Lab 1

Simples architectures IoT basées sur Arduino Nano

Avant de commencer le travail sur les exercices et le projet, nous devons préparer et comprendre notre environnement de travail basé sur l'Arduino IDE et Linux OS.
D'abord, nous devons télécharger la version la plus récente d'Arduino IDE et l'installer sur notre système (Linux). Notez que la récente version d'Arduino IDE s'exécute sur les deux plates-formes architecturales: **Intel x86** et **ARM**.
Le logiciel requis peut être trouvé sur cette page: **`https://www.arduino.cc/fr/Main/Software.`**
Après le téléchargement, nous devons décompresser et installer l'IDE Arduino.
Vous trouverez un répertoire principal Arduino et les sous-répertoires des bibliothèques et des croquis.
L'installation initiale est suffisante pour commencer les premières conceptions avec des cartes compatibles Arduino: **Uno**, **Nano**, **Mega**.
L'utilisation de cartes (processeurs) plus avancés tels que ESP8266 nécessite l'installation de ces dispositifs via les outils **Preferences** puis **Board Manager**.
Dans la **première architecture** avec Arduino IDE, nous allons utiliser une carte Arduino Nano avec un capteur - DTH11/22 comme entrée et une LED RVB comme sortie.

La **deuxième architecture** avec Arduino IDE, utilise la carte Arduino Nano avec des capteurs température/humidité/lumière et la carte d'extension Ethernet pour envoyer les données via l'Internet vers un serveur ThingSpeak.

1.1 Arduino Nano avec capteur analogique de l'intensité de la lumière et un indicateur LED tricolore

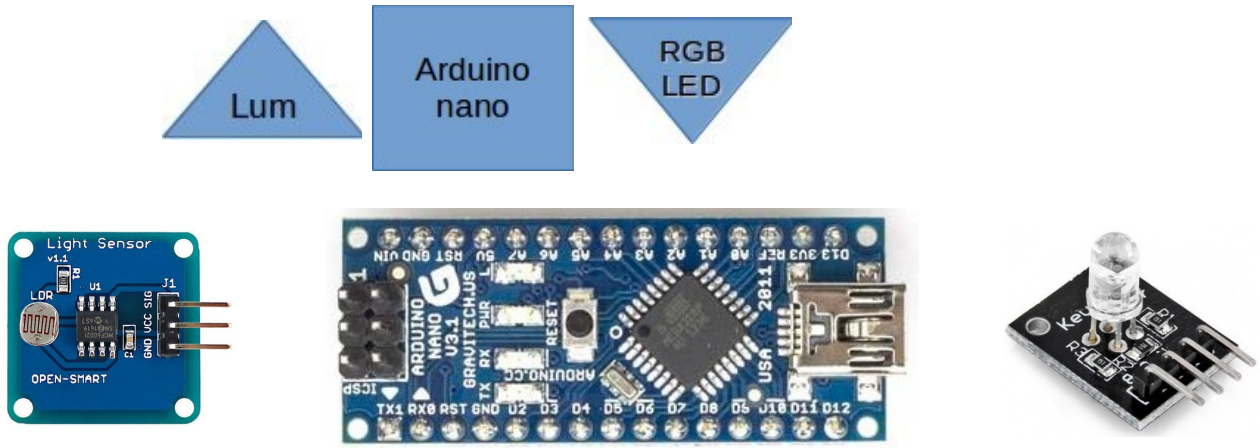


Figure 1.1. Schéma symbolique de l'architecture et les composants : capteur de luminosité – Nano – LED RGB

Connexions des composants :

Capteur	Nano
GND	GND
VCC	5V
SIG	A0 (convertisseur ADC résolution 10 bits)

LED-RGB	Nano
GND	GND
B	D6
G	D7
R	D8

Le code Arduino sans la diode RGB :

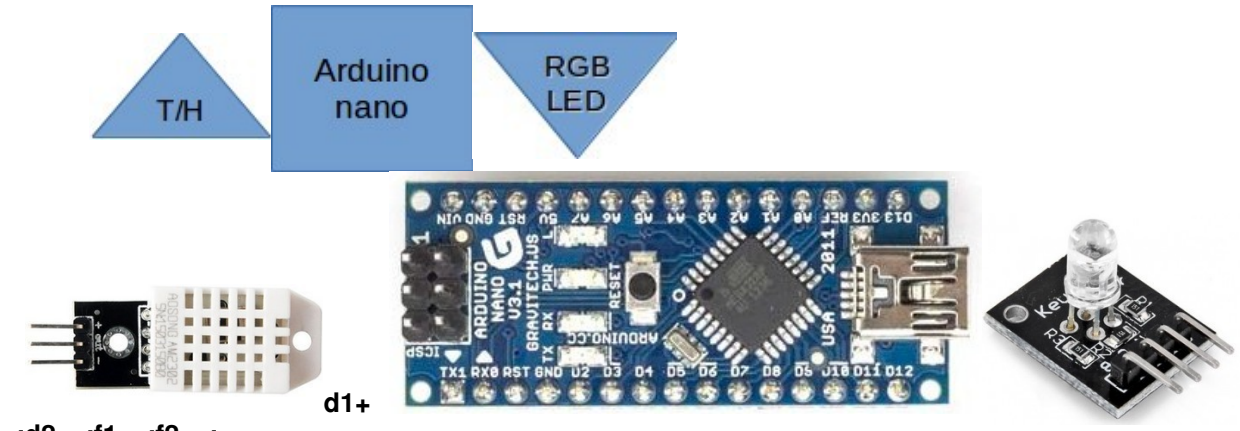
```
int photocellPin = 0;     // the cell and 10K pulldown are connected to A0
int photocellReading;     // the analog reading from the sensor divider

void setup(void) {
  Serial.begin(9600);
 pinMode(6, OUTPUT);  // BLUE
 pinMode(7, OUTPUT);  // GREEN
 pinMode(8, OUTPUT);  // RED
}

void loop(void) {
  photocellReading = analogRead(photocellPin);
  Serial.print("Analog reading = ");
  Serial.println(photocellReading);     // the raw analog reading
  delay(100);
}
```

Exercice 1.1 :

Testez l'exemple donné, Choisissez les intervalles de la luminosité à signaler sur la diode RGB (0-20 – RED, 21-100 – GREEN, 101 – BLUE)

1.2 Arduino Nano avec capteur DHT 11/22 et un indicateur LED tricolore

1.2.1 Architecture et connexions



d1+

+;d2++;f1++;f2++;

Figure 1.2 Schéma symbolique et les composants de l'architecture : capteur de température/humidité – Nano – LED RGB

Connexions :

Capteur DHT	Nano
GND	GND
VCC	5V
Dout	D5

1.2.2 Code Arduino :

```
#include "DHT.h"
#define DHTPIN 5     // what digital pin we're connected to ?
#define DHTTYPE DHT11   // DHT 11
//#define DHTTYPE DHT22   // DHT 22
// Connect pin VCC of the sensor to +5V
// Connect pin 5 of the sensor to whatever your DHTPIN is
// Connect pin GND
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  pinMode(6, OUTPUT);  // BLUE
  pinMode(7, OUTPUT);  // GREEN
  pinMode(8, OUTPUT);  // RED
  Serial.println("DHTxx test!");
  dht.begin();
}
void loop() {
    digitalWrite(6,LOW);digitalWrite(7,LOW);digitalWrite(8,LOW);
  // Wait a few seconds between measurements.
  delay(2000);
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
```

```
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print("Humidity: ");Serial.print(h);Serial.print("%\t");
  Serial.print("Temperature: ");Serial.print(t);Serial.println("*C");
  if(t<21.0) { digitalWrite(6, HIGH);  }     // set LED BLUE
  ..  // a completer
}
```

1.2.3 Protocole DHT

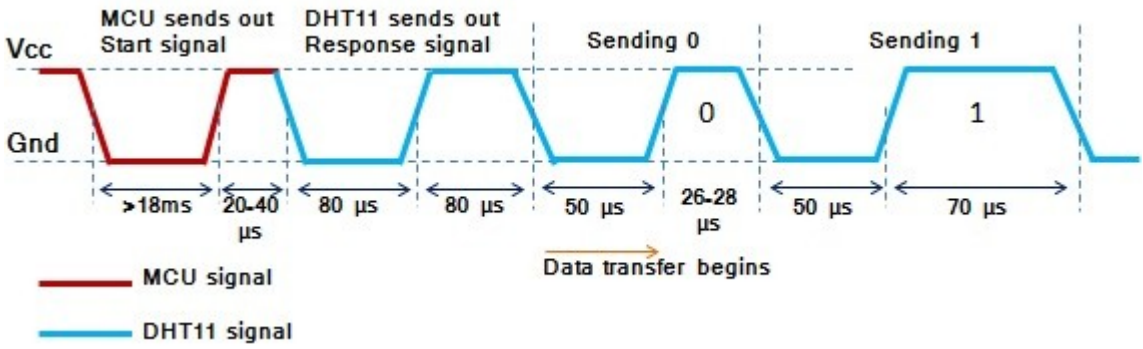


Figure 1.3. Le diagramme du protocole DHT11/22

La bibliothèque **DHT.h** contient les fonctions de communication avec les capteurs DHT11/22 qui opèrent selon le protocole spécifique.
Le chronogramme ci-dessus décrit le protocole de transfert de données entre un MCU et le capteur DHT11. Le MCU déclenche une transmission de données en émettant un signal "**Start**".
La broche MCU doit être configurée en sortie.
Le MCU envoie d'abord sur la ligne de données un signal **LOW** pendant au moins 18 ms, puis envoie **HIGH** pour les 20-40µs suivants avant de la libérer.
Ensuite, le capteur répond au signal "**Start**" de MCU en émettant le signal **LOW** pendant 80µs suivi d'un signal HIGH qui dure aussi 80µs.
Après la détection du signal de réponse du capteur, le MCU devrait être prêt à recevoir les données du capteur.
Le capteur envoie alors 40 bits (5 octets) de données en continu sur la ligne de données.
Notez que lors de l'émission des octets, le capteur envoie le bit le plus important en premier.

Exercice 1.2 :

a. Etudier le protocole du capteur DHT(11/22)
b. Compléter l code Arduino présenté ci-dessus – il s'agit d'afficher la couleur bleue pour la température inférieure à 21°C, la verte pour la température entre 21°C et 25°C et la rouge au delà 25°C

1.3 Arduino Nano, capteurs et le bus I2C pour l'afficheur OLED

1.3.1 Architecture et connexions

Dans cet exercice nous utilisons un Arduino Nano et le connecter avec deux composants partageant le bus I2C.

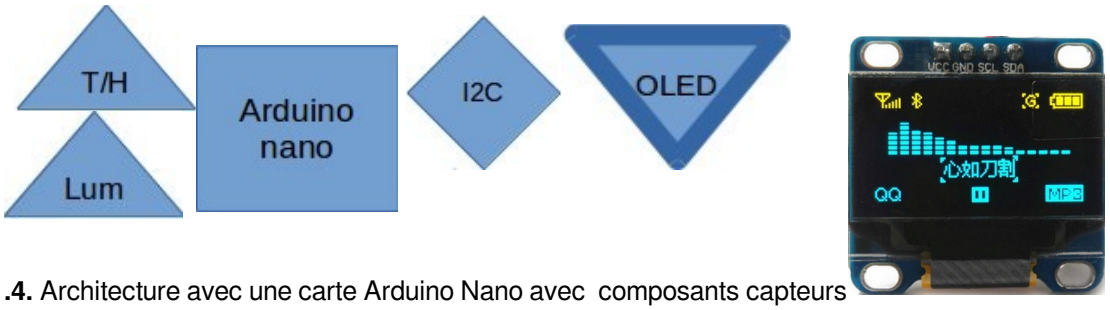


Figure 1.4. Architecture avec une carte Arduino Nano avec composants capteurs et l'écran OLED

Connexions :

Ecran OLED	Nano
GND	GND
VCC	5V
SDA	A4
SCK	A5 (pins I2C par défaut)

1.3.2 Bus et protocole I2C et le test de la présence

Dans le protocole I2C les échanges ont toujours lieu entre un seul maître et un (ou tous les) esclave(s), toujours à l'initiative du maître (jamais de maître à maître ou d'esclave à esclave).
La connexion est réalisée par l'intermédiaire de deux lignes :

- **SDA** (Serial Data Line) : ligne de données bidirectionnelle,
- **SCL** (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

Il ne faut également pas oublier la masse qui doit être commune aux équipements.
Les 2 lignes sont tirées au niveau de tension V_{DD} (ici 3.3 V) à travers des résistances de pull-up (R_P).

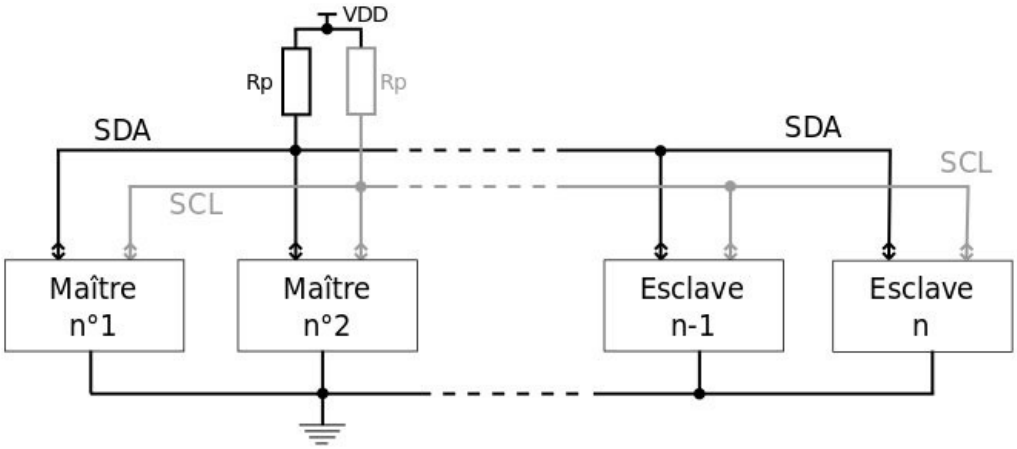


Figure 1.5. Connexions électriques du bus I2C

Le protocole d'échange

Le bus ne dispose que de deux lignes SCL et SDA. Les différents états que vont pouvoir prendre ces lignes vont permettre d'exprimer les étapes d'un échange. On parle de conditions car il s'agit en fait de changements d'état.
Pour prendre un exemple, un maître veut faire un échange avec un esclave.

1. Le bus étant libre (**free time**), le maître prend le bus en imposant une condition de démarrage (**start condition**).
2. Le maître envoie l'**adresse de l'abonné** esclave visé et le sens du transfert **read/write**. Pour cela, il

génère le signal d'horloge SCL périodique et transmet l'adresse bit après bits (**bit transfer**) sur SDA.
3. L'esclave qui se reconnaît renvoie un acquittement (**acknowledge** ou **ACK**) sur **SDA**.
4. Le transfert se poursuit par l'envoi de donnée, nous allons voir comment, et après chaque envoi par l'émetteur, le récepteur envoie un acquittement.
5. L'échange se termine normalement à l'initiative du maître (sauf exception) et s'achève par une condition d'arrêt (**stop condition**).
6. Éventuellement, le maître peut remplacer la condition d'arrêt par une condition de redémarrage (**restart condition**) dans le cas où il veut changer d'abonné ou de sens d'échange.
7. Les bits sont transférés par paquet de 8, bit de poids fort en premier (à l'inverse du rs232), qu'il s'agisse des données ou des adresses. Le récepteur envoie un acquittement après chaque envoi de 8 bits par l'émetteur. l'acquittement est le 9ième bit.

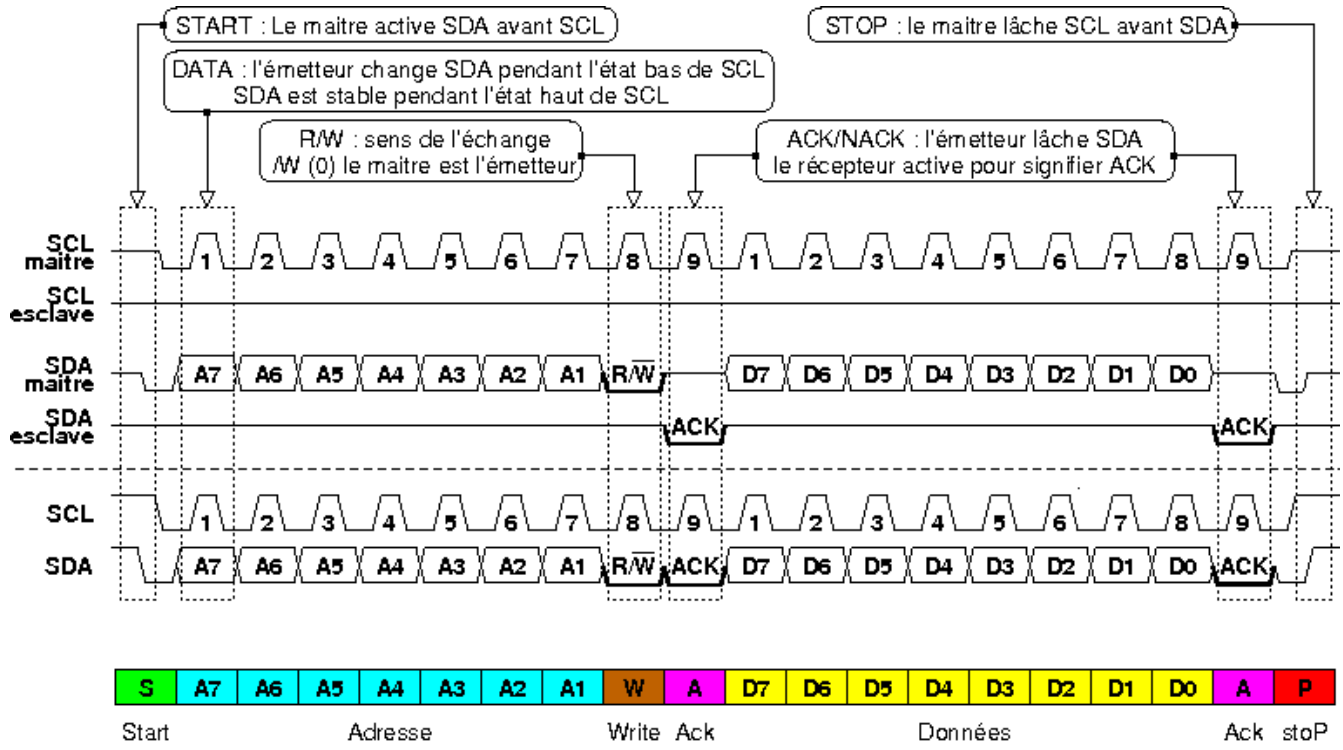


Figure 1.6. L'envoi de l'adresse et des donnés sur le bus I2C

L'adressage

Après avoir émis une condition de départ, le maître indique avec quel esclave il souhaite se connecter, pour cela il envoie sur le bus l'adresse de l'esclave composé de 7 bits (donc 128 adresses différentes), puis un dernier bit indiquant le sens du transfert : Lecture (1) ou Écriture (0). Tous les esclaves scrutent le bus et voient la condition de départ, celui qui reconnait son adresse envoie un acquittement (SDA à 0).

1.3.4 Code Arduino pour tester la présence des dispositifs connectés sur le bus I2C

Dans la première partie de cet exercice nous allons tester la présence d'un dispositif connecté sur le bus I2C de la carte Arduino Nano.
Pour ce test il faut connecter l'écran OLED sur les broches A4 - SDA, A5 – SCL avec l'alimentation de 5V.

```
#include <Wire.h>   // A4 - SDA, A5 – SCL sur Nano
void setup()
{
  Wire.begin();
  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}
```

```
 void loop()
{
  byte error, address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++ )  // I2C address scaner
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
     if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
      Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");
      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16) Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");
  delay(5000);           // wait 5 seconds for next scan
}
```

Le résultat affiché sur le terminal :

```
I2C Scanner
Scanning...
I2C device found at address 0x3C  !
done
```

1.3.5 Code Arduino pour l'architecture de la figure 1.4, incluant deux capteurs et un afficheur (écran OLED)

```
#include "DHT.h"
#define DHTPIN 5     // what digital pin we're connected to
#define DHTTYPE DHT11   // DHT 11
#include <Wire.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
DHT dht(DHTPIN, DHTTYPE);
```

```
#if (SSD1306_LCDHEIGHT != 64)
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

int photocellPin = 0;     // A0
int photocellReading;
float t,h;

int readDTH()
{
  h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  t = dht.readTemperature();
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
  return 1;
  }
  return 0;
}

char buf[32];
int res;
void affiche(void)
{
  display.setTextSize(2);
  display.setTextColor( WHITE );
  display.clearDisplay();
  res=readDTH();
  display.setCursor(4,0);
  sprintf(buf,"lumi:%d",photocellReading);
  display.println(buf);
  display.setCursor(4,20);
  sprintf(buf,"temp:%d",(int)t);
  display.println(buf);
  display.setCursor(4,40);
  sprintf(buf,"humi:%d",(int)h);
  display.println(buf);
  display.display();
}

void setup()
{
  Serial.begin(9600);
  dht.begin();
  // Initialise la communication I²C à l'adresse 0x3C.
  display.begin(SSD1306_SWITCHCAPVCC,0x3C);
  display.clearDisplay();
}

void loop()
{
  delay(1000);
  photocellReading = analogRead(photocellPin);
  Serial.print("Analog reading = ");
  Serial.println(photocellReading);     // the raw analog reading
  affiche();
  delay(1000);
}
```

Exercice 1.3:

1. Trouver l'adresse du capteur HTU21D dans fichiers HTU.h et/ou HTU.cpp.
2. Le capteur HTU21D fournit des valeurs précises de température et d'humidité avec deux décimales enregistrées sur le format à virgule flottante.
Le code ci-dessus avec la fonction d'affichage affiche uniquement des valeurs entières.
Modifiez ce code pour afficher les décimales.