

STATION METEO



Lien github: https://github.com/Bil-Theo/SE_Project.git

1.Introduction.....	2
2.Spécifications externes de l'application.....	3
2.1.Présentation des capteurs.....	3
2.2.Principales fonctionnalités.....	5
2.3.Bords du modèle.....	6
2.4.Tables des Entrées/sorties.....	6
2.4.1. Tables des entrées.....	6
2.4.2.Tables des sorties.....	6
2.5.Table des interruptions avec gestion des priorités.....	7
3.Décomposition préliminaire de l'application.....	8
3.1.Flôt de données.....	8
3.2.Flôt d'événement.....	8
3.3.Machine d'états.....	9
3.3.1.Machine d'état de l'IHM.....	9
3.3.2.Machine d'état globale.....	9
4.Développement et validation unitaire.....	10
4.1.Capteur de direction du vent.....	10
4.2.Capteur de quantité de pluie.....	12
4.3.Capteur de vitesse du vent.....	14
4.4.Capteur de température et d'humidité.....	17
4.5.Capteur de pression.....	19
5.Développement et validation des éléments d'assemblage.....	20
5.1.Agrégation des mesures.....	20
5.1.1.Mesures de tous les capteurs.....	20
5.1.2.Enregistrement sur la carte SD.....	20
5.2.Gestion de l'IHM.....	22
5.2.1.Croquis des pages.....	22
5.2.2.Affichage réel des pages.....	23
6.Développement et validation de l'application.....	24
7.Tableau récapitulatif des fonctionnalités réalisées.....	24
8.Conclusion.....	25
9.Composition et responsabilités dans l'équipe projet.....	25
9.1.Composition de l'équipe.....	25
9.2.Table de description des tâches réalisées par personne de l'équipe projet.....	26
10.Annexe.....	27
11.Bibliographie.....	28

1.Introduction

Dans le cadre de ce projet, nous avons entrepris le développement d'une station météorologique autonome capable de mesurer et d'analyser diverses variables climatiques, telles que la température, l'humidité, la pression atmosphérique, la direction du vent et la vitesse du vent. L'objectif principal de ce projet est de concevoir une solution fiable et performante pour collecter des données météorologiques précises en temps réel, les stocker dans une carte SD, tout en permettant une gestion efficace et une visualisation claire de ces données via une interface homme-machine (IHM).

Le projet s'articule autour de plusieurs étapes clés, incluant l'analyse détaillée de la solution, le développement des fonctionnalités principales, ainsi que la validation unitaire et l'intégration des différents composants du système. La première phase de l'analyse est centrée sur la définition des spécifications de l'application, détaillant les entrées et sorties à gérer ainsi que les principales fonctions à réaliser.

Une attention particulière a été portée à la décomposition préliminaire de l'application, avec une identification précise des flux de données et des événements, ainsi que la gestion des différents états du système via une machine d'état. Chaque sous-partie du projet a été développée de manière modulaire, permettant une validation unitaire avant l'intégration finale de l'ensemble du système.

L'interface utilisateur et l'agrégation des mesures ont été des éléments essentiels dans le développement, assurant une expérience fluide pour l'utilisateur tout en garantissant l'exactitude et la fiabilité des données présentées. Ce rapport présente ainsi une analyse complète de notre solution, de sa conception à son intégration, avec un suivi précis de chaque fonctionnalité réalisée et validée.

Ce document présente l'analyse technique détaillée de notre projet et décrit les différentes étapes de son développement, allant de la définition des spécifications aux tests de validation, en passant par l'intégration des composants. Enfin, un tableau récapitulatif des fonctionnalités réalisées et des responsabilités au sein de l'équipe projet est fourni pour donner un aperçu complet de l'avancement du projet.

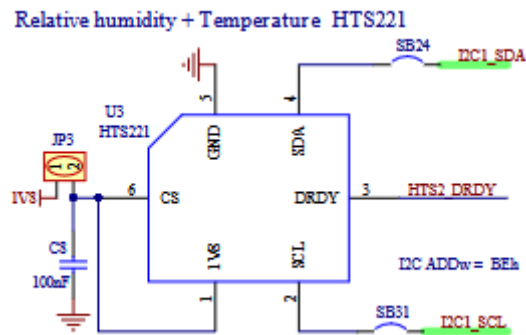
2.Spécifications externes de l'application

Le projet combine les capteurs de la carte NucleoX-NUCLEO-IKS01A3 avec ceux de la station "Weather Meters" de SparkFun. Ces derniers sont utilisés pour mesurer la vitesse et la direction du vent ainsi que la quantité de pluie, tandis que ceux de la nucléo permettront de mesurer la température, la pression ainsi que le taux d'humidité, offrant ainsi une solution

complète pour la collecte de données météorologiques précises et actualisées. Les données acquises seront ensuite affichées sur l'écran du kit STM32F746g-DISCO à l'aide de plusieurs interfaces utilisateur.

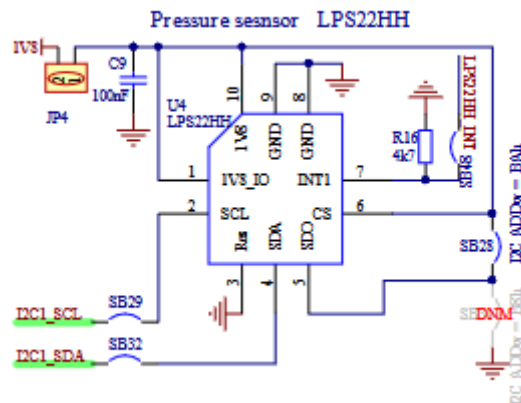
2.1.Présentation des capteurs

1. Capteurs de la carte Nucléo IKS03A5
 - a. Capteur de température et d'humidité (HTS221)



Ce capteur permet de mesurer des valeurs d'humidité de 0 à 100% ainsi que des valeurs de température comprises entre -40°C et +120°C. Il communique ces données via une liaison I2C. Les spécifications de ce capteur sont donc parfaitement appropriées à l'utilisation que nous voulons en faire.

- b. Capteur de pression (LPS22HH)



Ce capteur permet de mesurer des valeurs de pression comprises entre 260hPa et 1260hPa et communique via une interface I2. Il est parfaitement adapté à notre usage puisque à l'altitude où nous allons utiliser la station, nous nous situons autour des 1013hPa.

- c. Capteur de vitesse de vent



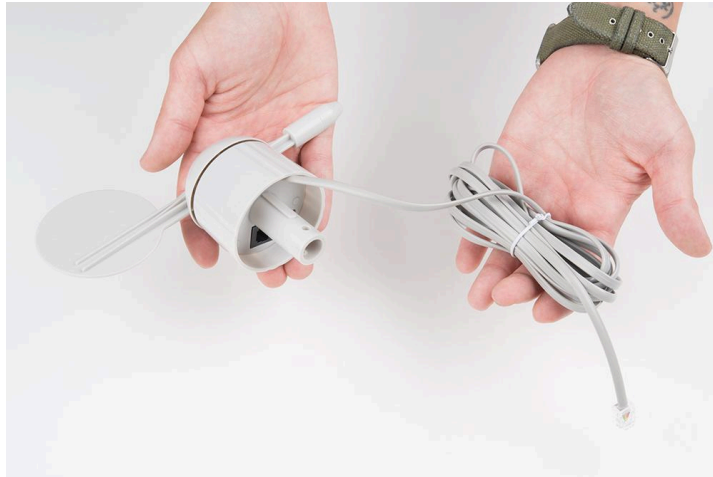
Le capteur de vitesse de vent de la station "Weather meters" est composé d'une hélice avec des demi-sphères, lorsque celle-ci tourne et effectue un tour, un aimant se ferme dans le capteur. C'est ce signal généré que nous allons scruter. On note qu'une impulsion correspond à un tour et la datasheet nous décrit qu'un tour par seconde correspond à une vitesse de vent de 2.4Km/h.

d. Capteur de pluie



Ce capteur permet de mesurer la quantité de pluie. La quantité d'eau tombée fait basculer un bucket et active un switch à chaque bascule, c'est donc l'impulsion due à ce switch que nous allons scruter. La datasheet nous informe que chaque fois que le bucket bascule, cela correspond à une quantité de pluie tombée de 0.2794mm.

e. Capteur de direction du vent



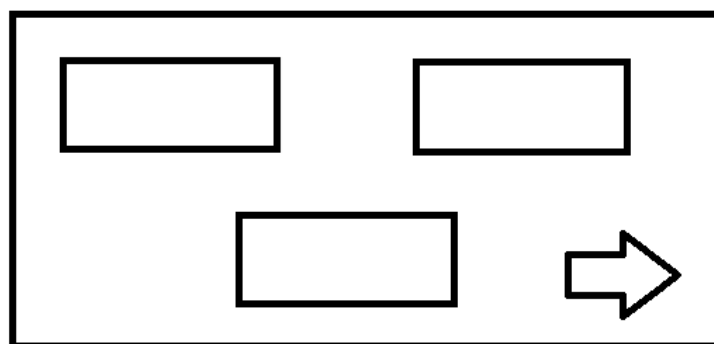
Ce capteur va nous permettre de déterminer la direction du vent, il fonctionne comme un pont diviseur de tension, c'est à dire qu'en fonction de l'angle de la girouette, un interrupteur relié à une résistance va se fermer, et donc un pont diviseur va se créer avec la résistance interne du capteur et une résistance externe. Nous allons donc pouvoir mesurer la tension à la sortie de ce pont et relier cette tension à une direction grâce au tableau de valeur fournie par la datasheet en Annexe 1.

2.2.Principales fonctionnalités

Notre système final tel que nous l'avons imaginé devra répondre à plusieurs fonctionnalités qui sont les suivantes:

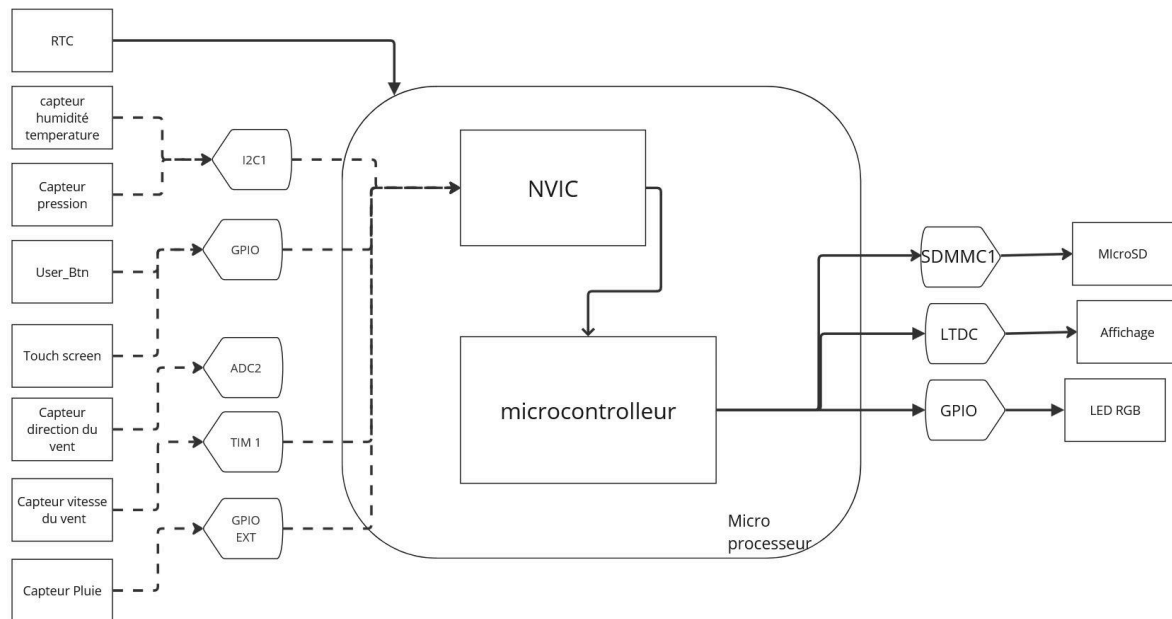
- Permettre de collecter les données de températures, pression, humidité, quantité de pluie, vitesse du vent ainsi que direction du vent.
- Permettre l'affichage des données collectées sur l'écran de votre carte.
- Permettre de sauvegarder les données acquises dans un fichier texte ou tableur sur une carte microSD.
- Permettre à l'utilisateur de choisir la fréquence d'acquisition des données.
- Permettre au système entier une faible consommation d'énergie.

Pour répondre à ces paramètres, nous avons imaginé une interface utilisateur comme suit:



Les cadres représentent l'endroit où seront affichées les différentes valeurs des capteurs et la flèche permettra de passer d'une page à l'autre.

2.3.Bords du modèle



2.4.Tables des Entrées/sorties

La table des entrées sorties permet de visualiser quels sont les pins utilisés.

2.4.1. Tables des entrées

Entrées	Pin correspondants sur STM32f746
"User_Btn" pour allumer l'écran après un mode veille.	PI11
"GPIO" emplacement de carte microSD.	PC13
"I2C1" pour les capteurs de température, pression et humidité.	PB8 et PB9
"TIMER 1" en input capture pour le capteur de vitesse du vent.	PA8
"GPIO" en external interrupt pour le capteur de quantité de pluie.	PA15
"ADC2" pour le capteur d'orientation du vent.	PA0

2.4.2.Tables des sorties

Sorties	Pin correspondants sur STM32f746
---------	----------------------------------

“SDMMC1” pour le stockage sur la carte microSD.	PC8,PC12 et PD2
“GPIO_RGB” pour la led de contrôle.	PH9,PH10 et PH11
“LTDC” pour l'affichage LCD sur l'écran.	PA11,PA12,PD3,PG6,PG10,PG11,PH13,PH15,PI0,PI4,PI5,PI9,PI10,PI14,PJ2,PJ5,PK1,PK5, PK6 et PK7
“RGB” pour faire clignoter nos led qui précise l'état actuel de l'Application	PH9, PH10, PH11

2.5.Table des interruptions avec gestion des priorités

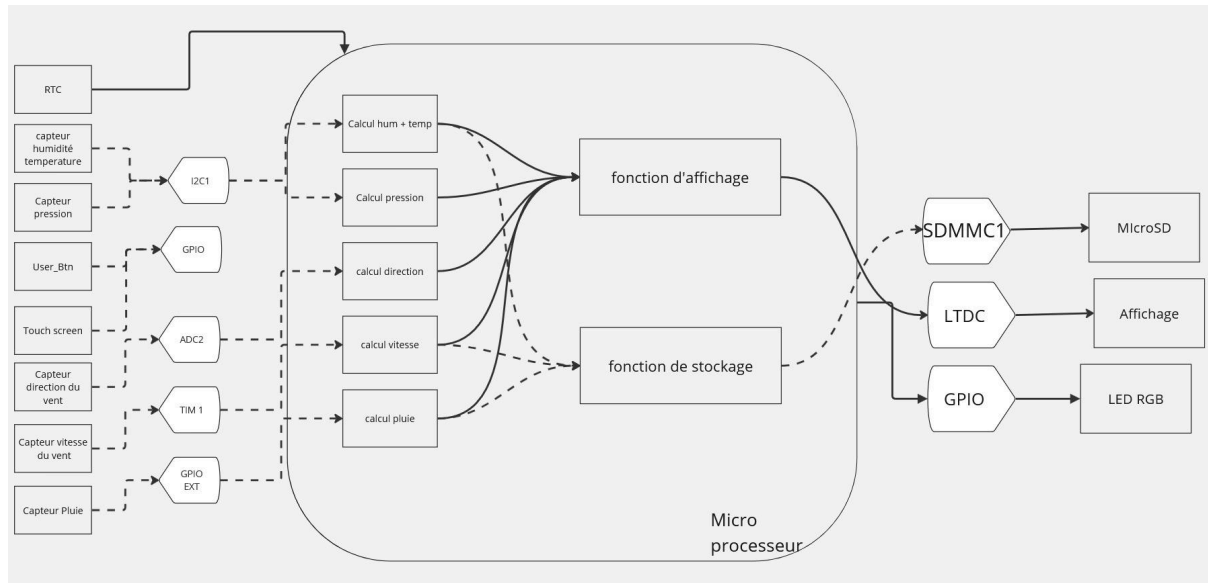
Interruptions	Priorités de 0 à 5 pour la plus faible.
Enregistrement sur carte microSD	0
Gestion de l'affichage	1
Capteur de pluie	2
Capteur de vitesse et direction du vent	3
IRQ TIM2	2

Configuration			
NVIC		Code generation	
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
ADC1, ADC2 and ADC3 global interrupts	<input checked="" type="checkbox"/>	3	0
EXTI line[9:5] interrupts	<input type="checkbox"/>	0	0
TIM1 break interrupt and TIM9 global interrupt	<input type="checkbox"/>	0	0
TIM1 update interrupt and TIM10 global interrupt	<input type="checkbox"/>	0	0
TIM1 trigger and commutation interrupts and TIM11 global interrupt	<input type="checkbox"/>	0	0
TIM1 capture compare interrupt	<input checked="" type="checkbox"/>	3	0
I2C1 event interrupt	<input type="checkbox"/>	0	0
I2C1 error interrupt	<input type="checkbox"/>	0	0
USART1 global interrupt	<input type="checkbox"/>	0	0
EXTI line[15:10] interrupts	<input type="checkbox"/>	2	0
FMC global interrupt	<input type="checkbox"/>	0	0
SDMMC1 global interrupt	<input checked="" type="checkbox"/>	0	0
TIM5 global interrupt	<input checked="" type="checkbox"/>	1	0

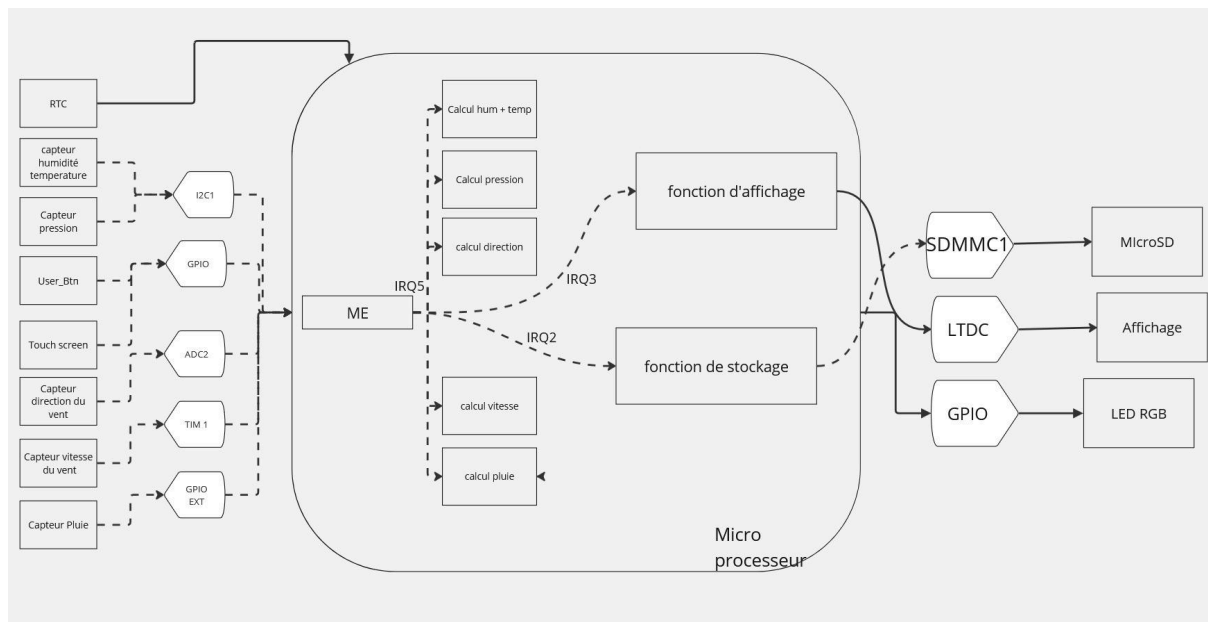
3. Décomposition préliminaire de l'application

Afin de décomposer notre application et faciliter la mise en place logicielle de celle-ci, nous avons commencé par tracer les différents diagrammes caractérisant le fonctionnement. Les diagrammes de flots de données et d'événements permettent de comprendre les interactions entre les composants du système et les machines d'états expliquant comment le système fonctionne dans sa globalité.

3.1. Flot de données

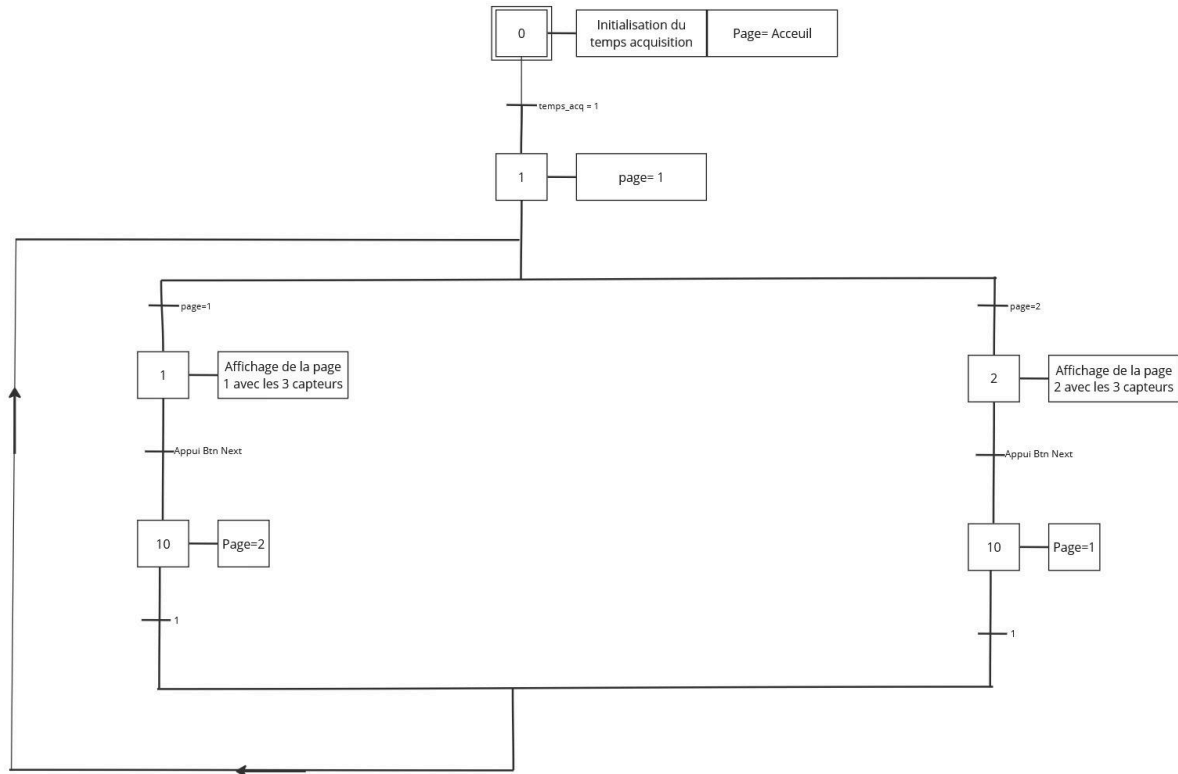


3.2. Flot d'événement

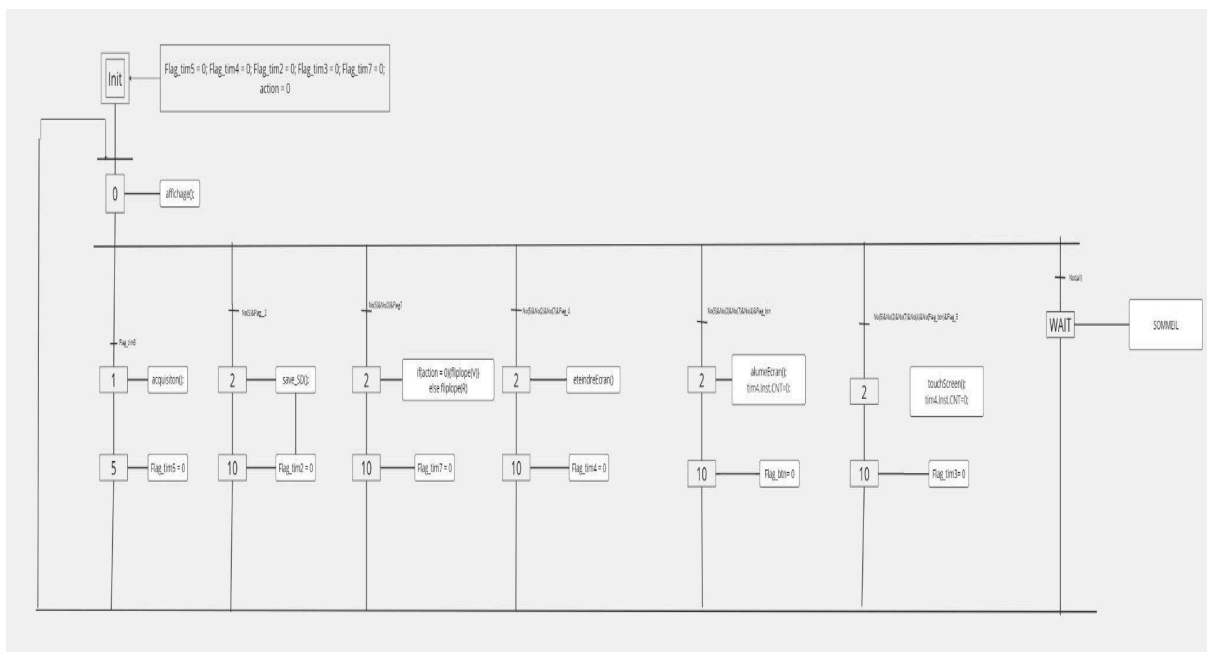


3.3.Machine d'états

3.3.1.Machine d'état de l'IHM



3.3.2.Machine d'état globale



4. Développement et validation unitaire

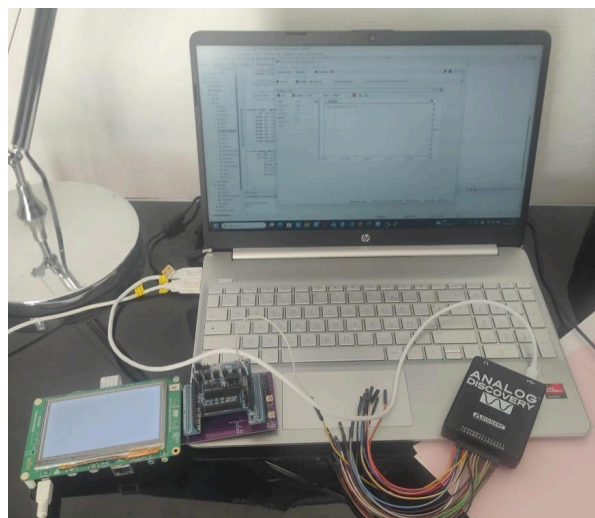
4.1. Capteur de direction du vent

Afin de récupérer la valeur de la direction du vent nous avons utilisé l'analog to digital converter (ADC) de notre STM32. Le principe est de venir lire une valeur de tension à la sortie du pont diviseur de tension formé par le capteur et la résistance externe et de l'associer à une direction à l'aide de l'annexe 1. Ici notre ADC est sur 16 bits il peut donc renvoyer au maximum une valeur de 4095 qui correspondra à 3.3V. Après avoir lu la valeur à l'aide de l'ADC on la convertit en une valeur de tension puis on la compare aux différentes directions pour renvoyer le bon résultat.

```
const char* getWindDirection(float voltage) {
    if ((voltage >= 0.7) & (voltage <= 0.9)) return "OUEST";
    else if ((voltage > 0.9) & (voltage <= 1.35)) return "NORD-OUEST";
    else if ((voltage > 1.35) & (voltage <= 1.55)) return "NORD";
    else if ((voltage > 2.2) & (voltage <= 2.4)) return "NORD-EST";
    else if ((voltage > 3.03) & (voltage <= 3.23)) return "EST";
    else if ((voltage > 2.86) & (voltage <= 3.02)) return "SUD-EST";
    else if ((voltage > 2.66) & (voltage <= 2.86)) return "SUD";
    else if ((voltage > 1.8) & (voltage <= 2.0)) return "SUD-OUEST";
    else return "Direction Inconnue";
}

void Read_ADC2_Channel1() {
    HAL_ADC_Start(&hadc2);
    if (HAL_ADC_PollForConversion(&hadc2, 1000) == HAL_OK) {
        uint32_t adcValue = HAL_ADC_GetValue(&hadc2);
        float voltage = (adcValue / 4095.0) * 3.3; // Conversion en volts
        HAL_ADC_Stop(&hadc2);
        direction = getWindDirection(voltage);
        printf("Tension: %.2fV, Direction: %s", voltage);
    }
}
```

Ce code nous permet donc d'effectuer les tâches cités précédemment, nous allons donc faire un test avec l'analog discovery pour voir si cela fonctionne bien:



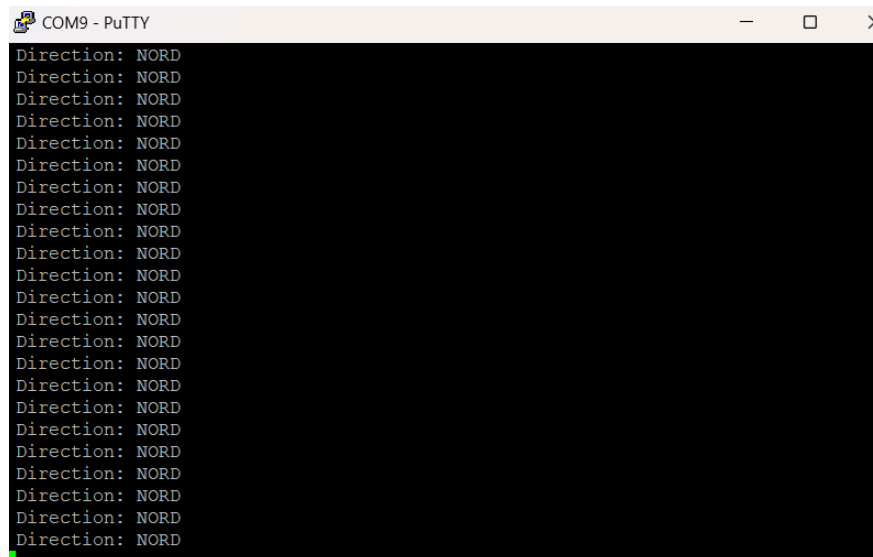
Une fois le branchement fait, on envoie une tension de 1.8V sur le pin correspondant à l'ADC de notre carte et on observe la console:

```
Tension: 1.88V, Direction: SUD-OUEST  
Tension: 1.11V, Direction: NORD-OUEST  
Tension: 1.10V, Direction: NORD-OUEST  
Tension: 1.12V, Direction: NORD-OUEST  
Tension: 1.09V, Direction: NORD-OUEST  
Tension: 0.77V, Direction: OUEST  
Tension: 0.78V, Direction: OUEST  
Tension: 0.81V, Direction: OUEST  
Tension: 3.11V, Direction: EST  
Tension: 3.12V, Direction: EST
```

La console affiche bien les bonnes directions en fonction de la tension lue, on peut donc valider partiellement le code et faire un test avec le vrai capteur.

On place donc, ensuite le capteur en direction du nord pour effectuer une simulation réelle avec le vrai capteur:





```
COM9 - PuTTY
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
Direction: NORD
```

L'affichage sur notre console nous permet de valider le fait que le capteur est totalement fonctionnel nous pouvons donc passer à un autre capteur.

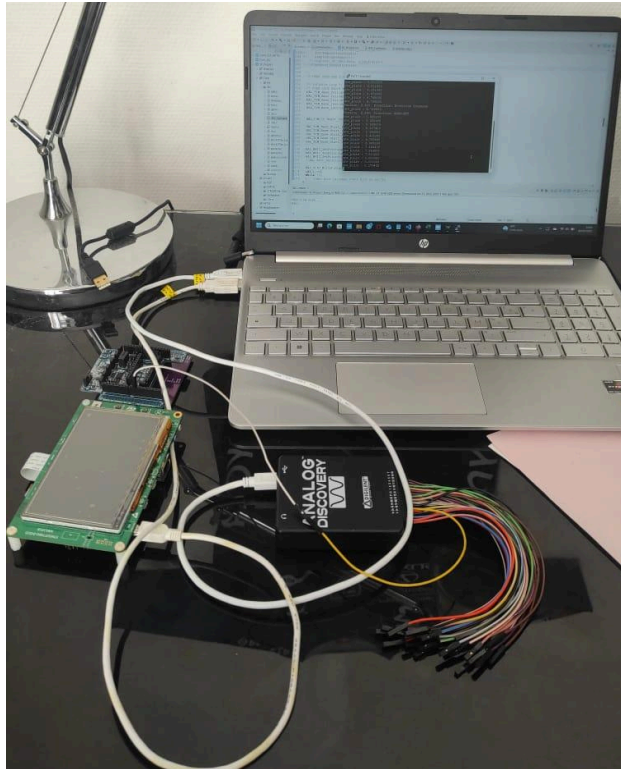
4.2. Capteur de quantité de pluie

Afin de récupérer les valeurs du capteur de pluie, nous avons configuré le pin PA15 en "external interrupt" puis codé une fonction qui incrémente un compteur d'interruption. En effet à chaque fois que le bucket du capteur est plein, il bascule et génère une interruption, chaque interruption correspond à 0.2794mm de pluie d'après la datasheet, nous multiplions donc le nombre d'interruptions par cette valeur pour connaître la quantité de pluie tombée.

```
void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin )
{
    if (GPIO_Pin==GPIO_PIN_15)
    {
        static uint32_t derniertick=0;
        float tick_actuel = HAL_GetTick();
        if ( (tick_actuel-derniertick)>1000){ // si il y a au moins 1 s de delai entre les deux inetrruptions (evite le rebond)
            interrupt_count++; // Incrémenter le compteur d'interruptions
            derniertick=tick_actuel;
        }
    }
}

void detect_pluie()
{
    qte_pluie = interrupt_count * (0.2794); // Calcul de la pluie en mm
    printf("qte_pluie : %f\r\n",qte_pluie);
}
```

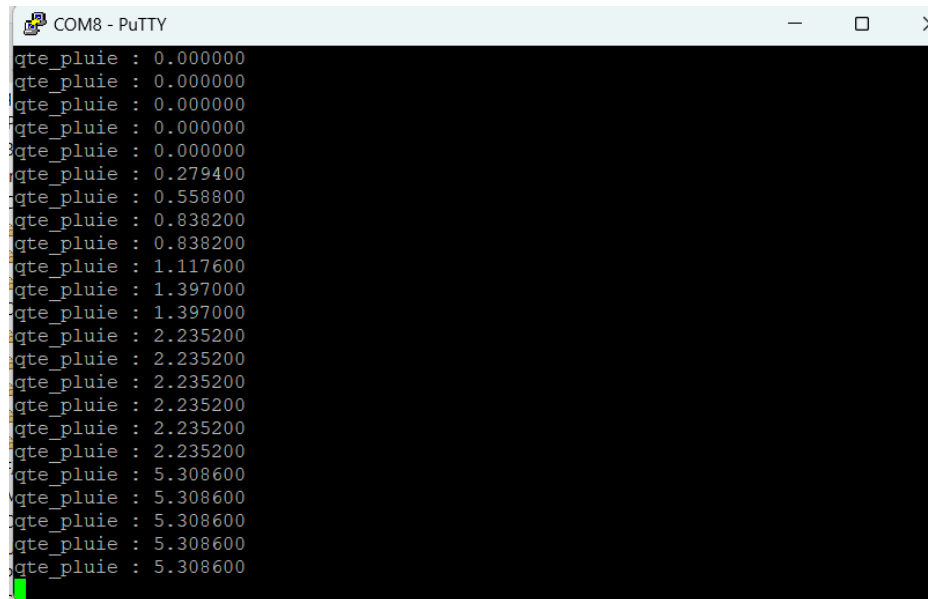
Après avoir effectué le code, nous testons avec notre analog discovery en générant des interruptions sur le pin approprié:



On regarde le résultat de cette manipulation dans la console et on observe que le code à l'air de fonctionner, on remarque que l'on compte bien des valeurs multiples de 0.2794.

```
COM3 - PuTTY
qte_pluie : 4.749800
qte_pluie : 5.029200
qte_pluie : 5.308600
qte_pluie : 0.000000
qte_pluie : 0.279400
qte_pluie : 0.558800
qte_pluie : 0.838200
qte_pluie : 1.117600
qte_pluie : 1.397000
qte_pluie : 1.676400
qte_pluie : 1.955800
qte_pluie : 2.235200
```

On branche maintenant directement notre capteur physique sur notre carte et on affiche les données toutes les secondes:



```
COM8 - PuTTY
qte_pluie : 0.000000
qte_pluie : 0.000000
qte_pluie : 0.000000
qte_pluie : 0.000000
qte_pluie : 0.000000
qte_pluie : 0.279400
qte_pluie : 0.558800
qte_pluie : 0.838200
qte_pluie : 0.838200
qte_pluie : 1.117600
qte_pluie : 1.397000
qte_pluie : 1.397000
qte_pluie : 2.235200
qte_pluie : 2.235200
qte_pluie : 2.235200
qte_pluie : 2.235200
qte_pluie : 2.235200
qte_pluie : 2.235200
qte_pluie : 5.308600
qte_pluie : 5.308600
qte_pluie : 5.308600
qte_pluie : 5.308600
qte_pluie : 5.308600
```

On remarque que le capteur fonctionne très bien, les valeurs qui se répètent correspondent aux moments où on ne fait pas basculer le système, cela valide le fait qu'il n'y a pas d'interruptions lorsque le système ne bascule pas. On considère alors que le capteur et son code sont totalement validés.

4.3. Capteur de vitesse du vent

Afin de récupérer les données du capteur de vitesse du vent nous avons configuré le Timer1 en mode "input capture" avec une résistance pull up. Nous avons codé un programme qui incrémente une variable à chaque fois qu'on a un front montant sur le Channel 1 du Timer1. Cette variable est ensuite divisée par le temps d'acquisition puis multipliée par 2.4 puisque la datasheet nous donne que pour 1tour/s cela correspond à 2.4km/h.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM1 && htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
        // Incrémente le compteur de ticks à chaque capture
        tick_count++;
    }
}

void Get_Wind_Speed() {
    // Récupération des ticks capturés
    float ticks_per_second = tick_count;

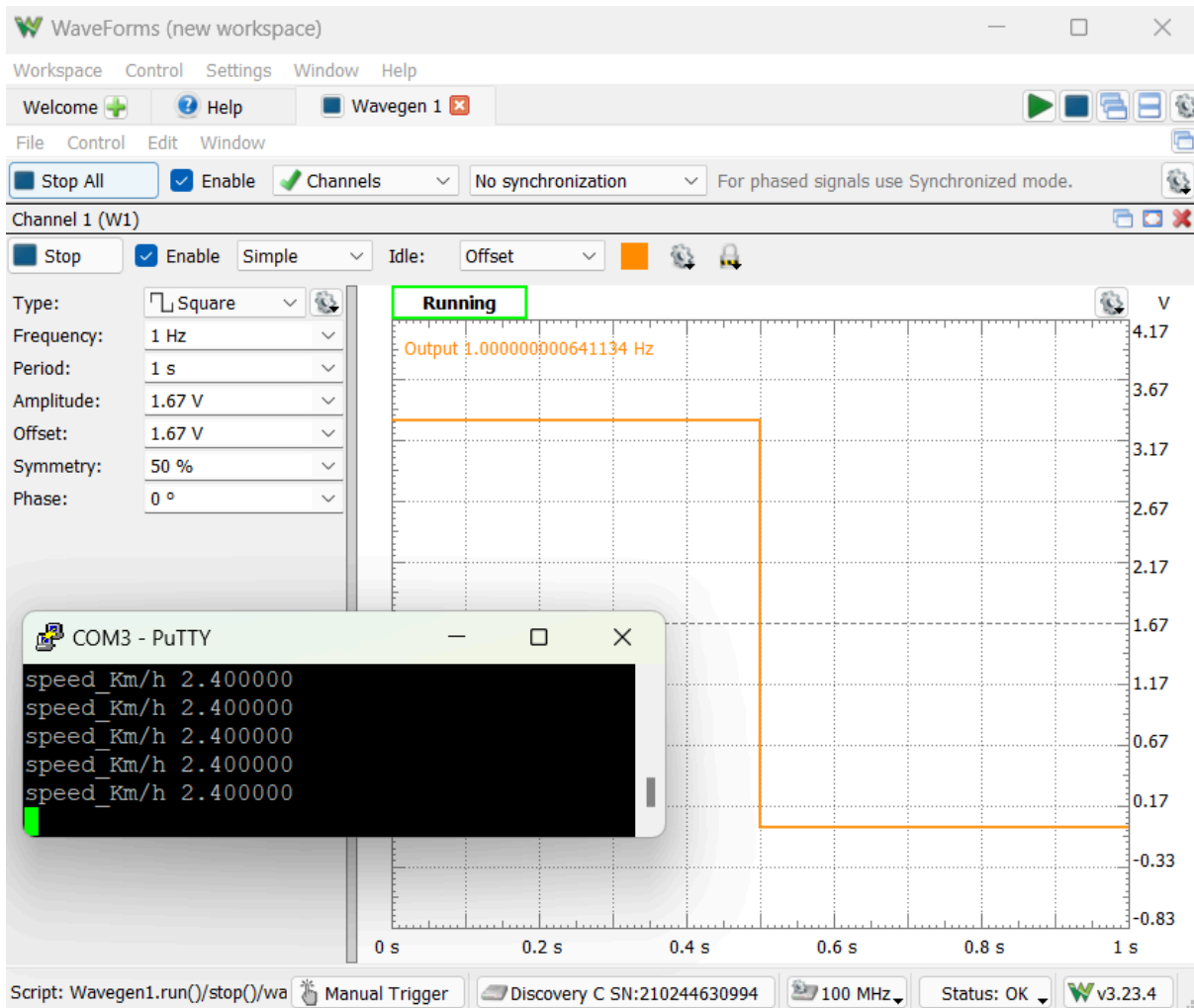
    // Réinitialisation du compteur pour la prochaine mesure
    tick_count = 0;

    // Conversion des ticks/s en vitesse (km/h)
    speed_kmh = (ticks_per_second/temp_acq) * 2.4;
    printf("speed_Km/h %f\r\n", speed_kmh);
}
```

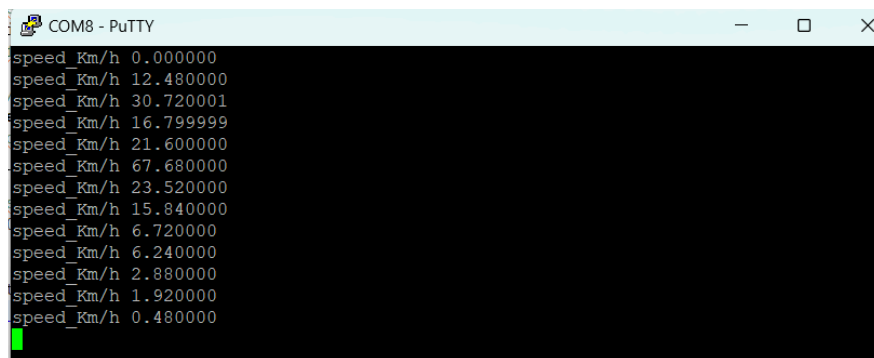
Nous utilisons notre analog discovery pour simuler le signal produit par le capteur de vitesse du vent puis on branche sur notre pin PA8. Pour le test nous avons utilisé un signal avec une période d'une seconde afin générer 5 front montant dans 5 secondes soit $(5/5)*2.4=2.4\text{km/h}$ on devrait voir une vitesse de 2.4 km/h d'afficher dans la console.



On remarque que la console affiche une valeur de 2.4km/h ce qui confirme que le compteur de front montant s'incrémente bien et valide donc notre programmation.



On branche maintenant notre capteur physique sur notre carte, et on vérifie l'affichage de notre vitesse, ici on affiche toute les 5 secondes la valeur de la vitesse moyenné sur 5 secondes on remarque que la vitesse augmente lorsqu'on fait tourner le système à la main puis descend lorsque que l'hélice deselere. Cela valide complètement la configuration de notre capteur.



4.4. Capteur de température et d'humidité

Afin de collecter les données du capteur de température et d'humidité, nous avons configuré le bus I2C1 sur les pins PB8 et PB9 de notre microcontrôleur. Nous avons ensuite intégré les drivers nécessaires au fonctionnement des capteurs. Puis codé un programme pour venir lire cette valeur, ce programme étant basé sur celui utilisé dans la mode d'utilisation des drivers.

```
void get_grandeur_values_sensor_hts221() {
    hts221_reg_t reg;
    hts221_status_get(&dev_ctx, &reg.status_reg);

    if (reg.status_reg.h_da) {
        /* Read humidity data */
        memset(&data_raw_humidity, 0x00, sizeof(int16_t));
        hts221_humidity_raw_get(&dev_ctx, &data_raw_humidity);
        humidity_perc = linear_interpolation(&lin_hum, data_raw_humidity);

        if (humidity_perc < 0) {
            humidity_perc = 0;
        }

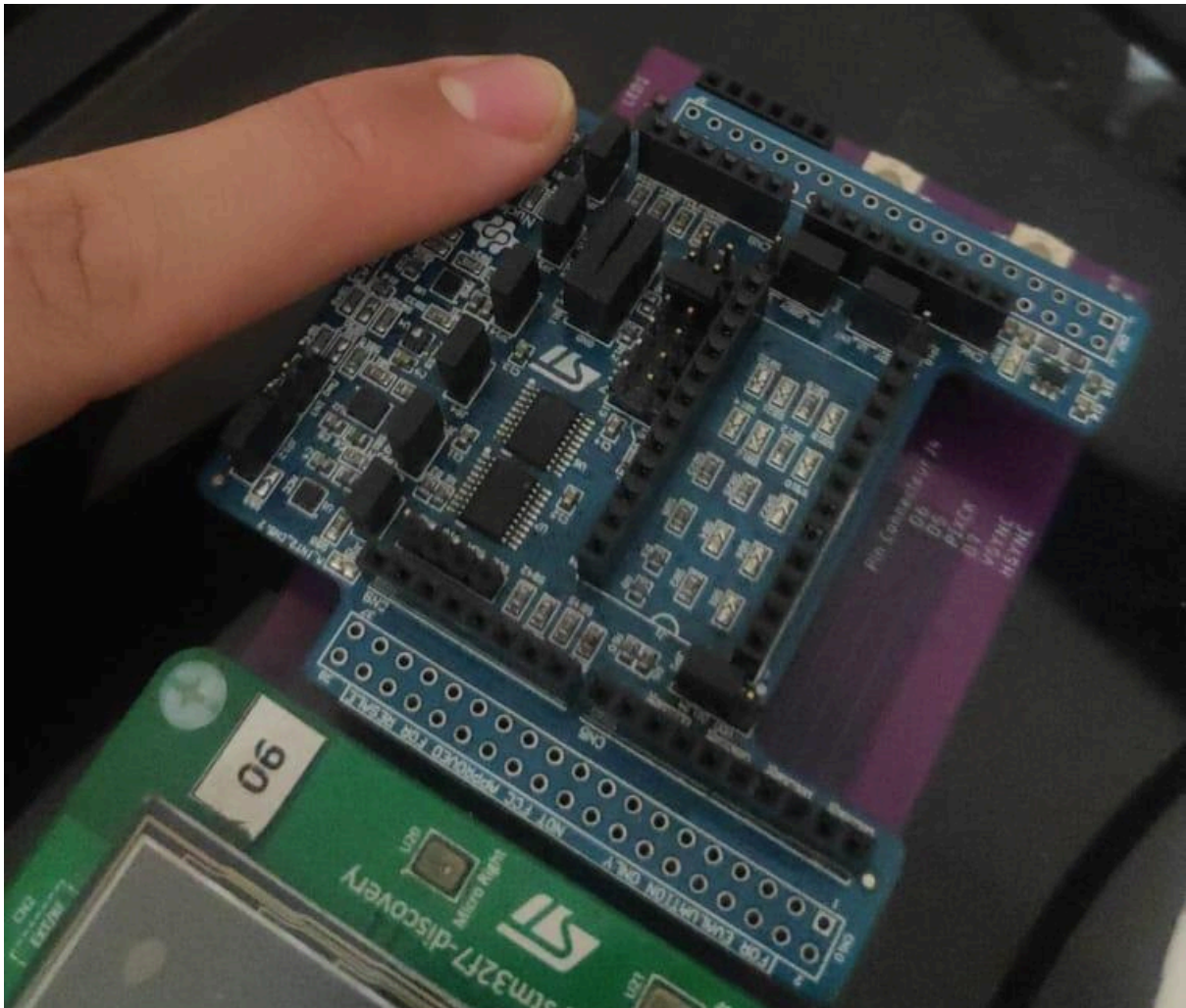
        if (humidity_perc > 100) {
            humidity_perc = 100;
        }

        grandeur.hum = humidity_perc;
        printf("*****\r\n");
        printf("hum:%f\r\n", humidity_perc);
    }

    if (reg.status_reg.t_da) {
        /* Read temperature data */
        memset(&data_raw_temperature, 0x00, sizeof(int16_t));
        hts221_temperature_raw_get(&dev_ctx, &data_raw_temperature);
        temperature_degC = linear_interpolation(&lin_temp,
                                                data_raw_temperature);

        grandeur.temp = temperature_degC;
        printf("*****\r\n");
        printf("temp:%f\r\n", humidity_perc);
    }
}
```

Une fois le programme terminé, nous le transférons dans notre microcontrôleur et observons les résultats dans la console:



```
PuTTY (inactive)
hum:40.635677
*****
temp:25.141100
*****
hum:48.349293
*****
temp:30.011578
*****
hum:50.564404
*****
temp:30.893633
*****
hum:57.110241
*****
temp:32.523518
*****
hum:63.606331
*****
temp:33.232994
*****
hum:67.759010
*****
temp:33.558971
*****
```

Comme nous l'observons dans la console, les valeurs d'humidité et de température semblent cohérentes, nous les avons fait augmenter en posant le doigt sur le capteur et on observe bien une augmentation dans les valeurs transcrites. Nous validons donc entièrement ce capteur.

4.5. Capteur de pression

Afin de collecter les données de notre capteur de pression, nous utilisons le bus I2C1 configuré sur les pins PB8 et PB9 de notre microcontrôleur. Puis nous développons le code en se basant une nouvelle fois sur le drivers à intégrer pour ce capteur.

```
void get_values_pressure_sensor_lps22hh(void)
{
    /* Read samples in polling mode (no int) */
    /* Read output only if new value is available */
    lps22hh_read_reg(&dev_ctx, LPS22HH_STATUS, (uint8_t *)&reg, 1);

    if (reg.status.p_da) {
        memset(&data_raw_pressure, 0x00, sizeof(uint32_t));
        lps22hh_pressure_raw_get(&dev_ctx, &data_raw_pressure);
        pressure_hPa = lps22hh_from_lsb_to_hpa( data_raw_pressure);
        printf("*****");
        printf("pression : %f\r\n",pressure_hPa);
    }
}
```

une fois le codage terminé, on test notre système en flashant le code dans notre microcontrôleur et on relève les valeurs affichées dans la console:



The image shows a side-by-side comparison. On the left is a web browser window displaying a weather forecast for Poitiers from METEO France. The forecast for the next 3 days is shown, with the current pressure at 1023 hPa. On the right is a terminal window titled 'PuTTY (inactive)' showing a series of pressure readings from the LPS22HH sensor. The readings are in hPa and range from approximately 1024.73 to 1025.04.

Pressure (hPa)
1024.736816
1024.684814
1024.674805
1024.690918
1024.714355
1024.675293
1024.663330
1024.739990
1024.732910
1024.831787
1024.837891
1024.861572
1024.873779
1024.872314
1024.850928
1024.809180
1024.891357
1024.937744
1024.961914
1024.976807
1025.032471
1024.969482
1025.042480

On observe bien des valeurs de pression affichées dans la console qui semblent correctes, mais n'ayant pas de moyen de mesurer la pression environnante ou encore de monter en altitude, nous sommes obligés de comparer ces valeurs aux prévisions météo de METEO FRANCE. En comparaison on trouve des valeurs similaires à celles prévues, nous pouvons donc valider ce capteur.

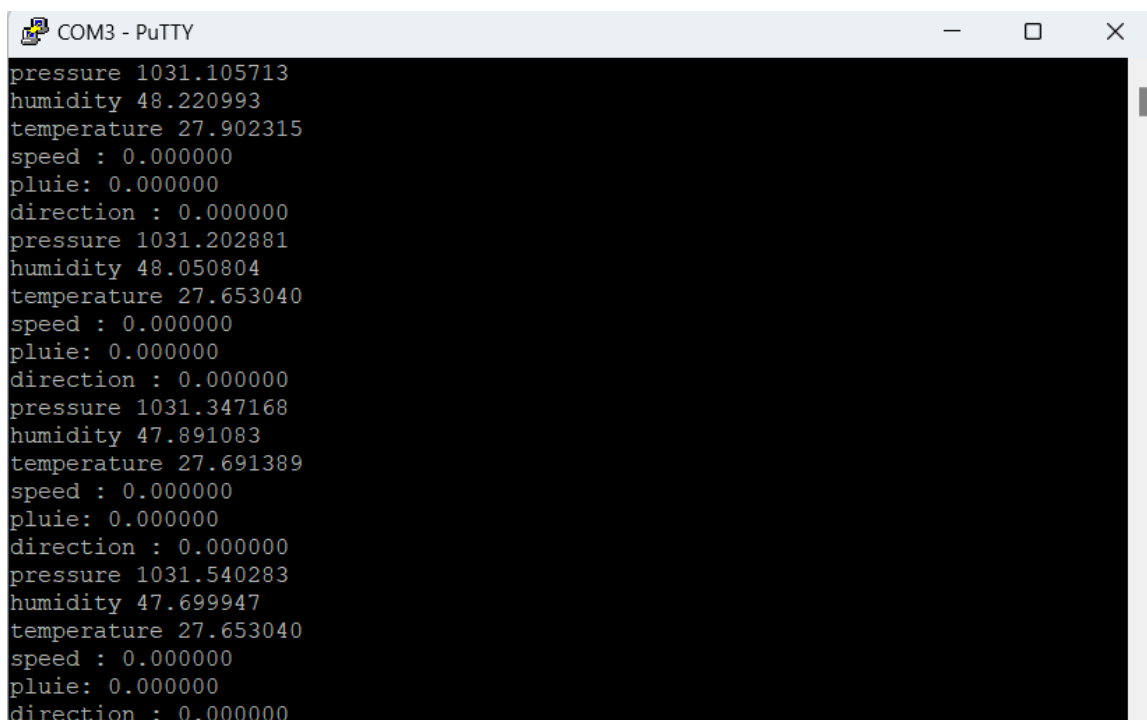
5.Développement et validation des éléments d'assemblage

Dans la partie précédente, nous avons validé le fonctionnement de tous les capteurs un à un, il s'agit désormais de les faire fonctionner en même temps afin de collecter les données en même temps. Une fois cette étape validée, il s'agira d' afficher les mesures sur l'écran.

5.1.Agrégation des mesures

5.1.1.Mesures de tous les capteurs

Une fois tous les codes validés, nous les avons assemblés afin de finaliser le projet et permettre un affichage de toutes les données en même temps. Nous avons ensuite vérifié que tout fonctionnait en utilisant le timer 5 on affiche toutes les 5 secondes les valeurs des capteurs dans la console. Pour cette manipulation nous n'avions pas branché les capteurs de la station météo mais simplement ceux de la carte Nucléo c'est pour cela qu'il y est affiché 0 mais nous utilisons quand même ,dans l'affichage, la valeur renvoyé par la fonction de lecture des valeurs des capteurs. Cela nous permet de valider l'agrégation des mesures puisque si cela ne fonctionnait pas il n'y aurait tout simplement pas de valeurs.



```
COM3 - PuTTY
pressure 1031.105713
humidity 48.220993
temperature 27.902315
speed : 0.000000
pluie: 0.000000
direction : 0.000000
pressure 1031.202881
humidity 48.050804
temperature 27.653040
speed : 0.000000
pluie: 0.000000
direction : 0.000000
pressure 1031.347168
humidity 47.891083
temperature 27.691389
speed : 0.000000
pluie: 0.000000
direction : 0.000000
pressure 1031.540283
humidity 47.699947
temperature 27.653040
speed : 0.000000
pluie: 0.000000
direction : 0.000000
```

5.1.2.Enregistrement sur la carte SD

Nous avons développé un code permettant de d'enregistrer les données acquises par les capteurs sur une carte microSD. Pour cela on commence par formater la carte SD puis on crée un fichier ".csv" avant de le remplir avec la date et l'heure ainsi que les valeurs du capteur souhaité.

```

void register_SD_CARD(float *array, int size) {
    if (f_mount(&SDFatFS, (TCHAR const *)SDPath, 0) != FR_OK) {
        Error_Handler();
    } else {
        printf("1er reussi\r\n");
        if (f_mkfs((TCHAR const *)SDPath, FM_ANY, 0, workBuffer, sizeof(workBuffer)) != FR_OK) {
            printf("Formatting failed\r\n");
            Error_Handler();
        } else {
            printf("2e reussi\r\n");
            if (f_open(&SDFile, "yala.csv", FA_OPEN_APPEND | FA_WRITE) != FR_OK) {
                printf("File opening for writing failed\r\n");
                Error_Handler();
            } else {
                printf("3e reussi\r\n");
                char buffer[100];
                UINT byteswritten;
                FRESULT res;
                snprintf(buffer, sizeof(buffer), "Year/Month/Day;Hour:Minute:Second;Value\r\n");
                res = f_write(&SDFile, buffer, strlen(buffer), (void *)&byteswritten);
                if ((byteswritten == 0) || (res != FR_OK)) {
                    printf("Error writing header to file\r\n");
                    Error_Handler();
                }
                printf("4e reussi\r\n");
                for (int i = 0; i < size; i++) {
                    RTC_TimeTypeDef sTime;
                    RTC_DateTypeDef sDate;
                    HAL_RTC_GetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
                    HAL_RTC_GetDate(&hrtc, &sDate, RTC_FORMAT_BIN);
                    snprintf(buffer, sizeof(buffer), "%04d/%02d/%02d;%02d:%02d:%02d;%d\r\n",
                        2000 + sDate.Year, sDate.Month, sDate.Date, sTime.Hours, sTime.Minutes, sTime.Seconds, array[i]);
                    res = f_write(&SDFile, buffer, strlen(buffer), (void *)&byteswritten);
                    if ((byteswritten == 0) || (res != FR_OK)) {
                        printf("Error writing to file\r\n");
                        Error_Handler();
                        break;
                    }
                }
                f_close(&SDFile);
                printf("Data written to CSV successfully\r\n");
            }
        }
    }
}

```

Nous avons inséré des “printf” dans le code pour pouvoir suivre l’évolution du stockage dans notre console :

```

COM3 - PuTTY
1er reussi
2e reussi
3e reussi
4e reussi
Data written to CSV successfully

1er reussi
2e reussi
3e reussi
4e reussi
Data written to CSV successfully

```

On récupère la carte SD et on vérifie que les données sont bien écrites dans un fichier “.csv”:

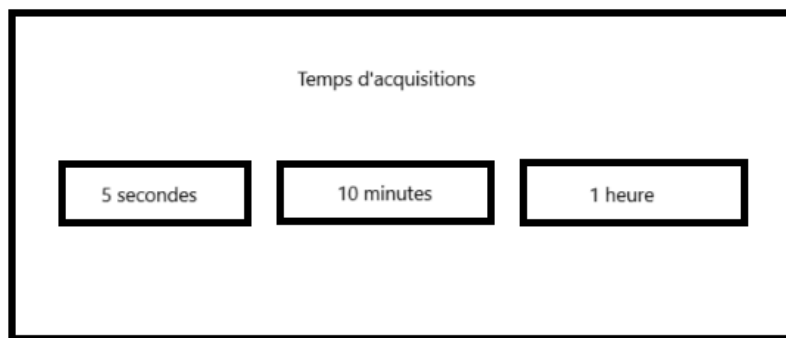
A1	Year/Month/Day			
	A	B	C	D
1	Year/Month/Day	Hour:Minute:Second	Value	
2	01/01/2000	00:01:52	28.113241	
3	01/01/2000	00:01:52	27.825615	
4	01/01/2000	00:01:52	27.825615	
5	01/01/2000	00:01:52	27.806440	
6	01/01/2000	00:01:52	26.866859	
7	01/01/2000	00:01:52	26.790159	
8	01/01/2000	00:01:52	26.694284	
9	01/01/2000	00:01:52	26.675108	
10	01/01/2000	00:01:52	26.617582	
11	01/01/2000	00:01:52	26.579233	
12	01/01/2000	00:01:52	26.560059	
13				

5.2.Gestion de l'IHM

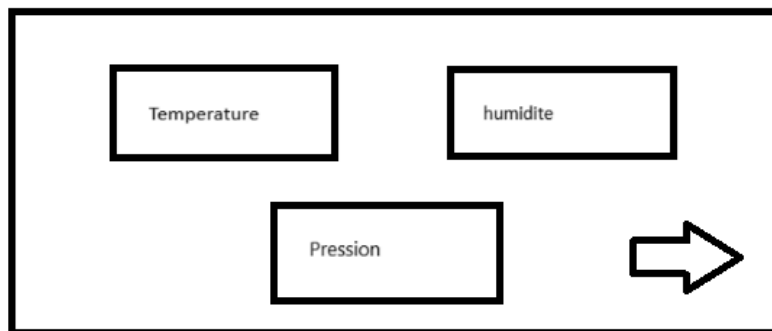
Afin de gérer l'interface homme machine, nous avons préalablement fait un croquis de ce que devrait être notre interface avec les différentes pages. Nous avons ensuite utilisé les fonctions de la BSP afin de dessiner sur l'écran les différents boutons dont on allait avoir besoin. Enfin pour créer un bouton next, nous avons importé une image bitmap.

5.2.1.Croquis des pages

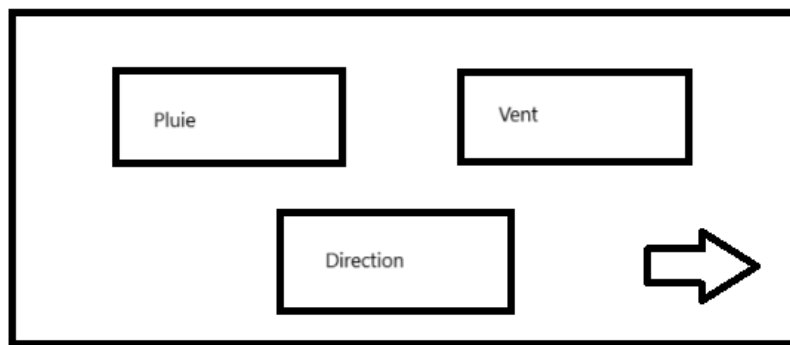
Page d'accueil:



Page 1:



Page 2:

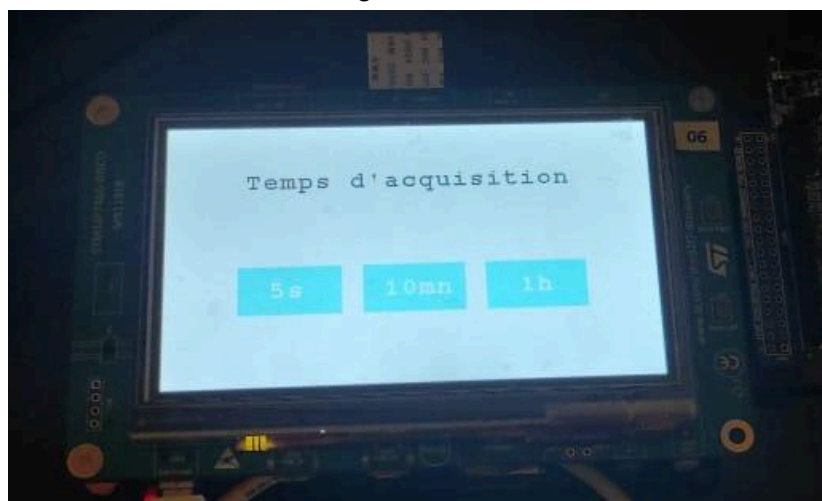


C'est donc à partir de ces pages ci que nous avons dessiné en lignes de code notre interface.

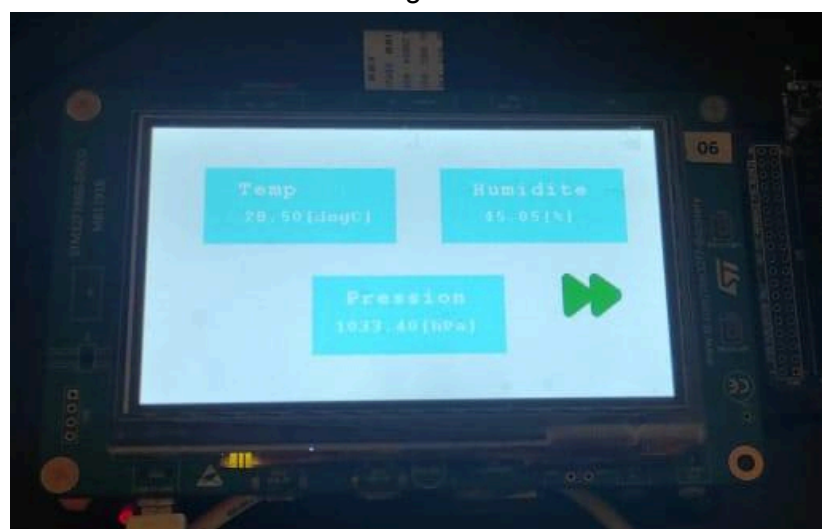
5.2.2. Affichage réel des pages

Une fois le code implémenté dans notre microcontrôleur, nous pouvons observer les différentes pages s'afficher:

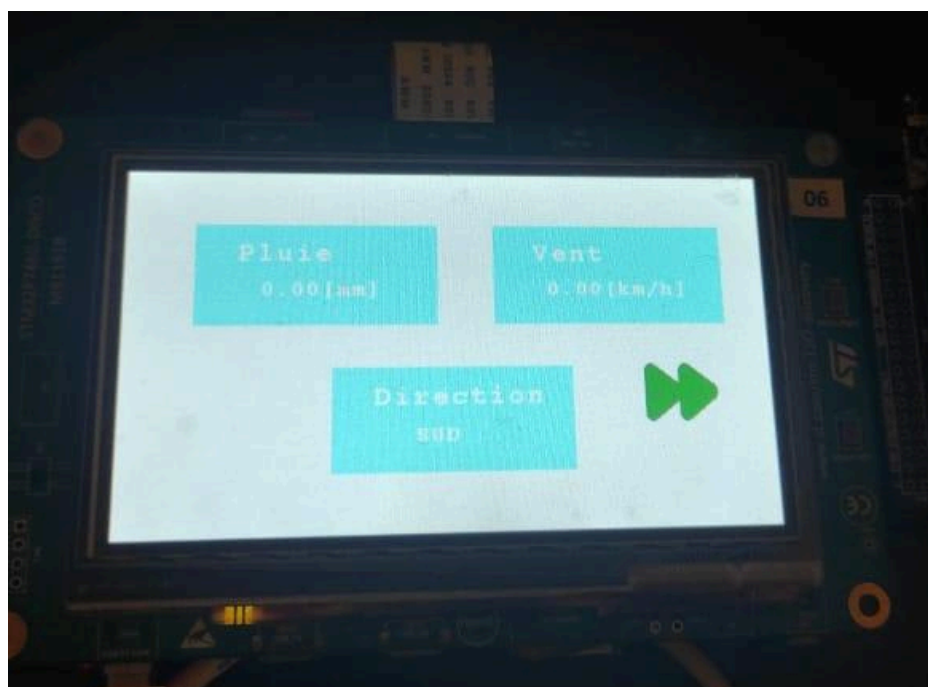
Page d'accueil:



Page 1:



Page 2:



Une fois l'intégration faite, on remarque que notre est bien fonctionnel et que les pages s'affichent bien. Il suffit de cliquer sur le bouton next pour passer d'une page à l'autre. Lorsque que le système s'allume, il propose à l'utilisateur de choisir le temps d'acquisition puis passe à la page 1 directement après le choix.

6.Développement et validation de l'application

Une fois les différents codes assemblés, il reste plus qu'à utiliser notre application en situation réelle. Nous avons donc effectué différents tests sur notre application en veillant bien à vérifier que les données acquises étaient correctes. Une fois ces tests validés, nous avons estimé que le développement de cette application touchait à sa fin.

7.Tableau récapitulatif des fonctionnalités réalisées

Fonctionnalité du système	Tâche réalisée	Tâche implémentée	Tâche validée
Capteur de température et humidité	oui	oui	oui
Capteur de pression	oui	oui	oui
Capteur de vitesse du vent	oui	oui	oui
Capteur de direction de vent	oui	oui	oui

Capteur de pluie	oui	oui	oui
Led RGB	oui	oui	oui
Mise en repos de l'écran	oui	oui	oui
Choix du temps d'acquisition	oui	oui	oui
Affichage des valeurs	oui	oui	oui
Intégration de la RTC	oui	oui	oui

8.Conclusion

Ce projet de station météorologique autonome nous a permis d'atteindre les objectifs fixés tout en développant des compétences techniques et organisationnelles significatives en systèmes embarqués notamment sur le l'objet STM32F746G disco. En suivant une méthodologie rigoureuse, nous avons conçu un système capable de mesurer, analyser, stocker et visualiser des données météorologiques en temps réel avec fiabilité et précision. La conception modulaire, associée à une validation unitaire de chaque composante, a assuré une intégration finale fluide et robuste. Nous avons appris à gérer des données en temps réel, à les stocker efficacement sur une carte SD, et à concevoir une interface utilisateur intuitive, garantissant une expérience utilisateur optimale. Par ailleurs, le travail en équipe a été un élément central, nous enseignant la répartition des responsabilités, la coordination des efforts, et le suivi précis de l'avancement des tâches. Ce projet nous a également permis d'approfondir nos connaissances en électronique, programmation embarquée, machines d'état et développement d'IHM. En somme, il a constitué une expérience complète et formatrice, renforçant nos compétences techniques tout en développant notre capacité à mener à bien des projets complexes, de leur conception à leur intégration finale.

9.Composition et responsabilités dans l'équipe projet

9.1.Composition de l'équipe

AHAMED Layla
BEGOIN Arthur
ITOUA NGALOMI Bil Théodore

9.2. Table de description des tâches réalisées par personne de l'équipe projet

NOM Prénom	Tâche affectée	Tâche effectuée
ITOUA Théo	Implémentation IHM	Oui
ITOUA Théo	Implémentation Carte Nucleo	Oui
ITOUA Théo	Implémentation fonctionnement général	Oui
BEGOIN Arthur	Implémentation acquisition Pluviomètre	Oui
BEGOIN Arthur	Implémentation carte SD	Oui
BEGOIN Arthur	Calcul ARR et PSC de timers	Oui
ITOUA & BEGOIN & AHAMED	Tests et validations du programme	Oui
ITOUA & BEGOIN & AHAMED	Rédaction rapport	Oui
AHAMED Layla	Configuration projet et création	Oui

10. Annexe

Annexe 1:



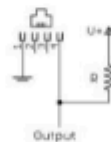
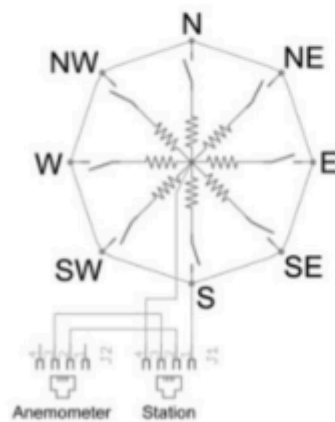
Shenzhen Fine Offset Electronics Co., Ltd
深圳市欧赛特电子有限公司

Address: 4/F, Block C, Jiujiu Industrial City, Shajing Town, Bao'an District, Shenzhen City, China

the anemometer and wind vane(pins 2 and 3)

Wind Vane

风向计它有八个开关，每个开关连接到一个不同的电阻器。叶片的磁铁可以同时关闭两个开关，使多达 16 个不同的位置被指示。外部电阻可以用来形成一个分压器，产生一个电压输出。可以用 a/d 转换器来测量。如下所示。



Example wind vane interface circuit. Voltage readings for a 5 volt supply and a resistor value of 10k ohms are given in the table.

Direction (degrees)	Resistance (ohms)
0	33K
22.5	6.57K
45	8.2K
67.5	891
90	1K
112.5	688
135	2.2K
157.5	1.41K
180	3.9K
202.5	3.14K
225	16K
247.5	14.12K
270	120K
292.5	42.12K
315	64.9K
337.5	21.88K

表中给出了所有 16 个可能位置的电阻值。

The wind vane is the most complicated of the three sensors. It has eight switches, each connected to a different resistor. The vane's magnet may close two switches at once, allowing up

11.Bibliographie

<https://learn.sparkfun.com/tutorials/weather-meter-hookup-guide#resources-and-going-further>
<https://meteofrance.com/previsions-meteo-france/poitiers/86000>