French student report

Rapport de projet C++
Station météo

Auteurs :
Thomas ABGRALL
Eric REBILLON

MASTER Informatique embarqué

YNOV Mastère Aéronautique et système embarqué

April 5, 2020

Contents

Contents **1**

I Partie de groupe **2**

1 Introduction **3**

II Thomas ABGRALL **4**

2 Introduction **5**
2.1 Prise en main 5

3 Reception **6**
3.1 Lecture données reçu 6

4 Multi taches **7**

5 Affichage **8**

III Eric Rebillon **9**

6 Capteur d'Altitude **10**
6.1 Mise en place du Capteur 10
6.1.1 Commande 10
6.1.2 Trames d'envoyé et réception 11
6.2 Test du Capteur 12

7 Envoie de la valeur **13**
7.1 Mise en place de la communication 13
7.2 Test de la communication 15

8 Conclusion **16**

Part I

Partie de groupe

Chapter 1

Introduction

Ce projet a été réalisé dans le cadre du cour de programmetion CPP.
Il a pour but d'approfondir et mettre en oeuvre nos compétences en programmation orientée objet.
Pour celà nous avons réalisé une mini station météo qui à pour visé de mesurer et afficher les valeurs de pression et température avec une appplication.
Le projet est donc composé en deux parties:

- L'une est chargé de la reception des données avec le port série et l'affichage des données,
- La seconde mesurer les valeurs avec la carte STM32 est les transmettres en communication série.

Le projet est open-source et présent sur Github: **`https://github.com/ThomasAbg/StationMeteo`**

Part II

Thomas ABGRALL

Chapter 2

Introduction

Nous allons éclaircir dans ce chapitre la partie application du projet.
Cette partie doit permettre de lire les valeurs envoyées par la STM32 via le port série, puis les afficher via une interface.
L'affichage doit être continu ainsi que la reception.

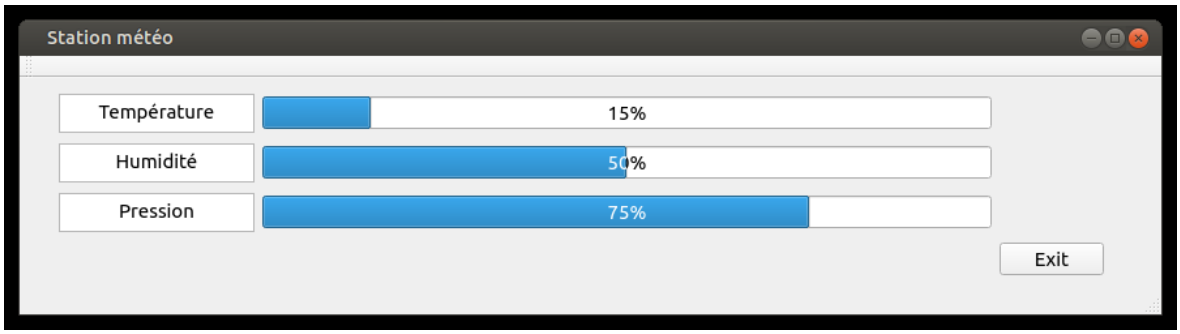


FIGURE 2.1: Aperçu de l'application

2.1 Prise en main

1. Recupérer le projet

```
git clone https://github.com/ThomasAbg/StationMeteo
```

2. Executer le code:

```
./StationMeteo/InterfaceStation/Application_meteo
```

   Il est aussi possible d'utiliser QtCreator.

3. Optionnel, si vous souhaitez compiler le code:

```
votrechemin/StationMeteo/InterfaceStation/
Application_meteo.pro -spec linux-g++
CONFIG+=debug CONFIG+=qml_debug &&
/usr/bin/make qmake_all main.o
```

   Il est aussi possible d'utiliser QtCreator.

Chapter 3

Reception

3.1 Lecture données reçu

Pour accédere aux données écrite dans le port série (dit "COM") il faut réquisitionner, le configurer.

```
m_serial=new QSerialPort(this);
m_serial->setPortName("/dev/ttyUSB0");
m_serial->setBaudRate( QSerialPort::Baud115200);
m_serial->setDataBits( QSerialPort::Data8);
m_serial->setStopBits(QSerialPort::OneStop);sc
m_serial->setParity(QSerialPort::NoParity);
m_serial->setFlowControl(QSerialPort::NoFlowControl);
```

FIGURE 3.1: Réquisition et configuration du port série

Maintenant viens le plus important, la lecture du port, pour celà je lie la fonction readyRead de la bibliothèque QSerialPort a ma fonction de lecture. La fonction readyRead renvoi toutes nouvelles données reçu dans le port qui à été configuré (objet: m_serial).

```
connect(m_serial,&QSerialPort::readyRead,this,&SerialCapture::read);
```

FIGURE 3.2: Connecte la fonction readyRead fonction

```
//envoie des donnée dans le QList
void SerialCapture::read()
{
    QByteArray buf=m_serial->readAll();
    m_data.append(buf.toDouble());

}
```

FIGURE 3.3: Fonction lecture données

Chapter 4

Multi taches

Afin de pourvoir au besion d'afficher les données en même temps que l'on les recoivent, on est contraint à faire de faire deux action à la fois.
Pour fairer celà il y a plusieurs solutions: multi-processus, multi-thread.
J'ai choisie le multi-threading car les threads partagent les mêmes ressources (data) car ils se trouvent de le même processus, et ils sont plus légés que de réquisitionner un autre processus.

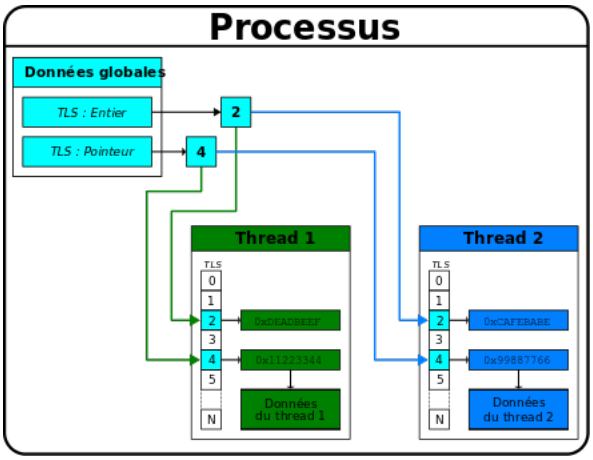


FIGURE 4.1: Illustration threads

Dans notre cas le Thread1 est l'affichage et le Thread2 la lecture du port série.

Chapter 5

Affichage

Pour faire l'interface de l'application j'ai utilisé le designer de Qt Creator, ce dernier donne acces à une large gamme de composants d'interface préconfigurés. Voici le résultat sur l'interface du designer.

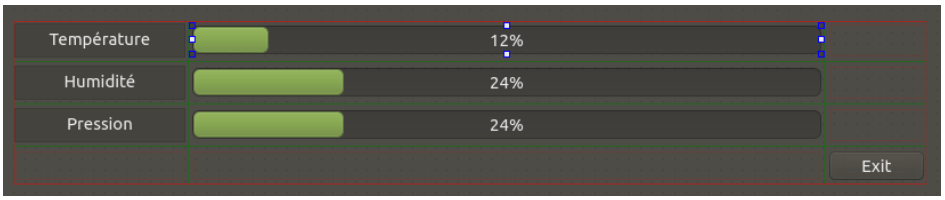


FIGURE 5.1: Aperçu de l'application sur designer Qt Creator

Part III

Eric Rebillon

Chapter 6

Capteur d'Altitude

Pour le projet, ayant déjà fait le capteur définie dans les consignes du projet, je met en place le capteur MS5607-02BA03.

6.1 Mise en place du Capteur

Le Capteur est disponible dans deux protocoles :

1. SPI
2. I2C

Par défaut le capteur est en I2C, on peut passer le Capteur en SPI en dessoudant quelques résistances.
D'après les données constructrices l'adresse I2C permettant la communication avec le module est 0xEC en lecture ou 0XED en écriture.

Adresse I2C	Réponse du Capteur
0xEE	Renvoie des valeurs (trouvées pendant les test)
0x76 in Read 0x77 in Write	Renvoie Aucune Valeur (Définie dans la datasheet)

6.1.1 Commande

Toutes les commandes suivantes sont disponibles dans la datasheet MS5607-02BA03

Nom de la commande	Valeur Hexadécimale
Reset	0x1E
Convert D1 256	0x40
Convert D1 512	0x42
Convert D1 1024	0x44
Convert D1 2048	0x46
Convert D1 4096	0x48
Convert D2 256	0x50
Convert D2 512	0x52
Convert D2 1024	0x54
Convert D2 2048	0x56
Convert D2 4096	0x58
ADC Read	0x00
Prom Read	0xA0 to 0xAE

Tableau 1

Le capteur fonctionne avec ces commandes.

1. D1 : représente la donnée pression
2. D2 : représente la donnée température

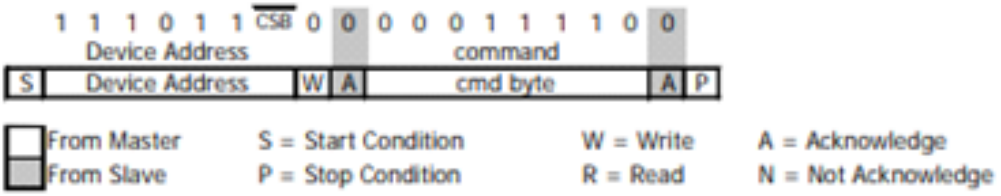
Tous deux peuvent avoir une résolution de :256

1. 512
2. 1024
3. 2048
4. 4096

La résolution change le temps de calcul du capteur, selon l'utilisation nous pouvons avoir plus ou moins besoin de résolution grande. Dans notre cas pas utile, le capteur sera juste une indication pour l'utilisateur.
Remise à zéro : 0x1E
ADCRead permet de dire au capteur que nous voulons la donnée pression-température.
PromRead permet de récupérer des coefficients pour affiner et calibrer les données reçues du capteur. Il faut juste appliquer les calculs indiqués sur les données reçues.

6.1.2 Trames d'envoyé et réception

Dans chaque trame I2C, il faudra envoyer l'adresse I2C du module suivit de la commande. Sachant que la commande est précédée et suivit d'une Acknowledge.
L'acknowledge est générer par le module, donc si pendant le debug il n'a pas d'acknowledge cela signifie que :
la trame n'est pas envoyée correctement
L'adresse I2C n'est pas correcte
Capteur est défectueux



Pour visualiser la trame :
Figure 10 : Image Datasheet Trame
Slave : Capteur
Master : Carte STM32
Chaque commande envoyée au capteur devra être réalisée de cette façon.

Algorithme global

Pour récupérer une valeur selon les informations auparavant, il faut faire de cette façon : au démarrage il faut remettre à zéro, le capteur puis demander les 9 coefficients.
On envoie les résolutions voulues au capteur. On stocke les 2 valeurs reçues puis conversion. La conversion est en cours de réalisation.

6.2 Test du Capteur

Lors de la mise en place du capteur, le module réponds à une adresse I2C non définie dans la datasheet , les adresses définies dans la datasheet ne fonctionne pas.
Le module envoie des valeurs incohérentes puis après des semaines de test le module ne réponds plus. Le module est très difficile à mettre en place.

Chapter 7

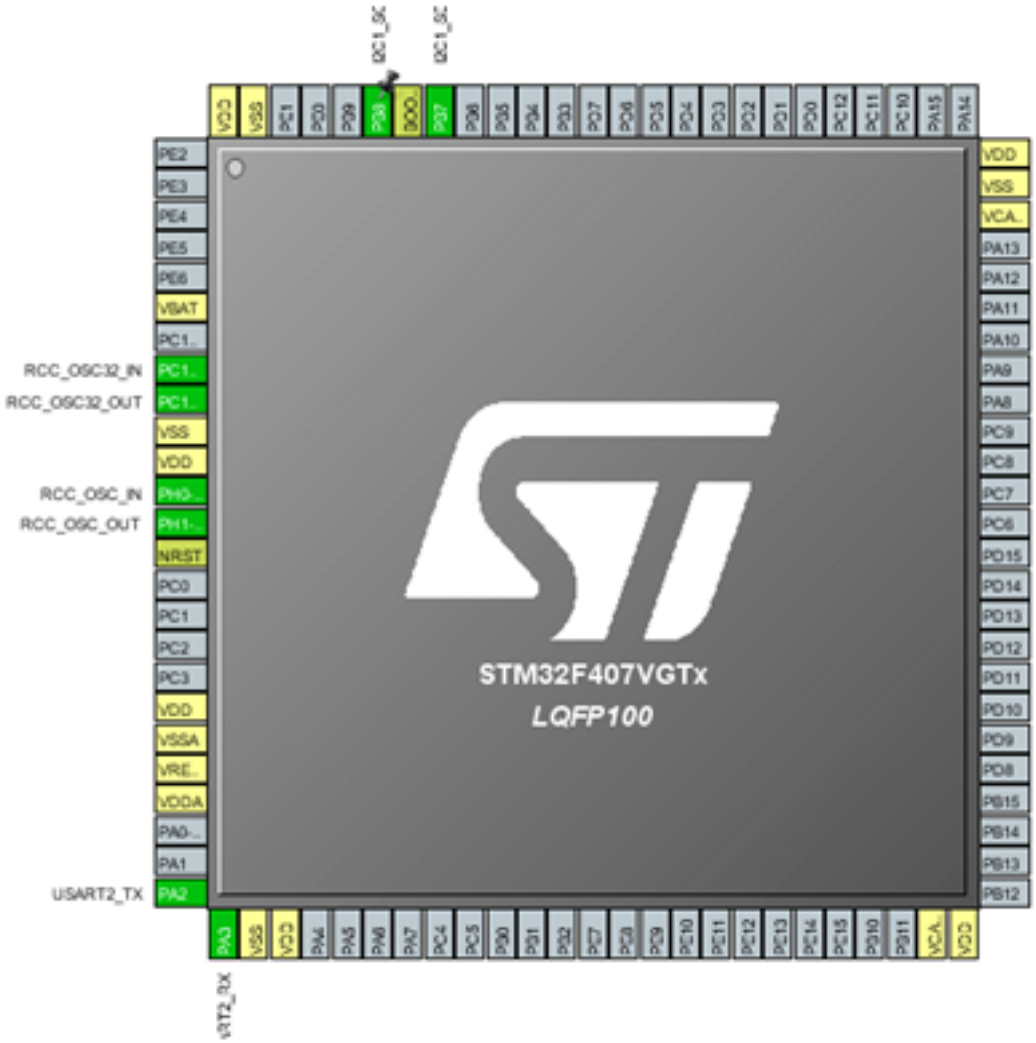
Envoie de la valeur

7.1 Mise en place de la communication

Après avoir fait cela on met en place le protocole UART, pour l'envoie, les paramètres :

1. Asynchrone
2. 115200 baud , pour la vitesse de transmission
3. 8bits de données
4. Pas de bits de parité
5. 1 bits de stop
6. Reception et transmission

Seulement activation de l'UART2, UART2 est celui qui est reliée à l'USB de la Shield de la STM32.
Une fois avoir mis en place l'UART avec le CUBMX



Une fois les paramètres mise en place on génère le code et toute les initialisations utile pour notre cas.
Maintenant nous pouvons utiliser les HALs
Pour la fonction a utilisé :

1. HAL_UART_Transmit : pour transmettre une valeur en UART

Dans la fonction, 4 paramètres :

1. L'uart sur lequels envoyer les données
2. Le tableau contenant les données a envoyées
3. Le nombre d'octets ou de caractère a envoyés
4. Le Timeout

Pour envoyer les valeurs du capteur il faut d'abord convertir le type reçues par le capteur :

1. Non signés, 32 bits

Pour convertir le format de données il faut réaliser un cast.
void Send_Data(uint32_t data)

```
{
    char tab[4];
    int test = (int)data;
    itoa(test, tab,10);
    HAL_UART_Transmit(&huart2, tab, 4, 2000);
    HAL_UART_Transmit(&huart2, "\r\n", 3, 2000);
}
```

Une fois la valeur convertit au bon format de données, il faut convertit le type Int to Char, autrement dit types entiers vers un type caractères.
Pour cela on utilise la fonction de la librairie <stdio>, pour utiliser la fonction itoa, avec un le première paramètre c'est la valeur entière à convertir en caractères, le deuxième paramètre est le tableau de caractères et le dernier paramètres et le format de donnée que l'ont veut avoir autrement dit décimale, binaire, hexadécimale, etc ...
On envoie avec la fonction HAL présenter ci-dessus en mettant en paramètre le tableau contenant les caractères convertit du type entier.

7.2 Test de la communication

Pour tester l'envoie, comme le capteur ne fonctionne pas correctement est très difficile à mettre en place je ferait varier une valeur du même type que le capteur. De 0 à 100 (comme la temperature). Et on envoie les données.

```
void TestSend_Data()
{
    uint32_t RandomValue = 0;
    for (RandomValue = 0; RandomValue < 99; ++RandomValue) {
    Send_Data(RandomValue);
    }
}
```

On teste ce qui est reçue par le périphérique.
Pour le teste, on utilise putty avec les paramètres suivants :

1. Le COM de la shield de la STM32
2. Vitesse : 115200 Baud

Chapter 8

Conclusion

L'application est complétement fonctionnelle, et la partie mesure de donnée sur la STM32 n'a pas pu être mise en oeuvre car il n'y avais des problème d'exploitation du capteur.

Ce projet nous a permit pratiquer la programmation orienté objet, le multi-tâche, l'utilisation de port série.