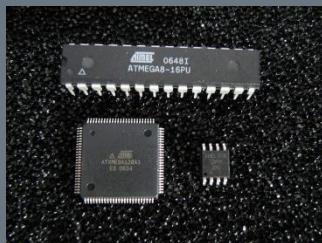
- CISC est constitué d'instructions uniques pouvant exécuter plusieurs opérations de bas niveau, par exemple, "charger de la mémoire une opération arithmétique, et un stockage en mémoire). Les processeurs CISC sont également capables d'exécuter des opérations multi-étages.
 - C'est un type de microprocesseur qui contient des instructions spécialisées simples/complexes.
 - Les processeurs du CISC ont permis de simplifier le code et de le rendre plus court afin de réduire les besoins en mémoire.
 - Dans les processeurs CISC, chaque instruction comporte plusieurs opérations de bas niveau. Cela rend les instructions CISC courtes, mais complexes.
- Quelques avantages des processeurs CISC :
 - Les exigences en matière de mémoire sont réduites au minimum ;
 - L'exécution d'une seule instruction permet de réaliser plusieurs tâches de bas niveau ;
 - L'accès à la mémoire est plus souple en raison du schéma d'adressage complexe ;
 - Les emplacements mémoire peuvent être directement accessibles par les instructions CISC ;
- Inconvénients des processeurs CISC :
 - Bien que la taille du code soit réduite au minimum, le code nécessite plusieurs cycles d'horloge pour exécuter une seule instruction. Cela réduit l'efficacité du système ;
 - CISC a été conçu pour réduire au minimum les besoins en mémoire lorsque la mémoire est plus petite et plus coûteuse . Aujourd'hui, la mémoire est peu coûteuse et la majorité des nouveaux systèmes informatiques disposent d'une grande quantité de mémoire.

Architecture et familles de microcontrôleur

Structure de la mémoire, architecture de microprocesseurs et famille de microcontrôleurs

Les différents microcontrôleurs qui ont le même noyau, c'est-à-dire qui partagent le même processeur et exécutent le même jeu d'instructions, sont appelés une famille de microcontrôleurs. Les différents périphériques d'une même famille ont le même noyau, mais ils diffèrent par leurs capacités d'entrée/sortie et la taille de leur mémoire.

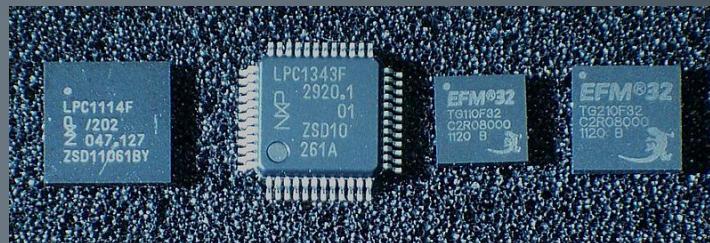
- ❑ **Exemple :** Tous les microcontrôleurs de la famille 8051 (MCS51) ont une unité centrale similaire et exécutent le même jeu d'instructions. Cependant, les différents membres de la famille ont des nombres et des types de ports d'entrée/sortie différents ainsi que des types et des tailles de mémoire différents.



Atmel AVR

AVR est une famille de microcontrôleurs développée depuis 1996 par Atmel, acquise par Microchip Technology en 2016.

- ❑ Architecture processeur: Harvard (modifiée) ;
- ❑ Jeu d'instructions: RISC (8 bits).



ARM Cortex-M

Cortex-M est une famille de processeur développée par ARM Ltd, servant à la fois de microprocesseur et de microcontrôleur à destination de l'embarqué.

- ❑ Architecture processeur: ARM (propriétaire) ;
- ❑ Jeu d'instructions: RISC (32 bits).

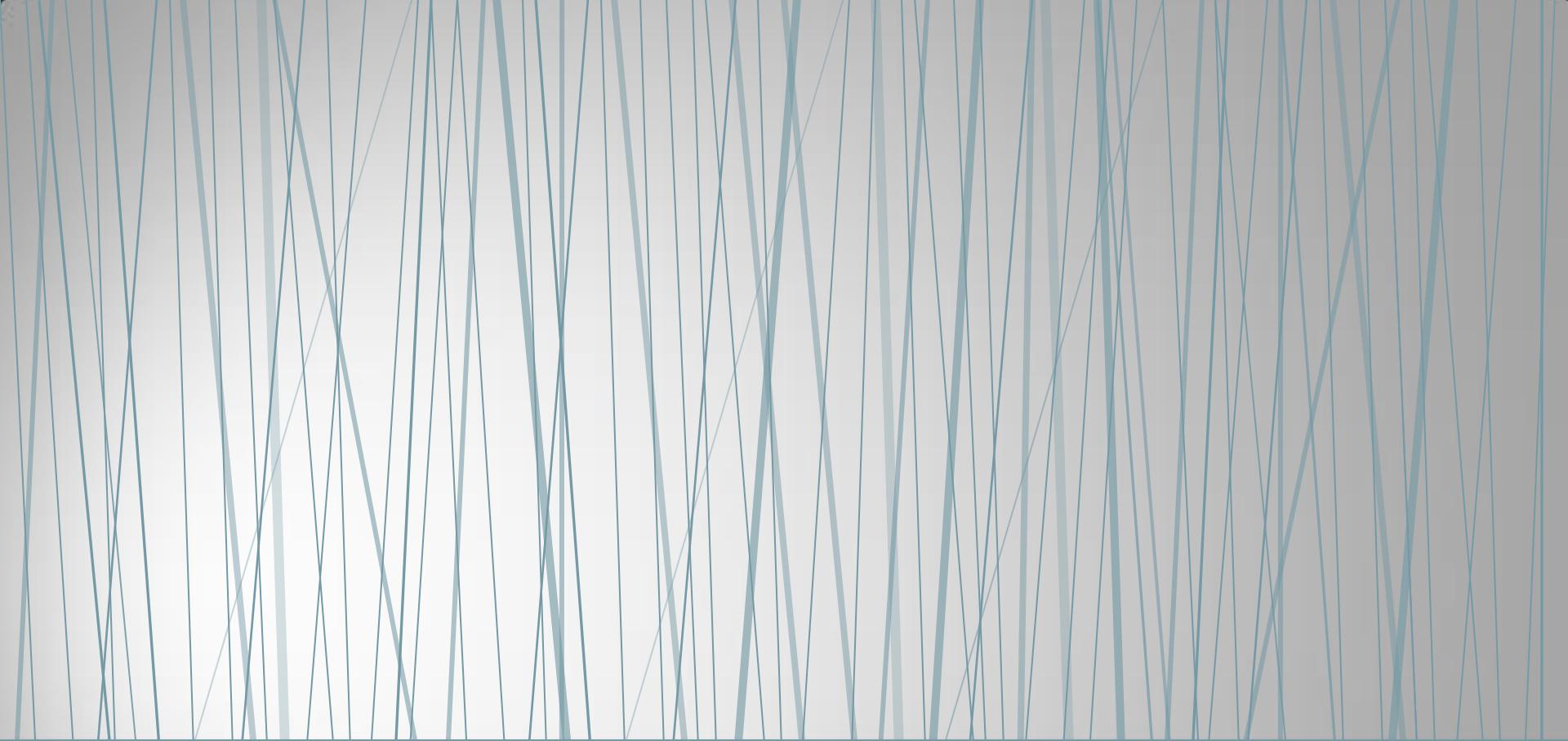
Une distinction est également faite entre la famille Cortex-A destinée au marché des Smartphones et des tablettes tactiles et la famille Cortex-R destinée au temps réel.



PIC

PICMicro est une famille de microcontrôleurs de la société Microchip.

- ❑ Architecture processeur: Harvard;
- ❑ Jeu d'instructions: RISC (8, 16 et 32 bits).



Capteurs, actionneurs et programmation des microcontrôleurs

- Capteurs
 - Actionneurs
 - Programmation des microcontrôleurs
-

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

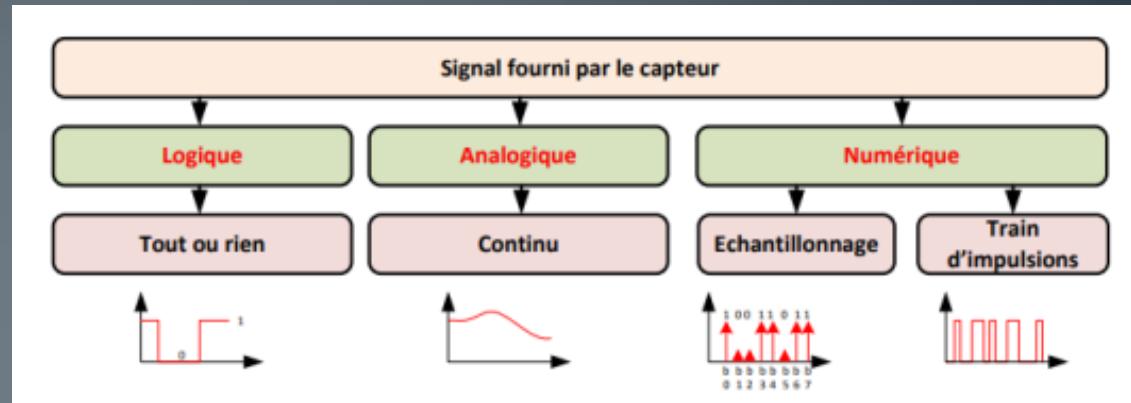
Le capteur est l'organe permettant d'élaborer, à partir d'une grandeur physique observée (température, pression, position, concentration, etc.), une grandeur physique utilisable (souvent électrique) à des fins de mesure ou de commande.

- Ils sont donc à la base des systèmes d'acquisition de données et leur mise en œuvre est du domaine de l'instrumentation ;
- Un capteur est caractérisé soit par son temps de réponse, sa linéarité, la grandeur physique observée, sa gamme de mesure, sa précision, sa sensibilité, sa résolution, etc.
- Ils sont classés en types de capteurs, à savoir les capteurs actifs, les capteurs passifs, etc.
 - Les capteurs actifs ont la capacité de convertir en énergie électrique la forme d'énergie de la mesurande : thermique, mécanique, , en utilisant des effets tels que l'induction électromagnétique, Hall, piézoélectrique, thermoélectrique, photoélectrique, Faraday, Ils sont présentés comme un générateur car ils délivrent à leur sortie soit une tension, soit un courant, soit une charge électrique ;
 - Les capteurs passifs sont une sorte d'impédance sensible à la mesure. Ils peuvent délivrer une grandeur telle que la variation de l'impédance, la résistance, l'inductance ou la capacité et il est donc nécessaire de leur appliquer une tension pour obtenir un signal de sortie ;

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

- Les capteurs peuvent être classés en trois groupes selon la nature de l'information de sortie :



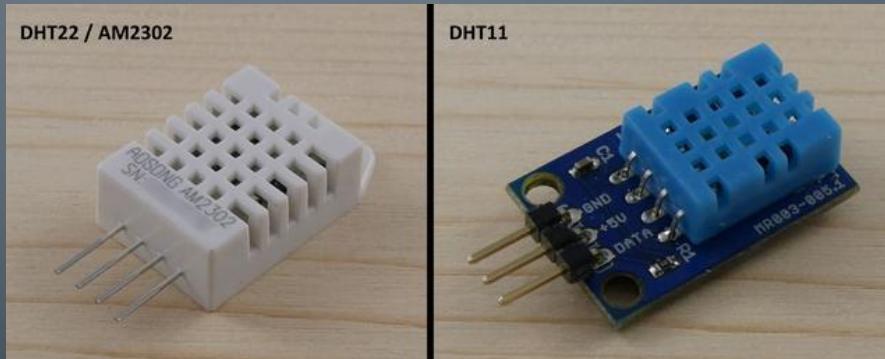
- Voici, ci-dessous, quelques-uns des capteurs disponibles et accessibles sur le marché :
 - Capteurs de température analogique : LM35, TMP36, ... ;
 - Capteurs de température numérique : DS18 ;
 - Capteurs de température et d'humidité : DHT11, DHT22, SHT11, SHT15, SHT21, ... ;
 - capteur de luminosité : Photorésistance, capteur infrarouge (pour éviter les obstacles),
 - capteur de détection de rayons ultraviolets, capteur de couleur ;
 - Capteur de distance : Capteurs à ultrasons (HC-SR04) ;
 - Capteur de mouvement : HC-SR501 ;
 - Capteur de gaz et de fumée : les MQ (MQ2, MQ3, ...) ;
 - Capteur de son analogique : KY-038 ;
 - Capteur de vibration numérique : SW-420 ;
 - Capteur ou lecteur RFID : RC522 ;
 - Capteur ou lecteur d'empreintes digitales : AS608 ;
 - Capteur ou détecteur d'incendie : KY-026 ;

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

□ Exemple: le DHT22 et le DHT11

- Le capteur DHT22 / AM2302 est capable de mesurer des températures de -40 °C à +125°C avec une précision de +/- 0,5°C et une humidité relative de 0 à 100% avec une précision de +/- 2% (+/- 5% aux extrêmes, 10% et 90%). Une mesure peut être effectuée toutes les 500 millisecondes.
- Le capteur DHT11 est capable de mesurer des températures de 0 °C à +50°C avec une précision de +/- 2°C et une humidité relative de 20 à 80% avec une précision de +/- 5%. Une mesure peut être effectuée toutes les secondes.
- Ils consomment 3,3 volts ou 5 volts et ont le même câblage et protocole de communication.



Images

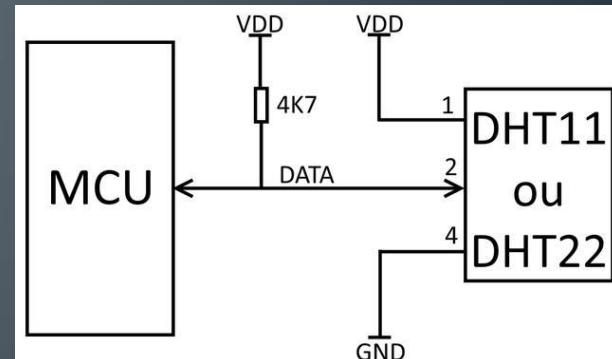


schéma de câblage

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

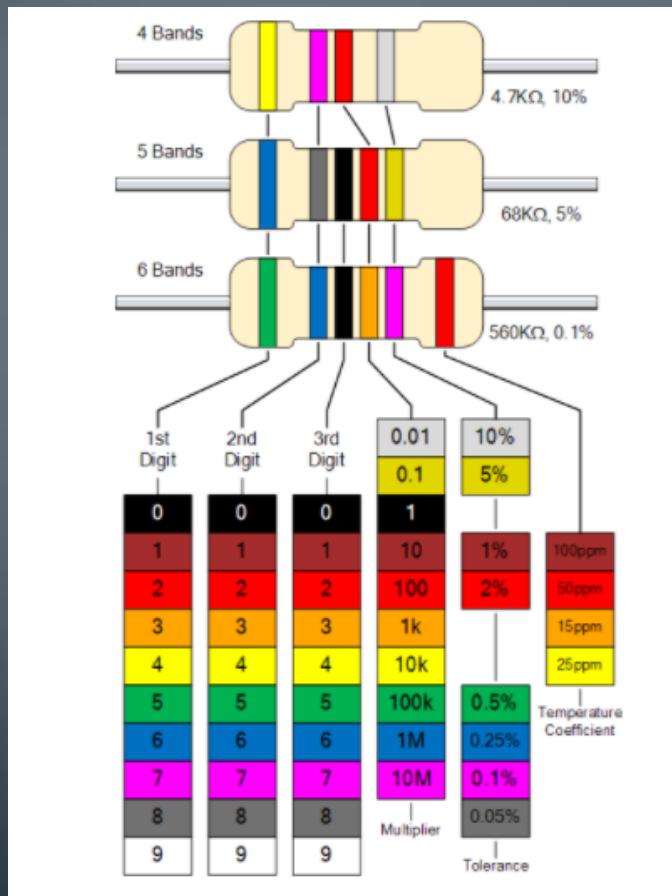
Les boutons poussoirs qui sont des dispositifs de commande manuelle permettant de contrôler un aspect d'une machine ou d'un processus (par exemple, allumer/éteindre une lampe, etc.). C'est un interrupteur monostable, c'est-à-dire lorsqu'il est relâché, il retourne seul dans la position repos.



Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Une **résistance** est un composant électronique ou électrique dont la principale caractéristique est d'opposer une plus ou moins grande résistance à la circulation du courant électrique. Il est exprimé en ohms: Ω .



La loi d'Ohm: $U=R*I$

- Une résistance est un milieu peu conducteur et donc les électrons peinent à s'y déplacer.
- Leur énergie se dissipe alors en général sous forme de chaleur. Perte par effet joule.
- La valeur d'une résistance est déterminée par ses bandes de couleurs.
 - Il est très utile de connaître la valeur d'une résistance, bien que la plupart des modules comprennent maintenant les composants nécessaires à l'interfaçage avec les microcontrôleurs.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Un **actionneur** est un composant d'une machine qui permet d'effectuer des mouvements physiques en convertissant l'énergie (électrique, pneumatique, hydraulique, ...) en force mécanique (mouvement linéaire(poussée/traction) ou rotatif).

- Les actionneurs peuvent être dans un système de commande simple, tel qu'un moteur mécanique, ou dans un système informatique, tel qu'un robot.
- Les actionneurs peuvent être classés en fonction du type d'énergie qu'ils utilisent et du type de mouvement qu'ils produisent.
 - **Les actionneurs électriques** : ce sont les plus courants. Ils convertissent l'énergie électrique du courant continu ou alternatif en énergie mécanique. On distingue les actionneurs électriques linéaires et les actionneurs électriques rotatifs (les moteur en mouvement continu, les servomoteurs et les moteurs pas à pas).
 - **Les actionneurs pneumatiques** : Les actionneurs pneumatiques permettent d'effectuer de grands mouvements linéaires ou rotatifs à basse pression en utilisant de l'air comprimé. Leur force réside dans leur mouvement rapide, point à point, et ils ne sont pas facilement endommagés par les butées.
 - **Les actionneurs hydraulique** : Les actionneurs hydrauliques utilisent l'énergie hydraulique pour créer un mouvement linéaire ou rotatif et sont très puissants en raison de la quasi incompressibilité des liquides.

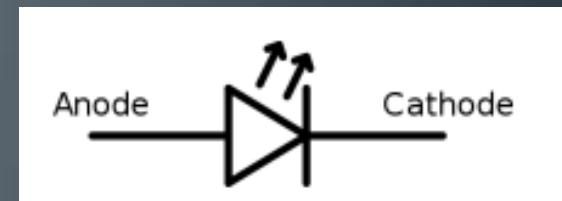
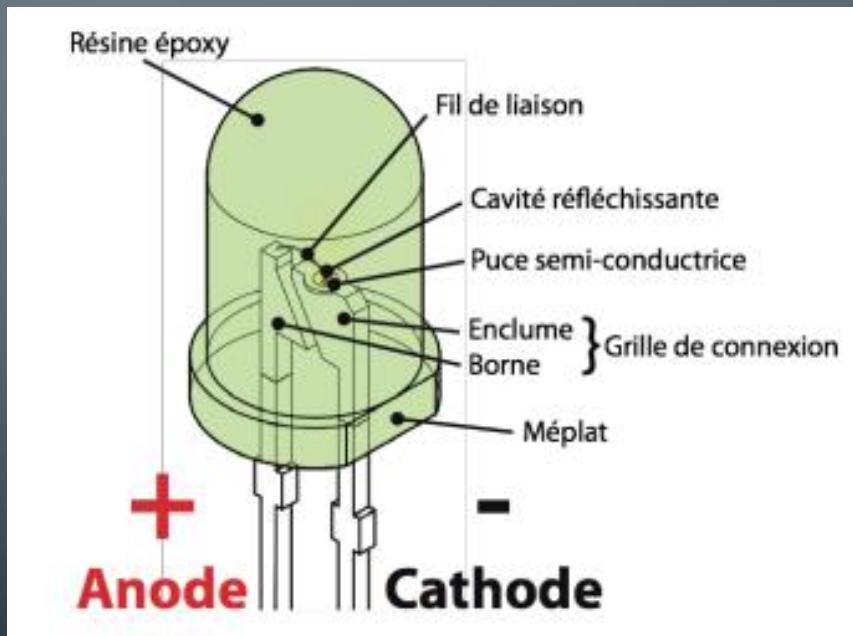


Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Les ampoules et les **leds** peuvent être mis dans cette catégorie. Elles ont la capacité de transformer l'énergie électrique en énergie lumineuse.

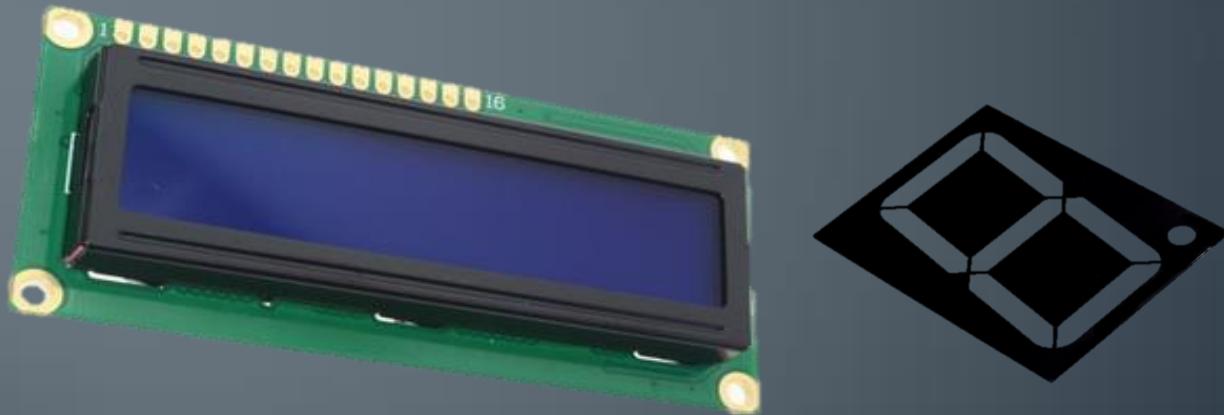
- Les leds sont polarisés, c'est-à-dire qu'elles ont la particularité de ne laisser passer le courant électrique que dans un sens (de l'anode vers la cathode).
- L'anode correspond à la broche la plus longue. Elle est symbolisée comme suit:

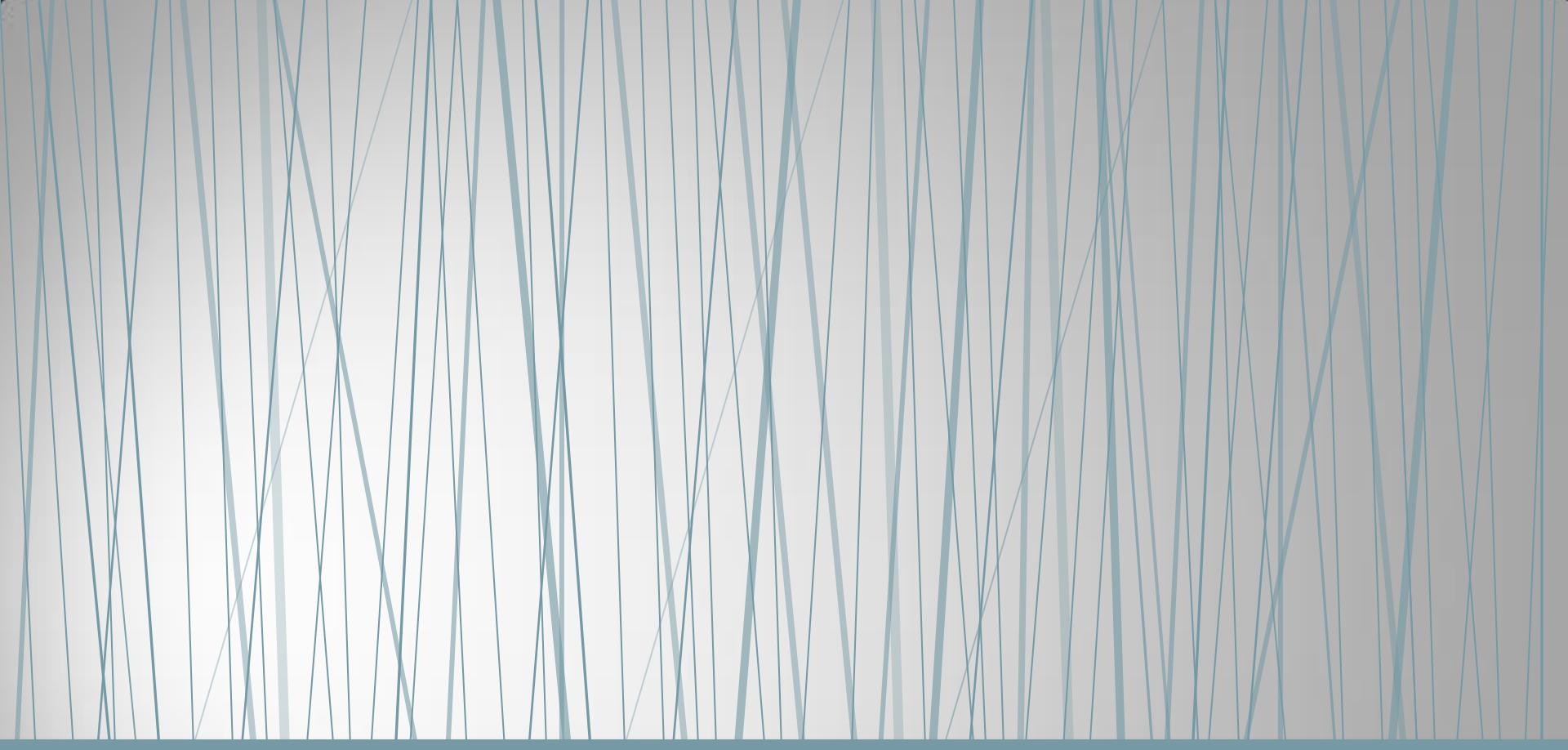


Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Les écrans LCD (Liquid Crystal Display) et les afficheurs 7 segments sont également largement utilisés dans le domaine des systèmes embarqués (montres, tableau de bord, calculatrices, etc.) en raison de leur faible consommation d'énergie et de leur faible coût d'acquisition.





Programmation des microcontrôleurs

La programmation sous Arduino

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Tout microcontrôleur est appelé à être programmé car étant sous le un programme stocké. Dans ce cas, il suffit de connaitre le langage dans lequel il est implémenté, de savoir manipuler les bibliothèques qu'il propose et puis d'être familier avec IDE (Integrated Development Environment) utilisé pour la programmation. Dans cette section, nous serons intéressé par la programmation sous Arduino et sur Raspberry.

I. La programmation sous Arduino

Arduino est un microcontrôleur open-source qui permet la programmation et l'interaction. Il est basé sur C/C++ avec une bibliothèque Arduino pour lui permettre d'accéder au matériel.

Jusqu'à présent, grâce à sa nature open-source, les utilisateurs ont construit des cartes Arduino de différentes tailles, formes et niveaux de puissance pour contrôler leurs projets.

□ Arduino est composé de deux parties principales :

- La carte Arduino, qui est le matériel sur lequel vous travaillez lorsque vous construisez vos objets ;
- L'IDE Arduino, qui est le logiciel que vous exécutez sur votre ordinateur pour créer un croquis (un petit programme informatique) que vous téléchargez sur la carte Arduino.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

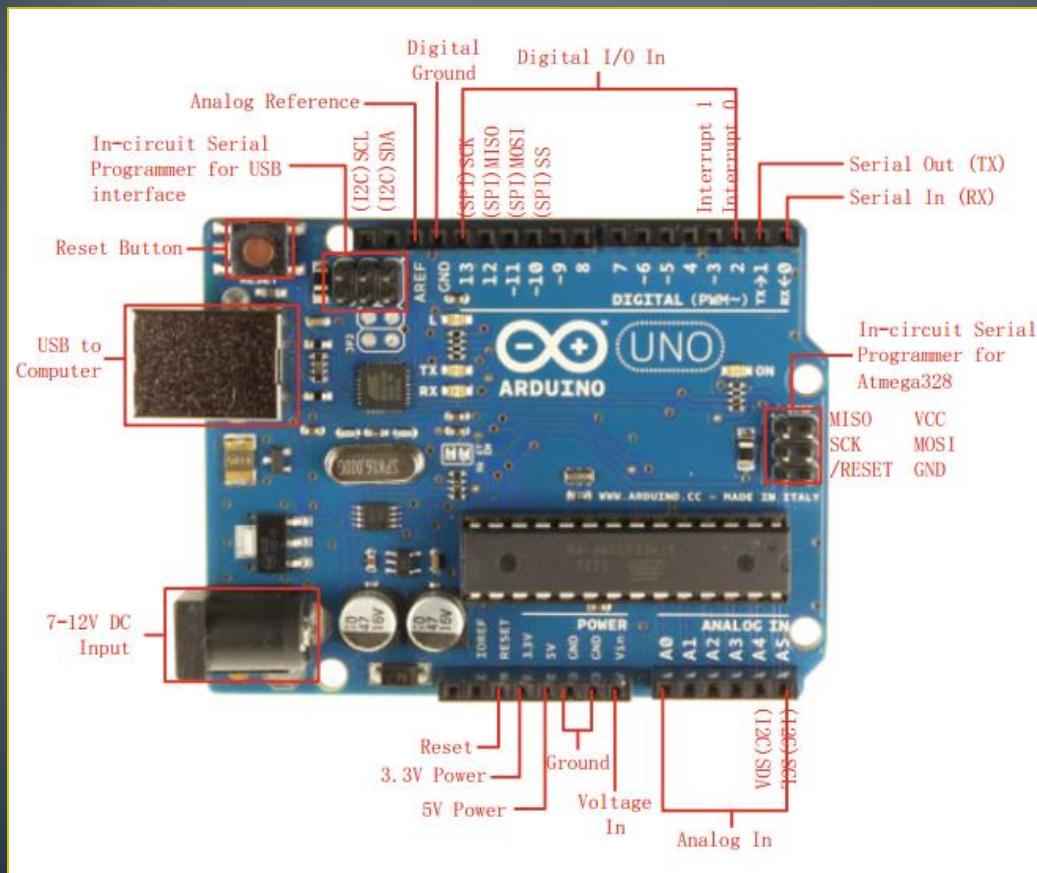
Il existe de nombreux types de microcontrôleurs Arduino qui diffèrent non seulement par leur conception et leur fonctionnalité, mais aussi par leur taille et leurs capacités de traitement. Cependant, seuls deux modèles utilisent des puces complètement différentes : le modèle Standard utilise la puce Atmega 8/168/328 et le modèle Mega utilise la puce Atmega1280, plus robuste, avec plus de broches d'entrée/sortie.

Il existe de nombreux autres fabricants qui utilisent des schémas open-source fournis par Arduino pour fabriquer leurs propres planches (soit identiques à l'original, soit avec des variations pour ajouter des fonctionnalités), par exemple, DFRobot.

Pour des besoins de simulation, nous utilisons uniquement la carte Arduino Uno R3.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs



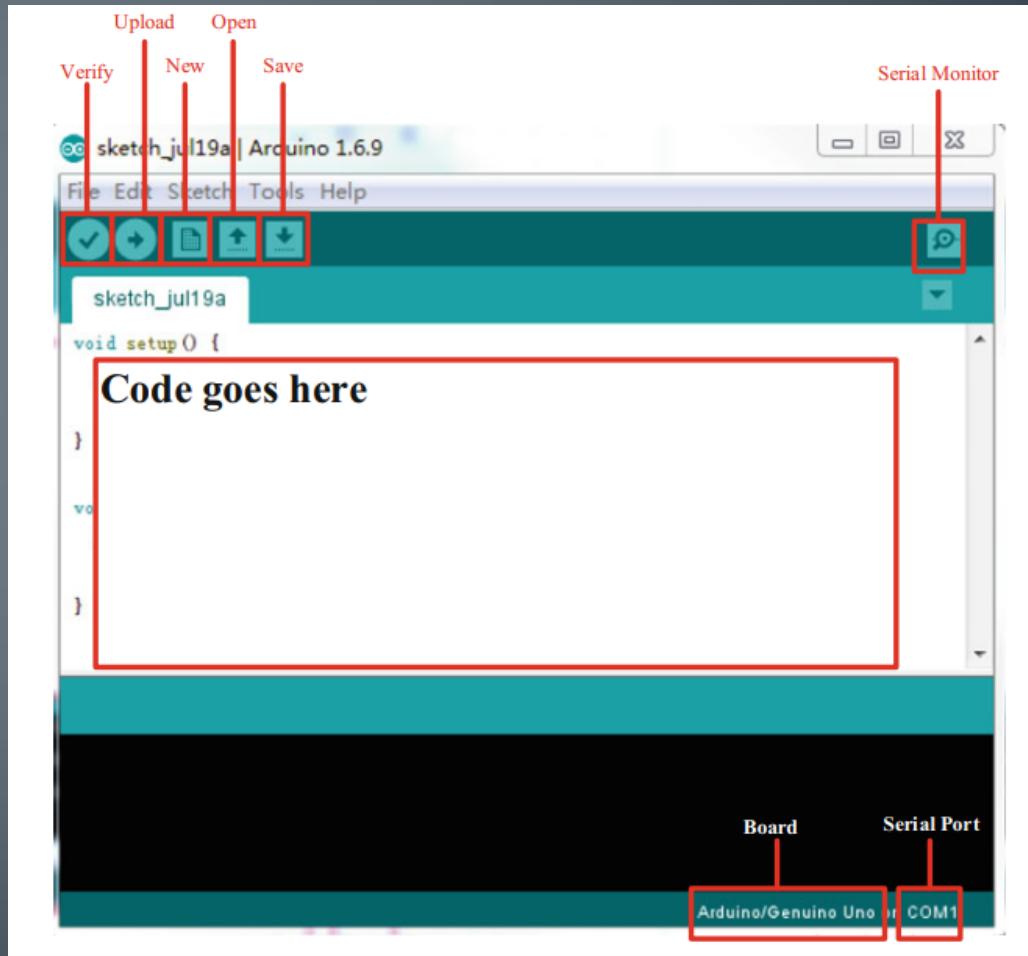
Uno est essentiellement constituée de :

- 14 broches d'entrée/sortie numériques ;
- 6 entrées analogiques;
- un résonateur céramique de 16 MHz ;
- une connexion USB ;
- une prise d'alimentation ;
- un bouton de réinitialisation ;
- Etc.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

- Avant de commencer votre travail, vous devez d'abord télécharger l'environnement de développement (l'IDE) sur : wwwarduino.cc/en/Main/Software .



Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Le langage Arduino est basé sur C/C++ et supporte toutes les constructions C standard ainsi que certaines fonctionnalités C++. Il est lié à AVR Libc et permet l'utilisation de n'importe laquelle de ses fonctions.

Structures	
void setup()	La fonction est appelée quand un sketch commence. Utilisez-la pour initialiser des variables, les pin modes, commencer à utiliser des bibliothèques, etc. La fonction de configuration ne sera exécutée qu'une seule fois, après chaque mise sous tension ou réinitialisation de la carte Arduino.
void loop()	la fonction fait des boucles consécutives, permettant à votre programme de changer et de répondre. Utilisez-le pour contrôler activement la carte Arduino.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Structure de contrôle	
<pre>if (condition) { //instructions }</pre>	Teste si une certaine condition a été atteinte. Utilisé en conjonction avec un opérateur de comparaison.
<pre>switch (var) { case label : //instructions break ; ... default : //instructions }</pre>	Vous permet de spécifier différents codes qui doivent être exécutés dans diverses conditions.
<pre>for(init; condition; incrément){ //instructions }</pre>	Crée une boucle pour les opérations dont on connaît le nombre de répétition.
<pre>while (condition) { //instructions }</pre>	Boucles continues, et infinies, jusqu'à ce que l'expression à l'intérieur de la parenthèse, devienne fausse.
<pre>do { //instructions } while (condition);</pre>	Fonctionne de la même manière que la boucle while, à l'exception du fait que l'état est testé à la fin de la boucle, tout comme la boucle do, qui sera toujours exécutée au moins une fois.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Structure de contrôle	
break	Sorties d'une boucle do, for ou while, en contournant la condition normale de la boucle.
continue	Saute le reste de l'itération actuelle d'une boucle (do, for, ou while). Il continue en vérifiant l'expression conditionnelle de la boucle, et procède à toutes les itérations suivantes.
return	Termine une fonction et renvoie une valeur d'une fonction à la fonction appelante.
goto	Transfère le flux du programme vers un point labellisé dans le programme.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Autre syntaxe	
;	Chaque déclaration se termine par un point-virgule.
{ }	Les accolades bouclées viennent toujours par paires ; ils sont utilisés pour définir le début et la fin des fonctions, des boucles et des instructions conditionnelles.
//	Commentaire d'une seule ligne.
/* */	Commentaire de plusieurs lignes.
#define	Utilisé pour donner un nom à une valeur constante.
#include	Inclure les bibliothèques extérieures dans votre croquis.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Opérateurs de comparaison	
$x == y$	x est égal à y .
$x != y$	x n'est pas égal à y .
$x < y$	x est inférieur à y .
$x > y$	x est supérieur à y .
$x <= y$	x est inférieur ou égal à y .
$x >= y$	x est supérieur ou égal à y

Les opérateurs booléens	
$&&$	ET logique. Vrai seulement si les deux opérandes sont vrais.
$ $	OU logique. Vrai si l'un des opérandes est vrai.
$!$	NON logique. Vrai si l'opérande est faux

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Constantes	
HIGH / LOW	Lors de la lecture ou de l'écriture sur une broche numérique, il n'y a que deux valeurs possibles qu'une broche peut prendre (ou être réglée sur) : HIGH et LOW
true / false	Niveaux logiques (résultat d'une comparaison) : false est défini comme 0, true est défini comme 1 (mais plus largement, tout sauf 0).
INPUT / OUTPUT	Les broches numériques peuvent être utilisées en INPUT (entrée) ou en OUTPUT (sortie). Changer une broche de INPUT à OUTPUT avec pinMode() change radicalement le comportement électrique de la broche.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Types de données	
void	Utilisé dans les déclarations de fonction pour indiquer que la fonction ne renvoie aucune information.
boolean	Un booléen possède l'une des deux valeurs suivantes, vraie ou fausse.
char	Un type de données qui stocke une valeur de caractère.
unsigned char	Un type de données non signé qui occupe 1 octet de mémoire. Identique au type de données byte. Le type de données char non signé encode les nombres de 0 à 255.
byte	Un byte stocke un nombre non signé de 8 bits, de 0 à 255.
int	Les nombres entiers sont votre principal type de données pour le stockage des nombres, et stockent une valeur de 2 octets. Cela donne une plage de -32 768 à 32 767
unsigned int	Les ints non signés (entiers non signés) sont identiques aux int en ce sens qu'ils stockent une valeur de 2 octets. Au lieu de stocker des nombres négatifs, ils ne stockent que des valeurs positives, ce qui donne une plage utile de 0 à 65 535 .

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Types de données	
word	Un word stocke un nombre non signé de 16 bits, de 0 `a 65536. Même chose qu'un int non signé.
long	Les variables de type long sont des variables de taille étendue pour le stockage des nombres et stockent 32 bits (4 octets), de -2 147 483 648 à 2 147 483 647.
unsigned long	Contrairement aux longs standards, les longs non signés (unsigned long) ne stockent pas de nombres négatifs, ce qui fait que leur plage de stockage va de 0 `a 4 294 967 295.
float	Type de données pour les nombres `a virgule flottante, un nombre qui a un point décimal.
double	Les nombres à virgule flottante sont souvent utilisés pour approximer des valeurs analogiques et continues car ils ont une plus grande résolution que les nombres entiers. Les nombres à virgule flottante ont une précision de 6 à 7 chiffres décimaux.
string	Les chaînes sont représentées sous forme de tableaux de type char et se terminent par null.
arrays	Il est souvent pratique, lorsqu'on travaille avec de grandes quantités de texte, comme dans le cas d'un projet avec un écran LCD, de mettre en place une série de chaînes.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Fonctions : E/S numériques	
pinMode(pin, mode)	Configure la broche spécifiée pour qu'elle se comporte comme une entrée ou une sortie. pin est le numéro de pin.
digitalWrite(pin, value)	Ecrire une valeur HIGH ou LOW sur une broche numérique.
digitalRead(pin)	Lire la valeur à partir d'une broche numérique spécifiée. Le résultat sera soit HIGH, soit LOW.
Fonctions : E/S analogiques	
analogReference(type)	La tension de référence par défaut est de 5 V. Cela peut être changé en un autre type et une résolution différente en utilisant cette fonction.
analogRead(pin)	Lire la valeur de la broche analogique spécifiée et renvoie une valeur comprise entre 0 et 1023 pour représenter une tension entre 0 et 5 V (par défaut). Il faut environ 0,0001 s pour lire une broche analogique.
analogWrite(pin,value)	Ecrire une valeur analogique (onde PWM) sur une broche. valeur est le rapport cyclique : entre 0 (toujours désactivé) et 255 (toujours activé). Fonctionne sur les broches 3, 5, 6, 9, 10 et 11.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Fonctions : communication série	
Serial.begin(9600)	Utilisé pour commencer les communications série, généralement à un débit de 9600 bauds (bits par seconde).
Serial.print(val,format)	Imprime les données sur le port série sous forme de texte ASCII lisible par l'homme.
Serial.println(val)	Imprime le val suivi du retour chariot.
Serial.available()	Obtenez le nombre d'octets (caractères) disponibles pour la lecture à partir du port série. Il s'agit des données déjà arrivées et stockées dans le tampon de réception série (qui contient 128 octets).
Serial.read()	Lire les données série entrantes.
Serial.write()	Ecrire des données binaires sur le port série.
Serial.end()	Désactive la communication série, ce qui permet d'utiliser les broches RX et TX pour les entrées et sorties générales.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Temps	
delay(ms)	Suspend le programme pendant la durée (en millisecondes) spécifiée en paramètre.
delayMicroseconds(us)	Met le programme en pause pendant la durée (en microsecondes) spécifiée en paramètre.
micros()	Retourne le nombre de microsecondes depuis que la carte Arduino a commencé à faire fonctionner le programme actuel.
millis()	Renvoie le nombre de millisecondes écoulées depuis que la carte Arduino a commencé à exécuter le programme en cours.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Fonctions : mathématiques

min(x, y)	Calcule le minimum de deux nombres.
max(x, y)	Calcule le maximum de deux nombres.
abs(x)	Calcule la valeur absolue d'un nombre.
pow(base, exponent)	Calcule la valeur d'un nombre élevé à une puissance.
sqrt(x)	Calcule la racine carrée d'un nombre.
map(value, fromLow, fromHigh, toLow, toHigh)	Remappe un nombre d'une plage à une autre. Autrement dit, une valeur de fromLow serait mappée à toLow, une valeur de fromHigh à toHigh, des valeurs intermédiaires à des valeurs intermédiaires.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

Fonctions : trigonométrique

sin()	Calcule le sinus d'un angle (en radians). Le résultat sera compris entre -1 et 1.
cos()	Calcule le cos d'un angle (en radians). Le résultat sera compris entre -1 et 1.
tan()	Calcule la tangente d'un angle (en radians). Le résultat sera compris entre l'infini négatif et l'infini.

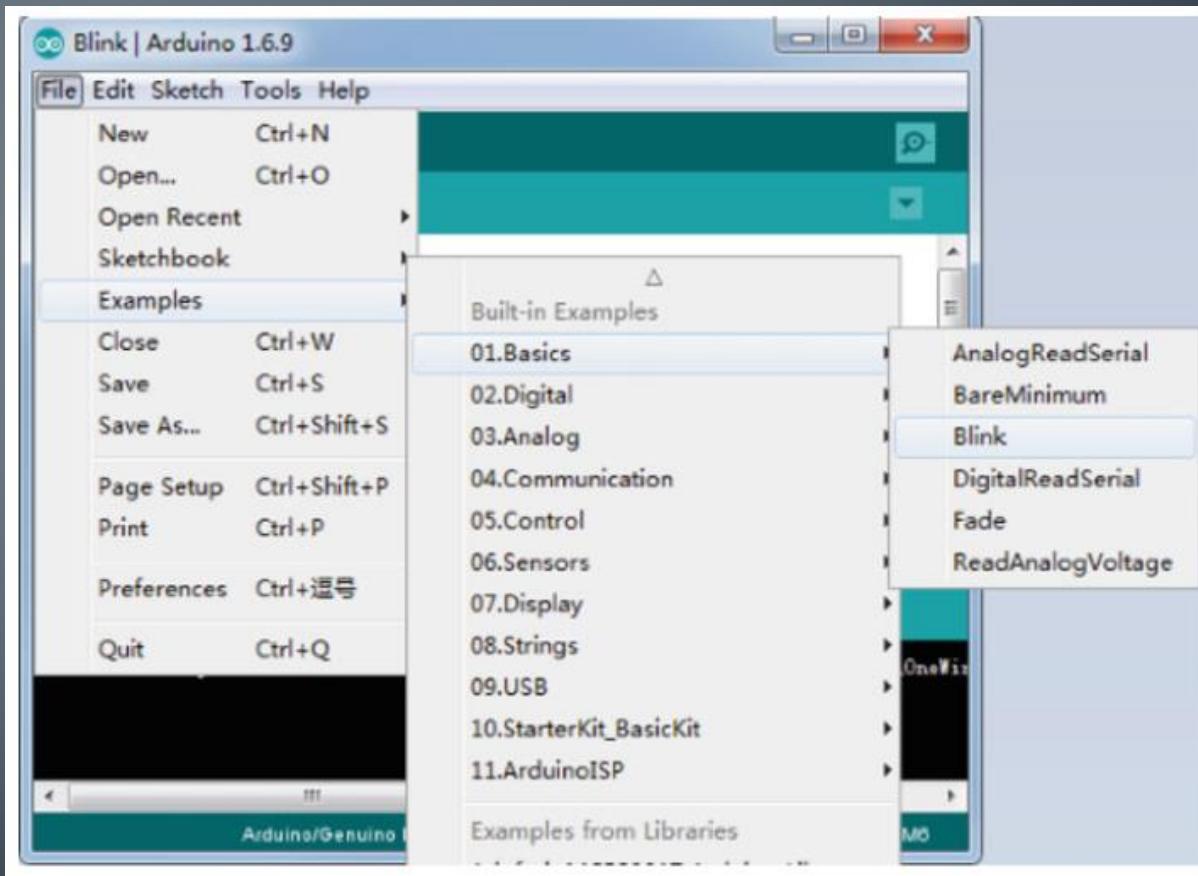
Fonctions : nombres aléatoires

randomSeed(seed)	Initialise le générateur de nombres pseudo-aléatoires, le faisant démarrer à un point arbitraire de sa séquence aléatoire.
random()	La fonction aléatoire génère des nombres pseudo-aléatoires.

Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

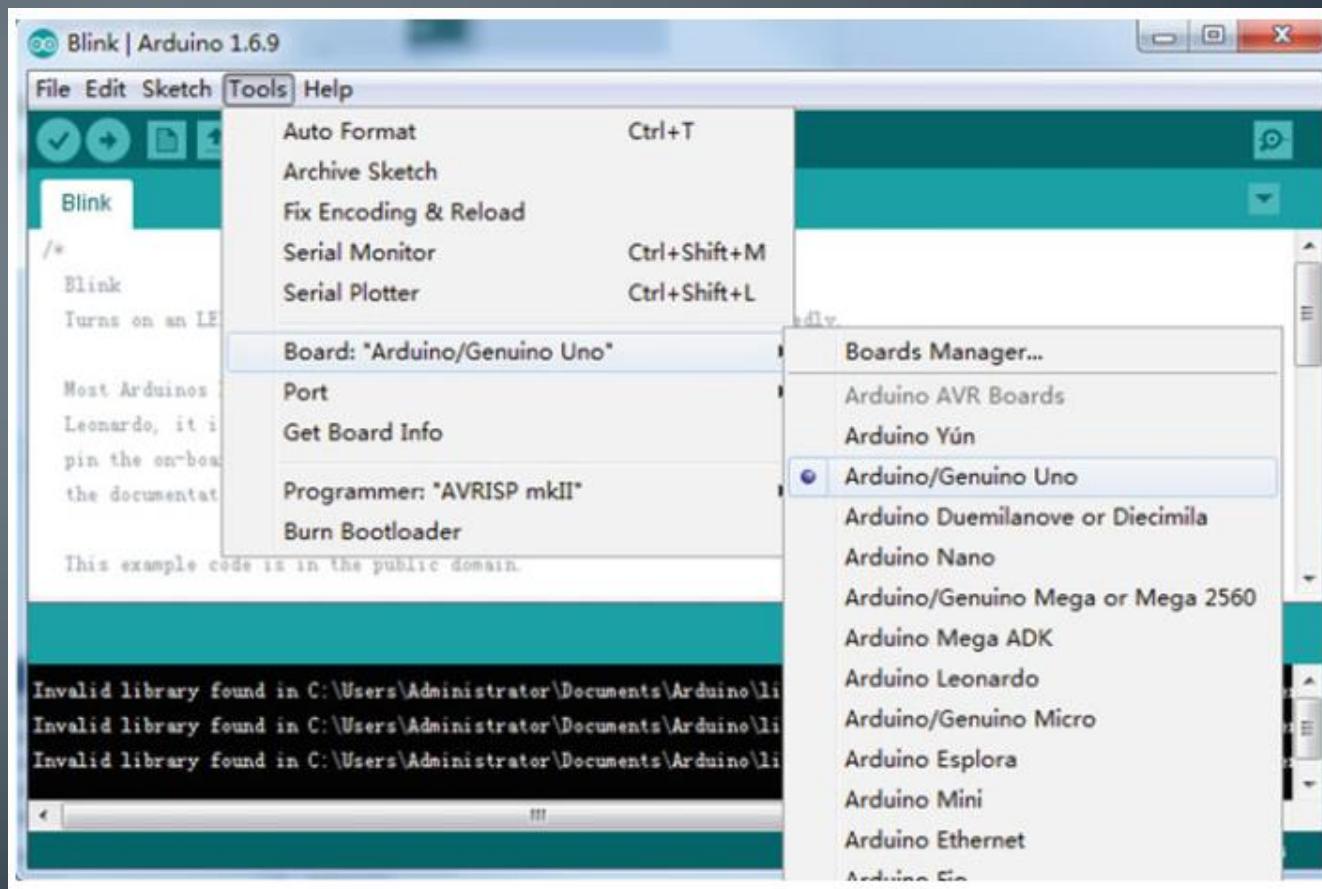
- **Exemple avec "Blink" dans IDE :** Modifiez l'exemple de Blink pour allumer/éteindre une led ;



Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

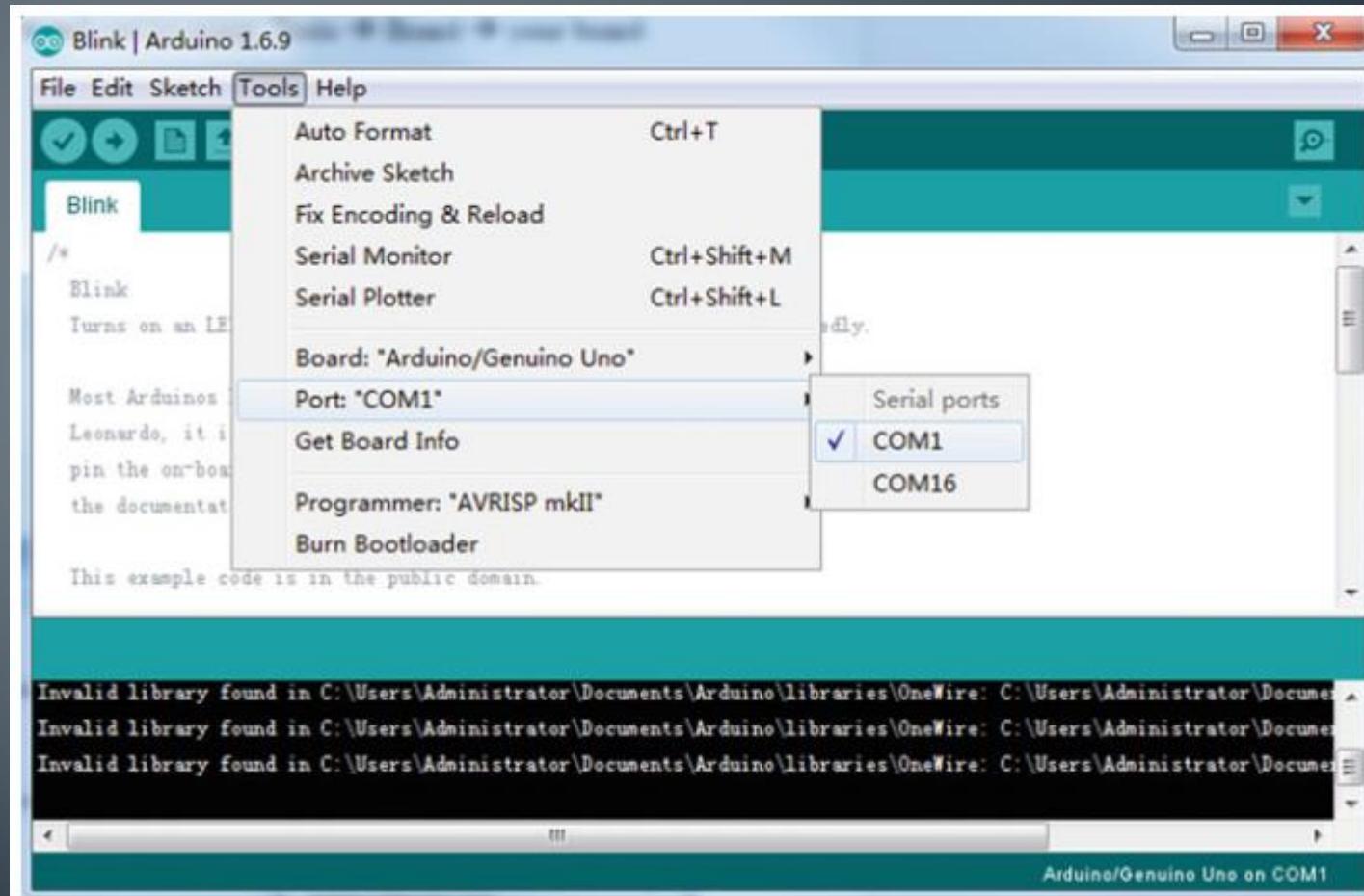
- Exemple avec "Blink" dans IDE : Sélection du type de carte Arduino dans l'IDE



Capteurs, actionneurs et programmation des microcontrôleurs

Capteurs, actionneurs et programmation des microcontrôleurs

- Exemple avec "Blink" dans IDE : Sélection du port série





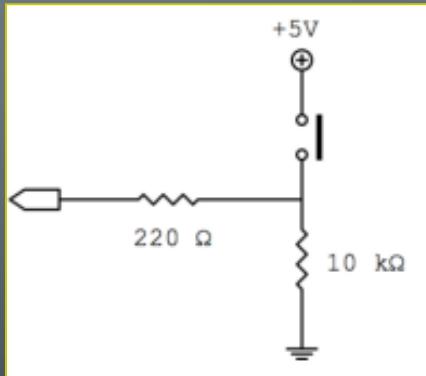
Etudes de cas: Arduino

Etude de cas: Arduino

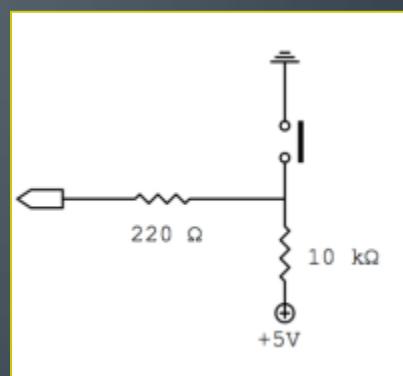
Bouton poussoir et Led

Mécaniquement, lorsqu'on appuie ou relâche un bouton-poussoir, il y a toujours des rebonds car le contact n'est jamais instantané ou ni parfait. Ces phénomènes sont des parasites qui peuvent tromper l'Arduino.

- Le signal n'est pas forcément clair: le bouton poussoir peut se comporter comme une antenne dans un environnement pollué par des ondes électromagnétiques et va donc générer un courant dans le circuit qui peut être interprété comme un signal (phénomène d'induction).
- Pour remédier à cela, il suffit d'utiliser des résistances pour effectuer des montages en Pull-Down ou en Pull-Up/.



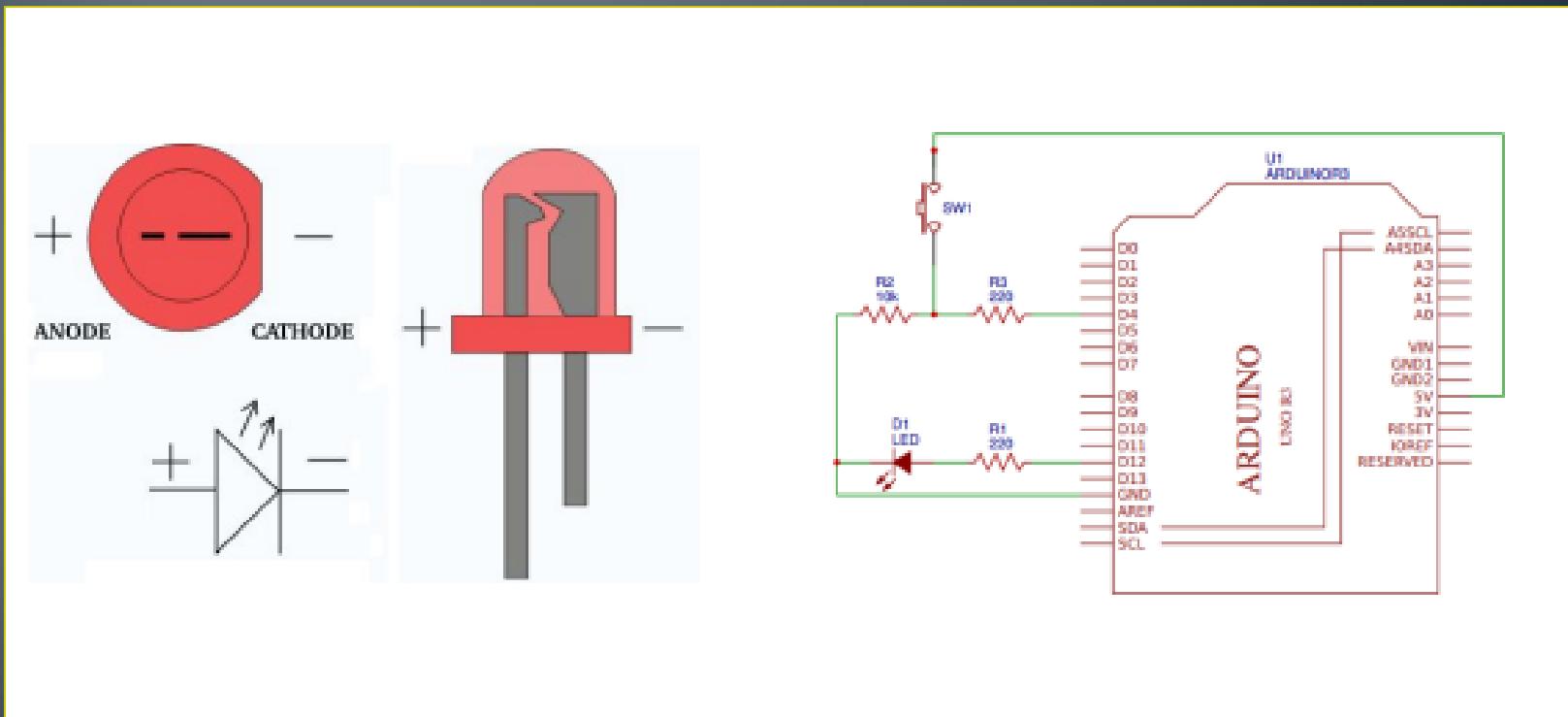
Résistance pull down par défaut



Résistance pull up par défaut

Etude de cas: Arduino

- Utiliser un bouton poussoir pour allumer/éteindre une led;

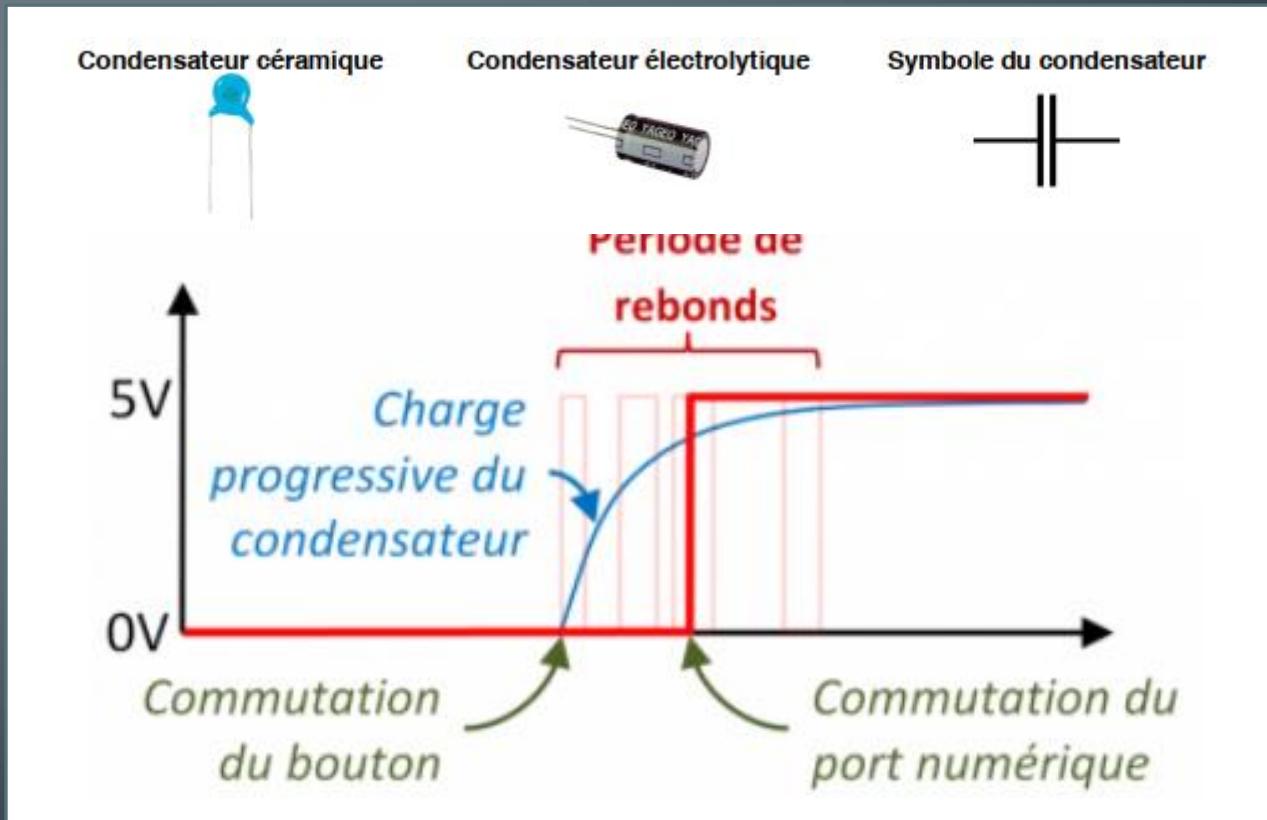


- Ajouter un Buzzer pour capter le clic sur le bouton

Etude de cas: Arduino

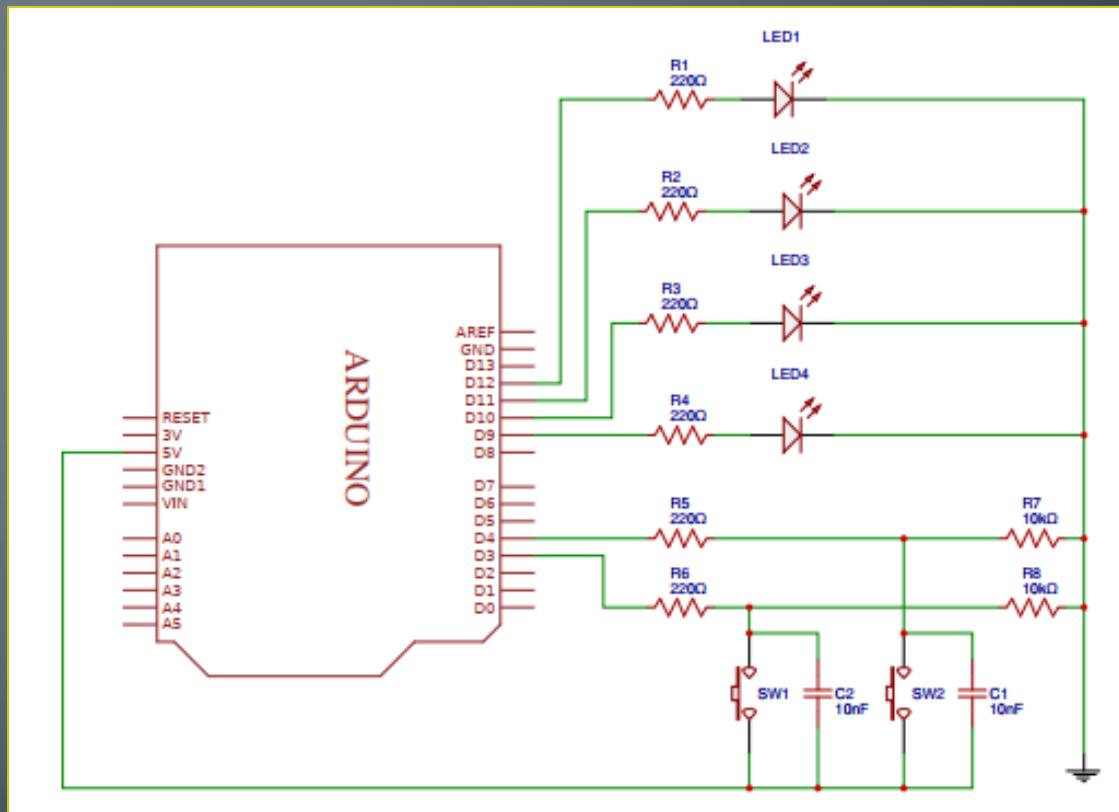
Utilisation d'un condensateur pour déparasiter

Constat : Malgré des pull-down, les boutons poussoirs sont peu précis. Il y a moyen de encore déparasiter et d'absorber ces rebonds en montant en parallèle du bouton-poussoir, un condensateur de faible capacité (ex : 10nF).



Etude de cas: Arduino

- Faire un bargraphe de 4 LED avec deux boutons poussoirs. L'un d'eux servira à augmenter la valeur du bargraphe, tandis que l'autre servira à la diminuer.

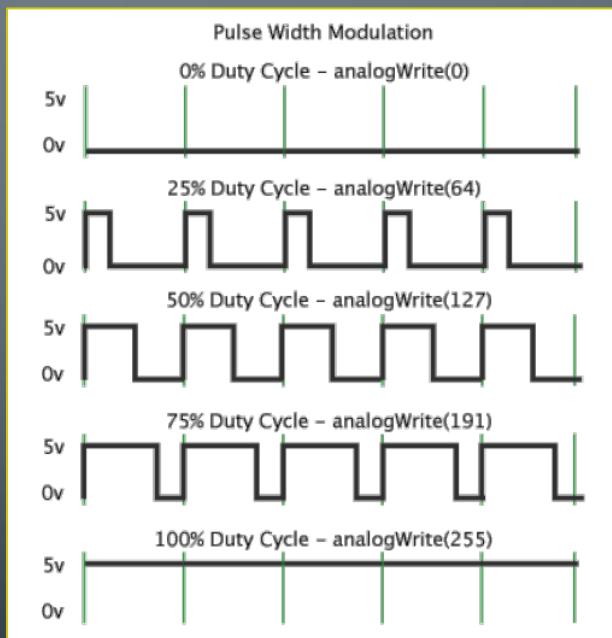


Etude de cas: Arduino

Utilisation de la modulation PWM

PWM pour Pulse Width Modulation (modulation de largeur d'impulsion en français) vous permet de faire varier les périodes haute (on) et basse (off) des broches à une fréquence élevée.

- **Exemple:** Pour un cycle de 25% en position haute et 75% en position basse, la LED sera moins lumineuse par rapport à un cycle de 50% sur 50% (voir quelques phases ci-dessous).

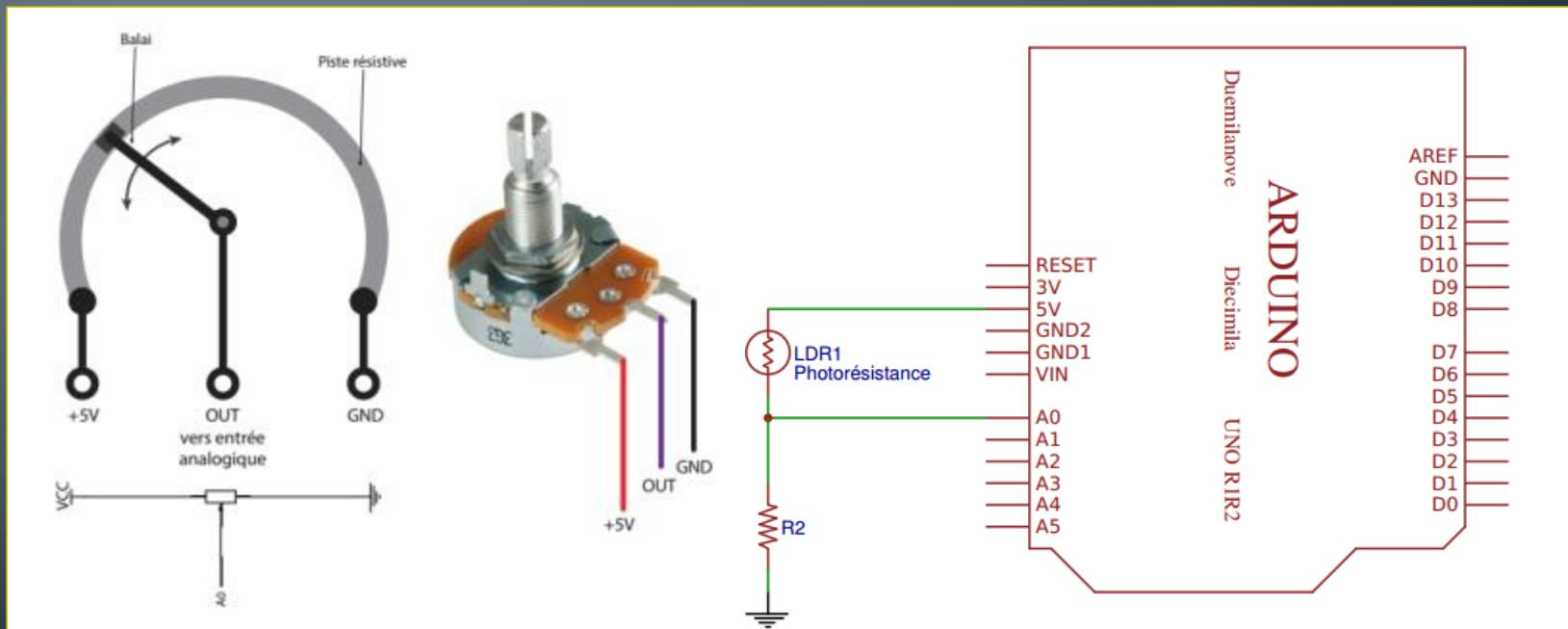


- Les broches capables de prendre en charge la fonction PWM sont identifiées par un (~) et pour l'utiliser, on utilise AnalogWrite au lieu de l'instruction digitalWrite.
- Utilisez un bouton pour faire varier la luminosité d'une LED en 5 clics:
 - Clique 1 : 25% en position haute ;
 - Clique 2 : 50% en position haute ;
 - Clique 3 : 75% en position haute ;
 - Clique 4 : 100% en position haute ;
 - Clique 5 : 0% en position haute.

Etude de cas: Arduino

Utilisation du potentiomètre

Un potentiomètre est une résistance variable. Il permet donc de modifier la tension d'un circuit. Il est analogique, ce qui veut dire qu'il peut être branché à broche analogique (A0 à A5) de l'Arduino. Ce qui veut dire qu'on peut récupérer les valeurs comprises entre 0 et 1023.

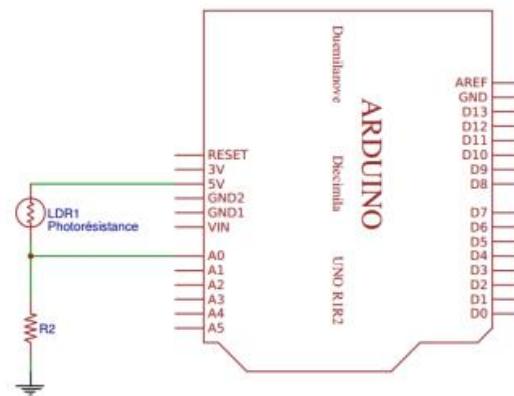
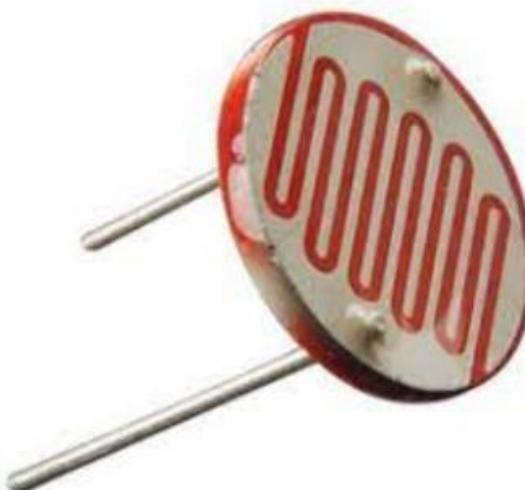


Etude de cas: Arduino

Utilisation d'un capteur analogique

Les capteurs analogiques produisent en sortie une grandeur physique (tension ou courant) dont la valeur est proportionnelle à la grandeur physique mesurée. Il délivre ainsi en sortie une infinité de valeurs continues. En Arduino, les capteurs analogiques sont lus via les ports analogiques (A0, A1, A2, A3, A4 et A5) grâce à la fonction **analogRead()** et les valeurs retournées sont comprises entre 0 et 1023. Ces valeurs sont les images des tensions mesurées, elles-mêmes comprises entre 0V et +5V.

- Utiliser une photorésistance pour mesurer la luminosité ambiante et afficher les valeurs sur un écran LCD ;

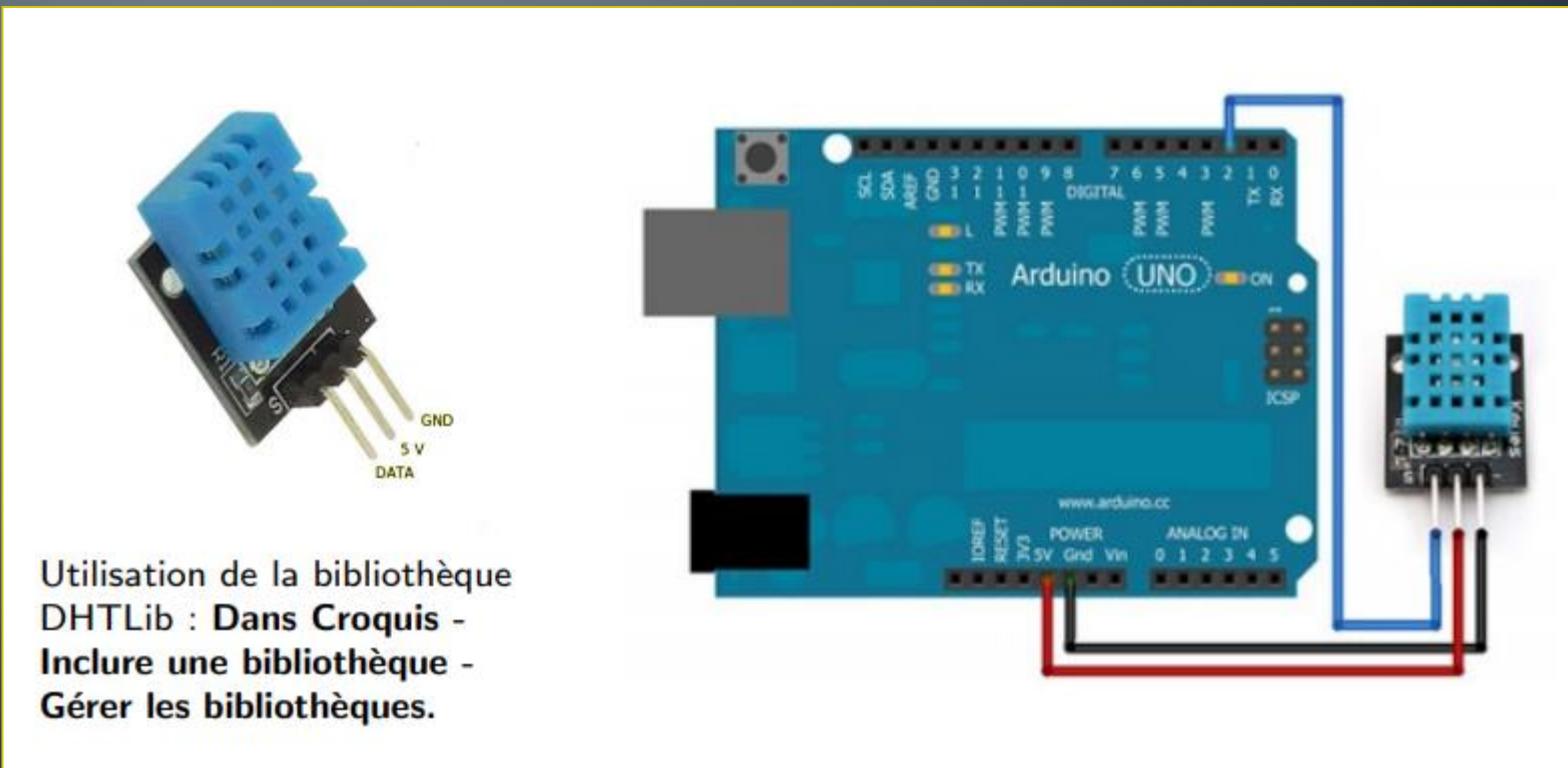


Etude de cas: Arduino

Utilisation d'un capteur numérique

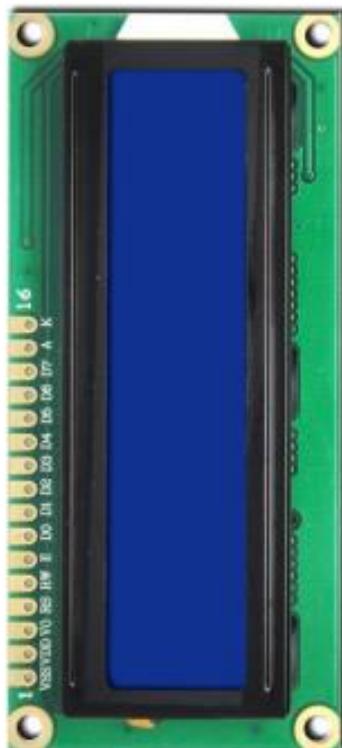
Les capteurs numériques produisent en sortie une séquence d'états logiques. Les capteurs numériques donnent en sortie une valeur finie (des valeurs discrètes). En Arduino, les capteurs numériques sont lus via les ports numériques (0, 1, 2, ..., 13) grâce à la fonction `digitalRead()`:

- Utiliser un DHT11 et afficher les valeurs de température et d'humidité sur le moniteur série;



Etude de cas: Arduino

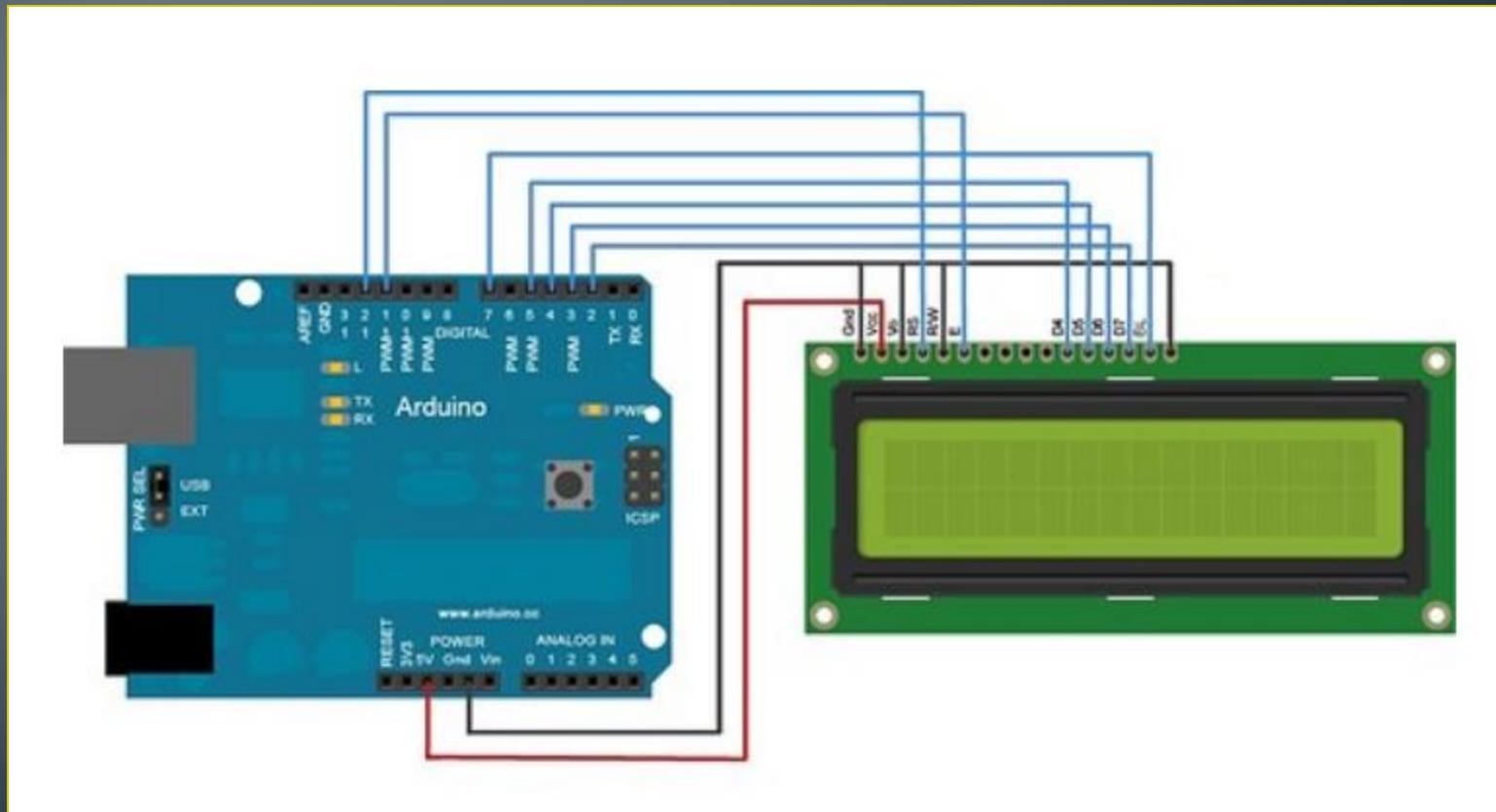
- Utiliser un écran LCD pour afficher les valeurs de température et d'humidité Collectées par le DHT11 ;



- K katode. borne – de la LED de rétroéclairage ;
- A anode. borne + de la LED de rétroéclairage ;
- D4, D5, D6 et D7 pour la transmission des données à afficher ;
- D0, D1, D2 et D3 sont reliés au ground (GND) ;
- E enable. active ou non l'affichage ;
- RW Read or Write. Toujours à la masse ;
- RS Register Select. Permet de sélectionner la zone mémoire ;
- V0 Broche de contraste. Connecté à une sortie PWM ou à un potentiomètre ;
- VDD Broche d'alimentation. Typiquement connectée à la broche 5V de l'Arduino ;
- VSS Relier à la masse de l'écran.

Etude de cas: Arduino

- Utiliser un écran LCD pour afficher les valeurs de température et d'humidité Collectées par le DHT11 ;



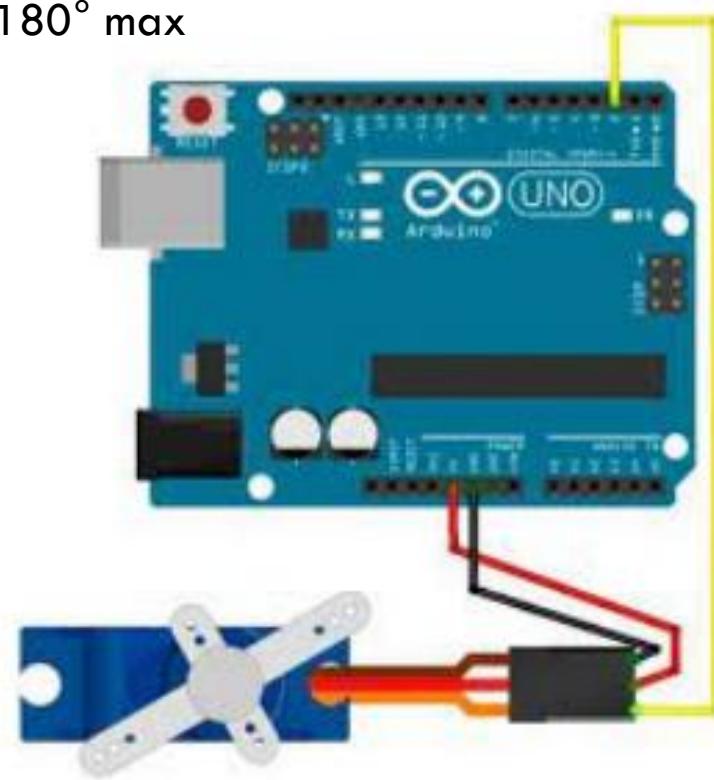
Etude de cas: Arduino

Utilisation d'un servomoteur

..
Rotation sur 180° max

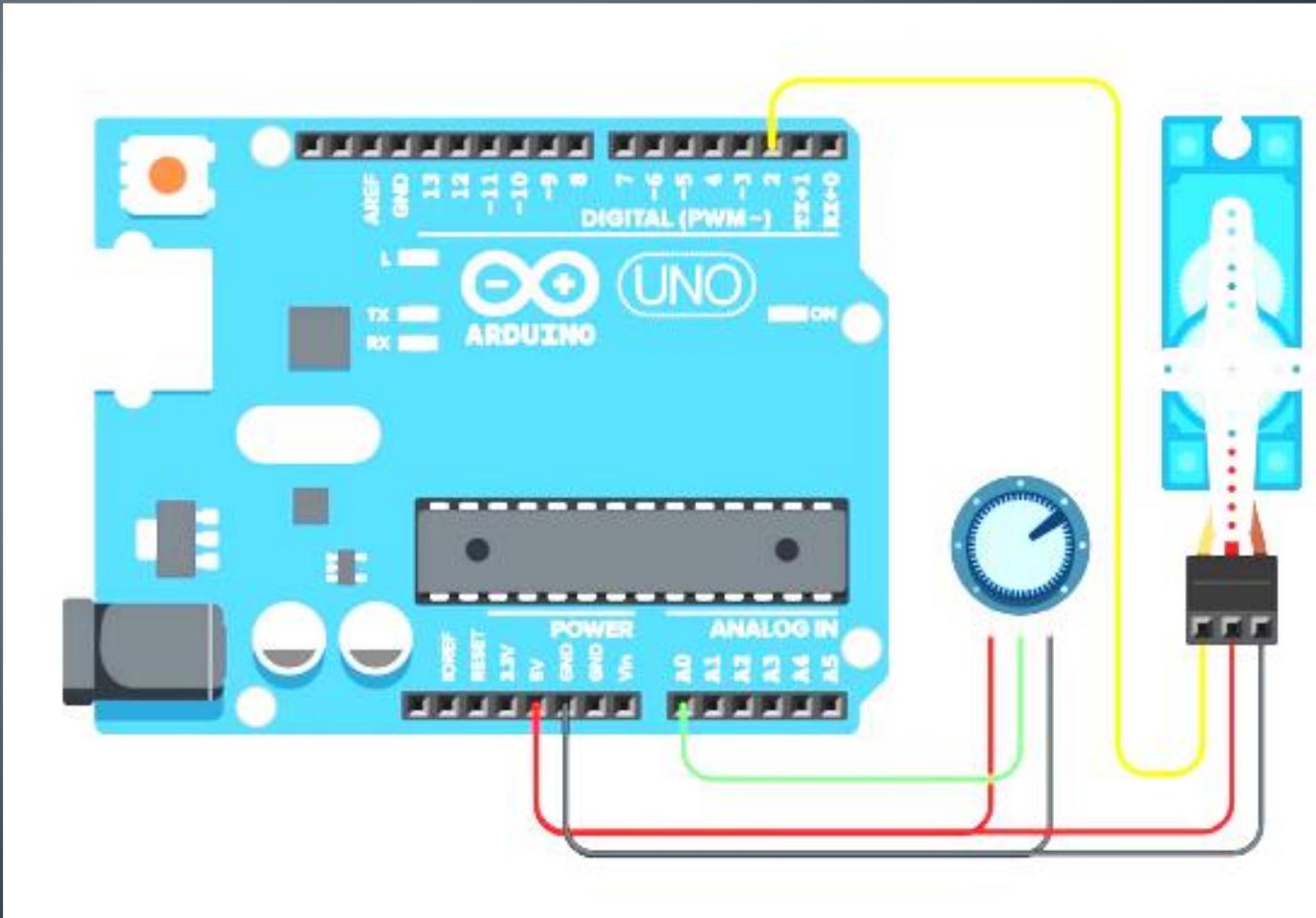


Inclure la bibliothèque Servo : Croquis -
Inclure une bibliothèque - Servo



Etude de cas: Arduino

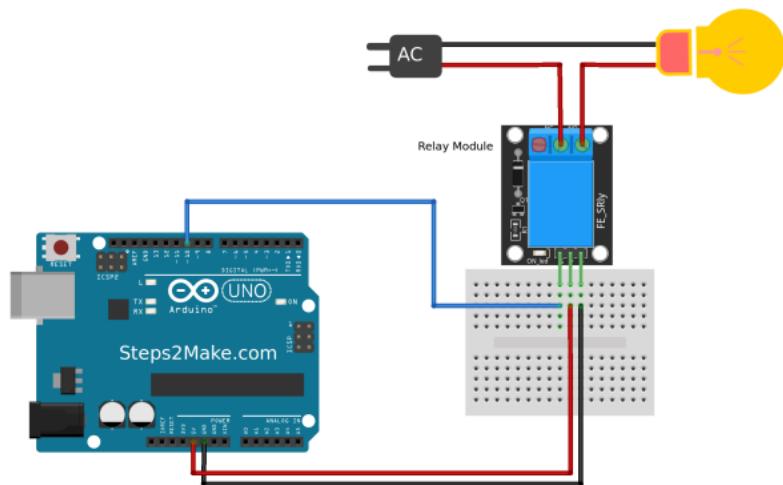
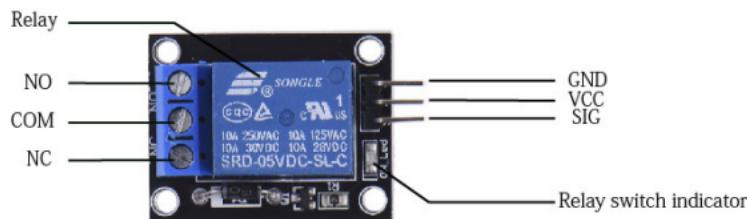
- Servo et potentio



Etude de cas: Arduino

Utilisation d'un relais (commutateur électrique) pour commander un second circuit ;

..



À suivre

Feedback sur:
pape.abdoulaye.barro@gmail.com

Programmation des microcontrôleurs

La programmation sous Raspberry Pi

La programmation sous Raspberry Pi

Généralités

- Dans ce chapitre, nous allons utiliser un nano ordinateur avec un écran hautement défini pour traiter les données collectées par des capteurs: Raspberry Pi .

Raspberry Pi est un nano-ordinateur, de la taille d'une carte de crédit, équipé d'un microprocesseur ARM, de la mémoire RAM, d'une carte vidéo, d'une carte Ethernet, du Wi-Fi et du Bluetooth, d'un connecteur USB-2 et d'un connecteur HDMI.

- Il a fait son apparition (en public) en 2012 avec comme objectif principal d'offrir aux étudiants et autres intéressés, un outil expansible et très accessible leur permettant d'apprendre plus efficacement l'informatique en général et la programmation en particulier.
- En 2006, les premiers prototypes sont développés sur des **microcontrôleurs Atmel ATmega 644**.

La programmation sous Raspberry Pi

Caractéristiques

- Les Raspis sont très attrayants grâce à leurs périphériques de bas niveau intégrés.
 - La carte comporte des broches d'entrée/sortie à usage général (GPIO), dont certaines prennent en charge les communications I2C, SPI et UART (de type RS-232);
 - De plus, un bus audio spécifique (**I2S**) est disponible, ainsi qu'une interface **CSI** haute vitesse pour connecter la caméra Pi sur mesure, et une interface **DSI** pour connecter les panneaux LCD;
 - Certaines de ces caractéristiques peuvent être utilisées pour mettre en œuvre la même fonctionnalité que sur l'**Arduino**;
 - Il peut être utilisé également comme système informatique hôte standardisé (**Unité centrale**).

La programmation sous Raspberry Pi

Installation et configuration

- Raspberry Pi est livrée sans système d'exploitation. Il fonctionne avec plusieurs variantes de Linux, notamment, Debian et autres logiciels compatibles.
- Par défaut, il utilise Raspbian est un système d'exploitation optimisé et gratuit basé sur Debian. Il existe plusieurs versions de raspbian:
 - ❑ Wheezy du Debian 7;
 - ❑ Jessie du Debian 8;
 - ❑ Stretch du Debian 9;
 - ❑ Buster du Debian 10

La programmation sous Raspberry Pi

Installation et configuration

Installation

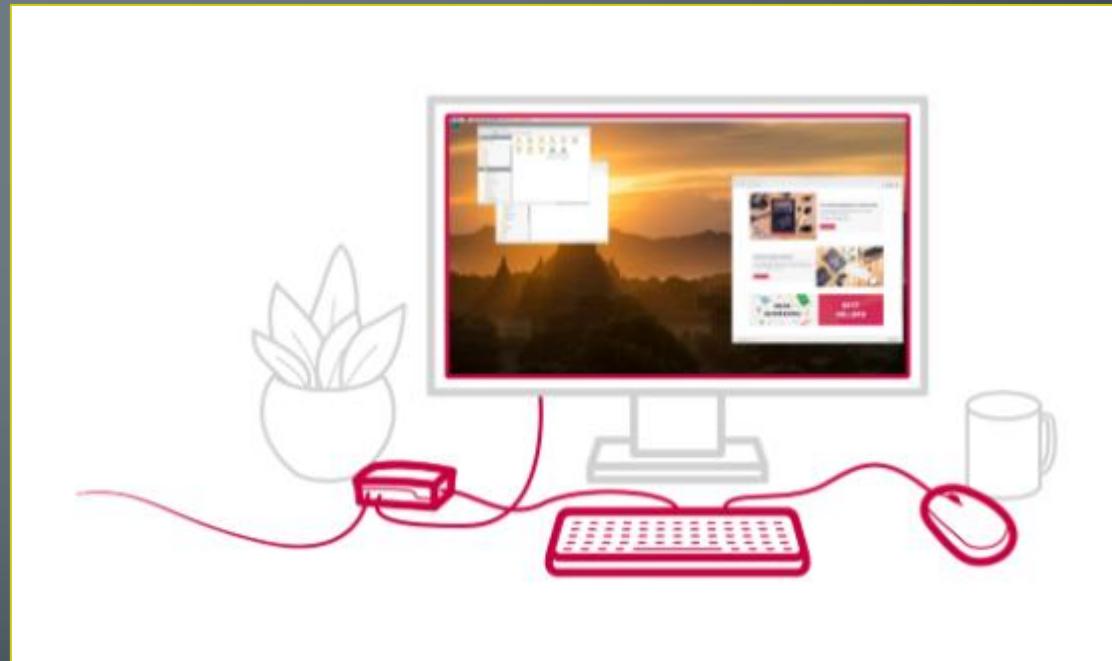
- Avant de procéder à l'installation du Raspberry Pi, il faut disposer, d'une SD carte d'au moins 16Go, d'une image de raspbian et d'un installateur/Extracteur (logiciel) d'image.
 - Pour Raspbian, une archive (.zip) est disponible sur:
<https://www.raspberrypi.com/software/operating-systems/>
 - Pour le logiciel, il s'agit de Win32DiskImager (il en existe d'autre) accessible sur: <https://sourceforge.net/projects/win32diskimager/>
 - Il est également possible d'utiliser un installateur qui se chargera d'extraire une image et puis de l'installer sur la carte. Il est disponible sur le site:
<https://www.raspberrypi.com/software/>
- Sur un ordinateur, charger Raspbian sur la carte SD, éjecter et insérer la à son emplacement sur le Raspberry et puis brancher la carte.

La programmation sous Raspberry Pi

Installation et configuration

Configuration

- Au démarrage, vous devez renseigner le login (*pi*) et le mot de passe (*raspberry*). À noté que le clavier est en **QWERTY**.



La programmation sous Raspberry Pi

Installation et configuration

Configuration

- Il existe deux manière d'accéder au menu de configuration du raspberry Pi:
 - Soit en ligne de commande en tapant au console: `sudo raspi-config`
 - Ou directement dans le **Menu principal**, puis **Préférences**, ensuite **Configuration du Raspberry Pi**.
- Vous pouvez configurer:
 - Le clavier
 - La localisation
 - Le fuseau horaire
 - La prise en main à distance via SSH
 - L'utilisation des ports GPIO
 - La Caméra
 - ...
- Si vous désirez changer le mot de passe taper sur le terminal: `sudo passwd`
- Redémarrer ensuite la carte en tapant: `sudo reboot`

La programmation sous Raspberry Pi

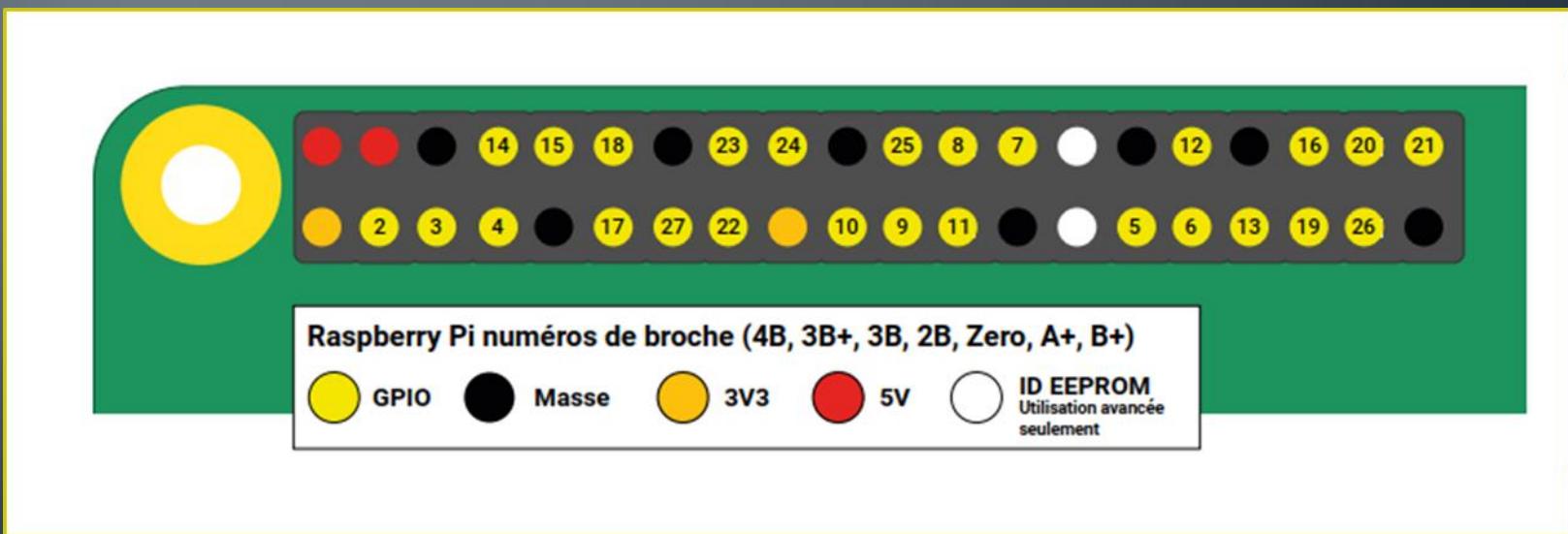
GPIO avec Python

- le port GPIO (*general-purpose input/output*) vous permet de connecter des périphériques comme des LED et des interrupteurs au Raspberry Pi et de pouvoir les commander via un programme que vous avez créé. Les broches peuvent être utilisées aussi bien en entrée qu'en sortie.
- Pour utiliser les ports GPIO, il faut installer le module *RPi.GPIO* à l'aide de la ligne de commande :
 - `sudo apt-get install RPi.GPIO`
- Pour les utiliser avec un programme Python, il faut installer la bibliothèque *Python-GPIO*:
 - `sudo apt-get update`
 - `sudo apt-get install python-dev`
 - `sudo apt-get install python-rpi.gpio`

La programmation sous Raspberry Pi

GPIO avec Python

- Le connecteur GPIO est constitué de 40 broches mâles dont certaines sont destinées aux projets d'**informatique physique**, d'autres sont consacrées à l'alimentation, d'autres encore sont réservées pour la communication. Ci-dessous, une image du connecteur.



La programmation sous Raspberry Pi

GPIO avec Python

Numérotation des ports GPIO

- La bibliothèque *RPi.GPIO* supporte différentes méthodes pour accéder aux ports, nous en distinguons deux(2):
 - le mode *BCM* (*en rouge*) qui correspond à la numérotation électronique de la puce;
 - le mode alternatif *BOARD* (*en noire*) qui correspond à celle de la sérigraphie du connecteur de la carte.



La programmation sous Raspberry Pi

GPIO avec Python

- Pour pouvoir utiliser les GPIO avec Python, il faut utiliser un IDE de programmation python sur Raspbian (nous utilisons *Thonny*).
- On peut parcourir quelques notions de base en Python sans entré dans les détails (puisse qu'on s'adresse à un public déjà informé). Nous allons voir les **types**, les **variables**, les **conditions**, les **boucles**, les **fonctions** et les **bibliothèques**.
 - Un **type** caractérise le contenu d'une variable. Il peut s'agir d'un **chiffre**, d'un **caractère**, d'un **texte**, d'une valeur de **type vrai ou faux**, d'un **tableau de valeurs**, etc.
 - Python est un langage à **typage dynamique**,
 - ce qui veut dire qu'il n'est pas nécessaire de déclarer les variables avant de pouvoir leur affecter une valeur.
 - Le type de données peut aussi changer en cours de l'exécution du programme.
 - La fonction **type()** permet de connaître le type de la valeur d'une variable. Par exemple, si `a=10`, alors **type(a)** donnera `int`.
 - Les types sont: `int` pour les nombres entiers, `float` pour les nombres à virgules flottante, `str` pour les chaînes de caractères, `bool` pour les booléens, `list` pour une collection d'éléments séparés par des virgules, `complex` pour les nombres complexes, etc. .
 - À partir des types de base, il est possible d'en élaborer de nouveaux comme :
 - Le **tuple** qui est une collection ordonnée de plusieurs éléments. Par exemple `a=(3, 7, 10);`
 - Le **dictionnaire** qui est un rassemblement d'éléments identifiables par une clé. Par exemple `d={"x": 4, "y": 2};`

La programmation sous Raspberry Pi

GPIO avec Python

- Une **variable** est une zone mémoire dans laquelle une valeur est stockée. En Python, la **déclaration** d'une variable et son **initialisation** se font en même temps. Par exemple, `a=2`. Il faut cependant respecter les règles usuelles suivantes:
 - Le nom doit commencer par une lettre ou par un underscore ;
 - Le nom d'une variable ne doit contenir que des caractères alphanumériques courants;
 - On ne peut pas utiliser certains mots réservés.
- On utilise une **structure conditionnelle** pour faire exécuter un bout de code si certaines conditions sont remplies. Les structures conditionnelles en Python sont:
 - La condition **if** ("si") ;
 - La condition **if...else** ("si...sinon") ;
 - La condition **if...elif...else** ("si...sinon si... sinon") .Ici, les opérateurs de comparaisons et les opérateurs logiques (`and`, `or`, `not`) sont largement utilisés.
- Une **boucle** est utilisée pour exécuter en plusieurs fois un bloc d'instructions tant qu'une condition donnée est vérifiée. Nous avons accès à deux boucles en Python :
 - La boucle **while** ("tant que...") dont le nombre d'itération n'est pas connu à l'avance;
 - La boucle **for** ("pour...") dont le nombre d'itération est bien connu. Avec la boucle `for`, on peut utiliser la fonction **range()** pour définir une plage de valeurs. Exemple:
 - `range(5)` permet de générer les valeurs 0, 1, 2, 3 et 4;
 - `range(5, 10)` permet de générer les nombres 5, 6, 7, 8 et 9;
 - `range(6, 10, 2)` permet de générer les nombres entre 6 et 10 par pas de 2 (6, 8 et 10);
 - L'instruction **break** permet de stopper l'exécution d'une boucle lorsqu'une certaine condition est vérifiée.
 - L'instruction **continue** permet elle d'ignorer l'itération actuelle de la boucle et de passer directement à l'itération suivante.

La programmation sous Raspberry Pi

GPIO avec Python

- Lorsqu'on a à ré-écrire plusieurs fois un bout de code dans un programme, avec éventuellement des changements de données, il est préférable de le regrouper sous un nom unique appelé **fonction**. Pour déclarer une fonction en Python, la syntaxe est la suivante:

```
def nom_fonction(<paramètres>):  
    instructions
```

Il existe des fonctions prédéfinies en Python. Exemple `print()`, `input()`, etc. .

- Une bibliothèque est un ensemble de modules (classes, fonctions,...) ajoutant des possibilités étendues à Python. Pour utiliser les fonctionnalités d'une bibliothèque, il va falloir l'inclure dans le programme en question avec le mot clé **import** comme suit:

```
import nom_de_la_bibliothèque
```

La programmation sous Raspberry Pi

GPIO avec Python

- L'utilisation des GPIOs nécessite l'inclusion de la bibliothèque dans le fichier (d'extension `.py`) en créant un objet `GPIO` comme suit:
 - `import RPi.GPIO as GPIO`
- On peut également importer la bibliothèque `time` propre à python, dans laquelle est contenue la fonction `sleep` avec laquelle vous pouvez définir facilement des temps d'attente dans un programme, comme suit:
 - `import time`
 - `time.sleep(5) //attente de 5s`

La programmation sous Raspberry Pi

GPIO avec Python

- Il va falloir définir le mode de numérisation également:
 - `GPIO.setmode(GPIO.BOARD) // mode board`
 - `GPIO.setmode(GPIO.BCM) // mode bcm`
- Pour déclarer et initialiser une entrées-sorties (E/S), il suffit de préciser son numéro, son mode d'utilisation (entrée ou sortie) et éventuellement son état initial (*mode sortie uniquement*) comme suit:
 - `GPIO.setup(6, GPIO.IN) // broche 6 comme entrée numérique`
 - `GPIO.setup(8, GPIO.OUT) // broche 8 comme sortie numérique`
 - `GPIO.setup(8, GPIO.OUT, initial=GPIO.HIGH) // broche 8 est une sortie initialisée à l'état haut.`
- L'état des E/S, le module `RPi.GPIO` accepte des variables dédiées (`GPIO.HIGH` ou `GPIO.LOW`), des entiers (1 ou 0) ou des booléens (`True` ou `False`).

La programmation sous Raspberry Pi

GPIO avec Python

- Pour changer l'état d'une sortie numérique, on procède comme suit:
 - `GPIO.output(8, GPIO.LOW)` // la broche 8 est mise à BAS
- Il est possible de connaître l'état d'une entrée en utilisant la fonction `input`.
 - `GPIO.input(6)` // on interroge la broche 6 pour connaître son état
- Pour connaître la configuration d'une entrée/sortie numérique, on utilise la fonction `gpio_funtion`. Les valeurs renvoyées sont alors `GPIO.INPUT`, `GPIO.OUTPUT`, `GPIO.SPI`, `GPIO.I2C`, `GPIO.HARD_PWM`, `GPIO.SERIAL` ou `GPIO.UNKNOWN`
 - `state=GPIO.gpio_funtion(8)`
 - `print(state)` // affiché dans la console la configuration de la broche
- A la fin du programme, il est conseillé d'effectuer une purge des ressources en utilisant la fonction `cleanup`.
 - `GPIO.cleanup()` // libère toutes les entrées/sortie utilisées

La programmation sous Raspberry Pi

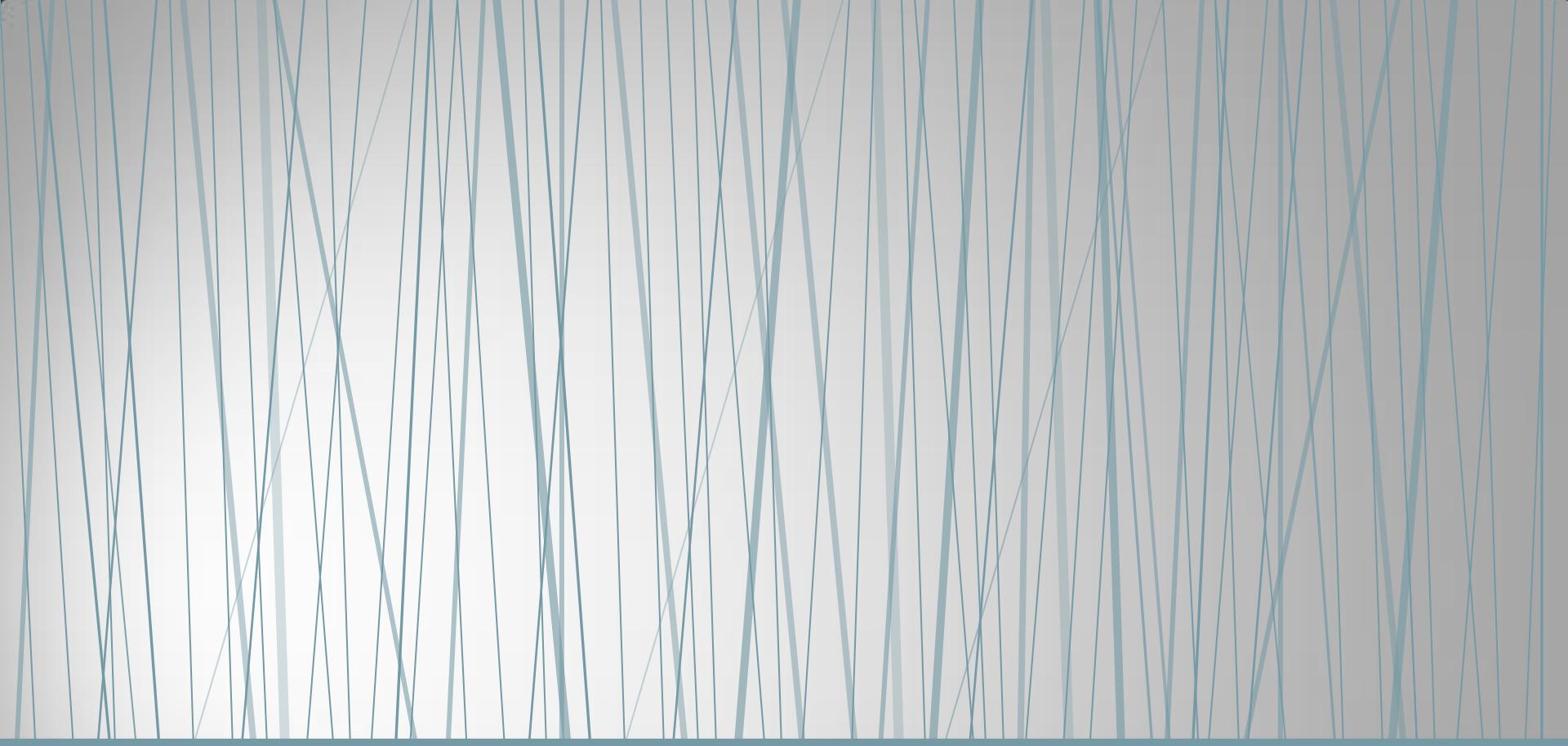
GPIO avec Python: exemple de Blink

- import RPi.GPIO as GPIO
- import time
- GPIO.setwarnings(False)
- GPIO.setmode(GPIO.BCM)
- GPIO.setup(21, GPIO.OUT)
- nbreRepetition = input("donner le nombre de fois la LED doit clignoter?\n")
- i=0
- while i < nbreRepetition :
 - GPIO.output(21, True)
 - time.sleep(1)
 - GPIO.output(21, False)
 - time.sleep(1)
 - i = i+1
- GPIO.cleanup()

La programmation sous Raspberry Pi

Lancement du programme au démarrage

- En informatique physique, on aura besoin d'exécuter un programme au démarrage de la carte. Or sur Raspberry pi, ce mécanisme n'est pas directement pris en compte. Ce sera donc au programmeur de le gérer.
- La méthode la plus simple de lancer un programme au démarrage de la Raspberry Pi est d'utiliser le fichier `rc.local` qui se trouve dans `/etc`.
 - Ce script est sensé contenir des lignes de commandes qui seront exécutées juste avant que la Raspberry Pi n'ait fini de booter. Le script se termine par `exit 0`, indiquant la fin des exécutions. Il va falloir donc ajouter votre programme juste avant.
 - Si, par exemple, votre programme est nommé `test.py` et se trouvant dans `/home/pi`, alors on écrit `/usr/bin/python3 /home/pi/test.py`.
 - Il va falloir faire très attention car votre programme doit rendre la main au script pour que le Raspberry Pi puisse finir de booter. Votre programme doit de ce fait terminer son exécution ou bien s'il doit tourner en boucle infinie, on doit alors le lancer en tâche de fond en ajoutant un `&` après la commande comme suit: `/usr/bin/python3 /home/pi/test.py &`.

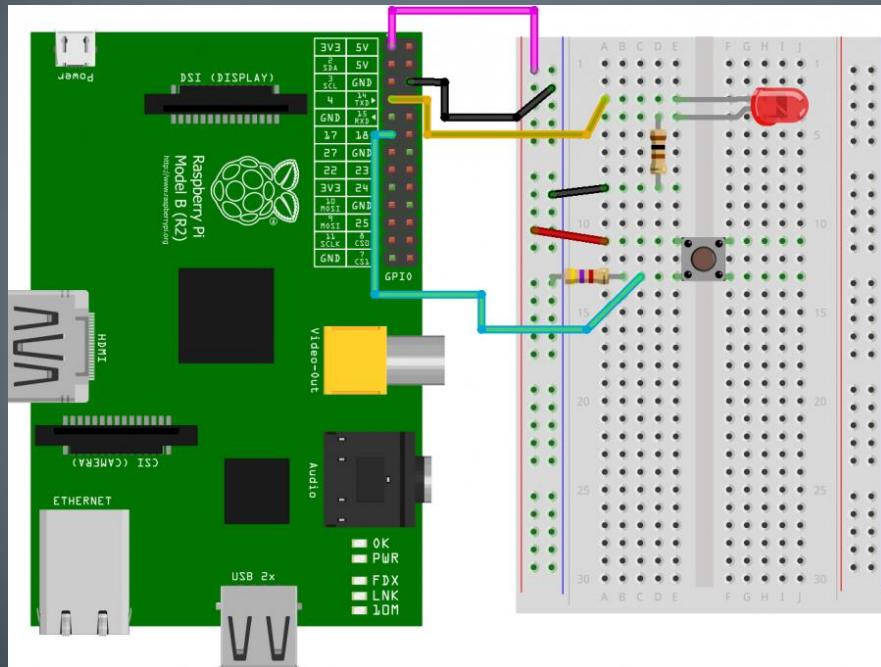


Etudes de cas: GPIO

Etude de cas: GPIO

Bouton poussoir et Led

Utiliser des résistances et un bouton poussoir en mode Pull-down pour allumer et éteindre une LED.



```
import time  
from RPi import GPIO  
  
GPIO.setwarnings(False)  
GPIO.setmode(GPIO.BCM)  
  
GPIO.setup(21, GPIO.OUT)  
GPIO.setup(7, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)  
  
while True:  
    val = GPIO.input(7)  
  
    if val :  
        GPIO.output(21, GPIO.HIGH)  
    else:  
        GPIO.output(21, GPIO.LOW)  
  
    time.sleep(1)
```

Etude de cas: GPIO

Utilisation d'un capteur numérique: le DHT11

Il existe une bibliothèque python nommée Adafruit DHT, permettant la manipulation des capteurs de type DHT. Nous l'utilisons, ici, pour manipuler le DHT11. On peut aussi tenter d'écrire nos propres codes, ce qui peut-être nous prendra un peu de temps.

Nous allons donc commencer à installé installez quelques bibliothèques Python:

```
sudo apt-get install build-essential python-dev
```

- Si GIT n'existe pas, il va falloir l'installer en tapant:

```
sudo apt-get install git-core
```

- S'il signale des erreurs, il va falloir effectuer une mise à jour de la liste des paquets:

```
sudo apt-get update
```

- Clonez ensuite la librairie Adafruit depuis leur dépôt :

```
git clone https://github.com/adafruit/Adafruit\_Python\_DHT.git
```

- Accéder à la répertoire Adafruit_Python_DHT:

```
cd Adafruit_Python_DHT
```

- Installez ensuite la bibliothèque

```
sudo python setup.py install
```

- Si tout s'était bien passé, on doit pouvoir lire maintenant la température et l'humidité. La façon la plus simple est d'utiliser d'abord les fichiers de démonstration directement à travers un script test situé dans exemples:

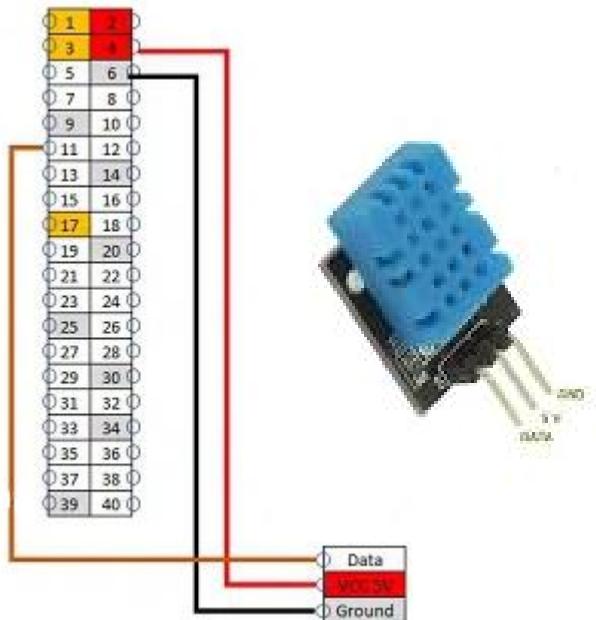
```
cd exemples
```

```
sudo ./AdafruitDHT.py 11 17
```

Avec 11 représentant le type de capteur(on pouvait mettre 22 s'il s'agit du DHT22), et 17 c'est le numéro du GPIO (mode BCM).

Etude de cas: GPIO

Après avoir installer la bibliothèque Adafruit et tester son fonctionnement, nous pouvons maintenant écrire notre scripts.



```
import time
Import Adafruit_DHT

sensorType=Adafruit_DHT.DHT11
gpioPin=17 // correspondant au broche 11 en mode BOARD

while True:

    hum, temp = Adafruit_DHT.read_retry(sensorType, gpioPin)

    if (hum is not None and temp is not None):
        print('Temp={0:0.1f}°C Hum={1:0.1f}%'.format(temp, hum))
    else:
        print('Erreur de lecture. Encore une fois!')

    time.sleep(1)
```

Etude de cas: GPIO

utilisation d'un écran LCD

Pour pouvoir utiliser un écran LCD, il va falloir installer la bibliothèque RPLCD.

- Pour installer la bibliothèque RPLCD, nous devons d'abord installer le Python Package Index, ou PIP (si ce n'est pas encore installé), en tapant:

```
sudo apt-get install python-pip
```

- Maintenant que tout est OK, installons RPLCD:

```
sudo pip install RPLCD
```

On est donc prêt pour la manipulation

```
import time
import Adafruit_DHT
from RPLCD.gpio import CharLCD

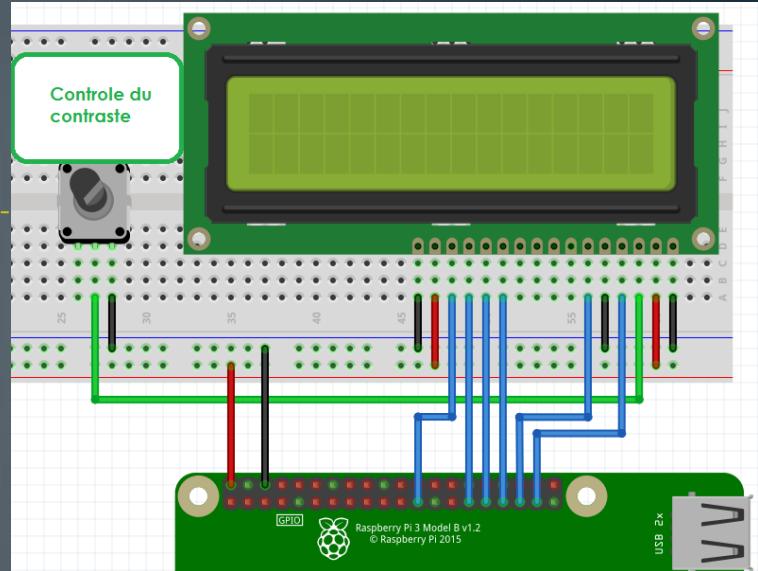
sensorType=Adafruit_DHT.DHT11
gpioBCMPin=17 // correspondant au broche 11 en mode BOARD
lcd = CharLCD(cols=16, rows=2, pin_rs=37, pin_e=35, pins_data=[33, 31, 29, 23])

while True:

    hum, temp = Adafruit_DHT.read_retry(sensorType, gpioBCMPin)

    if (hum is not None and temp is not None):
        lcd.cursor_pos = (0, 0)
        tempString= "Temp:" + str(temp) + " °C"
        lcd.write_string(tempString)
        lcd.cursor_pos = (1, 0)
        humString= "Hum:" + str(hum) + "%"
        lcd.write_string(humString)
    else:
        print('Erreur de lecture. Encore une fois!')

    time.sleep(1)
```

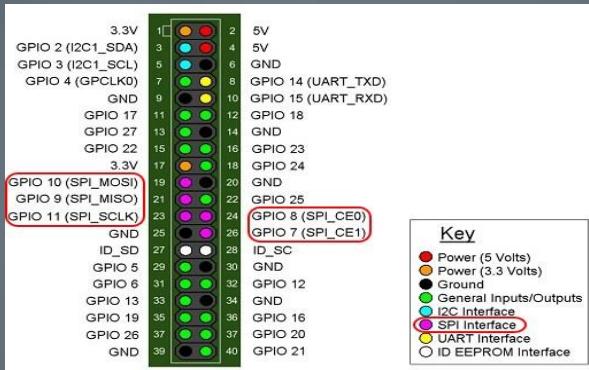


Etude de cas: GPIO

Signaux analogique sur Raspberry Pi

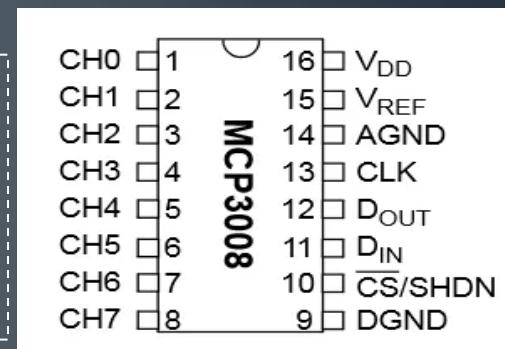
Le Raspberry Pi ne possède pas de port analogique. Pour cela, il va falloir utiliser un convertisseur analogique-numérique (ADC). Si on utilisait le MCP3008 par exemple, on peut disposer jusqu'à 8 entrées analogiques. Le module est connecté au Raspberry Pi via le bus SPI (à activé aussi).

- Connexion entre Raspberry Pi et MCP3008



RPI

Pin 1 (3.3V)	Pin 16 (VDD)
Pin 1 (3.3V)	Pin 15 (VREF)
Pin 6 (GND)	Pin 14 (AGND)
Pin 23 (SCLK)	Pin 13 (CLK)
Pin 21 (MISO)	Pin 12 (DOUT)
Pin 19 (MOSI)	Pin 11 (DIN)
Pin 24 (CE0)	Pin 10 (CS/SHDN)
Pin 6 (GND)	Pin 9 (DGND)



MCP3008

- Commencerons par mettre à jour les paquets et installer les outils Python Developer:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install python-dev
```

- Par la suite, nous allons télécharger, décompresser et installer la bibliothèque SpiDev :

```
wget https://github.com/doceme/py-spidev/archive/master.zip
```

```
unzip master.zip
```

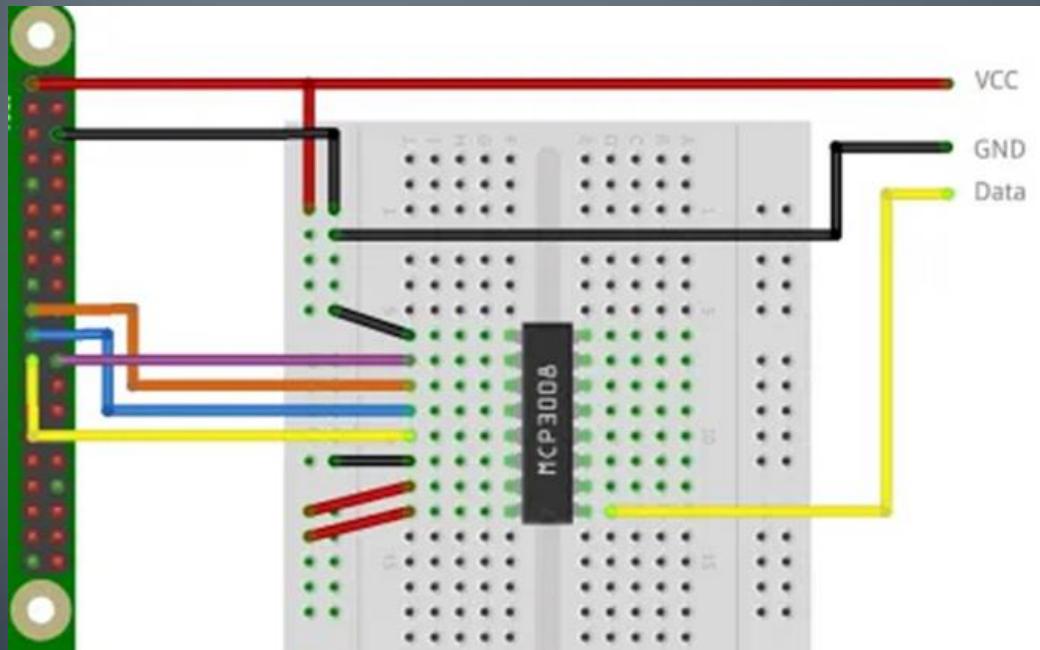
```
cd py-spidev-master
```

```
sudo python setup.py install
```

Etude de cas: GPIO

Signaux analogique sur Raspberry PI

- En utilisant le fichier MCP3008.py , on peut facilement manipuler le module. Il est directement accessible sur: <http://tutorials-raspberrypi.de/wp-content/uploads/scripts/MCP3008.py>.



```
from MCP3008 import MCP3008  
  
convertor= MCP3008()  
data= convertor.read( channel = 0 )  
print(" Tension appliquée : %.2f " % data )
```

- Utiliser un potentiomètre en guise d'exemple

Programmation des microcontrôleurs

La programmation sous Raspberry Pi: la Caméra

La programmation sous Raspberry Pi

La caméra

- Vous pouvez utiliser le module caméra de Raspberry Pi pour mettre en place un objet interactif tel qu'un robot ou encore un instrument de l'Intelligence artificielle.
- Le module est un petit circuit imprimé doté d'une limeade qui se connecte au port CSI (Caméra Serial Interface) du Raspi et fournit des images fixes haute résolution et des signaux vidéo animés qui peuvent être utilisés directement ou intégrés dans un programme.
- Les **Modules Camera standard** de Raspberry Pi sont basés sur un capteur d'images Sony IMX219 de **8 mégapixels** pouvant prendre des photos comportant jusqu'à 8 millions de pixels (jusqu'à 3 280 pixels de large sur 2 464 de haut) et pouvant aussi capturer des séquences vidéo en résolution **Full HD** à une cadence de 30 images par seconde (**30 ips**).
- La caméra peut être réglée pour capturer des images à une cadence plus élevée tout en diminuant la résolution: 60 ips pour les vidéos en 720p, et jusqu'à 90 ips pour les vidéos en 480p (VGA).

La programmation sous Raspberry Pi

Installation et configuration de la caméra

- Avant de pouvoir utiliser la caméra, vous devez indiquer au Raspberry Pi qu'elle est connectée. Pour cela, il faut ouvrir le **menu** de l'icône de framboise, choisir la catégorie **Préférences** et cliquer sur **Configuration du Raspberry Pi**. Une fois l'outil chargé, il faut cliquer sur l'onglet **Interfaces**, chercher Caméra dans la liste, cliquer Activé pour l'activer et puis terminer avec OK.
- Pour confirmer que votre **Module** est correctement installé, vous pouvez utiliser l'outil **raspistill** en ligne de commande en tapant:
raspistill -o test.jpg.

La programmation sous Raspberry Pi

Présentation et utilisation de PiCamera

- La manière la plus simple de contrôler le Module Camera est d'utiliser Python, via la bibliothèque **picamera** qui est très pratique.
- Elle permet un contrôle total de la caméra (prévisualisation, la capture d'images et de vidéos, ...) en l'intégrant dans vos programmes, en le combinant éventuellement avec des programmes utilisant le module GPIO.
- Pour pouvoir l'utiliser, il suffit d'importer les bibliothèques dont votre programme a besoin en saisissant:
 - `from picamera import PiCamera`
 - Puis créer un objet PiCamera: `camera = PiCamera()`

Photos:

- On peut commander avec les fonctions `start_preview()` et `stop_preview()` pour la visualisation.
- On peut aussi pivoter l'image (à 90° par exemple) en mettant:
`camera.rotation = 90`
- Il est aussi possible de capturer et enregistrer une image en mettant:
`camera.capture('..chemain/image.jpg')` ex: `/home/pi/Desktop/image.jpg`.

Vidéos:

- On peut également manipuler des flux vidéos avec les fonctions `start_recording('..chemain/video.h264')` pour capturer et `stop_recording()` pour arrêter.

La programmation sous Raspberry Pi

PiCamera: cas pratique avec un bouton poussoir

- Enregistrement de séquence d'image:

La programmation sous Raspberry Pi

PiCamera: cas pratique avec la détection de mouvement

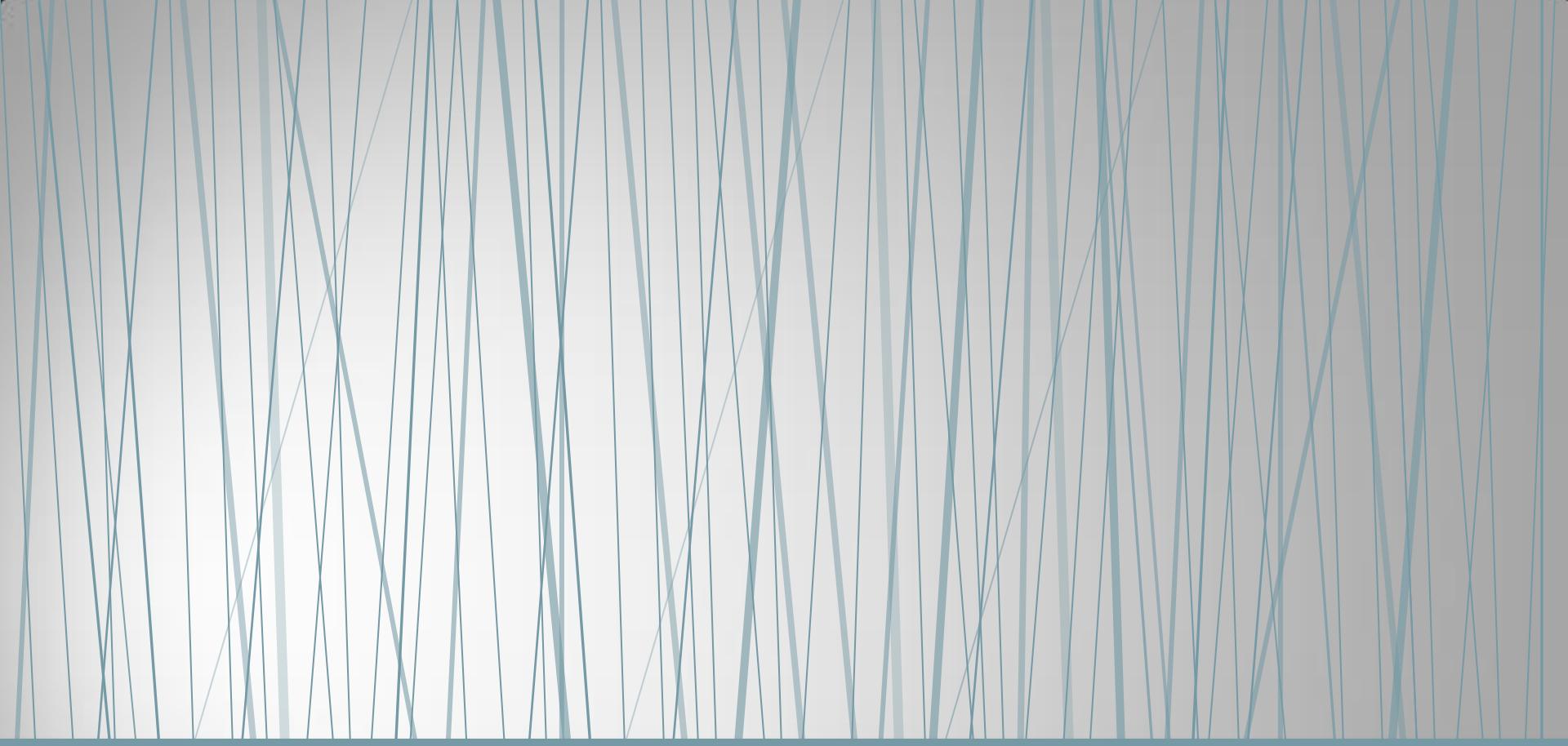
- Enregistrement de séquence d'image:

La programmation sous Raspberry Pi

Paramètres avancés de la caméra

À suivre

Feedback sur:
pape.abdoulaye.barro@gmail.com



Initiation à l'impression 3D

FIN

Feedback sur:
pape.abdoulaye.barro@gmail.com