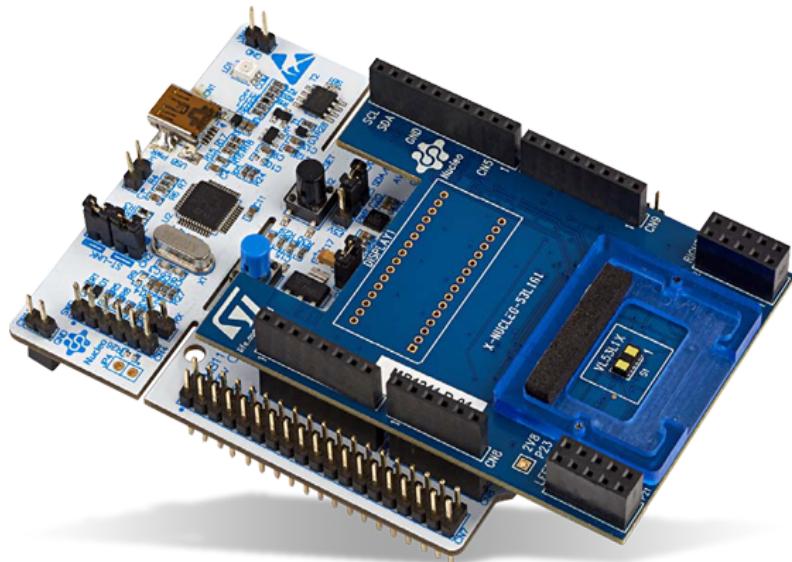

Bureau d'études



TP de BASE

Encadrant:
Thierry Perisse

Auteurs:
Skhiri Mohamed Wassim - Taibi Reda

Contents

1	Introduction	3
2	Objectif	3
3	Materiels Utilisés	3
4	Premier projet sous STM32CubeIDE : LED	5
5	Projet simple sous STM32CubeIDE : BP-LED	5
5.1	Cahier de charge	5
5.2	Programmation	5
5.3	Résultat obtenu	5
6	Projet simple GPIO avec interruptions : BP-LED-Interrupt	6
7	Projet simple : LCD I2C	7
8	Projet1 : Capteur DTH22 + LCD	8
8.1	Cahier de charge	8
8.2	Schéma de câblage	8
8.3	Visualisations oscilloscope	8
8.4	Programmation	9
8.5	Résultat Obtenu	10
9	Projet2 : Capteur BMP280 + LCD	10
9.1	Cahier de charge	10
9.2	Schéma de câblage	10
9.3	Visualisations oscilloscope	11
9.4	Programmation	11
9.5	Résultat Obtenu	13
10	Conclusion	13

Table de matiere

1 Introduction

Ce travail pratique vise à explorer la gamme de cartes STM32 ainsi que les processeurs ARM, en utilisant des capteurs pour répondre à des exigences spécifiques. L'objectif principal est de se familiariser avec les outils de développement de la carte NUCLEO STM32 fournie. Ces séances offriront une opportunité de s'acquérir des compétences sur les logiciels et le matériel largement utilisés dans l'industrie. L'expérience est structurée en deux parties distinctes. Dans la première partie, nous aborderons les principes de fonctionnement. La seconde partie du TP se concentrera sur le processus de lecture des données provenant du capteur DHT22, avec affichage des résultats sur le même écran LCD, suivi par l'utilisation du capteur de pression BMP280. L'objectif global est de comparer les deux protocoles de communication utilisés pour chaque capteur.

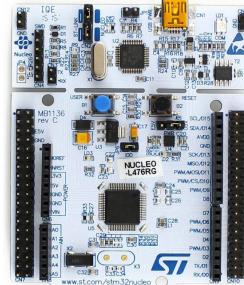
2 Objectif

L'objectif principal de ce TP est d'initier les participants à la manipulation des outils et du logiciel associés. Nous débuterons en apprenant à faire clignoter une LED, puis nous progresserons en ajoutant un bouton-poussoir. Nous aborderons ensuite l'utilisation des interruptions pour une meilleure gestion. Enfin, nous explorerons l'utilisation de capteurs et d'interfaces de communication pour afficher les données de température, d'humidité et de pression, offrant ainsi une expérience complète et pratique dans le domaine de l'embarqué.

3 Matériels Utilisés

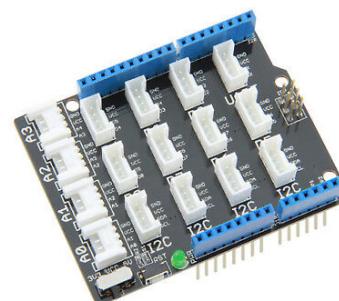
Nucleo L476RG

La carte Nucleo L476RG est une plateforme de développement embarqué qui intègre un microcontrôleur STM32L4 de la famille ARM Cortex-M4. Elle offre une gamme de fonctionnalités et de périphériques permettant le prototypage rapide et le développement d'applications variées dans le domaine de l'embarqué.



Base shield

Le Base Shield STM est une carte d'extension spécifiquement conçue pour faciliter le prototypage et l'expérimentation avec les cartes de développement STM32 de la famille Nucleo. Il offre une interface standardisée pour connecter facilement divers modules et capteurs à une carte Nucleo, permettant ainsi de créer rapidement des prototypes d'applications embarquées sans nécessiter de soudure ou de câblage complexe.



Afficheur LCD

L'afficheur LCD Grove 16x2 Black on Yellow est un type spécifique d'afficheur LCD conçu pour être compatible avec les modules Grove. Il dispose de 16 caractères sur 2 lignes et affiche du texte en noir sur un fond jaune, offrant une bonne lisibilité dans différentes conditions d'éclairage.



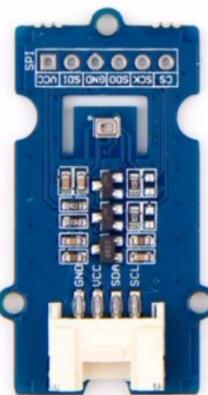
Capteur DHT22

Le DHT22 est un capteur d'humidité et de température numérique de précision. Il est largement utilisé dans les projets électroniques et les applications domestiques pour mesurer à la fois l'humidité relative et la température de l'air ambiant. Ce capteur est connu pour sa précision et sa fiabilité, ainsi que pour sa facilité d'utilisation avec des microcontrôleurs tels que les cartes Arduino ou les cartes STM32. Le DHT22 utilise un protocole de communication propriétaire pour transmettre les données de mesure, ce qui le rend compatible avec une variété de plates-formes de développement..



Capteur Bmp280

Le BMP280 est un capteur de pression atmosphérique numérique de haute précision. Il est utilisé pour mesurer la pression atmosphérique absolue et la température ambiante. Ce capteur est souvent utilisé dans les projets de météorologie, de domotique, de drones et d'autres applications où la mesure précise de la pression atmosphérique est nécessaire. Le BMP280 est apprécié pour sa petite taille, sa consommation d'énergie réduite et sa grande précision, ce qui en fait un choix populaire pour de nombreux projets électroniques. Il peut être facilement interfacé avec des microcontrôleurs tels que les cartes Arduino ou les cartes STM32 grâce à son interface numérique standard, ce qui le rend adapté à une large gamme d'applications.



Câble USB

Le câble USB est utilisé pour la connexion entre la carte et l'ordinateur hôte. Il permet le transfert de données et parfois l'alimentation électrique de la carte.



4 Premier projet sous STM32CubeIDE : LED

Dans cette première phase, nous commençons par la création du notre premier projet en créant un programme visant à faire clignoter une LED. Pour ce faire, nous suivrons les étapes de création et de configuration du projet, en intégrant le code préexistant conçu à cet effet.

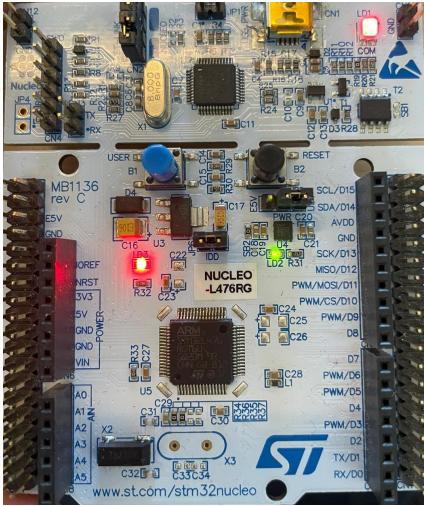


Figure 1: Led allumée

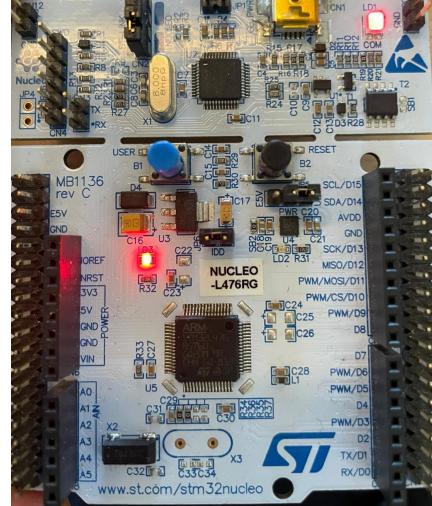


Figure 2: Led éteinte

Notre premier projet a bien fonctionné et nous observons que la LED reste allumée pendant une durée de 500 millisecondes, puis s'éteint pendant 200 millisecondes.

5 Projet simple sous STM32CubeIDE : BP-LED

5.1 Cahier de charge

Dans cette section, nous allons procéder à la programmation en intégrant le bouton-poussoir : en l'appuyant, nous activerons l'allumage de la LED pendant une période déterminée.

Il convient de rappeler que le bouton bleu est destiné à l'utilisateur, tandis que le bouton noir est utilisé pour effectuer le reset.

En suivant les mêmes étapes que précédemment, nous avons créé un nouveau projet, implémenté le code correspondant, puis chargé ce programme sur la carte.

5.2 Programmation

Nous avons intégré le code nécessaire pour détecter l'état du bouton-poussoir et contrôler l'allumage de la LED en conséquence. Cela implique la configuration des broches associées au bouton-poussoir en entrée et à la LED en sortie.

5.3 Résultat obtenu

Nous avons observé le comportement du système en appuyant sur le bouton-poussoir et en vérifiant si la LED s'allume comme prévu. Nous avons également vérifié que la LED s'éteint lorsque le bouton est relâché.

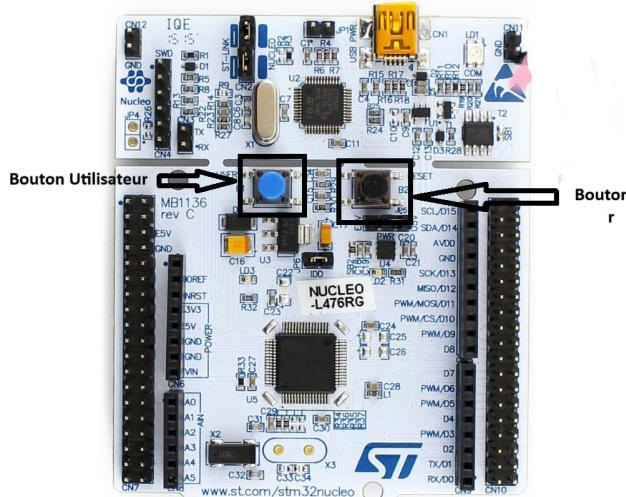


Figure 3: Boutons

En résumé, cette partie du projet a ajouté une fonctionnalité interactive en permettant à l'utilisateur de contrôler l'allumage de la LED en appuyant sur un bouton-poussoir.

6 Projet simple GPIO avec interruptions : BP-LED-Interrupt

Dans la suite, nous allons aborder le concept des interruptions. Un appui sur le bouton-poussoir RESET B2 de la carte NucleoL152RE initialise le programme. La LED LD2 change d'état lorsque le bouton-poussoir utilisateur B1 est relâché ou si un front descendant apparaît sur les broches PA8 ou PA9. En suivant les étapes mentionnées sur le site et en effectuant la configuration appropriée, nous avons réussi à faire fonctionner ce projet avec succès.

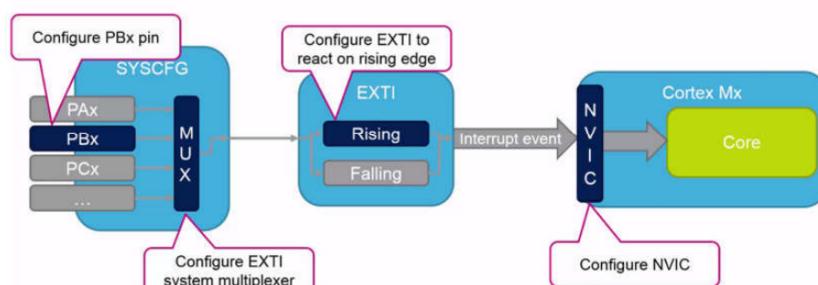


Figure 4: Configuration EXTI

Dans cette partie nous avons introduit les interruptions, qui permettent au microcontrôleur de répondre immédiatement à des événements externes sans nécessiter de surveillance constante de l'état des broches d'entrée. En cas de détection d'un événement, tel qu'un appui sur un bouton-poussoir, une interruption est déclenchée, interrompant l'exécution normale du programme pour gérer cet événement.

Nous avons configuré les broches du bouton-poussoir pour déclencher des interruptions sur un front descendant, permettant ainsi au microcontrôleur de réagir lorsque le bouton est relâché. En conséquence, nous avons programmé le microcontrôleur pour changer l'état de la LED en réponse à ces interruptions, assurant que l'appui sur le bouton-poussoir entraîne l'allumage ou l'extinction de la LED, selon son état précédent.

En résumé, cette partie du projet a étendu la fonctionnalité précédente en ajoutant la gestion des interruptions, ce qui permet une réactivité accrue du système aux événements externes.

7 Projet simple : LCD I2C

Dans cette phase du projet, l'ajout de nouveaux périphériques à notre carte STM32, tels que l'écran LCD et le Base Shield, a été une étape importante. Avant de commencer, il était essentiel de comprendre comment configurer correctement chaque port pour assurer une communication fluide entre les composants. Nous avons opté pour le protocole I2C (Inter-Integrated Circuit) pour connecter l'écran LCD au microcontrôleur. L'I2C est un protocole de communication série bidirectionnel efficace qui nécessite seulement deux fils, ce qui le rend idéal pour notre application.

En outre, pour configurer correctement les connexions, il était important de consulter le pinout du microcontrôleur sur la carte Nucleo et le header Arduino sur la plateforme Mbed. Ces références nous ont fourni les informations nécessaires sur les broches disponibles et leurs fonctions, ce qui nous a permis de sélectionner les ports appropriés et de configurer les connexions de manière adéquate.

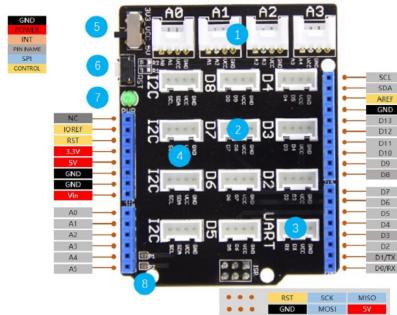


Figure 5: Shield base

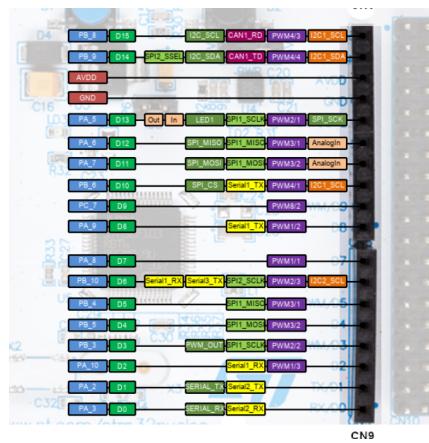


Figure 6: Stm32 pin

Notre premier test consistera simplement à afficher le mot "bonjour" sur l'écran. Après avoir récupéré les bibliothèques fournies et implémenté les codes nécessaires, nous avons réussi à afficher le mot "bonjour" sur l'écran LCD avec succès.



Figure 7: Affichage du mot "Bonjour"

Cette section nous a démontré l'importance de la configuration adéquate en choisissant les ports appropriés, le protocole nécessaire, ainsi que l'utilisation des bibliothèques. Nous utiliserons ces connaissances dans la suite du projet pour afficher les valeurs telles que l'humidité, la pression et la température pour chacun des deux capteurs.

8 Projet1 : Capteur DTH22 + LCD

8.1 Cahier de charge

Après avoir réalisé les premiers projets simples pour tester les composants et obtenir une première idée, nous aborderons maintenant l'utilisation de notre premier capteur de température, le DHT22. Ce projet vise à afficher les valeurs de la température et de l'humidité sur l'écran LCD.

8.2 Schéma de câblage

Avant d'attaquer la programmation et la configuration, nous commençons par réaliser le schéma de câblage pour ce projet. Pour cela, nous avons utilisé Fritzing, et le schéma de câblage est le suivant :

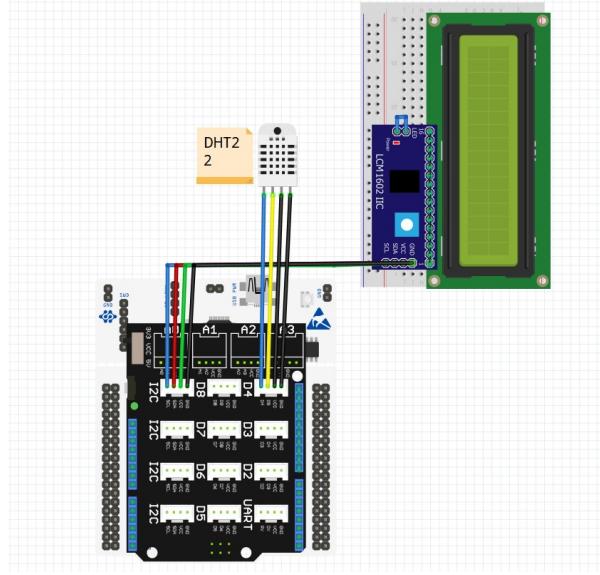


Figure 8: Schéma de câblage N°1

8.3 Visualisations oscilloscope

Dans cette phase, nous commencerons par tester le capteur en utilisant l'oscilloscope. Nous utiliserons le logiciel PicoScope pour vérifier si le capteur répond correctement et génère les mêmes signaux que ceux spécifiés dans sa documentation technique.

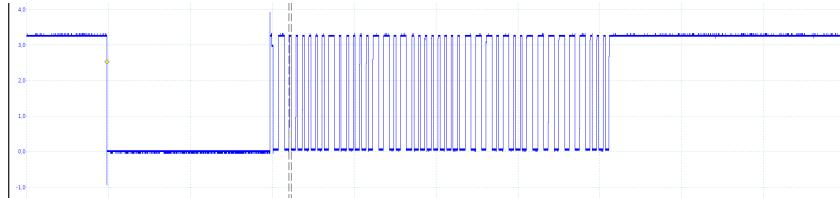


Figure 9: Trame DHT22

Dans la suite ainsi l'objectif principal était d'afficher les délais pour chaque bit transmis et de comparer ces résultats avec les spécifications du fabricant telles que détaillées dans la datasheet. Un exemple illustratif des résultats de l'oscilloscope est présenté ci-dessous.

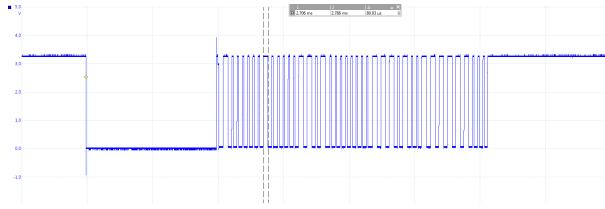


Figure 10: Durée d'un bit à 1

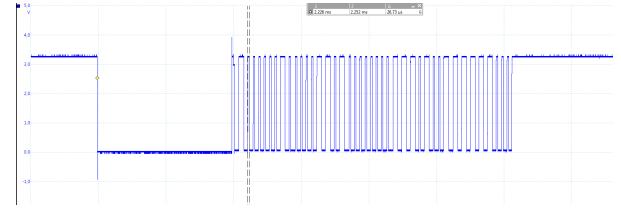


Figure 11: Durée d'un bit à 0

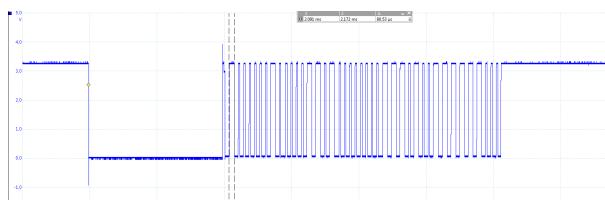


Figure 12: Mesure état haut réponse

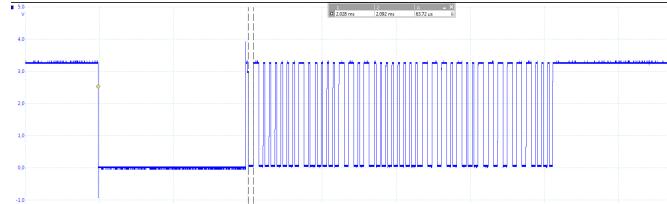


Figure 13: Mesure état bas réponse

Cet ensemble de signaux nous a grandement aidés à mieux comprendre le fonctionnement du capteur et à valider notre programme en le comparant avec les spécifications de la datasheet. En observant les signaux générés par le capteur et en les comparant avec les attentes définies par le fabricant, nous avons pu confirmer la fiabilité de notre implémentation et identifier d'éventuels écarts à corriger. Ainsi, cette analyse approfondie nous a permis d'assurer le bon fonctionnement de notre système et d'optimiser ses performances.

Après l'analyse du signal provenant du capteur DHT22, les octets capturés pour l'humidité sont 00000100 (octet 1) et 00001101 (octet 2), tandis que ceux pour la température sont 00000000 (octet 1) et 11101001 (octet 2). Le calcul de la somme de contrôle donne 11111000. Ces octets encodent respectivement une humidité de 52.5

8.4 Programmation

Après avoir réalisé le câblage, nous avons utilisé le logiciel STM32CubeMX pour configurer les broches nécessaires à notre projet et générer le code en C. Ensuite, nous avons apporté les modifications nécessaires au programme créé dans STM32CubeIDE en suivant les instructions de la datasheet. Les parties importantes du code sont présentées ci-dessous. Le reste du code est disponible dans notre dépôt GitHub.

```

1  while (1)
2  {
3      /* USER CODE END WHILE */
4
5      /* USER CODE BEGIN 3 */
6      start_DHT22();
7      Reception=DHT22_Verification_Reponse();
8      Humidite_octet_1 = Lecture_DHT22();
9      Humidite_octet_2 = Lecture_DHT22();
10     Temperature_octet_1 = Lecture_DHT22();
11     Temperature_octet_2 = Lecture_DHT22();
12     somme = Lecture_DHT22();
13     Temperature= (Temperature_octet_1<<8)|Temperature_octet_2;
14     Humidite=(Humidite_octet_1<<8)|Humidite_octet_2;
15     Temp=(float)(Temperature/10.0);
16     Humi=(float)(Humidite/10.0);
17     affichage_temperature(Temp);
18     affichage_humidite(Humi);
19     HAL_Delay(2500);
20 }
```

Ce code, contenu dans la boucle principale, réalise une série d'opérations pour lire les données de température et d'humidité à partir du capteur DHT22 et les afficher sur l'écran LCD. Tout d'abord,

il initialise la communication avec le capteur en envoyant un signal de démarrage. Ensuite, il vérifie si le capteur répond correctement. En fonction de la réponse du capteur, il lit les octets de données correspondant à la température et à l'humidité. Ces octets sont ensuite combinés pour former des valeurs numériques représentant la température et l'humidité réelles. Ces valeurs sont converties en float et ajustées pour obtenir les unités correctes (degrés Celsius et pourcentage). Enfin, ces valeurs sont affichées sur l'écran LCD, et le programme attend 2,5 secondes avant de répéter le processus pour une nouvelle lecture des données du capteur.

8.5 Résultat Obtenu

Une fois le câblage effectué et le code écrit, nous avons téléchargé notre programme sur la carte pour lancer notre projet. Les résultats de cette démarche sont décrits ci-dessous.



Figure 14: Affichage de la température et de l'humidité

Nous avons réussi à afficher avec succès les deux valeurs, à savoir la température et l'humidité, sur l'écran LCD.

9 Projet2 : Capteur BMP280 + LCD

9.1 Cahier de charge

Après avoir affiché les valeurs de température et d'humidité à partir du capteur DHT22, nous passons maintenant au deuxième capteur, le BMP280. Notre objectif est de travailler avec ce nouveau capteur et son protocole de communication pour afficher à la fois la température et la pression.

9.2 Schéma de câblage

Avant de passer à la programmation, nous devons d'abord créer le schéma de câblage de ce projet sur Fritzing. Ceci nous permettra de mieux comprendre les étapes à suivre avant de passer à la pratique. Voici le schéma de câblage pour ce montage.

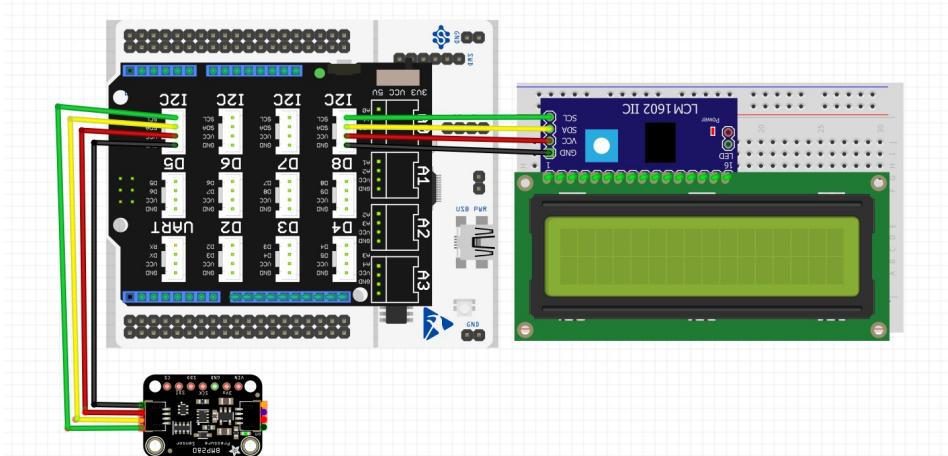


Figure 15: Schéma de câblage N°2

9.3 Visualisations oscilloscope

Dans cette phase, nous commencerons par tester le capteur en utilisant l'oscilloscope. Nous utiliserons le logiciel PicoScope pour vérifier si le capteur répond correctement et génère les mêmes signaux que ceux spécifiés dans sa documentation technique.

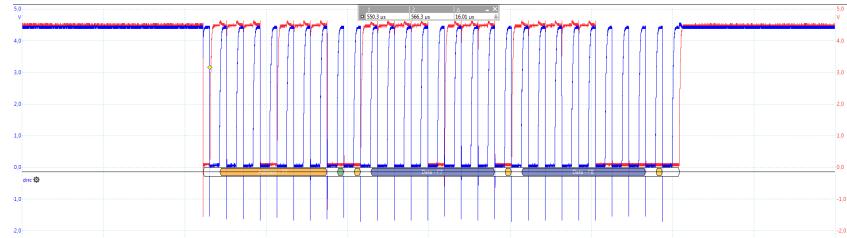


Figure 16: Trame BMP280

Nous constatons d'après ce signal que le capteur répond et fournit des données cohérentes conformément à sa fiche technique.

9.4 Programmation

Maintenant, nous allons passer à la programmation après avoir réalisé le câblage. Nous allons créer un nouveau projet et effectuer la configuration nécessaire en suivant les mêmes étapes que précédemment. Il est important de noter que le capteur BMP280 utilise le protocole de communication I2C, tout comme l'écran LCD. L'I2C est un protocole de communication série bidirectionnel permettant l'échange de données entre différents périphériques connectés sur le même bus, utilisant uniquement deux fils : SDA (Serial Data Line) et SCL (Serial Clock Line).

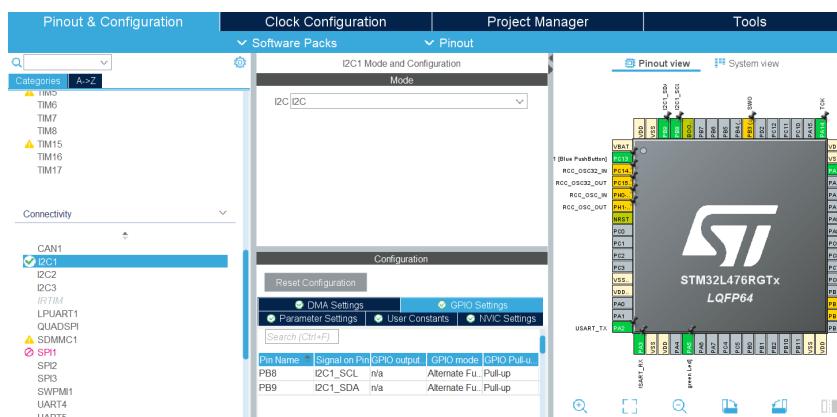


Figure 17: Configuration I2C

Il suffit d'utiliser un seul canal I2C, en l'occurrence l'I2C1, pour connecter à la fois l'écran LCD et le capteur BMP280. Cela est possible car le protocole I2C permet la communication avec plusieurs périphériques sur le même bus, chacun étant identifié par une adresse unique. Ainsi, tant que les adresses des périphériques connectés sont différentes, ils peuvent être contrôlés individuellement via le même canal I2C. Dans ce scénario, le microcontrôleur STM32 dispose d'un seul périphérique matériel I2C (I2C1). En connectant l'écran LCD et le capteur BMP280 à ce même canal I2C, nous pouvons les adresser individuellement en utilisant leurs adresses respectives sur le bus I2C. Cela permet une gestion efficace des ressources matérielles et simplifie la configuration du microcontrôleur.

Ci-dessous, partie du code principal qui permet d'afficher les valeurs de température et de pression détectées par le capteur BMP280 :

```

1  while (1)
2  {
3      /* USER CODE END WHILE */
4      BMP280_Measure();
5
6      char data[255] = "";
7      sprintf(data, "Pressure: %2.1f Temperature: %2.1f C \n\n",
8          (float)Pressure, (float)Temperature);
9      HAL_UART_Transmit(&huart2, (uint8_t*)data, strlen(data), 0xFFFF);
10
11     buf[0] = CAPTEUR_CMD_MSB_press;
12     buf[1] = CAPTEUR_CMD_LSB_press;
13     HAL_I2C_Master_Transmit( &hi2c1, BMP_ADDR, buf, 2, HAL_MAX_DELAY);
14     HAL_UART_Transmit(&huart2, buf, strlen((char*)buf),
15                         HAL_MAX_DELAY);
16
17     // Affichage sur la console UART
18
19     HAL_Delay(100);
20
21     // Affichage sur l'écran LCD
22     lcd_init(&hi2c1,&lcdData);
23     lcd_position(&hi2c1, 0, 0);
24     lcd_print(&hi2c1, "Temp : ");
25     lcd_position(&hi2c1, 7, 0);
26     char temperatureString[10];
27     sprintf(temperatureString, "%2.1f", (float)Temperature, "hPa");
28     lcd_print(&hi2c1, temperatureString);
29     lcd_print(&hi2c1, "C");
30     lcd_position(&hi2c1, 0, 1);
31     lcd_print(&hi2c1, "Press: ");
32     lcd_position(&hi2c1, 7, 1);
33     char pressureString[10];
34     sprintf(pressureString, "%2.1f", (float)Pressure, "C");
35     lcd_print(&hi2c1, pressureString);
36     lcd_print(&hi2c1, "Pa");
37
38     HAL_Delay(5000);
39     /* USER CODE BEGIN 3 */
40 }
/* USER CODE END 3 */
}

```

Ce code représente une boucle infinie dans laquelle les mesures du capteur BMP280 sont effectuées périodiquement. Avant chaque mesure, une commande est envoyée au capteur pour l'inviter à réaliser une nouvelle mesure. Les valeurs de pression et de température mesurées sont ensuite transmises via l'UART pour être affichées sur la console. En parallèle, ces mêmes valeurs sont également affichées sur un écran LCD connecté, après avoir été converties en chaînes de caractères formatées. La boucle est alors mise en pause pendant un certain laps de temps pour permettre une lecture aisée des valeurs affichées.

9.5 Résultat Obtenu

Une fois le câblage effectué et le code écrit, nous avons téléchargé notre programme sur la carte pour lancer notre projet. Les résultats de cette démarche sont décrits ci-dessous.

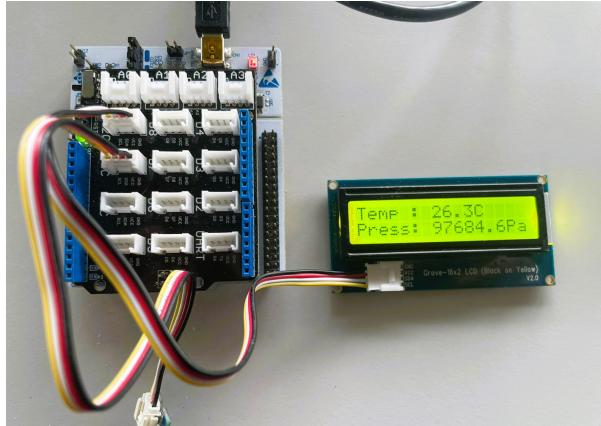


Figure 18: Affichage de la tempreature et la pression

10 Conclusion

En conclusion, ce travaux pratiques a offert une opportunité précieuse aux participants d'explorer la gamme de cartes STM32 et les processeurs ARM, tout en se familiarisant avec les outils de développement de la carte NUCLEO STM32 fournie. À travers une approche structurée en deux parties, les participants ont acquis des compétences pratiques en manipulant les capteurs, en analysant les trames de données, en utilisant le logiciel STM32IDE, et en maîtrisant les protocoles de communication nécessaires. Cette expérience a permis de développer une compréhension approfondie de la configuration, de la programmation et de l'utilisation des capteurs avec la carte STM32, renforçant ainsi les connaissances essentielles dans le domaine de l'embarqué.