Laboratoires IoT de base

Mise en oeuvre des architectures IoT à la base de IoT-DevKit de SmartComputerLab

Contenu

Laboratoires IoT de base

Mise en oeuvre des architectures IoT à la base de IoT-DevKit de SmartComputerLab

0. Introduction

Dans les laboratoires IoT nous allons mettre en œuvre plusieurs architectures IoT intégrant les terminaux (T), les passerelles (*gateways* - G), et les serveurs (S) IoT type **ThingSpeak** ou brokers **MQTT**. Le développement sera réalisé sur les cartes **IoTDevKit** de **SmartComputerLab**.

Le kit de développement contient une carte de base "basecard" pour y accueillir une unité centrale et un ensemble de cartes d'extension pour les capteurs, les actuateurs et les modems supplémentaires.
L'unité centrale est une carte équipée d'un **SoC ESP32** et d'un modem **LoRa** (*Long Range*).

Le premier laboratoire permet de préparer l'environnement de travail et de tester l'utilisation des capteurs connectés sur le bus I2C (température/humidité/luminosité/pression) et d'un afficheur. Pour nos développements et nos expérimentations nous utilisons le **IDE Arduino** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale.

Le deuxième laboratoire est consacré à la pris en main du modem WiFi intégré dans la carte principale (**Heltec ESP32-WiFi-LoRa**). La communication WiFi en mode station et un client permet d'envoyer les données des capteurs vers un serveur **WEB**. Dans notre cas nous utilisons le serveur de type **ThingSpeak**; (**`ThingSpeak.fr/com`**). Ces serveurs sont accessibles gratuitement, mais leur utilisation peut être limitée un temps (le cas de **`ThingSpeak.com`**).

Le troisième laboratoire permet de découvrir le protocole **MQTT** et d'expérimenter avec plusieurs exemples de **brokers** et **clients** basés sur différentes bibliothèques. Dans le laboratoire vous pouvez communiquer et échanger vos donnés IoT à distance via les broker externes.

Le quatrième laboratoire est orienté sur la technologie de communication **Bluetooth** – **Bluetooth Classic** (**BT**) et **Bluetooth Low Energy** (**BLE**). Vous allez expérimenter avec cette technologie à l'aide de vos smartphones. Vous allez développer une passerelle **Bluetooth-WiFI** sur votre **DevKit**.

Le cinquième laboratoire cible développement de simples **serveurs WEB** s'exécutant sur notre **DevKit**. Ces serveurs peuvent fonctionner en mode de station (**`STA`**) ou en mode Point d'Accès (**`AP`**). Il peuvent utiliser exclusivement la mémoire interne ou un contenu externe enregistré sur un **carte SD**.

Le sixième laboratoire introduit la programmation **OTA** (**Over The Air**). Ce mode de programmation permet de charger à distance les nouvelles applications via une connexion WiFi.

Après ces laboratoires préparatifs nous vous proposons plusieurs **laboratoires thématiques** qui permettrons de concevoir et développer différentes sortes d'applications IoT sous la forme de **mini-projets**. On vous présentera quelques exemples des architectures IoT dans les domaines d'application différents tels que la messagerie, le sécurité, l'environnement, le commerce,etc.

Vous pouvez vous inspirer de ces laboratoires ou de proposer d'autres de complexité comparable.

Nous vous fournirons (selon les disponibilités) des cartes d'extension et de capteurs/actionneurs nécessaires pour la réalisation de ces projets.

0.1 ESP32 Soc – une unité avancée pour les architectures IoT

ESP32 est une unité de microcontrôleur avancée conçue pour le développement d'architectures IoT. Un ESP32 intègre deux processeurs RISC 32-bit fonctionnant à 240 MHz et plusieurs unités de traitement et de communication supplémentaires, notamment un processeur ULP (**Ultra Low Power**), des modems WiFi / Bluetooth /BLE et un ensemble de contrôleurs E/S pour bus série (UART, I2C, SPI). , ..). Ces blocs fonctionnels sont décrits ci-dessous dans la figure suivante.

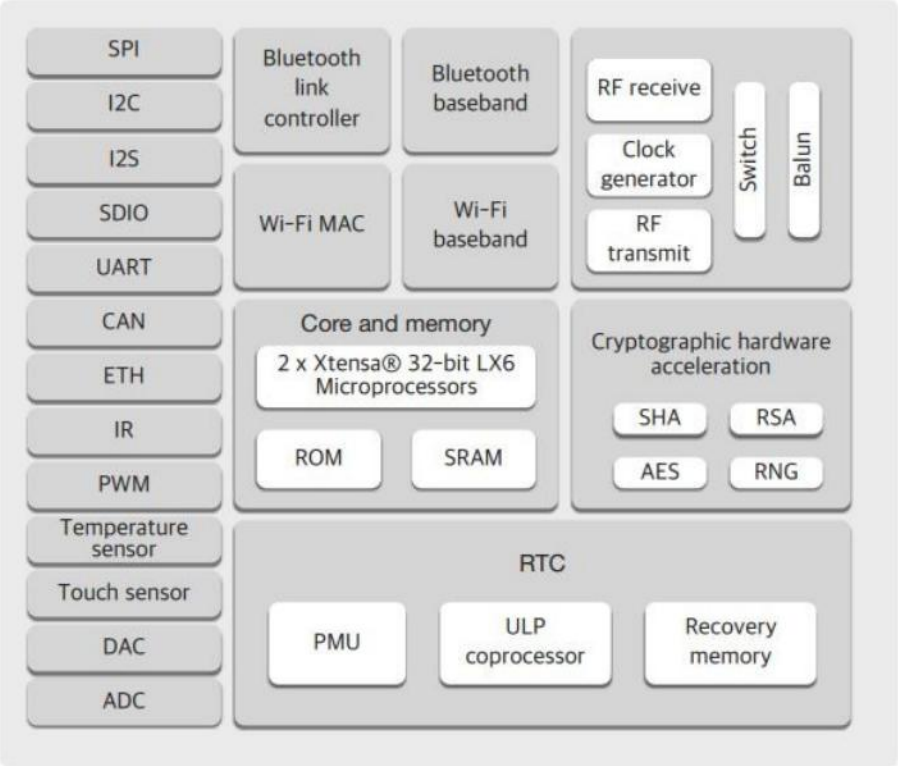


Figure 0.1 ESP32 SoC – architecture interne

0.2 Carte Heltec WiFi LoRa

De nos jours, les SoC ESP32 sont intégrés dans un certain nombre de cartes de développement qui incluent des circuits supplémentaires et des modems de communication. Notre choix est la carte **ESP32-Heltec WiFi-LoRa** qui intègre le modem LoRa **Semtech S1276/78** et (éventuellement) un écran . La figure suivante montre la carte **ESP32-Heltec WiFi-LoRa**

Figure 0.2 Carte MCU **ESP32-Heltec WiFi-LoRa**

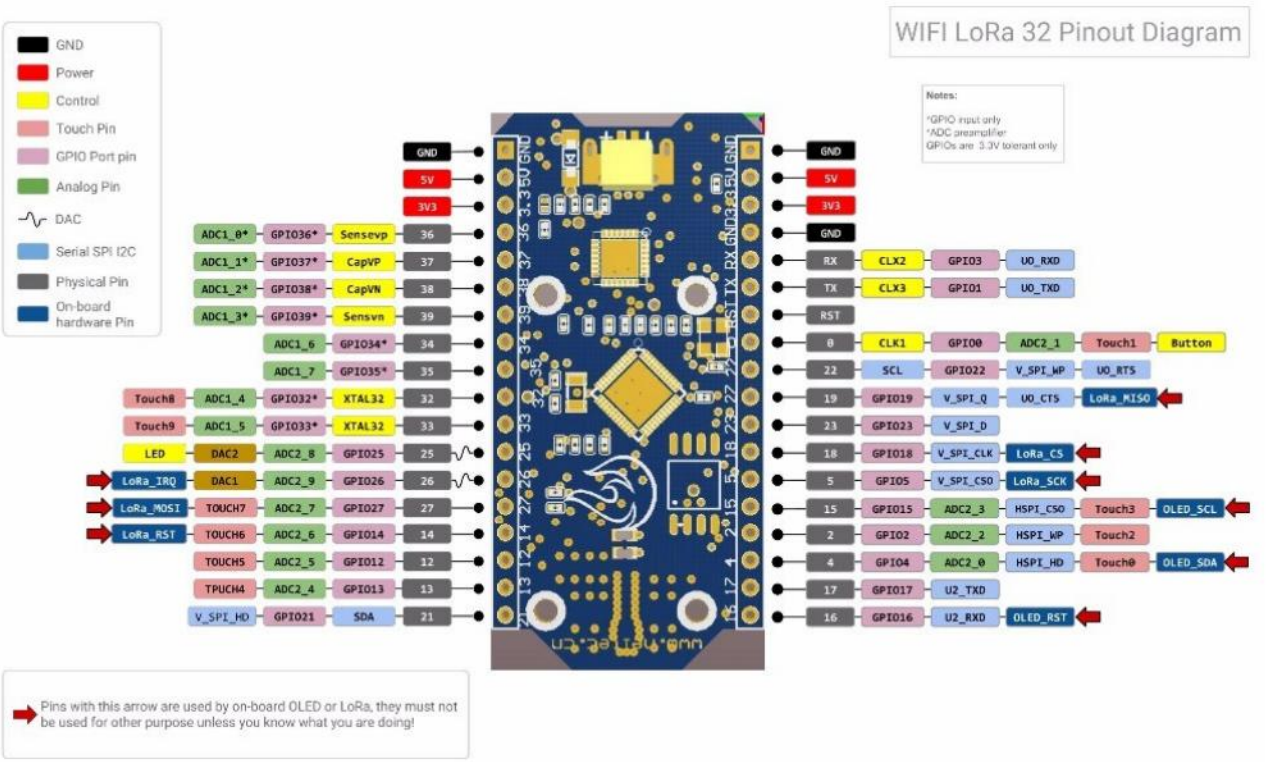


Figure 0.3. Carte **ESP32-Heltec WiFi-LoRa** et son *pin-out*

Comme nous pouvons le voir sur la figure ci-dessus, la carte Heltec expose des broches 2x18. Certaines de ces broches sont utilisées pour connecter le modem LoRa SX1276/8 (SPI) et l'afficheur OLED (I2C), d'autres broches peuvent être utilisées pour connecter des composants externes tels que des capteurs ou des actionneurs.

0.3 IoT DevKit une plate-forme de développement IoT

Une intégration efficace de la carte Heltec sélectionnée dans les architectures IoT nécessite l'utilisation d'une plate-forme de développement telle que IoT DevKit proposée par **SmartComputerLab**.
L'**IoTDevKit** est composé d'une carte de base et d'un grand nombre de cartes d'extension conçues pour l'utilisation efficace des bus de connexion et de tous les types de capteurs et d'actionneurs.

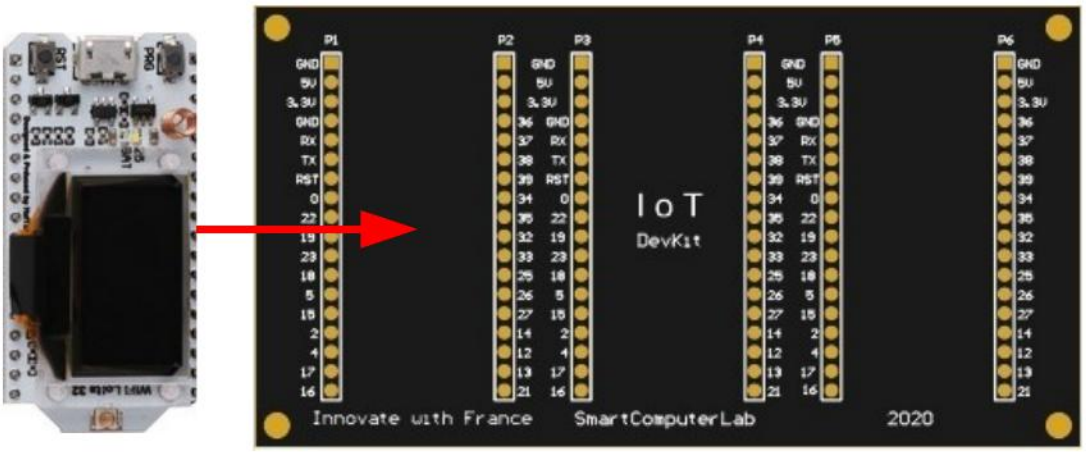
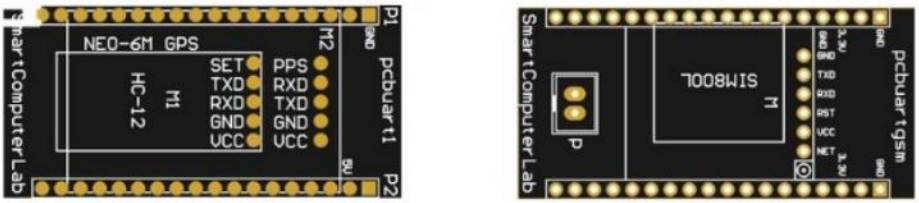


Figure 0.4. **IoT DevKit**: carte de base (type bridge) avec l'unité principale et les emplacements pour les cartes d'extension

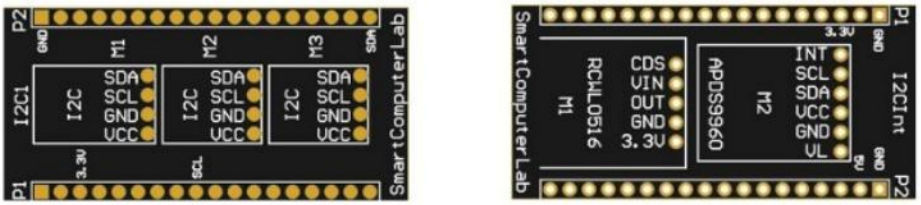
0.3.1 Cartes d'extension simples – quelques exemples

Figure 0.5. **IoT DevKit**: cartes d'extension, exemple d'implantation sur les d'extension **I2C** avec capteurs **BH1750** et **HTU21D** et une carte d'extension **UART** avec module GPS NEO-M6

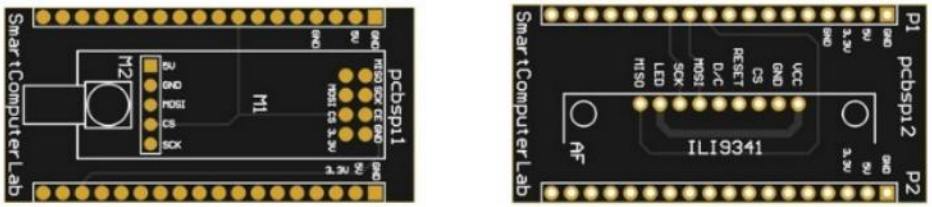
Cartes d'extension pour le bus **UART** (deux exemples):



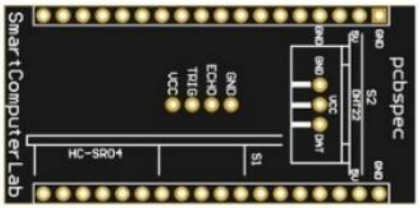
Cartes d'extension pour le bus **I2C** (deux exemples):



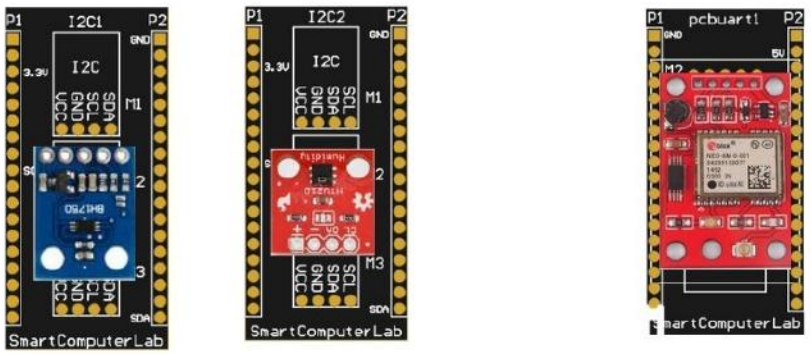
Cartes d'extension pour le bus **SPI** (deux exemples):



Cartes d'extension spécifiques (un exemple):

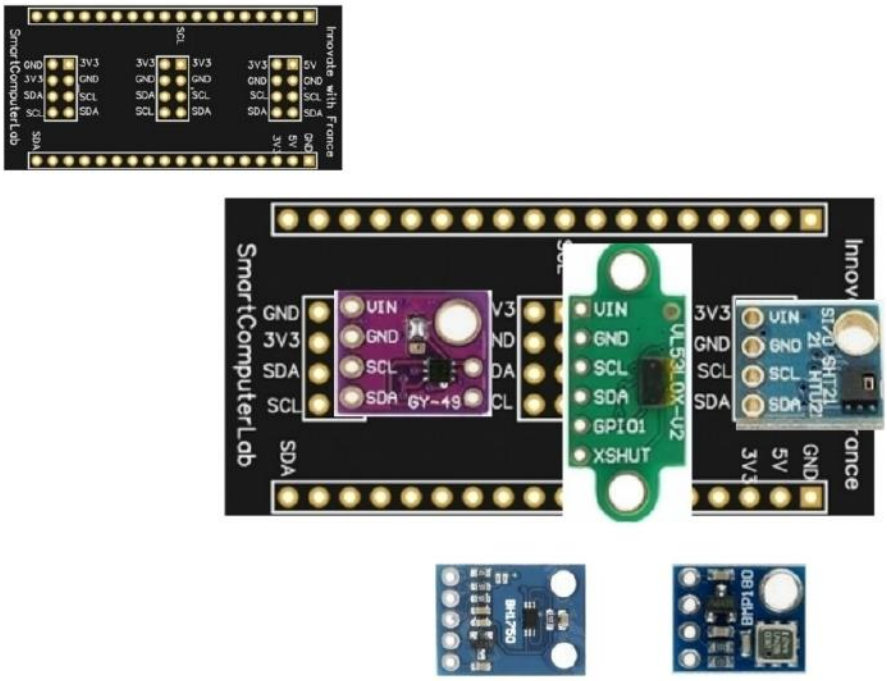


Cartes d'extension **I2C** avec capteurs **BH1750** et **HTU21D** et une carte d'extension **UART** avec module GPS NEO-M6

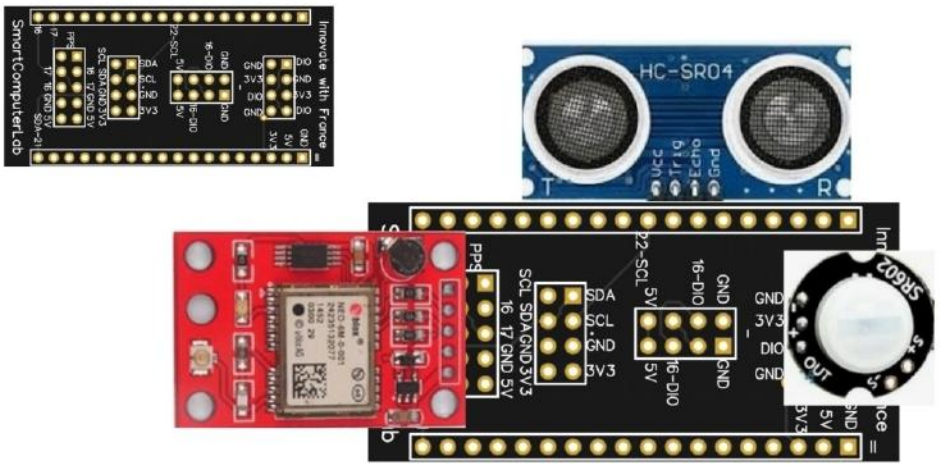


0.3.2 Cartes d'extension multi-capteurs – quelques exemples

Carte **multi-capteurs I2C**



Carte **multi-capteurs** : PIR, Echo, GPS (**UART**), **I2C**



0.4 L'installation de l'Arduino IDE sur un OS Ubuntu

Pour nos développements et nos expérimentations nous utilisons l'**IDE Arduino** comme outils de programmation et de chargement des programmes dans la mémoire flash de l'unité centrale.
Afin de pouvoir développer le code pour les application nous avons besoin d'un environnement de travail comprenant; un PC, un OS type Ubuntu 16.04LTS, l'environnement Arduino IDE, les outils de compilation/chargement pour les cartes ESP32 et les bibliothèque de contrôle pour les capteurs, actuateurs (par exemple **relais**), et les modems de communication.

Pour commencer mettez le système à jour par :

```
sudo apt-upgrade  et  sudo apt update
sudo apt-get install python-serial python3-serial
```

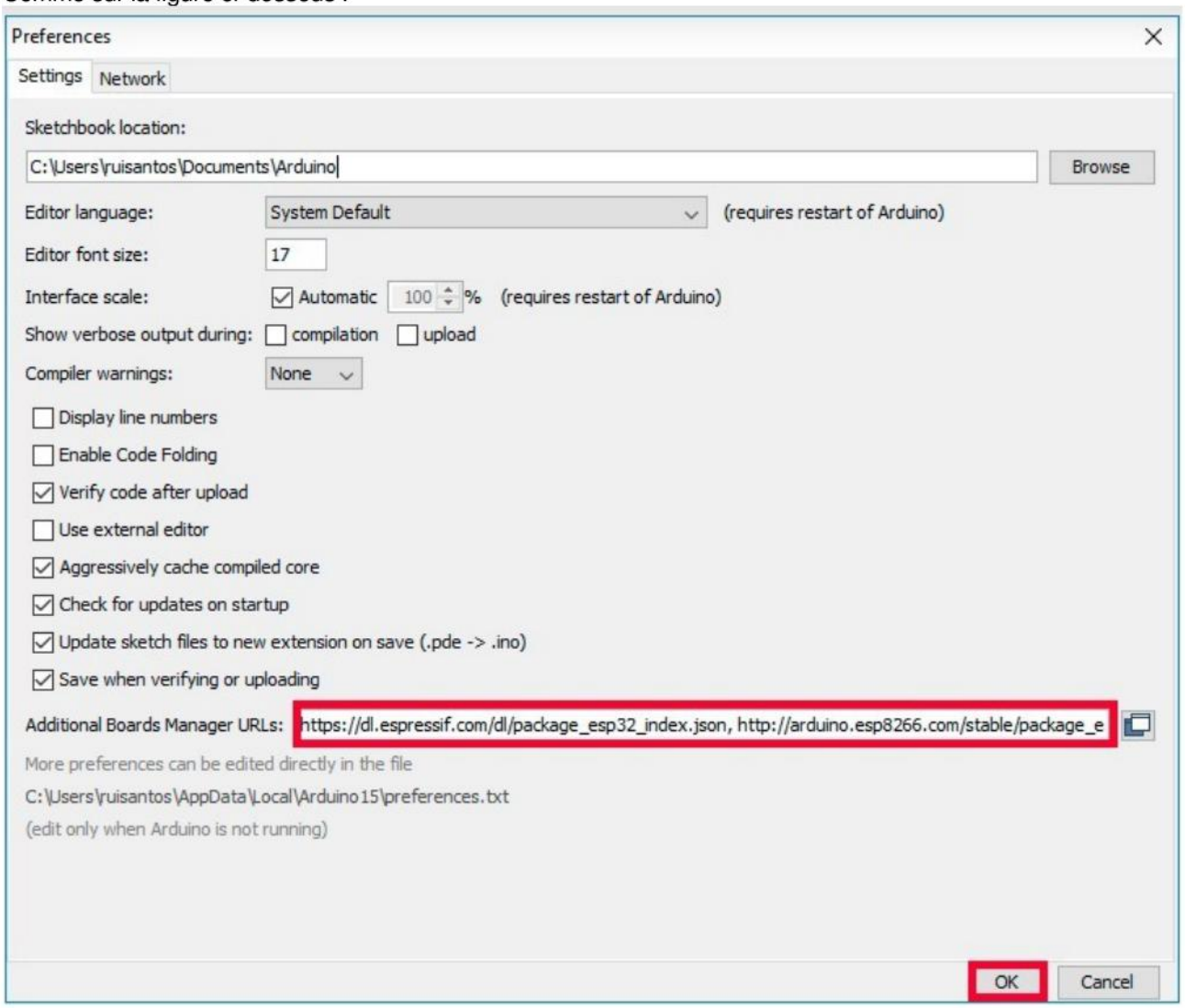
Ensuite il faut installer le dernier Arduino IDE à partir de **`https://www.arduino.cc`**

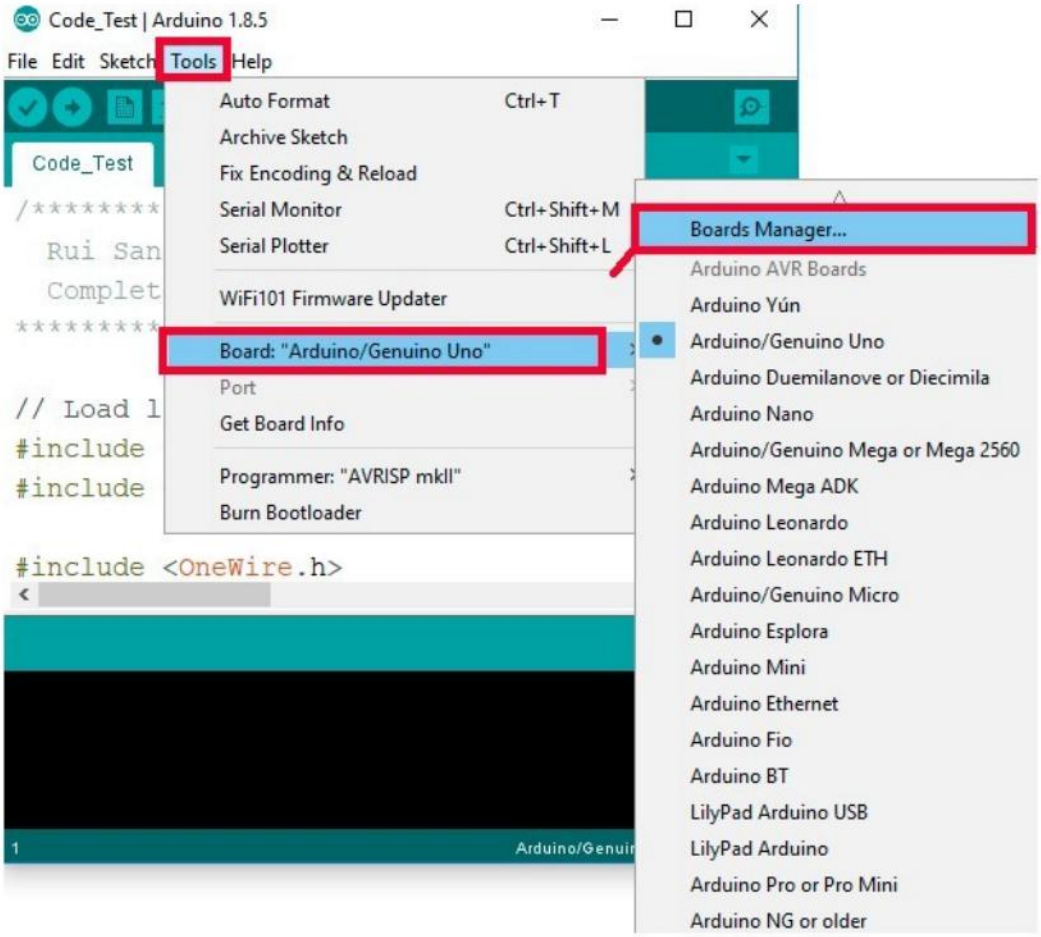
0.4.1 Installation des nouvelles cartes ESP32 et ESP8266

Après son installation allez dans les **Preferences** et ajoutez deux **Boards Manager URLs** (séparées par une virgule) :

`https://dl.espressif.com/dl/package_esp32_index.json`,
`http://arduino.esp8266.com/stable/package_esp8266com_index.json`

Comme sur la figure ci-dessous :





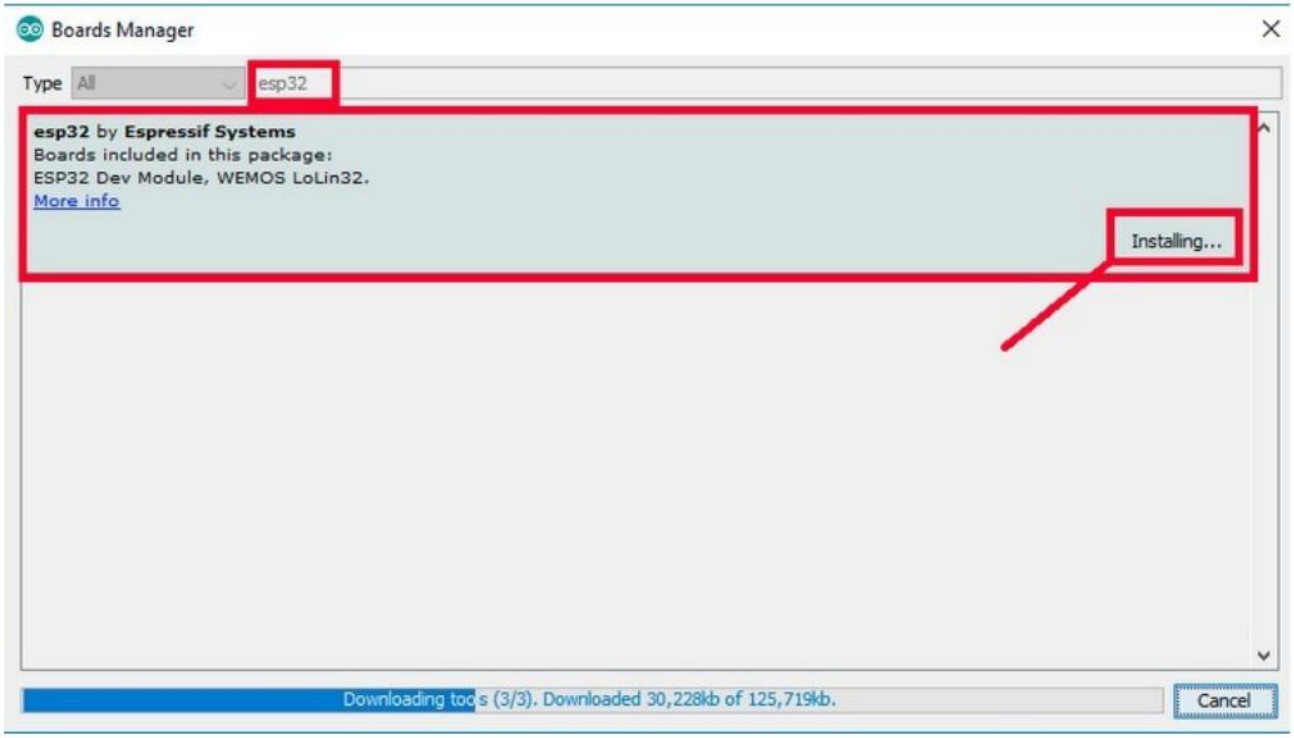


Figure 0.6. L'intégration des cartes **ESP32** et **ESP8266** dans l'environnement IDE Arduino

0.4.2 Préparation d'un code Arduino pour la compilation et chargement

La compilation doit être effectuée sur la carte **Heltec_WIFI_LoRa_32.** Elle doit être sélectionnée dans le menu **`Tools→Board`**

Avant la compilation il faut installer les bibliothèques nécessaires pour piloter différents dispositifs.
Par exemple l'image ci-dessous montre comment installer la bibliothèque **`U8g2`** qui pilote les écrans type **OLED**.

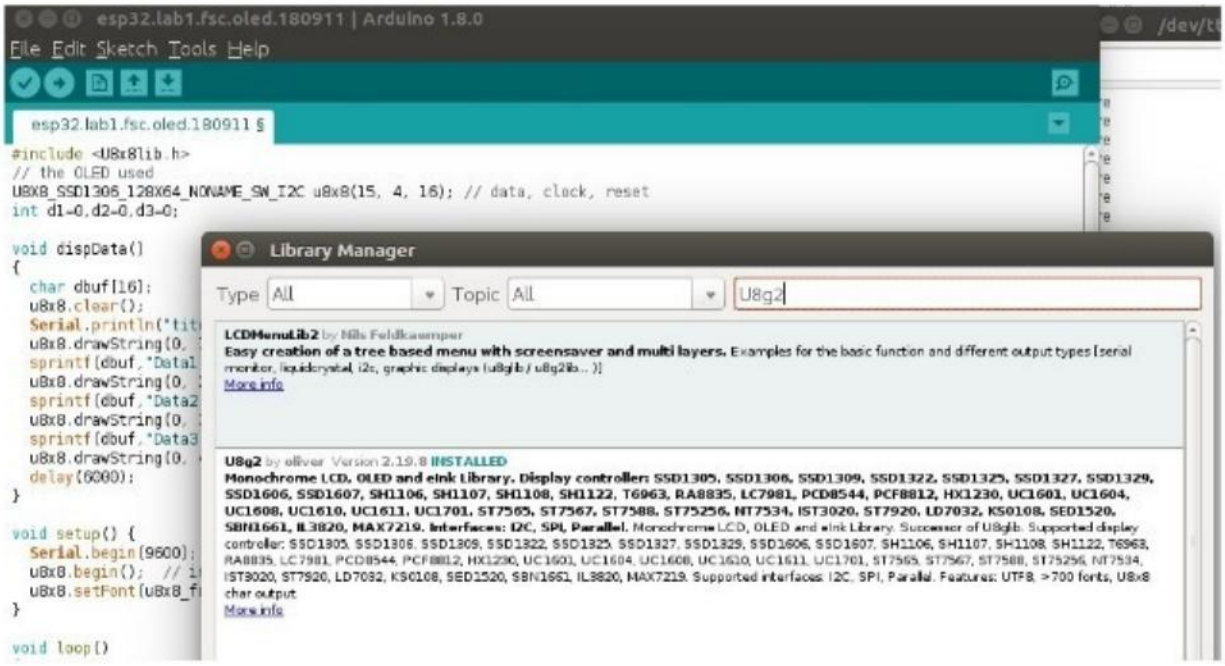


Figure 0.7. Intégration d'une bibliothèque Arduino (**U8g2**)

Laboratoire 1

1.1 Premier exemple – l'affichage des données sur l'écran OLED

Dans cet exercice nous allons simplement afficher un titre et 2 valeurs numériques sur l'écran **OLED** intégré dans la carte ESP32.

Figure 1.1 Ecran **OLED** de la carte ESP32 (Heltec WiFi LoRa 32)

Vous devez installer la bibliothèque **U8g2** (**https://github.com/olikraus/u8g2**). Cela peut être trouvé dans le gestionnaire de bibliothèque IDE Arduino. Ouvrez **Sketch> Include Library> Manage Libraries** et recherchez, puis installez **U8g2**. Notez que l'écran **OLED** est connecté sur un bus I2C avec trois broches : **SCL** – **clock**, **SDA** – **data/address** et **RESET**.

```
#include <U8x8lib.h>  // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); // clock, data, reset
int d1=0,d2=0,d3=0;

void dispData()
{
  char dbuf[16];
  u8x8.clear();
  Serial.println("titre");
  u8x8.drawString(0,1,"titre"); // 0 – colonne (max 15), 1 – ligne (max 7)
  sprintf(dbuf,"Data1:%d",d1); u8x8.drawString(0,2,dbuf);
  sprintf(dbuf,"Data2:%d",d2); u8x8.drawString(0,3,dbuf);
  sprintf(dbuf,"Data3:%d",d3);u8x8.drawString(0,4,dbuf);
  delay(6000);
}

void setup() {
  Serial.begin(9600);
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);
}

void loop()
{
d1++;d2+=2;d3+=4 ;
// ici appeler la fonction d'affichage
}
```

A faire

Compléter, compiler, et charger ce programme.

1.2 Deuxième exemple – capture et affichage des valeurs

1.2.1 Capture de la température/humidité par SHT21

Dans cet exercice nous allons lire les valeurs fournies par le capteur Température/Humidité **SHT21** et afficher les 2 valeurs sur l'écran OLED intégré dans la carte ESP32.
Le capteur **SHT21** doit être connecté sur le bus **I2C**, donc il nous faut une carte d'extension I2C avec une configuration des broches identique à celle du capteur (VIN correspond à 3V3).

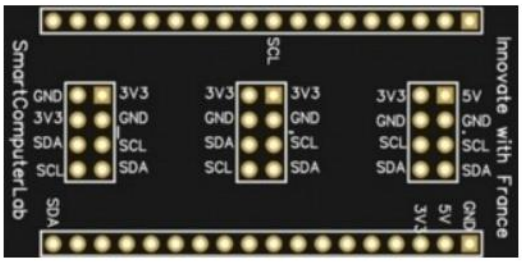


Figure 1.2. **IoT DevKit :** carte d'extension multi **I2C** et un capteur de température/humidité **SHT21**

Code Arduino

Code Arduino

```
// https://github.com/chaveiro/SHT2x-Arduino-Library

#include <Wire.h>
#include "SHT2x.h"

SHT2x SHT2x;

void setup()
{
  Wire.begin(21,22);
  SHT2x.begin();
  Serial.begin(9600);
}

void loop()
{
  uint32_t start = micros();
  Serial.print("Humidity(%RH): ");
  Serial.print(SHT2x.GetHumidity(),1);
  Serial.print("\tTemperature(C): ");
  Serial.print(SHT2x.GetTemperature(),1);

  uint32_t stop = micros();
  Serial.print("\tRead Time: ");
  Serial.println(stop - start);
  delay(1000);
}
```

1.2.2 Capture de la température/humidité par HTU21D

Dans cet exercice nous allons lire les valeurs fournies par le capteur Température/Humidité **HTU21D** et afficher les 2 valeurs sur l'écran OLED intégré dans la carte ESP32.
Le capteur **HTU21D** doit être connecté sur le bus **I2C**, donc il nous faut une carte d'extension **I2C** et l'emplacement avec la configuration des broches identique à celle du capteur.

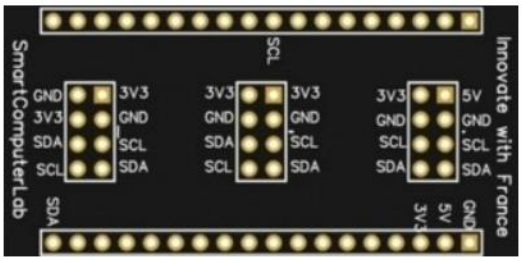


Figure 1.3 IoT DevKit : carte d'extension multi **I2C** et un capteur de température/humidité **HTU21D**

Code Arduino

```
#include <Wire.h>
#include "SparkFunHTU21D.h"
//Create an instance of the object
HTU21D sensor;

void setup()
{
  Serial.begin(9600);
  Serial.println("HTU21D Example!");
  myHumidity.begin();
}

void loop()
{
  float humd = sensor.readHumidity();
  float temp = sensor.readTemperature();
  Serial.print("Time:");
  Serial.print(millis());
  Serial.print(" Temperature:");
  Serial.print(temp, 1);
  Serial.print("C");
  Serial.print(" Humidity:");
  Serial.print(humd, 1);
  Serial.print("%");
  Serial.println();
  delay(1000);
}
```

Attention :

Pour le bon fonctionnement il faut **enlever le capteur SHT21**.

1.2.3 Capture de la luminosité par BH1750

Dans cet exercice utilisez la carte **I2C1** pour le capteur **de la luminosité BH1750**

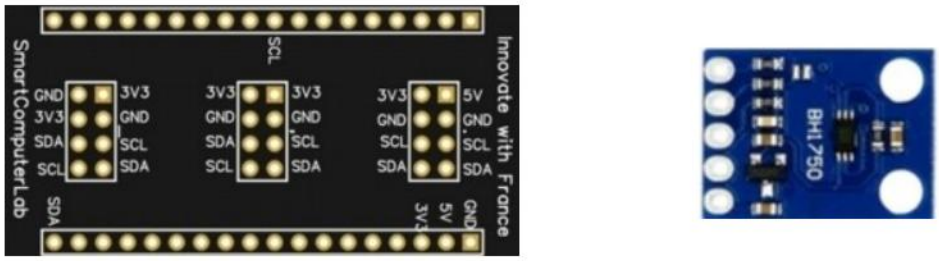


Figure 1.4. **IoT DevKit :** cartes d'extension multi - **I2C** pour le capteur Luminosité **BH1750**

Code Arduino

Attention : Avant la compilation il faut installer la bibliothèque **`BH1750.h`**

https://github.com/claws/BH1750

```
#include <Wire.h>
#include <BH1750.h>
BH1750 lightMeter;

void setup(){
  Wire.begin(21,22);  // carte d'extension I2C1 ou petite carte I2C
  Serial.begin(9600);
  lightMeter.begin();
  Serial.println("Running...");
  delay(1000);
}

void loop() {
  uint16_t lux = lightMeter.readLightLevel();
  delay(1000);
  Serial.print("Light: ");
  Serial.print(lux);
  Serial.println(" lx");
  delay(1000);
}
```

1.2.4 Capture de la luminosité par MAX44009

Dans cet exemple nous utilisons un capteur de luminosité type **MAX44009** (GY-49). Ce capteur est connectée comme le capteur BH1750 sur le bus I2C.
Nous communiquons avec ce capteurs **directement** par les trames **I2C** ce qui permet de mieux comprendre le **fonctionnement de ce bus**.

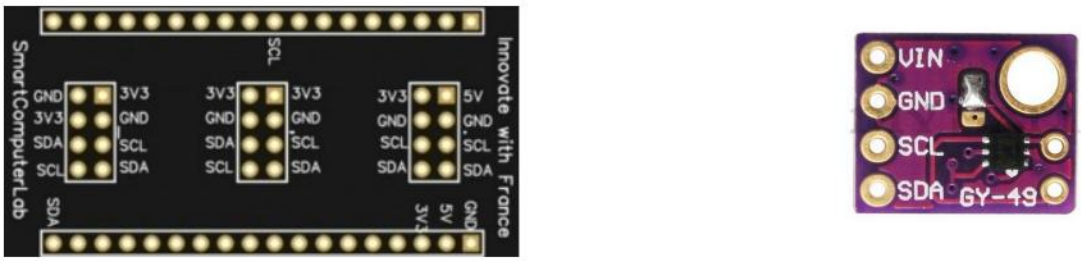


Figure 1.5 IoT DevKit : cartes d'extension **I2C** et un capteur de luminosité **MAX44009 (GY-49)**

Code Arduino :

```
#include<Wire.h>
#define Addr 0x4A

void setup()
{
Wire.begin(21,22);
Serial.begin(9600);
Wire.beginTransmission(Addr);
Wire.write(0x02); Wire.write(0x40);
Wire.endTransmission();
delay(300);
}

void loop()
{
unsigned int data[2];
Wire.beginTransmission(Addr);
Wire.write(0x03);
Wire.endTransmission();
Wire.requestFrom(Addr, 2); // Request 2 bytes of data
// Read 2 bytes of data luminance msb, luminance lsb
if (Wire.available() == 2)
{
data[0] = Wire.read(); data[1] = Wire.read();
}
// Convert the data to lux
int exponent = (data[0] & 0xF0) >> 4;
int mantissa = ((data[0] & 0x0F) << 4) | (data[1] & 0x0F);
float luminance = pow(2, exponent) * mantissa * 0.045;
Serial.print("Ambient Light luminance :"); Serial.print(luminance);
Serial.println(" lux");
delay(500);
}
```

1.2.5 Capture de la pression/température avec capteur BMP180

Le capteur BMP180 permet de capter la pression atmosphérique et la température. Sa précision est seulement de **+/- 100 Pa** et **+/-1.0C**.

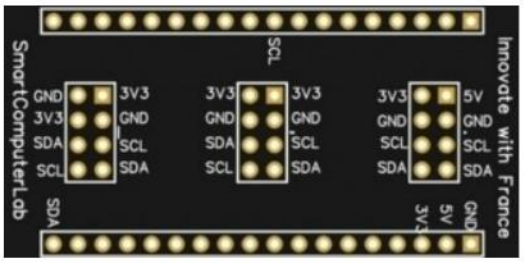


Figure 1.6. **IoT DevKit :** carte d'extension multi **I2C** et un capteur de pression/température **BMP180**

La valeur standard de la pression atmosphérique est :

101 325 Pa = 1,013 25 bar = 1 atm

Code Arduino - BMP180

```
#include <Arduino.h>
#include <Wire.h>
#include <BMP180I2C.h>
#define I2C_ADDRESS 0x77
BMP180I2C bmp180(I2C_ADDRESS);

void setup() {
      Serial.begin(9600);
      Wire.begin();
      if (!bmp180.begin())
      {
            Serial.println("check your BMP180 Interface and I2C Address.");
            while (1);
      }
      bmp180.resetToDefaults();
      //enable ultra high resolution mode for pressure measurements
      bmp180.setSamplingMode(BMP180MI::MODE_UHR);
}

void loop() {
      delay(1000);
      if (!bmp180.measureTemperature())
      {
      Serial.println("could not start temperature measurement");
            return;
      }
      //wait for hasValue() returned true.
      do
      {
            delay(100);
      } while (!bmp180.hasValue());
      Serial.print("Temperature: ");
      Serial.print(bmp180.getTemperature());
      Serial.println(" degC");
```

```
      if (!bmp180.measurePressure())
      {
      Serial.println("could not start perssure measurement");
            return;
      }
      do
      {
            delay(100);
      } while (!bmp180.hasValue());
      Serial.print("Pressure: ");
      Serial.print(bmp180.getPressure());
      Serial.println(" Pa");
}
```

A faire

1. Complétez les programmes ci-dessus afin d'afficher les données de la Luminosité sur l'écran OLED

2. Développer une application avec la capture et l'affichage de l'ensemble de trois données : Température/Humidité et Luminosité.

3. Développer une application avec la capture et l'affichage de l'ensemble de trois données : Température/Humidité et Pression (BMP180).

1.2.6 Détection de mouvement avec capteur PIR (SR602)

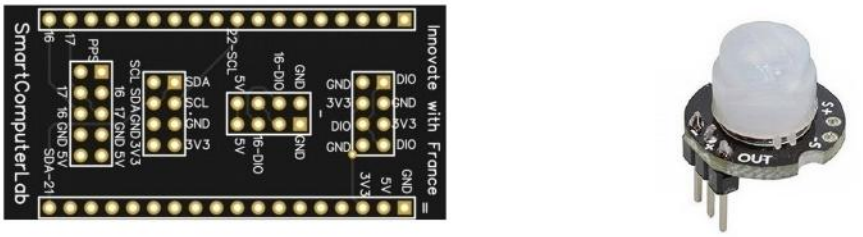


Figure 1.7 Carte **multi-capteurs (I2C, UART, one-wire)** et capteur **PIR (SR602)**

Code Arduino

```
#include <U8x8lib.h>  // bibliothèque à charger a partir de
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(15,4,16); // clock, data, reset
int d1=0,d2=0,d3=0;
#define PIR 16  // problème ?
bool MOTION_DETECTED = false;
int counter=0;

void dispData()
{
  char dbuf[16];
  u8x8.begin();  // initialize OLED
  u8x8.setFont(u8x8_font_chroma48medium8_r);u8x8.clear();
  u8x8.drawString(0,1,"Motion detected");
  sprintf(dbuf,"Count:%d",counter); u8x8.drawString(0,3,dbuf);
  delay(100);
}

void pinChanged() { MOTION_DETECTED = true; }

void setup() {
Serial.begin(9600);pinMode(PIR,INPUT);
attachInterrupt(PIR, pinChanged, RISING);
}

void loop()
{
  int i=0;
  if(MOTION_DETECTED){ Serial.println("Motion detected.");
      delay(1000);counter++;
      MOTION_DETECTED = false;
      Serial.println(counter);
      dispData();
      pinMode(PIR,INPUT) attachInterrupt(PIR, pinChanged, RISING);
      }
}
```

A faire :

1. Analyser le programme ci-dessus

Pourquoi **doit on réinitialiser l'écran et l'interruption à chaque pas d'utilisation.**

1.2.7 Mesure de distance avec capteur VL53L0X

Le **VL53L0X** est un module de **télémétrie laser** à **temps de vol** (**ToF**) offrant une mesure de distance précise quelles que soit la cible contrairement aux technologies conventionnelles. Il peut mesurer des distances absolues jusqu'à 2m**. (le capteur fonctionne mieux avec Vcc=5V)**

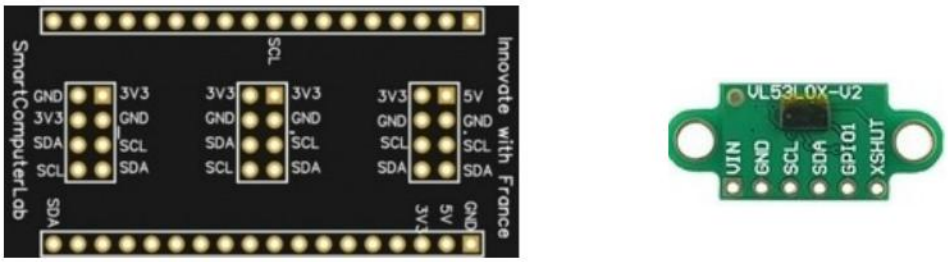


Figure 1.8 Carte multi-capteurs (I2C) et capteur **VL53L0X (laser-ToF)**

Fonctions :

```
setMode(ModeState mode, PrecisionState precision)
```

- `Continuous--->Continuous measurement model`
- `Single------->Single measurement mode`
- `High--------->Accuracy of 0.25 mm`
- `Low---------->Accuracy of 1 mm`

```
void start() - This function is used to enabled VL53L0X
float getDistance()  - This function is used to get the distance
uint16_t getAmbientCount() - This function is used to get the ambient count
uint16_t getSignalCount() - This function is used to get the signal count
uint8_t getStatus(); - This function is used to get the status
void stop() - This function is used to stop measuring
```

Le code

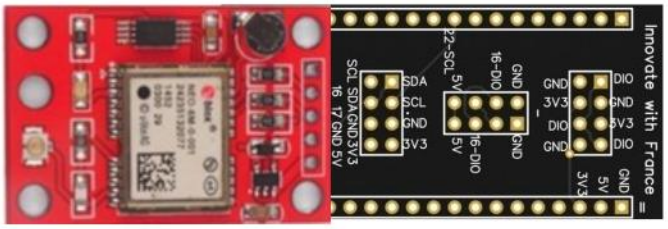
```
#include "Arduino.h"
#include "Wire.h"
#include "DFRobot_VL53L0X.h"

DFRobotVL53L0X sensor;

void setup() {
  Serial.begin(9600);
  Wire.begin(21,22);   // SDA, SCL
  sensor.begin(0x50);//Set I2C sub-device address
  //Set to Back-to-back mode and high precision mode
  sensor.setMode(Continuous,High);
  //Laser rangefinder begins to work
  sensor.start();
}

void loop()
{
  Serial.print("Distance: ");
  Serial.print(sensor.getDistance());Serial.println("mm");

  delay(500); // delay does not affect the measurement accuracy
}
```

A faire :

1. Testez le programme ci-dessus et ajoutez l'affiche sur l'écran OLED

1.2.8 Temps réel et positionnement GPS avec NEO-6M

NEO-6M est un module de **global positioning system** (GPS). Il est très populaire, économique et de hautes performances avec une antenne patch en céramique, une puce de mémoire intégrée et une batterie de secours est intégrée sur le module.

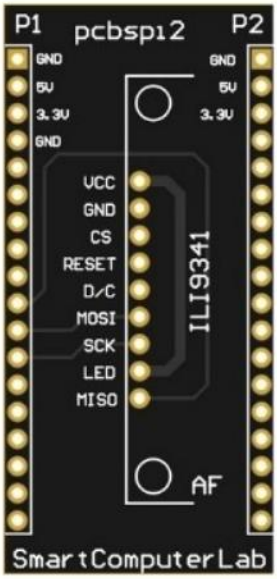


Le code

```
#include <TinyGPS++.h>
HardwareSerial uart(2);
TinyGPSPlus gps;
void setup(){
  Serial.begin(9600);
  uart.begin(9600, SERIAL_8N1, 17, 16);}

  int i=0,ihour;
  char gpst[2048],ttime[16],ftime[16],hour[8],nhour[2];
  char *ptr=NULL;

  void loop(){
    while (uart.available() > 0)
      {
        gps.encode(uart.read());
        if (gps.location.isUpdated())
          {
            Serial.print("Latitude= ");
            Serial.print(gps.location.lat(), 6);
            Serial.print(" Longitude= ");
            Serial.println(gps.location.lng(), 6);
            Serial.print("Date:");
            Serial.print(gps.date.month());
            Serial.print("/");
            Serial.print(gps.date.day());
            Serial.print("/");
            Serial.print(gps.date.year());
            Serial.print("   Time:");
            if (gps.time.hour() < 10)
              Serial.print("0");
            Serial.print(gps.time.hour());
            Serial.print(":");
            if (gps.time.minute() < 10)
              Serial.print("0");
            Serial.print(gps.time.minute());
            Serial.print(":");
            Serial.println(gps.time.second());
          }
        }
      }
```

1.2.9 Affichage sur l »écran TFT - IL9341

L'exemple suivant montre l'utilisation d'une carte pour les écrans **SPI-TFT** type **IL9341**. Nous utilisons ici la bibliothèque **`TFT_eSPI.h`** avec une interface **SPI configurable** dans le fichier **`User_Setup.h`**



Un exemple du code :

Attention : Il faut préparer les connections sur le bus SPI dans le fichier User_Setup.h présent dans votre répertoire **`TFT_eSPI-master`** dans le **`libraries`** de **`Arduino`**.

```
#include <TFT_eSPI.h> // Graphics and font library for ILI9341 driver chip
#include <SPI.h>
// to be defined in User_Setup.h
//#define TFT_MISO 19 // Not connected
//#define TFT_MOSI 23
//#define TFT_SCLK 18
//#define TFT_CS 5 // Chip select control pin
//#define TFT_DC 2 // Data Command control pin
//#define TFT_RST 4 // Reset pin (could connect to RST pin)
// GND tp GND , VCC to 3.3V and LED to 3.3V
#define TFT_GREY 0x5AEB // New colour
TFT_eSPI tft = TFT_eSPI(); // Invoke library
void setup(void) {
Serial.begin(9600);
tft.init();
tft.setRotation(3);
}

int count=0;
char disp[12];

void loop() {
Serial.println("in the loop");
// Fill screen with grey
// tft.fillScreen(TFT_GREY);
tft.fillScreen(TFT_BLACK);
// Set "cursor" at top left corner of display (0,0) and select font 2
// or stay on the line is there is room for the text with tft.print)
tft.setCursor(0, 0, 4);
```

```
// Set the font colour to be white with a black background, set text size
multiplier to 1
if(count>12)tft.setTextColor(TFT_RED,TFT_BLACK);
else tft.setTextColor(TFT_GREEN,TFT_BLACK);
tft.setTextSize(8);
// We can now plot text on screen using the "print" class
sprintf(disp,"%3.3d",count); count++;
tft.println(disp);
tft.setCursor(20, 150, 2);
tft.setTextSize(3);
if(count>12) tft.println("Porte fermee");
else tft.println("Porte ouverte");

tft.setCursor(200, 220, 2);
// Set the font colour to be yellow with no background, set to font 7
tft.setTextColor(TFT_YELLOW); tft.setTextSize(1);
tft.println("SmartComputerLab");
// Set the font colour to be red with black background, set to font 4
//tft.setTextColor(TFT_RED,TFT_BLACK);
//tft.setTextFont(4);
//tft.println(3735928559L, HEX); // Should print DEADBEEF
// Set the font colour to be green with black background, set to font 4
//tft.setTextColor(TFT_GREEN,TFT_BLACK);
//tft.setTextFont(4);
//tft.println("Groop");
//tft.println("I implore thee,");
// Change to font 2
//tft.setTextFont(2);
//tft.println("my foonting turlingdromes.");
//tft.println("And hooptiously drangle me");
//tft.println("with crinkly bindlewurdles,");
// This next line is deliberately made too long for the display width to test
// automatic text wrapping onto the next line
//tft.println("Or I will rend thee in the gobberwarts with my
blurglecruncheon,see if I don't!");
// Test some print formatting functions
//float fnumber = 123.45;
// Set the font colour to be blue with no background, set to font 4
//tft.setTextColor(TFT_BLUE);
//tft.setTextFont(4);
//tft.print("Float = "); tft.println(fnumber);
// Print floating point number
//tft.print("Binary = "); tft.println((int)fnumber, BIN); // Print as integer
value in binary
//tft.print("Hexadecimal = "); tft.println((int)fnumber, HEX); // Print as
integer number in Hexadecimal
delay(3000);
}
```

A faire :

1. Modifier le contenu du programme ci-dessus pour afficher , par exemple:
`IoT - Lab1` sur votre écran TFT
2. Ajouter un **capteur** , par exemple température-humidité sur la carte multi-capteurs **I2C** et afficher les valeurs captées
3. Ajouter le **modem GPS** , la carte multi-capteurs **PIR, UART** et afficher les valeurs captées