Lab 5

Raspberry PI3 (Raspberry Pi Z W) et un module Lora

Dans ce TP nous allons utiliser une carte Raspberry Pi Zero W.

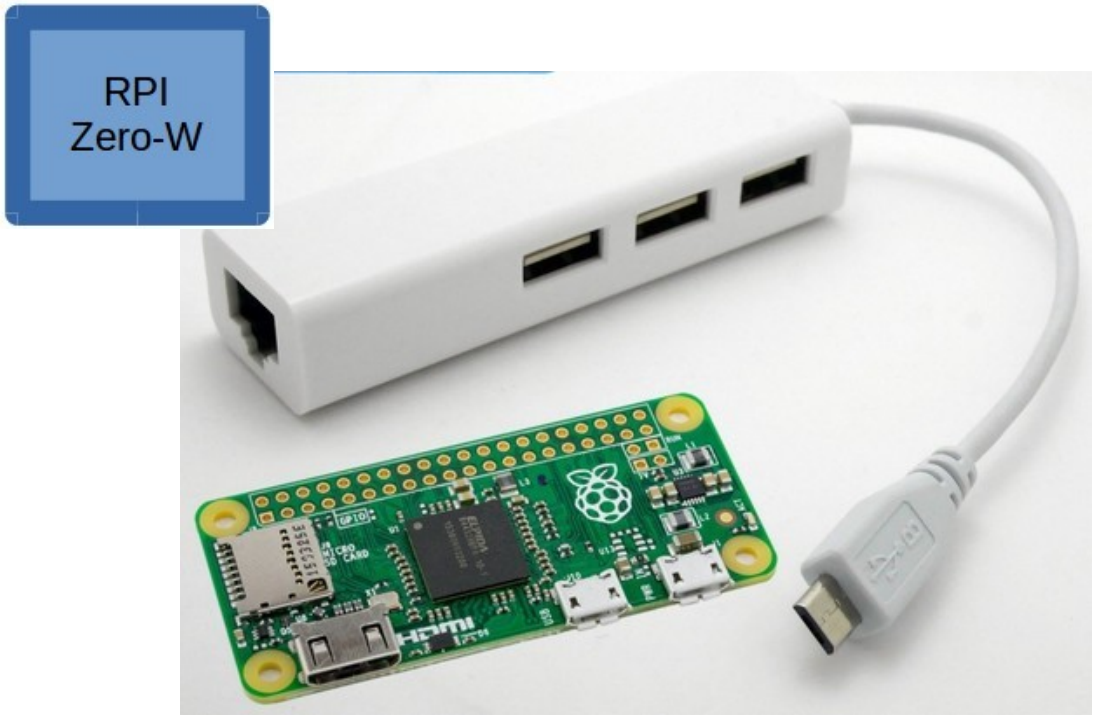


Figure 5.1 Raspberry Pi Zero (W) avec une interface Ethernet/USB ;

Le Raspberry Pi Zero W possède toutes les fonctionnalités du Pi Zero original, mais comprend une connectivité supplémentaire:

- LAN sans fil 802.11 b / g / n
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

Comme le Pi Zero, il a aussi:

- Processeur 1GHz,
- 512 Mo de RAM
- Mini ports HDMI et USB On-The-Go
- Alimentation via port Micro USB
- En-tête à 40 broches
- Connecteur vidéo composite
- Connecteur de caméra CSI

La carte RPIZ(W) fonctionne avec le OS Raspbian (Pixel). Dans notre cas nous n'avons pas besoin de son interface graphique et tout le travail va se dérouler via un terminal SSH accessible à distance de notre poste de travail (PC).

Le système Raspbian est complété par plusieurs packages incluant la bibliothèque Radio Head , la même qu'on va exploiter sur la nos cartes Wemos D1 pour la communication Long Range avec les modems LoRa. Pour communiquer avec la carte nous allons utiliser l'interface Ethernet ajoutée via un adaptateur micro USB/Ethernet. Cet adaptateur dispose également de trois portes USB.

5.1 Interfaces réseau `wlan0` et `eth0`

D'abord, nous allons configurer notre interface **`wlan0`** avec une interface statique IP et **`eth0`** avec une adresse IP statique ou dynamique.

Vous voyez ci-dessous le fichier de configuration:
`/etc/network/interfaces`

```
# eth0 and wlan0 configuration file
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
#iface eth0 inet static
        address 172.19.65.55
        netmask 255.255.248.0
        gateway 172.19.64.3
        dns-nameservers 172.19.0.4
allow-hotplug wlan0
iface wlan0 inet static
        address 192.168.8.1
        netmask 255.255.255.0
        gateway 192.168.8.1
```

Notez que nous avons deux configurations pour l'interface **`eth0`**, une avec IP dynamique fournie par protocole DHCP et une avec IP statique spécifiée pour l'utilisation à l'université.

5.1.1 L'arrêt du `network-manager`

Dans la plupart des cas, l'utilisation de RPI ou OPI pour les projets liés à IoT nécessite la configuration manuelle des interfaces réseau. Cela signifie que le gestionnaire de réseau existant devrait être stoppé par une commande et une modification dans la configuration du **network-manager**.

```
sudo stop network-manager
echo "manual" | sudo tee /etc/init/network-manager.override
```

5.1.2 Configuration de `hostapd` , `dnsmasq` et `iptables`

L'installation d'un point d'accès WiFi soft nécessite chargement de deux paquets **`hostapd, iptables`** et **`dnsmasq`**.

```
sudo apt-get update
sudo apt-get install hostapd iptables dnsmasq
```

`hostapd` permet de lancer les fonctions du point d'accès tandis que **`dnsmasq`** fournit les adresses IP demandes par les clients.
Nous commençons par la création du fichier **`hostapd.conf`** :

```
sudo vi /etc/hostapd/hostapd.conf with the following contents:
# This is the name of the WiFi interface we configured above
interface=wlan0
# Use the nl80211 driver with the brcmfmac driver
driver=nl80211
# This is the name of the network
```

```
ssid=smtrAP.1
# Use the 2.4GHz band anf 802.11g standard
hw_mode=g
# Use channel 1
channel=1
# Enable 802.11n
ieee80211n=1
```

Nous devons également indiquer à **hostapd** où rechercher le fichier de configuration lorsqu'il démarre lors du boot.
Ouvrez le fichier de configuration par défaut avec:

```
sudo vi /etc/default/hostapd
```

trouvez la ligne :
#DAEMON_CONF=""

et le remplacez par
DAEMON_CONF="/etc/hostapd/hostapd.conf"

Nous pouvons vérifier si cela fonctionne à ce stade en exécutant:
sudo service hostapd start

Installation du serveur dns

```
sudo vi /etc/dnsmasq.conf
interface=wlan0
dhcp-range=192.168.8.100,192.168.8.200,255.255.255.0,12h
```

Nos clients wifi se connectant sur l'interface wlan1 se verront attribuer une IP entre **192.168.8.100** et **192.168.8.200**, avec un masque de sous-réseau **255.255.255.0**, et un bail de 12 heures

Installation de Iptables et configuration du bridge (eth0 - wlan0)

Activons le port ***forwarding*** :
sudo nano /etc/sysctl.conf

ajouter :
net.ipv4.ip_forward=1

On redirige la connexion venant de **eth0** vers **wlan0** :

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED
-j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

Il ne reste plus qu'à rebooter, et lancer le tout :

```
sudo service dnsmasq start
sudo service hostapd start
```

5.2 Interface GPIO et WiringPi

WiringPi est une librairie écrite en C permettant l'accès au GPIO (***General Purpose Input/Output***)du BCM2835 utilisé par le Raspberry Pi. Elle est utilisable en C et C++ ainsi qu'avec de nombreux autres langages

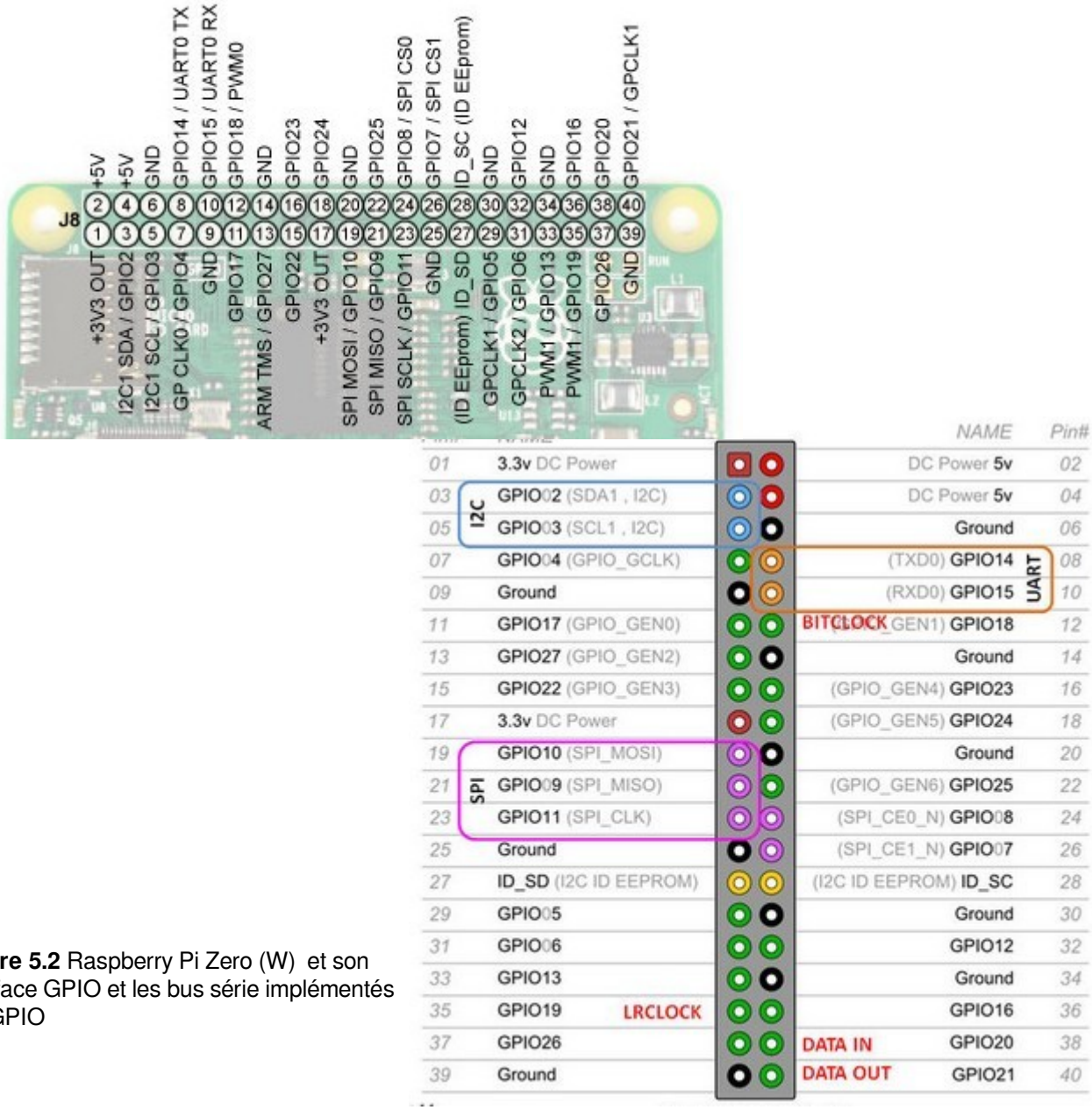


Figure 5.2 Raspberry Pi Zero (W) et son interface GPIO et les bus série implémentés sur GPIO

WiringPi est conçue pour être familière aux personnes qui utilise Arduino .
Le RPI Zero dispose d'un connecteur 40-broches GPIO qui transporte des signaux et des bus (de données). Il y a 12 broches E/S digital d'utilisation générale – elles peuvent être programmée soit en entrée, soit en sortie digital. Une de ces broches peut également être désignée pour servir de sortie PWM.
Il existe une interface **I2C** 2-fils (***2-wire***), une interface **SPI** 4-fils (**4-wire**) avec une seconde ligne de sélection (ce qui fait 5 broches au total pour le SPI) et un UART série avec 2 broches complémentaires.

Les interfaces I2C, SPI et UART peuvent aussi être utilisée comme E/S d'utilisation générale lorsqu'elles ne sont pas utilisée comme Bus. Cela fait un total de 12 + 2 + 5 + 2 = 21 broches E/S.

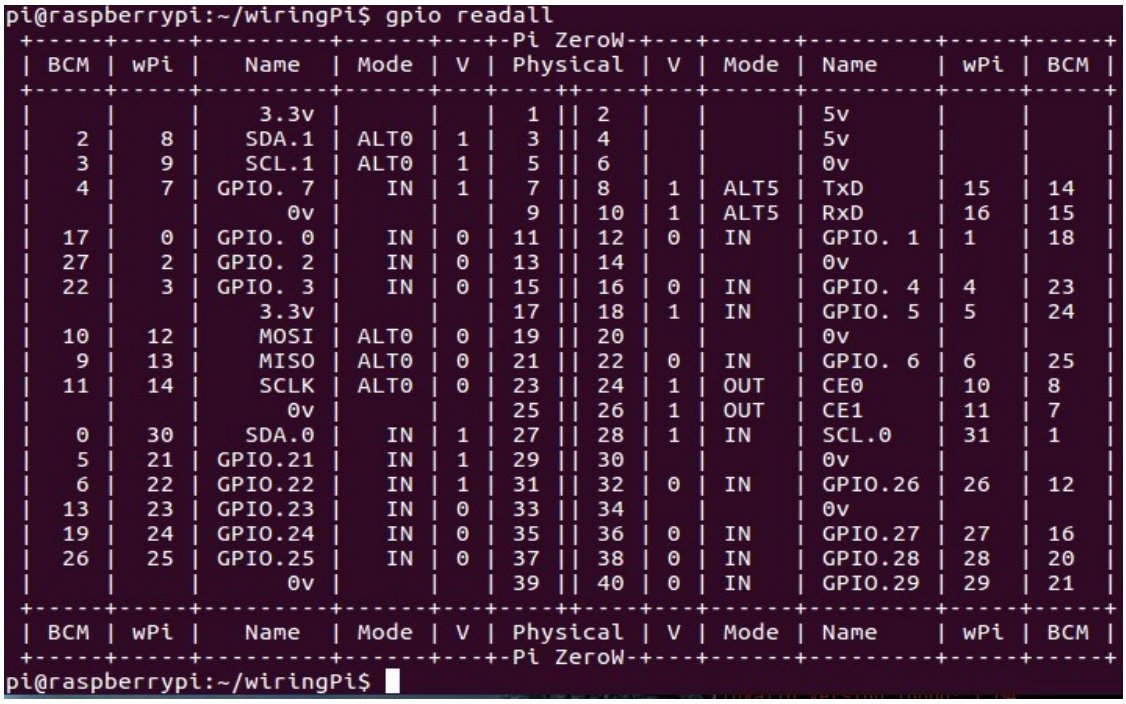
5.2.1 Installer WiringPi

On récupèrer la dernière version de **WiringPi**:
`git clone git://git.drogon.net/wiringPi`

On se place dans le bon répertoire:
`cd wiringPi`

On lance la compilation:
/**build**
Maintenant **WiringPi** est installé pour tester on écrit:

`gpio readall`



```
pi@raspberrypi:~/wiringPi$ gpio readall
 +-----+-----+---------+------+---+-Pi ZeroW-+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT5 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT5 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 1 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+-Pi ZeroW-+---+------+---------+-----+-----+
pi@raspberrypi:~/wiringPi$
```

Les commandes de base

Définir un GPIO en mode sortie le GPIO 0 dans cet exemple.
`sudo gpio mode 0 out`

Changer l'état d'un GPIO:
`sudo gpio 0 1`
`sudo gpio 0 0`

5.2.2 Lire les données d'un capteur DHT11

L'exemple suivant permet de lire les données (température, humidité) à partir du simple capteur DHT11. Le fonctionnement de la ligne série est programmé. Le résulta de la lecture est simplement affiché dans le terminal.

```
#include <wiringPi.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#define MAX_TIMINGS 85
#define DHT_PIN 7
int data[5] = { 0, 0, 0, 0, 0 };
```

```
void read_dht_data()
{
    uint8_t laststate    = HIGH;
    uint8_t counter        = 0;
    uint8_t j            = 0, i;
    data[0] = data[1] = data[2] = data[3] = data[4] = 0;
    /* pull pin down for 18 milliseconds */
    pinMode(DHT_PIN, OUTPUT);
    digitalWrite(DHT_PIN, LOW);
    delay(18);
    /* prepare to read the pin */
    pinMode(DHT_PIN, INPUT );
    /* detect change and read data */
    for ( i = 0; i < MAX_TIMINGS; i++ )
    {
        counter = 0;
        while ( digitalRead(DHT_PIN) == laststate )
        {
            counter++;
            delayMicroseconds(1); // 1 by default
            if ( counter == 255 )
            {
                break;
            }
        }
laststate = digitalRead(DHT_PIN);
        if ( counter == 255 )
            break;
        /* ignore first 3 transitions */
        if ( (i >= 4) && (i % 2 == 0) )
        {
            /* shove each bit into the storage bytes */
            data[j / 8] <<= 1;
            if ( counter > 16 )
                data[j / 8] |= 1;
            j++;
        }
    }
/*
     * check we read 40 bits (8bit x 5 ) + verify checksum in the last byte
     * print it out if data is good
     */
    if ( (j >= 40) &&
         (data[4] == ( (data[0] + data[1] + data[2] + data[3]) & 0xFF) ) )
    {
        float h = (float)((data[0] << 8) + data[1]) / 10;
        if ( h > 100 )
        {
            h = data[0];    // for DHT11
        }
```

```
        float c = (float)(((data[2] & 0x7F) << 8) + data[3]) / 10;
        if ( c > 125 )
        {
            c = data[2];    // for DHT11
        }
        if ( data[2] & 0x80 )
        {
            c =-c;
        }
        float f = c * 1.8f + 32;
        printf("Humidity=%.1f %% Temperature=%.1f *C (%.1f *F)\n",h,c,f);
    }
    else  {
       delay(100);
       //printf( "Data not good, skip\n" );
    }
}
int main( void )
{
    printf( "DHT22 temperature/humidity test\n" );
    if ( wiringPiSetup() ==-1 )
        exit( 1 );
    while ( 1 )
    {
        read_dht_data();
        delay( 1000 ); /* wait 2 seconds before next read */
    }
    return(0);
}
```

Attention :

La compilation du programme ci-dessus est effectuée par la commande :

```
cc -o dht11 dht11.c -lwiringPi -lm
```

5.3 Utiliser bus I2C sur GPIO

Attention :

Avant de commencer l'utiilisation des bus I2C et SPI ils doivent être leurs drivers doivent être activés dans la commande **`raspi-config`**.
Sur RPI3 ou RPIZ(W) nous pouvons tester l'état du bus I2C par la commande **`i2cdetect`** :

```
sudo apt-get install i2c-tools
ls /dev/*i2c*
/dev/i2c-1
```

5.3.1 Lire les données de capteur HTU21D

Dans le premier exemple d'utilisation du bus I2C nous prenons le capteur de température-humidité HTU21D.
Etant donné que le capteur est physiquement connecté nous pouvons voir son adresse ci-dessous :

```
pi@raspberrypi:~/wiringPi/examples$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

ce qui signifie que l'adresse **`0x40`** est activée avec un capteur - dans ce cas **HTU21D**, qui est capteur de température et d'humidité.
Le capteur est connecté sur les broches I2C : 3 – SDA, 5 – SCL, 1 – 3.3V et 9 – GND.

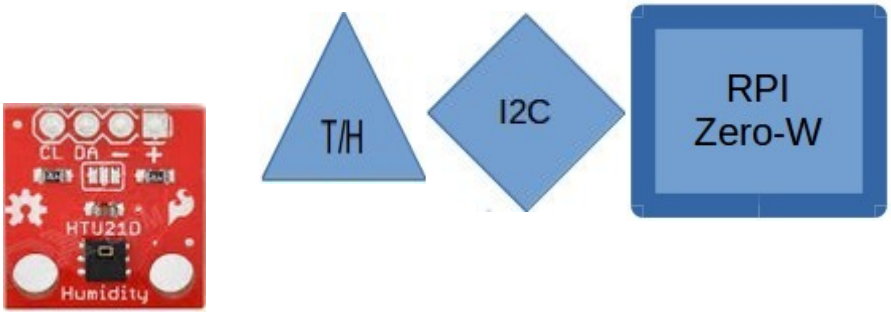


Figure 5.3 Raspberry Pi Zero (W) avec un capteur – **HTU21D** connecté sur le bus I2C

Pour capturer les données de ce capteur, nous pouvons écrire le code suivant :

```
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include "HTU21D.h"
int main ()
{
        wiringPiSetup();
        int fd = wiringPiI2CSetup(HTU21D_I2C_ADDR);
        if (0>fd )
        {
```

```
        printf ("Unable to open I2C device\n"; exit (1); }
while(1) {
        sleep(3);
        printf("%5.2fC\n", getTemperature(fd));
        printf("%5.2f\%\n", getHumidity(fd));
        }
        return 0;
}
```

Pour compiler le programme ci-dessus, nous avons besoin des fichiers **HTU21D.h** et **HTU21D.c**. Ils doivent être stockés dans le même répertoire que programme lui-même.
Ces deux fichiers peuvent être téléchargés à partir de
https://github.com/leon-anavi/rpi-examples

Pour compiler le programme avec les bibliothèques et fichiers requis, nous utilisons:

```
cc -o htu21d htu21d.c  HTU21D.c  -lwiringPi -lpthread -lm
```

5.3.2 Lire les données de capteur BH1750

Dans le deuxième exemple d'utilisation du bus I2C nous prenons le capteur de luminosité BH1750. L'adresse I2C de ce capteur est **0x23**. La broche **addr** du capteur doit être connectée sur terre (GND).

Figure 5.4 Capteur – **BH1750**

Voici le code C permettant de lire les données de ce capteur :

```
#include <stdio.h>
#include "BH1750.h"

int main()
{
        int fd = wiringPiI2CSetup(BH1750_ADDR) ;
        while(1)
                {
                printf("Lux: %d \n", getLux(fd));
                delay(2000);
                }
        return 0 ;
}
```

La compilation de cet exemple est effectuée par :

```
cc -o bh1750 bh1750.c BH1750.c -lwiringPi -lm
```

5.3.3 Utiliser bus I2C et envoyer les données sur le serveur ThingSpeak.com

Le code suivant montre comment envoyer les données du capteur au serveur **thingspeak.com**. Notez que la luminosité est envoyé comme valeur constante. Pour fournir les données réelles, nous devons connecter en parallèle le capteur de luminosité et ajouter les opérations nécessaires dans le code.

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>
#include<netdb.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<string.h>
#include <wiringPi.h>
#include <stdint.h>
#include "HTU21D.h"
#define MAX_TIMINGS 85
int tssend(char *key,char *temp,char *humi, char *lumi)
{
int port=80;
char *tsip="184.106.153.149";
int sd,i;
struct sockaddr_in sa;
char response[30000+1];  // + 1 is for null
int lsa=16;
char request[100];
char filename[100];
sprintf(filename,"https://api.thingspeak.com/update?api_key=%s&field1=%s&field2=
%s&field3=%s",key,temp,humi,lumi);
sprintf(request,"GET %s HTTP/1.1\r\nHost: %s\r\n\r\n",filename,tsip);
printf("%s\n\n",request);
sa.sin_family= AF_INET;
sa.sin_port = htons(port);
sa.sin_addr.s_addr=inet_addr(tsip);
sd=socket(AF_INET,SOCK_STREAM,0);
connect(sd,(struct sockaddr *)&sa,lsa);
if(send(sd,request,strlen(request),0) < strlen(request))
{
printf("Send Error!!\n");return(-1);
}
memset(response,0x00,30000);
if(recv(sd,response,30000,0)==0)
{ printf("Recv Error!!\n");return(-2); }
close(sd); // we dont need it any more
return(strlen(response)); // for debugging purposes
}
```

```
int main(int argc , char *argv[])
{
int ret;
int fd;
char temp[8],humi[8],lumi[8]="105";
char *key="EFV3QXPSA2S7910R";  // to be changed !!!
wiringPiSetup();
fd=wiringPiI2CSetup(HTU21D_I2C_ADDR);
if(fd<0) { printf("HTU error\n"); exit(1); }
while(1)
        {
        sleep(5);
        memset(temp,0x00,8); memset(humi,0x00,8); //memset(lumi,0x00,8);
        sprintf(temp,"%5.2f",getTemperature(fd));
        sprintf(humi,"%5.2f",getHumidity(fd));
        tssend(key,temp,humi,lumi);
        sleep(55);
}
}
```

Exercice:

Remplacer le capteur HTU21D par le capteur de luminosité BH1750. Laisser les valeurs de température-humidité comme constantes.

5.4 Utiliser bus UART sur GPIO pour la communication avec un modem GPS

Bus UART (**Universal Asynchronous Receiver-Tranmitter**) est souvent utilisé pour connecter différentes modules de capture ou de communication. Bus UART sur le GPIO de RPIZ est implémenté sur les broches **8 (Tx)** et **10 (Rx) (voir Figure 5.2)**
Dans l'exemple suivant nous montrons comment utiliser le bus UART pour communiquer avec un module GPS.

Par défaut, le Raspberry Pi utilise l'UART comme console série. Nous devons désactiver cette fonctionnalité pour pouvoir utiliser l'UART pour notre propre application.

Préparation du bus UART pour RPI3 ou RPIZ-W

La première chose que nous ferons est de sauvegarder le fichier **`cmdline.txt`** avant de l'éditer.
`sudo cp /boot/cmdline.txt /boot/cmdline_backup.txt`

Nous devons éditer **`cmdlint.txt`** et supprimer l'interface série.

`sudo vi /boot/cmdline.txt`

Supprimez la **`console=serial0,115200`** et enregistrez le fichier
Tapez
`sudo reboot` pour redémarrer le Pi.

Avant de commencer à écrire notre propre code, testez le GPS en utilisant des applications disponibles.
`sudo apt-get install gpsd gpsd-clients`

Ensuite initialisez le port série:
`stty -F /dev/ttyAMA0 9600`

Maintenant activez le démon **`gpsd`**:
`sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock`

Finalement affichez les données GPS par :
`cgps -s`

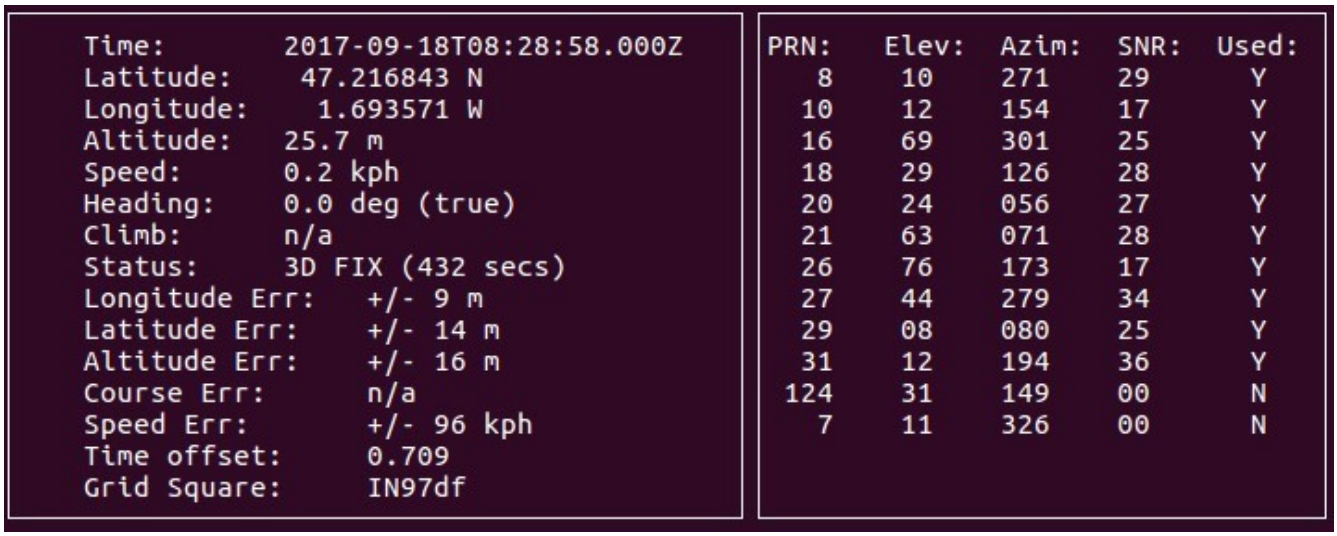


Figure 5.5 L'affichage des données captées par le module GPS **NEO-6M**

Le programme **`cgps`** affiche les données GPS comme sur la **Figure 5.5**. Si vous voulez récupérer ces données

pour les envoyer séparément sur un autre dispositif ou un autre site (par exemple **ThingSpeak**) il vous faut un programme en C/C++ qui capte les données du serveur gps (**gpsd**).

Voici une solution :

```
wget https://github.com/RedBeardCode/gpsd_example
cd gpsd_example
unzip gpsd_example-master.zip
cd gpsd_example-master/
make
./gpsd_example
```

Le fichier **main.cpp** compilé par **make** (**Makefile**) :

Le fichier d'origine **main.cpp** a été modifié comme ci-dessous pour simplifier son utilisation future.

```
/*****************************************
 *  Gpsd client example
 *  created by P.Bakowski - smartcomputerlab.org
 *****************************************/

#include <libgpsmm.h>
#include <thread>
#include <time.h>
//Controlling the thread flow
bool gReadGps = true;
time_t itime;
// Thread which reads all position form the gpsd server
void read_gpsd_data() {
    gpsmm gps_rec("localhost", DEFAULT_GPSD_PORT);
    if (gps_rec.stream(WATCH_ENABLE|WATCH_JSON) == NULL) {
        printf("No GPSD running.\n");
        return;
    }
    while (gReadGps) {
        struct gps_data_t *nd; // newdata
        if (!gps_rec.waiting(50000000))
            continue;
        if ((nd = gps_rec.read()) == NULL) {
            printf("Read error.\n");
            return;
        } else {
                itime= (time_t)nd->fix.time;
                printf("alt:%f, lat:%f, long:%f, speed:%f\n",nd-
>fix.altitude,nd->fix.latitude,nd->fix.longitude,nd->fix.speed);
                printf("time=%s\n",ctime(&itime));
        }
    }
}
///Starts the thread which reads the position and stores it in a Contaier
```

```
int main(void)
{
    std::thread gps_thread(read_gpsd_data);
    for (int i=0; i<1000; i++)
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
     }
    gReadGps = false;
    gps_thread.join();
    return 0;
}
```

Voici la trace d'exécution du programme **gpsd_example** :

```
pi@raspberrypi:~/gpsd_example-master$ ./gpsd_example

alt:nan, lat:nan, long:nan, speed:nan
time=Thu Jan  1 01:00:00 1970

alt:22.800000, lat:47.216865, long:-1.693605, speed:0.103000
time=Mon Sep 18 15:48:52 2017

alt:22.800000, lat:47.216865, long:-1.693607, speed:0.103000
time=Mon Sep 18 15:48:53 2017
```

5.5 Utiliser bus UART sur GPIO pour la communication avec un modem radio HC-12

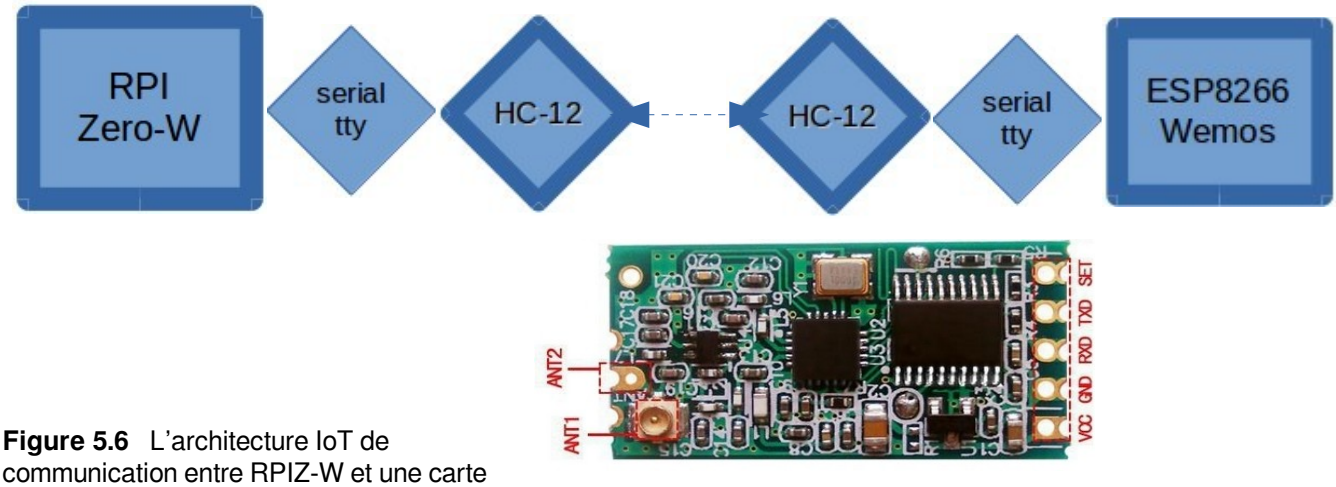


Figure 5.6 L'architecture IoT de communication entre RPIZ-W et une carte Wemos D1 à la base des modules radio HC-12 (433.4-473.0MHz)

Dans le schéma symbolique ci-dessus nous avons une simple architecture avec 2 modules de communication radio (HC-12) connectés via le bus UART. Le module (modem) est configurable , nous pouvons choisir un sous-bande de fréquences , le puissance d'émission et le débit de la communication (mode). Les canaux sont éloignés de 400kHz ce qui donne 100 canaux disponibles.
Les quatre modes sont FU1, FU2, FU3, et FU4 . En état de repos la consommation est respectivement de: 3.6mA, 80uA, 16mA, et 6mA Le courant maximum en état d'émission est 100mA (20dBm). La sensibilité de réception est -117 dBm , le débit maximal 5Kbit. La distance de communication en air est 1000 m.

Le broche **`SET`** en état **`LOW`** met le modem en mode configuration par le biais de commandes **`AT`**. Par exemple **`AT+C001`** signifie canal nr 1, **`AT+P8`** signifie la puissance d'émission 8 (maximale).
Vous pouvez retrouver d'autre commandes dans la documentation du module HC-12.

Code Arduino (Wemos D1) d'une simple application type chat :

```
#include <SoftwareSerial.h>
SoftwareSerial hc12(D1,D2) ; // Rx, Tx Wemos D1 mini
int i=0,c=0; char m;
void setup()
{
Serial.begin(9600) ;
pinMode(D7,OUTPUT) ;  // connected to SET
digitalWrite(D7,LOW) ; // enter AT command mode
hc12.begin(9600) ; delay(100);
Serial.println("set channel 1");
hc12.print("AT+C001\r\n") ; // set to channel 1
delay(100) ;
hc12.print("AT+P8\r\n") ; // set max power  20dBm
delay(100) ;
digitalWrite(D7,HIGH) ; // enter transparent mode
delay(200);
Serial.println("enter transparent mode");
}
```

```
void loop()#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#define SET 0   // SET pin on the HT12 modem pin 11 - wiringPi
int setup(int dev, int channel)
{
char com[32];
pinMode(SET,OUTPUT);delay(200);
printf("enter configuration mode\n");
digitalWrite(SET,LOW);delay(1000);
printf("set channel %3.3d\n",channel);
memset(com,0x00,32);
sprintf(com,"AT+C%3.3d\r\n",channel);
serialPuts(dev,com);delay(300);
serialPuts(dev,"AT+P8\r\n");  // highest transmission power
delay(300);
digitalWrite(SET,HIGH);delay(100);
printf("enter transparent mode\n");
return 0;
}
int main (int c,char **a)
{
  int fd ;
  int count ;
  int i;
  char buff[64];
  unsigned int nextTime ;
  if(c<2) { printf("Usage:%s channel_num\n",a[0]);exit(1);}
  if ((fd = serialOpen ("/dev/ttyAMA0", 9600)) < 0)
  { printf("Unable to open serial device:\n") ;return 1 ; }
  if (wiringPiSetup () == -1)
  { printf("Unable to start wiringPi\n") ;return 1 ;}
  setup(fd,atoi(a[1]));
  nextTime = millis () + 300 ;
  for (count = 0 ; count < 256 ; )
  {
    if (millis () > nextTime)
    {
      printf("\nYour message: ");
      memset(buff,0x00,64);gets(buff);
      fflush (stdout);
      i=0; while(buff[i]) { serialPutchar(fd,buff[i]);i++; }
      nextTime += 800 ;++count ;
      if(count>255) count=0;
    }
    delay (3) ;
```

```
    while (serialDataAvail (fd))
    {

      printf ("%c",serialGetchar(fd)) ;
      fflush (stdout) ;
    }
  }
  printf ("\n") ;
  return 0 ;
}
{
  if(Serial.available()>0) { m=Serial.read(); hc12.write(m);
  }
  if(hc12.available())
  { c++;if(c%16==0) Serial.println();
    Serial.write(hc12.read());}
}
```

Code C coté RPIZ-W :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <wiringPi.h>
#include <wiringSerial.h>
#define SET 0   // SET pin on the HT12 modem pin 11 - wiringPi
int setup(int dev, int channel)
{
char com[32];
pinMode(SET,OUTPUT);delay(200);
printf("enter configuration mode\n");
digitalWrite(SET,LOW);delay(1000);
printf("set channel %3.3d\n",channel);
memset(com,0x00,32);
sprintf(com,"AT+C%3.3d\r\n",channel);
serialPuts(dev,com);delay(300);
serialPuts(dev,"AT+P8\r\n");  // highest transmission power
delay(300);
digitalWrite(SET,HIGH);delay(100);
printf("enter transparent mode\n");
return 0;
}
int main (int c,char **a)
{
  int fd ;
  int count ;
  int i;
  char buff[64];
```

```
  unsigned int nextTime ;
  if(c<2) { printf("Usage:%s channel_num\n",a[0]);exit(1);}
  if ((fd = serialOpen ("/dev/ttyAMA0", 9600)) < 0)
  { printf("Unable to open serial device:\n") ;return 1 ; }
  if (wiringPiSetup () == -1)
  { printf("Unable to start wiringPi\n") ;return 1 ;}
  setup(fd,atoi(a[1]));
  nextTime = millis () + 300 ;
  for (count = 0 ; count < 256 ; )
  {
    if (millis () > nextTime)
    {
      printf("\nYour message: ");
      memset(buff,0x00,64);gets(buff);
      fflush (stdout);
      i=0; while(buff[i]) { serialPutchar(fd,buff[i]);i++; }
      nextTime += 800 ;++count ;
      if(count>255) count=0;
    }
    delay (3) ;
    while (serialDataAvail (fd))
    {
      //printf ("->%3d",serialGetchar(fd)) ;
      printf ("%c",serialGetchar(fd)) ;
      fflush (stdout) ;
    }
  }
  printf ("\n") ;
  return 0 ;
}
```

Exercice:

Récupérer les données GPS du serveur **gpsd** et les envoyer sur **thingspeak.com**. Utilisez un softAP sur un smartphone. Ci-dessous l'exemple d'utilisation de **YotaPhoneAP** avec le mot de passe **admsmtr2017**. Modifier les lignes nécessaires associées au **wlan0** dans le fichier **/etc/network/interfaces** :

```
auto wlan0
#allow-hotplug wlan0
#iface wlan0 inet static
iface wlan0 inet dhcp
  wpa-ssid YotaPhoneAP
  wpa-psk admsmtr2017
#  wpa-ssid eduram
#  wpa-psk scotland2010
#    address 192.168.8.1
#    netmask 255.255.255.0
#    gateway 192.168.8.1
#    nameserver 192.168.8.1
```

Attention :
Si vous laissez

```
#iface wlan0 inet static et les adresses
#    address 192.168.8.1
#    netmask 255.255.255.0
#    gateway 192.168.8.1
#    nameserver 192.168.8.1
```

non commentée,
le RPIZ (W) va fonctionner comme point d'accès et un bridge **wlan0** vers l'interface **eth0**.

Après la modification lancez la commande suivante pour voir si votre RPIZ(W) est bien connecté au point d'accès du smartphone.

```
pi@raspberrypi:/etc/network$ ip a
1: ..
2: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether b8:27:eb:8c:b8:55 brd ff:ff:ff:ff:ff:ff
    inet 192.168.43.141/24 brd 192.168.43.255 scope global wlan0
       valid_lft forever preferred_lft forever
3: ..
```