Université Paul Sabatier

Compte Rendu de TP :

Bureau d'Etudes

Auteurs :

Abderrahmane AMOUR
Mohammed LATRECHE

Encadrant :

Mr. Thierry PERISSE

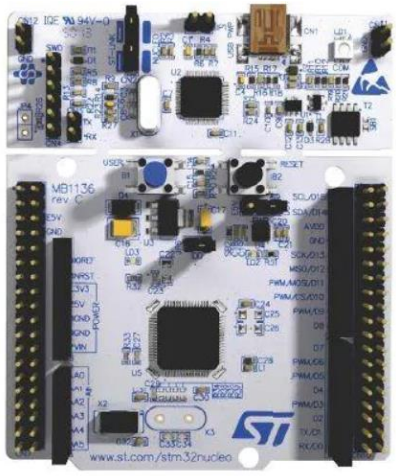
Introduction :

Dans le cadre de ce projet, nous avons exploré l'utilisation de FreeRTOS (Free Real-Time Operating System) pour développer un système multitâche sur un microcontrôleur STM32. FreeRTOS est un système d'exploitation temps réel open-source largement utilisé, offrant des fonctionnalités avancées de planification, de synchronisation et de gestion des tâches.

L'objectif principal de ce projet était de développer un système capable de lire les valeurs de température, de pression et d'humidité à partir d'un capteur, d'afficher ces valeurs sur un écran LCD et de les transmettre via une communication Bluetooth. Pour atteindre cet objectif, nous avons utilisé le microcontrôleur STM32, le capteur BMP280, l'écran LCD ST7735 et les bibliothèques logicielles appropriées.

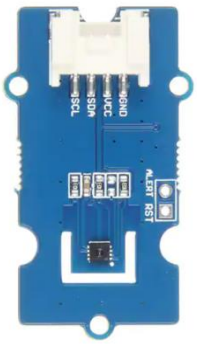
Matériel utilisé :

- Une carte STM32L152RE
- Un capteur SHT31
- Un capteur BMP280
- Un écran LCD ST7735
- Module Bluetooth HC-05



Carte STM32L152RE

Capteur BMP280



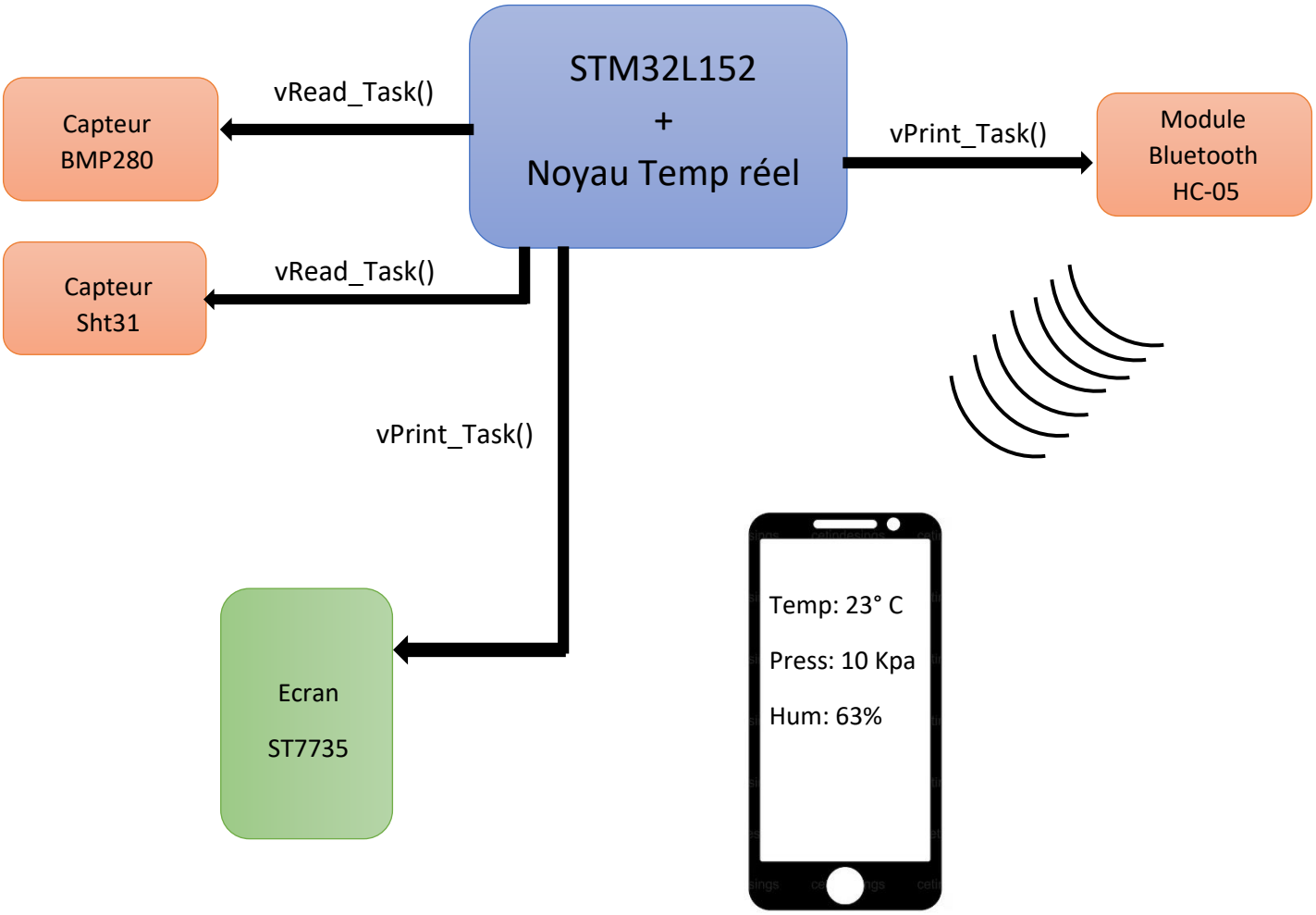
Capteur SHT31

Ecran LCD ST7735

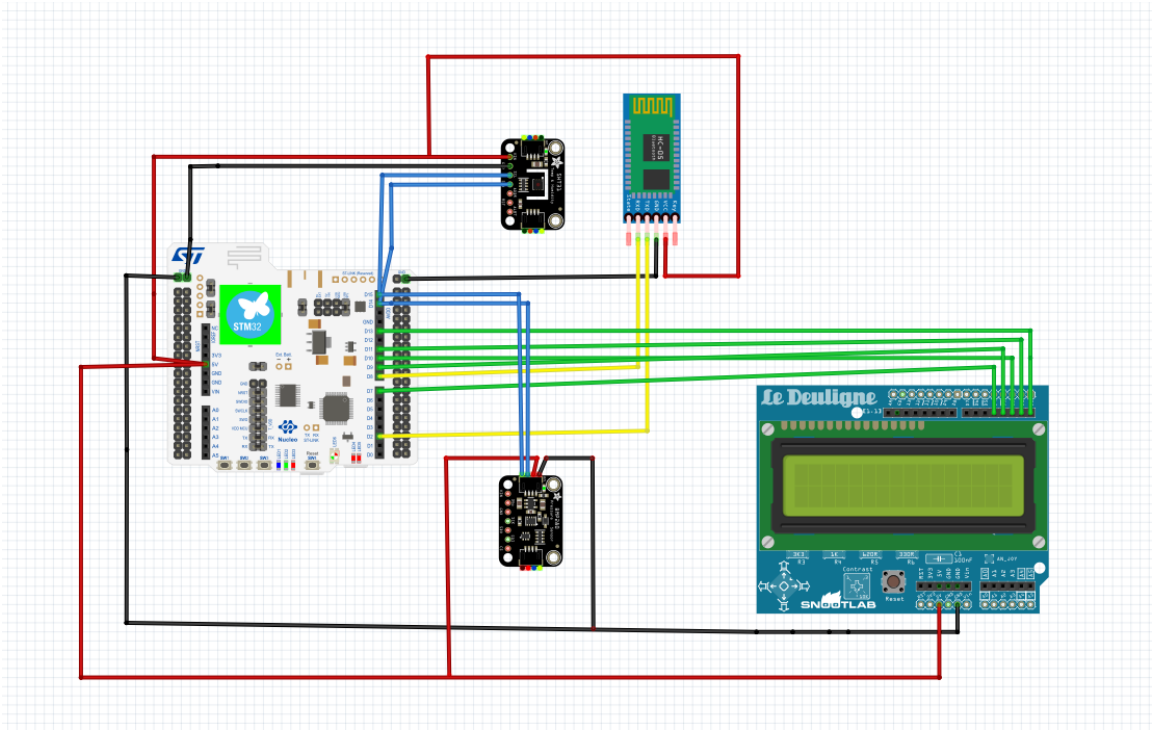
Module HC-05

1. Vision Globale du système :

Dans notre système, l'os contrôle l'exécution des tâches et la transmission des données entre les deux tâches, comme il est illustré sur la figure dessous :



2)- Schéma de câblage :



3. Partie temps réel :

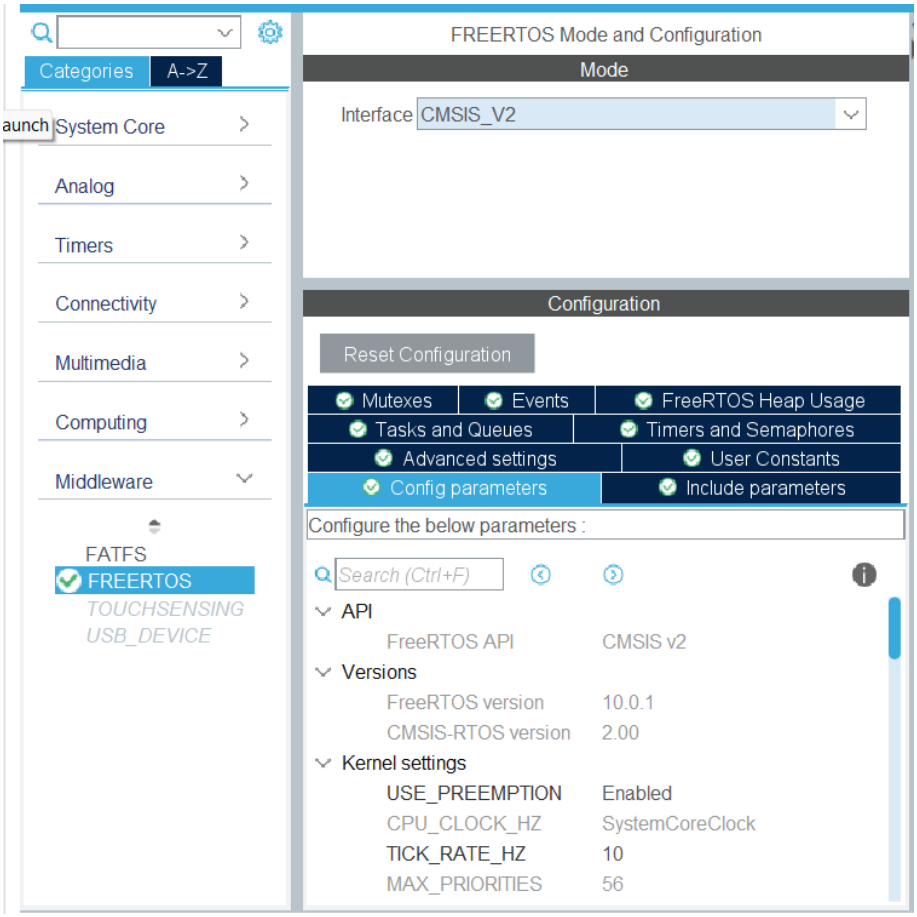
I. FreeRtos : pour la gestion des taches, nous avons utilisé l'os en temp réel FreeRTOS, qui est un système d'exploitation en temps réel open source destiné aux systèmes embarqués. Il a été créé par Richard Barry et est développé et maintenu par une communauté active.

FreeRTOS est conçu pour être compact, portable et facile à utiliser, ce qui en fait un choix populaire pour les applications à contraintes de temps réel sur une large gamme de microcontrôleurs et de processeurs. Il offre des fonctionnalités de planification préemptive et coopérative, ainsi que des mécanismes de gestion des tâches, de la mémoire, des files d'attente, des sémaphores et des temporisateurs.

Le système d'exploitation permet d'organiser le code en tâches, qui peuvent être de taille variable et exécutées de manière concurrente ou séquentielle. Chaque tâche possède une priorité, ce qui permet de garantir l'exécution en temps réel des tâches les plus critiques.

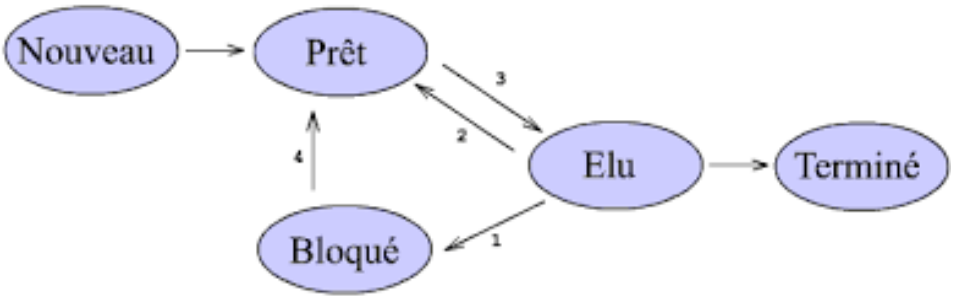
A)- Activation de FreeRTOS en cubeMX :

Sur la fenêtre Middleware, on peut activer freeRTOS, on peut configurer certains paramètres comme la vitesse de switching entre les tâches, la taille mémoire de chaque tâche, on peut même créer des tâches avec cet interface qui seront intégrés dans le code.



B)-Gestion des tâches :

Dans notre système, il existe deux tâches, une pur la lecture des données : vReadTemperature_pressure_humidity_ValuesTask(), l'autre pour l'affichage des données et l'envoi par bluetooth : vPrintTemperatureValueTask, l'ordonnonceur qui utilise l'algorithme d'ordonnancement Roun Robin attribue un quantum de temps fixe à chaque tâche prête à s'exécuter. Lorsque le quantum de temps est écoulé, la tâche en cours est suspendue et la tâche suivante dans la file d'attente est activée comme illustré dans la figure suivante :



D’abord la tâche est créé avec la fonction xTaskCreate(), cette fonction prend plusieurs paramètres :

```
BaseType_t xTaskCreate(TaskFunction_t pvTaskCode,
                       const char *const pcName,
                       const configSTACK_DEPTH_TYPE usStackDepth,
                       void *pvParameters,
                       UBaseType_t uxPriority,
                       TaskHandle_t *const pxCreatedTask);
```

pvTaskCode : C'est un pointeur vers la fonction qui implémente le code de la tâche. Cette fonction doit être de type **void** et prendre un pointeur générique (void*) comme paramètre.

pcName : C'est une chaîne de caractères qui représente le nom de la tâche. Ce nom est utilisé à des fins de débogage et d'identification.

usStackDepth : C'est une valeur entière qui définit la taille de la pile (stack) allouée à la tâche, en mots (taille d'un mot dépend du microcontrôleur ou du processeur utilisé). La taille de la pile doit être suffisante pour exécuter la tâche sans débordement de pile.

pvParameters : C'est un pointeur générique (void*) qui est passé en tant que paramètre à la fonction de la tâche. Il peut être utilisé pour transmettre des données à la tâche lors de sa création.

uxPriority : C'est un entier non signé qui définit la priorité de la tâche. Plus la valeur est élevée, plus la priorité est élevée. Les priorités sont gérées par l'ordonnanceur de FreeRTOS pour déterminer l'ordre d'exécution des tâches.

pxCreatedTask : C'est un pointeur vers une variable de type **TaskHandle_t** qui est utilisée pour récupérer le descripteur de la tâche créée. Ce descripteur peut être utilisé ultérieurement pour manipuler ou supprimer la tâche.

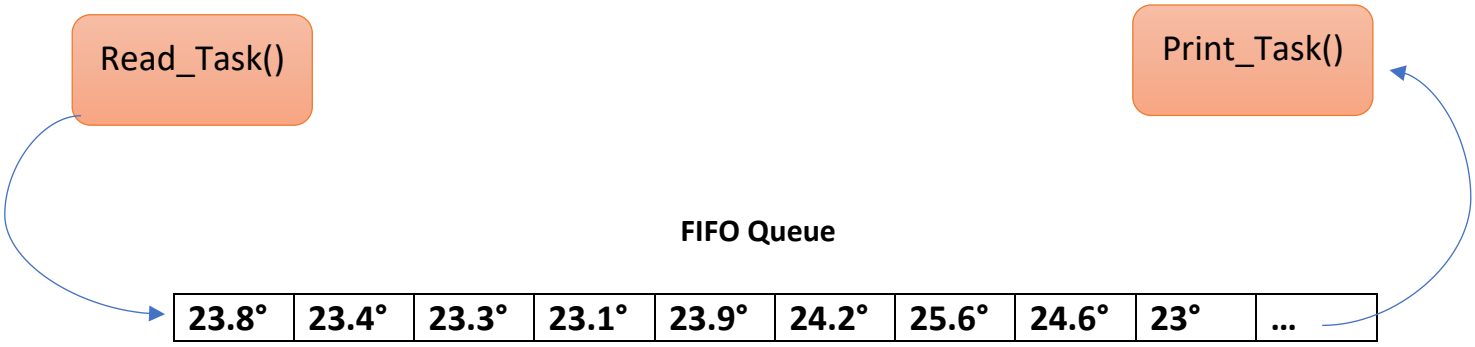
```
xTaskCreate(vReadTemperature_pressure_humidity_ValuesTask,
            "Reading temperature",
            100,
            NULL,
            1,
            &xTempReadHandle);

xTaskCreate(vPrintTemperatureValueTask,
            "Printing temperature",
            300,
            NULL,
            1,
            &xTempPrintHandle);
```

Après la création des tâches, on lance l'ordonnanceur avec : `osKernelStart();`

L'ordonnanceur va s'occuper de l'exécution des tâches et va faire le contexte switching entre les tâches, donc les tâches vont passer par les étapes : créé, en cours d'exécution, bloqué et terminée.

C)-La communication inter-threads :

Pour établir la communication inter-threads, nous avons utilisés des queues FIFO, la tache de lecture capture les données et les met dans la queue, quand le contexte switching est fait, la tâche d'affichage récupère les données de la queue et les affiche en locale et les envoie par bluetooth, nous avons crée trois queues pour les valeurs de la température, la pression, l'humidité :



- Création de la queue : pour crée une que on utilise la fonction xQueueCreate() :

```
QueueHandle_t xQueueCreate(const UBaseType_t uxQueueLength,
                           const UBaseType_t uxItemSize);
```

uxQueueLength : C'est un entier non signé qui représente la longueur maximale de la file d'attente, c'est-à-dire le nombre maximum d'éléments qu'elle peut contenir. La valeur 0 indique une file d'attente de longueur illimitée.

uxItemSize : C'est un entier non signé qui spécifie la taille, en octets, de chaque élément dans la file d'attente. Il est important de noter que la taille de tous les éléments doit être la même. Les éléments peuvent être de taille différente, mais chaque élément doit avoir une taille fixe.

Pour notre cas nous avons créé les tâches comme ceci :

```
xPrintQueue_temp = xQueueCreate(50, sizeof(float));
xPrintQueue_press = xQueueCreate(50, sizeof(float));
xPrintQueue_hum = xQueueCreate(50, sizeof(float));
```

-L'envoi d'une valeur dans la queue :

Pour insérer une valeur dans la queue nous avons utilisé la fonction :

BaseType_t xQueueSend(QueueHandle_t xQueue, const void *pvItemToQueue, TickType_t xTicksToWait);

xQueue : Il s'agit du descripteur de la file d'attente à laquelle vous souhaitez envoyer des données.

pvItemToQueue : C'est un pointeur vers les données que vous souhaitez envoyer à la file d'attente. Il est généralement nécessaire de passer un pointeur vers les données (ou une copie des données) que vous souhaitez transmettre.

xTicksToWait : C'est le délai maximum (en ticks) que la tâche est prête à attendre si la file d'attente est pleine. Si la file d'attente est pleine et que le délai d'attente s'écoule, la fonction renverra errQUEUE_FULL.

- La récupération des données de la queue :

Pour récupérer une valeur de la queue (qui est la première insérée) on utilise la fonction : `xQueueReceive`

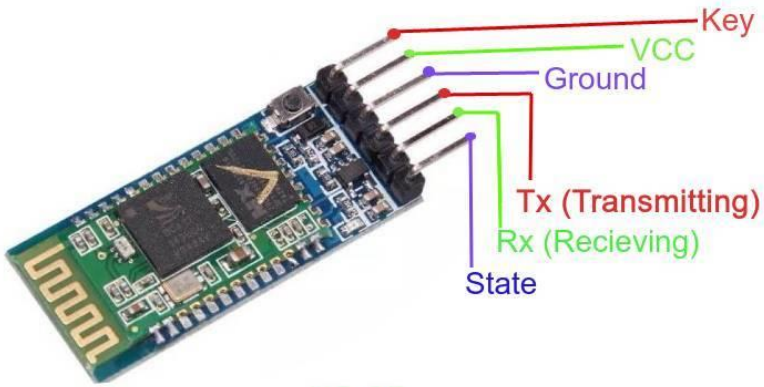
```
BaseType_t xQueueReceive(QueueHandle_t xQueue,
                         void *pvBuffer,
                         TickType_t xTicksToWait);
```

pvBuffer : C'est un pointeur vers une mémoire tampon (buffer) où l'élément reçu sera stocké.

4. Partie Bluetooth :

Le module Bluetooth HC-05 est couramment utilisé pour établir une connexion sans fil entre deux appareils électroniques tels que des smartphones, des ordinateurs, des microcontrôleurs, etc. Il est caractérisé par son faible coût, sa faible consommation d'énergie et sa facilité d'utilisation. Le module est alimenté à partir de 3,6 V à 6 V et peut être connecté à un microcontrôleur UART pour permettre la communication sans fil entre deux dispositifs.

Le HC-05 utilise le protocole Bluetooth version 2.0 avec EDR (Enhanced Data Rate) pour assurer une transmission de données rapide et fiable. Il a une portée de communication d'environ 10 mètres et peut être configuré en mode maître ou esclave.



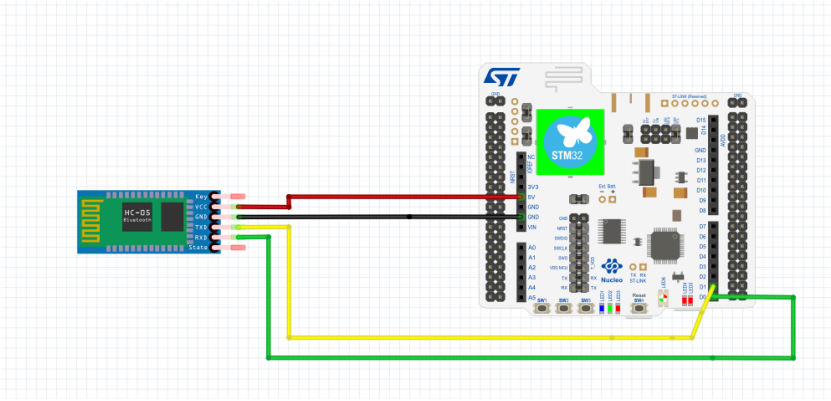
HC-05 (Bluetooth Module)

Le HC-05 avec notre projet :

Pour ce faire, le module HC-05 doit être connecté à la carte STM32, qui sera chargée de collecter les données de température et de pression à l'aide de capteurs SHT31 &BMP280. On a programmé la carte STM32 pour transmettre ces données via une connexion Bluetooth à l'application mobile en utilisant la ciommunication UART du microcôntroleur.

le projet implique la création d'un système station de météo à distance de la température et de la pression en utilisant le module Bluetooth HC-05 avec la carte STM32 et une application mobile.

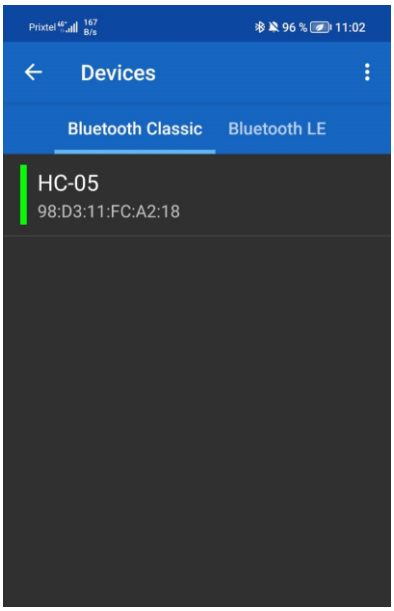
Le schéma de cablage pour notre projet :

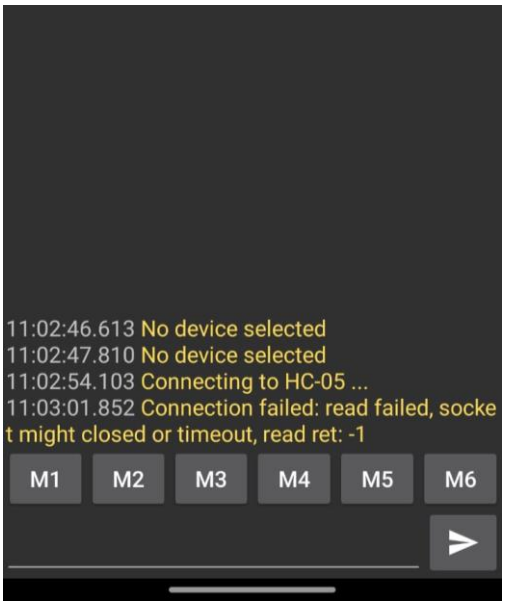


Le terminal Bluetooth & l'application Bluetooth :

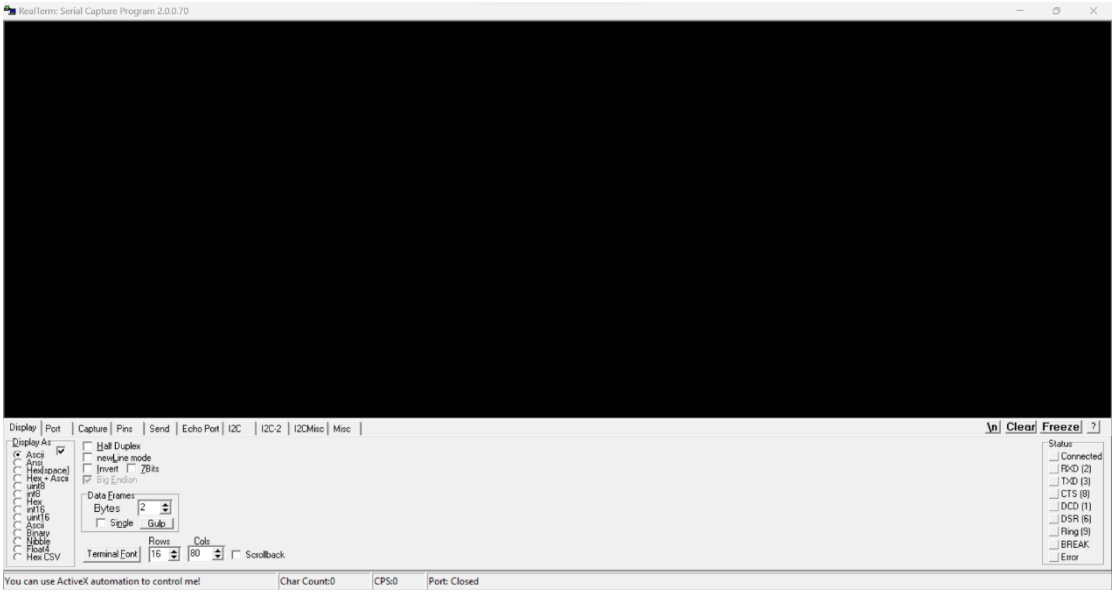
Dans notre projet, nous avons utilisé deux applications pour récupérer les données de température, pression et humidité à partir d'un microcontrôleur STM32. Nous avons utilisé l'application mobile Terminal Bluetooth et le terminal Bluetooth sur le PC pour réaliser cette tâche.

L'application mobile Terminal Bluetooth a été utilisée pour récupérer les données de température, pression et humidité en utilisant la communication Bluetooth. Cette application offre une interface utilisateur conviviale et des fonctionnalités pour visualiser les données en temps réel. Les données peuvent également être enregistrées sous forme de fichiers CSV pour une utilisation ultérieure.





Le terminal Bluetooth sur le PC est une application qui offre une interface en ligne de commande pour communiquer avec les périphériques Bluetooth. Nous avons utilisé cette application pour récupérer les données de température, pression et humidité en utilisant la communication Bluetooth. Les données ont été affichées sur la ligne de commande et peuvent également être enregistrées sous forme de fichiers CSV pour une utilisation ultérieure.



Dans l'ensemble, l'utilisation de l'application mobile Terminal Bluetooth et du terminal Bluetooth sur le PC a permis une récupération efficace des données de température, pression et humidité à partir du microcontrôleur STM32. Les deux applications offrent des fonctionnalités pour visualiser et enregistrer les données, ce qui est utile pour une analyse ultérieure.

Programmation pour la fonction de communication :

Pour la fonction Bluetooth les données de température, pression et humidité doivent être acquises à partir des capteurs en utilisant les interfaces appropriées I2C Les données acquises sont stockées dans des variables pour un traitement ultérieur.

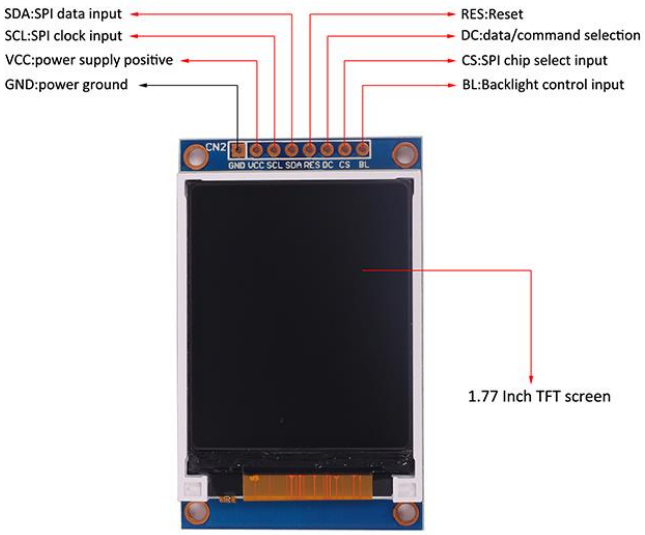
Envoi des données via Bluetooth : Les données acquises sont envoyées via Bluetooth en utilisant les commandes appropriées. Les données sont encapsulées dans un paquet Bluetooth et sont envoyées en utilisant l'interface de communication Bluetooth configurée précédemment.

```
while (1)
{
  /* USER CODE END WHILE */
    HAL_UART_Transmit(&huart1,buffer, sizeof(buffer),100);
    HAL_Delay(500);

  /* USER CODE BEGIN 3 */
```

Réception des données sur l'application mobile ou le terminal Bluetooth : Les données sont reçues sur l'application mobile ou le terminal Bluetooth configuré pour recevoir les données. Les données sont ensuite décodées et affichées en temps réel ou stockées pour une utilisation ultérieure.

5)- Affichage en locale :

Pour afficher les valeurs des mesures en locale nous avons utilisé l'écran ST7735, cet écran est connecté à la carte stm32 avec le protocole SPI :



On active le protocole SPI avec CubeMX, on utilise les pins suivants :

PC7 : Reset / **PA9** : DC / **PB6 :** CS / **PA7:** MOSI / **PA5 :** Clock.

On utilise les fonction suivantes :

ST7735_Init(); pour initilaliser l'écran

ST7735_DrawString(1,4,Data_temp,ORANGE); pour affiche rune chaine de caractère

ST7735_FillScreen(uint16_t color); pour remplir l'écran avec une couleur

6)- Conclusion :

En conclusion, ce projet met en œuvre l'utilisation de FreeRTOS pour développer un système multitâche sur un microcontrôleur STM32. Le projet comprend la lecture des valeurs de température, de pression et d'humidité à partir d'un capteur BMP280, ainsi que l'affichage de ces valeurs sur un écran LCD ST7735 et leur transmission via une communication UART.

L'architecture logicielle est basée sur FreeRTOS, qui fournit une planification et une gestion des tâches efficaces. Deux tâches ont été créées : une tâche de lecture des valeurs de température, de pression et d'humidité à partir du capteur, et une tâche d'affichage de ces valeurs sur l'écran LCD et leur transmission via UART.

La communication entre les tâches est réalisée à l'aide de files d'attente (queues). Les valeurs lues sont placées dans des files d'attente spécifiques, puis récupérées et affichées par la tâche d'affichage. Cela permet une communication sûre et synchronisée entre les tâches.

Le système fonctionne de manière fiable et stable, en exploitant les capacités de planification de FreeRTOS pour exécuter les tâches de manière concurrente. Les files d'attente garantissent la cohérence des données et évitent les problèmes de concurrence.

Ce projet démontre l'efficacité et la flexibilité de FreeRTOS pour développer des systèmes embarqués multitâches. Il offre une solution robuste pour gérer des tâches concurrentes, facilitant ainsi le développement d'applications temps réel sur des microcontrôleurs.