Réalisation systèmes

Parcours : M1 SME

Compte Rendu

BE : STATION METEO

Binôme :

- Anis Mourad BENNACER
- Hichem Ibrahim Salah

Responsable TP :

* Thierry PERISSE

2022/2023

Introduction :

Durant le 2ème semestre nous avons été amenés à réaliser un projet dans le cadre d'un bureau d'étude, ce projet a pour but d'améliorer notre façon d'appréhender une problématique liée à notre domaine d'étude et trouver des solutions grâce à diverse documentations tout en travaillant de manière autonome. Pour ce bureau d'étude, on doit réaliser une station météo à l'aide de la carte STM32L152RE, deux capteurs (Dht22 & BMP280) et deux modules de communication Xbee pro. La réalisation de notre projet va se deviser en deux parties.

La première partie comporte les TP de base qui ont pour but de familiariser avec la carte, nos deux capteurs ainsi que actionneurs dont on dispose.

La deuxième partie est la partie réalisation, où on doit assembler les différents composants et créer un système qui doit remplir une certaine tache.

Matériels utilisés :

Le STM32L152RE de la famille STM32 de STMicroelectronics. Il fait partie de la série STM32L1 est un microcontrôleur basse consommation avec un processeur ARM Cortex-M3, qui est spécifiquement conçue pour les applications nécessitant une faible consommation d'énergie. Il offre diverses interfaces de communication, notamment l'I2C, l'UART, le SPI et le CAN. Ces interfaces permettent de communiquer avec d'autres périphériques et de connecter le microcontrôleur à des capteurs, des actionneurs ou d'autres systèmes.

Il propose différents modes de gestion de l'énergie, tels que les modes veille, stop, standby et shutdown, permettant d'optimiser la consommation d'énergie en fonction des besoins de l'application, ce qu'il le rend idéal pour les applications où la gestion de l'énergie est essentielle.

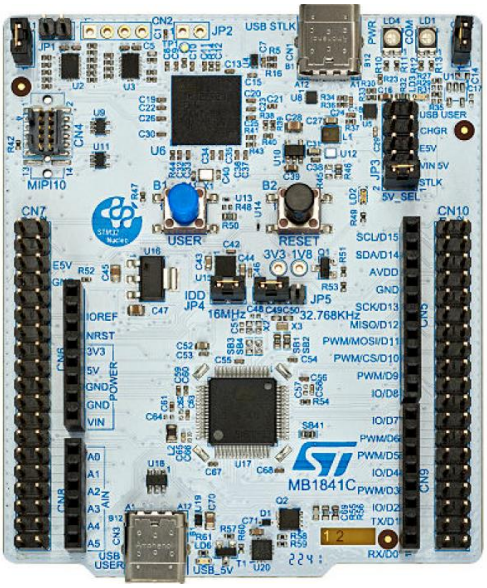

Figure 1 STM32 NUCLEO-L152RE

Le capteur BMP280 est un capteur de pression atmosphérique et de température précis et compact. Il est largement utilisé dans diverses applications telles que la météorologie. Il mesure la pression atmosphérique et la température ambiante avec une grande précision. Le capteur BMP280 communique via une interface de bus série, généralement I2C ou SPI, ce qui le rend compatible avec une large gamme de microcontrôleurs et de plateformes de développement. Il dispose également d'une faible consommation d'énergie, ce qui en fait un choix populaire pour les applications alimentées par batterie.

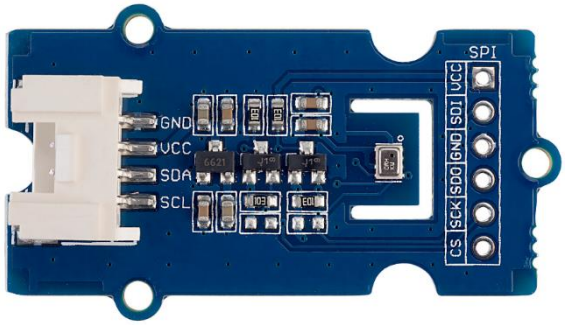

Figure 2 Capteur BMP280

Le capteur DHT22 est un capteur d'humidité et de température numérique utilisé dans de nombreuses applications. Il est souvent utilisé dans les dispositifs météorologiques.
Il est doté d'un élément de capteur à semi-conducteur qui mesure à la fois l'humidité relative et la température ambiante. Il est conçu pour offrir une grande précision et une plage de mesure étendue. Le capteur DHT22 communique avec d'autres appareils via un signal numérique. Il utilise une interface à un seul fil (One-Wire) pour transmettre les données de température et d'humidité. Cela facilite son intégration avec une variété de microcontrôleurs et de plateformes de développement.

L'un des avantages de ce capteur est sa fiabilité et sa stabilité à long terme. Il est conçu pour compenser automatiquement les variations de température et d'humidité, ce qui permet de maintenir des mesures précises et cohérentes sur une période prolongée.
Il est généralement alimenté par une tension de 3,3 V et consomme très peu d'énergie, ce qui en fait un choix idéal pour les projets à faible consommation d'énergie.

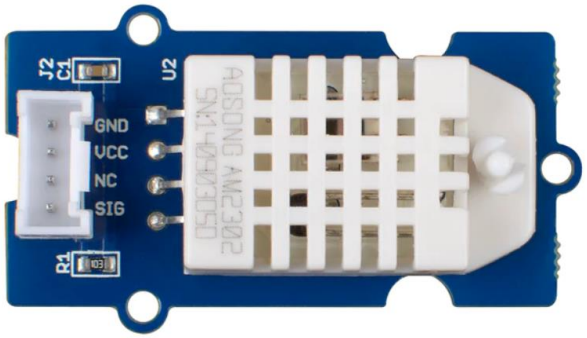

Figure 3 Capteur de température et d'humidité DHT22

L'écran LCD I2C se compose d'un écran à cristaux liquides et d'un module d'interface I2C intégré. Le module d'interface I2C permet de contrôler l'affichage et de transférer les données via le bus I2C, simplifiant ainsi la connexion avec le microcontrôleur.

Au lieu d'utiliser plusieurs broches pour le contrôle et la transmission des données, l'écran LCD I2C utilise seulement deux broches : SDA (Serial Data Line) et SCL (Serial Clock Line), qui sont utilisées pour la communication I2C.

Figure 4 16x2 LCD I2C (White on blue)

Le module Base Shield est une carte d'extension conçue pour faciliter le prototypage et l'utilisation de modules et de capteurs avec des microcontrôleurs ou des cartes de développement. Il est généralement associé à des plates-formes telles que STM32, permettant une connexion facile et rapide de différents composants électroniques.

Le module Base Shield fournit une interface standardisée et un agencement de broches préconfiguré qui simplifie grandement le câblage et l'interconnexion des composants, il peut également comporter des connecteurs pour des broches d'alimentation (5V, 3.3V, GND) permettant d'alimenter les différents composants connectés.

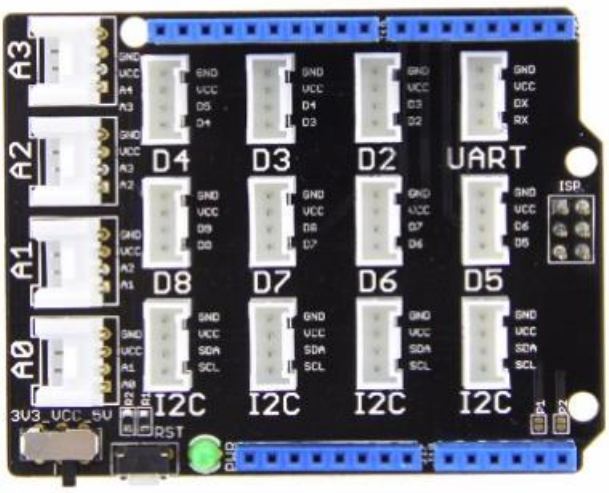

Figure 5 Base Shield V2

Le module XBee Pro est une solution intégrée offrant une connectivité de point de terminaison sans fil aux périphériques. Ce module utilise le protocole de réseau IEEE 802.15.4 pour une mise en réseau rapide de point à multipoint ou entre homologues. Il est conçu pour les applications à haut débit nécessitant une faible latence et un délai de communication prévisible. Ce module RF a été conçu pour répondre aux normes IEEE 802.15.4 et pour répondre aux besoins uniques des réseaux de capteurs sans fil à faible coût et à faible consommation. Le module nécessite une alimentation minimale et fournit une transmission fiable des données entre les périphériques. Le module fonctionne dans la bande de fréquence ISM 2,4 GHz.

Figure 6 Module Xbee Pro

Bureau d'étude :

Configuration sur STM32CubeIDE :

A présent nous allons configurer les broches des cartes nécessaires pour tous les composants.

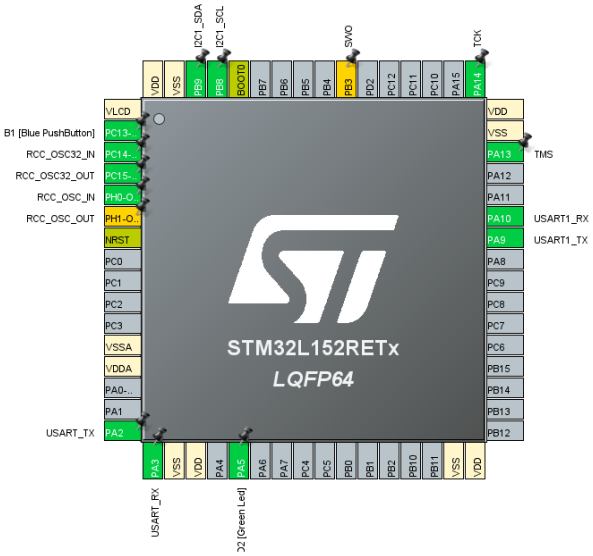

Figure 7 Configuration microprocesseur carte 1

Sur la figure 7, on voit la configuration du microprocesseur de la carte 1 qui va gérer un afficheur LCD, un module Xbee Pro et le capteur BMP280.

Pour communiquer avec le LCD et le capteur, on a activé les pins PB8 et PB9 correspondants aux SDA et SCL du module I2C1.

Pour l'émission et réception via Xbee, on a activé les pins PA9 et PA10 correspondants au Tx et Rx du USART1 avec une vitesse de 19200bauds, 8bits de données, 1bit de stop et 0bit de parité.

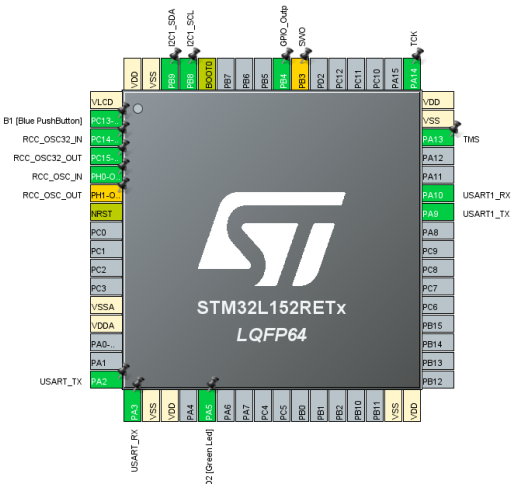

Figure 8 Configuration microprocesseur carte 2

Sur la figure 8, on voit la configuration du microprocesseur de la carte 2 qui va gérer un afficheur LCD, un module Xbee Pro et le capteur DHT22.

La configuration est la même que celle de la première carte mais avec le pin PB4 correspondant au pin D5 du shield en plus. Ce pin sert à communiquer le capteur DHT22.

Schéma de Câblage :

Grace au logiciel Fritzing, nous avons réalisé les schémas de câblage suivants :

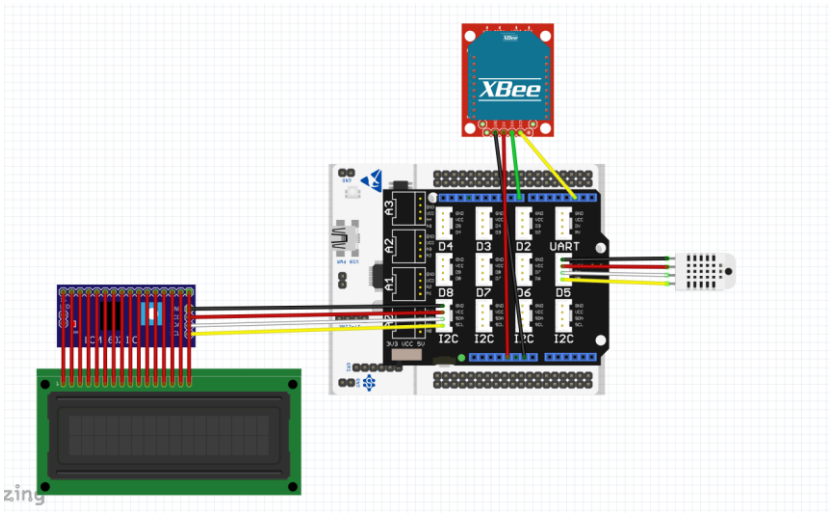

Figure 9 câblage carte DHT22

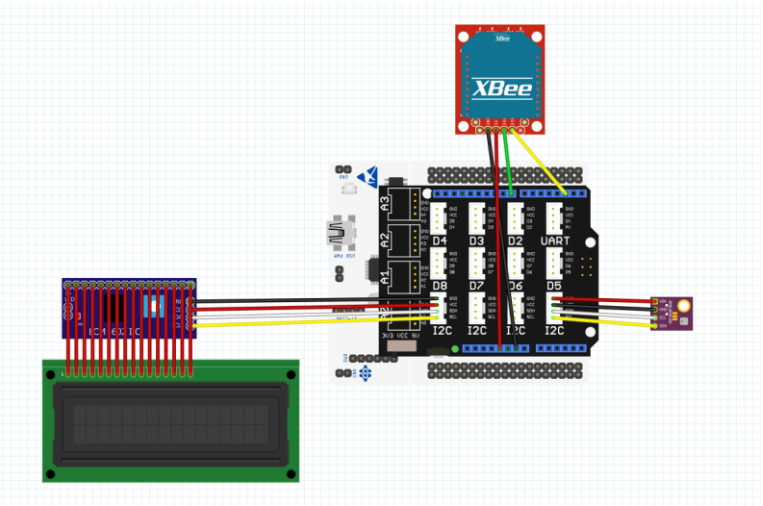

Figure 10 câblage carte BMP280

Programmation :

Passons à la partie programmation de notre projet : La fonction suivante permet de lire une donnée à partir du capteur DHT22 :

```
void Read_data (uint8_t *data) // Fonction qui lis des données à partir du DHT22
  {
    int i, k;
    for (i=0;i<8;i++)
    {
        if (HAL_GPIO_ReadPin (GPIOB, GPIO_PIN_4) == GPIO_PIN_RESET)
        {
            (*data)&= ~(1<<(7-i)); //data bit is 0
            while(!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4)));
            DWT_Delay_us(40);
        }
        else                        //data bit is 1
        {
            (*data)|= (1<<(7-i));
            for (k=0;k<1000;k++)
            {
                if (HAL_GPIO_ReadPin (GPIOB, GPIO_PIN_4) == GPIO_PIN_RESET)
                  {
                    break;
                  }
            }
            while(!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4)));
            DWT_Delay_us(40);
        }
     }
  }
```

Puis, dans la boucle while on met le code suivant pour récupérer toutes les mesures du DHT22

```
//Communication avec DHT22
HAL_Delay(1000);// Effectuer des mesures chaque seconde
Set_Pin_Output(GPIOB, GPIO_PIN_4); //Configuration du pin en sortie
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET); //Mise à 0 du pin
DWT_Delay_us(1000); //Envoi de la premiere partie du signal de commande
HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET); // Mise a 1 du pin
DWT_Delay_us(30); //Envoi de la deuxieme partie du signal de commande
Set_Pin_Input(GPIOB, GPIO_PIN_4); //Configuration du pin en entrée

// Lecture des données
while(!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4)));
for (k=0;k<1000;k++)
{
    if (HAL_GPIO_ReadPin (GPIOB, GPIO_PIN_4) == GPIO_PIN_RESET)
    {
      break;
    }
}

while(!(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4)));
DWT_Delay_us(40);

Read_data(&dataH1); //Lecture du premier octet d'humidité
Read_data(&dataH2); //Lecture du deuxieme octet d'humidité
Read_data(&dataT1); //Lecture du premier octet de temperature
Read_data(&dataT2); //Lecture du deuxieme octet de temperature
Read_data(&SUM);  //Lecture d'octet de sécurité

check = dataH1 + dataH2 + dataT1 + dataT2;

RH = (dataH1<<8) | dataH2;  //Concatenation des octets d'humidité
TEMP = (dataT1<<8) | dataT2; //Concatenation des octets de temperature

Humidite = (int) RH / 10.0; //Les valeurs recues sont egales à fois les valeurs reelles
Temperature = (int) TEMP / 10.0;

HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET); //Preparation pour lecture suivante
```

Pareil pour le capteur BMP280, ce code permet de récupérer les données

```
bmp280_init(&bmp280, &bmp280.params);
while (1)
{
  /* USER CODE END WHILE */
      HAL_Delay(1000);
   while (!bmp280_read_float(&bmp280, &temperature, &pressure, &humidity)) {
            size = sprintf((char *)Data,"Temperature/pressure reading failed\n\r");
            HAL_UART_Transmit(&huart1, Data, size, 1000);
            HAL_Delay(2000);
        }
   pressure=pressure/100.;
```

Et puis pour la partie affichage : chaque carte envoi ses données vers l'autre en passant par USART1 grâce au module Xbee Pro

```
//Transmission des données en Xbee à travers l'UART
  size=sprintf((char *)Data,"%.1f", Humidite);
  HAL_UART_Transmit(&huart1, Data, 4,1000);

  size=  sprintf((char *)Data1, "%.1f", Temperature); //
  HAL_UART_Transmit(&huart1, Data1, 4,1000);



 size = sprintf((char *)Data,"%.1f", temperature);
 HAL_UART_Transmit(&huart1, Data, 4,1000);
HAL_Delay(100);

 size = sprintf((char *)Data,"%.f", pressure);
 if (size==3) {
   HAL_UART_Transmit(&huart1,"0",1,1000);
   HAL_UART_Transmit(&huart1, Data, 3,1000);
 }
 else {
     HAL_UART_Transmit(&huart1, Data, 4,1000);

 }
```

Et chaque carte reçoit les données de l'autre pour l'afficher sur son LCD, cela se fait sur interruption (Grace à la fonction HAL_UART_Receive_DMA qui exécute l'interruption à chaque fin de réception et se relance toute seule après)

Comme chaque carte envoie deux données à la fois, on a mis un indice i qui permet de réceptionner une donnée à la fois et l'afficher correctement sur le LCD.

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
 { if(i==1)  {
             lcd_position(&hi2c1,11,0);
         lcd_print(&hi2c1,(char*)Rx1);
            lcd_position(&hi2c1,15,0);
            lcd_print(&hi2c1,"C");
         i=2;
             }
     else if(i==2) {  if (Rx1[0]==9) {
                     Rx1[4]=Rx1[3];
                     Rx1[3]=Rx1[2];
                     Rx1[1]=Rx1[0];
                     Rx1[0]=0; }
                     lcd_position(&hi2c1,9,1);
                     lcd_print(&hi2c1,(char*)Rx1);
                        lcd_position(&hi2c1,13,1);
                        lcd_print(&hi2c1,"hPa");
                     i=1;
             }
     }
```

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{ if(i==1)  {
             lcd_position(&hi2c1,11,0);
             lcd_print(&hi2c1,(char*)Recep);
             lcd_position(&hi2c1,15,0);
             lcd_print(&hi2c1,"%");
         i=2;
             }
     else if(i==2) {

                     lcd_position(&hi2c1,11,1);
                     lcd_print(&hi2c1,(char*)Recep);

                     lcd_position(&hi2c1,15,1);
                     lcd_print(&hi2c1,"C");
                     i=1;
             }

     }
```

Réalisation et manipulation :

Après la configuration des cartes et la compilation du code, on configure les modules Xbee Pro avec le logiciel XCTU pour leur donner les mêmes paramètres de l'USART1.

On televerse ce dernier sur nos cartes et on exécute les programmes, voici une photo du système en marche :

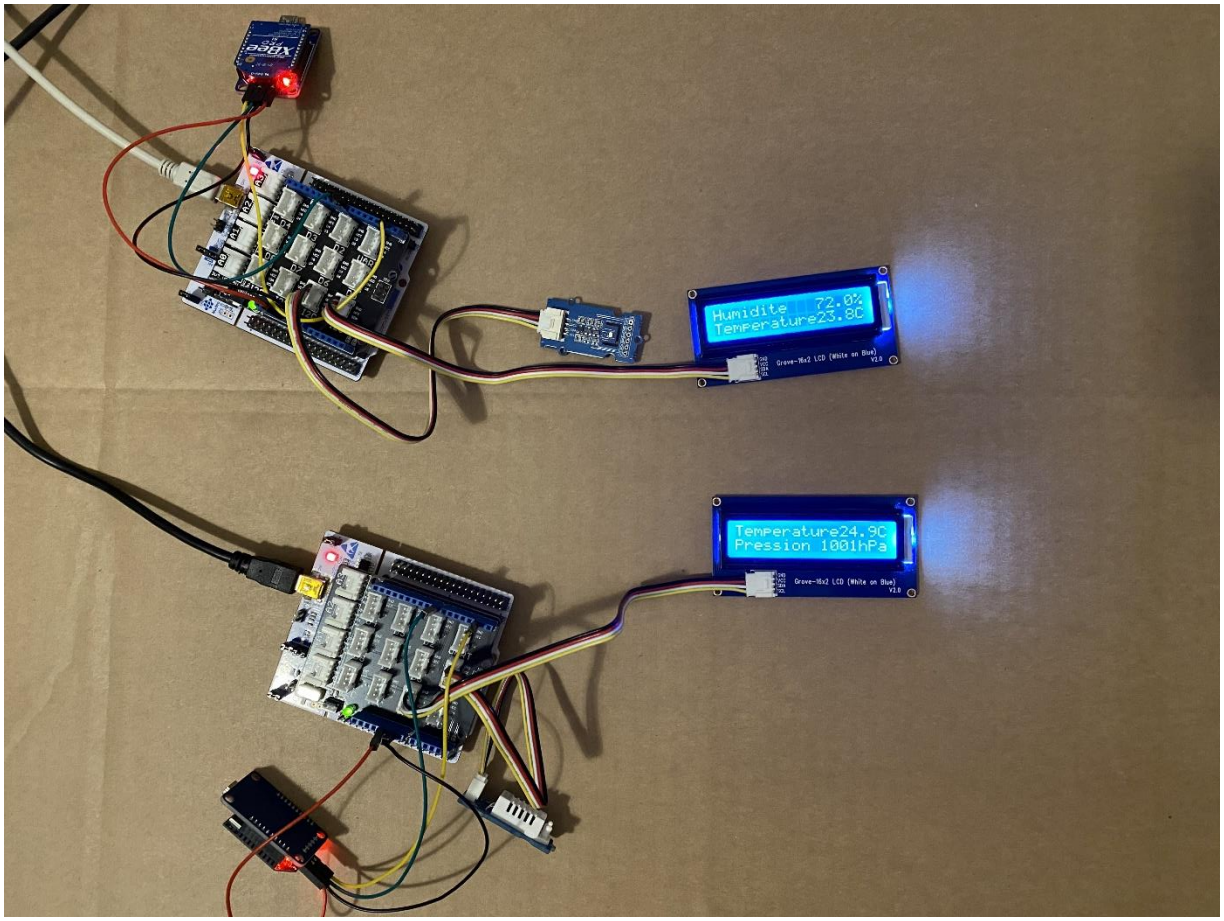

Figure 11 Réalisation du BE

La première carte (En haut) reliée au capteur BMP280 de pression et température affiche humidité et température donc elle affiche les données du DHT22 et de même pour la deuxième carte (En bas), Elle est reliée ay DHT22 mais elle affiche température et pression, donc les données du BMP280,

On a bien un affichage croisé.

Consommation :

En branchant en série une carte en marche avec un résistance shunt (1ohm), le tout relié à un générateur 5V, et on bronche un oscilloscope aux bornes de la résistance, on voit la figure suivante

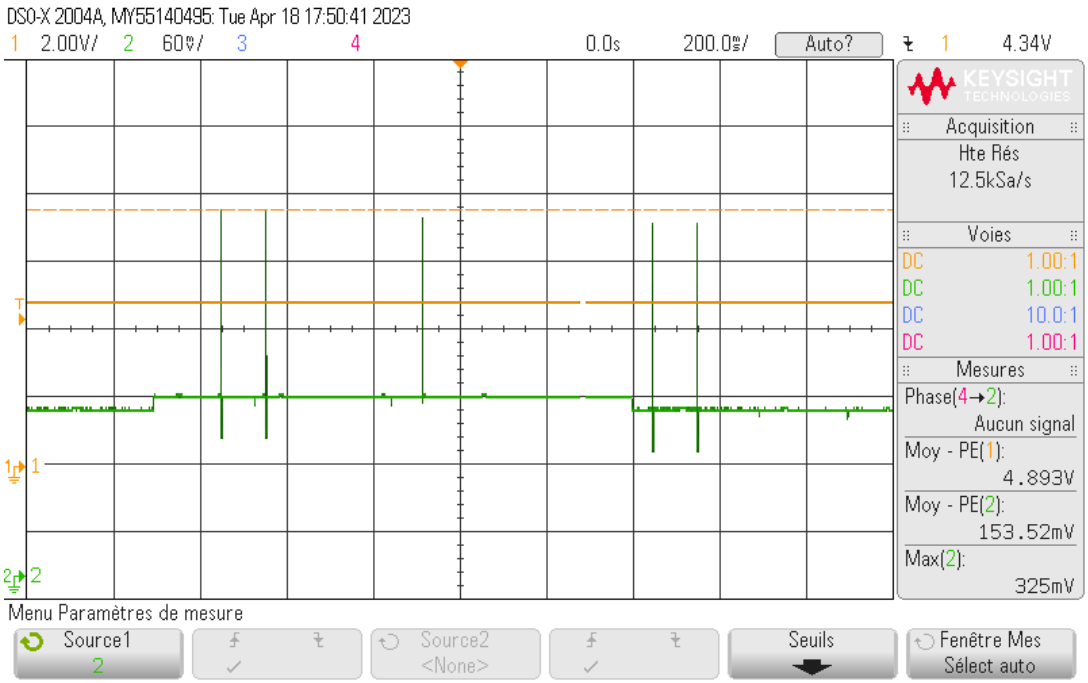

Figure 12 Tension de résistance en oscilloscope

La ligne verte est la tension de la résistance et comme R=1ohm alors la tension correspond au courant sortant de la carte (donc le courant consommée par la carte en marche)

Avec des valeurs de pic de 325 milliAmpères (lors de transmission avec Xbee) et un courant moyen de 153 milliAmpères, une carte consomme 153 mAh et donc 3672 mAh en 24h, ce qui correspond à une batterie de smartphone consommée en une journée.

Conclusion :

A la fin de ce BE, nous nous sommes familiarisés avec l'environnement de développement cubeIDE et avec la configuration de microprocesseur STM32 et de cartes NUCLEO.

Nous avons aussi appris à analyser des datasheets de différents capteurs pour mettre en place des codes pour lire ces capteurs et récupérer leurs données et conditionner leur affichage.