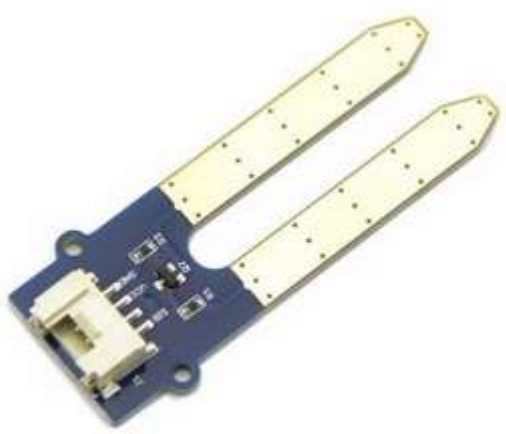
Projet plante connectée

Description :
ce projet vise à concevoir un système de surveillance et d'arrosage automatique pour les plantes. Il repose sur un capteur d'humidité du sol permettant de mesurer en temps réel le niveau d'humidité, un STM32 pour le traitement des données et une pompe contrôlée par PWM pour ajuster l'arrosage en fonction des besoins. Un module Bluetooth via un ESP32 assure la communication avec un smartphone, offrant ainsi une interface utilisateur intuitive pour surveiller et contrôler le système à distance. Cette solution intelligente optimise l'arrosage, réduit le gaspillage d'eau et garantit de meilleures conditions de croissance des plantes.

Moisture sensor :



Seeeduino	Grove-Moisture Sensor
5V	Red
GND	Black
Not Conencted	White
A0	Yellow

Ce capteur est utilisé pour détecter l'humidité du sol et voir s'il y a de l'eau autour , ce capteur est alimenté avec 5V et il a une sortie analogique entre 0V (min) et 5V (max)

Pour utiliser ce capteur nous avons besoin de l'ADC de notre microcontrôleur, donc nous avons activé l'ADC1 en single-ended vu que notre référence pour les mesures c'est 0V (GND) en suite nous avons configuré le pin PA0 comme input (ADC1_IN5).

Dans le main nous avons fait l'initialisation de l'ADC :
```
MX_ADC1_Init();
```

Et pour récupérer les valeurs il faut suivre ces étapes :
-Activer l'ADC .

```
HAL_ADC_Start(&hadc1);//activer l'ADC
```
-Attendre la fin de la conversion .
```
if (HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK)
```

-Lire la valeur de l'ADC
```
adc_value = HAL_ADC_GetValue(&hadc1);
```

-Calculer la valeur de la tension pour l'utiliser dans la vérification des résultats.
```
tension = (float) adc_value*5/4095 ;
```
-Calculer la valeur de l'humidité en pourcentage .
```
humidity_percentage = (float) adc_value / 4095 * 100;
```

-En fin transformer les valeurs de l'humidité. Et de la tension en caractères pour pouvoir les afficher sur l'écran LCD.

```
snprintf(buffer_humidite, sizeof(buffer_humidite), "Humidity:%.2f%%  \n\r",
humidity_percentage);
snprintf(buffer_tension, sizeof(buffer_tension), "Tension:%.2f V  \n\r",
tension);
```

La mesure de l'humidité est effectuée chaque seconde grâce à l'interruption du timer TIM2. Lorsque cette interruption se produit, une routine de service d'interruption (ISR) est exécutée, déclenchant ainsi la conversion analogique-numérique (ADC) pour récupérer les mesures selon les étapes décrites précédemment.

Configuration de l'interruption du timer TIM2

Pour régler le timer en mode interruption on suit les étapes suivantes :

- Activer le timer : dans Timers on choisit TIM2 et on active son horloge en choisissant Internal clock comme source d'horloge.
- Activer l'interruption : en cochant la case TIM2 global Interrupt dans NVIC Settings

Ces deux étapes permettent d'activer le timer et d'autoriser son interruption.

Pour défenir à quel moment cette interruption ce produit il faut choisir les réglages suivant dans Parameter Settings :

- Choisir la valeur de Prescaler : cette valeur permet de diviser la fréquence d'horloge qui alimente le timer, dans notre cas la source d'horloge du timer est Internal clock qui est à 80Mhz, en choisissant une valeur de 7999 la fréquence est divisée en 8000 ce qui fait que le compteur du timer incrémente de 1 chaque 0.1ms
- Choisir une valeur de Counter Period (ARR) : cette valeur définit le seuil à partir duquel le compteur déborde (overflow), lorsque le compteur atteint cette valeur une demande d'interruption est générer (IRQ Handler) cette IRQ est traiter par le NVIC qui fait appel à une ISR(un bout de code qui est exécuter à chaque nouvelle interruption) .A chaque fois cette valeur est atteint la remise à zero du compteur est fait automatiquement. Dans notre cas on veut une interruption chaque seconde, comme le compteur incrémente chaque 0.1ms ce qui veut dire qu'il faut 10000 incrémentations pour atteindre une seconde, donc la valeur de ARR est de 9999.

Une fois la configuration effectuée, il ne reste plus qu'à initialiser le timer en mode interruption en appelant la fonction **HAL_TIM_Base_Start_IT(&htim2)** dans le main, et d'écrire le code qui permet de récupérer les mesure de l'ADC dans l'ISR du timer **HAL_TIM_PeriodElapsed Callback**()

Lorsqu'une interruption se produit, elle suspend l'exécution du code principal (main) pour être traitée. Il est donc essentiel qu'elle soit aussi courte que possible afin d'éviter tout ralentissement du programme.Pour cela, un flag **(flag_finADC)** est utilisé dans l'ISR du timer. Ce dernier est activé dès que la conversion ADC est terminée et que la valeur de l'humidité a été récupérée. Ainsi, cette valeur peut être traitée dans la boucle principale (main) sans bloquer le processeur.

Il est également important de réinitialiser ce flag dans le main afin de permettre la réception de nouvelles conversions.

Control de la vitesse de la pompe :

La vitesse du moteur de la pompe est régulée à l'aide d'un signal PWM. En ajustant le rapport cyclique, il est possible de contrôler facilement la vitesse du moteur.

La variation du rapport cyclique modifie la tension de sortie sur la broche de la carte STM32 : rapport cyclique de 0 → la sortie est de 0V, rapport cyclique de 1 → la sortie est de 3.3V.

Pour augmenter cette tension et permettre un contrôle efficace du moteur, un GROVE MOSFET v1.1 est connecté à la broche de la carte. Son fonctionnement est le suivant :

- Lorsque la sortie de la STM32 est 0V, le MOSFET délivre 0V au moteur.
- Lorsque la sortie est 3.3V, le MOSFET fournit 5V au moteur.

Ainsi, en modulant le rapport cyclique du signal PWM, on ajuste la tension appliquée au moteur, permettant un contrôle précis de sa vitesse.

Pour générer un signal PWM, on utilise le timer TIM3 du STM32 avec la configuration suivante :

- Activer le timer TIM3 en choisissant Internal Clock comme source d'horloge (80Mhz)
- Choisir le canal sur lequel le signal PWM est générer dans notre cas on choisit le chanel 2
- Choisir le Pin (pin PB5), sur le sheild c'est D4
- Choisir la période du signal du PWM en modifiant la valeur du Prescaler et de Counter Period(ARR), en prenant les valeurs 7999 pour le prescaler et 99 pour ARR on obtient un signal PWM d'une période de 10ms.

Une fois la configuration effectuée, il ne reste plus qu'à initialiser le timer TIM3 en mode PWM en appelant la fonction **HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2).**

Pour modifier le rapport cyclique du signal PWM et ajuster la vitesse de la pompe, on utilise la fonction **pompe_vitesse().** Cette fonction prend en paramètre la valeur du PWM(entre 0 et 100, elle se repose sur la fonction **__HAL_TIM_SET_COMPARE()**, qui met à jour le registre correspondant(registre CRR)

Envoie/réception par Bluetooth

La carte STM32 ne disposant pas de connectivité Bluetooth, un ESP32 est donc utilisé comme passerelle entre la STM32 et le téléphone. La communication entre ces deux cartes est assurée via l'interface USART1, configurée en mode asynchrone avec un débit de transmission de 9600 bits/s. Cette configuration est réalisée dans l'outil CubeMX, qui génère automatiquement la fonction **MX_USART1_UART_Init()** pour initialiser l'USART. Afin d'assurer une communication fiable, il est impératif que les deux cartes soient paramétrées avec la même vitesse de transmission.

a) **Envoie des données vers le téléphone :** L'envoi des données vers le téléphone s'effectue toutes les 10 secondes grâce à la fonction **HAL_UART_Transmit().** Les valeurs mesurées, notamment l'humidité et la tension correspondante, sont converties en chaîne de caractères ASCII à l'aide de la fonction **sprintf()** et stockées dans **buffer_humidite** et **buffer_tension,** puis transmises via USART1 vers l'ESP32, qui se charge ensuite de les envoyer au téléphone via Bluetooth. Afin de garantir que l'ESP32 reçoive correctement les données, un timeout d'1 seconde est défini dans **HAL_UART_Transmit(),** permettant ainsi à la STM32 d'attendre la réponse de l'esp pour une seconde avant de mettre fin à la communication et retourner une erreur.

b) **Réception des données envoyées par le téléphone :** Comme mentionné précédemment, la vitesse du moteur de la pompe est contrôlée par un signal PWM, on a choisi que ce rapport

cyclique peut être ajuster à distance par le téléphone. Cette valeur est transmise par le téléphone sous forme de 3 octets, chaque octet représentant un caractère ASCII de la valeur du PWM. La réception de ces données est assurée par la fonction **Reception_PWM_Value()**, qui repose sur **HAL_UART_Receive().**

Lorsqu'un premier caractère est détecté, la fonction **HAL_UART_Receive()** entre en mode bloquant jusqu'à la réception complète du message de 3 octets, qui est ensuite stocké dans un buffer de réception **(buffer_reception).** Une fois les 3 octets reçus, la fonction retourne le statut **HAL_OK**, confirmant ainsi la fin de la réception.

Si aucun octet n'est reçu avant l'expiration du timeout de 50 ms, la fonction retourne une erreur timeout, et la valeur du PWM reste inchangée. En revanche, si la réception est réussie **(HAL_OK),** les données sont copiées dans le buffer PWM **(PWM_buffer),** et une temporisation de 50 ms est appliquée pour garantir leur intégrité. Par ailleurs, une LED connectée à la broche **PC0** est activée à chaque réception d'une nouvelle valeur PWM depuis le téléphone.

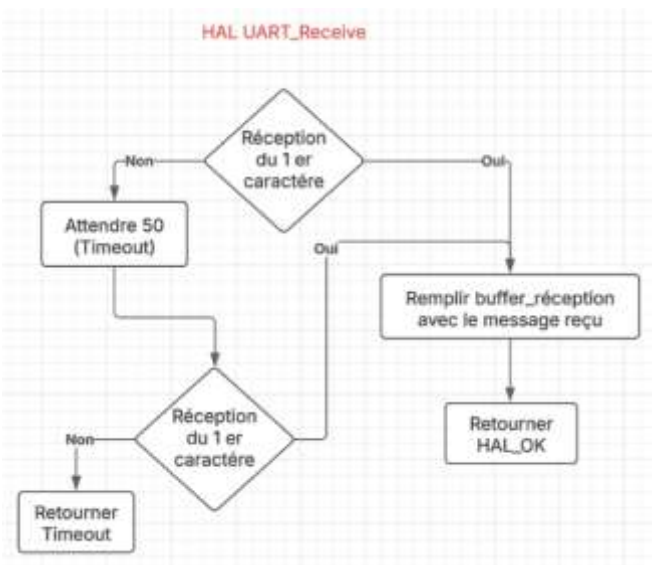


Figure.1

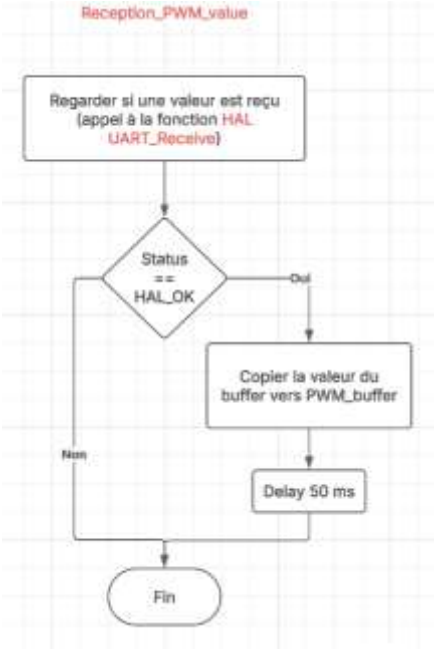


Figure.2

Les Figures 1 et 2 illustre le fonctionnement des fonctions **HAL_UART_Receive()** et **Reception_PWM_Value()** respectivement.

c) **Décodage du message reçu :** Une fois la valeur du rapport cyclique PWM reçue sous forme de caractères ASCII, elle doit être convertie en une valeur entière exploitable. Cette conversion est réalisée par la fonction **PWM_Value()**, qui extrait la valeur numérique du 9buffer PWM **(PWM_buffer).**
La fonction commence par convertir les trois caractères ASCII en un entier en appliquant les pondérations correspondantes aux centaines, dizaines et unités. Une fois la transformation effectuée, elle vérifie que la valeur obtenue est bien comprise entre 0 et 100. Si elle dépasse 100, la fonction retourne 100 afin de garantir que le rapport cyclique reste dans une plage valide. Sinon, elle renvoie directement la valeur convertie.
La valeur obtenue est ensuite utilisée dans la fonction **pompe_vitesse()** pour ajuster la vitesse du moteur de la pompe dans la boucle principale (main). Cette approche permet de s'assurer que la valeur reçue via Bluetooth est correctement interprétée et utilisée pour le contrôle du moteur.

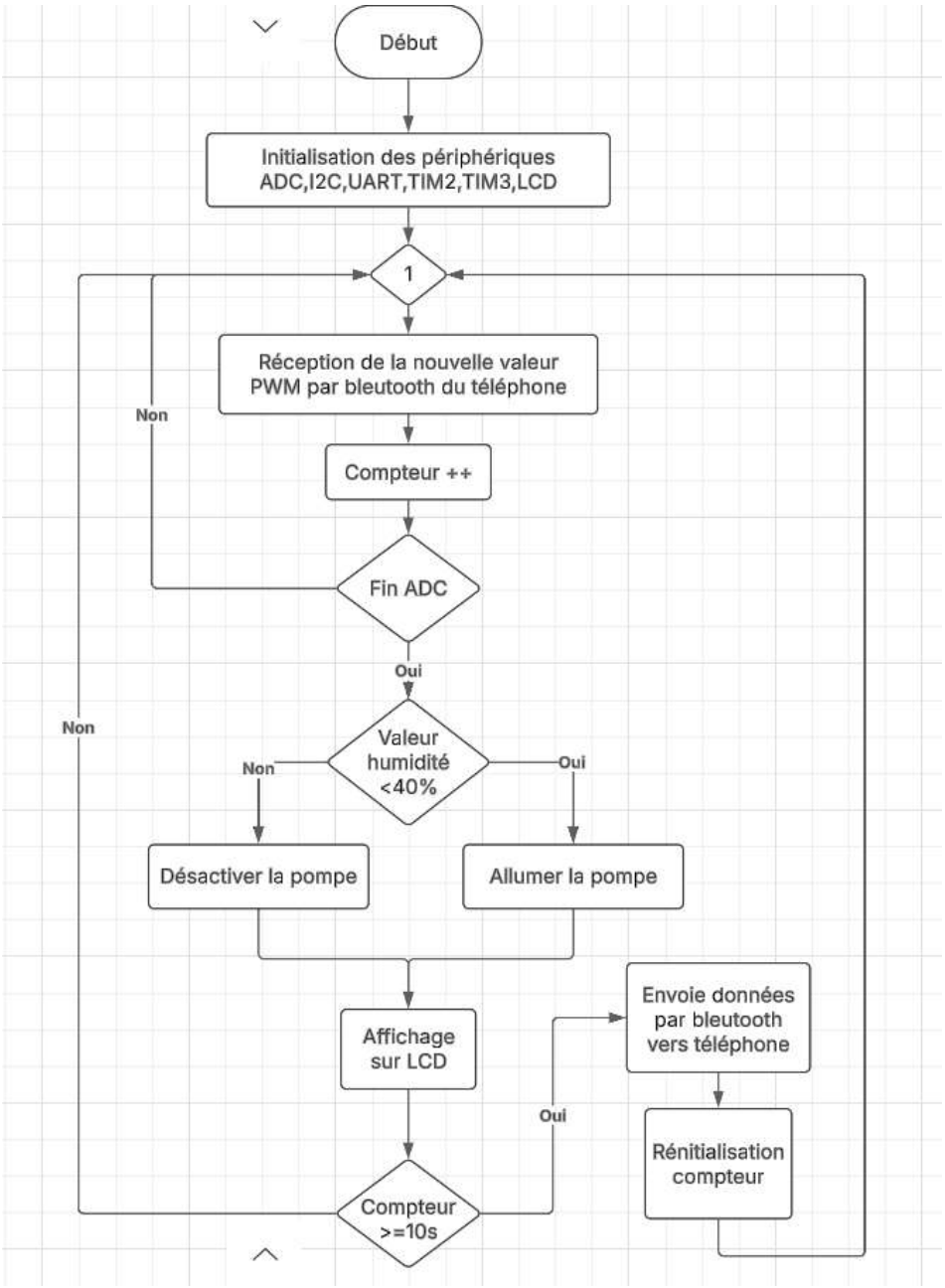


Figure.3 Algorigrame fonction main.

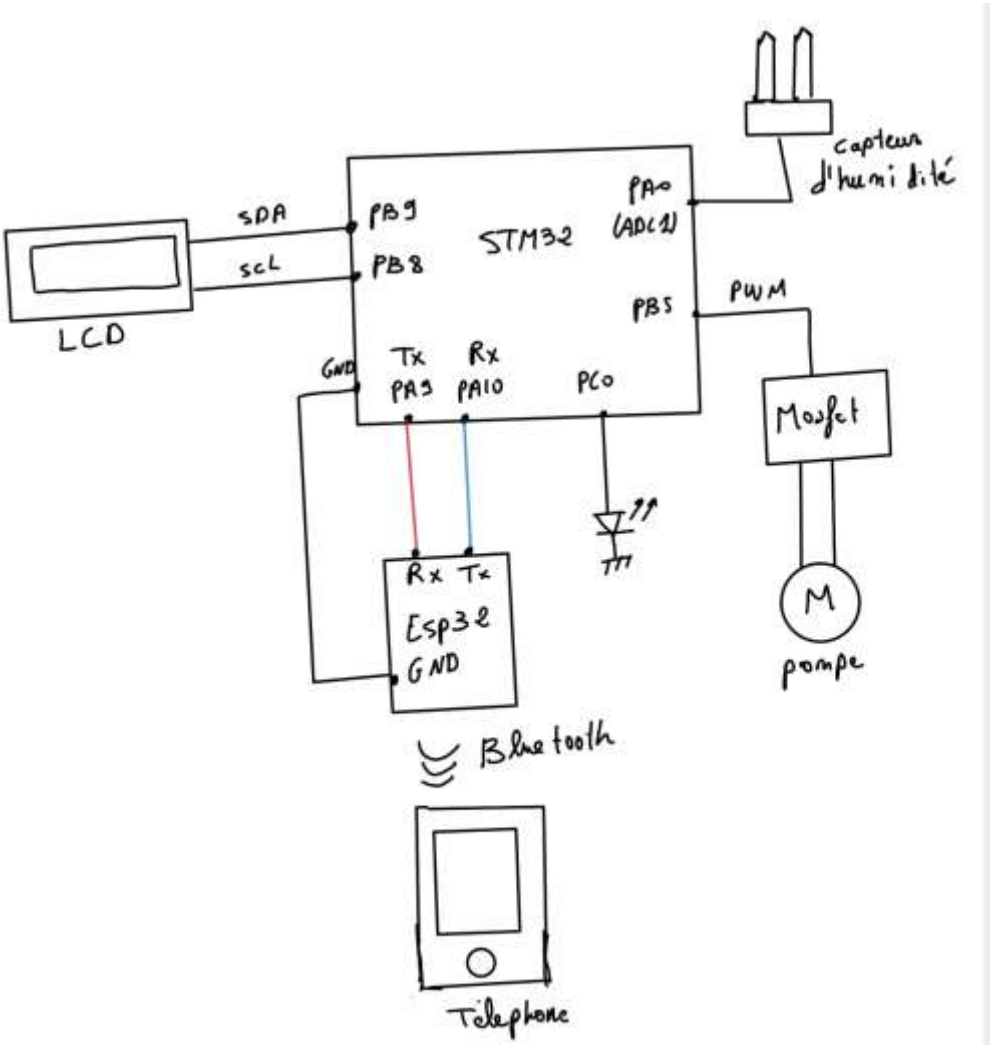


Figure.4 Schéma global du montage.