



RAPPORT  
TPs de Base

Réalisation Systèmes

Encadré par  
PERISSE Thierry

MBOUNGOU Frangely  
BA WAZIR Ahmed

# I - Introduction

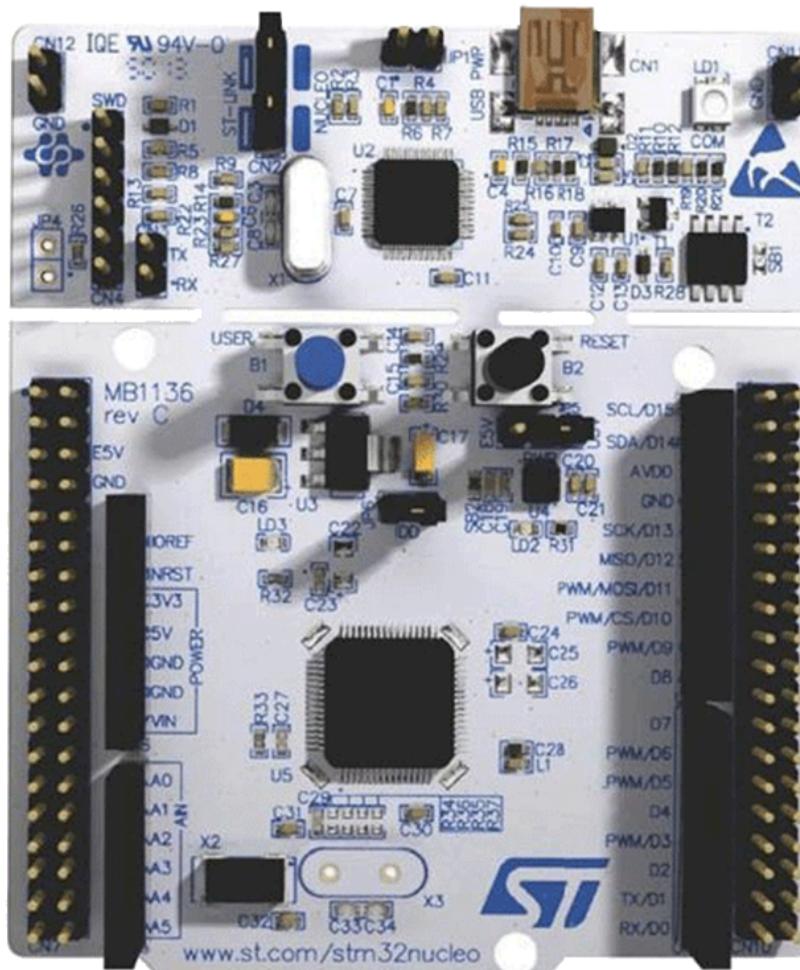
Le but de ces TP de base est de se soumettre à l'étude et la mise en oeuvre de systèmes adaptés aux exigences des systèmes embarqués. Pour cela, on a donc été amené à travailler sur la carte de développement NUCLEO embarquant un microcontrôleur ARM Cortex STM32 sous environnement de développement STM32CubeIDE.

La réalisation de notre BE c'est donc fait autour d'une lecture de données que renvoie un capteur afin de pouvoir les interpréter et les retranscrire tout en expliquant comment les différents matériels communiquent entre eux. Les capteurs à notre charge étaient le DHT22 et le SHT31.

# II - Bureau d'Études

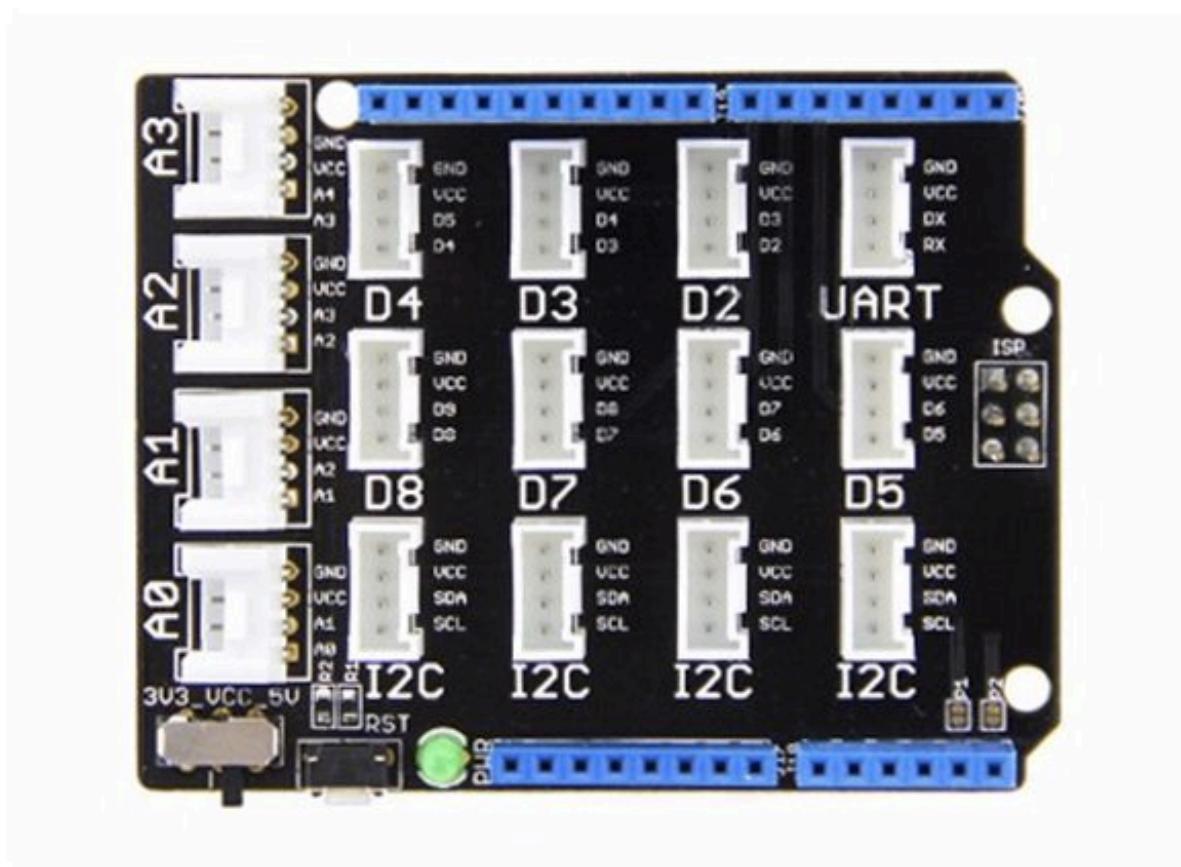
## A- Matériels

Carte NUCLEO STM32-L476RG



Comme évoqué précédemment, la carte NUCLEO L476RG est une carte de développement faisant partie de la famille STM32 que l'on retrouve notamment dans des procédés à faible puissance ou encore à faible consommation d'énergie. Elle possède une large gamme de périphériques de communication comme : USB, CAN, I2C, SPI, UART. Elle intègre une interface de programmation ST-Link et un processeur 32 bits Cortex-M4 ce qui lui permet d'offrir une large gamme de fonctionnalités et de ressources. Dans notre cas elle a été la plateforme de développement pour la mise en application de nos capteurs.

## MODULE GROVE BASE SHIELD



Le module Grove Base Shield est un module d'extension permettant de raccorder de façon efficace les différents modules électroniques tels que les capteurs et

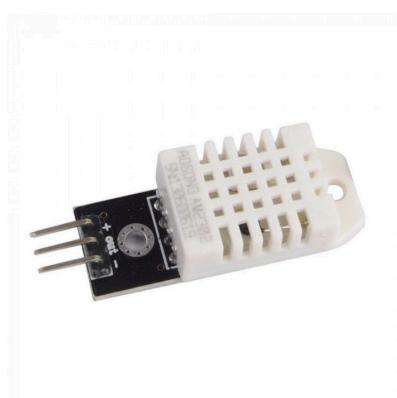
l'écran LCD au travers d'entrées/sorties préalablement configurées suivant la communication que l'on souhaite mettre en place.

## AFFICHEUR LCD I2C



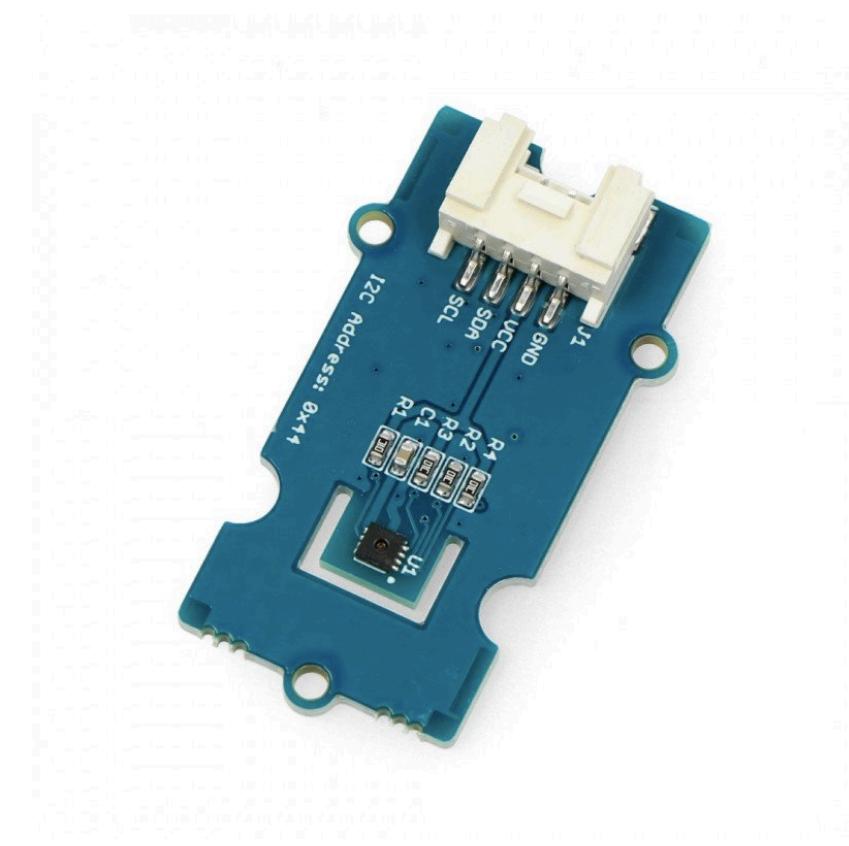
L'afficheur LCD est donc l'interface qui va nous permettre de retranscrire les informations que l'on souhaite recevoir. Il utilise le protocole de communication I2C (Inter-Integrated Circuit) qui permet la communication entre plusieurs périphériques à l'aide d'un bus à deux fils (SCL et SDA). Ainsi notre afficheur convertit des données séries I2C en données parallèle pour l'affichage sur l'écran.

## CAPTEUR DHT22



Le capteur DHT22 est un capteur numérique d'humidité capacitif et un thermistor qui relève et mesure la température et l'humidité avec une précision de +/- 0,5°C pour la température et +/- 2% pour l'humidité. Il utilise une interface à un seul fil « One-Wire » comme protocole de communication.

## CAPTEUR SHT31

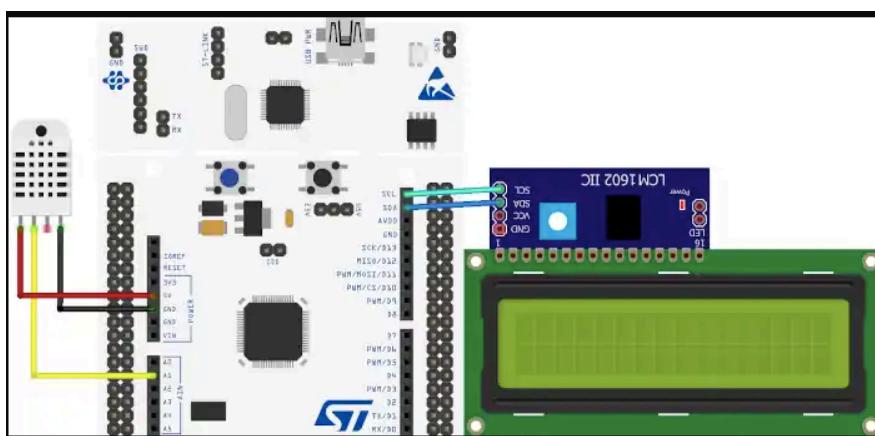


Le capteur SHT31 est lui aussi un capteur numérique qui relève et mesure la température et l'humidité avec une précision de +/- 0,3°C pour la température et +/- 2% pour l'humidité. Sa communication se fait par I2C.

## B- Réalisation

### DHT22

#### Schéma de câblage



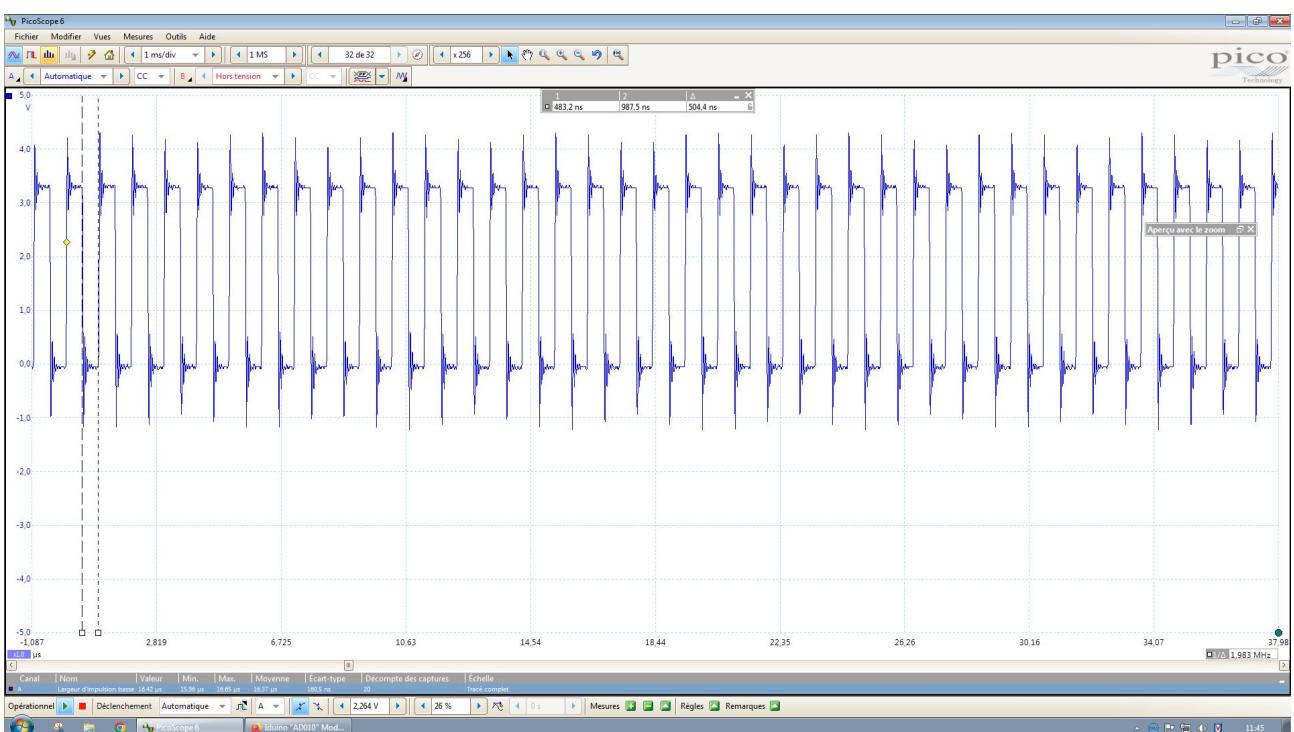
## Programmation

L'objectif est de suivre la doc constructeur afin de respecter le fonctionnement du capteur.

Dans un premier temps on réalise une fonction delay pour mettre en place notre temporisation en millisecondes.

```
void delay (uint16_t delay)//Mise en place d'un timer pour la milliseconde
{
    __HAL_TIM_SET_COUNTER (&htim3, 0);
    while ((__HAL_TIM_GET_COUNTER(&htim3)) < delay);

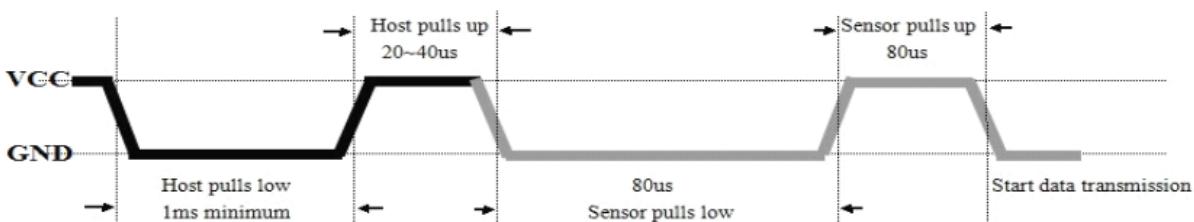
}
```



On le vérifie notamment sur le picoscope.

Ensuite, on suit les différentes étapes décrites dans la doc constructeur.

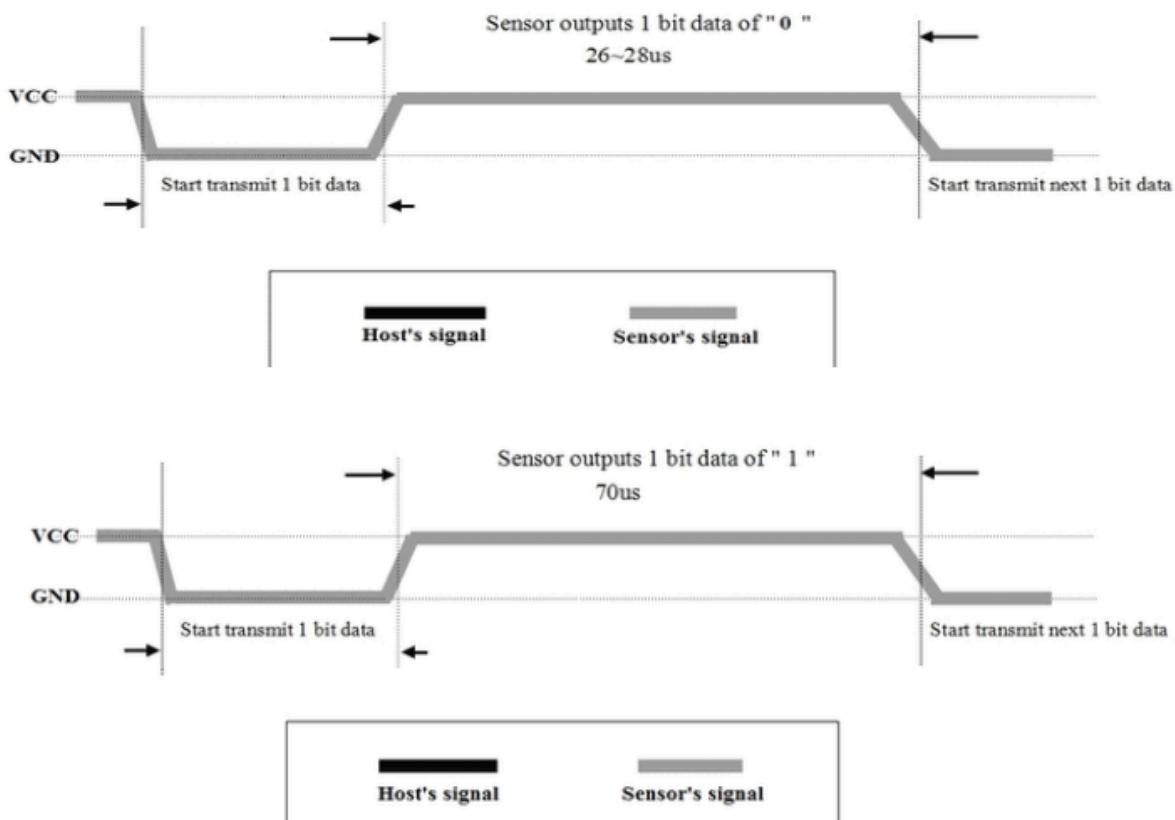
On procède tout d'abord à une initialisation du capteur



On définit la broche en sortie, on met en pin à bas pendant au moins une milliseconde puis on repasse le pin à haut dans un temps compris entre 20 et 40 us et donc on redéfinit la broche comme entrée.

```
Data_Output(GPIOA, GPIO_PIN_8); //on définit la broche comme sortie
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET); //On tire la broche vers le bas
delay(1200); //on attend environ minimum 1 ms
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET); //On tire la broche vers le haut
delay(30); //on attend 30 us
Data_Input(GPIOA, GPIO_PIN_8); //on définit la broche comme entrée pour recevoir les données
```

On peut par la suite procéder à la transmission des données vers le capteur depuis le microcontrôleur.



La transmission se fait bit par bit en commençant par une position basse. La valeur du bit est déterminer par la longueur du signal en position haute : entre 26-28us le bit est "0", et pour 70 us le bit est "1".

```

void Read_data (uint8_t *data)
{
    int i, k;
    for (i=0;i<8;i++)
    {
        if (HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_8) == GPIO_PIN_RESET)//|
        {
            (*data)&= ~(1<<(7-i)); //data bit is 0
            while(!(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8))); //on attend que le pin passe à haut
            delay(40); //on attend 40us
        }
        else //data bit is 1
        {
            (*data)|= (1<<(7-i));
            for (k=0;k<1000;k++)
            {
                if (HAL_GPIO_ReadPin (GPIOA, GPIO_PIN_8) == GPIO_PIN_RESET)
                {
                    break;
                }
            }
            while(!(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8)));
            delay(40);
        }
    }
}

```

Le DHT22 renvoie 40 bits tel que :

DATA = données RH intégrales 8 bits + données RH décimales 8 bits + données T intégrales 8 bits + données T décimales 8 bits + somme de contrôle 8 bits

```

while(!(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8))); //on attend que le pin passe en mode haut
delay(40); //on attend 40 us

Read_data(&dataH1); //on récupère les adresses stockées
Read_data(&dataH2);
Read_data(&dataT1);
Read_data(&dataT2);
Read_data(&SUM);

//Affichage
if (SUM==((dataH1+dataH2+dataT1+dataT2) & 0x00FF))
{
}

```

Si la transmission est correctement faite, **SUM** doit être égale aux 8 derniers bits de DATA.

```

if (SUM==((dataH1+dataH2+dataT1+dataT2) & 0x00FF))
{
Humidite = (((dataH1<<8) | dataH2)*0.1f);
Temperature = (((dataT1<<8) | dataT2)*0.1f);
}

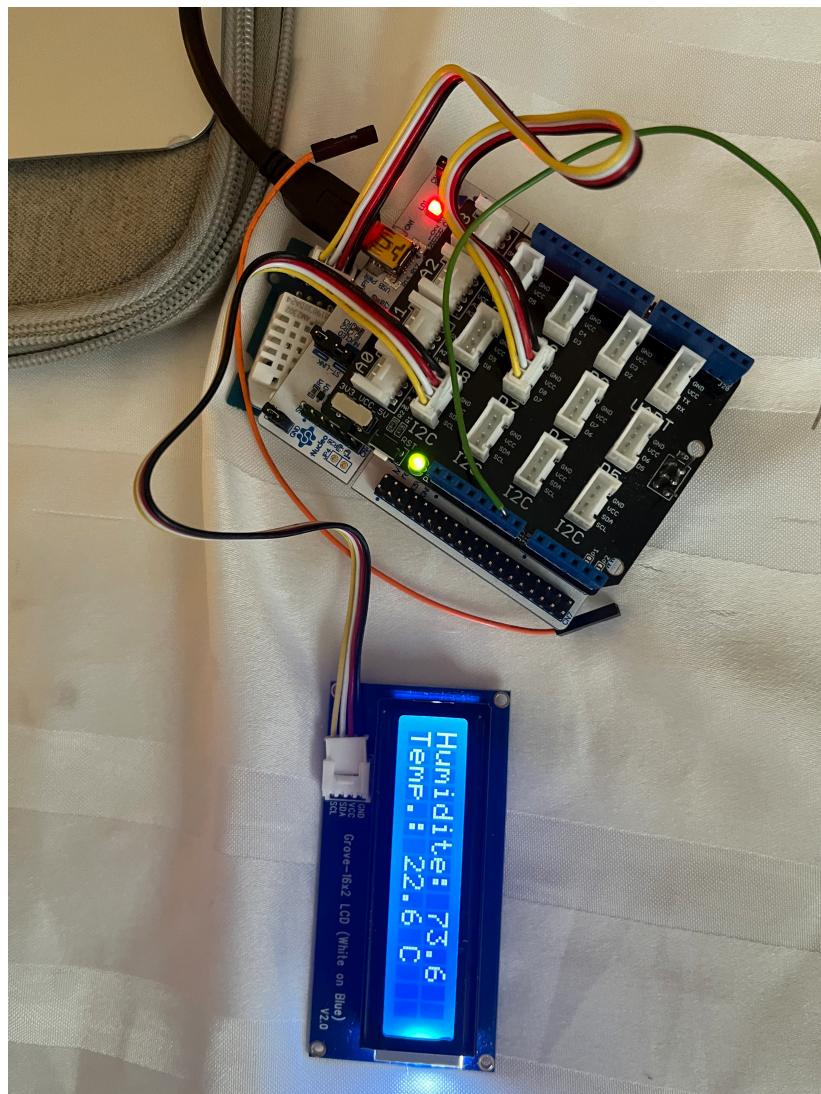
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET); //on repasse la broche à haut pour la prochaine lecture

/*commence transmission vers LCD*/
clearlcd();
sprintf(bufRH,"Humidite: %.1f %", Humidite);
sprintf(bufT, "Temp.: %.1f C", Temperature);
lcd_position(&hi2c1,0,0);
lcd_print(&hi2c1,bufRH);
lcd_position(&hi2c1,0,1);
lcd_print(&hi2c1,bufT);
reglagecouleur(0,0,255);

```

On récupère ensuite les données stockées pour les afficher sur le lcd.

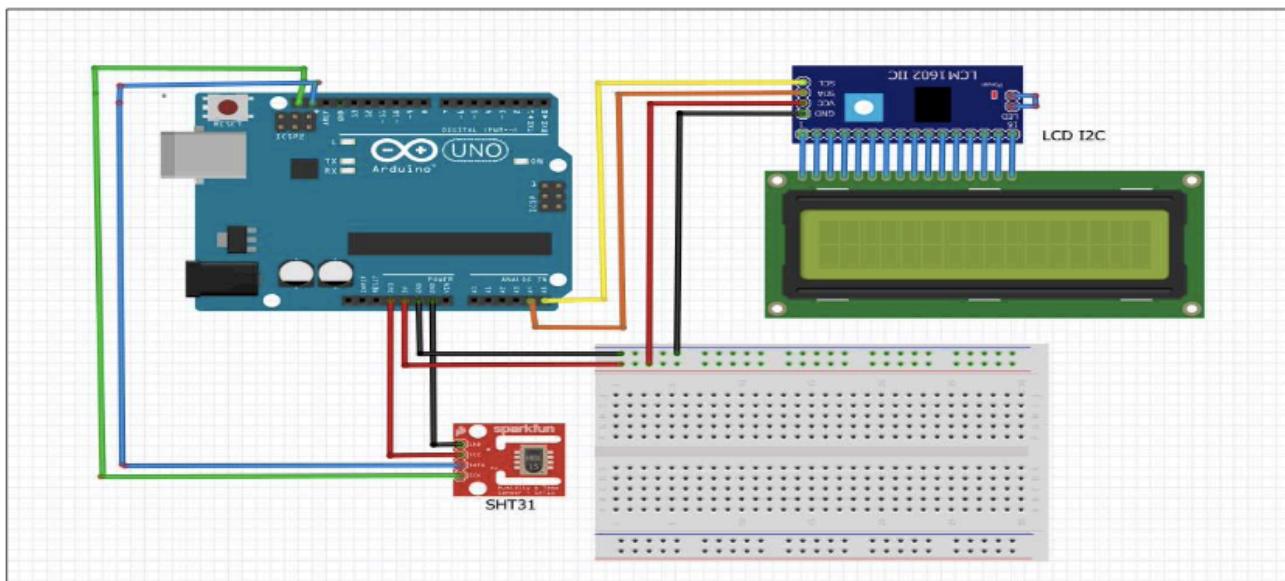
## Résultat



Le résultat escompté ayant été obtenu après le départ en stage, la visualisation de la trame via le picoscope n'a pas pu être réalisée.

## SHT31

Schéma de câblage

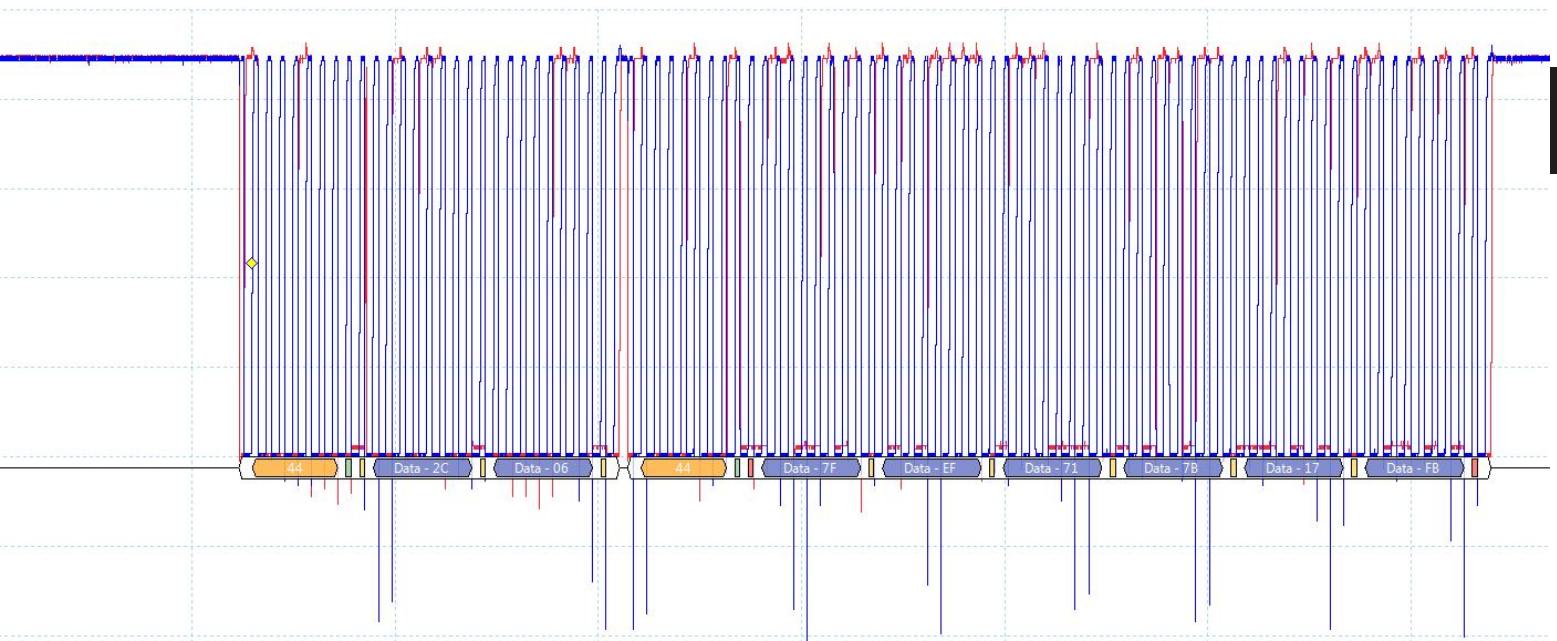


La carte Arduino modélise le module grove base shield.

## Programmation et visualisation

Le microcontrôleur est configuré en tant que maître et le capteur en tant qu'esclave.

Celui-ci va donc transmettre une séquence de bits au démarrage pour initier la communication, et donc envoyer l'adresse du capteur pour y lire et écrire les données.



```

/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
    buf[0] = CAPTEUR_CMD_MSB;//on met les paramètres de capteur pour le mode de fonctionnement
    buf[1] = CAPTEUR_CMD_LSB;//on met la valeur de l'octet de paramètres de capteur
    ret = HAL_I2C_Master_Transmit( &hi2c1, CAPTEUR_ADDRS, buf, 2, HAL_MAX_DELAY);//envoie des octets via I2C
    if ( ret != HAL_OK)
    {
        strcpy((char*)buf, "erreur_T!!\r\n");
    }
    else
    {

        ret = HAL_I2C_Master_Receive( &hi2c1, CAPTEUR_ADDRS, buf, 6, HAL_MAX_DELAY);//réception des octets
        if ( ret != HAL_OK)
        {
            strcpy((char*)buf, "erreur_R!!\r\n");
        }
    }
    value = buf[1] | buf[0] << 8;// la variable value contient la valeur des octets des MSB et LSB de la réponse
    temp = -45 + 175 * ( (float)value / 65535);// on calcule la valeur de la température
    Entier_part = (int) temp;// récupération de la partie décimale de la température
    Decimal_part = temp;// récupération de la partie décimale de la température
    Decimal_part *= 100;
    Decimal_part = Decimal_part - (Entier_part * 100);// calcule de la partie décimale de la température
    value = buf[4] | buf[3] << 8;

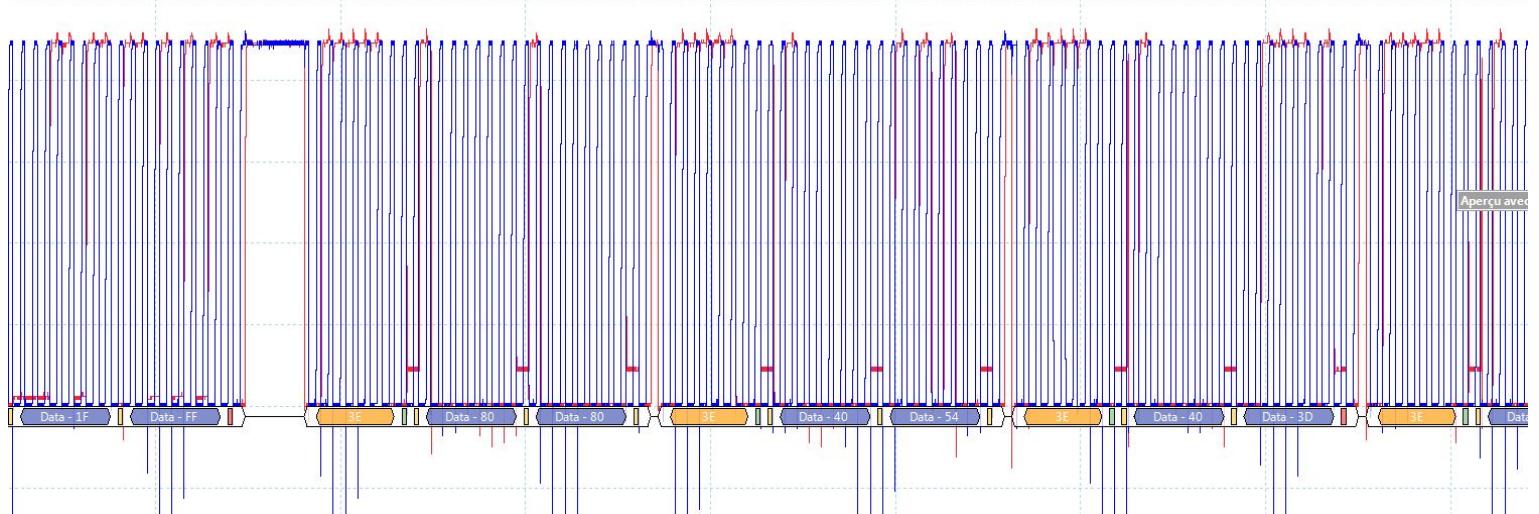
    umid = -49 + 315 *( (float)value / 65535);
    sprintf( (char*)buf, "%u.%u C ; %u D", (unsigned int) Entier_part,(unsigned int) Decimal_part,(unsigned int) Decimal_part);
    lcd_position(&hi2c1,0,0);
    lcd_print(&hi2c1,"T = ");//affichage de la température
    lcd_position(&hi2c1,7,0);
    lcd_print(&hi2c1,buf);//affichage de la valeur de la température
    lcd_position(&hi2c1,0,1);

    lcd_print(&hi2c1,"H = ");//affichage de l'humidité
    lcd_position(&hi2c1,7,1);
    lcd_print(&hi2c1,&buf[10]);//affichage de la valeur de l'humidité
}
}
HAL_UART_Transmit(&huart2, buf, strlen((char*)buf), HAL_MAX_DELAY);//transmission des octets
HAL_Delay(1000);

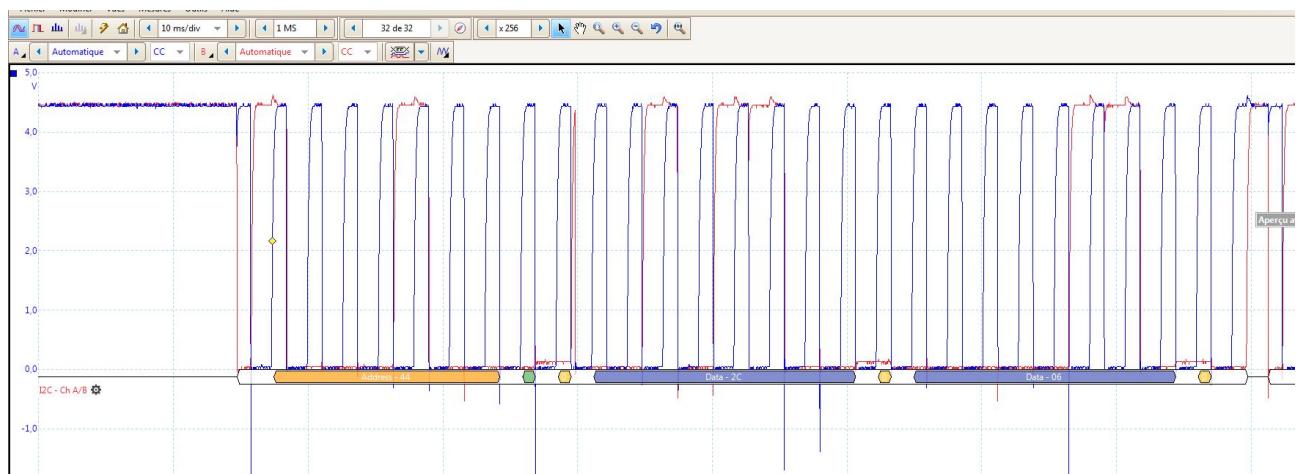
```

Ci-dessus, on visualise la trame renvoyant la commande et la réponse associé

Par la suite, le microcontrôleur va envoyer une commande pour récupérer les données sur l'humidité et la température, à laquelle va répondre le capteur en y transmettant les données.



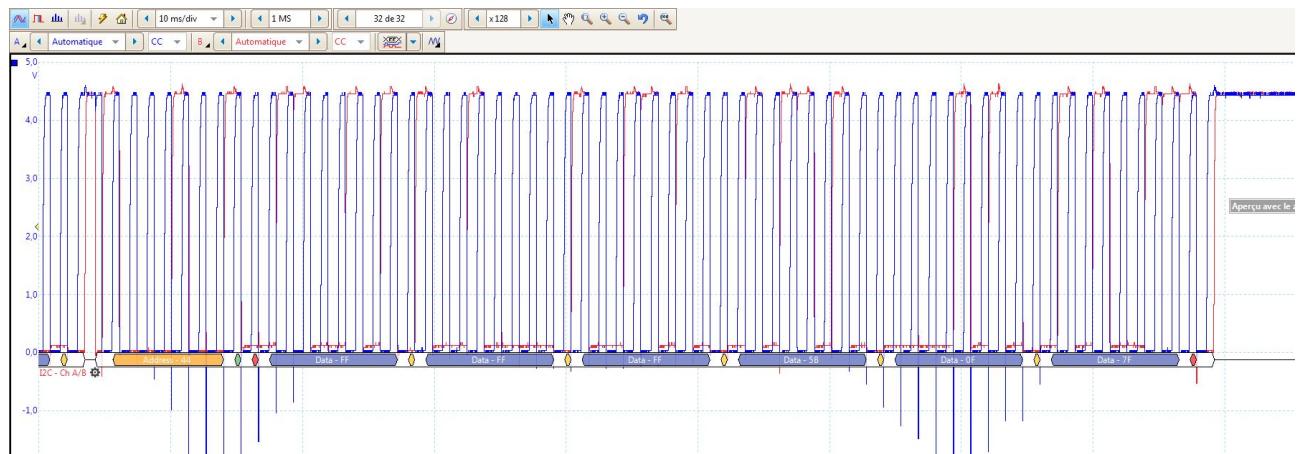
On retrouve donc la trame de transmission complète incluant l'initialisation du lcd.



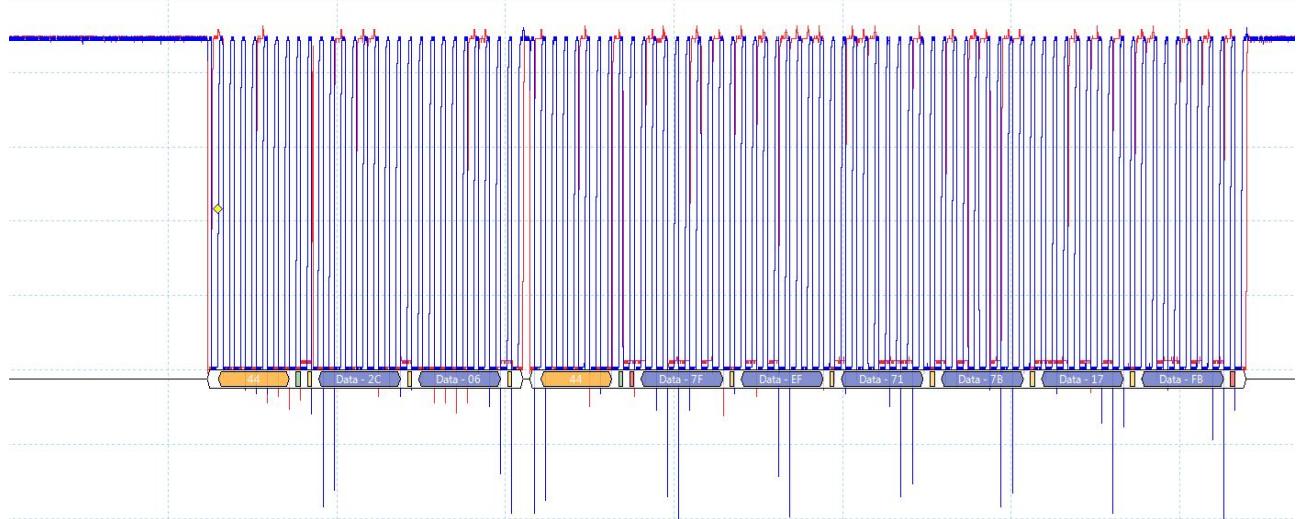
Trames

Commande

Début de réponse du capteur avec les données sur la température et l'humidité



Commande et réponse



## II - Conclusion

À travers ces Tps de base, on a pu avoir une première main sur l'émission-transmission de données par un microcontrôleur vers un capteur en vue de faire fonctionner un système et comprendre la notion d'adressage liés aux systèmes embarqués