

ENSEA

Beyond Engineering

Bus & Réseaux

A la découverte du IMU 10DOF : BMP 280

Compte rendu de travaux pratiques

3^{ème} année MSC

COMPTE RENDU TP BUS & RÉSEAUX

0 - Sommaire

I - Objectif	2
II - Setup du STM32446	2
III - Capteur BMP280	3
1. Adresses I ² C	3
2. Identification	3
3. Mode normal	4
4. Étalonnage du composant	4
5. Registre de température	5
6. Registre de pression	5
7. Calcul de température et de pression compensée	6
8. Résumé	6
IV. Communication I²C	7
1. Identification du capteur BMP280	7
2. Configuration du BMP280	9
3. Récupération de l'étalonnage, de la température et de la pression	9
4. Résultat final : Obtention des datas	10
a. Lecture des datas	10
b. Calcul à partir des datas	11
c. Affichage des datas	11
V - Conclusion	12

I - Objectif

L'objectif de cette séance de travaux pratiques est de mettre en œuvre une chaîne complète d'acquisition et de communication de données entre un microcontrôleur STM32 et un capteur environnemental BMP280, permettant la mesure et la transmission de la température et de la pression atmosphérique.

Dans un premier temps, il s'agira d'établir la communication entre le microcontrôleur et le capteur via le bus I²C. Pour cela, une configuration adaptée de la carte Nucleo sera réalisée sous **STM32CubeIDE** en utilisant la bibliothèque **HAL (Hardware Abstraction Layer)**. Une attention particulière sera portée à la sélection des broches compatibles avec le bus I²C et à la configuration des interruptions associées.

Dans un second temps, la liaison **UART sur USB** sera configurée afin d'assurer la communication entre le STM32 et l'ordinateur hôte. La redirection de la fonction `printf` vers cette liaison permettra de vérifier le bon fonctionnement du système et de faciliter le débogage. Un test de transmission simple (type *heartbeat*) sera effectué pour valider la chaîne de compilation et la communication série.

La troisième phase consistera à exploiter le capteur **BMP280** : identification du composant via la lecture du registre ID, configuration des registres de mesure, et récupération des données brutes de température et de pression. Ces échanges se feront selon le protocole I²C, en respectant la séquence d'écriture et de lecture des registres décrite dans la documentation du capteur.

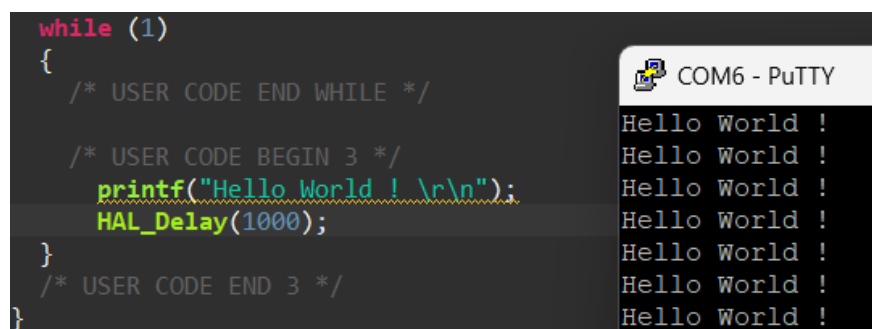
Enfin, les valeurs d'étalonnage internes du BMP280 seront utilisées pour calculer les valeurs **compensées** de température et de pression selon les formules fournies par le constructeur. Ces données, traitées en entier 32 bits pour des raisons de performance, seront transmises à l'utilisateur via la liaison UART sous un format lisible.

L'ensemble de ces étapes permettra d'aborder les notions fondamentales de communication série (I²C et UART), de manipulation de registres internes de capteurs, ainsi que de traitement numérique des mesures issues d'un système embarqué.

II - Setup du STM32446

Dans un premier temps, nous allons devoir nous occuper de configurer notre **STM32F446** (que l'on raccourcira par la suite par STM32). Dans un premier temps, afin de faciliter le **débogage futur**, nous allons faire en sorte que la fonction `printf` soit opérationnelle, c'est à dire qu'elle nous renvoie bien des chaînes de caractères par **UART** sur notre console (ici on utilisera **PuTTY**).

Pour cela, il nous suffit d'ajouter quelques lignes de codes dans le `main.c`. Une fois ceci fait, il ne nous reste plus qu'à brancher notre STM32 à notre ordinateur, regarder sur quel port elle est branchée (ici COM6) et vérifier que cela fonctionne bien en faisant en sorte que toutes les secondes un message "Hello World" s'affiche sur notre console.



```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    printf("Hello World ! \r\n");
    HAL_Delay(1000);
}
/* USER CODE END 3 */
}
```

COM6 - PuTTY

Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !
Hello World !

Figure 1 - Hello World (liaison UART)

III - Capteur BMP280

Intéressons nous désormais au capteur que nous allons utiliser : le BMP280. Avant de commencer toute manipulation, intéressons nous aux informations de ce dernier, et ce en passant par la datasheet.

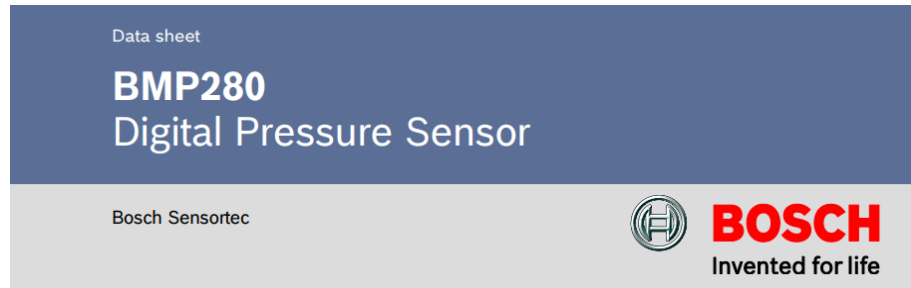


Figure 2 - Datasheet

Pour cela, nous avons une liste de course à effectuer, nous allons devoir trouver dans ce fichier tous les éléments suivants :

- Adresses I²C possibles
- Registre et valeur permettant d'identifier ce composant
- Registre et valeur permettant de placer le composant en mode normal
- Registre contenant l'étalonnage du composant
- Registre contenant la température (et le format)
- Registre contenant la pression (et le format)
- Fonctions permettant le calcul de la température et de la pression compensée (en format entier 32bits)

1. Adresses I²C

Pour trouver les adresses disponibles dans la datasheet, nous avons trouvé cette phrase intéressante : “The 7-bit device address is 111011x. The 6 MSB bits are fixed. The last bit is changeable by SDO value and can be changed during operation”. Cette dernière répond d'ailleurs à notre question. En effet, si nous traduisons cette dernière elle nous annonce que les six bits de poids fort sont fixes (**111011**) et que le dernier bit (x) peut être changer selon la tension que l'on applique au SDO. Ainsi, les deux adresses possibles sont les suivantes : **1110110** (0x76) et **1110111** (0x77) (la première si SDO est relié à la masse, la deuxième si SDO est relié à VDDIO).

Ici la carte est configurée à **0x77**. Nous le savons car c'est écrit sur la carte : “**ADDR : 0x77**”.

Dans le code, l'adresse devra s'écrire **0x77 << 1** car il faut laisser la place à un bit de configuration du bus en mode lecture ou d'écriture R/W.

2. Identification

Maintenant si l'on passe à l'identification de notre composant, vérifier que nous avons bien le bon, nous avons cette fois-ci la phrase suivante : “The “id” register contains the chip identification number chip_id[7:0], which is 0x58. This number can be read as soon as the device finished the power-on-reset.”. On retrouve donc directement l'identifiant de notre capteur : **0x58** ainsi que le registre dans lequel on le retrouve qui est le registre **id**, registre qui a comme adresse **0xD0**.

3. Mode normal

Passons au registre pour lequel, selon la valeur qu'il contient, se situe dans différents modes : normal, endormi ou forcé (normal, sleep ou forced). Ici, nous nous intéresserons principalement au mode normal. Ainsi, on trouve la sélection des modes de fonctionnement du capteur dans le registre **mode** à l'adresse **0xF4**.

3.6 Power modes

The BMP280 offers three power modes: sleep mode, forced mode and normal mode. These can be selected using the mode[1:0] bits in control register 0xF4.

Table 10: mode settings

mode[1:0]	Mode
00	Sleep mode
01 and 10	Forced mode
11	Normal mode

Figure 3 - Tableau des différents modes du capteur

Ainsi, si nous voulons placer notre capteur en mode normal, notre registre devra comporter la valeur **0b11**.

4. Étalonnage du composant

Tout capteur est étalonner selon des règles différentes, intéressons donc nous aux coefficients de calibration de notre BMP280. Ces derniers sont stockés dans une mémoire volatile et permette de compenser les mesures brutes que le capteur obtient. Pour cela, nous avons le tableau suivant nous permettant d'en apprendre plus sur cet étape d'étalonnage.

Register Address LSB / MSB	Register content	Data type			
0x88 / 0x89	dig_T1	unsigned short	0x94 / 0x95	dig_P4	signed short
0x8A / 0x8B	dig_T2	signed short	0x96 / 0x97	dig_P5	signed short
0x8C / 0x8D	dig_T3	signed short	0x98 / 0x99	dig_P6	signed short
0x8E / 0x8F	dig_P1	unsigned short	0x9A / 0x9B	dig_P7	signed short
0x90 / 0x91	dig_P2	signed short	0x9C / 0x9D	dig_P8	signed short
0x92 / 0x93	dig_P3	signed short	0x9E / 0x9F	dig_P9	signed short
			0xA0 / 0xA1	reserved	reserved

Figure 4 - Tableau des registres d'étalonnage

Ainsi, il nous faut lire à l'intérieur des registres allant de **0x88** à **0xA1** afin d'obtenir les coefficients d'étalonnage de notre capteur, en sachant que les six premiers sont pour la températures et les vingt autres pour la pression.

5. Registre de température

Notre capteur sert à mesurer une valeur, mais il faut savoir où cette dernière est stockée. Pour cela, commençons par nous intéresser au registre de température. En fouillant un peu, on trouve donc que les valeurs de températures brutes se trouvent dans les registres allant de **0xFA** à **0xFC** :

Table 25: Register 0xFA ... 0xFC “temp”

Register 0xF7-0xF9 “press”	Name	Description
0xFA	temp_msb[7:0]	Contains the MSB part ut[19:12] of the raw temperature measurement output data.
0xFB	temp_lsb[7:0]	Contains the LSB part ut[11:4] of the raw temperature measurement output data.
0xFC (bit 7, 6, 5, 4)	temp_xlsb[3:0]	Contains the XLSB part ut[3:0] of the raw temperature measurement output data. Contents depend on pressure resolution, see Table 4.

Figure 5 - Tableau des registres de la température brute

De plus, il y a même la possibilité de rajouter un “température oversampling” dans le cas où les clock ne sont pas adéquate entre le capteur et le microprocesseur :

Table 5: osrs_t settings

osrs_t[2:0]	Temperature oversampling	Typical temperature resolution
000	Skipped (output set to 0x80000)	–
001	×1	16 bit / 0.0050 °C
010	×2	17 bit / 0.0025 °C
011	×4	18 bit / 0.0012 °C
100	×8	19 bit / 0.0006 °C
101, 110, 111	×16	20 bit / 0.0003 °C

Figure 7 - Paramètre de l'oversampling de température

6. Registre de pression

Après la température, voyons où est stockée la pression brute. En fouillant un peu, nous trouvons ceci :

Register 0xF7-0xF9 “press”	Name	Description
0xF7	press_msb[7:0]	Contains the MSB part up[19:12] of the raw pressure measurement output data.
0xF8	press_lsb[7:0]	Contains the LSB part up[11:4] of the raw pressure measurement output data.
0xF9 (bit 7, 6, 5, 4)	press_xlsb[3:0]	Contains the XLSB part up[3:0] of the raw pressure measurement output data. Contents depend on temperature resolution, see table 5.

Figure 8 - Tableau des registres de la pression brute

On lit donc ici que les valeurs de pression se retrouvent dans trois registres différents aux adresses allant de **0xF7** à **0xF8**.

7. Calcul de température et de pression compensée

Dernière étape de recherche dans cette datasheet, nous devons trouver comment calculer la température et la pression compensée. Pour cela, Bosch nous recommande directement des formules à utiliser par le biais d'un code que voici :

```
BMP280_S32_t t_fine;
BMP280_S32_t bmp280_compensate_T_int32(BMP280_S32_t adc_T)
{
    BMP280_S32_t var1, var2, T;
    var1 = (((adc_T >> 3) - ((BMP280_S32_t)dig_T1 << 1))) * ((BMP280_S32_t)dig_T2) >> 11;
    var2 = (((((adc_T >> 4) - ((BMP280_S32_t)dig_T1)) * ((adc_T >> 4) - ((BMP280_S32_t)dig_T1))) >> 12) *
        ((BMP280_S32_t)dig_T3)) >> 14;
    t_fine = var1 + var2;
    T = (t_fine * 5 + 128) >> 8;
    return T;
}
```

Figure 9.1 - Calcul de la température compensée (en centi degrés celsius)

```
BMP280_U32_t bmp280_compensate_P_int64(BMP280_S32_t adc_P)
{
    BMP280_S64_t var1, var2, p;
    var1 = ((BMP280_S64_t)t_fine) - 128000;
    var2 = var1 * var1 * (BMP280_S64_t)dig_P6;
    var2 = var2 + ((var1 * (BMP280_S64_t)dig_P5) << 17);
    var2 = var2 + (((BMP280_S64_t)dig_P4) << 35);
    var1 = ((var1 * var1 * (BMP280_S64_t)dig_P3) >> 8) + ((var1 * (BMP280_S64_t)dig_P2) << 12);
    var1 = (((((BMP280_S64_t)1) << 47) + var1)) * ((BMP280_S64_t)dig_P1) >> 33;
    if (var1 == 0)
    {
        return 0; // avoid exception caused by division by zero
    }
    p = 1048576 - adc_P;
    p = (((p << 31) - var2) * 3125) / var1;
    var1 = (((BMP280_S64_t)dig_P9) * (p >> 13) * (p >> 13)) >> 25;
    var2 = (((BMP280_S64_t)dig_P8) * p) >> 19;
    p = ((p + var1 + var2) >> 8) + (((BMP280_S64_t)dig_P7) << 4);
    return (BMP280_U32_t)p;
}
```

Figure 9.2 - Calcul de la pression compensée (en Pascal)

8. Résumé

Proposons un résumé de tout ce que nous venons de trouver à des fins de lisibilité rapide.

Nom du registre	Adresse du registre
ADDR	0x77
ID	0XD0
MODE	0XF4
dig_T[1 : 3]	[0x88 : 0x8D]
dig_P[1 : 9]	[0x8E : 0x9F]
temp_msb	[0xFA : 0xFC]
press_msb	[0xF7 : 0xF9]

Figure 10 - Tableau résumé de la partie sur le BMP280

Désormais, nous avons tout ce qu'il nous faut : registres, valeurs à lire ou écrire dedans et formules. Nous pouvons donc désormais passer au code.

IV. Communication I²C

On passe désormais à la partie codage. Pour cela, sur STM32CubeIDE, nous configurons dans un premier temps l'I²C (l'UART ayant été configuré lors d'une partie précédente). En nous amusant avec l'ioc, nous mettons le SDA sur PB9 et le SCL sur PB8. Puis nous activons toutes les interruptions pour I²C ainsi que pour UART afin de pouvoir par la suite, demandé à notre capteur les valeurs par interruption.

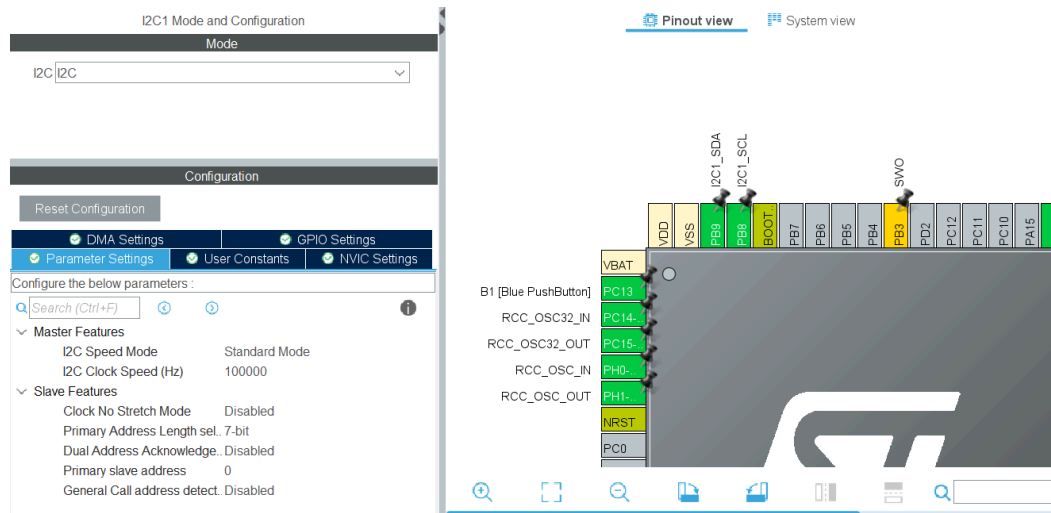


Figure 11 - Fenêtre ioc de notre carte (I²C)

Nous pouvons désormais passer à la partie codage, pour cela, nous allons écrire dans un seul fichier : `main.c`. Nous découperons cette partie codage en plusieurs sous-parties, où nous détaillerons chaque morceau de code. L'entièreté du `main.c` sera à retrouver sur notre github : [LIEN GITHUB](#).

1. Identification du capteur BMP280

Pour l'identification de la valeur du registre ID, Nous initialisons les valeurs des constantes qui vont nous servir :

```
/* USER CODE BEGIN PD */
#define BMP280_I2C_ADDR (0x77 << 1) // SDO = VDDIO & I2C sur 7 bits et non 8
#define REG_ID 0xD0                → Adresse du registre ID (0xD0)
#define REG_ID_VAL 0x58             → Valeur attendue dans le registre ID
#define NORMAL_MODE 0b11           → Capteur mis en normal mode
#define REG_RESET 0xE0              →
#define REG_CALIB 0x88
#define CALIB_LENGTH 26
#define REG_PRESS_MSB 0xF7          → Adresse de stockage de la pression
#define REG_TEMP_MSB 0xFA           → Adresse de stockage de la température
#define REG_CTRL_MEAS 0xF4          → Adresse de registre de mode et de oversampling
/* USER CODE END PD */

/* USER CODE BEGIN PV */
extern I2C_HandleTypeDef hi2c1;     → Handle de l'interface I2C1 utilisé pour communiquer avec le capteur
extern UART_HandleTypeDef huart2;   → Handle de l'UART2 utilisé pour l'affichage via le terminal série
/* USER CODE END PV */
```

Figure 12 - Private define

Puis, dans le main, nous définissons cette fois-ci quelques variables après les différentes initialisations :

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_I2C1_Init();

/* USER CODE BEGIN 2 */
printf("\n=== Initialisation du BMP280... ===\r\n");

/* Variables */
uint8_t id_reg = REG_ID;
uint8_t id;
```

Figure 13 - Définitions de variables

Ensuite, nous allons tester si le capteur est bien présent, si nous avons le bon ID et surtout si nous pouvons communiquer avec lui. Ainsi, pour tester la présence du capteur, nous allons envoyer l'adresse **REG_ID** puis recevoir le contenu du registre correspondant.

```
/* 1 - Présence du capteur */
HAL_I2C_Master_Transmit(&hi2c1, BMP280_I2C_ADDR, &id_reg, 1, HAL_MAX_DELAY);
HAL_I2C_Master_Receive(&hi2c1, BMP280_I2C_ADDR, &id, 1, HAL_MAX_DELAY);

if (id == 0x58)
    printf("BMP280 détecté (ID = 0x%02X)\r\n", id); // %02X : entier à deux chiffres
else
    printf("Erreur : BMP280 non détecté (ID lu = 0x%02X)\r\n", id);
```

Figure 14 - Code pour la vérification de la présence du capteur

Et en effet, nous avons bien dans la console PuTTY, le capteur BMP280 qui est détecté et la lecture du registre qui se fait bien :

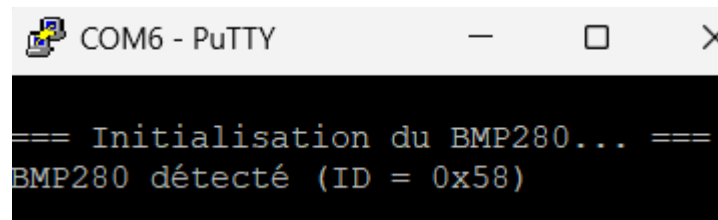


Figure 15 - Fenêtre PuTTY pour vérifier la communication avec le capteur

On communique donc bien avec ce dernier, nous pouvons donc désormais le configurer.

2. Configuration du BMP280

Désormais, pour faire la configuration du BMP280, nous devons mettre le capteur en mode normal, puis avec un pressure oversampling x16, et un température oversampling x2, ce qui donne une valeur de registre égale à 0x57. Nous devons donc remplir un buffer avec l'adresse du registre et cette valeur que nous devons mettre à l'intérieur de ce registre. Cette valeur sera transmise par la même fonction HAL que précédemment.

```
/* Variables */
uint8_t buf[2];

/* 2 - Configuration du capteur */
buf[0] = REG_CTRL_MEAS;
buf[1] = 0x57;
HAL_I2C_Master_Transmit(&hi2c1, BMP280_I2C_ADDR, buf, 2, HAL_MAX_DELAY);
```

Figure 16 - Code pour la configuration de notre capteur

On communique avec notre capteur, nous avons configuré son mode et son oversampling, il ne nous reste plus qu'à gérer l'étalonnage des valeurs et à récupérer ces dernières.

3. Récupération de l'étalonnage, de la température et de la pression

Maintenant, en nous basant sur la datasheet, il ne nous reste plus qu'à récupérer les différentes valeurs d'étalonnage. Pour cela, on envoie une demande au capteur afin que ce dernier nous les envoie.

```
/* Variables */
uint8_t calib[CALIB_LENGTH];

/* Étalonnage */
uint16_t dig_T1, dig_P1;
int16_t dig_T2, dig_T3; dig_P2, dig_P3, dig_P4, dig_P5, dig_P6, dig_P7, dig_P8, dig_P9;
int32_t t_fine;

/* 3 - Coefficients d'étalonnage */
uint8_t reg_calib = REG_CALIB;
HAL_I2C_Master_Transmit(&hi2c1, BMP280_I2C_ADDR, &reg_calib, 1, HAL_MAX_DELAY);
HAL_I2C_Master_Receive(&hi2c1, BMP280_I2C_ADDR, calib, CALIB_LENGTH, HAL_MAX_DELAY);

/* Cf Datasheet */
dig_T1 = (uint16_t)(calib[1] << 8 | calib[0]);
dig_T2 = (int16_t)(calib[3] << 8 | calib[2]);
dig_T3 = (int16_t)(calib[5] << 8 | calib[4]);

dig_P1 = (uint16_t)(calib[7] << 8 | calib[6]);
dig_P2 = (int16_t)(calib[9] << 8 | calib[8]);
dig_P3 = (int16_t)(calib[11] << 8 | calib[10]);
dig_P4 = (int16_t)(calib[13] << 8 | calib[12]);
dig_P5 = (int16_t)(calib[15] << 8 | calib[14]);
dig_P6 = (int16_t)(calib[17] << 8 | calib[16]);
dig_P7 = (int16_t)(calib[19] << 8 | calib[18]);
dig_P8 = (int16_t)(calib[21] << 8 | calib[20]);
dig_P9 = (int16_t)(calib[23] << 8 | calib[22]);
```

Figure 17 - Code pour la récupération des coefficients d'étalonnage

Tout est désormais prêt pour passer à l'étape finale : la récupération des data, et le calcul de ces dernières grâce à la formule que nous avons obtenu lors d'une partie précédente dans la datasheet.

4. Résultat final : Obtention des datas

Nous arrivons enfin à la dernière partie, celle où l'on obtient la pression et la température. Pour cela, nous allons diviser cette sous-partie en trois sous-parties. Une expliquant la lecture des données, une les calculs que l'on effectue dessus et enfin une dernière où l'on affiche ces résultats.

a. Lecture des datas

Commençons par la lecture des données, toujours en nous servant de la datasheet et des résultats que nous avons trouvé au début, il nous suffit de demander au capteur de nous envoyer les valeurs qu'il mesure, puis de les stocker dans une liste.

```
/* Variables */
uint8_t reg_data = REG_PRESS_MSB;
uint8_t data[6];

/* 4 - Lecture des datas */
printf("\n=== Lecture des données === \r\n");
HAL_I2C_Master_Transmit(&hi2c1, BMP280_I2C_ADDR, &reg_data, 1, HAL_MAX_DELAY);
HAL_I2C_Master_Receive(&hi2c1, BMP280_I2C_ADDR, data, 6, HAL_MAX_DELAY);

uint32_t adc_P = ((uint32_t)data[0] << 12) | ((uint32_t)data[1] << 4) | (data[2] >> 4);
uint32_t adc_T = ((uint32_t)data[3] << 12) | ((uint32_t)data[4] << 4) | (data[5] >> 4);
```

Figure 18 - Code pour lire les données du capteur

A partir de "4 - Lecture des datas", le code est placé dans le while(1), pour une lecture en continue des valeurs du capteur.

Après exécution, voilà ce que l'on obtient si l'on résume le tout sous forme d'un tableau :

data[0]	Pression MSB
data[1]	Pression LSB
data[2]	Pression XLSB
data[3]	Temp MSB
data[4]	Temp LSB
data[5]	Temp XLSB

Figure 19 - Résumé du contenu de la liste data

Expliquons désormais nos deux dernières lignes de calculs, dans un premier temps pour la pression puis pour la température. Pour la pression, on récupère dans un premier temps son MSB (qui sont les 8 bits les plus significatifs) que l'on déplace de 12 bits vers la gauche (les 12 premiers bits) puis les 4 bits de LSB décalés de 4 bits vers la gauche qui seront donc les bits 13 à 17 et enfin les 4 bits XLSB que l'on décale à droite pour ne garder que les bits de poids faibles (qui nous intéressent ici). Enfin, on utilise un OU afin de combiner ces trois ensembles de bits pour ne faire qu'un mot de 20 bits. Puis on refait exactement la même chose pour la température.

b. Calcul à partir des datas

Ensuite, à partir du code trouvé dans la datasheet (*partie III.7 Calcul de température et de pression compensée*), nous avons calculé les températures et les pressions compensées avec les valeurs obtenues lors de l'étalonnage. Puis nous effectuons une conversion pour avoir la température en degrés Celsius (au lieu de centiCelsius) et la pression en hPa.

```
/* Variables */
int32_t var1, var2, T;
int64_t var1_p, var2_p, p;

/* 5 - Température compensée */
var1 = (((adc_T >> 3) - ((int32_t)dig_T1 << 1))) * ((int32_t)dig_T2) >> 11;
var2 = (((adc_T >> 4) - ((int32_t)dig_T1)) * ((adc_T >> 4) - ((int32_t)dig_T1))) >> 12 * ((int32_t)dig_T3) >> 14;
t_fine = var1 + var2;

T = (t_fine * 5 + 128) >> 8;

var1_p = ((int64_t)t_fine) - 128000;
var2_p = var1_p * var1_p * (int64_t)dig_P6;
var2_p = var2_p + ((var1_p * (int64_t)dig_P5) << 17);
var2_p = var2_p + (((int64_t)dig_P4) << 35);
var1_p = ((var1_p * var1_p * (int64_t)dig_P3) >> 8) + ((var1_p * (int64_t)dig_P2) << 12);
var1_p = (((int64_t)1) << 47) + var1_p * ((int64_t)dig_P1) >> 33;

if (var1_p == 0) continue; // pour éviter une division par 0
p = 1048576 - adc_P;
p = ((p << 31) - var2_p) * 3125 / var1_p;
var1_p = (((int64_t)dig_P9) * (p >> 13) * (p >> 13)) >> 25;
var2_p = (((int64_t)dig_P8) * p) >> 19;
p = ((p + var1_p + var2_p) >> 8) + (((int64_t)dig_P7) << 4);

int32_t temperature_centi = T; // Température en centidegrès
int32_t pression_centi_hpa = (int32_t)(p/256); // conversion en hPa
```

Figure 20 - Code pour calculer la température et la pression compensée

On a donc désormais tout ce qu'il nous faut pour afficher nos valeurs mesurées, il ne nous reste plus qu'à rajouter cela à la fin de notre code.

c. Affichage des datas

Pour l'afficher, il nous suffit donc d'un printf, que nous décidons d'afficher avec une virgule fixe donnant deux chiffres après la virgule. Et nous répétons cela, toutes les 500ms, avec un HAL_Delay.

```
/* --- 7 : affichage --- */
printf("Temp = %ld.%2ld °C | Pression = %ld.%2ld hPa \r\n",
       temperature_centi / 100, abs(temperature_centi % 100),
       pression_centi_hpa / 100, abs(pression_centi_hpa % 100));

HAL_Delay(500);
```

Figure 21 - Code pour l'affichage des valeurs mesurées



```
COM6 - PuTTY

=== Lecture des données ===
Temp = 23.76 °C | Pression = 988.17 hPa

=== Lecture des données ===
Temp = 23.77 °C | Pression = 988.17 hPa

=== Lecture des données ===
Temp = 23.77 °C | Pression = 988.17 hPa

=== Lecture des données ===
Temp = 23.77 °C | Pression = 988.17 hPa

=== Lecture des données ===
Temp = 23.77 °C | Pression = 988.17 hPa

=== Lecture des données ===
Temp = 23.77 °C | Pression = 988.17 hPa

=== Lecture des données ===
Temp = 23.76 °C | Pression = 988.15 hPa
```

Figure 22 - Affichage des résultats par PuTTY

V - Conclusion

C'est donc avec cette figure finale que l'on termine cette séance de travaux pratique. Nous précisons d'ailleurs que lors de cette dernière partie, nous avons bien fait attention à ne pas afficher un float mais un nombre à virgule fixe avec deux chiffres après cette dernière. Important dans le monde des capteurs, bus et réseaux.