Brendan ROBIN **M1 SME**

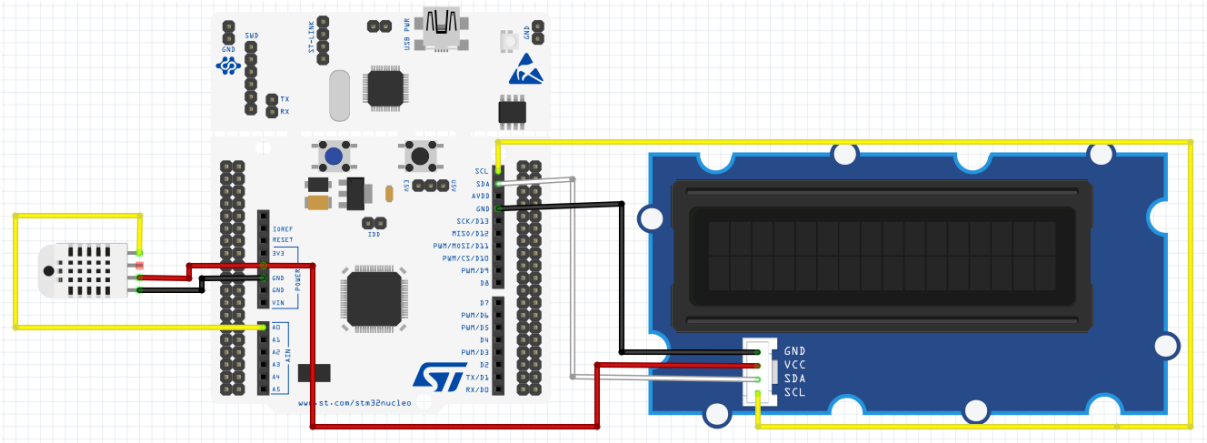
Rapport TP de base DHT22

Matériel : Pour ce TP de base nous avons un microcontrôleur STM32L476rg, un écran LCD (White on Blue) et un capteur de température et d'humidité.

Logiciel : CubeIDE qui est un logiciel pour paramétrer et programmer notre microcontrôleur STM32L476rg

But : Dans ce TP de base nous voulons capter la température et l'humidité d'une pièce grâce au capteur de DHT22 et le microcontrôleur STM32L476rg, puis afficher leur valeur sur notre écran LCD.

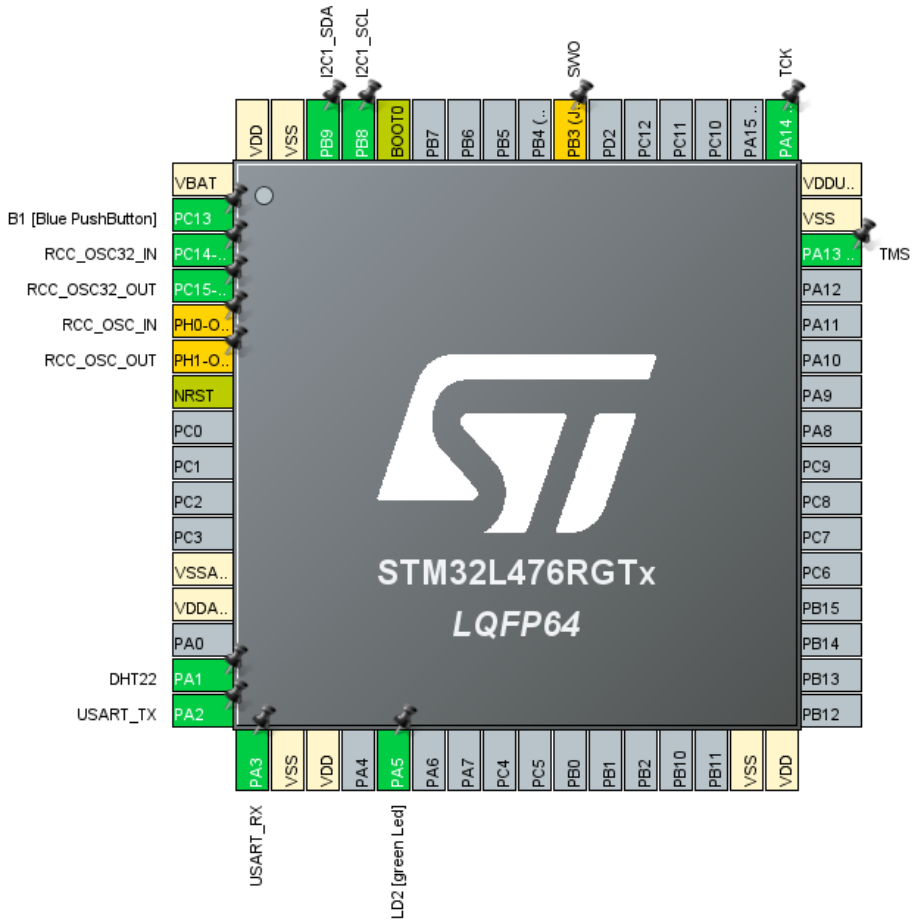
I/Schéma de câblage (fritzing)



La carte STM32 ci-dessus n'est pas ma carte STM32L476rg car mon modèle de carte n'est pas disponible pour fritzing, mais mes 2 autres composants (DHT22, Ecran LCD I2C) sont bien connectés aux bonnes broches de la carte STM32 sur le schéma qui correspond à mes broches réel sur ma carte.

II/Initialisation du microcontrôleur STM32L476rg

Avant toute chose, il faut d'abord paramétrer notre microcontrôleur selon notre utilisation.



Tout d'abord on a notre écran LCD qui fonctionne avec un protocole de communication I2C. Il faut donc paramétrer un port SDA et SCL sur le microcontrôleur pour permettre la communication avec l'écran LCD. On peut voir les ports sur le schéma au-dessus (PB8 & PB9).

Ensuite, comme notre capteur de température et d'humidité DHT22 fonctionne avec un protocole de communication One Wire, on lui attribue une broche sur le microcontrôleur qui est la broche PA1 comme on le voit sur le schéma ci-dessus. Cette broche doit être mise à 1 au repos d'après la datasheet, donc on va la paramétrer en pull-up.

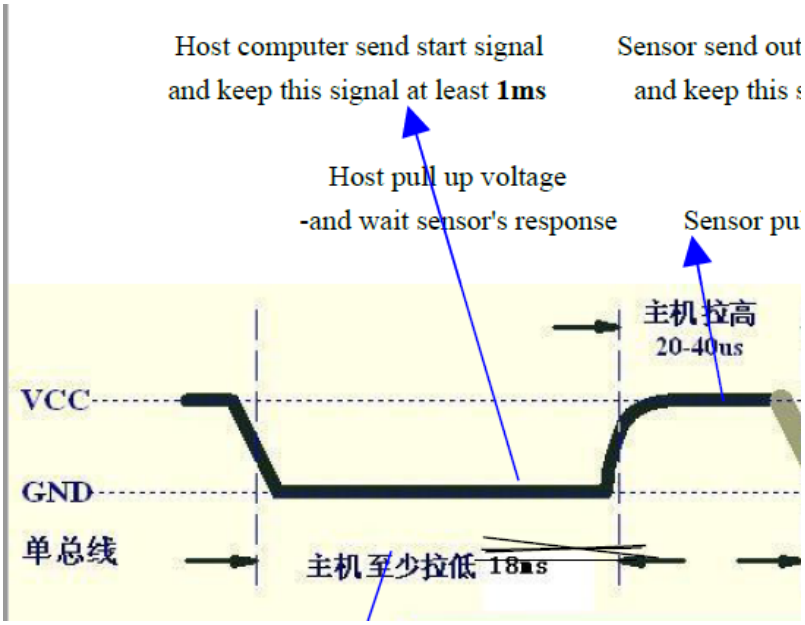
Enfin on n'oublie pas de paramétrer un timer en microseconde car c'est nécessaire pour le bon fonctionnement du capteur DHT22.

III/Initialisation du capteur DHT22

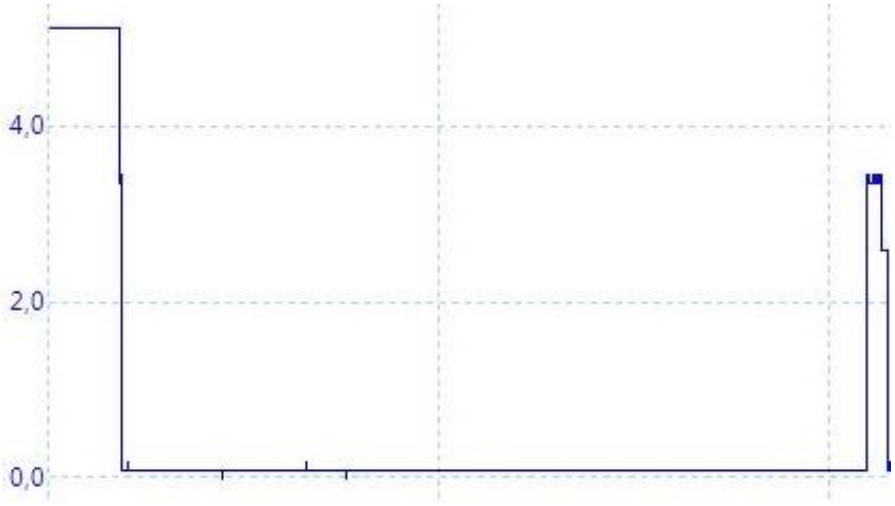
Tout d'abord on créer un fonction pour notre timer en microsecondes.

```
void Delay_us(uint16_t us)
{
    __HAL_TIM_SET_COUNTER(&htim2, 0);
    while(__HAL_TIM_GET_COUNTER(&htim2) < us);
}
```

Ensuite on regarde la datasheet du DHT22 pour voir comment elle s'initialise



Ça nous dit qu'il faut que la broche du signal reliant le microcontrôleur et le DHT22 soit mise à 0 pendant au minimum 1 ms et au maximum 18 ms, puis doit être mise à 1 pendant 20 à 40 µs.



Ici si on compare notre graphe obtenu avec le picoscope avec celle de datasheet, on voit qu'on est bien à 1 au repos, puis on est à 0 pendant exactement 1.2 ms, et on remonte à 1 pendant 30 µs. Par contre on voit que à droite sur la photo, au moment où on veut repasser à 0, la tension descend un peu avant de vraiment descendre à 0. Ce petit changement de hauteur est dû au PIN qu'on fait changer de sens, on le change en entrée.

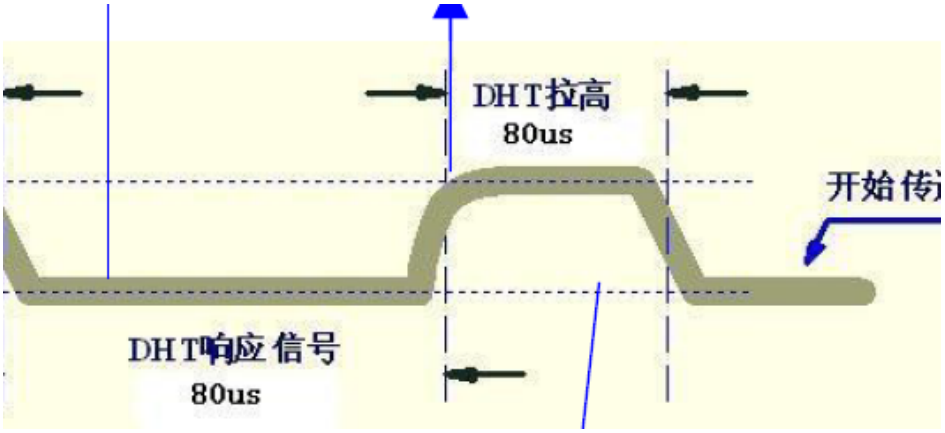
Voici les fonctions qu'on a utilisé pour initialiser le DHT22 :

```c
void Set_Pin_Output (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}

void Set_Pin_Input (GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    GPIO_InitStruct.Pin = GPIO_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOx, &GPIO_InitStruct);
}

void DHT22_start(void)
{
    Set_Pin_Output(DHT22_GPIO_Port, DHT22_Pin);//Mettre broche en sortie pour initialiser DHT22
    HAL_GPIO_WritePin(DHT22_GPIO_Port, DHT22_Pin, GPIO_PIN_RESET);
    Delay_us(1200);
    HAL_GPIO_WritePin(DHT22_GPIO_Port, DHT22_Pin, GPIO_PIN_SET);
    Delay_us(30);
    Set_Pin_Input(DHT22_GPIO_Port, DHT22_Pin); //Mettre broche en entrée pour recevoir réponse DHT22
}
```

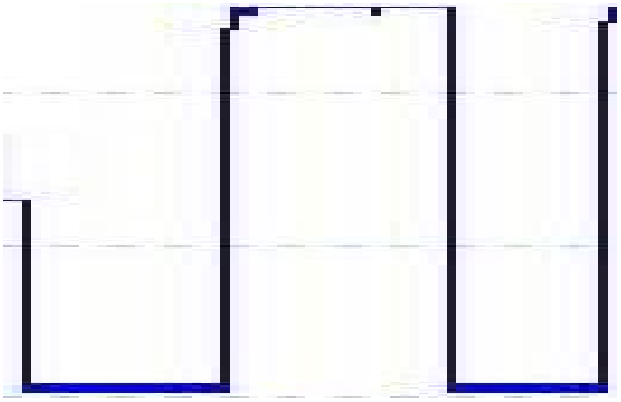
Comme on le voit, nous avons une fonction pour mettre une broche du microcontrôleur en sortie, puis une autre fonction mettre une broche en entrée. Puis on a une 3ième fonction qui va initialiser le DHT22 selon les instructions de sa datasheet expliqués précedemment.

IV/Réponse du DHT22

Notre DHT22 va répondre en 2 phases distinctes :



Tout d'abord, si son initialisation a bien fonctionné alors elle va répondre au microcontrôleur par un signal qui sera à 0 pendant 80 µs puis à 1 pendant 80 µs et enfin à 0 pendant 50 µs avant d'avoir une nouvelle mise à 1 qui sera notre premier bit de donnée fourni par le DHT22.

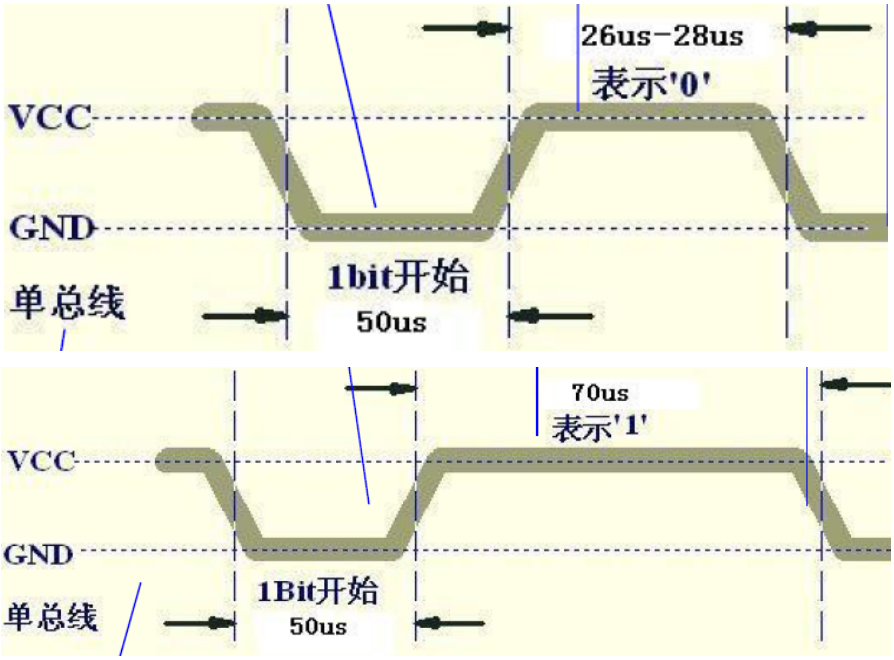


Ici comme sur la datasheet le capteur répond par un 0 puis un 1 et on revient à 0 avant de recevoir le premier bit. Par contre on n'a pas exactement les mêmes valeurs. On est d'abord à 0 pendant 73 µs puis à 1 pendant 78 µs d'après nos mesures contre 80 µs d'après la datasheet. Puis on est bien à 0 pendant 50 µs avant de recevoir notre premier bit. On est donc assez proche des valeurs données par la datasheet.

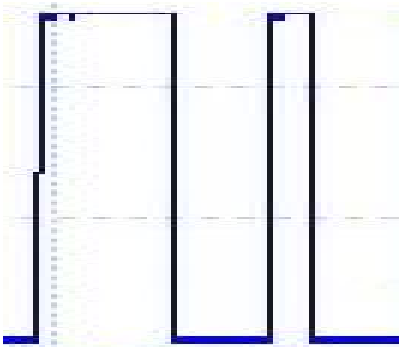
Pour programmer cette réception de la réponse du DHT22 on a mis en place cette fonction :

```
int DHT22_Answer(void)
{
    //Je choisit toujours 10 us de plus pour les temps pour une marge d'erreur
    int temps = 0, reponse = 0;
    //verifier qu'on est bien a 0 pendant 80 us
    while((HAL_GPIO_ReadPin(DHT22_GPIO_Port, DHT22_Pin) == GPIO_PIN_RESET) && (temps < 90))
    {
        Delay_us(1);
        temps++;
    }

    if(temps < 90)//si on est bien rester à 0 pendant de 80 us alors OK
    {
        temps = 0;
        //verifier qu'on est bien a 1 pendant 80 us
        while((HAL_GPIO_ReadPin(DHT22_GPIO_Port, DHT22_Pin) == GPIO_PIN_SET) && (temps < 90))
        {
            Delay_us(1);
            temps++;
        }

        if(temps < 90)//si on est bient rester à 1 pendant 80 us alors OK
        {
            //début de la trame de donnée 50 us à 0 puis 1er bit de donnée
            temps = 0;

            //vérifier la reception du premier bit
            while((HAL_GPIO_ReadPin(DHT22_GPIO_Port, DHT22_Pin) == GPIO_PIN_RESET) && (temps < 60))
            {
                Delay_us(1);
                temps++;
            }
            if(temps < 60)
            {
                reponse = 1;
            }
        }
    }

    Delay_us(40);

    return reponse;
}
```

Ensuite on va recevoir un signal qui sera une variation de mise à 1 et de mise à 0. Ce signal contiendra nos données d'humidité et de température. Le décryptage du signal sera de la manière suivante :



Si le signal est à un pendant 26 à 28 µs alors notre bit est un 0, sinon c'est que notre bit est un 1.

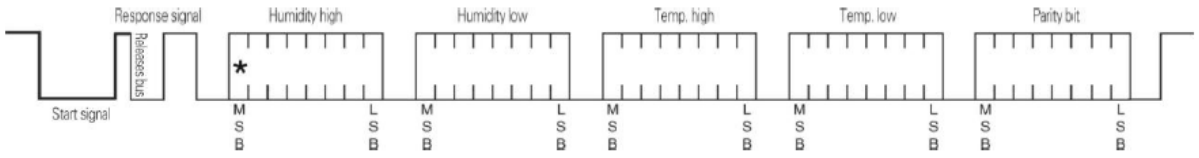


Voici une partie des données qu'on a reçues sur le picoscope, donc comme sur la datasheet on voit bien que la durée où on est à l'état 1 varie, quand la durée est courte on a un bit à 0 sinon on a un bit à 1.

Pour décrypter ce message venant du DHT22 on va donc utiliser cette fonction :

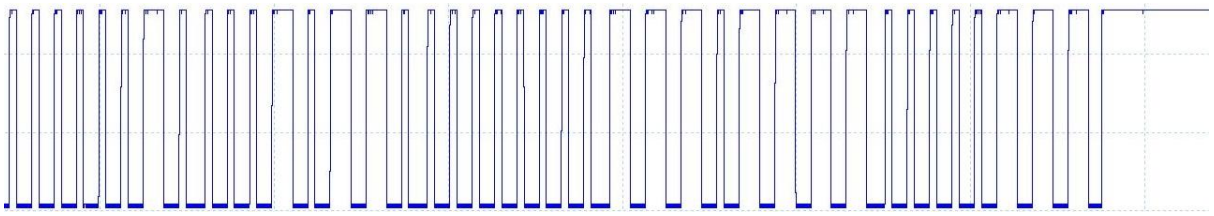
```
void DHT22_Read_Data (uint8_t *data)
{
    int i, k;
    for (i=0;i<8;i++)
    {
        if (HAL_GPIO_ReadPin (DHT22_GPIO_Port, DHT22_Pin) == GPIO_PIN_RESET)
        {
            (*data)&= ~(1<<(7-i)); //Mettre bit à 0
            while(!(HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1)));
            Delay_us(40);
        }
        else
        {
            (*data)|= (1<<(7-i)); //Mettre bit à 1
            for (k=0;k<1000;k++)
            {
                if (HAL_GPIO_ReadPin (DHT22_GPIO_Port, DHT22_Pin) == GPIO_PIN_RESET)
                  {
                    break;
                  }
            }
            while(!(HAL_GPIO_ReadPin(DHT22_GPIO_Port, DHT22_Pin)));
            Delay_us(40);
        }
     }
}
```

Enfin d'après la datasheet les données seront sous la forme suivante :



Pic5: AM2302 Single-bus communication protocol

On aura donc 16 bits pour la valeur de l'humidité, puis 16 bits pour la valeur de la température et 8 bits de parité à la fin qui servira à vérifier qu'on n'ait pas d'erreur. Normalement la valeur de l'octet de parité est égale à la somme des 4 octets précédents.



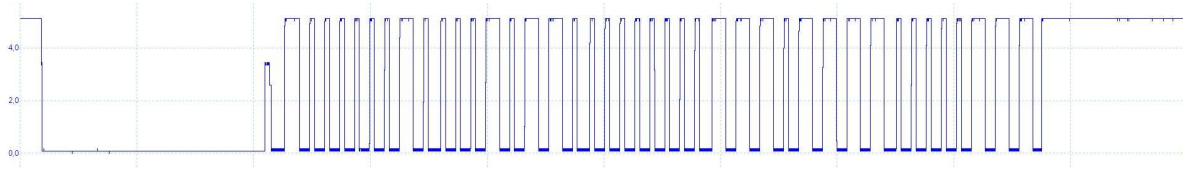
Ici sur notre résultat vu au picoscope, comme sur la datasheet on voit bien qu'on reçoit 40 bit qui seront interprétés comme expliqué précédemment .

On a ensuite utilisé ce code de la manière suivante pour lire les données fourni par le DHT22 :

```
if(reponse == 1)
{
    //Commencer aquérir les 5 différents octets
    DHT22_Read_Data(&HH);
    DHT22_Read_Data(&HL);
    DHT22_Read_Data(&TH);
    DHT22_Read_Data(&TL);
    DHT22_Read_Data(&SUM);

    check = HH + HL + TH + TL;
    if(check == SUM)//vérifier qu'il n'y a pas d'erreur dans les données recu
    {
        //combiner 2 octets d'humité et diviser resultat par 10 pour avoir humidité en %
        Humidite = (float) ((HH<<8) | HL) / 10;
        //combiner 2 octets de température et diviser resultat par 10 pour avoir température en °C
        Temperature = (float) ((TH<<8) | TL) / 10;
```

Donc comme expliquer, on va avoir 5 variables de 8 bits chacune qui vont prendre les valeurs des 5 octets envoyé par le DHT22, puis on vérifie qu'il n'y ait pas d'erreur, ensuite on fusionne les 2 octets de température ensemble et les 2 octets d'humidité ensemble et on divise les 2 valeurs obtenu par 10 car d'après la datasheet il faut cette division par 10 pour avoir des valeurs juste.

Voici une photo de notre trame entière :



V/Affichage des données obtenu sur l'écran LCD

```
void DHT22_Display_Data(void)
{
    sprintf(tabH,"Humidite: %.1f ", Humidite);
    sprintf(tabT, "Temp.: %.1f C   ", Temperature);
    lcd_position(&hi2c1, 0, 0);
    lcd_print(&hi2c1, tabH);
    lcd_print(&hi2c1, "%");
    lcd_position(&hi2c1, 0, 1);
    lcd_print(&hi2c1, tabT);
}
```

Pour afficher la température et l'humidité sur l'écran LCD on a créé cette fonction qui utilise des fonctions d'une bibliothèque qu'on a tous reçue et qui permet d'utiliser l'écran LCD.