

IOT : PARTIE PROGRAMMATION

TP 1 : I2C, capteurs et envoi sur Thingspeak :

Comme pour les précédents Tps, devant chaque manipulation, sera annoté le temps nécessaire pour compléter la manipulation.

La somme sera égale à 3h30, vous aurez donc 30 min en cas de blocage.

Rappels sur le bus I2C :

Le bus I2C, crée par Philips au début des années 80, permet comme tous les protocoles de communication que nous allons utiliser lors de ces Tps, de faire communiquer entre eux différents composants électroniques.

Le principal avantage du BUS I2C c'est qu'il n'a besoin que de 3 fils :

- SDA
- SCL
- MASSE

Principe de fonctionnement :

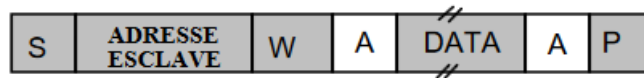
- Avant la prise de contrôle, SDA et SCL sont au repos à 1.
- Afin d'assurer la transmission de données il faut surveiller les deux conditions de départ et de repos :
 - 1 - SDA à 0 SCL à 1 : départ
 - 2 - SDA à 1 SCL à 1 : arrêt (retour à l'état d'avant prise de contrôle)

Une fois avoir vérifié que le BUS est libre, et pris le contrôle de celui-ci, le circuit en devient le maître : c'est lui qui génère le signal d'horloge

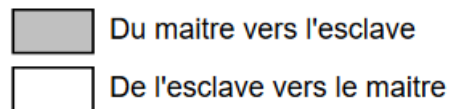
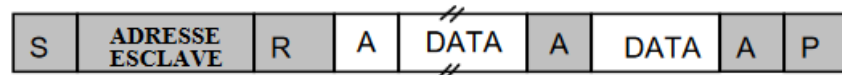
La transmission d'un octet se faire ensuite de cette façon :

- . Le maître transmet le bit de poids fort D7 sur SDA
- . Il valide la donnée en appliquant un niveau '1' sur SCL
- . Lorsque SCL retombe à '0', il poursuit avec D6, etc. jusqu'à ce que l'octet complet soit transmis.
- . Il envoie le bit ACK à '1' en scrutant l'état réel de SDA
- . L'esclave doit imposer un niveau '0' pour signaler que la transmission s'est déroulée correctement
- . Le maître voit le '0' (collecteur ouvert) et peut passer à la suite

Mode maitre transmetteur



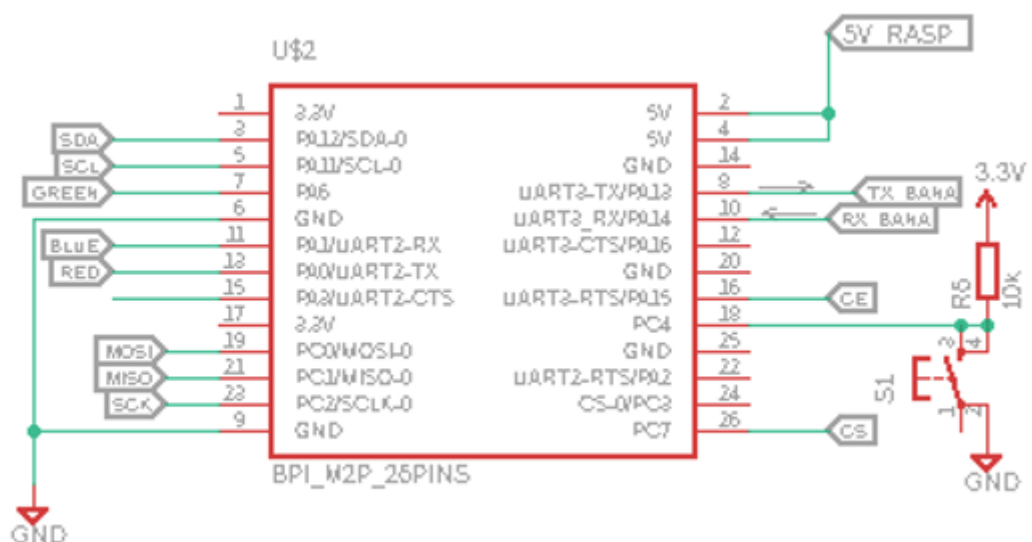
Mode maitre receveur

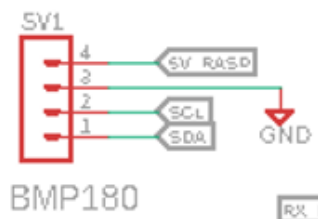


S : Condition de START

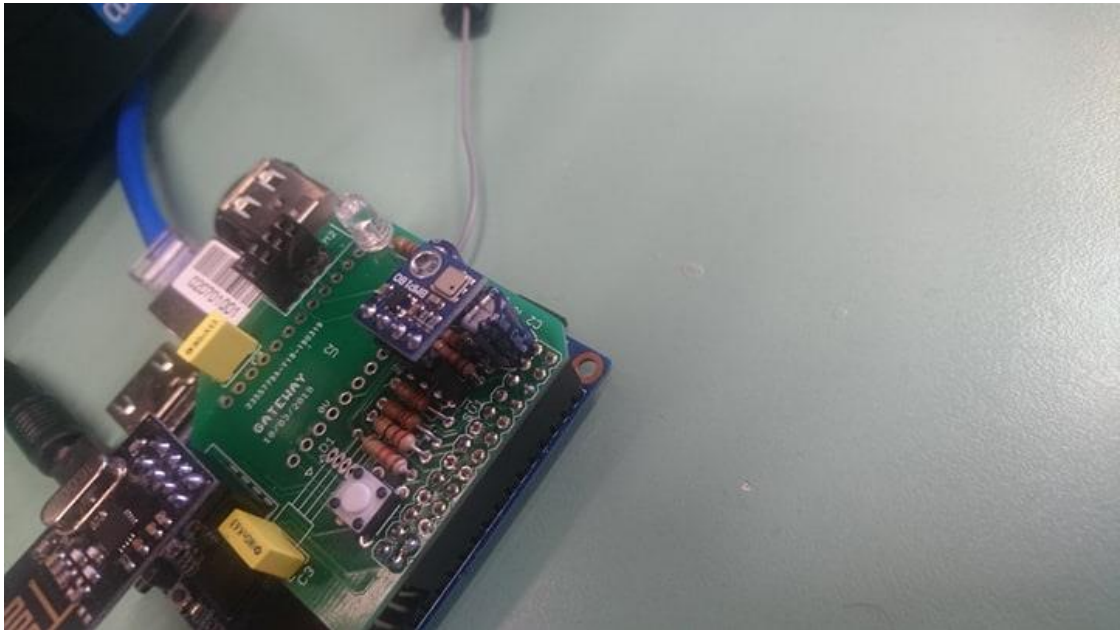
P : Condition de STOP

Maintenant que nous avons compris le principe de fonctionnement d'un BUS I2C, Nous allons utiliser l'I2C de notre carte banana pi pour lire des données d'un capteur, nous avons pour cela dans notre carte Gateway, un connecteur femelle connecté au SDA et au SCL que nous pourrons utiliser.





Ce connecteur est prévu pour le capteur BMP180, munissez-vous d'un de ces capteurs, et le connecter à votre Gateway.



Une fois le capteur connecté, nous allons passer sur notre banana pi afin d'activer l'i2c et télécharger les librairies nécessaires à sa prise en main.

```
sudo armbian-config
```

Une fois dans le menu graphique aller dans :

System → Hardware →

Et activer les i2c :

Bien maintenant que nous avons validé la connexion de notre capteur à notre cible, nous allons appliquer les différentes méthodes vues dans le TP 0 afin de lire les données de ce capteur.

Le noyau (kernel) est le programme qui se trouve dans le répertoire /boot et qui gère le matériel (RAM, flash, I2C, SPI, RTC,...), le réseau, ordonnance les tâches (programmes ou process), gère les systèmes de fichiers. C'est lui qui est chargé en RAM par le bootloader et qui ensuite va lancer le premier process init de votre distribution.

Pour que le noyau accède au matériel, et pour que le banana puisse communiquer avec ceux-ci, on doit connaître l'adresse où le matériel est connecté dans le registre du banana pi.

C'est ce que fait la bibliothèque Wiring Pi.

. Accès en passant par : /dev/mem

Cette partie est la plus facile, en effet une bibliothèque le fait déjà à notre place et nous évite de chercher les adresses manuellement, WiringPi.

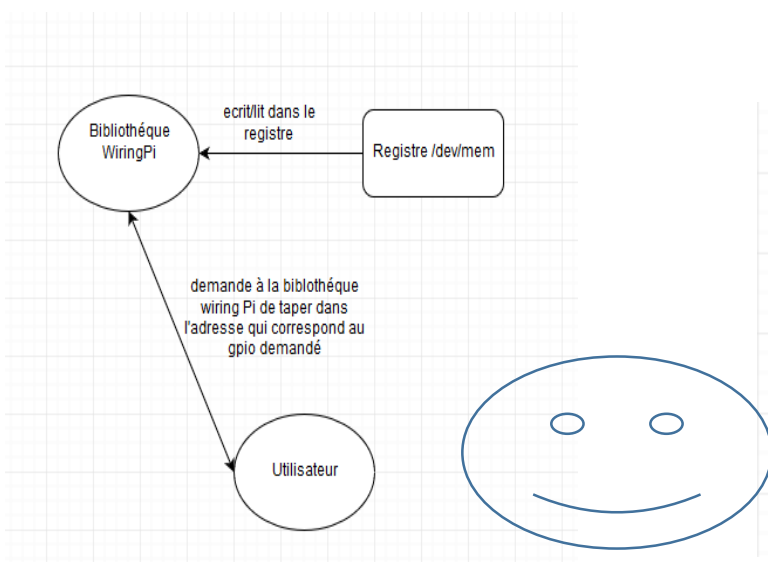
Cette bibliothèque a été codée pour ne pas avoir à trouver dans quel adresse du registre se situe chaque pin, et donc chaque matériel connecté à la cible.

Ce qui veut dire que si on veut piloter un gpio manuellement avec /dev/mem, il faudra trouver l'adresse du registre qui correspond à ce GPIO dans notre processeur.

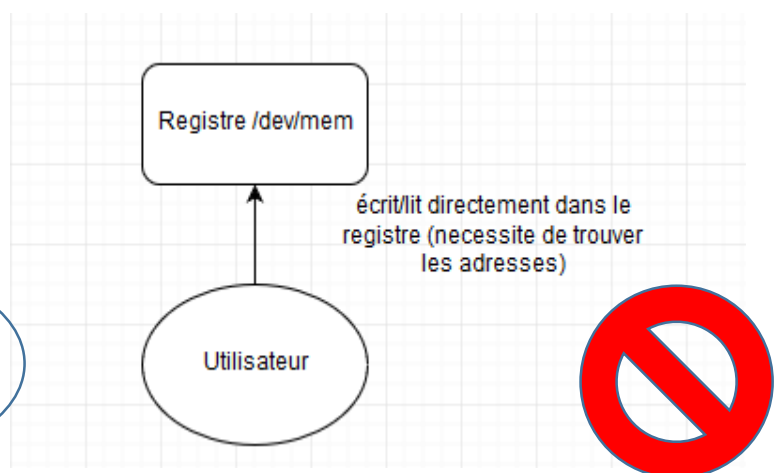
Soit dans notre cas, l'adresse du BUS I2C utilisé par notre capteur.

Attention cependant chaque librairie est codée pour un processeur en particulier, la bibliothèque que l'on a téléchargée est donc adaptée au processeur H3 Allwinner dont notre Banana Pi est équipé, si vous changez de carte avec un processeur BCM par exemple, les adresses de registre changent, il faudra donc changer de librairie et trouver la librairie WiringPi qui correspond à ce nouveau processeur.

Avec Wiring Pi



Sans Wiring Pi



Accès en passant par /dev/i2c-x

En général, pour contrôler nos bus i2c, on se réfère à notre driver kernel, comme ci-dessus, mais nous pouvons aussi y avoir accès via notre espace d'utilisateur à travers le répertoire /dev.

Pour éviter d'avoir à changer de librairie à chaque fois que l'on change de processeur sur notre carte, on peut directement aller écrire/lire dans le bus I2C en ouvrant le fichier i2c dans /dev.

Dans un premier lieu, vous allez tester la méthode avec la bibliothèque WiringPi le code sera fourni.

Dans un second temps, vous aurez à modifier un code qui utilise la méthode fopen en accédant à /dev/i2c-x, la majeure partie du code sera faite, vous n'aurez qu'à faire les calculs et la calibration en vous inspirant du premier code et en utilisant la datasheet du capteur.

Lien de la datasheet :

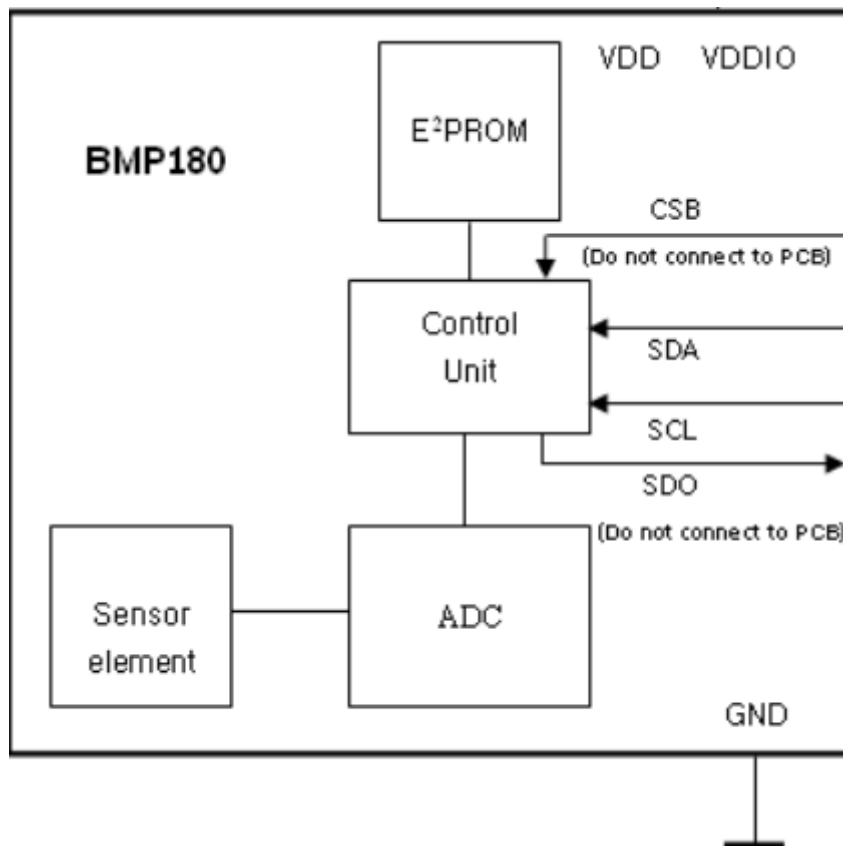
<https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf>

Avant d'entrer dans le code, expliquons comment fonctionne le capteur BMP180.

Le BMP180 est fait pour être directement connecté à un microcontrôleur via le bus I2C. Des données de calibration doivent être utilisées pour compenser les données de température et de pression brutes.

Le BMP180 consiste en un capteur piézorésistif (variance de la résistance en fonction des données reçues (cf cours P2 pour plus de détails) relié à un CAN (convertisseur analogique numérique). La sortie du CAN est ensuite connectée à une unité de contrôle qui gère la communication I2C.

Une mémoire morte, c'est-à-dire une mémoire qui permet de garder les données, même sans alimentation « E2PROM » est aussi connectée à l'unité de contrôle.



C'est la mémoire E2PROM qui enregistre les données de Calibration, avec 176 bits de calibration.

Le capteur envoie directement des données de pression et de température

. UP = donnée de pression (16 à 19 bit)

. UT = donnée de température (16bit)

Pour intégrer les données de calibration au capteur il faudra donc procéder de cette manière :

Les données de calibration sont enregistrées dans un registre, chaque donnée à son adresse : MSB : bit de poids fort LSB : bit de poids faible (Most Significant bit et Least Significant bit)

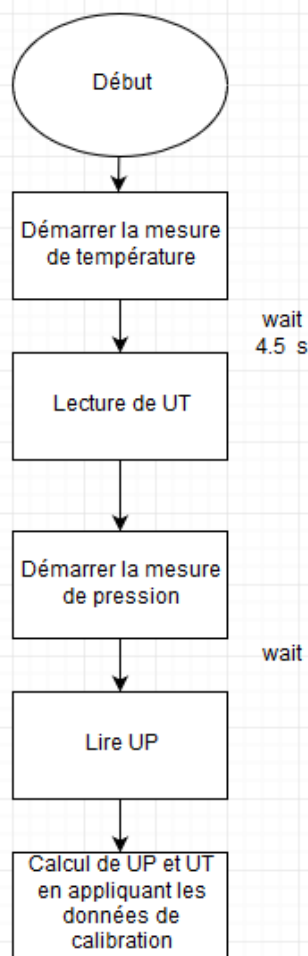


Table 5: Calibration coefficients

	BMP180 reg adr	
Parameter	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

Les formules de calcul sont dans la datasheet et/ou dans le code.

Nous allons installer la librairie WiringPi si ce n'est pas déjà fait :

```

cd
git clone git://git.drogon.net/WiringPi
cd wiringPi
./build

```


Vous devriez avoir quelque chose de similaire à ceci.

Dernière étape avant de commencer à coder, on va tester nos PINS pour voir si elles sont toutes fonctionnelles.

Débrancher tout ce qui est connecté au GPIO du banana pi avant.

```
cd /home/geii/WiringOP/gpio
chmod +x pintest
./pintest
```

```
This is a simple utility to test the GPIO pins on your revision 3
Raspberry Pi.

NOTE: All GPIO peripherals must be removed to perform this test. This
      includes serial, I2C and SPI connections. You may get incorrect results
      if something is connected and it interferes with the test.

This test can only test the input side of things. It uses the internal
pull-up and pull-down resistors to simulate inputs. It does not test
the output drivers.

You will need to reboot your Pi after this test if you wish to use the
serial port as it will be left in GPIO mode rather than serial mode.

Please make sure everything is removed and press the ENTER key to continue
or Control-C to abort...

      The main 8 GPIO pins 0: 7:
--> Pin 1 failure. Expected 1, got 0
One fault detected.
      The 5 SPI pins 10:14: OK
      The serial pins 15:16: OK
      The I2C pins 8: 9: OK
```

ATTENTION : veuillez à bien déconnecter votre Gateway avant de faire le test.

Vous devez avoir OK, si certaines PIN sont en failure, prévenez le professeur.

Nous sommes enfin prêt, vous pouvez reconnecter votre gateway sur votre banana pi. Nous allons installer un code qui va nous permettre de lire les données de pression et température, ce code donne aussi l'altitude calculée à partir de la pression.

Cependant avant, rendez-vous sur ce site, vous y trouverez le code que vous installerez dans la partie suivante.

Ne téléchargez pas ce code !

Cependant ce code est commenté en français, prenez le temps de tout lire, la datasheet du capteur à côté afin de comprendre comment le code est fait, cela vous servira par la suite lorsque vous devrez adapter ce code dans la deuxième méthode sans la bibliothèque WiringPi.

<https://gist.github.com/Mizaystom/c4bc70d93357adb7de64d973eb2967b3>

Une fois le code analysé et compris :

On télécharge et on désarchive le code :

```
cd ~  
wget http://osoyoo.com/driver/pi3_start_learning_kit_lesson_18/bmp180-c.tar.gz  
sudo tar zxvf bmp180-c.tar.gz
```

On rentre dans le répertoire dezippé :

```
cd bmp180-c
```

On compile :

```
sudo gcc -Wall -o bmp180 bmp180test.c -lwiringPi -lm
```

Vous pouvez lancer le programme :

```
sudo ./bmp180
```

```
geii@bpstage:~/bmp180-c$ sudo ./bmp180  
[sudo] password for geii:  
BMP180 Test Program ...  
  
Temperature : 26.00 C  
Pressure : 1005.76 Pa  
Altitude : 61.96 h
```

Vous remarquez que dans le répertoire, il y a aussi un fichier bmp180.h, dans ce fichier il y a tous les adresses de calibration qu'on a vue précédemment, ainsi dans le programme C on a juste besoin d'appeler les variables en prenant soin de bien inclure bmp180.h dans les headers.

Adresses de registre du bmp180 enregistrées dans bmp180.h

 bmp180.h

```
1  #ifndef _BMP180_
2  #define _BMP180_
3
4  //i2c address
5  #define BMP180_Address 0x77
6
7  //Operating Modes
8  #define BMP180_ULTRALOWPOWER 0
9  #define BMP180_STANDARD 1
10 #define BMP180_HIGHRES 2
11 #define BMP180_ULTRAHIGHRES 3
12
13 //BMP185 Registers
14 #define BMP180_CAL_AC1 0xAA //Calibration data (16 bits)
15 #define BMP180_CAL_AC2 0xAC //Calibration data (16 bits)
16 #define BMP180_CAL_AC3 0xAE //Calibration data (16 bits)
17 #define BMP180_CAL_AC4 0xB0 //Calibration data (16 bits)
18 #define BMP180_CAL_AC5 0xB2 //Calibration data (16 bits)
19 #define BMP180_CAL_AC6 0xB4 //Calibration data (16 bits)
20 #define BMP180_CAL_B1 0xB6 //Calibration data (16 bits)
21 #define BMP180_CAL_B2 0xB8 //Calibration data (16 bits)
22 #define BMP180_CAL_MB 0xBA //Calibration data (16 bits)
23 #define BMP180_CAL_MC 0xBC //Calibration data (16 bits)
24 #define BMP180_CAL_MD 0xBE //Calibration data (16 bits)
25 #define BMP180_CONTROL 0xF4
26 #define BMP180_TEMPDATA 0xF6
27 #define BMP180_PRESSUREDATA 0xF6
28
29 //Commands
30 #define BMP180_READTEMPCMD 0x2E
31 #define BMP180_READPRESSURECMD 0x34
32
33 #endif
```

Nous allons maintenant essayer de coder un autre programme en C, sans utiliser la librairie WiringPi pi.

Nous allons bien entendu utiliser la deuxième méthode qui consiste à directement ouvrir le fichier situé dans notre interface utilisateur /dev/i2c-0 à l'aide de la fonction file open.

Le fichier i2c-0 sera donc lu par notre programme et on ira écrire dans le fichier afin de lancer le capteur. On va donc d'abord télécharger un dossier avec la librairie bmp180.h, qui sera intacte.

Dans ce dossier vous aurez aussi le code C avec la méthode file open, cependant le code est incomplet, il faudra donc le compléter.

```
git clone https://gist.github.com/eb26755021b89094685de324e0f7009d.git
```

Pour le prof code complet :

<https://gist.github.com/Mizaystom/922c5d9aa853d6df2d58eb853fb69c2a>

[https://wiki.52pi.com/images/4/47/4.How to use BMP180 Barometer to detect pressur e.compressed.pdf](https://wiki.52pi.com/images/4/47/4.How_to_use_BMP180_Barometer_to_detect_pressur_e.compressed.pdf)

<https://xgoat.com/wp/2007/11/11/using-i2c-from-userspace-in-linux/>

Le dossier s'installe, vous pourrez y accéder en faisant un cd 9 et en appuyant sue tab pour compléter le nom du dossier.

Pour modifier le code, vous pourrez y accéder en en utilisant nano, ou en ouvrant un serveur vnc viewer avec code blocks d'installer : cf vos Tps de formation Linux.

Vous pouvez aussi utiliser votre IDE préféré et transférer les fichiers avec WINSXP, faites bien attention cependant à mettre le fichier d'entête bmp180.h dans le même répertoire que votre code.

Peut être utile :

<https://www.kernel.org/doc/Documentation/i2c/dev-interface>

Une fois votre code complété, il vous faudra installer quelques librairies qu'on utilise dans le code.

SMBus est un protocole qui définit plusieurs manières d'utiliser le bus I2C.. i2c-dev.h contient des fonctions pour toutes ses opérations.

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install python-smbus python-dev
sudo apt-get install libi2c-dev
```

Ensuite on configure le banana pi pour lancer automatiquement l'I2C.

```
sudo nano /etc/modules
```

Rajouter i2c-dev :

```
GNU nano 2.7.4      File: /etc/modules
brcmfmac
g_serial
i2c-dev
```

Ensuite rajouter l'utilisateur dans le groupe i2c afin d'éviter de lancer i2c tools toujours en super utilisateur :

```
sudo adduser geii i2c
```

Vous pouvez ensuite relancer votre cible (sudo reboot).

Une fois votre cible relancée, votre code prêt vous pouvez compiler avec cette commande :

```
gcc bmp180test2.c -o bmp180test2 -lm
```

Bien entendu, n'oubliez pas de vous placer dans le répertoire où est votre code avant `cd /votrerépertoire`.

Lm sert à compiler en appliquant la bibliothèque `<math.h>`.

Vous pouvez exécuter votre programme

```
sudo ./bmp180test2
```

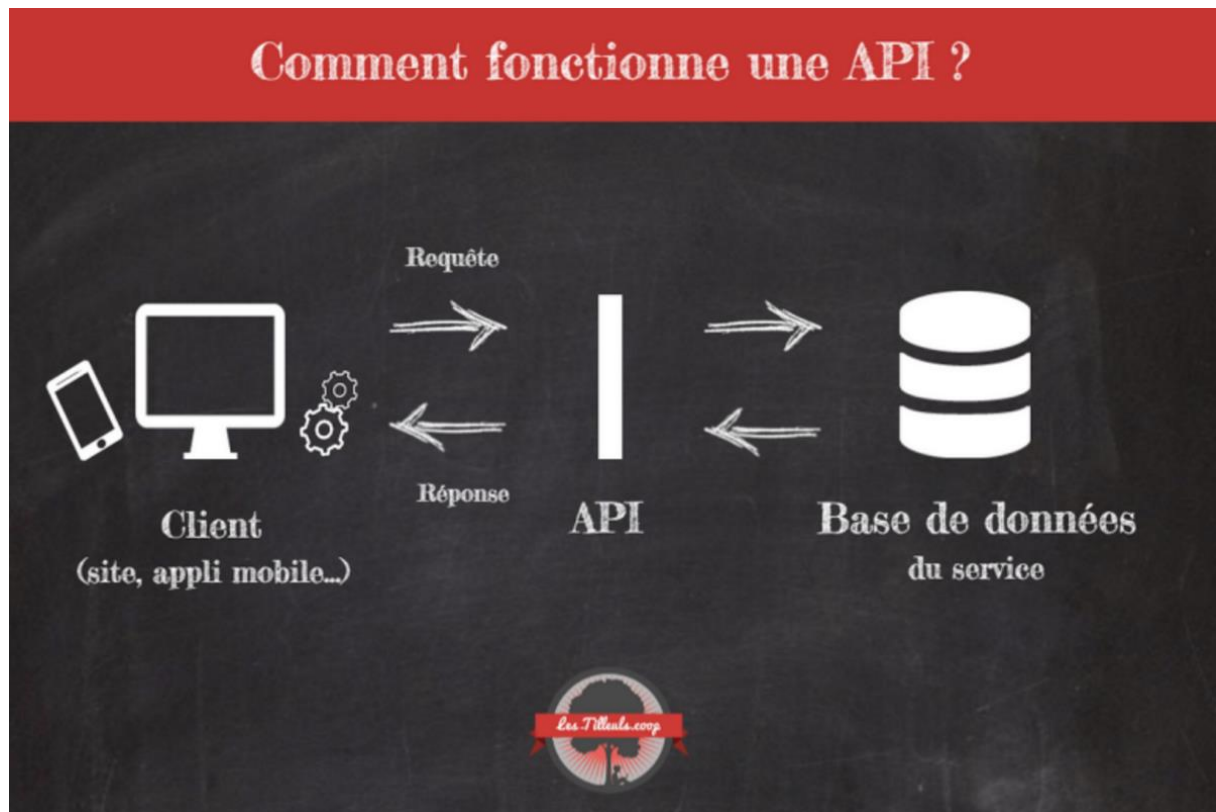
```
geii@bpstage:~/922c5d9aa853d6df2d58eb853fb69c2a$ sudo ./bmp180test2
Test du BMP180 avec ouverture de fichier ...

Temperature : 26.60 C
Pressure :    1009.21 Pa
Altitude :   34.36 h
```

Le programme marche ! (Ne pas hésiter à souffler légèrement sur le capteur pour voir si la température augmente). Nous allons passer à la dernière partie de ce TP.

ThingSpeak est une API et une application open source pour l'« Internet des objets », permettant de stocker et de collecter les données des objets connectés en passant par le protocole HTTP via Internet ou un réseau local.

Avec ThingSpeak, l'utilisateur peut créer des applications d'enregistrement de données capteurs, des applications de suivi d'emplacements et un réseau social pour objets connectés, avec mises à jour de l'état.



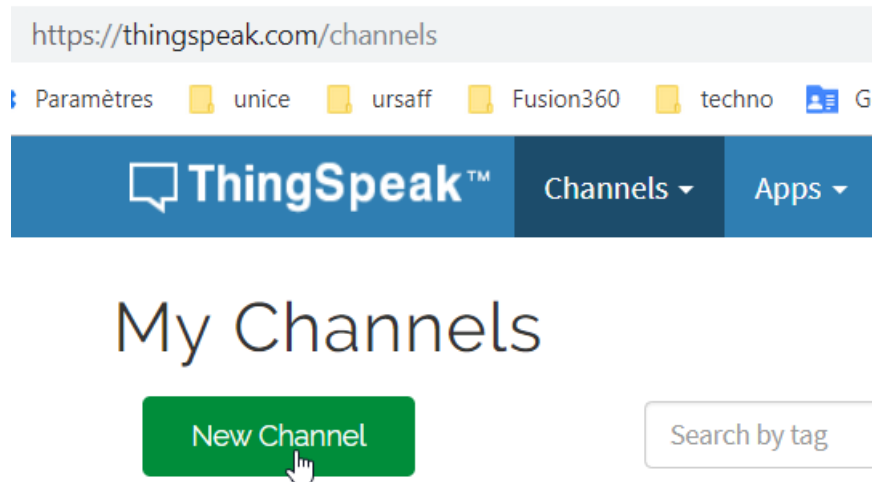
Notre API sera donc Thingspeak, et nous verrons comment mettre en place une base de données dans notre prochaine TP.

Un client peut venir récupérer des informations ou en envoyer via des requêtes http (cas de l'application Android par exemple).

Nous verrons dans cette deuxième partie comment envoyer les données qu'on a reçu de nos capteurs vers cette application Thingspeak.

Vous aurez donc compris qu'il y a plusieurs façons de transmettre de données, que ce soit avec le module radio nrf, avec l'esp8266 et ici directement en réseau à l'aide de notre banana pi.

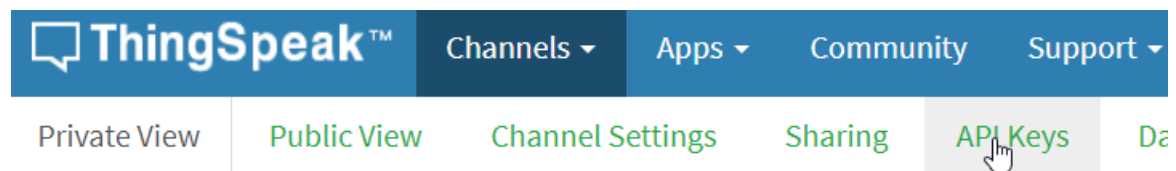
Créer un compte thingspeak si vous ne l'avez pas déjà fait.
Ensuite, créer un channel :



New Channel

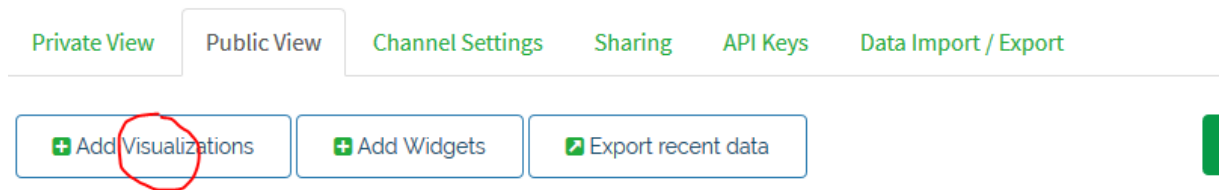
Name	<input type="text" value="testButton"/>	
Description	<input type="text" value="test de 2 boutons <u>bp1</u> et <u>bp2</u> sur <u>M2+</u>"/>	
Field 1	<input type="text" value="bp1"/>	<input checked="" type="checkbox"/>
Field 2	<input type="text" value="bp2"/>	<input checked="" type="checkbox"/>

Inutile de mettre Field2 bp2, il n'y a qu'un seul bouton sur notre Gateway.
Une fois le channel crée, récupérer votre clé API :



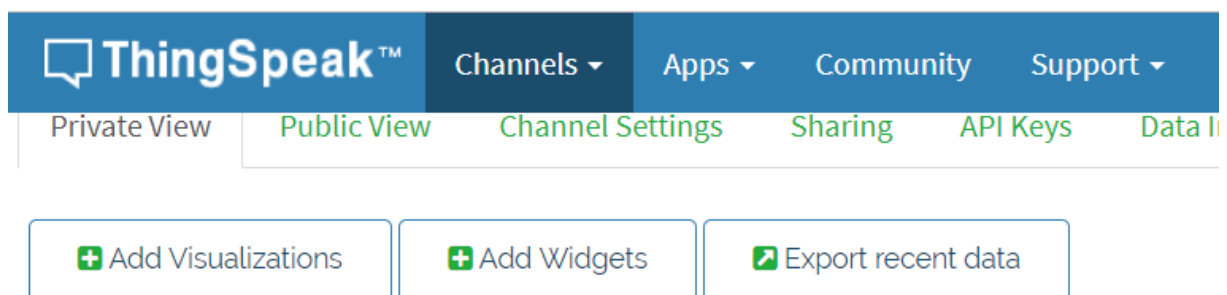
Notez votre clé API, elle est personnelle et vous en aurez besoin pour envoyer des données sur thingspeak.

Une fois votre channel créer et votre clé api notée, cliquez sur add visualizations :



Choisissez ensuite field1.

Une fois votre graphique apparu, il faut le modifier pour qu'il intègre que deux états :

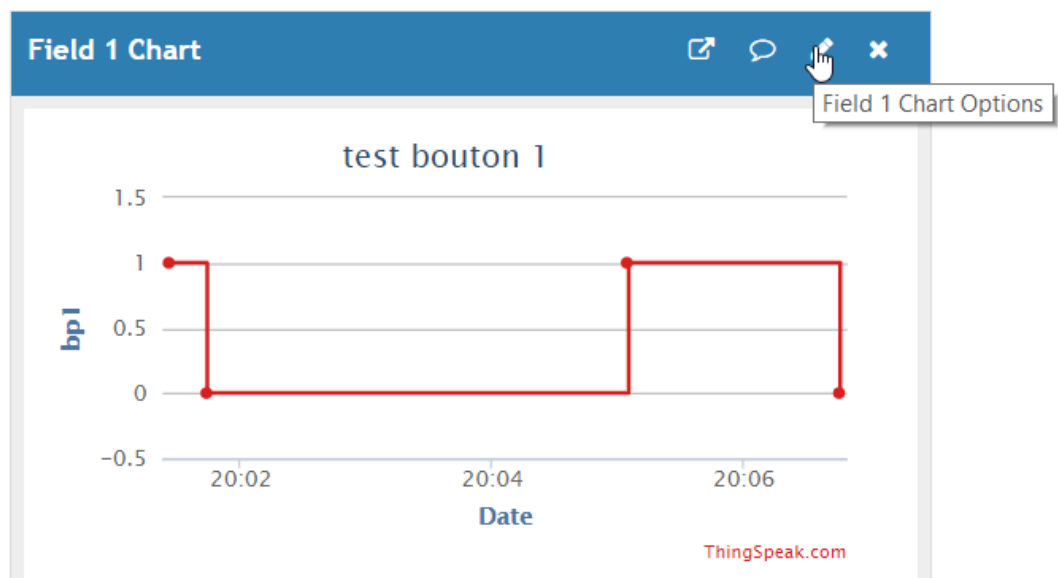


Channel Stats

Created: [10 minutes ago](#)

Last entry: [4 minutes ago](#)


Entries: 4



Appuyez sur Field1 chart options comme ci-dessus et le configurer comme ceci :

Field 1 Chart Options ×

Title:	<input type="text" value="test bouton 1"/>	Timescale:	<input type="text"/>
X-Axis:	<input type="text"/>	Average:	<input type="text"/>
Y-Axis:	<input type="text"/>	Median:	<input type="text"/>
Color:	<input type="text" value="#d62020"/>	Sum:	<input type="text"/>
Background:	<input type="text" value="#ffffff"/>	Rounding:	<input type="text"/>
Type:	<input type="text" value="step"/>	Data Min:	<input type="text"/>
Dynamic?:	<input type="text" value="true"/>	Data Max:	<input type="text"/>
Days:	<input type="text"/>	Y-Axis Min:	<input type="text"/>
Results:	<input type="text" value="60"/>	Y-Axis Max:	<input type="text"/>



Sauvegarder.

Bien maintenant que votre API Thingspeak est prête, nous allons faire un code pour envoyer l'état du bouton à chaque fois que celui-ci change d'état.

Sur votre cible en vous plaçant dans votre répertoire parent (cd) : copier le code incomplet en lançant la commande :

```
git clone https://gist.github.com/231c91ab410f895bf86955913b10e492.git
```

Comme fait pour les codes précédemment (nano ou déplacement avec WINscp), modifier le main, pour qu'il envoie le changement d'état du bouton .

Vous avez bien entendu besoin de votre clé API.

Une fois votre code terminé : compilez avec :

```
gcc thingspeakbp.c -o thingspeakbp -lwiringPi -lcurl
```

Correction :

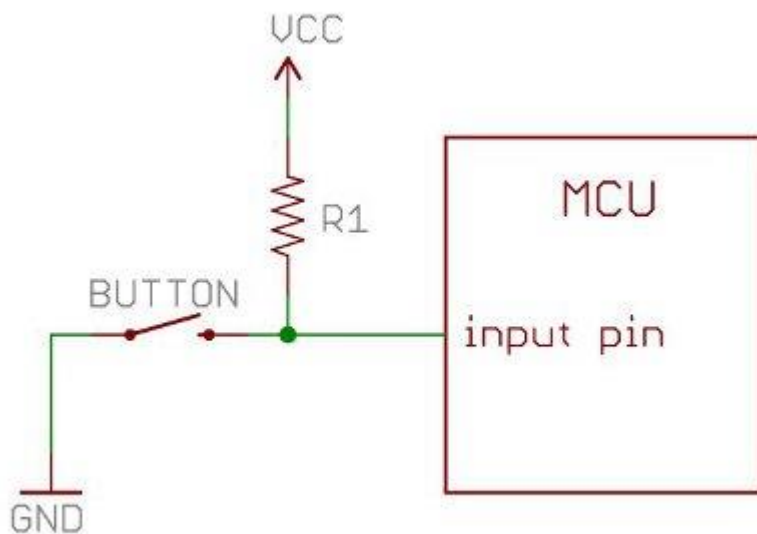
<https://gist.github.com/Mizaystom/6b6d337b5e05effd740794436ff0e284>

Vous pouvez ensuite exécuter votre code :

```
sudo ./thingspeakbp
```

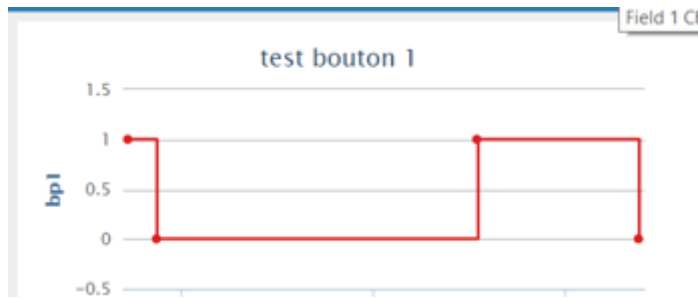
Appuyez sur le bouton, attendre maximum 15 secondes, et normalement l'état du bouton sera envoyé à thingspeak, relâcher le et 15 secondes plus tard le nouvel état est envoyé :

Notez que puisque le bouton est connecté en mode PULL UP, l'état est 0 quand on appui et 1 quand on appui pas : (cf schématique si nécessaire)



```
bp = 1
bp = 0
CHANGE on BP1
UpdateThingSpeak> send: [https://api.thingspeak.com/
0
bp = 1
CHANGE on BP1
UpdateThingSpeak> send: [https://api.thingspeak.com/
1
bp = 1
```

Vérifier sur thingspeak :



Modifier le programme pour qu'à chaque fois qu'il y a un changement d'état et qu'on envoie quelque chose sur thingspeak, la led rgb de la gateway s'allume en vert pendant 5 secondes, puis s'éteint.

Pour savoir quel pin Wpi correspond à la LED verte regarder le schématique pour savoir sur quel pin est connectée la led verte, une fois que vous connaissez la pin physique correspondant à cette led, appliquez ce qu'on a appris au TP0 pour allumer la Led avec :

```
$ sudo sh
# echo 0 > /sys/class/gpio/export
# echo out > /sys/class/gpio/gpio0/direction
# echo 1 > /sys/class/gpio/gpio6/value
```

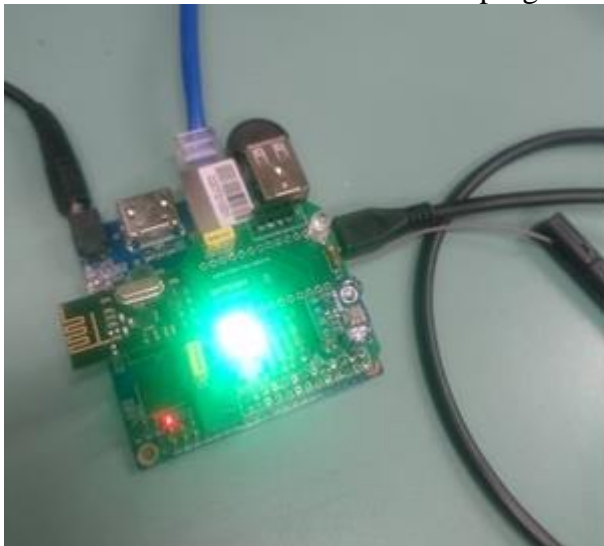
La led est à cathode commune elle s'allume à 0, elle devrait donc s'allumer dès que vous définissez la direction de celle-ci. Vous pourrez l'éteindre avec echo 1.

Quand vous faites l'export :

Si par exemple votre LED est la led physique 7, dans le schématique cela correspond à PAX. Pour l'allumer il faudra donc faire un echo X.

Si c'est PCY il faudra faire echo 64+Y.

Quand votre led est allumée, faire un gpio readall, regardez si c'est bien la pin physique 7 qui est en OUT, et notez la broche Wpi correspondante, c'est celle-là que vous devez utiliser lors de votre définition de broche dans le programme.



Correction :

<https://gist.github.com/Mizaystom/ccf2d03acf0dcac96982e3bc44f928ee>

Avant d'envoyer la température, l'altitude et la pression, nous allons envoyer la température du CPU à thingspeak.

La température du CPU en milidegrés est rangée dans un fichier système que l'on peut consulter en tapant :

```
more /sys/class/thermal/thermal_zone0/temp
```

```
geii@bpstage:~$ more /sys/class/thermal/thermal_zone0/temp
45254
```

A l'aide du code ci-dessous, changer le précédent code afin d'envoyer la température du CPU sur FIELD2 avant tout changement d'état possible.

```
int main()
{
    float systemp, millideg;
    FILE *thermal;
    char temp[20];
    int n;
    while (1)
    {

        thermal = fopen("/sys/class/thermal/thermal_zone0/temp", "r");
        n = fscanf(thermal, "%f", &millideg);
        fclose(thermal);
        systemp = millideg / 1000;
        printf("CPU temperature is %f degrees C\n", systemp);
        sprintf(temp, "%f", systemp);
        UpdateThingSpeak("VOTREAPI", "field2", temp);
        delay(15000);
    }
}
```

Rajout de field sur thingspeak :

Field 1



Field 2



Field 3



Attention ! La version gratuite de thingspeak ne permet qu'une actualisation seulement toutes les 15 secondes, pour éviter cela il faudra actualiser tous les fields au même temps ou attendre 15 secondes entre chaque actualisation. C'est pour cela qu'entre l'envoi de la température du CPU et le test de changement d'état, il faudra attendre 15 secondes, sinon la donnée ne sera pas actualisée sur thingspeak.

Corrigé :

<https://gist.github.com/Mizaystom/2952ec3d2f2b5daef1a20b970435c9e3>

Bien, attaquons-nous maintenant à la dernière partie.

Il faudra combiner le code de lecture des données du capteur et le dernier code que l'on vient de faire.

Lors d'un changement d'état il faudra :

- . Allumer la Led en vert
- . Envoyer le nouvel état du bouton
- . Envoyer la température du CPU
- . Envoyer la température extérieure (capteur).
- . Envoyer la pression atmosphérique (capteur).
- . Envoyer la hauteur (capteur).

Pour cela il faudra changer en plus du main une autre fonction et une déclaration de variable.

Voici la déclaration de la nouvelle variable :

```
char *URLtemplate ="https://api.thingspeak.com/update?api_key=%s&%s=%s&%s=%s&%s=%s&%s=%s&%s=%s";
```

Il faut maintenant que vous vous occupiez de changer la fonction updatethingspeak, et que vous combinez le code du bmp180 (celui avec la bibliothèque WiringPi) pour récupérer les données du capteur.

Bien entendu, il faudra aussi adapter votre thingspeak à toutes ses nouvelles données et se placer dans le dossier où il y a la bibliothèque bmp180.

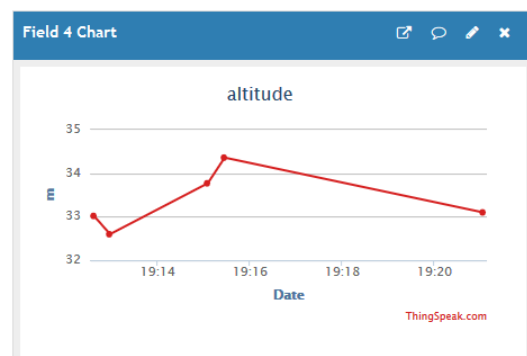
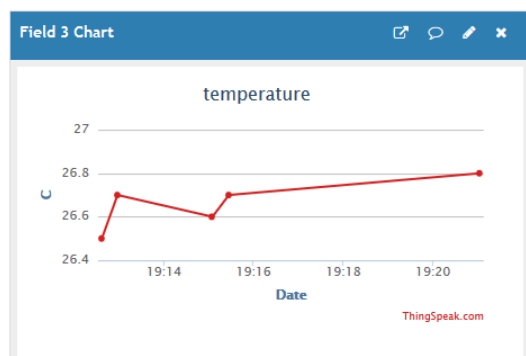
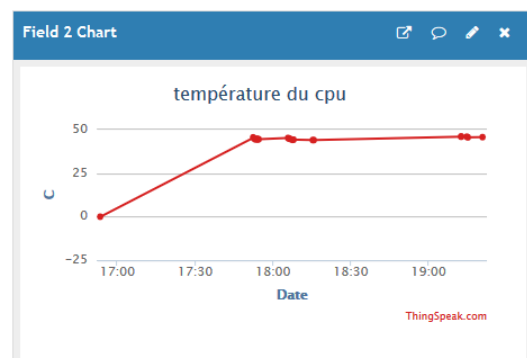
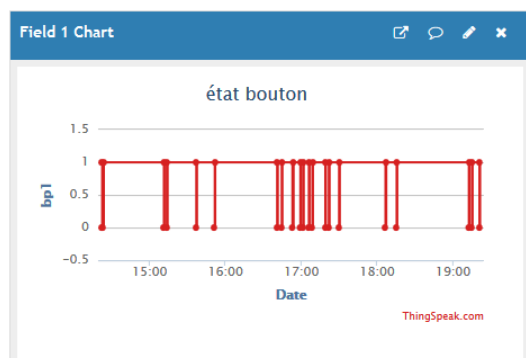
Compilation :

```
sudo gcc -Wall -o bmp180 bmp180test.c -lwiringPi -lm -lcurl
```

Wall permet d'afficher les avertissements

Rendu Final :

```
Temperature : 26.70 C
Pressure : 1009.31 Pa
Altitude : 32.69 h
CPU temperature is 45.980000 degrees C
bp = 1
CHANGE on BP1
UpdateThingSpeak> send: [https://api.thingspeak.com/update?&
SPONSE OK from ThingSpeak
```



Field 5 Chart

Corrigé : <https://gist.github.com/Mizaystom/1956d3c5dc7348504149072ae95db013>*

