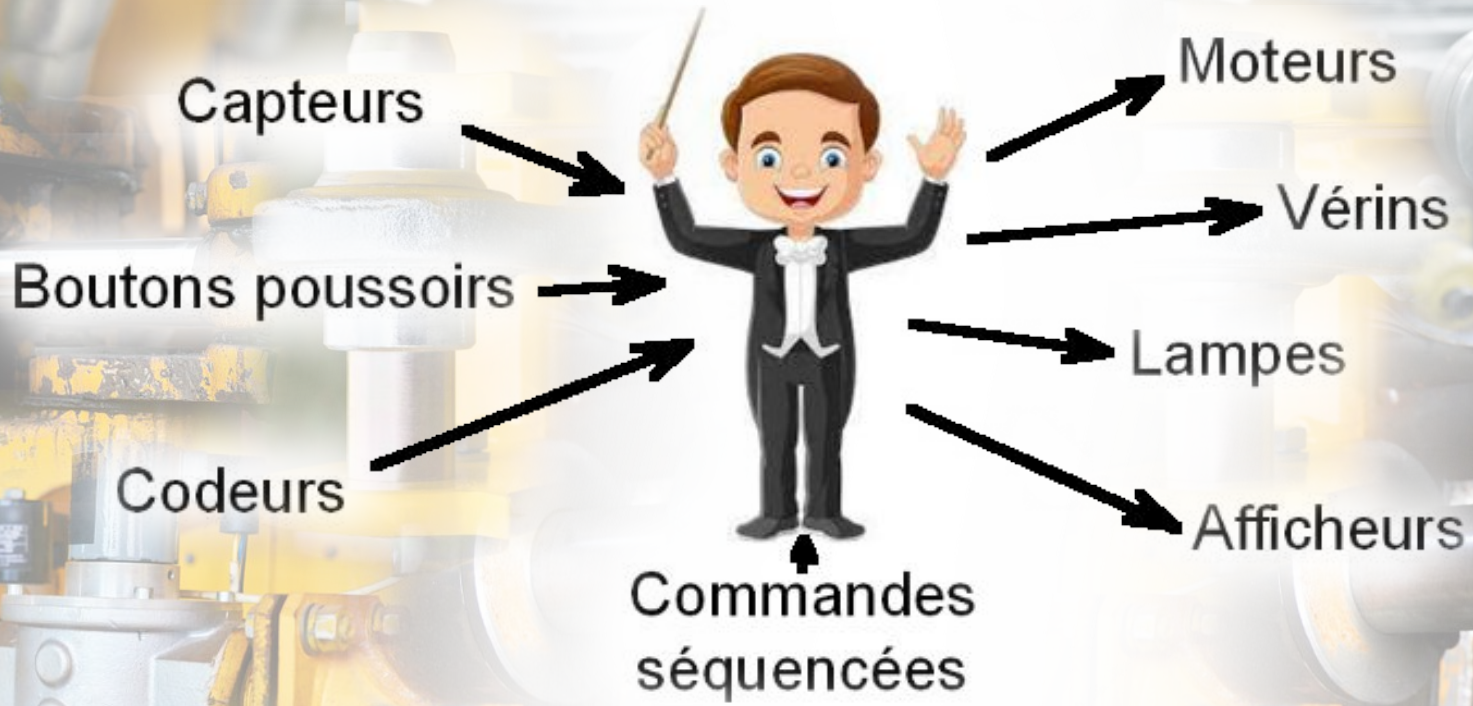
IDROLIK
AUTOMATISATION HYDRAULIQUE

Arduino - premier contact (suite)



IDROLIK
AUTOMATISATION HYDRAULIQUE

Sommaire

IDROLIK
AUTOMATISATION HYDRAULIQUE

Information préliminaire

- Cette présentation est accessible via le site idrolik.com
  - rubrique documentation
- Elle sera mise à jour sur le site au fur et à mesure qu'elle évoluera
- De cette manière, vous aurez accès aux exemples de programmes pour vérifier que ça marche

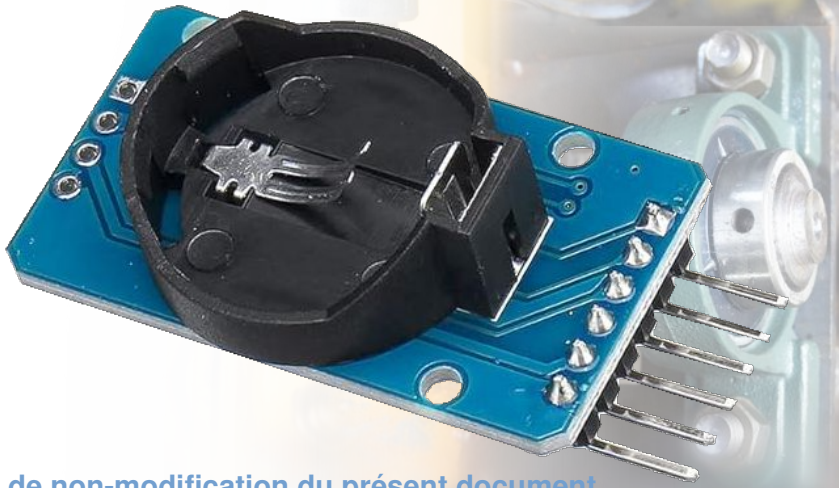
IDROLIK
AUTOMATISATION HYDRAULIQUE

Le bus I2C

IDROLIK
AUTOMATISATION HYDRAULIQUE

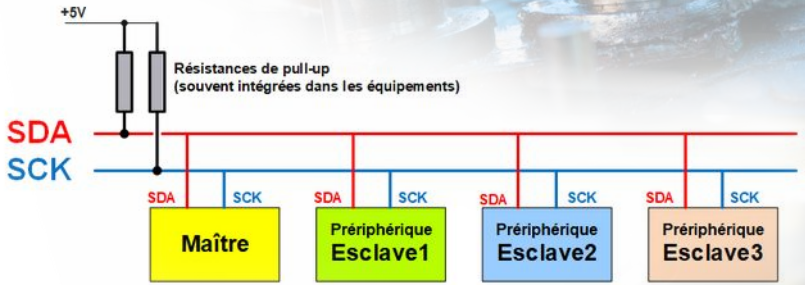
Le bus I2C

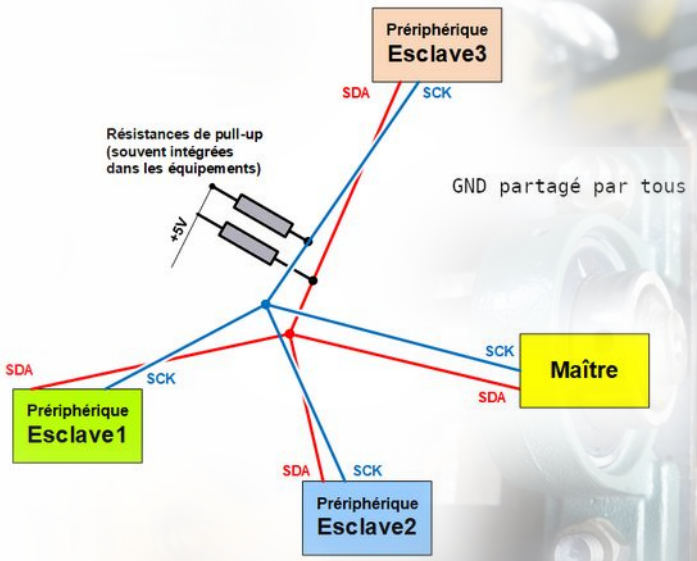
- Si vous voyez un module avec deux broches
  - « SCL » : Serial Clock
  - « SDA » : Serial Data
- Vous êtes en présence d'un circuit qui communique par I2C
  - Exemple ci-contre :
    module RTC (Real-Time Clock)
    ZS-042 DS3231



Le bus I2C

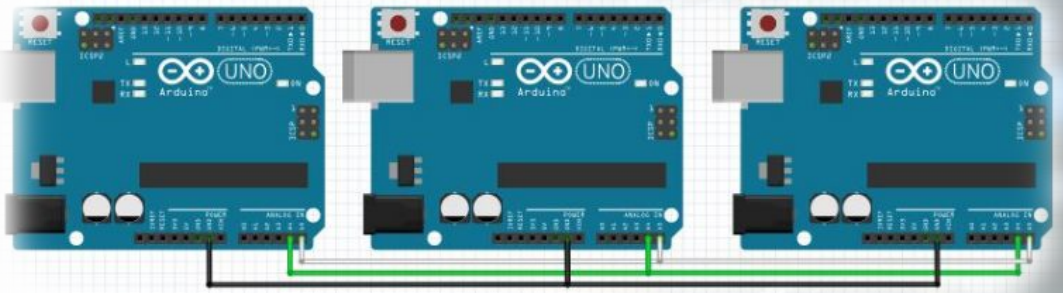
- I2C pour Inter-Integrated Circuit
  - Il s'agit d'un bus sur deux fils : SCL et SDA
  - SCK : Serial ClocK
  - SDA : Serial Data
  - Les deux fils sont partagés par tous les équipements (qui partagent également le même GND)
  - Au repos, ils maintenus à 5V par des résistances pull-up (~10 kΩ, souvent intégrées dans les équipements)
  - Les équipements forcent le niveau de ces signaux à zéro au besoin
- Chaque équipement a une adresse sur 7 bits
- Avant d'initier une conversation ...
  - Les lignes doivent être au repos (pas de communication en cours)
  - Bus au repos = les deux lignes à 5V



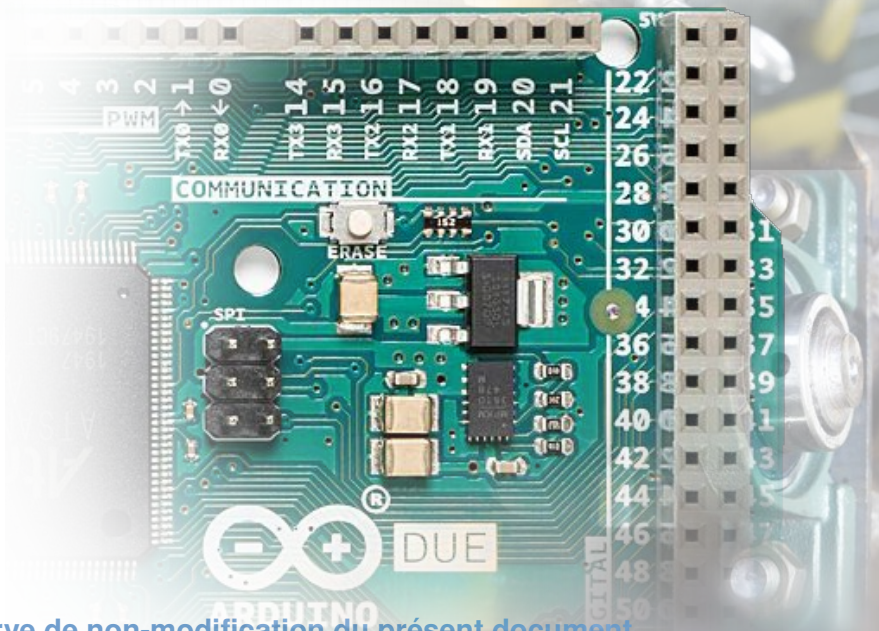


IDROLIK
AUTOMATISATION HYDRAULIQUE

Le bus I2C

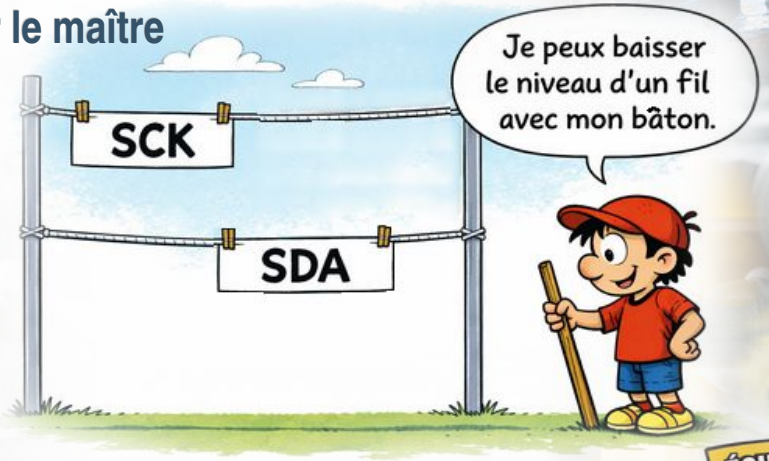


- Sur un bus I2C on peut trouver
  - Un ou plusieurs afficheurs I2C
  - Un RTC (Real-Time Clock) I2C
  - Un ou plusieurs microcontrôleurs
  - Des capteurs I2C …
- Les adresses des périphériques sont souvent (au moins partiellement) fixées par les fabricants de périphériques
  - Les adresses en dur peuvent limiter le nombre de périphériques



Le bus I2C

- Liaison série avec horloge SCK gérée par le maître
  - Maître = initiateur de l'échange
  - Fréquence SCK : 100 kHz
- Les équipements ne mettent jamais les lignes SCK et SDA à 1
  - Ce sont les résistances pull-up qui se chargent de les maintenir à 1
  - Lorsqu'un échange est en cours, les équipements peuvent abaisser le niveau d'une ligne pour mettre le bit correspondant à 0
- Longueur maxi du bus : 1 mètre
- ATTENTION : sur I2C, un seul équipement qui déconne en forçant indûment une des deux lignes à 0 et tout le bus est mort

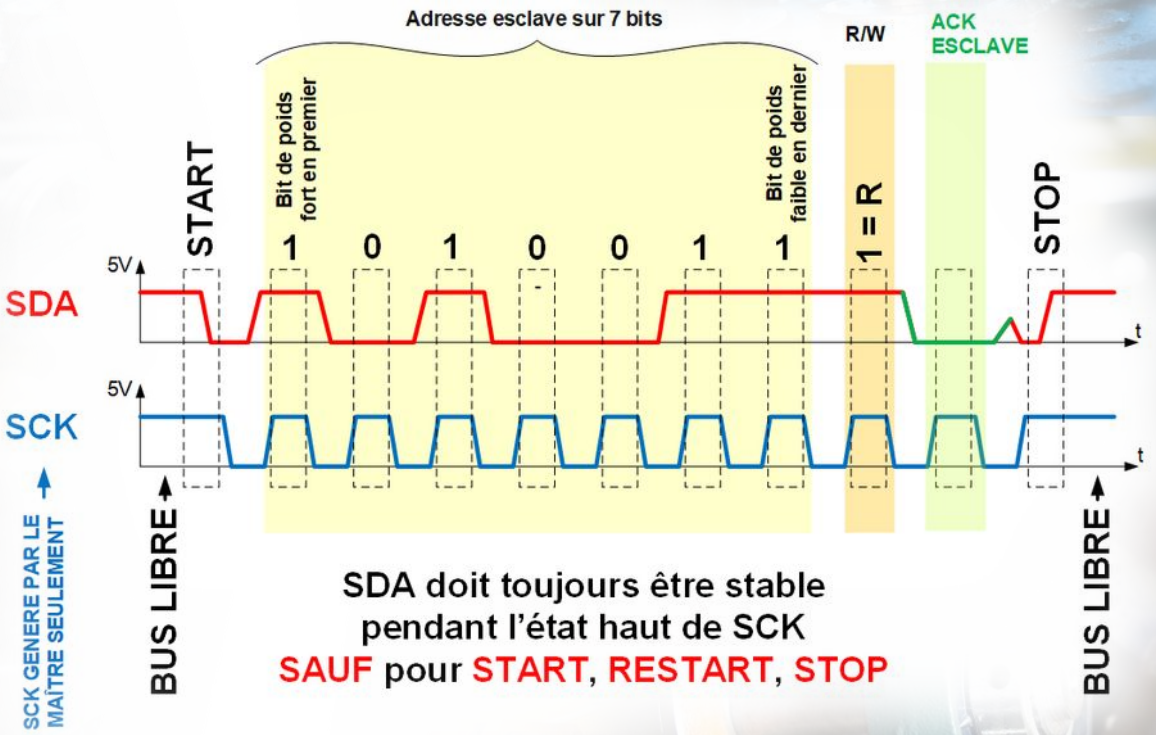


Le bus I2C

- A un instant donné :
  - Soit il ne se passe rien
  - Soit des échanges sont en cours, l'équipement maître envoie des octets ou en reçoit.
    On attend le signal « STOP »
- Un seul équipement initie chaque échange sur un bus I2C : LE MAITRE
  - Un échange commence avec « START »
    - C'est toujours le patron qui démarre la réunion en s'adressant à un ouvrier à la fois (« START »)
    - Dans l'usine, tout le monde se tait, seul le destinataire reçoit les informations que le maître lui envoie, ou transmet les informations que le maître attend.
- C'est le maître qui fixe la cadence sur la ligne SCK
- C'est le maître qui arrête la réunion quand il le décide (« STOP »)

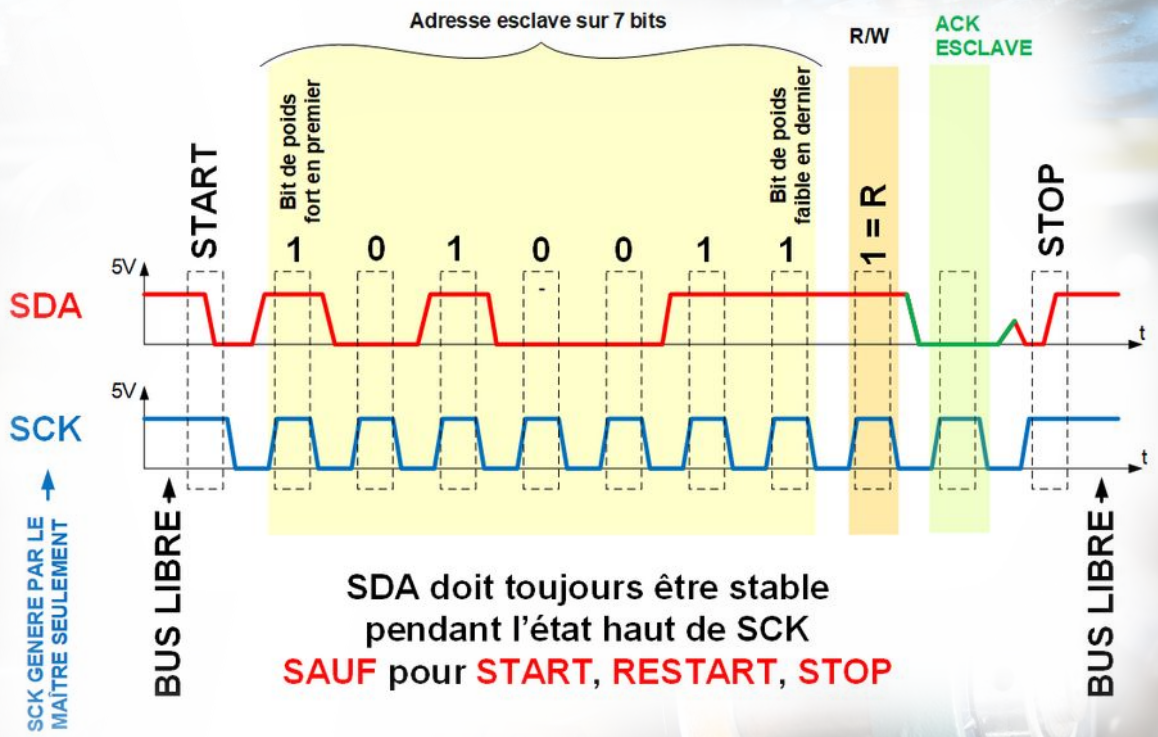
Le bus I2C

- « START » :
  - Le master force un front descendant sur SDA pendant qu'il laisse SCK à HIGH
- Le premier octet envoyé contient :
  - Les 7 bits de l'adresse du destinataire
  - 1 bit qui dit si le master transmet des données ou s'il en demande (0 = write, 1 = request for data)
- Echange :
  - Lors d'un échange chaque bit est à lire par le destinataire sur la ligne SDA pendant l'état HIGH de la ligne SCK
  - A part pour les signaux START et STOP, SDA doit toujours être stable pendant que SCK est à HIGH



Le bus I2C

- **Write : maître vers esclave, Demande de données : esclave vers maître**
  - Le destinataire (maître ou esclave) répond à chaque octet par un ACK (acknowledgment = accusé de réception)
  - ACK = SDA forcé à LOW par le destinataire (stable quand SCK est à HIGH)
- **« STOP »**
  - Chaque échange est clôturé par un « STOP » par le maître
  - « STOP » = le maître force un front montant sur SDA pendant que SCK est à 1



Le bus I2C

- **Un START (ou un RESTART) doit obligatoirement**
  - Être suivi de l'adresse de l'esclave (7 bits)
  - Le bit LSB (Least Significant Bit) indique la direction des données
    - 0 si le maître envoie des octets
    - 1 s'il demande des octets
- **Si personne ne répond (circuit destinataire non alimenté ou débranché)**
  - Pas de ACK (SDA non mis à zéro par l'équipement)
  - SDA reste à 1 = NACK, équipement non disponible.
- **Lorsque le destinataire reçoit des octets**
  - Il acquitte chaque octet reçu <u>en prenant le contrôle de SDA au 9ème bit</u>
  - S'il le force à zéro, c'est un ACK
  - S'il le laisse à 1, c'est un NACK et l'émetteur arrête d'envoyer des octets

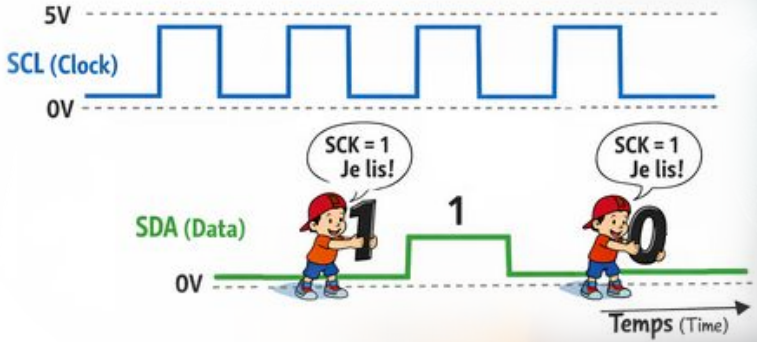
La bibliothèque « Wire.h »

- **Wire.h : bibliothèque pour communiquer sur le bus I2C**
  - Les fonctions de la bibliothèque « Wire » font appel à un module matériel TWI (Two Wire Interface) du microcontrôleur qui surveille et gère matériellement les lignes SCL et SDA
  - « #include <Wire.h> » en tout début de programme permet d'accéder aux fonctions de la bibliothèque
- **Chaque dispositif sur le bus doit avoir une adresse unique**
  - Avant de se choisir une adresse, on regarde ce qui va être connecté au bus parce que des constructeurs peuvent imposer des choses (plage d'adresses en fixant des bits ...)
  - Souvent dans les datasheets, on trouve les valeurs d'octets codées en Hexadécimal
    (= deux chiffres$_{\text{base 16}}$: 0, 1, 2, …, E, F)
  - Par exemple les trois écritures sont équivalentes :
    « Wire.begin(125); »
    « Wire.begin(0x7D) ; »
    « Wire.begin(0b01111101) ; »

« Wire.h »

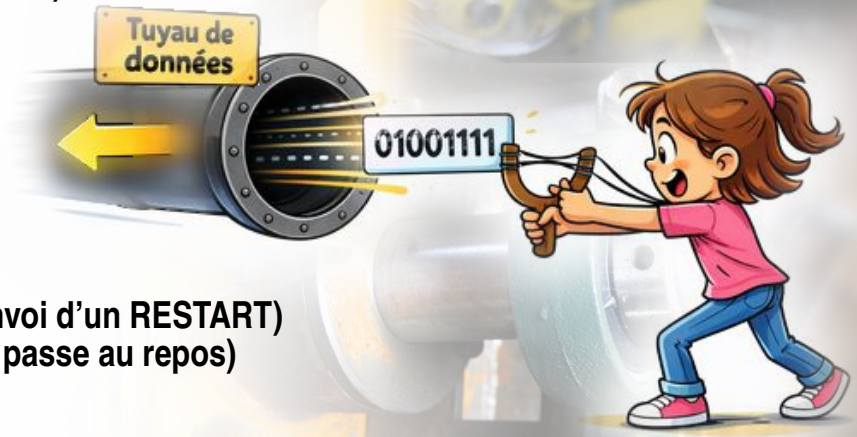
- **Taille des messages :**
  - Le bus I2C n'est pas conçu pour des transmissions massives de données dans un seul message
  - Prévoir des transferts < 32 octets (mémoires tampon limitées)
- **Fréquence SCK**
  - 100 kHz par défaut, peut être changée par le maître
  - Les esclaves échantillonnent la ligne SDA quand SCL est à 1 : ils s'adaptent tant qu'ils sont assez rapides
  - Certains équipements un peu plus lents peuvent faire du « clock stretching » : Lorsque le maître relâche SCK pour le laisser remonter à 1, l'esclave le maintient à 0 pour suspendre temporairement la communication.
- **Point d'attention :**
  - L'arduino fonctionne en 0 / 5V
  - D'autres circuits fonctionne en 0 / 3.3V
    Penser à adapter les tensions (level shifter I2C)

```
#include <Wire.h>
// ATTENTION : modification de la
// fréquence avec limitation a
// la fréquence maxi acceptable par
// le plus lent des équipements
// connectes au bus

void setup() {
  Wire.begin();
  Wire.setClock(150000); // 150 kHz
}
```



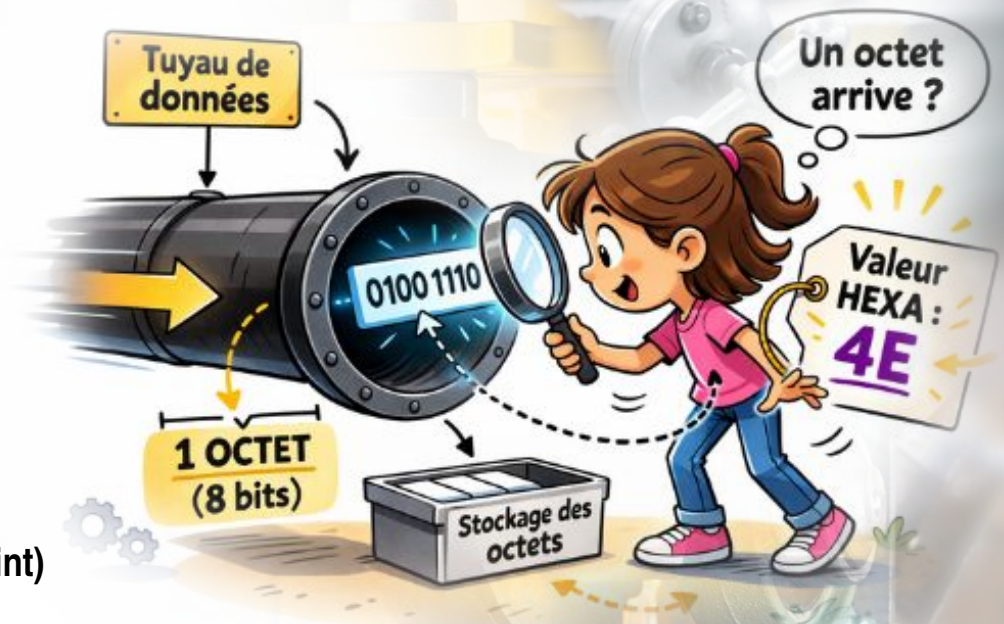
« Wire.h »

- « void Wire.begin(); »
  « void Wire.begin(int adresse Arduino); »
  - Si argument = adresse sur le bus, à choisir entre 8 et 127 (adresses 0 à 7 réservées)
    ATTENTION : ADRESSE SEULEMENT POUR UNE CARTE ARDUINO ESCLAVE
  - À mettre dans le « setup() » : initialisation/configuration du bus I2C
- « void Wire.beginTransmission(int adresse destinataire) ; »
  - Transmission : le bit de poids faible (qui suit l'adresse) = 0 (= write)
- « Wire.write(...) ; »
  - « Wire.write(byte c) ; »Envoie l'octet c
    « Wire.write(byte c[], size_t n) ; » envoie n octets du tableau c
- « byte Wire.endTransmission(boolean stop) ; »
  - stop = false : on arrête la transmission sans relâcher le bus (envoi d'un RESTART)
    stop = true : on arrête la transmission et on relâche le bus (qui passe au repos)
  - Valeur de retour
    0 : success, 1 : data too long to fit in transmit buffer, 2 : received NACK on transmit of address, 3 : received NACK on transmit of data, 4 : other error, 5 : timeout



« Wire.h »

- « byte Wire.requestFrom(int adresse, int nbre, boolean stop) ; »
  - Request : le bit de poids faible (qui suit l'adresse) = 1 (= read)
  - Adresse destinataire sur 7 bits (int)
  - Nombre d'octets demandés (int)
  - stop = true : envoi d'un stop et on relâche le bus
    stop = false : envoi d'un restart pour garder le bus
  - Valeur retournée : nombre d'octets obtenus (byte)
- « int Wire.available() ; »
  - Nombre d'octets à lire dans le buffer (int)
- « int Wire.read() ; »
  - Octet lu ou -1 si aucun octet dispo dans le buffer (int)
  - Exemple :
    char c ;
    c = Wire.read() ;



« Wire.h »

- Arduino Maître parle avec Arduino Esclave
  - Wire.onRequest();
  - Wire.onReceive();
- L'arduino esclave est passif vis-à-vis d'I2C
  - Il réagit quand on lui demande
  - Ça ne l'empêche pas d'avoir des données prêtes
- Rappel : en I2C, messages courts < 32 octets

Arduino maître

```
#include <Wire.h>

void setup() {
  Wire.begin();
}

void loop() {
  Wire.requestFrom(8, 1);

  while (Wire.available()) {
    int c = Wire.read();
  }

  delay(1000);
}
```

Arduino esclave

```
#include <Wire.h>

void setup() {
  Wire.begin(8);
  Wire.onRequest(sendData);
}

void sendData() {
  Wire.write(42);
}

void loop() {
}
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

Scan I2C

- Scan du bus I2C
  - L'envoi de l'adresse + bit « écriture »
  - Si personne à l'adresse indiquée : pas de ACK en mettant SDA à 0
  - Code erreur différent de 0
- Lorsqu'on connecte un nouveau périphérique I2C
  - Si on a confiance, on le connecte au bus en plus des équipements déjà connectés (et on espère qu'il n'y a pas de conflit d'adresse)
  - Si on est moins confiant, on le connecte seul au bus
  - On lance le scan pour voir quelle(s) adresse(s) répond(ent)

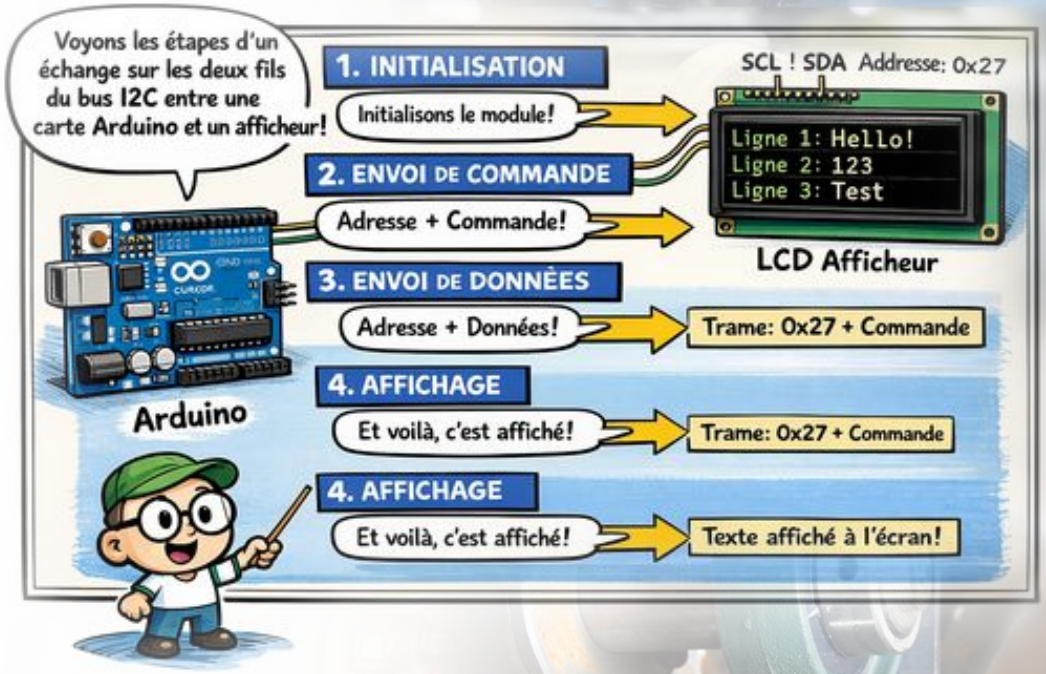
```
#include <Wire.h>
// SCAN BUS I2C

void setup() {
  Wire.begin();
  Serial.begin(9600);
  Serial.println("Scan I2C...");
}

void loop() {
  byte error, address;
  int nDevices = 0;

  for(address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0) {
      Serial.print("I2C trouvé à : 0x");
      if (address < 16) Serial.print("0");
      Serial.println(address, HEX);
      nDevices++;
    }
  }

  if (nDevices == 0)
    Serial.println("Aucun périphérique I2C");

  delay(5000);
}
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

L'afficheur LCD

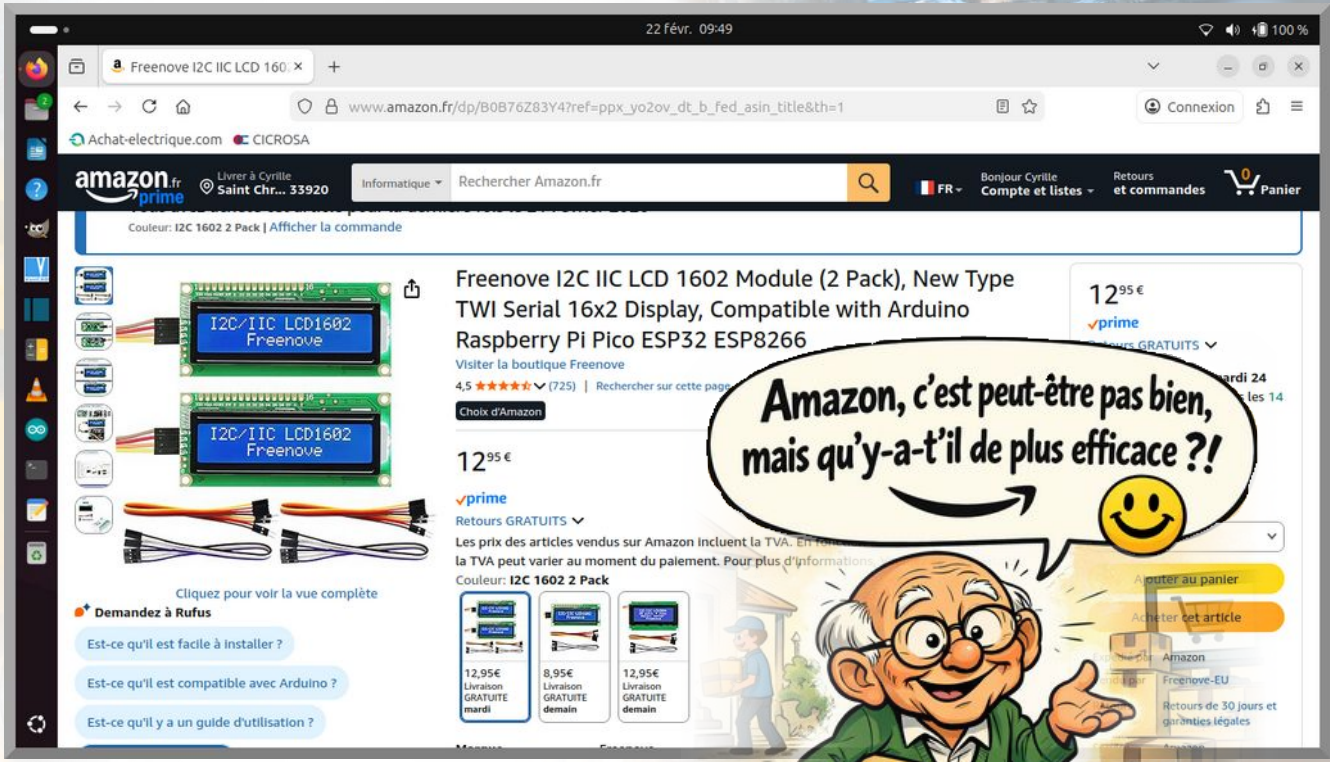
Dialogue afficheur

- **Dialogue dans un seul sens**
  - Messages de configuration
    - Effacement écran ...
  - Envoi du message à afficher
- **Intérêt d'un afficheur I2C :**
  - On peut mettre plusieurs afficheurs sur le bus I2C
  - Ça peut être utile si on veut présenter un panneau de commande un peu sympa avec différentes informations réparties sur un schéma d'installation (heure, températures, niveaux de liquide, poids ...)
- **Bonne pratique : prévoir des connecteurs pour la maintenance**
  - Il faut pouvoir déconnecter facilement chaque afficheur
  - Rappel : un seul afficheur qui dysfonctionne et le bus peut être par terre = plus d'affichage du tout
  - Objectif : faciliter la recherche de panne



Achat d'un afficheur

- Je suis allé sur Amazon
  - Ça n'est peut-être pas bien …
  - Mais c'est tellement efficace !
- J'ai pris pratiquement le premier afficheur I2C qui s'est présenté
  - Chaîne de recherche : « Arduino afficheur I2C »
  - 2 lignes de 16 caractères
  - J'ai pris un pack de 2, pour le cas où j'en détruirais un avec un mauvais branchement

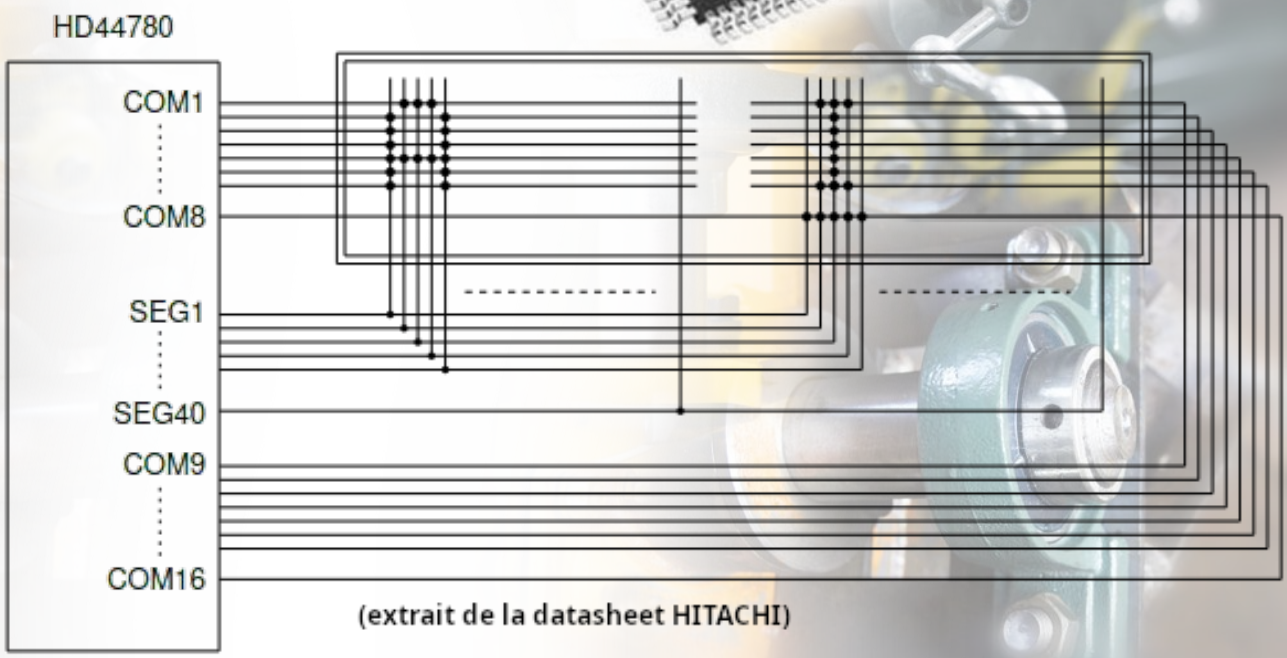


Composants du module

- Ce type de module d'affichage est composé de trois principaux éléments
- 1- L'expandeur I2C
  - Il reçoit les octets du bus I2C (bits reçus en mode série)
  - Il présente les bits sur les entrées du circuit d'affichage
  - Expandeurs série → parallèle utilisés par ce type d'afficheur : PCF8574 (NXP) ou PCF8574AT (Texas Instruments)
- 2- Le contrôleur d'affichage de matrices de points LCD
  - Il lit les bits qui se présentent en parallèle sur ses entrées
  - Lorsque l'afficheur est utilisé en direct (sans expandeur I2C), l'octet à afficher est présenté soit en 1 x 8 bits, soit en 2 x 4 bits pour économiser les sorties du microcontrôleur
  - Circuit historiquement plébiscité : HD44780 (Hitachi)
- 3- L'afficheur à cristaux liquides lui-même

HD44780 / compatibles

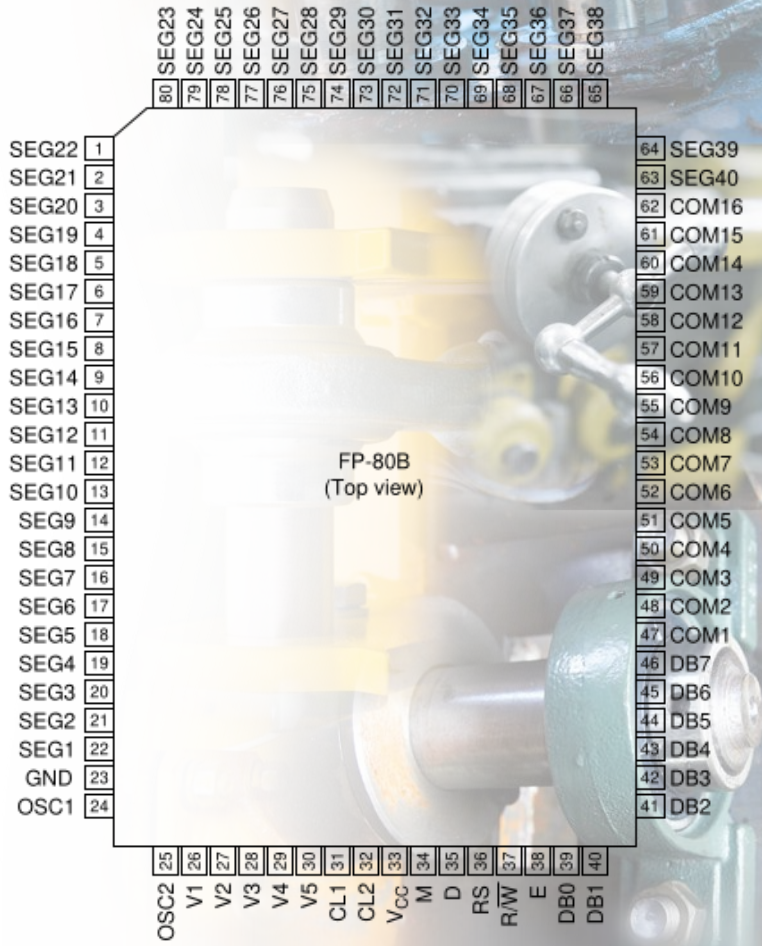
- Le contrôleur d'affichage lit les commandes
  - Clear,
  - Mode 8 / 4 bits,
  - Positionnement du curseur
  - ...
- Il lit les caractères à afficher
  - = « allume » une matrice de points de l'afficheur pour représenter un caractère au curseur courant
- Le contrôleur d'affichage est aussi utilisé sur des afficheurs non I2C
  - Présentation des octets en parallèle (un fil par bit)



(extrait de la datasheet HITACHI)

HD44780

- **Broches afficheur LCD :**
  - SEG1 à SEG40
  - COM1 à COM16
- **Data bus (instruction ou caractère) :**
  - DB0 (bit de poids faible) à DB7 (bit de poids fort)
- **Principaux sélecteurs**
  - RS (Register Select)
    - 0 : Instruction, 1 : Data
  - R/W
    - 0 : Write, 1 : Read
  - E (Enable)
    - Déclenche l'écriture / la lecture



Afficheur 1602A

- Lien indiqué dans l'emballage
  - « http://freenove.com/tutorial »
- Autres entrées afficheur 1602
  - Idem HD44780, avec
    - Vss : 0V (GND)
    - Vdd : 5V
    - V0 : input pour ajuster le contraste
    - A : +5V (anode rétroéclairage)
    - K : 0V (cathode rétroéclairage)

Afficheur 1602A

- **Principales instructions**
  - Clear : 0x01
  - Display ON/OFF control :0b00001xxx
  - Function Set : 0b001xxxxx
  - Curseur en début de seconde ligne (si 2 lignes : 0xC0)
- **Busy flag**
  - Lecture du bit de poids fort DB7
  - DB7 = 1 : afficheur occupé, on attend avant de lui envoyer un autre octet
  - DB7 = 0 : OK pour écriture d'un nouvel octet

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Code	0	0	0	0	0	0	1	D	C	B

D = 1: Display on, D = 0: Display off

C = 1: Cursor on, C = 0: Cursor off

B = 1: Blinks on, B= 0: Blinks off

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Code	0	0	0	0	1	DL	N	F	X	X

X: Do not care (0 or 1)

DL: It sets interface data length.

DL = 1: Data transferred with 8-bit length (DB7 - 0).

DL = 0: Data transferred with 4-bit length (DB7 - 4).

It requires two times to accomplish data transferring.

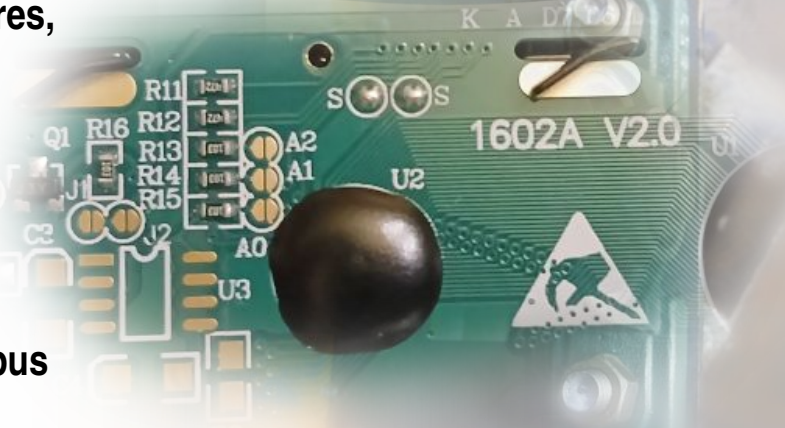
N: It sets the number of the display line.

N = 0: One-line display.

N = 1: Two-line display.

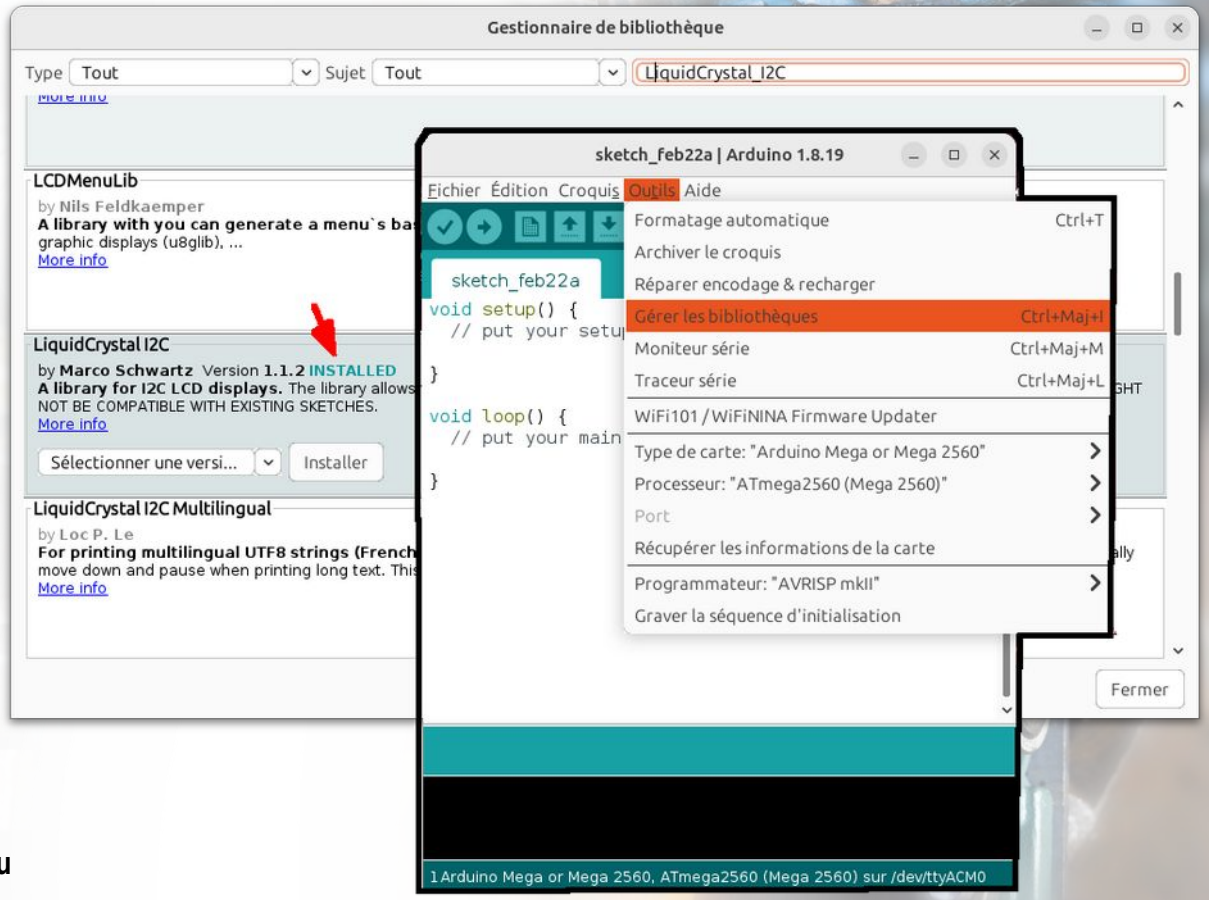
Adresse I2C

- L'adresse par défaut de l'expandeur est 0x27 ou 0x3F
  - Si ce n'est pas le cas, on pourra toujours scanner le bus I2C pour voir quelle adresse répond (ACK = quelqu'un met SDA à 0)
  - Plage d'adresses possibles suivant l'expandeur :
    PCF8574T : 0x20 → 0x27
    PCF8574AT : 0x38 → 0x3F
- Changement de l'adresse
  - 3 zones A0, A1 et A2 peuvent être pontées avec des soudures, chacune mettant le bit correspondant à 0 (présence de résistances pullup)
  - 0x27 = 0b00200111
  - Les trois derniers bits sont A2, A1 et A0
    Chacun peut être mis à 0 par soudure
  - On peut donc mettre 8 afficheurs de ce type sur un même bus I2C

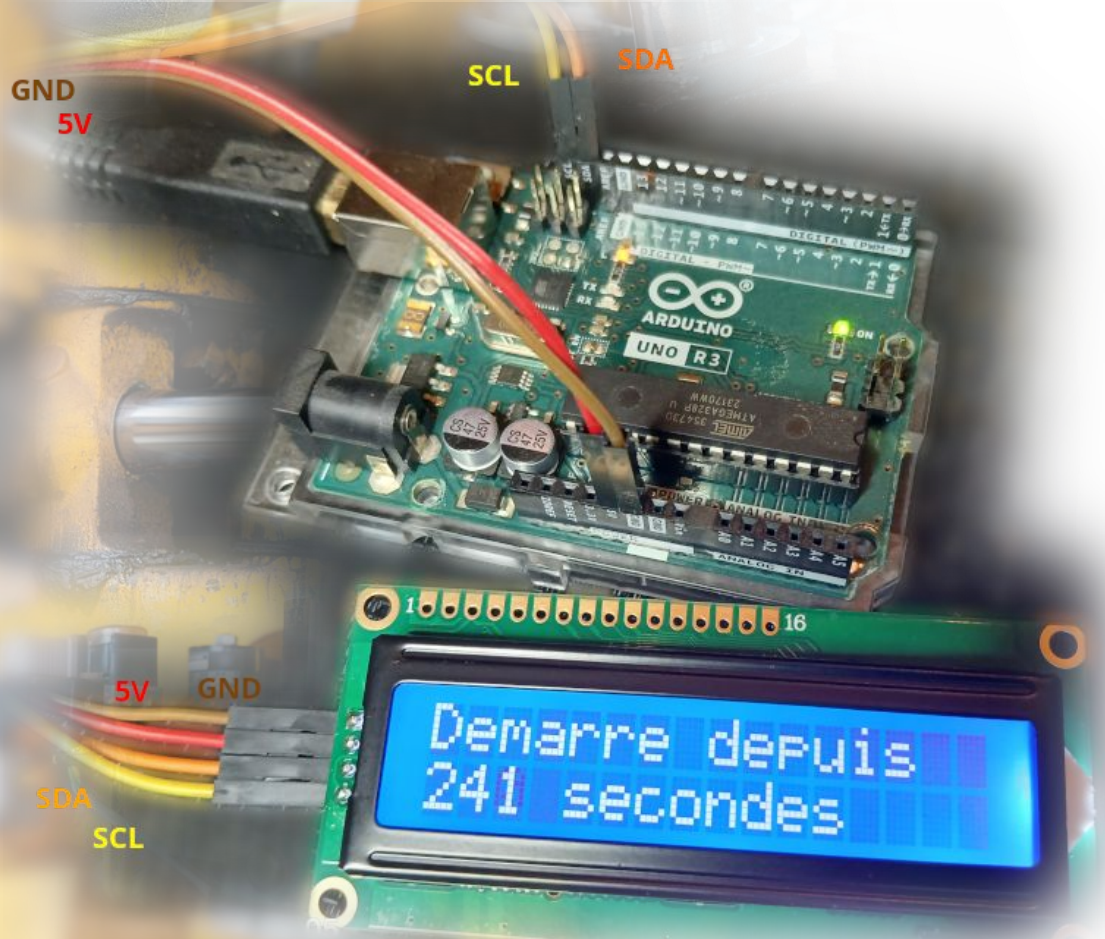


« LiquidCrystal_I2C »

- Dans mon cas une bibliothèque était déjà installée …
- Note : la commande « init() » de « LiquidCrystal_I2C » initie le bus I2C avec la bibliothèque « Wire »
  - Pas besoin de démarrer le bus I2C avec « Wire.begin() », LiquidCrystal_I2C le fait déjà
  - L'utilisation d'un afficheur I2C implique que l'Arduino démarre comme maître
- Attention :
  - Plusieurs bibliothèques « LiquidCrystal_I2C » existent, avec autant d'auteurs (Ici Marco Schwartz)
  - La manière dont l'expandeur présente les bits au contrôleur d'affichage peut amener une bibliothèque à ne pas fonctionner
  - Le Busy Flag n'est pas systématiquement pris en compte par la bibliothèque qui présuppose alors un délai « raisonnable » avant envoi d'un nouveau caractère à afficher



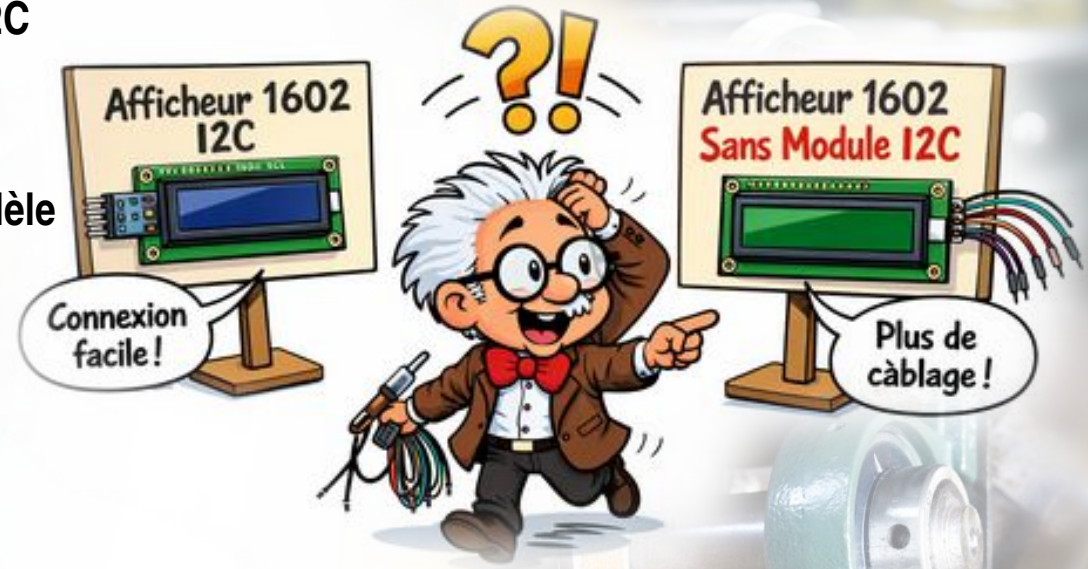
« LiquidCrystal_I2C »




```
#include <LiquidCrystal_I2C.h>

// definit une instance de la classe LiquidCrystal_I2C
// pour un afficheur deux lignes de 16 caracteres
// à l'adresse 0x27
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
   lcd.init(); // initialise l'instance lcd
   lcd.backlight(); // allume le rétro-éclairage
   lcd.clear();
   lcd.print("Demarre depuis");
}

void loop() {
   char buf[17]; // tableau a usage limite a la fonction

   lcd.setCursor(0, 1); // curseur en debut de 2eme ligne

   sprintf(buf, "%d secondes", millis()/1000);
   lcd.print(buf);
   delay(1000);
}
```

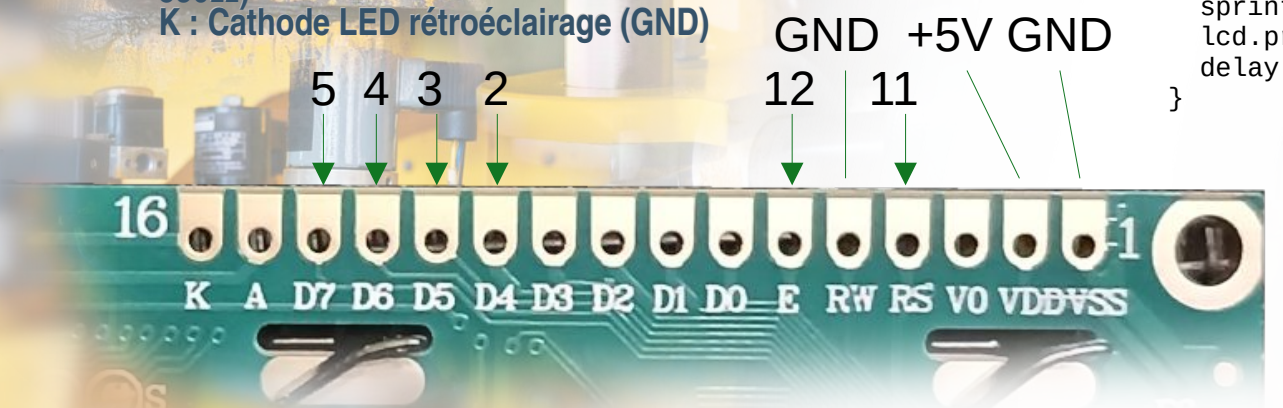
Affichage sans I2C

- On peut aussi afficher avec un afficheur non équipé d'un module I2C
  - Conforme avec l'idée de déporter via I2C les fonctions subalternes vers des Arduinos secondaires (solution préconisée plus loin)
  - Transfert des octets à afficher en parallèle
  - Fonctionnement plus simple
  - Bibliothèque « LiquidCrystal » uniquement basée sur la compatibilité HD44780 (devenue pratiquement un standard mondial)
- En contrepartie :
  - Plus de câblage
  - L'afficheur monopolise 6 sortie de l'Arduino



Affichage 1602A parallèle

- VSS : GND, VDD : +5V
- V0 : contraste (entre 0 et 5V). Connecté à GND, ça fonctionne. Normalement : potentiomètre pour ajuster la tension
- RS : Register Select (instr. / data)
- RW : Read/Write – on écrit, c'est un afficheur quand même !
- E : Enable
- D0 → D7 : data
- A : Anode LED rétroéclairage (reliée au +5V via une résistance de 330Ω)
  K : Cathode LED rétroéclairage (GND)

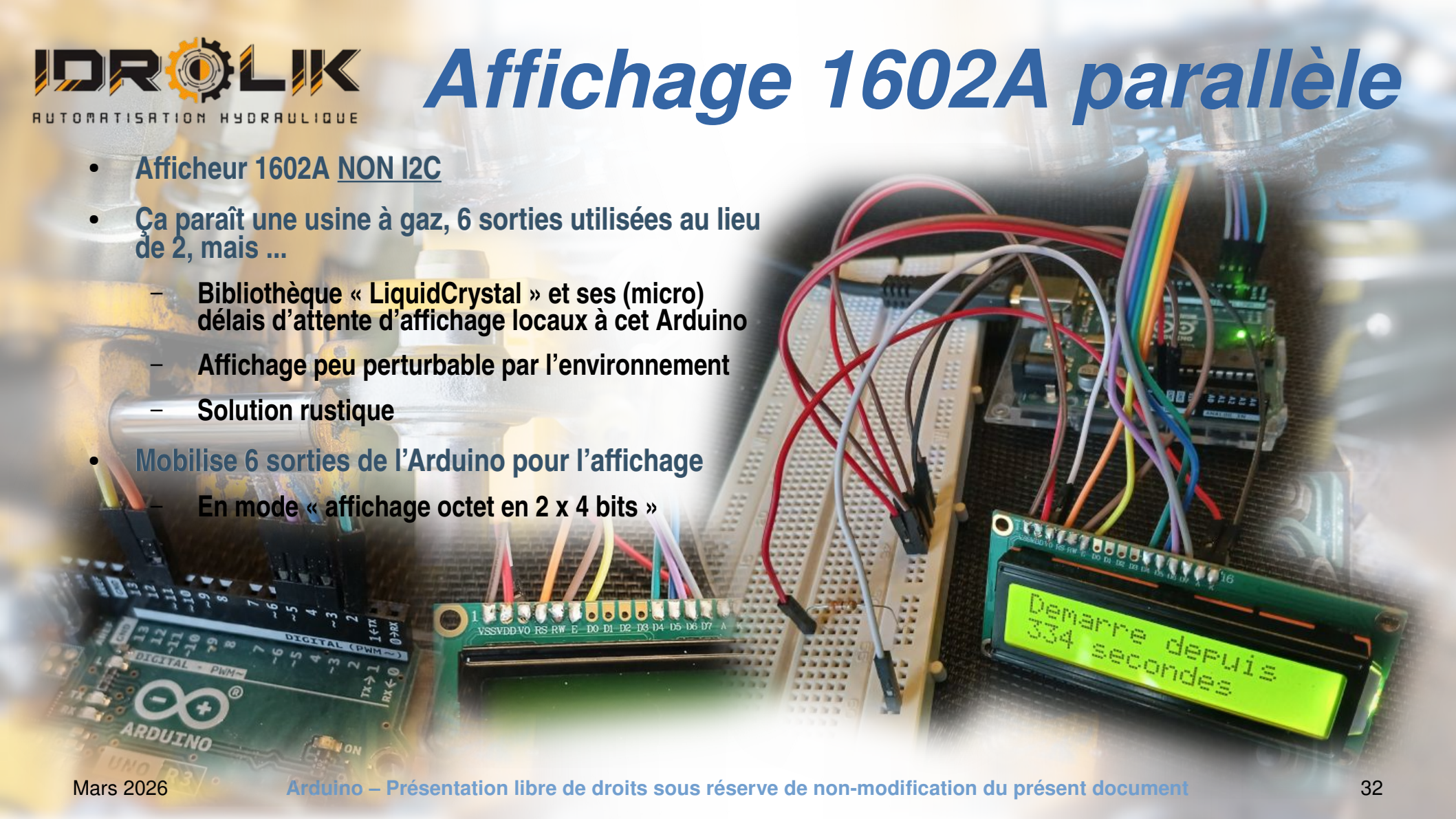



```
#include <LiquidCrystal.h>

// Transmission 2 x 4 bits pour économiser les pins
// LiquidCrystal(RS,enable,d4,d5,d6,d7)
LiquidCrystal lcd(11, 12, 2, 3, 4, 5);

void setup() {
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print("Demarre depuis");
}

void loop() {
  char buf[17];

  lcd.setCursor(0, 1);
  sprintf(buf, "%d secondes", millis() / 1000);
  lcd.print(buf);
  delay(1000);
}
```

IDROLIK
AUTOMATISATION HYDRAULIQUE

Affichage 1602A parallèle

- Afficheur 1602A NON I2C
- Ça paraît une usine à gaz, 6 sorties utilisées au lieu de 2, mais ...
  - Bibliothèque « LiquidCrystal » et ses (micro) délais d'attente d'affichage locaux à cet Arduino
  - Affichage peu perturbable par l'environnement
  - Solution rustique
- Mobilise 6 sorties de l'Arduino pour l'affichage
  - En mode « affichage octet en 2 x 4 bits »



Le VT100 des années 1980

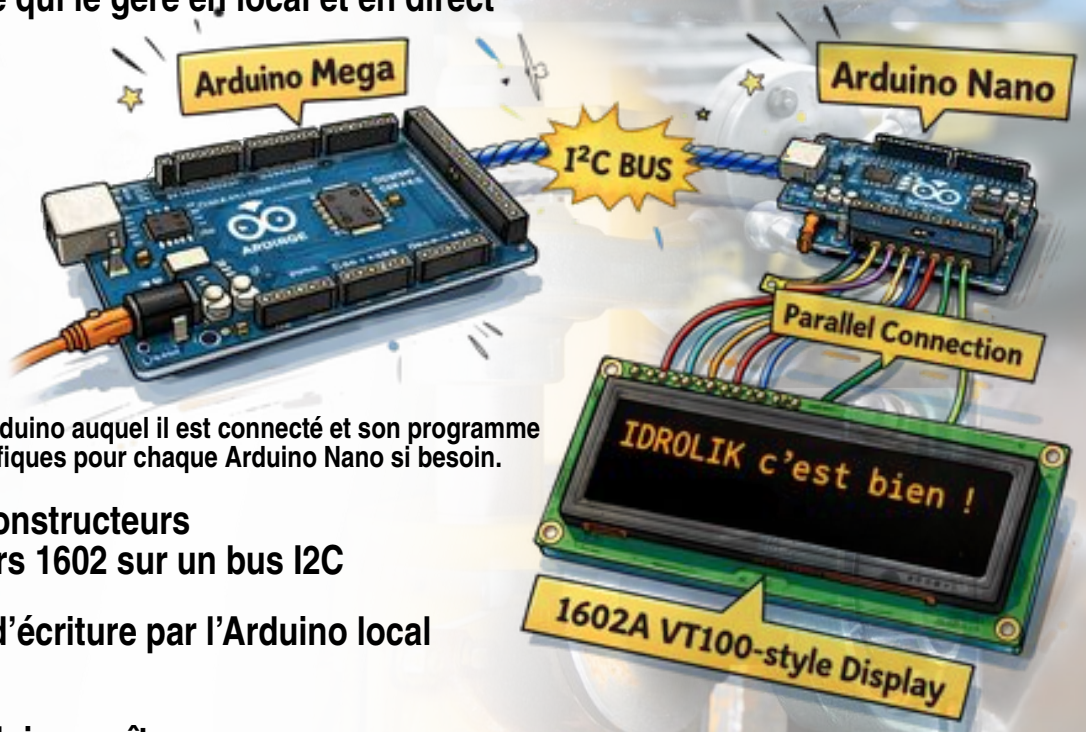
- **Le VT100**
  - Était un terminal « caractères »
  - Connecté à l'ordinateur via une liaison série
    - UART : Universal Asynchronous Receiver-Transmitter
    - Circuit toujours disponible sur Arduino !
  - Fabriqué à l'époque par Digital Equipments
  - Dans le début des années 1980
  - Une révolution à l'époque !

Génial tellement c'est simple !

- **Fonctionnement simple**
  - Des octets étaient envoyés au terminal via une liaison série
  - Les octets étaient soit affichés, soit interprétés comme des codes de contrôle
    - Codes ASCII ≥ 32 = caractères affichables
    - Codes ASCII < 32 = codes de contrôle
  - Exemples : caractère 13 pour « Carriage Return » (le retour chariot de la machine à écrire de l'époque)
    Caractère 10 (Line feed, passage à la ligne suivante)

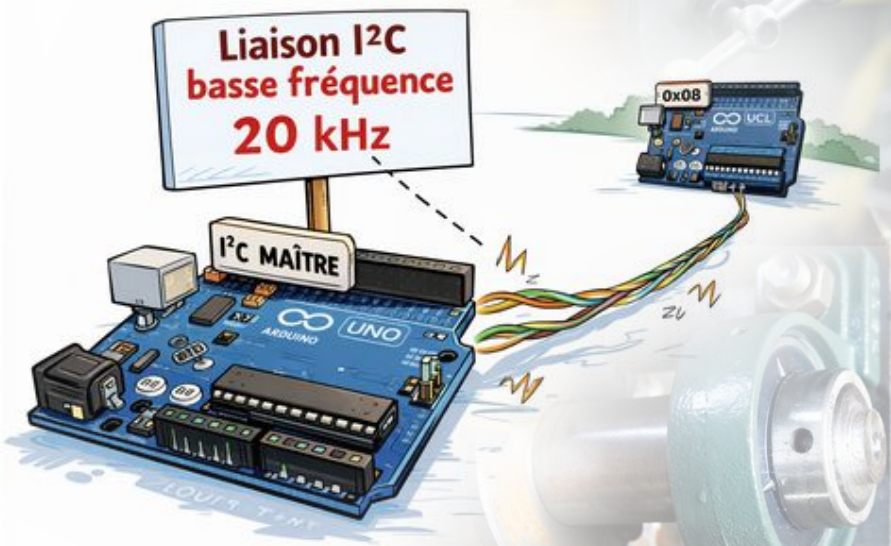
Config I2C préconisée

- Chaque périphérique est connecté au bus I2C via un Arduino Nano, affichages VT100-like
  - C'est l'Arduino Nano associé au périphérique qui le gère en local et en direct
- Avantages
  - Allègement de la charge du microcontrôleur maître (moins de bibliothèques à charger...)
  - Si périphérique initial devenu indisponible
    - Les échanges entre microcontrôleurs restent maîtrisés
    - Si matériel plus récent = adaptation peu intrusive :
      Le changement de périphérique ne concerne que l'Arduino auquel il est connecté et son programme de gestion (activité délocalisée). Bibliothèques spécifiques pour chaque Arduino Nano si besoin.
  - Plus de limitation des adresses I2C par les constructeurs
    Possibilité de mettre une centaine d'afficheurs 1602 sur un bus I2C
  - Gestion des espaces tampons et des délais d'écriture par l'Arduino local (qui compensent la non lecture du busy flag)
  - Gestion de type « Fire and forget » pour l'Arduino maître
- Inconvénient : plus cher, davantage de matériel



Config I2C préconisée

- Nécessite d'écrire le programme de l'Arduino esclave
- Permet de réaliser des afficheurs VT100-like
- Note :
  - Si le bus n'est utilisé QUE par des cartes Arduino, on crée un écosystème homogène.
  - On peut baisser la fréquence du bus de 100kHz à 20 kHz ou encore moins suivant distances
  - On rend son fonctionnement plus robuste
  - On peut envisager des distances entre équipements plus grandes de l'ordre de 2 ou 3 mètres
  - Des câbles torsadés SCK/GND et SDA/GND sont une bonne idée pour limiter les risques de perturbations électromagnétiques
  - Adapter les résistances de pull-up si besoin

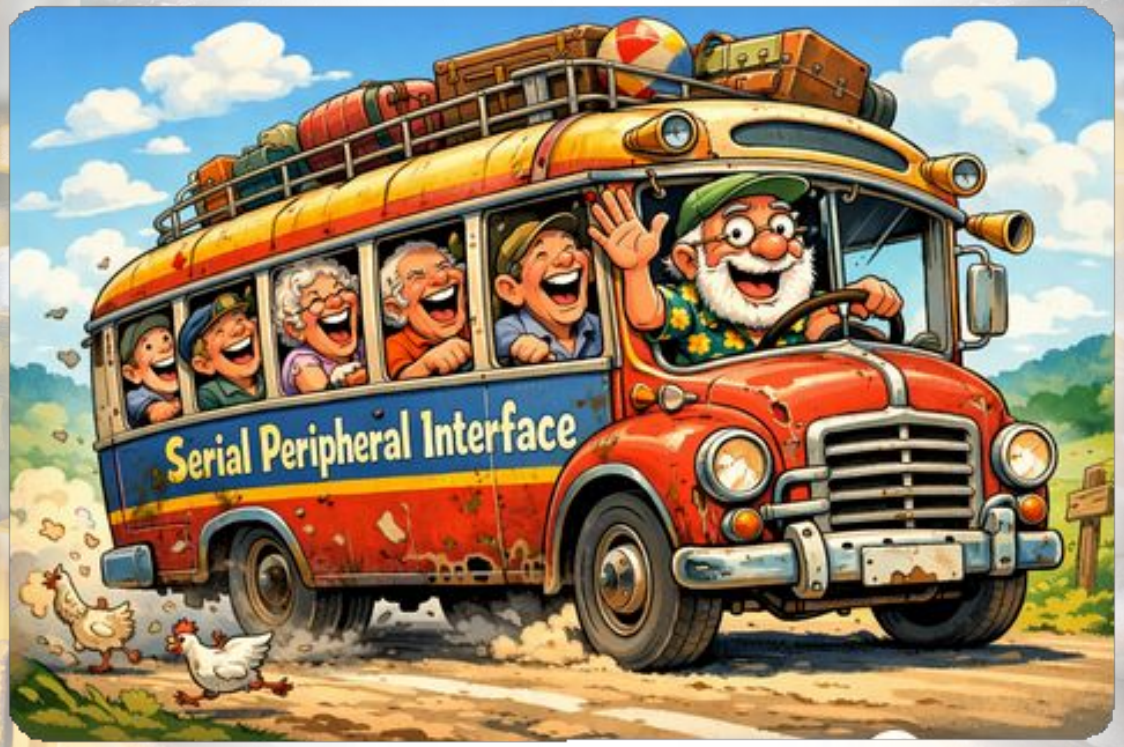


Config. I2C préconisée

- **Un Arduino maître déchargé des fonctions subalternes**
  - Il s'occupe des fonctions de plus haut niveau (travail de synthèse)
  - Il envoie des requêtes aux esclaves sans jamais attendre
  - Il interroge les esclaves « fournisseurs » à chaque boucle pour voir si des données sont disponibles
  - Il gère des dispositifs basiques :
    Leds d'activité et de diagnostic,
    Afficheurs leds 7 segments (indication phase en cours, code d'erreur …),
    Boutons poussoirs « stratégiques » (reset, modes de fonctionnement ...)
- **Des Arduinos esclaves, un pour chaque fonction de plus bas niveau**
  - Capteur, GPS …
    Afficheur, bargraphe, leds …
    Asservissement de la position d'un moteur ou d'un vérin sur une consigne
    Commandes de projecteurs leds 3 couleurs …
    Boutons poussoirs, potentiomètres, codeurs rotatifs ...
- **Les Arduinos esclaves, en plus de gérer leur capteur, leur moteur, leur vérin, leur projecteur (…) peuvent afficher en local les valeurs mesurées ou atteintes**
  (facilite le diagnostic en cas de dysfonctionnement, agréable pour l'utilisateur)

La brigade Arduino en action

Le bus SPI

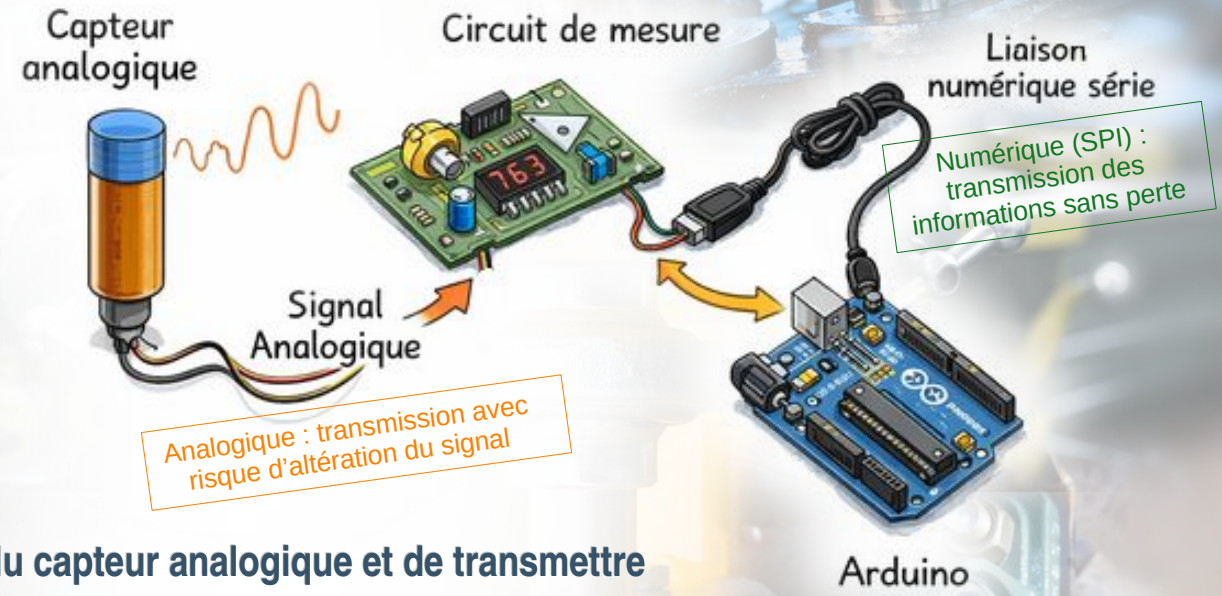


Le bus SPI

- Si vous trouvez un périphérique avec 4 broches en plus de VCC (+5V ou +3.3V) et de GND (Ground) :
  - Une broches « CS » (Chip Select)
  - Une broche « SCLK » (Serial Clock)
  - Une broche « SDI » (Serial Data In) ou « MOSI » (Master Out Slave In)
  - Une broche « SDO » (Serial Data Out) ou « MISO » (Master In Slave Out)
- Vous êtes en présence d'un circuit qui communique par SPI (Serial Peripheral Interface)
  - Exemple ci-contre : lecteur de carte SD

Le bus SPI

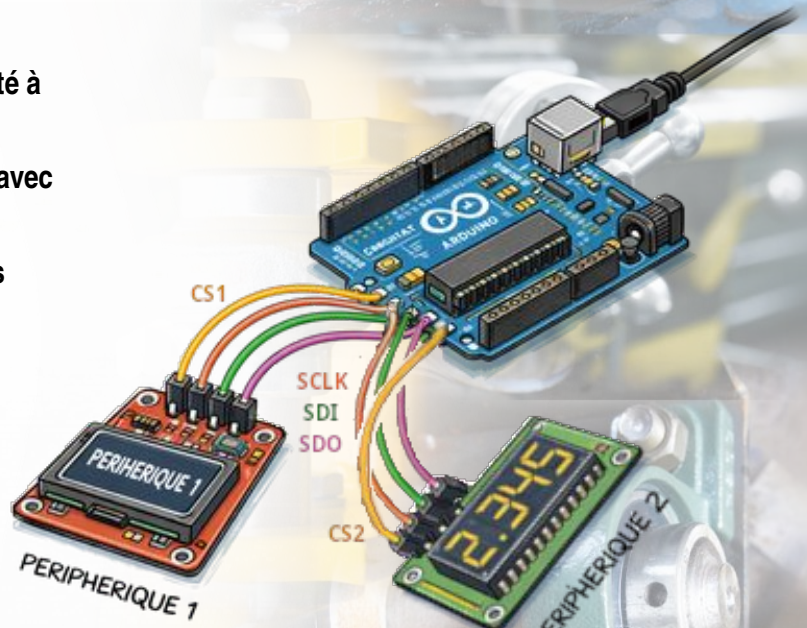
- SPI pour Serial Peripheral Interface
- Capteur analogique
  - Une grandeur physique est traduite en tension électrique
  - Une tension électrique est facilement perturbée par la longueur des fils entre le capteur et le point de mesure
    - Résistances de ligne
    - Parasites



- Donc on essaie de mesurer au plus près du capteur analogique et de transmettre numériquement la valeur mesurée
  - Un « 1 » transmis sous la forme d'une tension de 5V reste compris comme un « 1 » avec une tension de 4.5V = pas de perte d'information
- Un certain nombre de circuits destinés à traiter des mesures analogiques fournissent les valeurs via SPI
  - Exemple : le MAX31865 pour mesurer une température avec une PT100

Le bus SPI

- **SPI est :**
  - un bus série synchrone (comme I2C)
  - un bus de type maître-esclave (comme I2C), sans adressage (un fil connecté à chaque périphérique permet de désigner l'interlocuteur)
  - un bus full duplex (écritures/lectures dans les deux sens en même temps) avec un fil pour chaque sens
  - utilisé pour connecter un maître (Arduino) et un ou plusieurs périphériques (comme le MAX31865)
- **3 fils partagés par les périphériques**
  - SDI (Serial Data In) : MOSI (Master Out Slave In)
  - SDO (Serial Data Out) : MISO (Master In Slave Out)
  - SCLK (Serial Clock) : Clock générée par le master
- **Un fil CS pour chaque périphérique**
  - CS : Chip Select
    0 active le périphérique, 1 désactive le périphérique
  - Une broche SS (Slave Select) peut être disponible sur l'Arduino, l'utiliser comme CS pour le premier périphérique



Le bus SPI

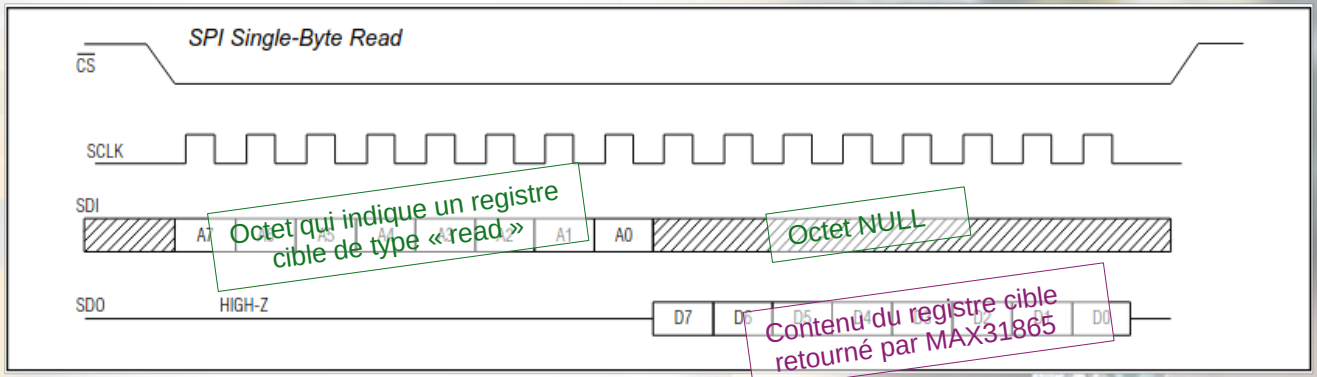
- Le maître met le CS du périphérique à 0 (il le sélectionne)
  - Une broche initialisée en sortie sur l'Arduino pour chaque périphérique
  - Les lignes SCLK, SDI et SDO sont partagées entre tous les périphériques
  - Les lignes SCLK, SDI et SDO sont gérées par un module matériel SPI intégré au microcontrôleur
    - On écrit un octet dans un registre du SPI, le microcontrôleur synchronise l'envoi sur SDI
    - Les octets reçus par le SPI sont placés dans un registre sans que le programme utilisateur ait à s'occuper de surveiller les lignes SCLK et SDO
- SDI : envoi d'octets au périphérique
  - Le maître place les bits sur SDI
  - À chaque changement d'état de SCLK (front montant / descendant suivant paramétrage), le bit correspondant est lu par le périphérique
    Exemple d'octet envoyé par le maître : registre demandé
- SDO : envoi d'octets au maître
  - Le périphérique place les bits sur SDO
  - Lorsque SCLK passe de 0 à 1 (front montant), le bit est lu par le maître
  - Note : le maître continue d'envoyer des octets vides pour générer l'horloge qui permet la réponse
- Le maître remet le CS du périphérique à 1 (il le désélectionne)

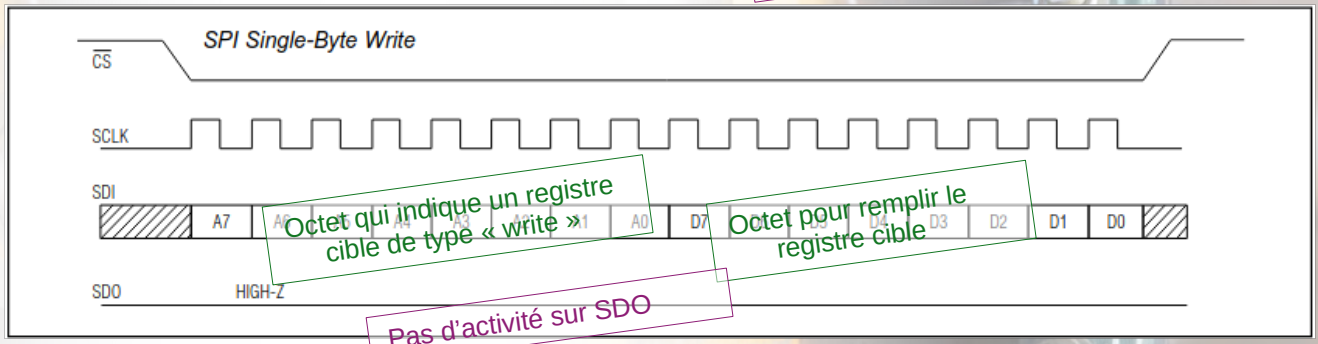
Fonctionnement SPI

- Mode full duplex, c'est comme le téléphone
  - Les deux correspondants peuvent parler en même temps Mais c'est le maître qui donne la cadence
  - Le maître peut parler et le correspondant dit [silence]
  - Le maître peut dire [silence] et le correspondant peut parler
  - C'est à la datasheet du périphérique de préciser comment se passe l'échange
- SPI ne précise pas comment les correspondants doivent se comprendre
  - La direction est codée dans l'octet envoyé
  - C'est comme au téléphone, entre un patron et un employé : suivant le début de conversation, l'employé sait s'il va recevoir des informations ou s'il va devoir en donner
- Le maître génère le signal d'horloge qui permet les transferts
  - Il génère le signal d'horloge quand il émet un octet sur SDI
  - Il envoie un octet NULL pour maintenir le signal d'horloge qui permet la réponse du périphérique sur SDO

SPI & MAX31865

- Extrait de la datasheet Analog Devices MAX31865
  - Le MAX31865 a des registres de type « read » et de type « write ».
  - Suivant le type de registre indiqué par le premier octet envoyé par le maître …
  - Registre de type « write » : Le périphérique attend un second octet de la part du maître pour remplir le registre cible
    - Typiquement registre de configuration
  - Registre de type « read » : Le périphérique profite de l'horloge générée par l'envoi de l'octet NULL par le maître pour envoyer sur SDO le contenu de son registre
    - Typiquement registre qui donne la valeur de la mesure





La bibliothèque SPI

- #include <SPI.h>
  - SPISettings est une classe utilisée pour définir les paramètres de communication SPI
  - Constructeur SPISettings(clock, bitOrder, dataMode) ;
    - Clock : 100000 = 100kHz
    - BitOrder : MSBFIRST ou LSBFIRST
    - DataMode : SPI_MODE0 à SPI_MODE3
- Commandes principales :
  - SPI.begin() : initialise le bus SPI
  - SPI.beginTransaction(SPISettings)
  - SPI.transfer()
    - SPI.transfer(byte) : 1 octet
    - SPI.transfer(buffer, size) : plusieurs octets
    - SPI.transfer16(uint16_t) : 16 bits
  - SPI.endTransaction()
  - SPI.end()

```
#include <SPI.h>
#define REG_CONFIG 0x00 // Exemples de registres :
#define REG_RTD_MSB 0x01 // ce sont les registres
#define REG_RTD_LSB 0x02 // du MAX31865

void maxWriteRegister(uint8_t reg, uint8_t value) {
  digitalWrite(PIN_CS, LOW);
  SPI.transfer(reg | 0x80);   // bit 7 = 1 => écriture
  SPI.transfer(value);
  digitalWrite(PIN_CS, HIGH);
  delay(100);
}

uint8_t maxReadRegister(uint8_t reg) {
  digitalWrite(PIN_CS, LOW);
  SPI.transfer(reg & 0x7F);   // bit 7 = 0 => lecture
  uint8_t value = SPI.transfer(0x00);
  digitalWrite(PIN_CS, HIGH);
  return value;
}

uint8_t msb, lsb, config;

SPI.begin();
SPI.beginTransaction(SPISettings(500000, MSBFIRST, SPI_MODE1));

config = CONFIG_BIAS | CONFIG_1SHOT | CONFIG_FILT50HZ;
maxWriteRegister(REG_CONFIG, config);
msb = maxReadRegister(REG_RTD_MSB);
lsb = maxReadRegister(REG_RTD_LSB);
```

Commandes données pour information, ce sont souvent les bibliothèques des capteurs qui gèrent les échanges par SPI

SPI_mode0 à SPI_mode3

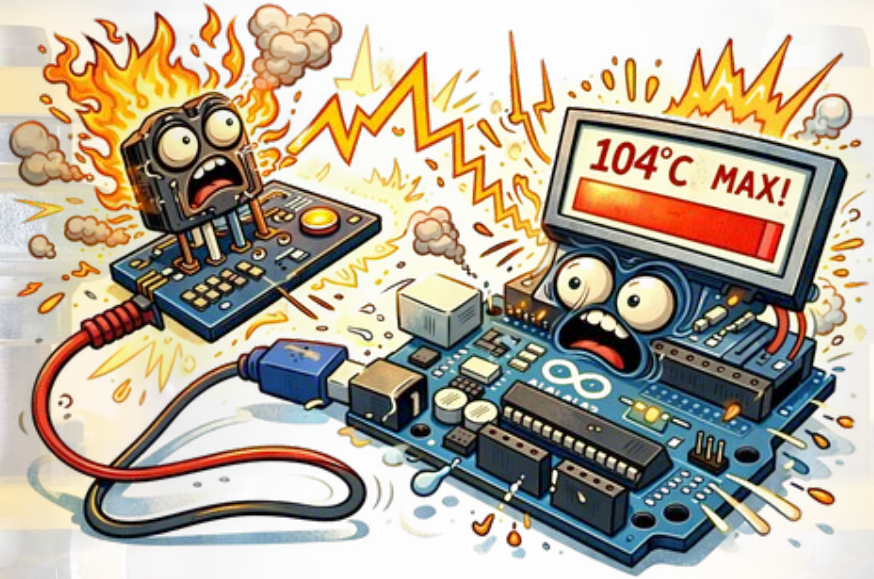
- **Les modes sont définis par :**
  - CPOL (Clock POLarity)
    CPOL = 0 : horloge au repos = LOW
    CPOL = 1 : horloge au repos = HIGH
  - CPHA (Clock PHAse)
    CPHA = 0 : lecture sur le premier front
    CPHA = 1 : lecture sur le deuxième front
- **Modes**
  - SPI_MODE0 : repos LOW, lecture sur front montant
  - SPI_MODE1 : repos LOW, lecture sur front descendant
  - SPI_MODE2 : repos HIGH, lecture sur front descendant
  - SPI_MODE3 : repos HIGH, lecture sur front montant

Mode2 et Mode3 plus rares

- **Le mode le plus répandu est le SPI_MODE0**
  - Mais ça vaut le coup d'aller voir la datasheet du composant
  - Le SPI_MODE1 reste cependant assez courant, les deux derniers sont plus rares

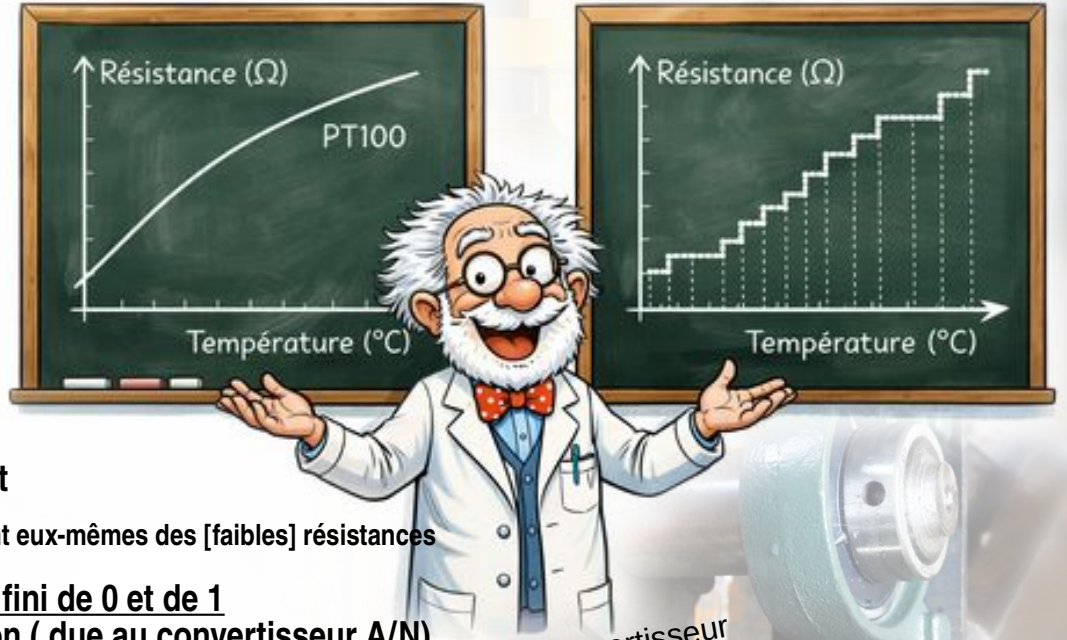
IDROLIK
AUTOMATISATION HYDRAULIQUE

Les capteurs analogiques



Numérique / analogique

- **Un capteur analogique traduit une grandeur physique continue en une autre grandeur physique continue**
  - Par exemple, « traduction » d'une température en résistance électrique ou en tension électrique
  - La température a une valeur exacte, la résistance / tension résultante aussi
  - La relation entre température et résistance est continue.
    = un changement infinitésimal de la température conduit à un changement infinitésimal de la résistance



- **Analogique ≠ numérique**
  - Analogique = valeur exacte
    Mesure possiblement altérée par l'environnement
    - Exemple : mesure d'une résistance via des fils qui sont eux-mêmes des [faibles] résistances
  - <u>Numérique = valeur approximée avec un nombre fini de 0 et de 1</u>
    (= nombre codé en binaire). On parle de résolution (due au convertisseur A/N).
    Exemple : tension entre 0 et 5V codée sur 10 bits
    Pas de perte d'information lors de la transmission d'une valeur numérique

CAN = Convertisseur Analogique Numérique

Capteurs analogiques

- Capteurs de température
  - LM35, TMP36, NTC/PTC
- Capteurs d'intensité lumineuse
  - Photo-résistance, Photodiode analogique, Phototransistor
- Capteurs de distance analogiques (réflexion infrarouge)
  - Sharp GP2Y0A21, Sharp GP2Y0A02
- Capteurs de pression ou de force
  - FSR (Force Sensitive Resistor), capteurs de pression analogiques
- Capteurs de position
  - Potentiomètre rotatif, Potentiomètre linéaire

Entrées analogiques

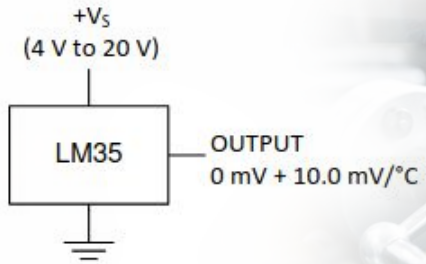
- « analogRead(pin) »
  - Lit une entrée analogique
  - Donne la tension lue sur 10 bits = entier entre 0 et 1023
  - Arduino ne sait pas lire une tension négative
    S'assurer que la tension qui se présente sur l'entrée analogique est bien entre 0 et 5V (sinon, risque de dégâts)
- Une fonction très pratique : « map() »
  - map(val, fromLow, fromHigh, toLow, toHigh) ;
  - Lecture d'une valeur val entre fromLow et fromHigh
  - Renvoi d'une valeur entre toLow et toHigh
- Lorsqu'on parle de tension analogique, attention aux longueurs de fil qui ajoutent de la résistance et affaiblissent et/ou perturbent le signal
  - Bonne pratique :
    Effectuer la mesure de tension au voisinage immédiat du capteur analogique, transmettre numériquement la valeur mesurée (par une liaison série de type I2C ou UART)

```
void setup() {
  pinMode(A1, INPUT);
  Serial.begin(9600); // pour affichage moniteur serie
}

void loop() {
  int i;
  long tension_mv;
  char buf[50];

  i = analogRead(A1); // i entre 0 et 1023
  tension_mv = map(i, 0, 2023, 0, 5000);
  // (tension_mv entre 0 et 5000 mv)
  // On aurait pu ecrire de maniere plus condensee :
  // tension_mv = map(analogRead(A1), 0, 2023, 0, 5000);

  sprintf(buf, "Tension lue sur A1 : %d mv", tension_mv);
  Serial.println(buf);

  delay(1000);
}
```

Capteur température LM35

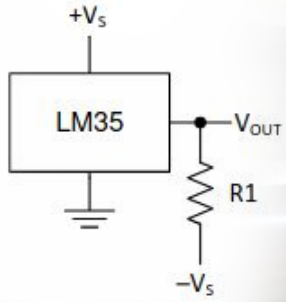
- Fournit une tension analogique proportionnelle à la température
  - 10 mV / °C
  - Plage de mesure de -55 °C à 150 °C
- Facile à utiliser pour une mesure entre 0 et 150°C
  - Tension de sortie :
    0V pour 0°C
    1,5 V pour 150°C
- Moins facile pour une mesure entre -55 et 150°C
  - Il faut une alimentation négative
  - Pour une alimentation négative de -5V, R1 = 100 kΩ
  - Arduino n'accepte pas une tension négative sur ses entrées, et ce n'est pas si simple d'adapter ...
  - Pour des températures négatives, la PT100 est plus accessible

Basic Centigrade Temperature Sensor (2°C to 150°C)



Facile à utiliser sur la plage 0 - 150°C

Full-Range Centigrade Temperature Sensor



Choose $R_1 = -V_S / 50\ \mu A$
$V_{OUT} = 1500$ mV at 150°C
$V_{OUT} = 250$ mV at 25°C
$V_{OUT} = -550$ mV at –55°C

Plus compliqué à utiliser sur la plage -55°C - 150°C

Capteur température LM35

- Pour lire une tension, pas besoin de bibliothèque …
  - AnalogRead() lit une tension entre 0 et 5V
  - Le convertisseur analogique - numérique traduit la tension en une valeur numérique sur 10 bits (entre 0 et 1023)
  - La valeur de tension obtenue est donc approximée (résolution ≈ 5mV)
- Rappel :
  - Une fonction pratique :
    map(val, from_low, from_high, to_low, to_high)
  - Exemple :
    « map(500, 0, 1023, 0, 5000) » fait la règle de trois : 500 sur une échelle de 0 à 1023 correspond à 1236 sur une échelle de 0 à 5000 (valeur retournée : 1236).

```
#define pinLM35 A0

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

void loop() {
  int tension_mv;
  char buf[31];

  // analogRead() retourne une tension entre 0 et 5V
  // codee sur 10 bits (0 : 0V,  1023 : 5V)
  tension_mv = (int)map(analogRead(pinLM35), 0, 1023, 0, 5000);

  // LM35 : 10 mV par °C
  float temperature = (float)tension_mv / 10.0;

  sprintf(buf, "Temperature : %.1f°C", temperature);
  Serial.println(buf);

  delay(1000); // 1 seconde
}
```

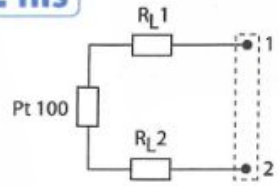
Vocabulaire : résistivité

- Tout matériau conducteur d'électricité est plus ou moins résistif
  - C'est à dire qu'il laisse passer le courant électrique sous l'effet d'une tension appliquée …
  - … mais pas sans le freiner un peu, ce qui limite l'intensité qui passe (I = U/R)
- En première approche : R = ρ x L / S
  - R : résistance (Ω)
    ρ : résistivité du matériau (Ωm)
    ρ_{cuivre} = 17,24 10^{-9} Ωm = 0,01724 Ω mm^2/m
    L : longueur (m)
    S : section (m^2)
- Par exemple, un fil en cuivre de section 1.5 mm^2 et de 1 m de longueur a une résistance de 0,011 Ω
  - Ça peut paraître négligeable mais quand on fait une mesure, ou quand on alimente une maison sur une grande distance …

Capteur de temp. PT100

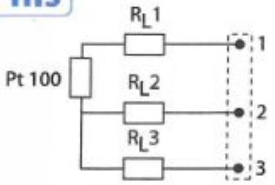
- **PT100 :**
  - Exploite la propriété du platine qui présente une résistivité variable avec la température
  - PT pour platine (élément résistif = fil de platine)
  - 100 pour 100 Ω à 0°C
- **Comme tout ce qui utilise la résistance**
  - Problématique de la longueur des fils qui ajoutent leur résistance à celle de la PT100 (proportionnelle à leur longueur et inversement proportionnelle à leur section)
  - 3 montages possibles : 2, 3 ou 4 fils
- **PT100 préférée : PT100 4 fils**
  - Parce que la mesure de tension ne consomme [pratiquement] pas de courant
  - Donc elle n'est pas sensible à la longueur des fils
  - Elle nécessite un circuit spécialisé pour faire circuler une intensité maîtrisée

2 fils — *le plus simple*



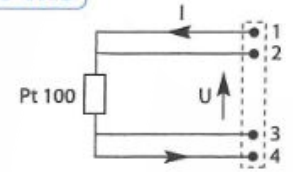
C'est la méthode de mesure la plus simple, mais les résistances de lignes (RL1 et RL2) sont en série avec l'élément sensible Pt 100. L'erreur correspond à RL1 + RL2, d'où un décalage de la température mesurée et de la température réelle. C'est le montage à éviter.

3 fils — *le plus utilisé*



Ce montage implique des résistances de lignes RL1-RL2-RL3 identiques. RL2+RL3 permettent de mesurer la résistance de lignes que l'on va soustraire à ce qui est mesuré aux bornes 1 et 2.

4 fils — *le plus précis*

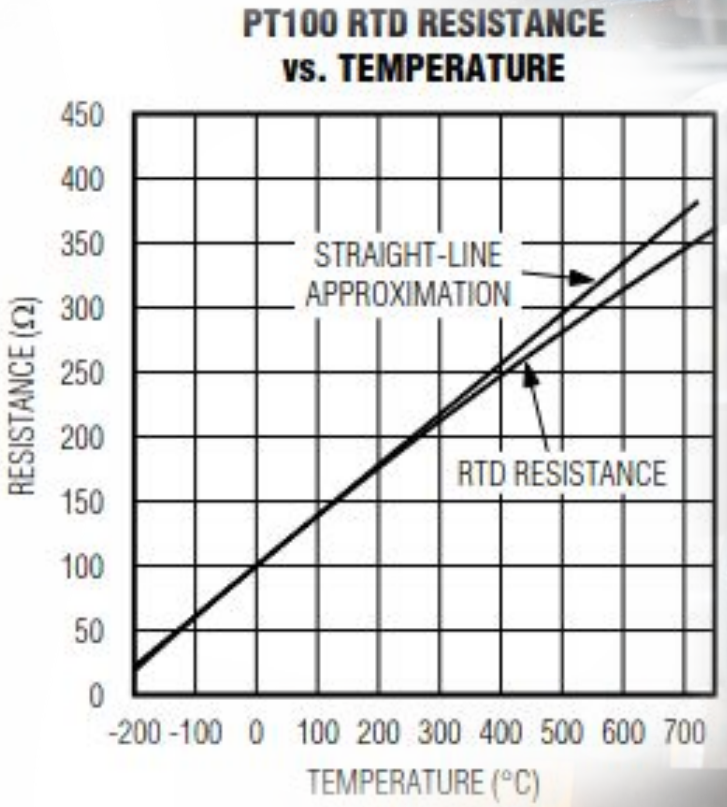


On fait passer un courant constant par les bornes 1 et 4 et l'on mesure directement la tension aux bornes de l'élément sensible Pt 100, ce qui permet complètement de s'affranchir des résistances de lignes.

(Ecole nationale supérieure de Lyon)

PT100 : R = f(T°C)

- Le courant qui traverse la PT100 ne doit pas excéder 1 mA
  - Pour limiter l'auto échauffement
- Quelques points
  - -200°C : 18,52 Ω
  - -20°C : 92,16 Ω
  - -10°C : 96,09 Ω
  - 0°C : 100 Ω
  - 10°C : 103,9 Ω
  - 20°C : 107,79 Ω
  - 100°C : 138,51 Ω
  - 850°C : 390,48 Ω



IDROLIK
AUTOMATISATION HYDRAULIQUE

MAX31865 pour PT100

- **MAX31865 : RTD-to-digital converter**
  - RTD : Resistance Temperature Detector
  - Optimisé pour PT100 (100 Ω à 0°C) ou PT1000 (1000 Ω à 0°C)
  - Convertisseur analogique-numérique 15 bits interne
  - Interface SPI (Serial Peripheral Interface)
- **3 principaux registres de 8 bits**
  - 0x00 (read) / 0x80 (write) : configuration
  - 0x01 (read) : RTD MSBs (Most Significant Bits)
  - 0x02 (read) : RTD LSBs (Less Significant Bits)
- **Extrait de la datasheet MAX31865 (Analog Devices)**
  - POR : Power On Reset (état au démarrage)

Table 1. Register Addresses and POR State

REGISTER NAME	READ ADDRESS (HEX)	WRITE ADDRESS (HEX)	POR STATE	READ/WRITE
Configuration	00h	80h	00h	R/W
RTD MSBs	01h	—	00h	R
RTD LSBs	02h	—	00h	R
High Fault Threshold MSB	03h	83h	FFh	R/W
High Fault Threshold LSB	04h	84h	FFh	R/W
Low Fault Threshold MSB	05h	85h	00h	R/W
Low Fault Threshold LSB	06h	86h	00h	R/W
Fault Status	07h	—	00h	R

Table 2. Configuration Register Definition

D7	D6	D5	D4	D3	D2	D1	D0
V_{BIAS} 1 = ON 0 = OFF	Conversion mode 1 = Auto 0 = Normally off	1-shot 1 = 1-shot (auto-clear)	3-wire 1 = 3-wire RTD 0 = 2-wire or 4-wire	Fault Detection Cycle Control (see Table 3)		Fault Status Clear 1 = Clear (auto-clear)	50/60Hz filter select 1 = 50Hz 0 = 60Hz

Table 4. RTD Resistance Registers Definition

REGISTER	RTD MSBS (01h) REGISTER								RTD LSBS (02h) REGISTER							
Bit	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
RTD Resistance Data	MSB	—	—	—	—	—	—	—	—	—	—	—	—	—	LSB	Fault
Bit Weighting	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	—
Decimal Value	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	—

MAX31865 pour PT100

- **La valeur codée sur 15 bits des registres 0x01 et 0x02**
  - Ne représente pas directement la résistance de la PT100
  - Elle correspond à une valeur numérique proportionnelle à cette résistance
  - Le MAX31865 mesure la résistance de la sonde (par exemple une PT100) en la comparant à une résistance de référence R_{REF}
  - Valeur sur 15 bits = $(R_{RTD} / R_{REF}) \times 2^{15}$
  - R_{REF} recommendée pour une PT100 : 430 Ω
- **Calcul de la température**
  - Lecture des registres 0x01 (MSBs) et 0x02 (LSBs)
  - Valeur sur 15 bits = (MSBs << 8 | LSBs) >> 1
  - R_{RTD} = valeur sur 15 bits x R_{REF} / 32768
  - Calcul de la température : équation Callendar-Van Dusen de la PT100 ($T°C = f(R_{RTC})$)

Bibliothèque MAX31865

- **Exemple d'utilisation :**
  - Outils + gérer les bibliothèques
  - Installation bibliothèque « Adafruit_MAX31865 »
  - Nécessite la bibliothèque « Adafruit_BusIO »
- **Notes :**
  - La bibliothèque Adafruit_MAX31865 n'utilise pas le signal « DRDY » (Data Ready) fourni par le MAX31865 via sa broche DRDY
    - Ça signifie qu'il demande la mesure et attend un temps « raisonnable » (une centaine de ms) avant de lire la mesure
  - Commande préprocesseur « #define mot1 mot2 »
    - Remplace mot1 par mot2 dans le texte avant de compiler
    - Rend le texte plus lisible
    - Ultra utile si on a un paramètre qui apparaît 1000 fois dans le texte, et qu'il est prévisible qu'on ait à le modifier
  - Les fonctions de la bibliothèque utilisées sont chargée dans la mémoire de l'Arduino qui gère le (les) MAX31865(s)
    - Un encouragement pour spécialiser des Arduinos par fonctions

```
#include <SPI.h>
#include <Adafruit_MAX31865.h>

#define RREF 430.0 // Resistance de reference
#define RNOMINAL 100.0 // Résistance nominale PT100

// Broches Arduino
#define MAX_CS   10 // SPI : Chip Select
#define MAX_MOSI 11 // SPI : SDI
#define MAX_MISO 12 // SDI : SDO
#define MAX_SCK  13 // SDI : SCLK

Adafruit_MAX31865 max = Adafruit_MAX31865(MAX_CS,
                        MAX_MOSI, MAX_MISO, MAX_SCK);

void setup() {
  Serial.begin(9600); // pour affichage console série
  Serial.println("Lecture PT100 via MAX31865");
  max.begin(MAX31865_4WIRE);  // IMPORTANT : mode 4 fils
}

void loop() {
  float temp;
  char buf[31];

  temp = max.temperature(RNOMINAL, RREF);
  sprintf(buf, "temperature = %f °C", temp);
  Serial.println(buf);

  delay(1000);
}
```

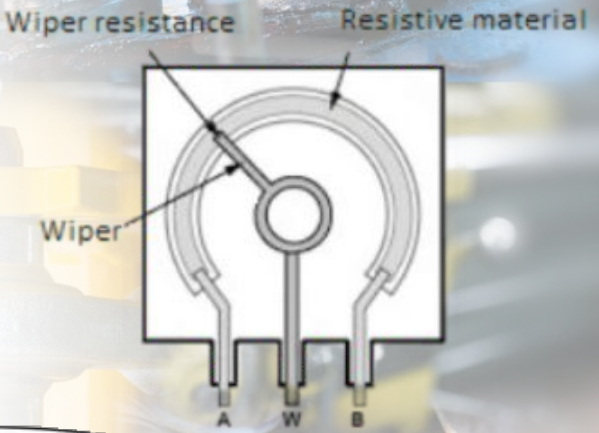
IDROLIK
AUTOMATISATION HYDRAULIQUE

PT100 + MAX31865

- Application pratique

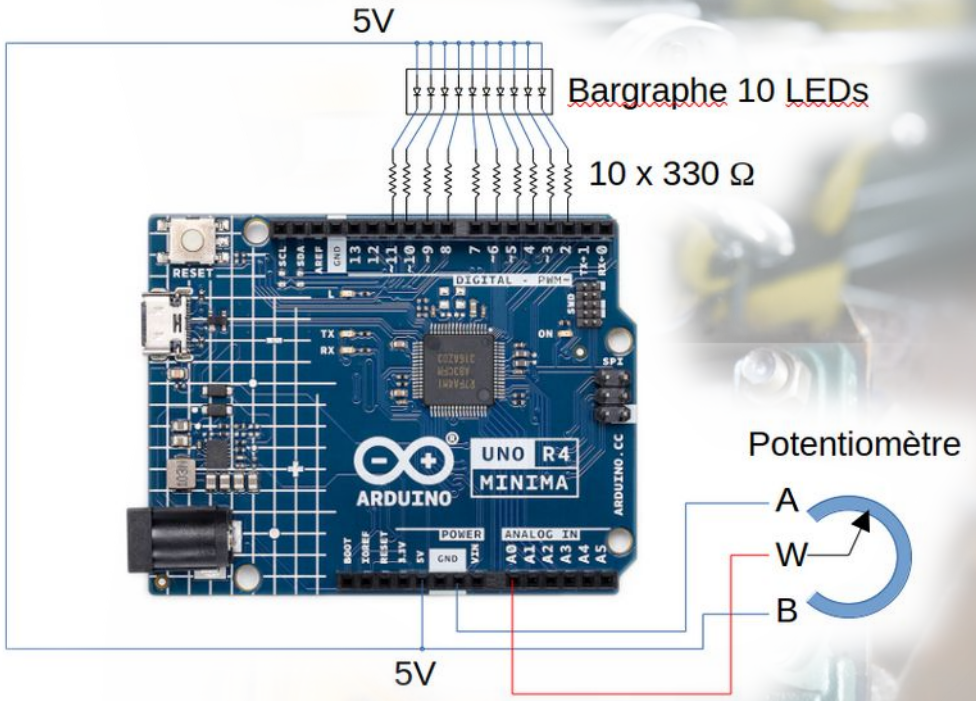
Le potentiomètre

- Un potentiomètre est une résistance réglable
- 2 types :
  - Potentiomètre rotatif
  - Potentiomètre linéaire
- Deux utilisations principales :
  - Réglage par l'utilisateur
    - Exemple : réglage du niveau de son d'un appareil audio
  - Retour d'information de déplacement
    - Exemple : information d'angle dans un servo de modélisme
- Principe :
  - Un balai est déplacé le long d'une piste résistive
  - La résistance entre le balai W et chacune des extrémités A et B de la piste est modifiée
  - La résistance entre les extrémités A et B est constante : elle doit être spécifiée lors de l'achat du potentiomètre

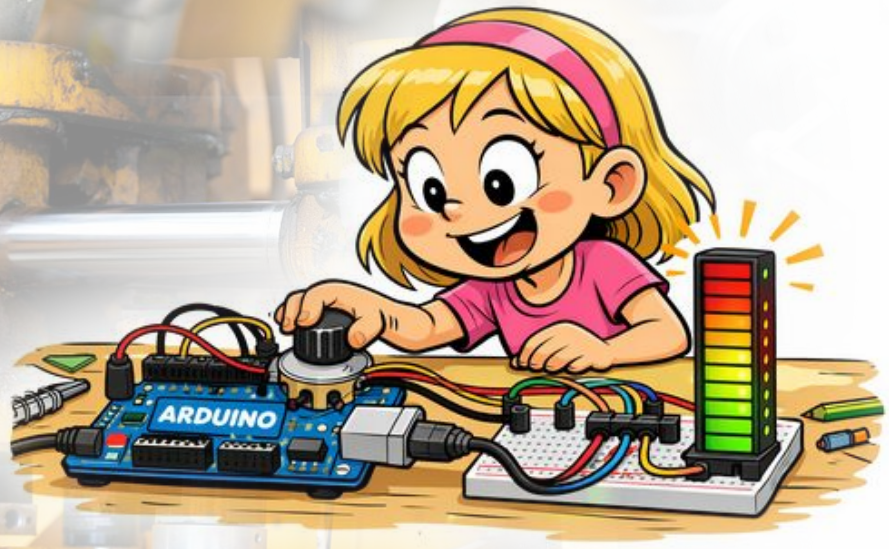


Le potentiomètre

- **Utilisation typique :**
  - On connecte la broche A au GND
  - On connecte la broche B au 5V
  - On connecte la broche milieu W à une entrée analogique de l'Arduino
- **Exemple : commande d'un bargraphe 10 LEDs avec un potentiomètre**
  - Bargraphe :
    - Connexion de chaque anode (+) de LED du bargraphe au 5V par l'intermédiaire d'une résistance 330 Ω
    - Connexion de chaque cathode (-) de LED à une entrée numérique de l'Arduino (0 allume la LED)
  - Potentiomètre :
    - Connexion de la broche A du au 5V de l'Arduino
    - Connexion de la broche B au GND de l'Arduino
    - Connexion de la broche W à une entrée analogique de l'Arduino



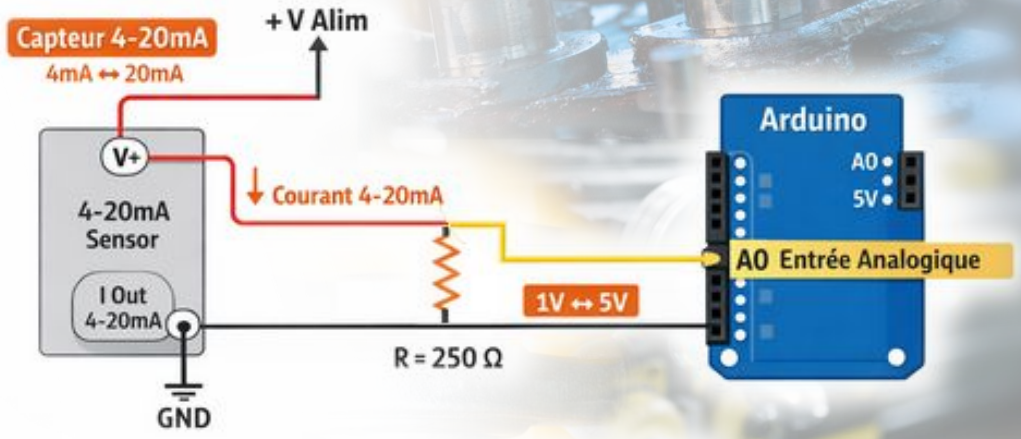
Le potentiomètre




```
int i, i_precedent, j, pin;

void setup() {
  // broches 2 a 11 Arduino pour les
  // 10 entrees du bargraphe
  for (i = 2; i < 12; i++) {
    pinMode(i, OUTPUT);
  }
  pinMode(A0, INPUT);

  i_precedent = 0;
}

void loop() {
  i = map(analogRead(A0), 0, 1023, 0, 5000);
  // i = tension A0 en mV

  if (i != i_precedent) {
    i_precedent = i;
    pin = 2;
    for (j = 200; j < 5000; i += 500) {
      if (i > j) {
        digitalWrite(pin, HIGH);
      } else {
        digitalWrite(pin, LOW);
      }
      pin ++;
    }
  }
  delay(200);
}
```

Boucle 4-20mA

- Certains capteurs fournissent leur information sous la forme d'une intensité électrique
  - Pas de pertes par résistance
  - Le capteur adapte le courant de sortie de manière à ce qu'il soit proportionnel à la mesure
- Lorsque la mesure est à zéro, il fait quand-même passer 4mA
  - Si la commande ne voit pas passer au moins 4 mA, le capteur n'est pas alimenté, il est endommagé, ou un fil est coupé
- On peut simplement connecter une résistance de 250 Ω entre la sortie du capteur et le GND de l'Arduino et mesurer la tension avec une entrée analogique
  - 4 mA : tension de 1V aux bornes de la résistance
  - 20 mA : tension de 5V aux bornes de la résistance
- Il existe des modules 4-20mA industriels plus précis, moins sensibles aux perturbations et qui évitent de griller l'Arduino en cas de surintensité.



On trouve aussi...

- **... des capteurs analogiques qui communiquent par**
  - **Modbus (relativement simple)**
    - Bus utilisé dans le bâtiment (CO2, particules fines, humidité, température...)
    - Modbus = tableau Excel distant
      Tu demandes une case
      On te renvoie la valeur
  - **IO link (nettement plus « usine à gaz »)**
    - Standard ouvert d'un consortium industriel mais dépend souvent d'outils fabricants
    - Permet de lire des mesures et des informations d'état, et de configurer le capteur
    - Couche physique standard mais pile logicielle complète non triviale et souvent propriétaire
    - Chaque capteur a un fichier IODD (données, paramètres, unités...)
    - Modules matériels spécifiques IO-Link Master ...