

FILIÈRE : INGÉNIERIE SYSTÈMES NUMÉRIQUES (ISN / SLE)

PROJET MODULE SYSTÈMES TEMPS-RÉELS TP
FREERTOS

Réalisation d'un capteur / afficheur de CO²
-température hygrométrie, sur détection de
personnes

Préparé par :

OTHMENE DEROUICH
WALID BRINI



Enseignant :

PROFESSEUR VINCENT BOMBARDIER

Contact :

Année académique :

2023/2024

Table des matières

Introduction	2
1 Modélisation des Cahier des charge	3
1.1 Description du cahier des charges :	3
1.2 Modèle simplifié :	3
1.3 Modèle SART :	4
1.4 Diagrammes :	5
1.4.1 Fonctionnement au niveau de la carte ESP32 :	5
1.4.2 Fonctionnement au niveau de la carte ESP32 OLED :	6
2 Réalisation et Résultats	7
2.1 Partie Hardware :	7
2.1.1 Materiel Utilisés	7
2.1.2 Schéma et câblages des composants	8
2.2 Partie Logicielle :	9
2.2.1 ESP32	9
2.2.2 ESP32-OLED	10
2.2.2.1 Définition des tâches	10
2.2.2.2 Définition des sémaphores	10
2.2.2.3 Choix des temps d'attente pour les tâches	11
2.2.2.4 Emploi de l'interruption	11
2.3 Mode d'emploi et Résultats	11
2.3.1 Mode d'emploi	11
2.3.2 Résultat final :	12
Conclusion	14

Table des figures

1.1	Modèle simplifié	3
1.2	Diagramme du Contexte du Projet	4
1.3	DFD0 et DFC0 : Gestion des mesures distance et CO2 par la carte ESP32	4
1.4	DFD1 et DFC1 : Gestion de modèle Producteur-Consommateur par la carte ESP32 OLED	5
1.5	Diagramme de Sequence du fonctionnement sur carte ESP32	5
1.6	Diagramme de Sequence du fonctionnement sur carte ESP32 OLED	6
2.1	Schéma et câblages des composants	8
2.2	Aucune personne est détectée	12
2.3	Affichage des mesures transmises par les différents capteurs	13
2.4	Serveur WEB local	13

Introduction

Les systèmes embarqués sont présents partout dans le monde et jouent un rôle important dans l'acquisition et le traitement des données. Ces systèmes sont basés sur des cartes ayant des fréquences différentes, et selon notre besoin, nous choisissons la carte adéquate qui répond aux exigences du cahier des charges. De plus, l'emploi des systèmes temps réel sur ces cartes est devenu de plus en plus intégrable grâce aux systèmes temps réel adaptés aux cartes. C'est le cas dans ce projet où nous avons employé l'ESP32 avec le système temps réel FreeRTOS afin d'acquérir des valeurs grâce aux capteurs et d'ordonnancer des tâches qui respectent des délais et des échéances bien déterminés.

1 | Modélisation des Cahier des charge

1.1 Description du cahier des charges :

Le cahier des charges impose la lecture des valeurs des capteurs de température, d'humidité et de CO₂, et de les afficher en présence d'une personne détectée à l'aide d'un capteur à ultrasons. Afin d'atteindre ces objectifs, on recourt à plusieurs techniques dans les cartes embarquées, comme les différents périphériques UART, SPI ou I2C, et les techniques d'ordonnancement des tâches en temps réel.

Dans notre cas, nous avons utilisé plusieurs techniques, et cela inclut :

- L'utilisation de deux cartes ESP32 nécessite l'établissement d'une communication série entre les deux cartes.
- La création d'un système consommateur-producteur est obligatoire pour :
 - La lecture des données des capteurs dans des tâches productrices et la publication des données dans des buffers.
 - La lecture régulière des valeurs depuis les buffers par des tâches consommatrices.

1.2 Modèle simplifié :

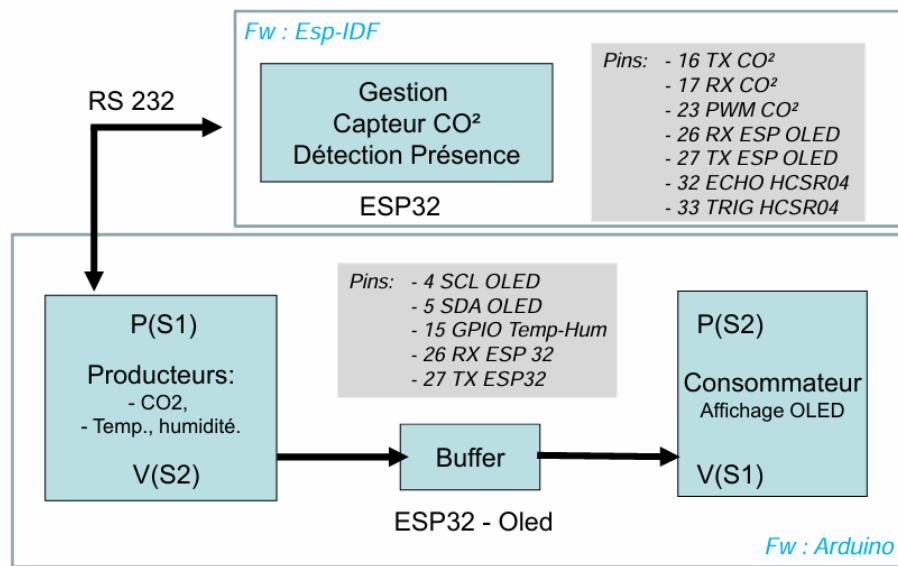


FIGURE 1.1 – Modèle simplifié

Le schéma ci-dessus présente l'architecture du projet, cela inclut l'utilisation des deux cartes ESP32 qui communiquent à l'aide du périphérique *RS 232*.

Pour le premier bloc, on a utilisé l'ESP32 avec le framework *ESP-IDF*. Cette carte assure :

- La lecture du *capteur CO₂*
- La détection de présence avec le *capteur ultrason HC-SR04*

Pour le deuxième bloc, on a utilisé l'ESP32 avec le framework *Arduino*. Cette carte assure :

- La lecture de la *température* et de l'*humidité*
- L'emploi d'un modèle *producteur-consommateur* à l'intermédiaire d'un *buffer*

Il est à noter que l'usage des *tâches* est obligatoire et qu'il est non envisageable d'utiliser la *boucle* pour lire les valeurs issues des capteurs, sinon on ne respecte plus les *échéances temporelles* et le principe d'un système temps réel n'est plus respecté.

1.3 Modèle SART :

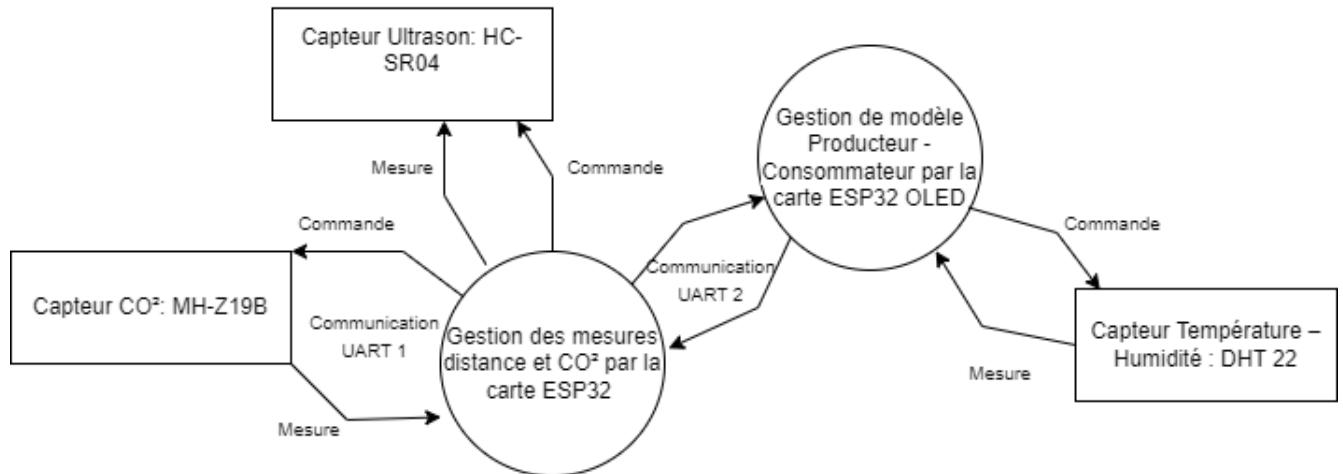


FIGURE 1.2 – Diagramme du Contexte du Projet

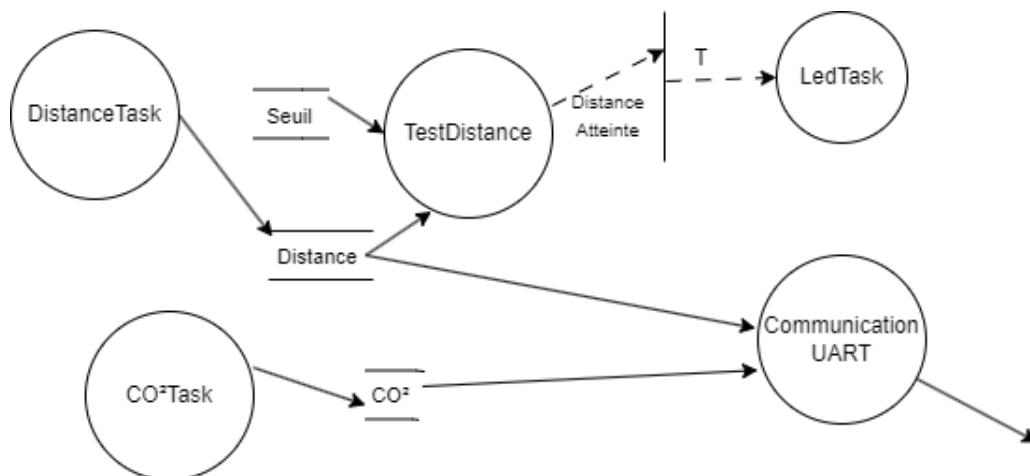


FIGURE 1.3 – DFD0 et DFC0 : Gestion des mesures distance et CO₂ par la carte ESP32

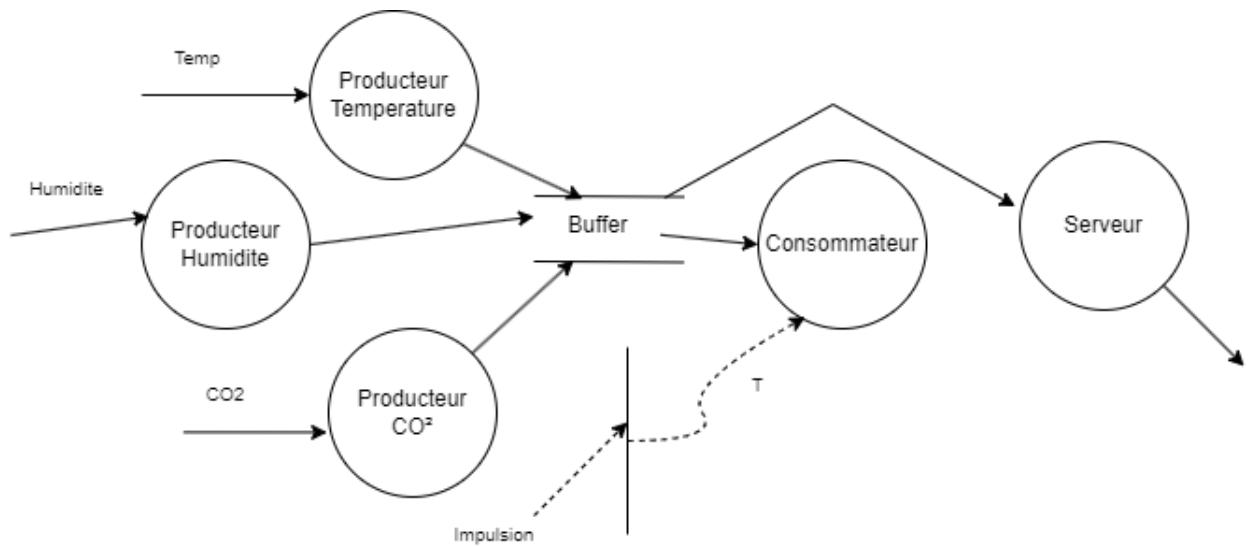


FIGURE 1.4 – DFD1 et DFC1 : Gestion de modèle Producteur-Consommateur par la carte ESP32 OLED

1.4 Diagrammes :

1.4.1 Fonctionnement au niveau de la carte ESP32 :

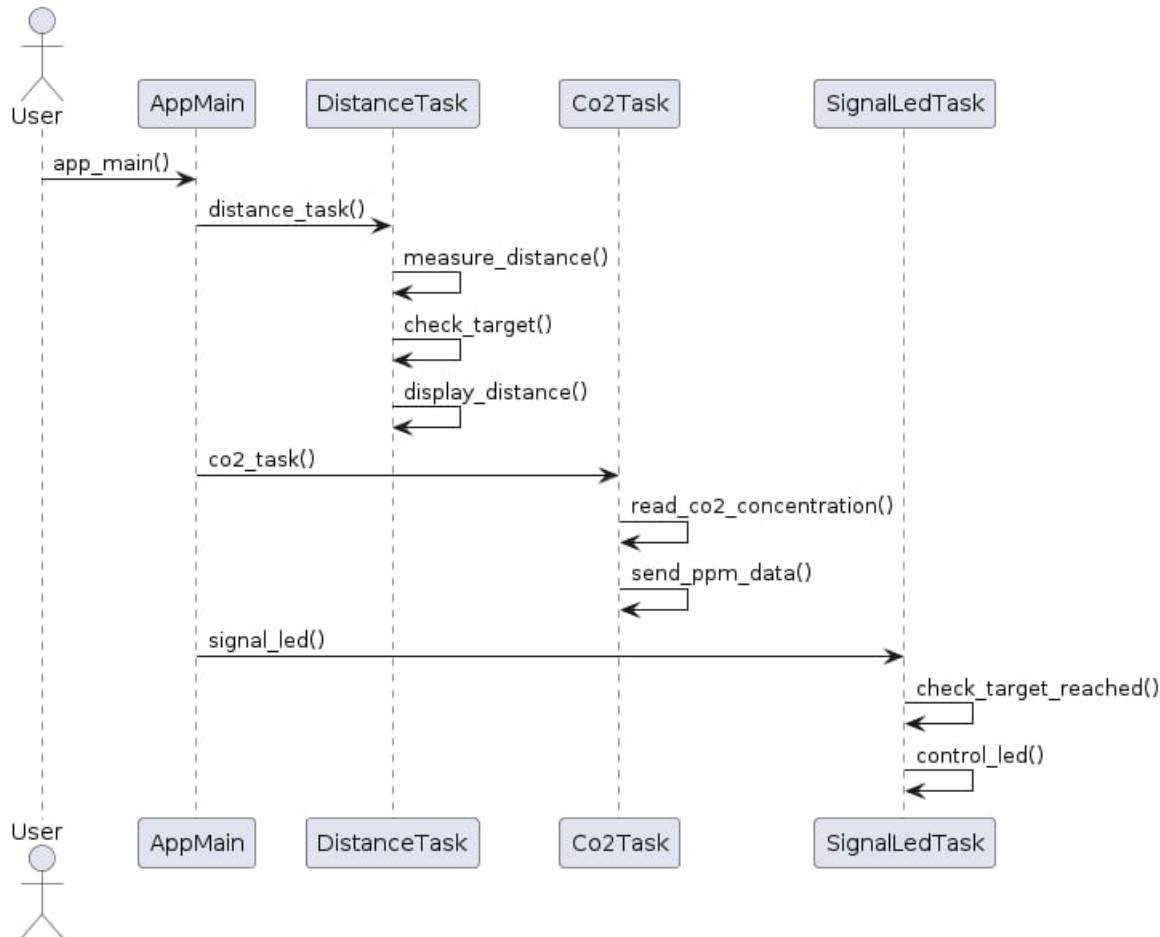


FIGURE 1.5 – Diagramme de Séquence du fonctionnement sur carte ESP32

1.4.2 Fonctionnement au niveau de la carte ESP32 OLED :

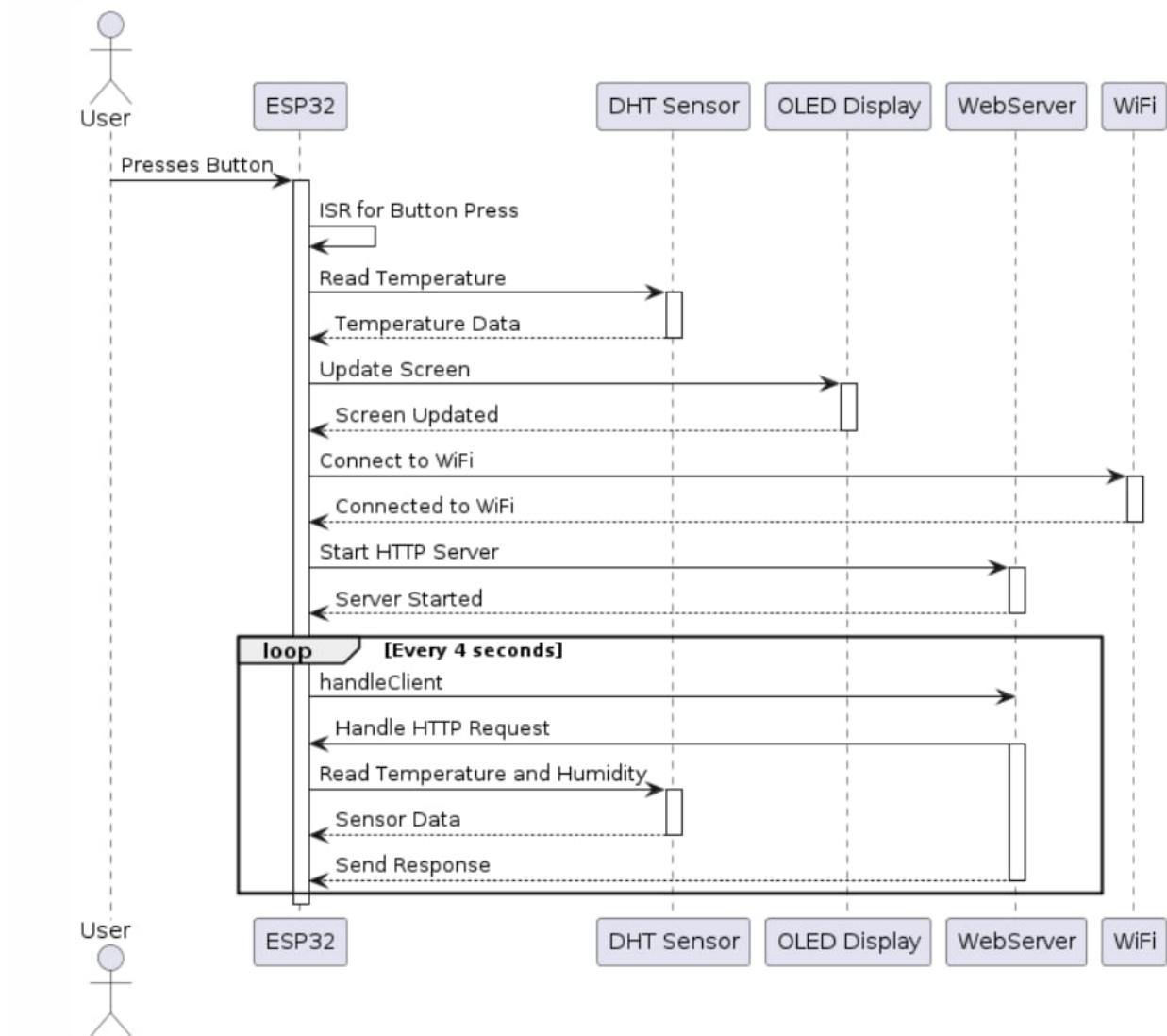


FIGURE 1.6 – Diagramme de Séquence du fonctionnement sur carte ESP32 OLED

2 | Réalisation et Résultats

2.1 Partie Hardware :

2.1.1 Matériel Utilisés

On a utilisé plusieurs capteurs et deux cartes ESP32. Leurs caractéristiques sont situées ci-dessous :

TABLE 2.1 – Caractéristiques des composants

Composant	Caractéristiques	Image
Capteur CO2 (MH-Z19B)	Mesure du dioxyde de carbone Plage de mesure : 0-5000 ppm Acquisition : UART ou bien mesure de PWM	
Capteur ultrason HC-SR04	Mesure de distance Plage de mesure : 2 cm - 400 cm Tension de fonctionnement : 5V Interface : GPIO	
Capteur DHT22	Mesure de température et d'humidité Alimentation : 3,3 à 6 Vcc Plage de mesure : - température : -40 à +80 °C - humidité : 0 à 100	
ESP32	Microcontrôleur ayant une Fréquence du processeur : jusqu'à 240 MHz. Mémoire flash : jusqu'à 16 MB Interfaces : UART, SPI, I2C, GPIO, Wi-Fi, Bluetooth.	
ESP32-OLED	Microcontrôleur ayant une Fréquence du processeur : jusqu'à 240 MHz. Mémoire flash : jusqu'à 16 MB Interfaces : UART, SPI, I2C, GPIO, Wi-Fi, Bluetooth Ecran Oled : Résolution 128 x 64. .	

2.1.2 Schéma et câblages des composants

On se base sur la figure 1.1 du modèle simplifié pour câbler le circuit, en respectant ainsi les broches spécifiées dans le schéma.

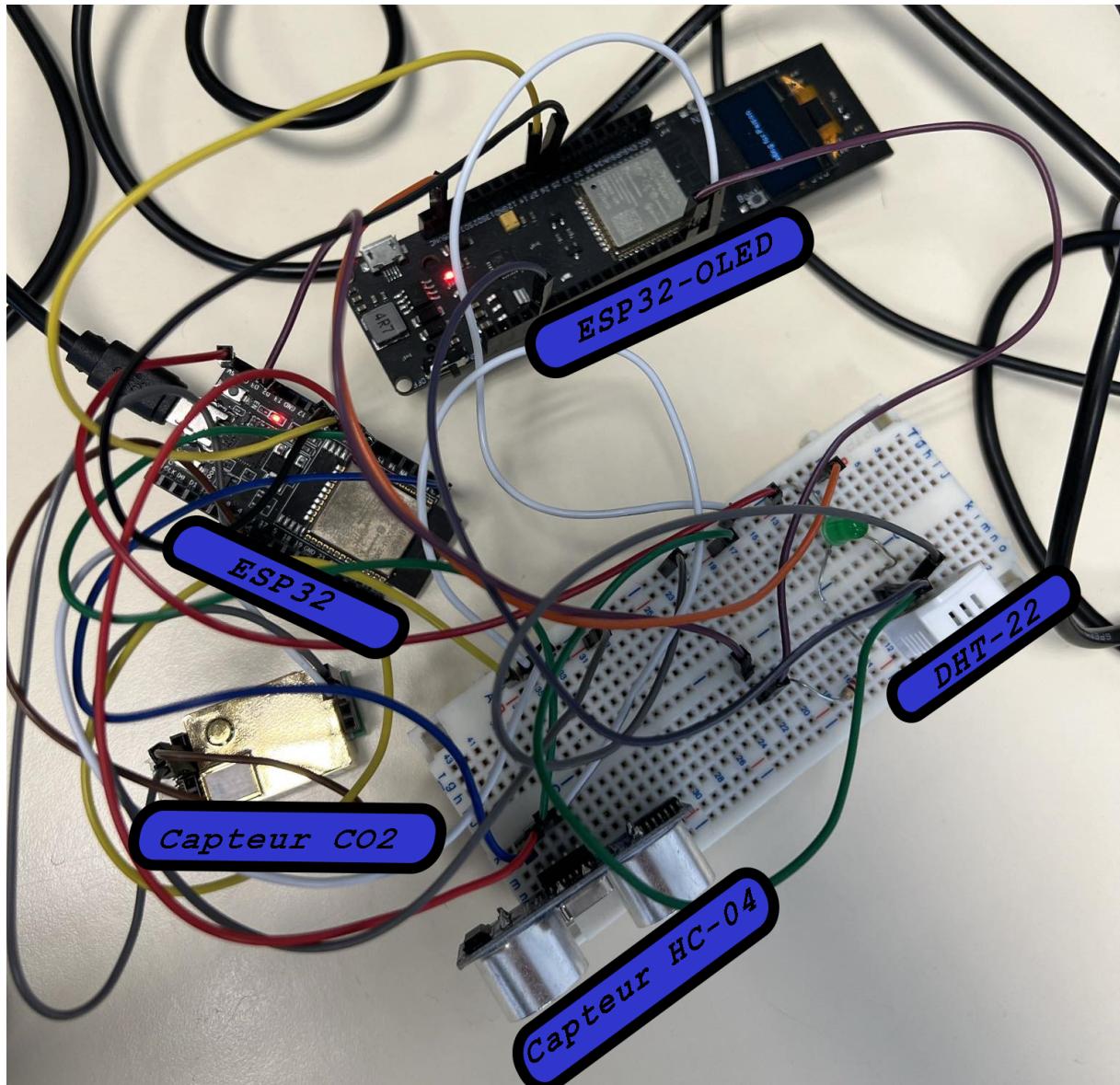


FIGURE 2.1 – Schéma et câblages des composants

Pour alimenter les deux cartes, nous utilisons le câble USB pour l'une des cartes, tandis que nous relions le 5V et la masse au breadboard pour alimenter la seconde carte. Il est important de noter que les masses doivent être communes afin d'assurer un même niveau de potentiel pour les deux cartes et les capteurs.

2.2 Partie Logicielle :

2.2.1 ESP32

Nom de la fonction	Signature	Explication
distance_task	void distance_task(void *pvParameter)	Mesure la distance à l'aide d'un capteur ultrasonique et affiche la valeur mesurée.
signal_led	void signal_led(void *pvParameter)	Allume une LED lorsque la distance cible est atteinte.
init_uart1	void init_uart1()	Initialise l'UART1 avec les paramètres de configuration définis.
init_uart2	void init_uart2()	Initialise l'UART2 avec les paramètres de configuration définis.
read_co2_concentration	void read_co2_concentration()	Lit la concentration de CO2 depuis un capteur via UART et affiche la valeur lue.
co2_task	void co2_task(void *pvParameters)	Lit la concentration de CO2 périodiquement et envoie les données via UART2.
app_main	void app_main()	Fonction principale qui configure les GPIO, initialise les UART et crée les tâches FreeRTOS.

```
init_uart1();
init_uart2();
xTaskCreate(distance_task, "distance_task", 2048, NULL, 5, NULL);
xTaskCreate(co2_task, "co2_task", 4096, NULL, 5, NULL);
xTaskCreate(signal_led, "led_task", 2048, NULL, 5, NULL);
```

Définition des périodes des tâches :

- **T_Distance_Task = 1 s**

La mesure de la distance est souvent critique dans des systèmes de détection d'obstacles ou de positionnement, où une réponse rapide est nécessaire pour réagir aux changements d'environnement.

Le choix d'une période plus courte permet de suivre de manière plus précise et régulière les variations de distance, ce qui est crucial pour la fiabilité du système.

- **T_signal_led = 3 s** Une période de 3 s est suffisante pour assurer que la LED réponde de manière visible et efficace à la détection de la distance cible atteinte. Cela est assez rapide pour que l'utilisateur puisse voir rapidement la réaction du système.

En équilibrant la période de cette tâche entre celles de la mesure de distance et de CO2, on assure une bonne synchronisation du système sans surcharge.

- **T_CO2_task = 4 s** La concentration de CO2 dans l'air change généralement plus lentement que la position d'un objet. Par conséquent, une période plus longue est suffisante pour capturer ces variations sans perte significative d'information. Elle n'a pas besoin d'être aussi fréquente que la mesure de distance pour être utile et efficace.

De plus , une période plus longue réduit la charge sur le processeur et les ressources système, permettant ainsi d'allouer plus de temps CPU à d'autres tâches critiques.

2.2.2 ESP32-OLED

2.2.2.1 Définition des tâches

Avant de définir les tâches, on définit les fonctions intermédiaires utilisées.

- On définit la procédure `void updateScreen()` qui met à jour l'écran OLED avec les informations collectées.
- On définit également la procédure `void drawWaiting()` qui met l'écran en mode d'affichage en attente de personne.
- La procédure `void handleRoot()` et `connect2Wifi()` jouent le rôle de configuration pour se connecter au réseau Wi-Fi et de création ainsi que de mise à jour du serveur web. Ce serveur envoie les informations et les affiche dans une interface web.

Pour la création des tâches :

On a créé 3 tâches producteurs pour produire les valeurs du CO₂, de la température et de l'humidité : `vProducteurTemperature`, `vProducteurCo2` et `vProducteurHumidite`. Ces tâches publient dans un buffer `tabMesure` une mesure de type `mesure_t`, définie comme suit :

```
typedef struct {  
    float mesure;  
    char type_capteur;  
} mesure_t;
```

La tache `vConsomateur` est tache responsable de lire les donnees du buffer et d'identifier le type de donnees consomé.

La définition de ces tâches est la suivante :

```
xTaskCreatePinnedToCore( vProducteurTemperature, "ProducteurTemp", 10000, NULL, 1, NULL , 0 );  
xTaskCreatePinnedToCore( vProducteurHumidite, "ProducteurHumidite", 10000, NULL, 1, NULL , 0 );  
xTaskCreatePinnedToCore( vProducteurCo2, "ProducteurCo2", 10000, NULL, 1, NULL , 0 );  
xTaskCreatePinnedToCore( vConsomateur, "Consomateur", 10000, NULL, 1 ,NULL , 0 );
```

On indique que toutes les tâches sont sur le même cœur du processeur pour signifier qu'elles sont concurrentes.

2.2.2.2 Définition des sémaphores

On définit deux Sémaphore S1 et S2 et un mutex comme cela :

```
s1 = xSemaphoreCreateCounting( TAILLE_MAX, TAILLE_MAX );  
s2 = xSemaphoreCreateCounting( TAILLE_MAX, 0 );  
mutex = xSemaphoreCreateMutex();
```

Les sémaphores S1 et S2, ainsi que le mutex, sont utilisés pour coordonner la production et la consommation de mesures dans le système.

Mutex :

- Utilisé pour assurer un accès exclusif à la ressource partagé le pointeur de tableau.
- **Producteur** : Acquiert le mutex avant de mettre à jour le pointeur de tableau pour éviter les conflits d'accès avec les autres producteurs et le consommateur.
- **Consommateur** : Acquiert également le mutex avant d'accéder au pointeur de tableau pour éviter les conflits avec les producteurs.

Sémaphore S1 :

- Utilisé pour contrôler l'accès à la zone critique où les mesures sont stockées.
- **Producteur** : Acquiert S1 après avoir mis à jour le pointeur de tableau sous la protection du mutex et écrit une nouvelle mesure.
- **Consommateur** : Libère S1 après avoir consommé une mesure et libéré le mutex pour permettre aux producteurs d'ajouter de nouvelles mesures.

Sémaphore S2 :

- Utilisé pour indiquer qu'une nouvelle mesure est prête à être consommée.
- **Producteur** : Libère S2 après avoir écrit une nouvelle mesure pour informer le consommateur qu'une nouvelle mesure est disponible.
- **Consommateur** : Acquiert S2 avant de consommer une mesure, s'assurant ainsi qu'il y a effectivement une mesure disponible à consommer.

2.2.2.3 Choix des temps d'attente pour les tâches

L'approche du choix du délai entre chaque publication des mesures est basée sur plusieurs considérations.

```
vTaskDelayUntil(&xLastWakeTime, pdMS_TO_TICKS(500));
```

Dans ce cas, un délai de 250 ms est choisi pour la température et l'humidité, tandis qu'un délai de 500 ms est choisi pour le CO₂. Ces délais sont sélectionnés en tenant compte que le CO₂ varie moins lentement dans la nature que la température et l'humidité.

D'après le datasheet du composant DHT22, il est spécifié que la fréquence d'échantillonnage ne doit pas dépasser 0,5 Hz (soit une mesure toutes les 2 secondes). Cela nécessite une période d'échantillonnage supérieure ou égale à 2 secondes.

2.2.2.4 Emploi de l'interruption

Nous avons employé l'interruption sur niveau haut avec une broche sur la carte ESP32-OLED, associée à la fonction *interruptPersonne* qui déclenche une temporisation. En effet, elle enregistre l'instant où l'interruption se produit, afin de mettre l'écran en marche pendant une période de 5 secondes.

```
attachInterrupt(digitalPinToInterruption(INTERRUPT_PIN), interruptPersonne, RISING);
```

2.3 Mode d'emploi et Résultats

2.3.1 Mode d'emploi

Afin de réaliser une démonstration avec les cartes ESP32 et ESP32 OLED contenant déjà le code téléchargé , on suit les étapes suivantes :

- alimenter la carte ESP32 OLED associée au capteur de température et humidité .

- mettre à zéro la carte ESP32 (l'appui sur le bouton **EN** appelé également **RESET** permet de le redémarrage forcé de l'ESP32).
- attendre un peu (1 minute) pour que la lecture du capteur CO₂ se stabilise .
NB : on a réglé la plage des valeurs mesurées par le capteur à [0 .. 5000] à l'aide de la commande

```
uart_write_bytes(UART_NUM_1, "\xFF\x01\x99\x00\x00\x00\x13\x88\xCB", 9)
```

pour éviter la saturation suite à une augmentation brusque.

- régler le nom du réseau WiFi sur votre machine qui va servir comme un point d'accès sans fil mobile et mot de passe à votre choix

```
const char *ssid = "user_name";
const char *password = "12345678";
```

Il faut s'assurer que la bande fréquentielle égale à **2.4 MHZ**.

On note que l'affichage commence dès qu'une personne est détectée à une valeur Réglable **DISTANCE_TARGET = 30 cm**

En cas de téléchargement de nouveau du code , on appuie sur **REBOOT** des 2 cartes .

2.3.2 Résultat final :

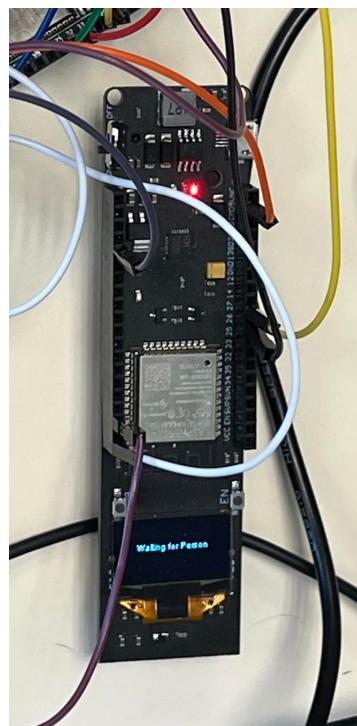


FIGURE 2.2 – Aucune personne est détectée

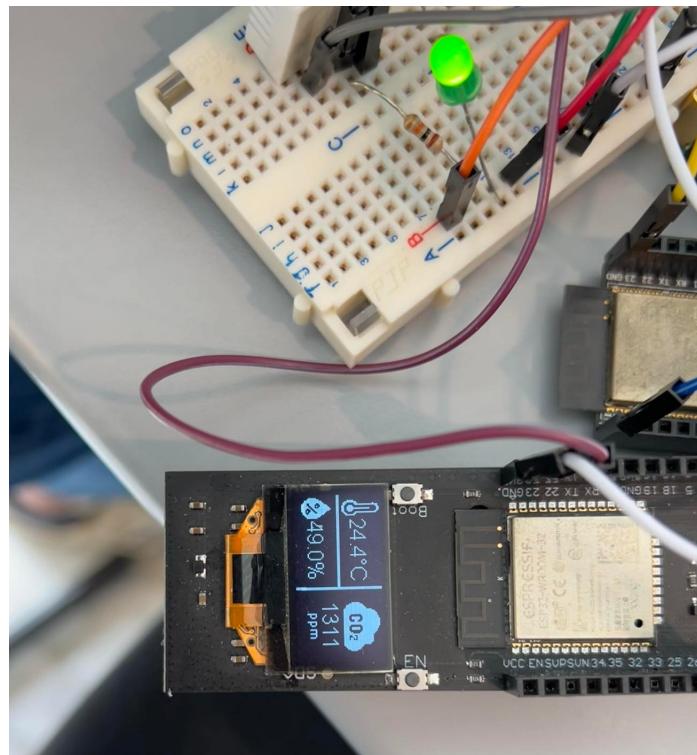


FIGURE 2.3 – Affichage des mesures transmises par les différents capteurs

NB : l'affichage est périodique et à chaque fois une personne est détectée , il se lance pendant 5s et se termine en cas d'absence d'une autre detection . On a ajouté aussi la possibilité d'éteindre l'affichage avec un bouton **Toggle Button** intégré dans une page web.

Intégration d'une page web : nous avons créé une tâche qui publie les données sur un serveur web toutes les secondes, comme suit :

Projet FreeRTOS ESP32 Data Monitoring



Toggle Button

FIGURE 2.4 – Serveur WEB local

Conclusion

Ce projet démontre une utilisation intelligente et synergique des cartes ESP32 et ESP32 OLED pour créer un système de surveillance environnementale efficace et évolutif. Grâce à l'intégration de divers capteurs et à une architecture bien pensée, ce système est capable de collecter, traiter et afficher des données cruciales telles que les niveaux de CO₂, la distance, la température et l'humidité assurant un affichage conditionnel et transmission filaire.

On a aussi maîtrisé l'utilisation des sémaphores dans l'implémentation du modèle Producteur - Consommateur et les mutex afin de protéger les sections critiques contenant les variables partagées