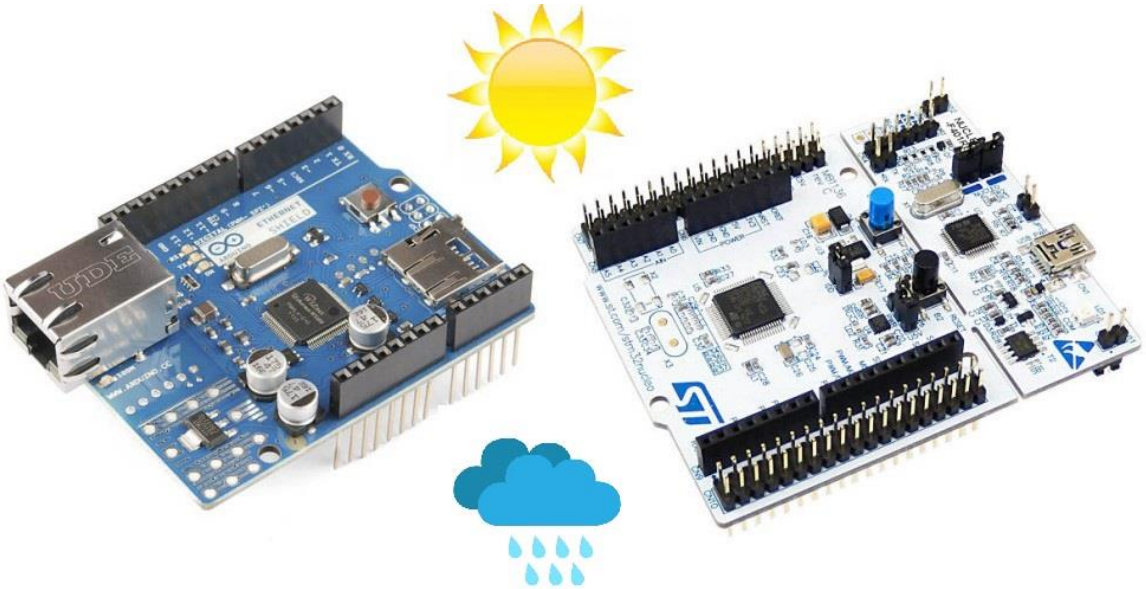
UNIVERSITE TOULOUSE III PAUL SABATIER
Faculté Sciences et Ingénierie

Rapport de projet



BE Web Server Météo STM32

JACQ Léo
YALA AMINE

Année 2021/2022

Table des matières

Introduction

Les stations météos comportent souvent une centrale de capture fixe permettant l'acquisition des données sur une zone assez large. Le souci de cela est que pour obtenir plus de données il faut multiplier ces bases qui, bien que performantes, représentent un cout élevé en matériel et installation.

Le but de ce projet est de montrer qu'avec un réseau utilisant la technologie XBEE il est possible de multiplier ces zones de capture en les faisant communiquer et centraliser ces données pour les consulter tout en ayant des composants simple d'utilisation et une technologie performante.

La réalisation de ce projet se fera sur une base de carte STM32 NUCLEO avec différents capteurs.

La première partie de ce rapport portera sur la réalisation des TP de base qui nous ont permis d'exploiter et comprendre le fonctionnement les capteurs nécessaires au développement du Web Server Météo.

TP de base Amine

Matériel utilisé

- Carte STM32 NUCLEO L152RE, plateforme ARM 32 bits/ 32 MHz et 64 broches.
- Capteur humidité/température SHT31 (Communication I2C)
- Afficheur LCD : Grove LCD RGB Backlight V4.0
- Base shield arduino V2 pour capteur grove.

Schéma et câblage

Pour cette partie, les deux composants utilisés pour la mise en œuvre ont un protocole de communication I2C (Ecran LCD et capteur SHT31).

Le protocole I2C définit la continuité des états logiques possibles sur SDA et SCL et comment le circuit doit réagir en cas de conflit. Sur le bus I2C, le niveau logique dominant est le 0. En l'absence de commande le niveau logique de repos est le 1. Pour contôler un bus il doit être dans un état repos (SDA et SCL = 1).

Condition d'arrêt : SDA devient 1 SCL reste à 1.

Pour utiliser cette communication et la gestion d'E/S des bibliothèques sont disponibles.

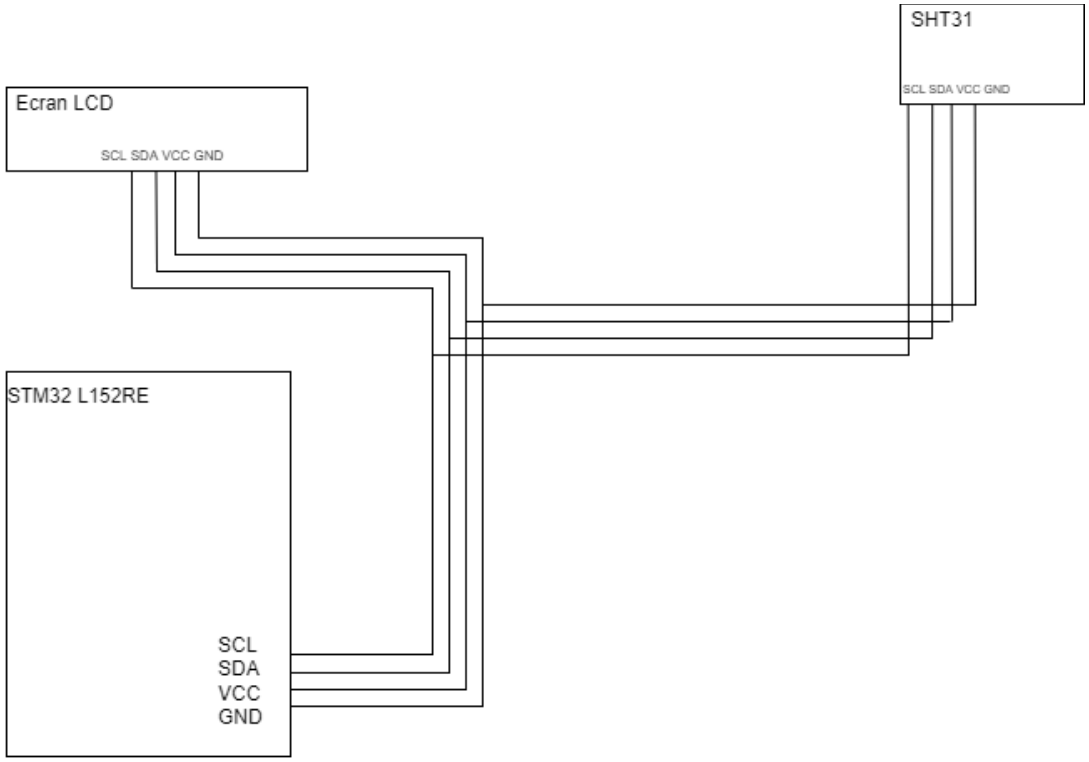


Figure 1 : Base câblage des composants

Programme

Principe

Le diagramme ci-dessous présente simplement la routine du programme permettant la collecte des données de températures et d'humidité ainsi que l'affiche sur l'écran LCD.

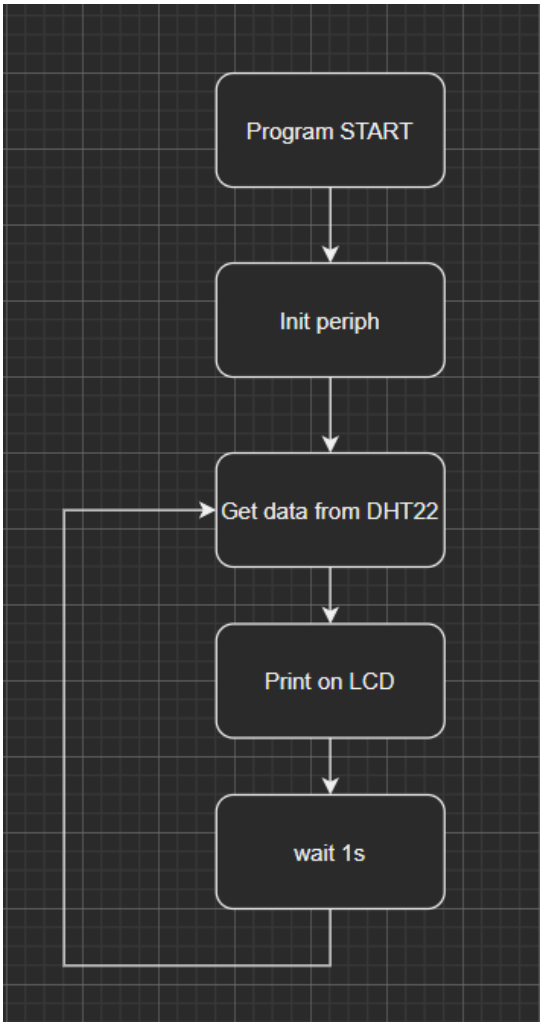


Figure 2 : Principe du programme

Initialisation de pins

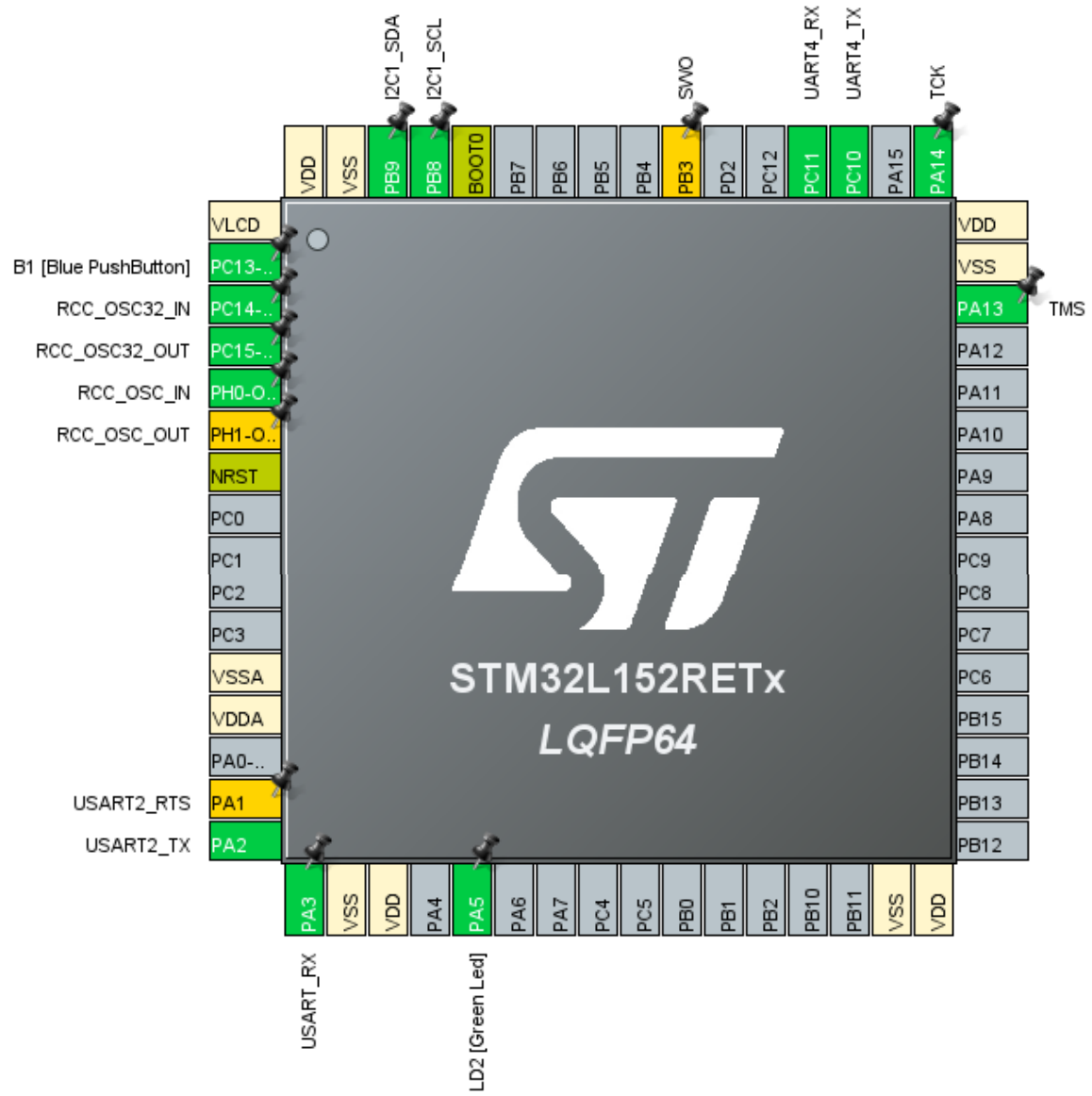


Figure 3 : Pin affectés

Communication I2C

Répartition

Comme les deux composant communique en I2C, il est important d'avoir un ordre d'envoie et de réception de données avec un période prédéfinie qu'on peut observer sur pico scope.

Sur cette partie, j'ai fait le choix de faire une lecture et un affichage de la température et de l'humidité toutes les 1000 ms.

Avant l'envoi ou la réception de la donnée une trame comportant l'adresse et le mode sur lequel on est (Write ou Read) est envoyée comme suit :

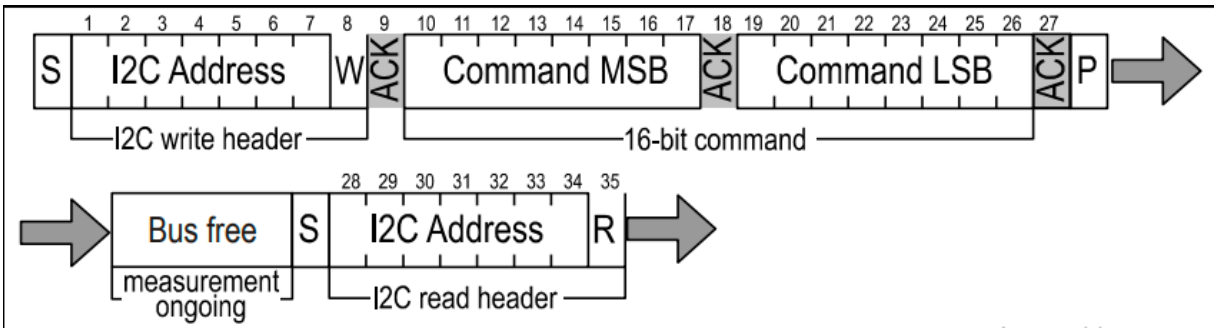


Figure 4 : Protocole I2C adresse + W/R

Pour notre cas, de manière périodique, nous allons d'abord nous mettre en mode lecture de la température & de l'humidité ensuite en monde écriture afin de les afficher sur le LCD.

SHT31

Fonctionnement du capteur

Comme cité précédemment, ce capteur fonctionne avec le protocole I2C.

Après avoir lu la datasheet du composant j'ai bien compris comment communiquer ce composant.

Sa trame est bien expliquée comme suit :

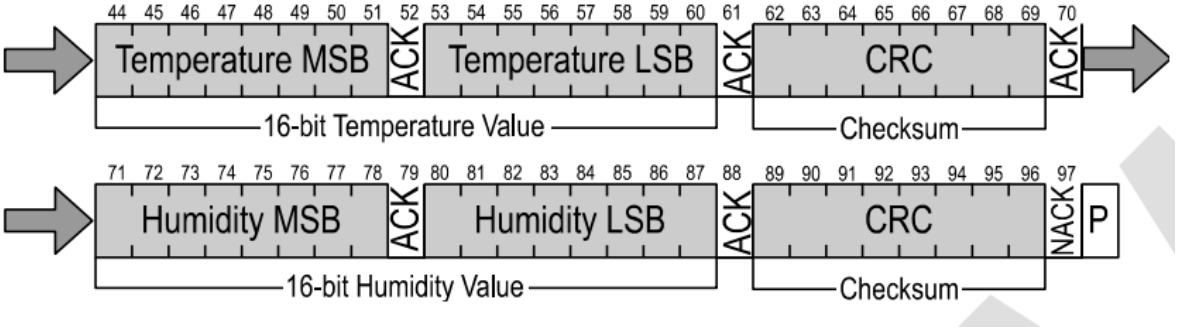


Figure 5 : Séquence envoyé par le capteur

Données brutes de la température : 16 bits (2 octets).

Données brutes de l'humidité : 16 bits (2octets).

Checksum : 2*8 bits.

Un acknowledge tous les 8bits.

Un bit NACK de fin de transmission.

Somme : 32 bits à décoder. + 16 bits de checksum + 6 acknowledge = 54 bits.

Initialisation du SHT31 et du LCD

Après avoir recherché des tutoriels qui traiter le même type de capteur et d'écran lcd, j'ai pu trouver une bibliothèque que répond à nos attentes :

Afin d'initialiser le capteur, j'ai utilisé cette fonction :

```
/************* Initialisation du capteur *****************/

void TMP_init(I2C_HandleTypeDef hi2c)
{
    hi2c_tmp = hi2c;
}
```

Figure 6 initialisation du capteur

Cette fonction est rappelée au début du main afin d'initialiser le capteur.

Fonctions utiles du SHT31

Afin de récupérer la température et l'humidité, on se doit d'utiliser une fonction de transmission de données, une fonction de réception des données et une fonction pour décoder la trame.

```
/*************Reception de la temperature et de l'humidite ****************/

void TMP_Receive(uint16_t address, uint8_t *Data, uint16_t len)
{


    HAL_I2C_Master_Receive(&hi2c_tmp, address, Data, len, TIMEOUT);

}
```

```
/************* Transmission du capteur *****************/
void TMP_Transmit(uint16_t address, uint8_t *Data, uint16_t len)
{
    HAL_I2C_Master_Transmit(&hi2c_tmp, address, Data, len, TIMEOUT);
}
```

```
/************* Décodage de la trame du capteur ****************/

void Temp_read( float *temp, float *humidity)
{
    float temperature=0;

    uint8_t Data[6]={0x24, 0x00};
    TMP_Transmit(ADRESSETEMP, Data, 2);
    HAL_Delay(50);
    TMP_Receive(ADRESSETEMP, Data, 6);

    temperature = Data[0] * 256 + Data[1];
    temperature = -45 + (175 * temperature / 65535.0);
    *temp=-45+175*(Data[0]<<8 | Data[1])/65535.0 ;
    *humidity=100*(Data[3]<<8 | Data[4])/65535.0 ;

}
```

Figure 7 : Fonctions utilisées sur le SHT31

Appel aux fonctions sur le while(1) du main

Toutes le fonctions utilisées pour lire et afficher la température et l'humidité sont appelées dans un while(1) dans le main afin de faire une lecture et un affichage cycliques et réguliers.

```
/***** Appel à la lecture et l'affichage de la temperature et l'humidité à partir du main ******/

 Temp_read(&temperature, &humidity);

 ftoat (temperature,res,1);
 lcd_position(&hi2c1,12,0);
 lcd_print(&hi2c1,res);

 ftoat (humidity,res,1);
 lcd_position(&hi2c1,10,1);
 lcd_print(&hi2c1,res);


print("la temperature est =%f\n\r ", temperature);
print("l'humidité est =%f\n\r ", humidity);
 HAL_Delay(1000);
```

Figure 8 : Lecture et affichage du main

Observation au pico scope

Afin d'observer ce qui se transmet entre la STM32 et le capteur/lcd, on utilise un pico scope qui nous permet de visualiser les signaux souhaités sur un ordinateur. Après avoir paramétré une lecture en protocole I2C on observe :

LCD

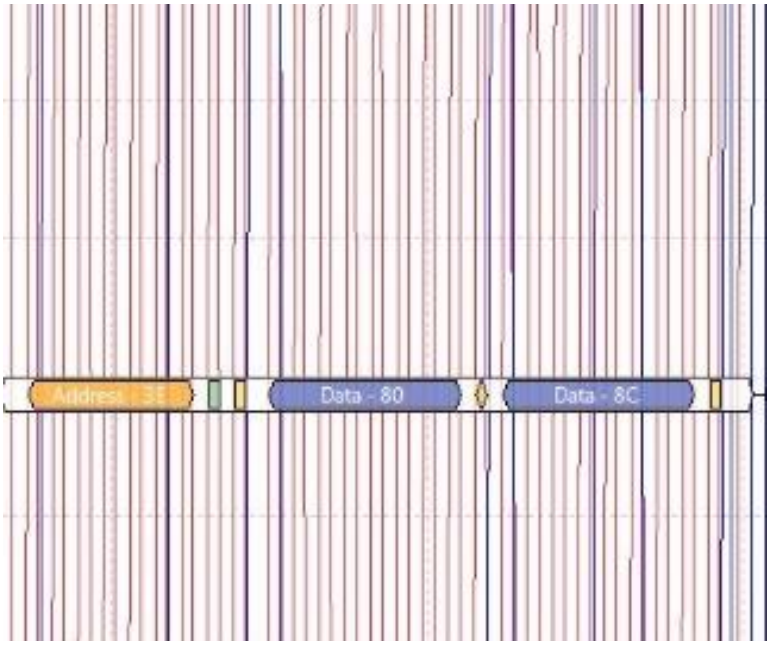


Figure 9 : Observation des signaux SCL et SDA

3 E correspond à l'adresse du LCD, on a pu le voir aussi sur sa datasheet. 80 et 8C correspondent aux LSB et MSB de la donnée à afficher sur l'écran.

SHT 31

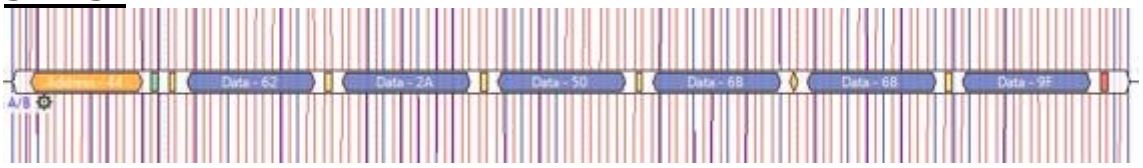


Figure 10 : Observation des signaux SCL et SDA

Paquet	Heure de début	Heure de fin	Address Bits	Address	Address + R/W	R/W	Addres
1	603,6 ms	603,9 ms	7	44	88	Write	0
2	654,6 ms	655,3 ms	7	44	89	Read	0
3	655,4 ms	655,7 ms	7	3E	7C	Write	0
4	655,7 ms	656 ms	7	3E	7C	Write	0
5	656 ms	656,3 ms	7	3E	7C	Write	0
6	656,3 ms	656,6 ms	7	3E	7C	Write	0
7	656,6 ms	656,9 ms	7	3E	7C	Write	0
8	656,9 ms	657,2 ms	7	3E	7C	Write	0
9	657,2 ms	657,5 ms	7	3E	7C	Write	0
10	657,5 ms	657,8 ms	7	3E	7C	Write	0
11	657,8 ms	658,1 ms	7	3E	7C	Write	0
12	658,1 ms	658,4 ms	7	3E	7C	Write	0
13	1,664s	1,664s	7	44	88	Write	0
14	1,715s	1,716s	7	44	89	Read	0
15	1,716s	1,716s	7	3E	7C	Write	0
16	1,716s	1,716s	7	3E	7C	Write	0
17	1,716s	1,717s	7	3E	7C	Write	0
18	1,717s	1,717s	7	3E	7C	Write	0
19	1,717s	1,717s	7	3E	7C	Write	0
20	1,717s	1,718s	7	3E	7C	Write	0
21	1,718s	1,718s	7	3E	7C	Write	0
22	1,718s	1,718s	7	3E	7C	Write	0
23	1,718s	1,718s	7	3E	7C	Write	0

Figure 11 : Décodage de la trame par l'application PICO SCOPE

TP de base Léo

Matériel

Le matériel utilisé dans le cadre de ce TP sera une carte SMT32 NUCLEO L152RE, un capteur humidité/température DHT22 et un afficheur LCD.

Pour faire une brève description des caractéristiques des composants nous avons :

- Plateforme ARM 32 bits basse consommation de 32Mhz et 64 broches.
- Capteur d'humidité/température communication one wire.
- LCD 2lignes 16 colonnes RGB communication I2C.

Le tout est connecté grâce à un base shield V2 pour capteur grove. Ainsi la partie câblage est donc simplifiée.

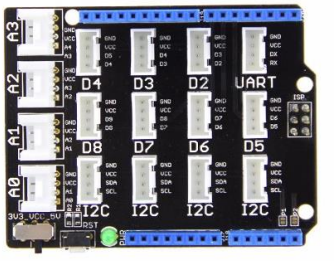


Figure 12 : Base shield arduino

Toutefois cette partie n'a pas totalement négligée car nécessaire au bon paramétrage du microcontrôleur.

Schéma et câblage

Le brochage des composants se fera comme suit :

Broche MCU	Broche capteur
PA1	Signal (SGN)
PB8	SCL
PB9	SDA

Tableau 1 : Brochage

La communication I2C et la gestion des entrées/sorties sont directement gérées par les différentes bibliothèques disponibles. La communication one-wire a été faite manuellement en modifiant la direction de la broche PA1 pour alterner entre entrée/sortie.

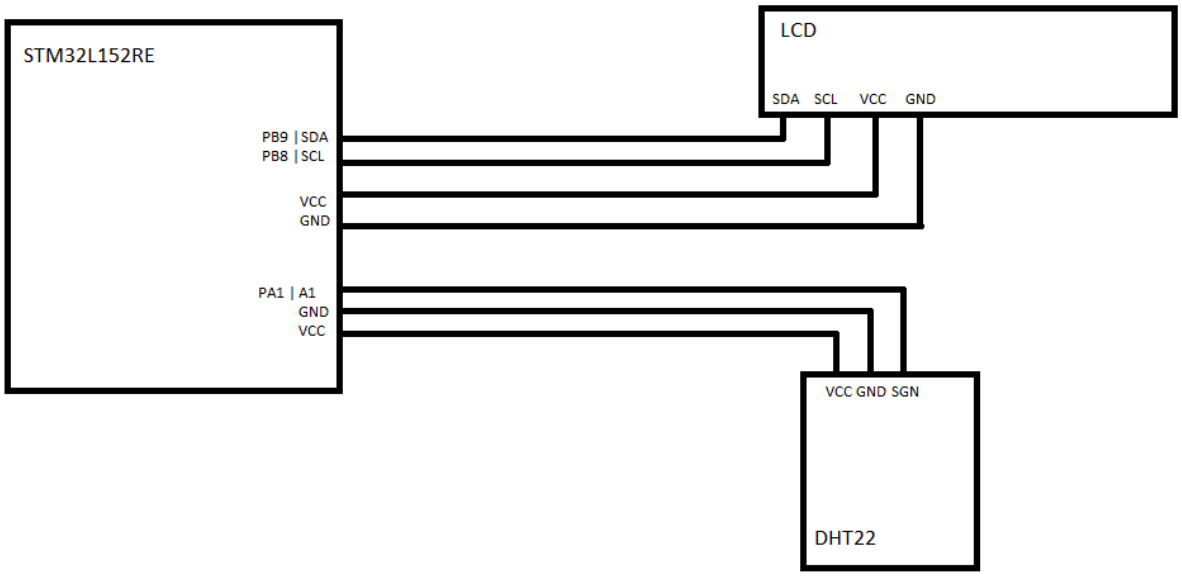


Figure 13 : Base câblage des composants

Logique programme

Le diagramme ci-dessous présente simplement la routine du programme permettant la collecte des données de températures et d'humidité ainsi que l'affiche sur l'écran LCD.

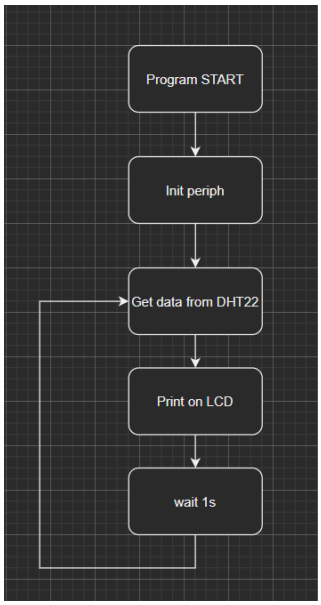


Figure 14 : Principe du programme

Projet Cube

La mise en place des fichiers de projet se fait au démarrage grâce à l'outil STM32CubeMX qui est un outil graphique de configuration de microcontrôleur. Il permet de générer un code de base pour initialiser simplement les périphériques et IO du microcontrôleur.

Ce logiciel permet donc de programmer les IO mais aussi de gérer les différentes clocks.

En plus des deux bus de communications pour le LCD et le capteur nous avons eu besoin d'un timer externe de 32 Mhz avec un pré-diviseur de 32 afin d'obtenir des périodes de 1microsecondes, unité de temps nécessaire à l'exploitation du DHT22.

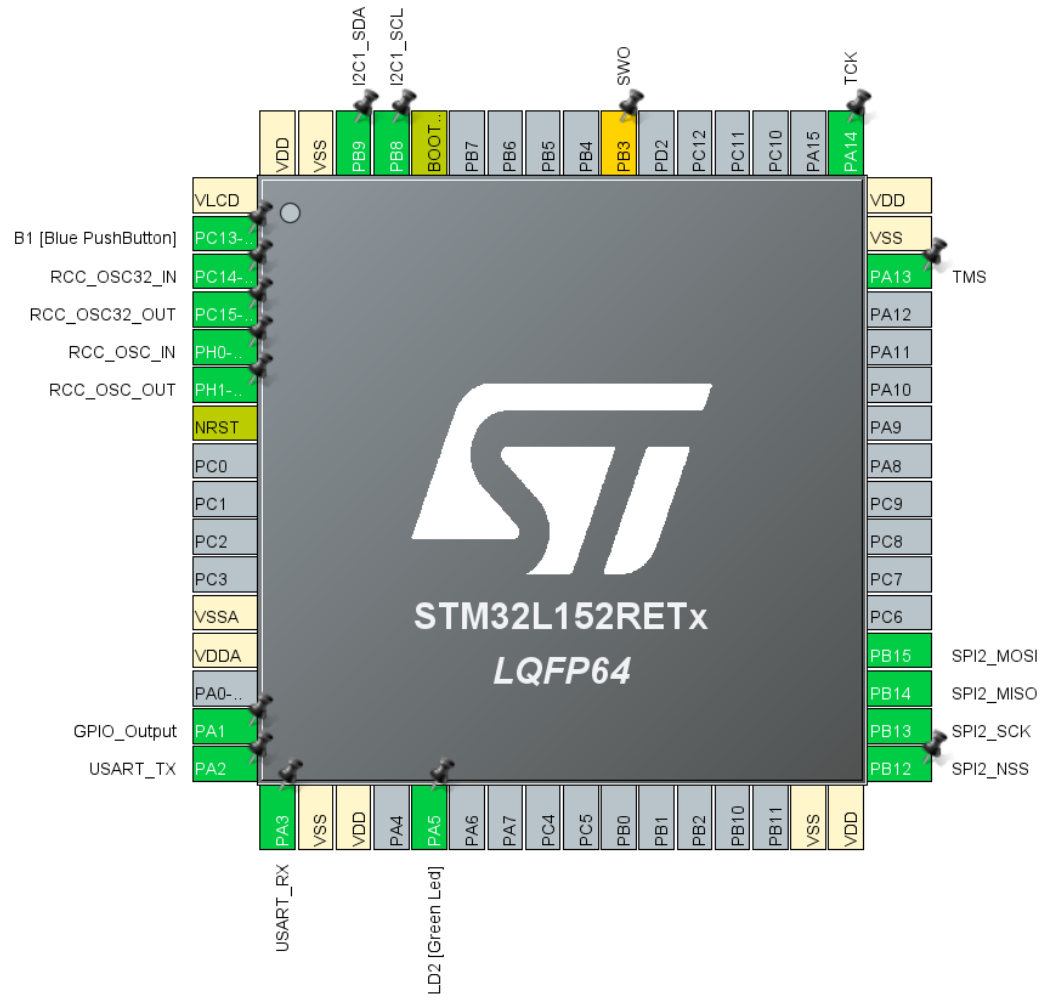


Figure 15 : Pin affectés

Délais microseconde

La couche d'abstraction matérielle ne comporte pas de fonction délais prenant en charge les microsecondes. Pour ce faire il nous faut donc dédier un timer afin d'obtenir les fonctions souhaitées. J'ai donc paramétré le timer 2 avec une fréquence de 32mhz et un pré diviseur de 32 afin d'obtenir une fréquence de fonctionnement de 1Mhz soit une microseconde de période.

La fonction delay_us() va donc prendre en paramètre un entier correspondant à la période d'attente. Au départ, le compteur du timer va être remis à 0 et la boucle while va s'exécuter tant que le compteur n'a pas atteint la valeur spécifiée dans la fonction

```
/* USER CODE BEGIN 1 */
void delay_us(uint16_t us)
{
    __HAL_TIM_SET_COUNTER(&htim2,0);  // set the counter value a 0
    while (__HAL_TIM_GET_COUNTER(&htim2) < us); // wait for the counter to reach the us input in the parameter
}

/* USER CODE END 1 */
```

Figure 16 : Fonction délais microseconde

Code DHT22

Fonctionnement du capteur

Avant de tenter de récupérer les données, il m'a fallu comprendre comment fonctionnait le capteur et donc lire la documentation technique de ce dernier.

Comme énoncé plus haut le capteur fonctionne en one wire, c'est donc la même broche qui servira d'entrée pour les demandes de conversion et de sortie pour l'envoi des données. Le protocole de communication est défini comme suit :

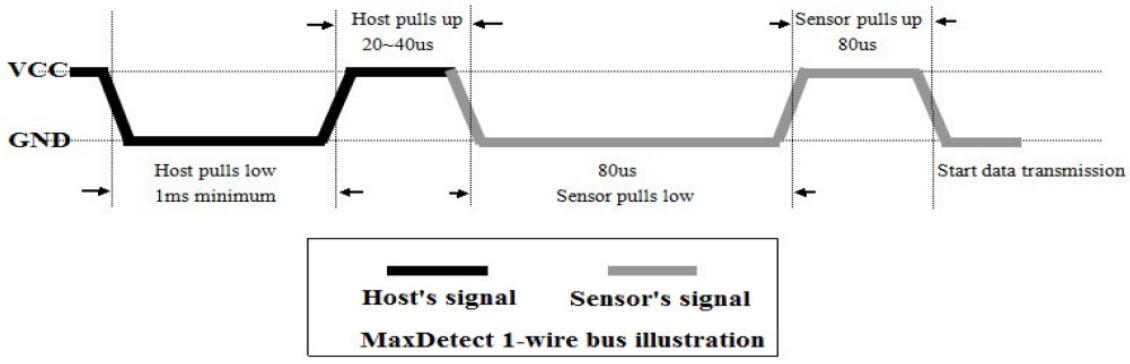


Figure 17 : Séquence de réveil du capteur

Au repos, la ligne est à l'état haut, pour se « réveiller » le capteur attend un signal à l'état bas pendant au moins 1ms, puis un signal haut entre 20 et 40us. A ce moment-là le capteur va répondre en mettant le signal l'état bas pendant 80us puis à l'état haut pendant 80us.

Une fois cette séquence écoulée commence l'envoi de la trame de données de 40 bits.

Chaque bit est séparé d'un signal de 50us à l'état bas puis le temps à l'état haut définira si le bit est un 1 ou un 0.

Le 0 est défini par un signal haut entre 26 et 28us.

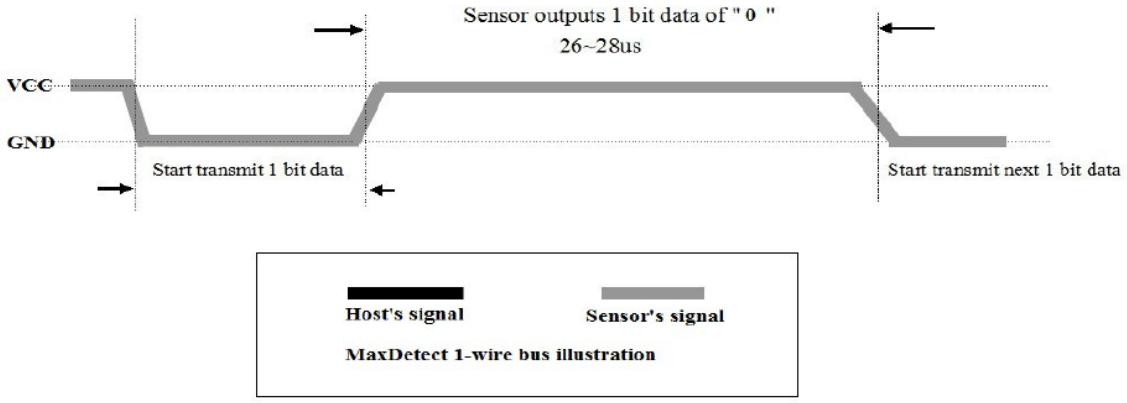


Figure 18 : Envoi 0

Le 1 est défini par un signal haut de 70us.

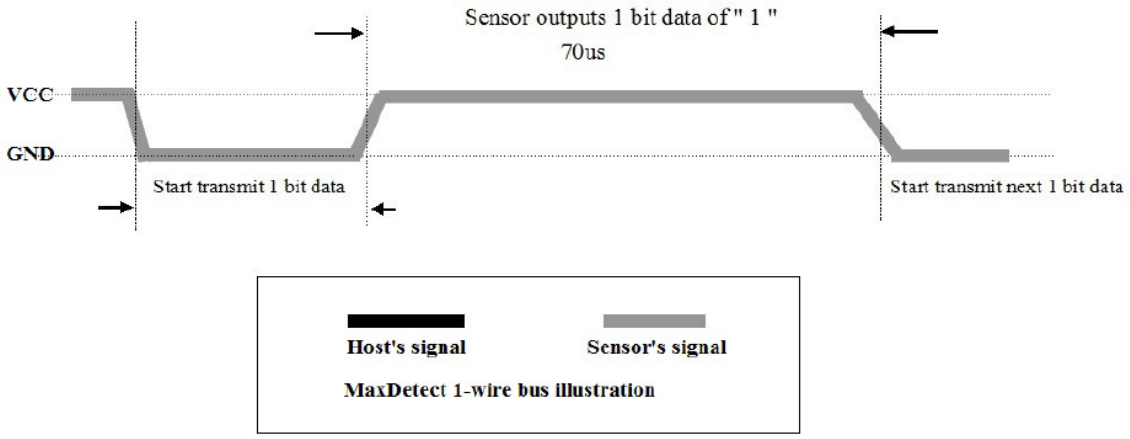


Figure 19 : Envoi 1

Une fois la séquence de 40 bits reçue il nous faut la décoder.

Octet 1	Octet 2	Octet 3	Octet 4	Octet 5
Humidité		Température		Checksum

Tableau 2 : Trame DHT22

Les données brutes d'humidité et de température sont codées sur 16 bits. Pour obtenir les valeurs de mesurandes il faut convertir ces 16 bits en base décimale et diviser le tout par 10.

Les 8 derniers bits servent au checksum pour la sécurité. Son calcul se fait par addition de tous les octets de données de la trame (octets 1-4). Si la valeur obtenue est identique à celle de la trame alors il n'y a pas d'erreur de transmissions, dans le cas contraire la trame a été altérée.

Initialisation

Plusieurs tutoriels et exemples d'utilisations à propos du DHT22 existent mais soit ne fonctionnait pas soit laissait des failles dans l'utilisation du programme.

J'ai donc développé ma propre bibliothèque afin d'utiliser le module plus simplement.

La fonction d'initialisation prend en paramètres une structure de type DHT22 ainsi que les ports et pin physiques reliés au module.

```
struct DHT22{

    GPIO_TypeDef *GPIOx;
    uint16_t GPIO_Pin;
    float temperature;
    float humidity;
};
```

Figure 20 : Structure DHT22

La structure DHT22 permet de définir un objet à chaque module en gardant en paramètre les données de températures et humidité mais aussi de raccordement au microcontrôleur, ainsi on peut utiliser facilement et distinctement plusieurs capteurs dans un seul programme.

Mesure

Comme nous avons pu le voir la récupération des données s'effectue en trois étapes :

-Réveil du capteur

-Réponse du capteur

-Envoi des données

```
if(DHT22_Start(&DHT22_1) == 0)
{
    if( DHT22_Check_Response(&DHT22_1)== 0)
    {
        if(DHT22_Read_Temp_Hum(&DHT22_1) == 0)
        {
            clearlcd();

            lcd_position(&hi2c1,0,0);

            memset(temp,0,sizeof(temp));
            sprintf((char*)temp,"Hum: %.2f %c ",DHT22_1.humidity, 0x25);
            lcd_print(&hi2c1,(char*)temp);

            lcd_position(&hi2c1,0,1);

            memset(temp,0,sizeof(temp));
            sprintf((char*)temp,"Temp: %.2fC  ",DHT22_1.temperature);
            lcd_print(&hi2c1,(char*)temp);

            HAL_Delay(1000);
        }
        else
        {
            HAL_Delay(10);
        }

    }
    else
    {
        HAL_Delay(10);
    }

}
else
{
    HAL_Delay(10);
}
```

Figure 21 : Corps du while

La fonction DHT22_Start() prend en paramètre l'adresse de la structure DHT22. Et joue la séquence de réveil spécifié sur la Figure 6.

```
/*Set pinmode to output and send > 1ms low signal,  20-40 us high signal and set input*/
uint8_t DHT22_Start (struct DHT22 *sensor_DHT22)
{
    DHT22_Set_Output(sensor_DHT22);
    HAL_GPIO_WritePin (sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin, GPIO_PIN_RESET);
    delay_us(1200);
    HAL_GPIO_WritePin (sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin, GPIO_PIN_SET);
    delay_us(30);
    DHT22_Set_Input(sensor_DHT22);

    return 0;
}
```

Figure 22 : Fonction DHT22_Start()

Une fois la séquence effectuée le pin est définie en entrée grâce à la fonction DHT22_Set_Input(), cette fonction va redéfinir les paramètres générés par CubeMX afin de définir le pin du DHT222 comme une entrée.

```
void DHT22_Set_Input (struct DHT22 *sensor_DHT22)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = sensor_DHT22->GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(sensor_DHT22->GPIOx, &GPIO_InitStruct);
}
```

Figure 23 : Fonction DHT22_Set_Input()

La même fonction pour rebasculer le pin en sortie a été implémentée.

```
void DHT22_Set_Output(struct DHT22 *sensor_DHT22)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = sensor_DHT22->GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(sensor_DHT22->GPIOx, &GPIO_InitStruct);

}
```

Figure 24 : Fonction DHT22_Set_Output()

La prochaine étape est donc de regarder la réponse du capteur suite à son activation. Pour ce faire et comme mentionné dans la datasheet du composant nous devant attendre deux changements d'états espacés chacun de 80 microsecondes. Afin d'éviter un blocage du programme dans une boucle ou un cas où le capteur ne répondrait pas ou serait débranché, un contrôle de sécurité est effectué pour terminer l'attente. Dans ce cas, est renvoyé une erreur pour la fonction afin d'indiquer que la communication a échouée.

```
/*Wait sensor response, 80 us low signal and 80 us high signal*/
uint8_t DHT22_Check_Response (struct DHT22 *sensor_DHT22)
{
    uint8_t wd_timer = 0;
    while(!(HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 85))
    {
        delay_us(1);
        wd_timer++;
    }

    if(wd_timer == 85)
    {
        return 1;
    }
    else
    {

        wd_timer = 0;
        while((HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 85))
        {
            delay_us(1);
            wd_timer++;
        }

        if(wd_timer == 85)
        {
            return 1;
        }
        else
        {
            return 0;
        }
    }

}
```

Figure 25 : Fonction DHT22_Check_Response()

Une fois le capteur réveillé et ayant répondu, ce dernier va envoyer les données sur 40 bits

La fonction DHT22_Read_raw va lire un octet de données bit pas bit en calculant le temps à l'état haut du signal du capteur et ainsi déterminer s'il s'agit d'un 0 ou d'un 1. L'octet est renvoyé dans un pointeur ce qui permet de gérer les erreurs si le signal ne correspond pas avec les données attendues.

```c
uint8_t DHT22_Read_raw (struct DHT22 *sensor_DHT22, uint8_t * data)
{
    uint8_t i = 0;
    uint8_t wd_timer = 0;

    for (i=0;i<8;i++)
    {
        wd_timer = 0;

        //Start bit of 50us
        while(!(HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 50))
        {
            delay_us(1);
            wd_timer++;
        }

        wd_timer = 0;

        while((HAL_GPIO_ReadPin(sensor_DHT22->GPIOx, sensor_DHT22->GPIO_Pin)) && (wd_timer < 90))
        {
            wd_timer += 10;
            delay_us(10);
        }


        if((wd_timer >= 20)&&(wd_timer <= 30 ))
        {
            *data &= ~1<<(7-i);
        }
        else if((wd_timer >= 60)&&(wd_timer <= 80 ))
        {
            *data |= 1<<(7-i);
        }
        else
        {
            return 1;
        }
    }

    return 0;
}
```

Figure 26 : Fonction DHT22_Read_Raw()

Chaque octet de la trame va donc pouvoir être lu à la suite et utilisé pour la calcul de la checksum. Si le calcul est juste alors on peut actualiser les données de l'objet DHT22, dans le cas connaitre on renvois une erreur.

Pour obtenir les valeurs de température et d'humidité il ne reste plus qu'a assembler les octets de poids forts et octets de poids faible de chaque mesure puis de diviser par 10 comme spécifié dans la datasheet et les données peuvent maintient être affichées.

```c
uint8_t DHT22_Read_Temp_Hum (struct DHT22 *sensor_DHT22)
{
    uint8_t Rh_byte1 = 0, Rh_byte2 = 0, Temp_byte1 = 0, Temp_byte2 = 0 , SUM = 0, SUM_temp = 0, read_error = 0;
    uint16_t RH = 0, TEMP = 0;


      read_error += DHT22_Read_raw(sensor_DHT22, &Rh_byte1);
      read_error += DHT22_Read_raw(sensor_DHT22, &Rh_byte2);
      read_error += DHT22_Read_raw(sensor_DHT22, &Temp_byte1);
      read_error += DHT22_Read_raw(sensor_DHT22, &Temp_byte2);
      read_error += DHT22_Read_raw(sensor_DHT22, &SUM);

      if(read_error == 0)
      {
          SUM_temp = Rh_byte1 + Rh_byte2 + Temp_byte1 + Temp_byte2 ;

          if(SUM == SUM_temp)
          {

              TEMP = ((Temp_byte1<<8)|Temp_byte2);
              RH = ((Rh_byte1<<8)|Rh_byte2);

              sensor_DHT22->temperature = (float) (TEMP/10.0);
              sensor_DHT22->humidity = (float) (RH/10.0);
              return 0;
          }
          else
          {
              return 1;
          }
      }
      else
      {
          return 1;
      }
}
```

Figure 27 : Fonction DHT22_Read_Temp_hum()

Code LCD

Initialisation

Les fonctions de gestion du LCD ont été prises dans une bibliothèque disponible en ligne et développée par Loovee de Seeed technology Inc.

Cette bibliothèque est assez clé en mains et nous permet d'utiliser le LCD simplement grâce à des fonctions prédéfinies sans avoir à reconstruire les différentes trames en fonction des actions que l'on veut effectuer.

Pour initialiser le module nous avons besoin de deux structures, la première du type I2C_HandleTypeDef et la seconde du type rgb_lcd. Respectivement ces structures comportent les paramètres physiques de la ligne I2C ainsi que ceux de l'objet LCD. Ces deux paramètres sont passés à la fonction lcd_init.

Affichage

Les trois fonctions qui nous intéressent dans notre cas d'utilisation sont clearlcd(), lcd_print() et lcd_position(). Lcd_position() permet de préciser la place du curseur en passant en paramètre la position de la colonne puis de la ligne.

La partie affichage de température et Humidité ressemble donc à cela :

```
clearlcd();

lcd_position(&hi2c1,0,0);

memset(temp,0,sizeof(temp));
sprintf((char*)temp,"Hum: %.2f %c ",DHT22_1.humidity, 0x25);
lcd_print(&hi2c1,(char*)temp);

lcd_position(&hi2c1,0,1);

memset(temp,0,sizeof(temp));
sprintf((char*)temp,"Temp: %.2fC  ",DHT22_1.temperature);
lcd_print(&hi2c1,(char*)temp);

HAL_Delay(1000);
```

Figure 28 : Affichage LCD

Avant chaque affichage le LCD est effacé puis le reste s'effectue en trois temps :

Affichage de l'humidité, repositionnement à la deuxième ligne puis affichage de la température.

L'affichage des deux variables est similaire :

- La fonction memset() permet la mise à 0 du vecteur temp (uint8_t de longueur 16).
- Construction de la donnée avec la fonction sprintf() qui formate une chaîne dans une variable de sortie avec les paramètres souhaités (similaire à l'utilisation de printf)
- Lcd_print envoie une commande d'écriture sur le LCD avec la chaîne souhaitée.

Le Hal_Delay est une fonction HAL de base permettant d'obtenir un délai en milliseconde. Ici de 100 afin de fixer la fréquence de prise de données et affichage a 1 Hz.

Projet Web Server

Organisation fonctionnelle du projet

Schéma global

Le projet comportera deux parties, une partie capture avec une carte et ses capteurs qui serviront de source de donnée. La seconde partie sera la partie centrale qui aura pour but de faire converger toutes les données et de les afficher sur une interface web.

Le schéma suivant montre le principe de communication des différents éléments.

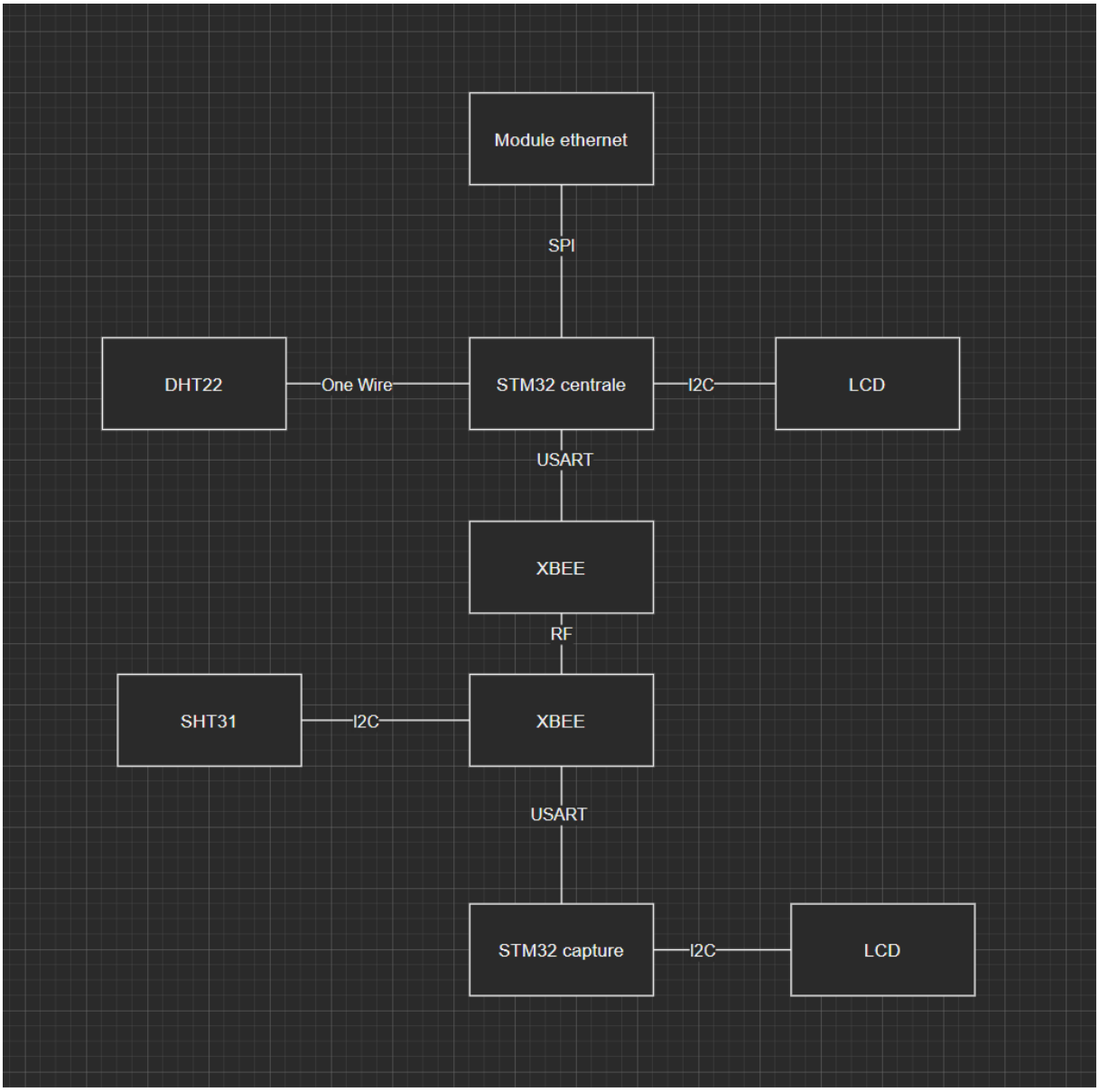


Figure 29 : Diagramme général

Dialogue entre cartes.

Afin de pouvoir travailler au mieux sur nos codes respectif nous avons établis dans un document partagé, le format d'échange des données entre cartes.

Capture vers centrale

La communication entre les cartes se fera via le protocole XBEE, configuré sur le canal 3333.

Pour mieux orchestrer une communication entre la centrale et un grand nombre de cartes nous avons décidé d'établir un modèle de trame spécifique. Ces dernières seront donc définies comme suit :

$**&i**=001**&t**=20.50**&h**=60.41\r\n

Caractère	Signification	Format	Exemple
i	Identifiant de carte	Entier 3 chiffres significatifs	&i=001
t	Température	Réel, partie entière 2 digit, partie décimale 2 digit.	&t=20.50
h	Humidité	Réel, partie entière 2 digit, partie décimale 2 digit.	&h=60.41
\r\n	Fin de données, retour chariot, saut de ligne	ASCII : 0x0D 0x0A	\r\n
Taille totale de la trame	25 octets		

Tableau 3 : Format trame envoi

Centrale vers capture

Cette partie n'a pas pu être implémentée par manque de temps mais un de nos objectif d'amélioration de ce projet est de dater les données afin de pouvoir faire un suivi de longue durée et mémoriser les différentes valeurs sur le temps.

Afin d'horodater les données envoyer il faut initialiser la base de temps des capteurs grâce au serveur la trame de réponse du serveur se fera donc suivant ce format :

"dd/mm/yyyy-hh:nn:ss"

Caractère	Signification	Format	Exemple
dd	Jour	Entier sur 2 octets entre 1 et 31	19

mm	Mois	Entier sur 2 octets entre 1 et 12	06
yyyy	Année	Entier sur 2 octets valeur min 2000	2000
hh	Heure	Entier sur 2 octets entre 0 et 24	21
nn	Minutes	Entier sur 2 octets entre 0 et 59	17
ss	Secondes	Entier sur 2 octets entre 0 et 59	00
Taille totale de la trame	19 octets		

Tableau 4 : Format trame retour

Introduction à la réalisation

Pour une meilleure compréhension et éviter les répétitions avec des fonctionnements déjà présentées dans la partie TP de base, nous développerons le trajet d'une donnée chronologiquement depuis la partie capteur jusqu'à l'affichage web en détaillant les étapes.

Structure du programme STM32

La structure du programme des deux cartes va rester la même, seront juste ajoutés une communication UART pour les deux modules et une seconde pour le module centrale afin de la faire dialoguer avec l'Arduino, la raison d'utilisation de l'Arduino sera détaillée plus bas.

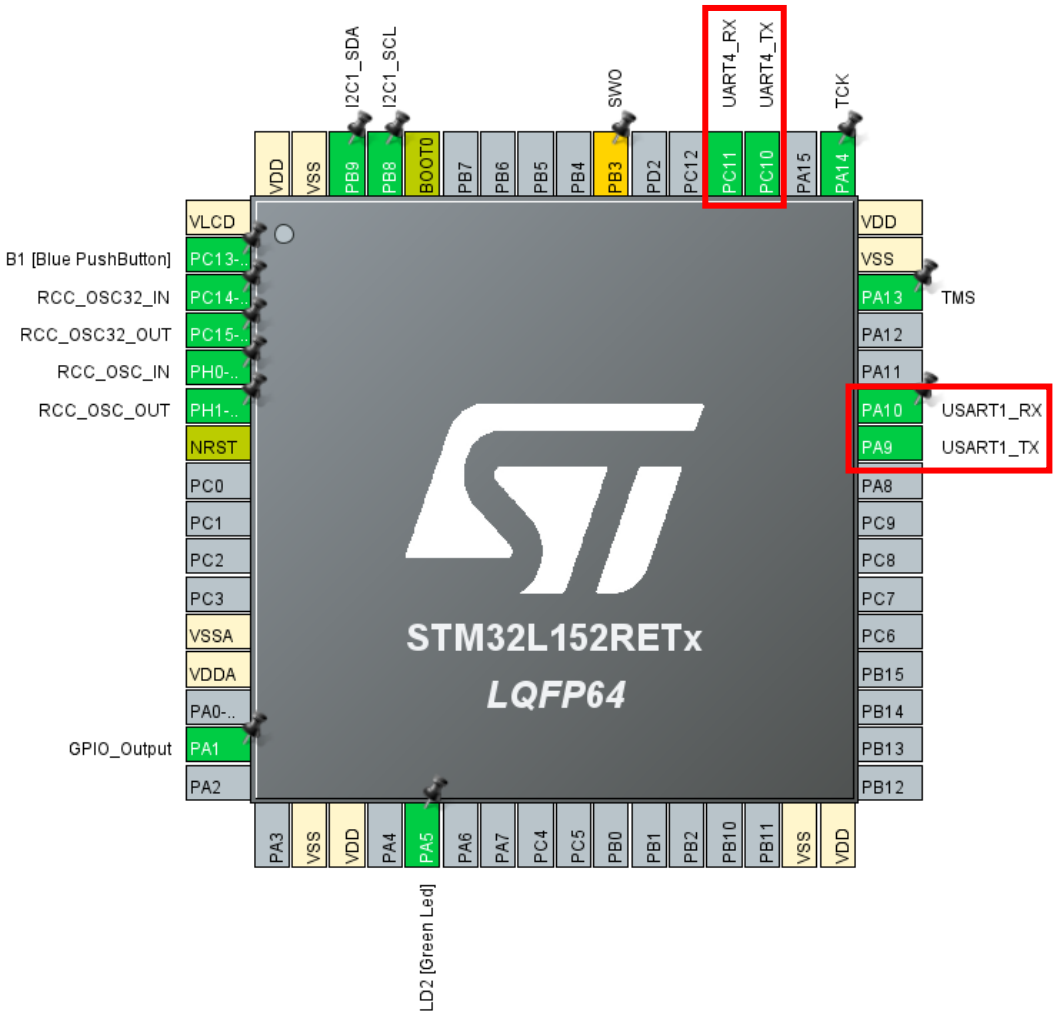


Figure 30 : Définition des pins

La figure ci-dessus présente la définition des pins pour les différents périphériques :

-USART1 : RX -> PA10 ; TX -> PA9 pour XBEE sur les deux cartes

-UART4 : RX -> PC11 ; TX -> PC10 pour Arduino sur la carte centrale

Envois partie capteur

Collecte et formatage

La carte capture va collecter les données, les formater et les envoyer via le module XBEE préalablement configuré avec l logiciel XCTU sur le canal 3333.

Comme précisé dans la partie organisation fonctionnelle du BE nous avons établis un format de trame.

La seconde tache après la collecte sera donc le formatage des données.

```
char send_buffer[100];

memset(send_buffer,0, sizeof(send_buffer));
strcat((char *)send_buffer,"$");


memset(temp,0, sizeof(temp));
sprintf((char *)temp,"&i=%s", device_ID);
strcat((char *)send_buffer,(char *)temp);


memset(temp,0, sizeof(temp));
sprintf((char *)temp,"&t=%.2f", temperature);
strcat((char *)send_buffer,(char *)temp);


memset(temp,0, sizeof(temp));
sprintf((char *)temp,"&h=%.2f", humidity);
strcat((char *)send_buffer,(char *)temp);
strcat((char *)send_buffer,(char *)"\r\n");
```

Figure 31 : Formatage des données d'envoi

Nous utilisons donc les fonctions de la bibliothèque string.h qui nous permet la gestion des chaines de caractère avec une grande facilité :

- void *memset(void * pointer, int value, size_t count) ; Affecte à l'adresse « pointer », sur « count » octets, la valeur « value »
- char *strcat(char * dest, const char * src) ; Ajoute le contenus du pointeur mémoire « src » (jusqu'au caractère '\0') à l'emplacement « dest ».

Envois des données

Une fois les données formatées dans la variable send_buffer, elles peuvent être envoyée au module XBEE relié en UART avec le STM32

```
HAL_UART_Transmit(&huart4,(uint8_t*)send_buffer, strlen((char *)send_buffer),100);
HAL_Delay(1000);
```

Figure 32 : Formatage des données d'envoi

La librairie HAL nous permet d'exploiter la liaison UART assez simplement avec des fonctions de haut niveau directement implémentées dans le projet, nous utilisons donc la fonction HAL_UART_Transmit() pour envoyer les données :

- HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout) ; Où les « Size » données contenues à l'adresse « pData » seront envoyées via l'uart « HandleTypeDef » avec un délais de sortie de « Timeout » microsecondes.

Réception partie centrale

Récupération des données

Le code est organisé par une machine à état permettant un meilleur orchestrèrent des différentes tâches que doit effectuer le module centrale.

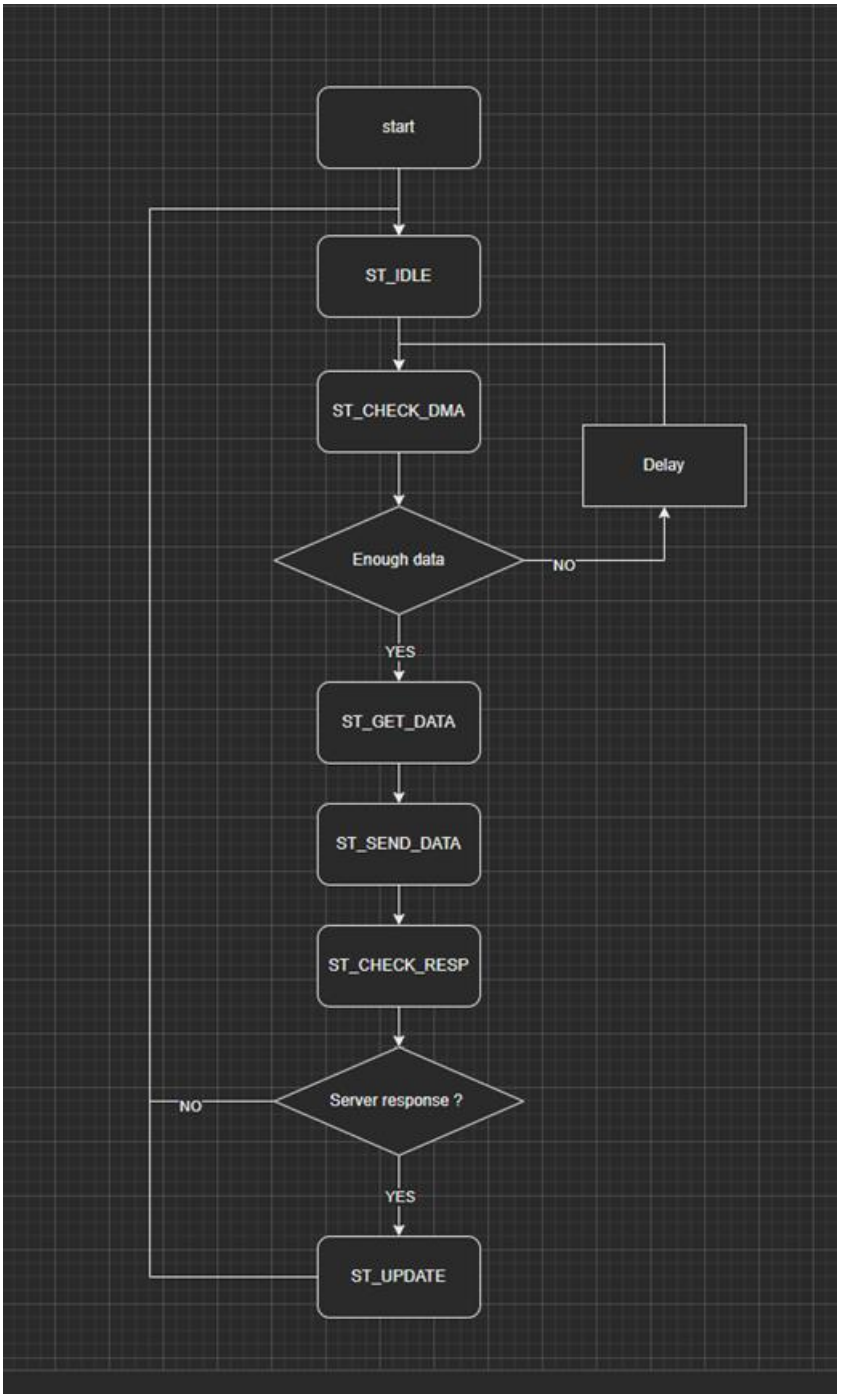


Figure 33 : Machine à états

Les données des modules capture n'arrivant pas à intervalle régulier, j'ai décidé d'implémenter un DMA sur la liaison RX de l'uart XBEE afin de pouvoir libérer du temps de traitement processeur et les manipulations de données. Tous les octets récupérés par le XBEE vont être inscrite à l'adresse mémoire spécifiée au DMA et la réception des données va pouvoir se faire en parallèle des autres taches sans manquer le moindre octet.

Les trames XBEE ont été définies avec un format fixe afin de simplifier leur utilisation. Des constantes vont alors pouvoir être utilisées pour définir la taille du buffer DMA, du nombre de dispositif connectés.

Les packages HAL implémentés dans le projet avec CubeIDE permettent de gérer plus facilement les interruptions et le DMA, pour le mettre en plus il suffit de le spécifier avec l'interface graphique et d'appeler la fonction HAL_Receive_DMA pour activer le procéder

```
#define DMA_SIZE 25//100

static uint8_t DMA_buff[DMA_SIZE]                    = "";

HAL_UART_Receive_DMA(&huart4, DMA_buff, DMA_SIZE);
/*Enable IRQ on UART1*/
```

Figure 34 : initialisation DMA

Formatage et envois

A son tour le module centrale va récupérer ses informations de température et humidité grâce à ses capteurs, les formater et les envoyer via transmission UART à l'Arduino de la même manière que pour le XBEE mais sur l'UART dédiée

Module Ethernet et Arduino

Le module Ethernet est un Arduino Ethernet shield 2. Ce dispositif s'adapte aux cartes format Arduino Uno et est compatible physiquement avec les carets STM32 Nucléo.

C'est un module basé sur le chipset Ethernet Wiznet W5500 qui propose une communication SPI ainsi qu'un port pour carte SD

Nativement les cartes STM32 Nucléo ne sont pas compatibles avec ce module, car ne dispose pas du connecteur Arduino ICPS nécessaire à la communication SPI.

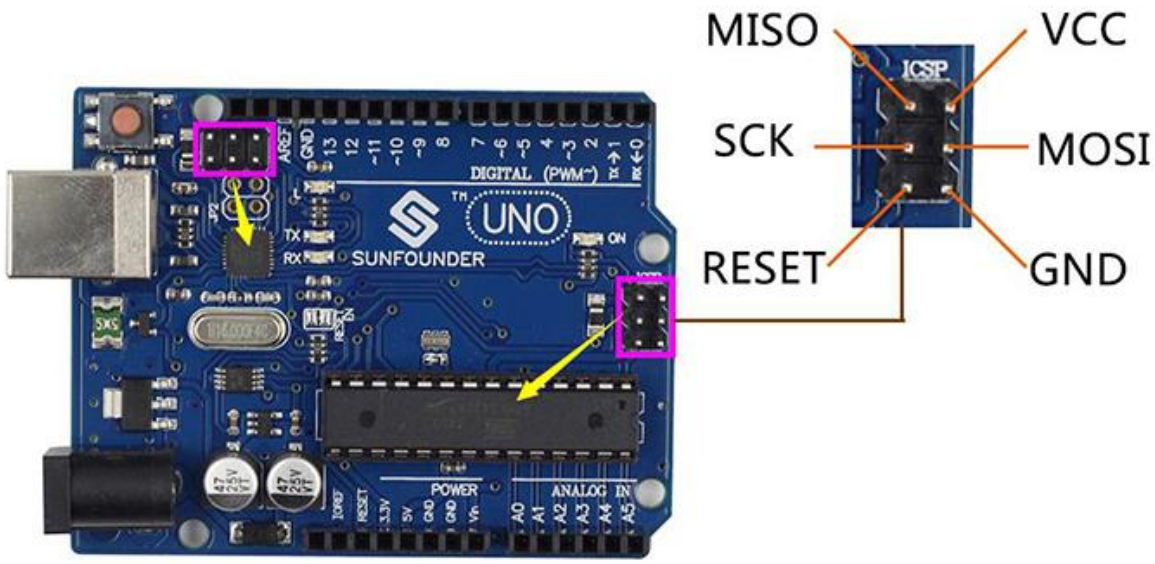


Figure 35 : ICPS Arduino

J'ai donc dû souder les fils nécessaires pour tenter d'établir la communication. Malheureusement, le manque de documentation, la complexité du portage et le temps ont fait que je n'ai pas réussi à adapter ce module. Les seuls exemples disponibles d'une adaptation réussite ont été effectués sur des cartes F401RE ou F103RE, j'avais à ma disposition une carte L152RE.

La solution à donc été de déporter le module sur une carte Arduino Uno et de relier UART cette Dernière à la carte Nucléo. La difficulté majeure a été de gérer l'envois périodique des données et l'accessibilité du serveur afin de ne pas perdre de données et d'éviter des temps morts lors d'une consultation via un navigateur WEB.

La figure ci-dessous présente donc le raccordement des différentes parties du module centrale :

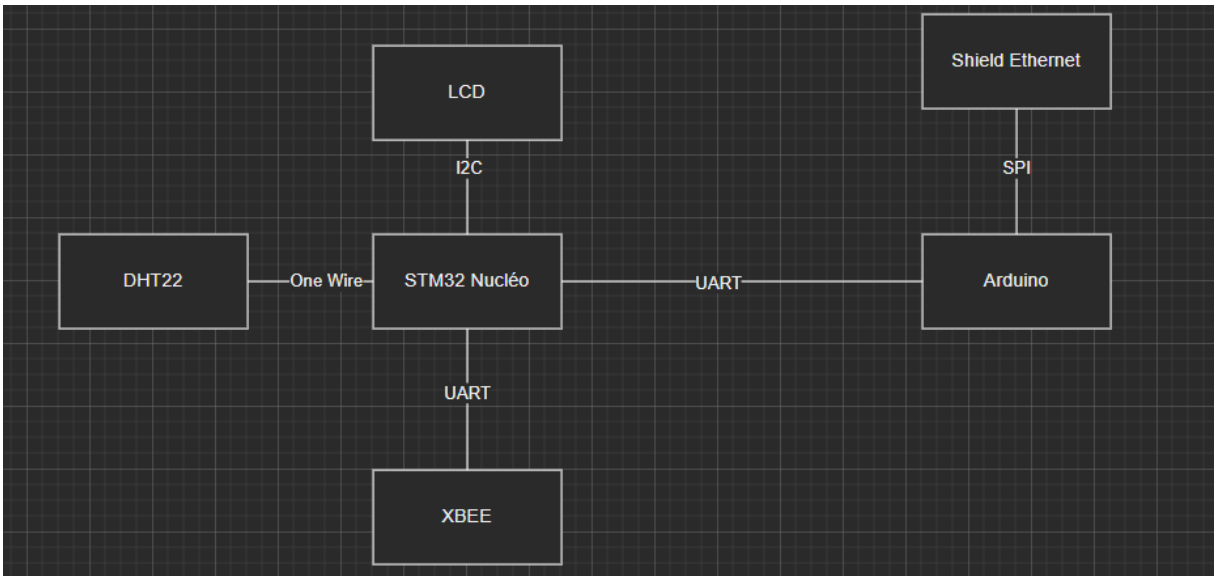


Figure 36 : Diagramme centrale

Structure du programme Arduino

La difficulté d'implémenter un serveur collectant des données en parallèle avec des cartes mono-cœur et à programmation séquentielle est qu'il faut optimiser les temps d'exécution des différentes taches afin de revenir le plus vite possible en attente si un client se connecte. Le programme suit donc la logique suivante :

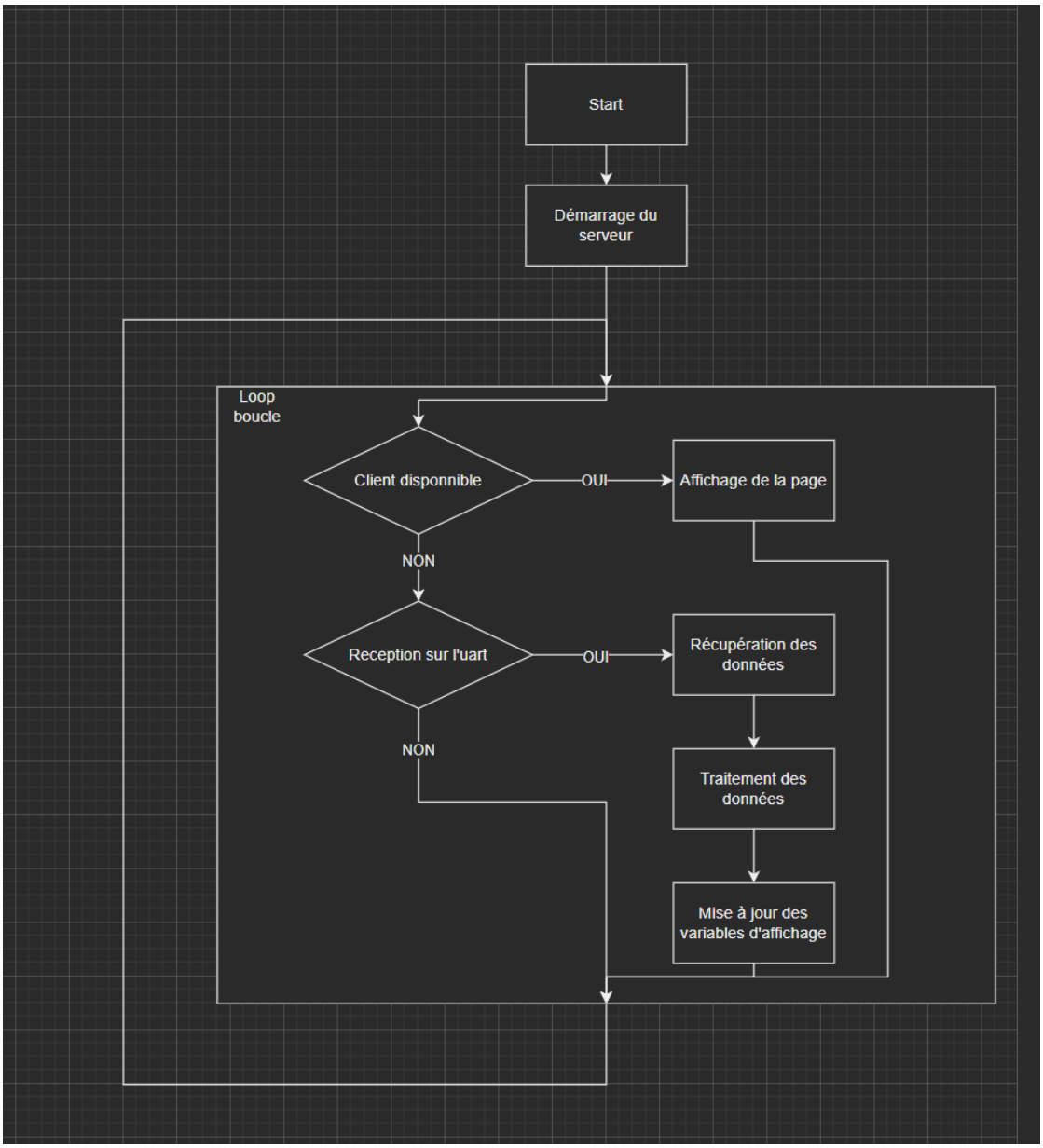


Figure 37 : Routine Arduino

L'Arduino est en attente soit de données sur la liaison série, soit d'une connexion par un client voulant consulter les données.

Dans le premier cas l'Arduino va récupérer et extraire les données pour mettre à jours les variables correspondant à chaque appareil.

```
struct Device {
  char id[4];
  char temp[6];
  char hum[6];
  unsigned long last_time;
};
Device all_device[NB_DEVICE];
```

Figure 38 : Données modules

L'Arduino interprète chaque module comme une structure de données rassemblées en un tableau de structures qui permet de gérer simplement chaque appareil et ses données en les identifiant avec un champs ID propre à chaque appareil.

Pour pallier au fait que les données n'ont pas pu être datées nous avons rajouté une petite variable à chaque dispositif qui spécifie quand la dernière données de chaque appareil a été reçue, c'est le champs « last_time »

Lors de la consultation, tous les dispositifs vont être affichés, si aucune donnée n'est présente alors il y aura un champs « none » et pour le reste température, humidité et temps Mcu de la dernière donnée perçue. Pour la comparaison le temps actuel de l'Arduino est affiché constamment. Ce temps correspond au temps de fonctionnement du microcontrôleur de l'Arduino.

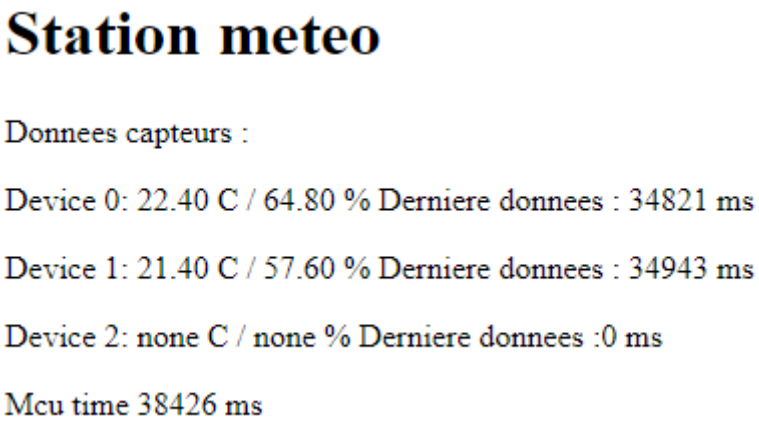


Figure 39 : Page HTML avec les données

En vue des capacités de l'Arduino à générer une page web il n'est pas possible de gérer le CSS sur ce genre d'affichage mais simplement des informations de base comme le montre la figure ci-dessus, ce qui est largement suffisant pour avoir un aperçu des données souhaitées.

Conclusion

Ce projet nous a permis de manier l'environnement CubeIde ainsi que d'approcher d'un point de appliqué la programmation de microcontrôleur pour un projet.

Ainsi nous avons pu voir différents protocoles de communication et mettre en œuvre les différentes fonctionnalités que propose les cartes NUCLEO.

Le projet initial à pus aboutir et nous a permis de proposer une démonstration fonctionnelle de l'idée de départ. La partie Ethernet n'étant pas compatible il nous a fallu l'adapter avec un Arduino afin de mener à bien le projet.

En termes d'amélioration nous avons soulevé plusieurs points :

- Remplacement de la carte STM32 L152RE par une carte compatible Ethernet.
- Ajout d'un module RTC pour horodatage des trames de données.
- Remplacement du module Ethernet par un module Wifi afin de faire de la carte une passerelle vers un serveur distant plutôt qu'un serveur pour mieux gérer le stockage des données et l'affichage.